1. **Поясните понятие «внедрение зависимости» (DI).**

Внедрение зависимости ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Dependency injection*, DI) — процесс предоставления внешней зависимости [программному компоненту](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Является специфичной формой «[инверсии управления](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F)» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Inversion of control*, IoC), когда она применяется к управлению зависимостями. В полном соответствии с [принципом единственной ответственности](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%B5%D0%B4%D0%B8%D0%BD%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B9_%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8) объект отдаёт заботу о построении требуемых ему зависимостей внешнему, специально предназначенному для этого общему механизму.

1. **Для чего применяется DI?**

Dependency injection (DI) или внедрение зависимостей представляет механизм, который позволяет сделать взаимодействующие в приложении объекты слабосвязанными. Такие объекты связаны между собой через абстракции, например, через интерфейсы, что делает всю систему более гибкой, более адаптируемой и расширяемой.

Нередко для установки зависимостей в подобных системах используются специальные контейнеры - IoC-контейнеры (Inversion of Control). Такие контейнеры служат своего рода фабриками, которые устанавливают зависимости между абстракциями и конкретными объектами и, как правило, управляют созданием этих объектов.

1. **Что такое Ninject?**

**Ninject -** это мощный и одновременно легкий в использовании IoC контейнер. По простоте использования, а также понятности api этот контейнер чем-то напоминает Autofac. Данный контейнер имеет очень мощную поддержку в онлайн-обществе.

**IoC-контейнер —** это какая-то библиотека, фреймворк, программа если хотите, которая позволит вам упростить и автоматизировать написание кода с использованием данного подхода на столько, на сколько это возможно.

Если сравнить с более низкоуровневыми технологиями, IoC-контейнер — это компоновщик, который собирает не объектные файлы, а объекты ООП (экземпляры класса) во время исполнения программы. Очевидно, для реализации подобной идеи было необходимо создать не только сам компоновщик, но и фабрику, производящую объекты. Аналогом такого компоновщика (естественно, более функциональным) является компилятор, одной из функций которого является создание объектных файлов.

1. **Поясните принцип подключения и конфигурации Ninject в ASP.NET-приложении.**

Первый этап заключается в подготовке Ninject к использованию. Для этого создается экземпляр ядра Ninject, который представляет собой объект, ответственный за распознавание зависимостей и создание новых объектов. Когда возникает потребность в каком-либо объекте, вместо применения ключевого слова new производится обращение к ядру.

Необходимо создать реализацию интерфейса Ninject.IKernel, что делается конструированием нового экземпляра класса StandardKernel. Библиотека Ninject может быть расширена и настроена для работы с различными видами ядра. (На самом деле, вы можете годами пользоваться Ninject и иметь дело только со StandardKernel.)

Второй этап процесса состоит в конфигурировании ядра Ninject с целью предоставления сведений о том, какие объекты реализации должны применяться для каждого интерфейса, с которым придется работать.

Последний этап — это действительное использование Ninject, что делается посредством метода Get() ядра

1. **Каким образом может быть выполнено DI в ASP.NET-приложении.**

DI/Ninject: прямое(непосредственное) применение DI.

1. **Назовите четыре вида DI, которые позволяет сделать Ninject в ASP.NET-приложении и поясните их особенности.**

**DI/Ninject:** прямое(непосредственное) применение DI.

**DI/Ninject:** глобальная регистрации зависимостей, **Global.asax**

**DI/Ninject:**  [inject] for properties

**DI/Ninject:**  [inject] for methods

1. **Поясните понятие «хэлпер».**

Tag-хелперы представляют собой функциональность, предназначенную для генерации HTML-разметки. Tag-хелперы используются в представлениях и выглядят как обычные html-элементы или атрибуты, однако при работе приложения они обрабатываются движком Razor на стороне сервера и в конечном счете преобразуются в стандартные html-элементы.

1. **Объясните последовательность разработки внутренних и внешних хэлперов.**

Встраиваемые хелперы объявляются с помощью дескриптора @helper в любом месте представления. Использоваться они могут только в том представлении, в котором объявлены.

Внешние создаются в отдельных файлах в виде методов расширений класса HtmlHelper. Расширять они могут как стандартные хелперы, так и встраиваемые пользовательские.

