Аутентификация и авторизация

ASP.NET Core имеет встроенную поддержку аутентификации на основе куки. Для этого в ASP.NET определен специальный компонент middleware, который сериализует данные пользователя в зашифрованные аутентификационные куки и передает их на сторону клиента. При получении запроса от клиента, в котором содержатся аутентификационные куки, происходит их валидация, десериализация и инициализация свойства User объекта HttpContext.

Для подключения функционала авторизации и аутентификации необходимо изменить содержимое класса Startup:

```
public void ConfigureServices(IServiceCollection services)
{
     string connection =
     Configuration.GetConnectionString("DefaultConnection");
     services.AddDbContext<UserContext>(options =>
     options.UseSqlServer(connection));
     // установка конфигурации подключения

     services.AddAuthentication(CookieAuthenticationDefaults.Authentic
     ationScheme)
         .AddCookie(options => //CookieAuthenticationOptions
     {
           options.LoginPath = new
           Microsoft.AspNetCore.Http.PathString("/Account/Login");
     });
     services.AddControllersWithViews();
}
public void Configure(IApplicationBuilder app)
{
   app.UseDeveloperExceptionPage();
   app.UseStaticFiles();
   app.UseRouting();
   app.UseAuthentication();    // аутентификация
   app.UseAuthorization();     // авторизация
   app.UseEndpoints(endpoints =>
   {
      endpoints.MapControllerRoute(
      name: "default",
      pattern: "{controller=Home}/{action=Index}/{id?}");
   });
}
```

UserContext – класс контекста данных (DbContext), который необходимо создать и поместить в Models.

Для установки аутентификации с помощью куки в вызов services.AddAuthentication передается значение CookieAuthenticationDefaults.AuthenticationScheme. Далее с помощью метода AddCookie() настраивается аутентификация. По сути, в этом методе производится конфигурация объекта CookieAuthenticationOptions, который описывает параметры аутентификации с помощью кук. В частности, в данном случае использовано одно свойство CookieAuthenticationOptions - LoginPath. Это свойство устанавливает относительный путь, по которому будет перенаправляться анонимный пользователь при доступе к ресурсам, для которых нужна аутентификация.

Метод app.UseAuthentication() встраивает в конвейер компонент AuthenticationMiddleware, который управляет аутентификацией. Его вызов позволяет установить значение для свойства HttpContext.User. И метод app.UseAuthorization() встраивает в конвейер компонент AuthorizationMiddleware, который управляет авторизацией пользователей и разграничивает доступ к ресурсам.

Далее нам потребуется контроллер AccountController. В нем нам важны два метода регистрации и авторизации пользователя:

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginModel model)
{
   if (ModelState.IsValid)
   {
      User user = await db.Users.FirstOrDefaultAsync(u => u.Email ==
      model.Email && u.Password == model.Password);
      if (user != null)
      {
         await Authenticate(model.Email); // аутентификация
         return RedirectToAction("Index", "Home");
      }
      ModelState.AddModelError("", "Некорректные логин и(или) пароль");
   }
   return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Register(RegisterModel model)
{
   if (ModelState.IsValid)
   {
      User user = await db.Users.FirstOrDefaultAsync(u => u.Email ==
model.Email);
      if (user == null)
      {
        // добавляем пользователя в бд
        db.Users.Add(new User { Email = model.Email, Password =
model.Password });
        await db.SaveChangesAsync();
        await Authenticate(model.Email); // аутентификация

        return RedirectToAction("Index", "Home");
      } else {
```

```
            ModelState.AddModelError("", "Некорректные логин и(или)
пароль");
        }
        return View(model);
}

private async Task Authenticate(string userName)
{
    // создаем один claim
    var claims = new List<Claim>
    {
        new Claim(ClaimsIdentity.DefaultNameClaimType, userName)
    };
    // создаем объект ClaimsIdentity
    ClaimsIdentity id = new ClaimsIdentity(
        claims, "ApplicationCookie",
        ClaimsIdentity.DefaultNameClaimType,
        ClaimsIdentity.DefaultRoleClaimType
    );
    // установка аутентификационных куки
    await HttpContext.SignInAsync(
        CookieAuthenticationDefaults.AuthenticationScheme,
        new ClaimsPrincipal(id)
    );
}
```

Для установки кук применяется асинхронный метод контекста HttpContext.SignInAsync(). В качестве параметра он принимает схему аутентификации, которая была использована при установки middleware app.UseCookieAuthentication в классе Startup. То есть в нашем случае это строка "Cookies". А в качестве второго параметра передается объект ClaimsPrincipal, который представляет пользователя.

Для правильного создания и настройки объекта ClaimsPrincipal вначале создается список claims - набор данных, которые шифруются и добавляются в аутентификационные куки. Каждый такой claim принимает тип и значение. В нашем случае у нас только один claim, который в качестве типа принимает константу ClaimsIdentity.DefaultNameClaimType, а в качестве значения - email пользователя.

И после вызова метода расширения HttpContext.SignInAsync в ответ клиенту будут отправляться аутентификационные куки, которые при последующих запросах будут передаваться обратно на сервер, десериализоваться и использоваться для аутентификации пользователя.

Метод Register сделан аналогично методу Login, только теперь мы получаем данные регистрации через объект RegisterModel и перед аутентификацией сохраняем эти данные в базу данных.

Подробнее: https://metanit.com/sharp/aspnet5/15.1.php

Ключевым инструментом для авторизации является атрибут AuthorizeAttribute из пространства имен Microsoft.AspNetCore.Authorization. Например, в прошлой теме данный атрибут ограничивал доступ к методу Index контроллера HomeController:

```
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.AspNetCore.Authorization;

public class HomeController: Controller
{
    [Authorize]
    public IActionResult Index()
    {
        return Content(User.Identity.Name);
    }
    // остальные методы
}
```

В этом случае доступ к методу Index имеют только те пользователи, которые залогинились в приложении. Анонимные пользователи же в данном случае при доступе к методу Index будут пер адресованы на форму входа в приложение.

Свойство HttpContext.User представляет объект интерфейса IPrincipal, который определен в пространстве имен System.Security.Principal. Этот интерфейс определяет метод IsInRole() и свойство Identity.

Свойство Identity возвращает объект интерфейса IIdentity, который связан с текущим запросом.

Метод IsInRole() в качестве параметра принимает роль и возвращает true, если текущий пользователь принадлежит данной роли.

Объект IIdentity, в свою очередь, предоставляет информацию о текущем пользователе через следующие свойства:

- AuthenticationType: тип аутентификации в строковом виде
- IsAuthenticated: возвращает true, если пользователь аутентифицирован
- Name: возвращает имя пользователя. Как правило, в качестве подобного имени используется логин, по которому пользователь входит в приложение

Для определения аутентифицирован ли пользователь, ASP.NET Core использует аутентификационные куки.

Мы можем применять эти свойства в контроллерах или представлениях.

```
public IActionResult Index()
{
  If (User.Identity.IsAuthenticated)
  {
    return Content(User.Identity.Name);
  }
  return Content("не аутентифицирован");
}
```

Подробнее: https://metanit.com/sharp/aspnet5/15.3.php