**🔹 السؤال 4**

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

**الإجابة:**

1. **استخدام appsettings.json**  
   ضع connection string في appsettings.json بدلًا من الكود مباشرة.
2. {
3. "ConnectionStrings": {
4. "DefaultConnection": "Server=.;Database=SchoolDB;Trusted\_Connection=True;"
5. }
6. }

ثم استدعِها في Program.cs أو Startup.cs:

builder.Configuration.GetConnectionString("DefaultConnection");

1. **استخدام Secret Manager**  
   أثناء التطوير، خزّن القيم الحساسة باستخدام secret manager:
2. dotnet user-secrets set "ConnectionStrings:DefaultConnection" "Server=.;Database=SchoolDB;Trusted\_Connection=True;"
3. **استخدام Environment Variables أو Azure Key Vault**  
   خزّن القيم الحساسة في متغيرات بيئة أو في خدمة مثل **Azure Key Vault**.

**🔹 السؤال 5 (Bonus)**

**What is AJAX? Mention 3 reasons to use it in MVC apps. Implement in simple demo.**

**الإجابة:**

* **AJAX** = Asynchronous JavaScript and XML  
  → تسمح بإرسال واستقبال البيانات من السيرفر بدون إعادة تحميل الصفحة بالكامل.

**3 أسباب لاستخدامها:**

1. تحسين تجربة المستخدم (عدم إعادة تحميل الصفحة).
2. تقليل استهلاك النطاق الترددي (Bandwidth).
3. سرعة استجابة أعلى عند التفاعل مع الواجهة.

**Demo بسيط:**

<!-- View -->

<button id="btn">Load Data</button>

<div id="result"></div>

<script>

$("#btn").click(function () {

$.ajax({

url: "/Student/GetAll",

type: "GET",

success: function (data) {

$("#result").html(data);

}

});

});

</script>

// Controller

public IActionResult GetAll()

{

var students = \_context.Students.ToList();

return PartialView("\_StudentList", students);

}

**🔹 السؤال 6 (Bonus)**

**If we consider Dependency Injection as a creational design pattern, make a LinkedIn article about Creational Design Patterns.**

**ملخص مقال LinkedIn مقترح:**

**🧠 Creational Design Patterns in .NET – Why Dependency Injection Matters**

In software engineering, **Creational Design Patterns** focus on how objects are created while hiding the creation logic from the user.  
This makes systems **more flexible** and **loosely coupled**.

**Examples include:**

* Singleton
* Factory Method
* Abstract Factory
* Builder
* Prototype
* **Dependency Injection (DI)**

**Dependency Injection** stands out as a pattern that promotes **Inversion of Control (IoC)**, where the dependencies are **provided** instead of being **created internally**.

In ASP.NET Core, DI is built-in:

builder.Services.AddScoped<IStudentRepository, StudentRepository>();

This makes your app easier to test, maintain, and extend.

**Conclusion:**  
Creational patterns are the backbone of clean architecture. Mastering them — especially Dependency Injection — is a must for every serious developer.