1. **Перечислите стандартные хэлперы входящие в состав ASP.NET.**

AnchorTagHelper представляет тег-хелпер, который позволяет создавать ссылки. Он может принимать ряд специальных атрибутов:

* **asp-controller**: указывает на контроллер, которому предназначен запрос
* **asp-action**: указывает на действие контроллера
* **asp-area**: указывает на действие область, в которой расположен контроллер или страница RazorPage (если они находятся в отдельной области)
* **asp-page**: указывает на RazorPage, которая будет обрабатывать запрос
* **asp-page-handler**: указывает на обработчик страницы RazorPage, которая будет применяться для обработки запроса
* **asp-host**: указывает на домен сайта
* **asp-protocol**: определяет протокол (http или https)
* **asp-route**: указывает на название маршрута
* **asp-all-route-data**: устанавливает набор значений для параметров
* **asp-route-[название параметра]**: определяет значение для определенного параметра
* **asp-fragment**: определяет ту часть хэш-ссылки, которая идет после символа решетки #. Например, "paragraph2" в ссылке <http://mysite.com/#paragraph2>

**ScriptTagHelper**

Для подключения внешних файлов скриптов применяется тег-хэлпер **ScriptTagHelper**. Тег, представляющий данный класс, может принимать ряд атрибутов:

* **asp-append-version**: если имеет значение true, то к пути к файлу скрипта добавляется номер версии
* **asp-fallback-src**: указывает вспомогательный путь к скрипту, который используется, если загрузка скрипта, указанного в атрибуте src пройдет неудачно
* **asp-fallback-test**: определяет выражение, которое тестирует загрузку основного скрипта из атрибута src
* **asp-src-include**: определяет шаблон подключаемых файлов, через запятую можно задать несколько шаблонов
* **asp-src-exclude**: определяет через запятую набор шаблонов для тех файлов, которые следует исключить из загрузки
* **asp-fallback-src-include**: определяет через запятую набор шаблонов файлов, которые подключаются в том случае, если загрузка основного скрипта из атрибута src прошла неудачно
* **asp-fallback-src-exclude**: определяет через запятую набор шаблонов файлов, которые следует исключить из загрузки в том случае, если загрузка основного скрипта из атрибута src прошла неудачно

**LinkTagHelper**

Класс LinkTagHelper определяет тег link, который используется для подключения файлов стилей. Он применяет следующие атрибуты:

* **asp-append-version**: если имеет значение true, то к пути к названию файла стиля добавляется номер версии
* **asp-fallback-href**: указывает вспомогательный путь к файлу стиля, который используется, если загрузка файла, указанного в атрибуте href пройдет неудачно
* **asp-fallback-test-class**: определяет класс, который используется для теста загрузки стиля из атрибута href
* **asp-fallback-test-property**: определяет свойство, которое используется для тестирования загрузки стиля из атрибута href
* **asp-fallback-test-value**: определяет значение свойства из атрибута asp-fallback-test-property, которое используется для теста загрузки стиля из атрибута href
* **asp-href-include**: определяет через запятую набор шаблонов подключаемых файлов стилей
* **asp-href-exclude**: определяет через запятую набор шаблонов для тех файлов, которые следует исключить из загрузки
* **asp-fallback-href-include**: определяет через запятую набор шаблонов файлов, которые подключаются в том случае, если загрузка основного файла стиля из атрибута href прошла неудачно
* **asp-fallback-href-exclude**: определяет через запятую набор шаблонов файлов, которые следует исключить из загрузки в том случае, если загрузка основного файла стиля из атрибута href прошла неудачно

Так, для создания формы используется класс **FormTagHelper**, представленный тегом form. Этот тег может принимать следующие атрибуты:

* **asp-controller**: указывает на контроллер, которому предназначен запрос
* **asp-action**: указывает на действие контроллера
* **asp-area**: указывает на название области, в которой будет вызываться контроллер для обработки формы
* **asp-antiforgery**: если имеет значение true, то для этой формы будет генерироваться antiforgery token
* **asp-route**: указывает на название маршрута
* **asp-all-route-data**: устанавливает набор значений для параметров
* **asp-route-[название параметра]**: определяет значение для определенного параметра
* **asp-page**: указывает на страницу RazorPage, которая будет обрабатывать запрос
* **asp-page-handler**: указывает на обработчик страницы RazorPage, который применяется для обработки запроса
* **asp-fragment**: указывает фрагмент, который добавляется к запрашиваемому адресу после символа #.

**LabelTagHelper** использует тег label для создания метки:

**InputTagHelper** создает поле ввода:

**TextAreaTagHelper** используется для создания многострочного текстового поля textarea. Данный хелпер применяет только атрибут asp-for:

**SelectTagHelper** создает элемент списка:

Тег **EnvironmentTagHelper** используется для генерации определенной разметки html в зависимости от состояния приложения: находится ли приложение в процессе разработки, тестирования или уже опубликовано на сервере.

**CacheTagHelper** обертывает контент представления и позволяет кэшировать его в памяти сервера. Он использует тег <cache>, и весь контент, определенный внутри этого тега, кэшируется в памяти.