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

Dependency Injection: программный механизм, позволяющий в автоматическом режиме создавать программный объект, с заданными жизненным циклом, способом применения и областью действия.

Такие объекты связаны между собой через абстракции, например, через интерфейсы, что делает всю систему более гибкой, более адаптируемой и расширяемой.

**Для чего применяется DI?**

позволяет создавать слабосвязанные компоненты;

повторное применение кода, упрощает внесение изменений, упрощает тестирование;

DI: чаще всего внедряется contextDB или репозиторий модели данных;

IoC - Inversion of Control, принцип программирования, позволяющий снизить зависимость между компонентами программ; DI – один из способов реализации IoC.

**Что такое Ninject?**

прямое(непосредственное) применение DI.

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

InTransientScope() - по умолчанию - новый на каждый вызов

InSingletonScope() - паттерн Singleton - один на все вызовы

InThreadScope() - новый экземпляр на каждый поток

InRequestScope() - новый экземпляр на каждый запрос

**Поясните понятие «хэлпер».**

Вообще helper можно перевести с английского как "вспомогательный метод". И фактически html-хелперы представляют собой вспомогательные методы, цель которых - генерация html-разметки.

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

**Внешние хелперы** – это расширяющие методы класса HtmlhHelper. Они отделены от места собственного применения (в представлении) и создаются в отдельном файле.

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

Html.BeginForm

Html.TextBox

Html.TextArea

Html.Label

Html.Hidden

Html.Password

Html.RadioButton

Html.CheckBox

Html.DropDownList (Html.ListBox для множественного выбора из списка)

Html.GetEnumSelectList

**Используя различные методы внедрения зависимостей, можно управлять жизненным циклом создаваемых сервисов. Сервисы, которые создаются механизмом Depedency Injection, могут представлять один из следующих типов**:

* **Transient**: при каждом обращении к сервису создается новый объект сервиса. В течение одного запроса может быть несколько обращений к сервису, соответственно при каждом обращении будет создаваться новый объект. Подобная модель жизненного цикла наиболее подходит для легковесных сервисов, которые не хранят данных о состоянии
* **Scoped**: для каждого запроса создается свой объект сервиса. То есть если в течение одного запроса есть несколько обращений к одному сервису, то при всех этих обращениях будет использоваться один и тот же объект сервиса.
* **Singleton**: объект сервиса создается при первом обращении к нему, все последующие запросы используют один и тот же ранее созданный объект сервиса

**Внедрение зависимостей устраняет эти проблемы следующим образом:**

* Используется интерфейс или базовый класс для абстрагирования реализации зависимостей.
* Зависимость регистрируется в контейнере служб. ASP.NET Core предоставляет встроенный контейнер служб, [IServiceProvider](https://learn.microsoft.com/ru-ru/dotnet/api/system.iserviceprovider). Как правило, службы регистрируются в файле Program.cs приложения.
* Служба *внедряется* в конструктор класса там, где он используется. Платформа берет на себя создание экземпляра зависимости и его удаление, когда он больше не нужен.

**Рассмотрим поэтапно, как происходит в данном случае внедрение зависимостей:**

1. Фреймворк MVC получает запрос и обращается к контроллеру HomeController
2. Фреймворк MVC обращается к классу сопоставления зависимостей (в данном случае класс NinjectDependencyResolver), чтобы тот создал новый объект HomeController, передавая параметр Type в метод GetService (в класс NinjectDependencyResolver)
3. Сопоставитель зависимостей вызывает инфраструктуру Ninject для создания нового объекта HomeController, передавая тип создаваемого объекта в метод TryGet
4. Ninject смотрит на конструктор HomeController и видит, что там используется зависимость от интерфейса IRepository, для которого он устанавливает сопоставление с конкретной реализацией
5. Ninject создает экземпляр класса BookRepository и затем использует его для создания контроллера HomeController
6. Ninject передает созданный объект HomeController сопоставителю зависимостей, который, в свою очередь, передает его фреймворку MVC. И далее происходит обработка запроса.

**Строго типизированный хелпер** похож на обычный, только в конце прибавляется суффикс **For**: LabelFor. Так как строго типизированные хелперы могут использоваться только в строго типизированных представлениях, то вначале представления указываем модель, которая будет использоваться: @model HtmlHelpersApp.Models.User. То есть, в вызове @Html.TextBoxFor(m=>m.Name) параметр m представляет переменную модели User. А лямбда-выражение m=>m.Age указывает, что данный хелпер будет генерировать поле для свойства Age

**Кроме обычных представлений метод действия может также возвращать частичные представления**. Их отличительной особенностью является то, что их можно встраивать в другие обычные представления.