Question Answering Bonus

1. How can we handle security issues concerning connection strings written as hardcode ? Name 3 approaches to do this.

AJAX stands for Asynchronous JavaScript and XML. It is a web development technique that allows web pages to be updated asynchronously by exchanging data with a server behind the scenes, so that parts of a web page can be updated without reloading the entire page.

Three Reasons to Use AJAX in MVC Apps

1. **Improved User Experience:** AJAX allows parts of the page to update asynchronously without blocking the entire page, making web applications feel faster and more responsive.
2. **Reduced Server Load and Bandwidth:** Instead of reloading the entire page, only necessary data is fetched and updated, reducing the server workload and the amount of data transferred.
3. **More Interactive and Dynamic UI:** AJAX enables interactive features such as live searching, filtering, or form validation without full page refresh, making the application more engaging.

Simple AJAX Demo in ASP.NET MVC

Controller:

```
public class AjaxDemoController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public JsonResult GetCurrentTime()
    {
        string currentTime = DateTime.Now.ToString("HH:mm:ss");
```

```
        return Json(currentTime, JsonRequestBehavior.AllowGet);
    }
}
```

View (Index.cshtml):

```
@{
    ViewBag.Title = "AJAX Demo";
}
<h2>AJAX Demo</h2>
<button id="getTimeBtn">Get Current Time</button>
<div id="result"></div>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
    $(document).ready(function () {
        $("#getTimeBtn").click(function () {
            $.ajax({
                type: "GET",
                url: '@Url.Action("GetCurrentTime", "AjaxDemo")',
                success: function (data) {
                    $("#result").html("Current Time: " + data);
                },
                error: function () {
                    alert("Error retrieving the time.");
                }
            });
        });
    });
</script>
```

In this demo, clicking the button triggers an AJAX GET request to the server action GetCurrentTime, which returns the current server time as JSON. The view updates only the div content without refreshing the entire page.

This illustrates the core AJAX benefit of partial updates with asynchronous calls in an MVC application.

2. If we consider Dependency injection as a creational design pattern. make a LinkedIn article about Creational Design Patterns.

Understanding Creational Design Patterns: Including Dependency Injection

In software development, designing code that is flexible, maintainable, and scalable is a top priority. Creational design patterns are one of the primary categories of design patterns that address object creation mechanisms. They help abstract the instantiation process, allowing a system to be independent of how its objects are created, composed, and represented.

What Are Creational Design Patterns?

Creational patterns provide various ways to create objects while hiding the creation logic from the client code. This promotes loose coupling and enhances code reuse. Common examples include:

- **Singleton:** Ensures a class has only one instance and provides a global point of access to it.
- **Factory Method:** Defines an interface for creating an object but lets subclasses decide which class to instantiate.
- **Abstract Factory:** Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
- **Builder:** Separates the construction of a complex object from its representation, allowing the same construction process to create different representations.
- **Prototype:** Creates new objects by copying an existing object, known as the prototype.

Is Dependency Injection a Creational Pattern?

While Dependency Injection (DI) is often referred to as a design pattern related to inversion of control, it fundamentally deals with how objects acquire their dependencies. It separates object construction and dependency provision from business logic, thereby reducing tight coupling and improving testability.

DI can be seen as a modern creational pattern because:

- It controls the creation of dependencies externally and injects them into dependent objects.
- It allows flexible configurations of an object's dependencies without changing its code.
- It greatly simplifies maintaining and extending large applications by effectively managing object creation and wiring.

Why Use Creational Patterns Like DI?

- **Loose Coupling:** Classes depend on abstractions (interfaces) rather than concrete implementations, making it easier to modify or swap components without affecting dependent code.
- **Improved Testability:** Dependencies can be replaced with mocks or stubs during testing, allowing isolated unit tests.
- **Better Maintainability:** Centralized control over object creation helps manage complexity and application configuration.
- **Adaptability:** Easily supports multiple implementations and dynamic configuration changes at runtime.

Conclusion

Creational design patterns are essential tools for writing clean, flexible, and scalable code. Dependency Injection stands out as a powerful creational approach in today's software development, enabling better separation of concerns and enhancing maintainability.

By embracing these patterns, developers can build robust systems prepared for growth, change, and testing efficiency.