//6

**1. Что такое авторизация**

Авторизация — это процесс определения, имеет ли пользователь право на выполнение определенных действий или доступ к ресурсам в системе. Она отвечает на вопрос "Что может делать пользователь?" Например, доступ к определённым страницам, выполнение операций (например, редактирование или удаление данных) и т.д. Авторизация обычно происходит после аутентификации.

**2. Что такое аутентификация**

Аутентификация — это процесс проверки подлинности пользователя, т.е. подтверждение его личности. Это может быть сделано с помощью логина и пароля, а также других методов, таких как двухфакторная аутентификация, отпечатки пальцев и т.д. Аутентификация отвечает на вопрос "Кто вы?".

**3. Что такое идентификация**

Идентификация — это процесс определения пользователя в системе. Обычно это включает в себя представление пользователя (например, имя пользователя или адрес электронной почты) и процесс аутентификации для подтверждения его личности. Идентификация — первый шаг перед аутентификацией.

**4. Что такое Identity API**

Identity API — это часть ASP.NET Core, которая предоставляет функциональность для управления пользователями, ролями и аутентификацией. Она включает в себя создание, удаление и управление пользователями, а также управление ролями и правами доступа. Identity API предлагает разработчикам готовые решения для интеграции функций безопасности в приложения.

**5. \_roleManager**

\_roleManager — это экземпляр класса RoleManager<T>, который используется для управления ролями в системе. Он предоставляет методы для создания, удаления и проверки ролей, а также для добавления ролей к пользователям. Использование RoleManager позволяет управлять доступом и разрешениями в приложении.

**6. \_userManager**

\_userManager — это экземпляр класса UserManager<T>, который используется для управления пользователями в системе. Он предоставляет методы для создания, удаления, обновления и аутентификации пользователей. UserManager также управляет паролями, подтверждением электронной почты и другими аспектами, связанными с пользователями.

**7. Подключение Identity API**

Для подключения Identity API в ASP.NET Core необходимо выполнить следующие шаги:

* Установить NuGet пакеты, такие как Microsoft.AspNetCore.Identity.EntityFrameworkCore (чтобы работать с бд как с объектами).
* Настроить контекст базы данных для использования Identity в методе ConfigureServices в классе Startup.
* Добавить необходимые сервисы для работы с аутентификацией и авторизацией, например:

csharp

Copy

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

**8. Пакеты Identity API**

Основные пакеты, которые обычно используются для работы с Identity API в ASP.NET Core:

* Microsoft.AspNetCore.Identity: основной пакет для работы с аутентификацией и авторизацией.
* Microsoft.AspNetCore.Identity.EntityFrameworkCore: пакет для интеграции Identity с Entity Framework Core.
* Дополнительные пакеты для двухфакторной аутентификации, подтверждения электронной почты и т.д.

**9. Где прописываются роли и права (гость/админ)**

Роли и права обычно прописываются в базе данных. При использовании Identity API роли могут быть созданы через RoleManager в коде или миграциями. Также можно настроить права доступа с помощью атрибутов в контроллерах, например:

csharp

Copy

[Authorize(Roles = "Administrator")]

public class AdminController : Controller

**10. Всё про миграцию**

Миграция в контексте Entity Framework Core — это способ управления изменениями в структуре базы данных. Это позволяет разработчикам:

* Создавать и изменять схемы базы данных с помощью кодовых изменений, а не вручную.
* Использовать команды, такие как Add-Migration, для создания новой миграции, которая отражает изменения в моделях данных.
* Применять миграции к базе данных с помощью команды Update-Database, что обновляет структуру базы данных в соответствии с последними миграциями.

Миграции помогают поддерживать синхронизацию между кодом и базой данных, упрощая управление изменениями и обеспечивая возможность отката изменений при необходимости.

//7

Автоизация

Аунтификация

индетификация

сессия , жизненный цикл

показать где сохраняется id сессии

секретный способ демонстрация

строка подключения бд

модели бд

инкременты … все методы

Сессия — это механизм, используемый в веб-приложениях для хранения информации о взаимодействии пользователя с приложением на протяжении определенного времени. Сессия позволяет сохранять данные между запросами от одного и того же клиента (пользователя), что дает возможность поддерживать состояние приложения.

**2) Жизненный цикл сессии**

Жизненный цикл сессии включает несколько ключевых этапов:

**1. Создание сессии**

* **Запрос клиента**: Когда пользователь впервые обращается к веб-приложению, сервер получает запрос без связанной сессии. В заголовках запроса нет идентификатора сессии (например, заголовок Cookie пуст).
* **Создание новой сессии**: Сервер создает новую сессию и генерирует уникальный идентификатор сессии.
* **Отправка идентификатора сессии**: Сервер отправляет идентификатор сессии клиенту, устанавливая соответствующую куку в заголовке ответа:

Copy

Set-Cookie: ASP.NET.SessionId=abc123; Path=/; HttpOnly

**2. Хранение данных**

* **Добавление данных в сессию**: В процессе взаимодействия с приложением сервер может сохранять данные, связанные с пользователем, в сессии. Эти данные могут быть добавлены или изменены в любое время.
* **Заголовки не изменяются**: На этом этапе заголовки HTTP не изменяются, поскольку данные хранятся на сервере и не отправляются клиенту.

**3. Использование сессии**

* **Отправка запросов**: Пользователь взаимодействует с приложением, отправляя запросы. В каждом запросе клиент отправляет куки, содержащие идентификатор сессии.
* **Доступ к данным**: Сервер использует идентификатор сессии, полученный из заголовка Cookie, для доступа к связанным данным:

Copy

Cookie: ASP.NET.SessionId=abc123

* **Поддержка состояния**: Сервер может извлекать данные, связанные с этой сессией, и использовать их для обработки запросов пользователя.

**4. Истечение срока действия**

* **Мониторинг активности**: Сервер отслеживает активность сессии. Если пользователь не взаимодействует с приложением в течение установленного времени (например, 20 минут), сессия может истечь.
* **Удаление данных**: По истечении срока действия сервер может удалить данные, связанные с сессией. Если пользователь снова попытается взаимодействовать с приложением, сервер не найдет идентификатор сессии, и будет создана новая сессия.

**5. Завершение сессии**

* **Завершение по запросу**: Пользователь может явно завершить сессию, например, при выходе из системы. В этом случае сервер очищает все данные, связанные с сессией.
* **Удаление куки**: Сервер отправляет обновленный заголовок Set-Cookie, чтобы удалить идентификатор сессии:

Copy

Set-Cookie: ASP.NET.SessionId=abc123; Path=/; HttpOnly; Expires=Thu, 01 Jan 1970 00:00:00 GMT

* **Завершение сессии**: После этого все данные, связанные с сессией, удаляются как на сервере, так и на клиенте.

**Куки** (или cookie) — это небольшие текстовые файлы, которые веб-сервер отправляет на устройство пользователя (например, компьютер, смартфон) и которые хранятся в браузере. Куки используются для хранения информации о пользователе и его предпочтениях, что позволяет веб-сайтам запоминать состояние между запросами.

Куки, заголовок Cookie – клиент, сессии, Set-Cookie - сервер