**AccountController:**

Поскольку в классе *Startup* были добавлены сервисы *Identity*, то здесь в контроллере через конструктор мы можем их получить. В данном случае мы получаем сервис по управлению пользователями - *UserManager* и сервис *SignInManager*, который позволяет аутентифицировать пользователя и устанавливать или удалять его куки.

С помощью метода *userManager*. *CreateAsync* пользователь добавляется в базу данных. В качестве параметра передается сам пользователь и его пароль.

Данный метод возвращает объект *IdentityResult*, с помощью которого можно узнать успешность выполненной операции. Вполне возможно, что переданные значения не удовлетворяют требованиям, и тогда пользователь не будет добавлен в базу данных. В случае удачного добавления с помощью метода *signInManager*. *SignInAsync*() устанавливаем аутентификационные куки для добавленного пользователя. В этот метод передаётся объект пользователя, который аутентифицируется, и логическое значение, указывающее, надо ли сохранять куки в течение продолжительного времени. И далее выполняем переадресацию на главную страницу приложения.

Если добавление прошло неудачно, то добавляем к состоянию модели с помощью метода *ModelState* все возникшие при добавлении ошибки, и отправленная модель возвращается в представление.

В Get-версии метода *Login* мы получаем адрес для возврата в виде параметра *returnUrl* и передаём его в модель *LoginViewModel*.

В Post-версии метода *Login* получаем данные из представления в виде модели *LoginViewModel*. Всю работу по аутентификации пользователя выполняет метод *signInManager.PasswordSignInAsync().* Этот метод принимает логин и пароль пользователя. Третий параметр метода указывает, надо ли сохранять устанавливаемые куки на долгое время.

Данный метод также возвращает *IdentityResult*, с помощью которого можно узнать, завершилась ли аутентификация успешно. Если она завершилось успешно, то используем свойство *ReturnUrl* модели *LoginViewModel* для возврата пользователя на предыдущее место. Для этого нужно ещё удостовериться, что адрес возврата принадлежит приложению с помощью метода *Url*.*IsLocalUrl()*. Это позволит избежать перенаправлений на нежелательные сайты. Если же адрес возврата не установлен или не принадлежит приложению, выполняем переадресацию на главную страницу.

Третий метод - метод *Logout* выполняет выход пользователя из приложения. За выход отвечает метод *signInManager.SignOutAsync()*, который очищает аутентификационные куки.

Если оба метода контроллера принимают одинаковый(-е) параметр(-ы), тогда мы не можем назвать методы одинаково. И поэтому один метод должен быть назван отлично от другого. Однако используемый над методом Get атрибут ActionName("MethodName") указывает, что этот метод также относится к действию MethodName, и поэтому мы к нему можем обращаться с запросом Home/MethodName.

**Разные View:**

Для валидации *на стороне клиента* применяется класс ValidationMessageTagHelper. Данный tag-хелпер используется с помощью применения к элементу <span > атрибута asp-validation-for. Атрибут *asp-validation-for* в качестве значения принимает название свойства модели, для которого будет выводиться сообщение об ошибке валидации.

ValidationSummaryTagHelper применяется для отображения сводки ошибок валидации. Он применяется к элементу div в виде атрибута *asp-validation-summary*:

|  |  |
| --- | --- |
| 1 | <div class="validation" asp-validation-summary="ModelOnly"/> |

В качестве значения атрибут asp-validation-summary принимает одно из значений перечисления ValidationSummary:

***None***: ошибки валидации не отображаются

***ModelOnly***: отображаются только ошибка валидации уровня модели, ошибки валидации для отдельных свойств не отображаются

***All***: отображаются все ошибки валидации

Эти тег-хелперы работают в паре с атрибутами модели (например, [Required(ErrorMessage = "Не указан электронный адрес")], [EmailAddress(ErrorMessage = "Некорректный электронный адрес")])

В форме в представлении для создания нового объекта или редактирования нужно вставить атрибут asp-route-id="@Model.Id". Установка этого атрибута позволяет при отправке данных вместе с ними также отправить и значение Model.Id. Без этого значения Entity Framework не сможет обновить соответствующую модель в базе данных.

Если параметр берётся из запроса, на форме нет необходимости указывать этот параметр.

Параметр asp-route-nameParameter - имя параметра, который передаётся в метод контроллера. В контроллере во входных параметрах можно задать значение по умолчанию:

public ActionResult Method(nameParameter =nameParameter)