Here are common interview questions and answers for MVC, .NET Core, and Entity Framework. This information is intended to help you prepare for a technical interview, and you should always be ready to provide more detail, code examples, and real-world scenarios based on your experience.

**MVC (Model-View-Controller)**

**1. What is the MVC design pattern?**

* **Answer:** MVC is a software architectural pattern that separates an application into three interconnected components:
  + **Model:** Represents the application's data and business logic. It handles data manipulation, validation, and communication with the database.
  + **View:** The user interface (UI) of the application. It displays the data from the Model and sends user input to the Controller.
  + **Controller:** The component that handles user input. It interacts with the Model to update data and selects the appropriate View to display the results. It acts as the intermediary between the Model and the View.

**2. What are the advantages of using ASP.NET MVC?**

* **Answer:**
  + **Separation of Concerns:** It promotes a clean separation of the UI, business logic, and data, making the application easier to maintain and extend.
  + **Testability:** Because of the separation of concerns, it's easier to perform unit testing on the Model and Controller components without needing to run the UI.
  + **Control over HTML:** Unlike Web Forms, MVC gives developers full control over the generated HTML, CSS, and JavaScript. This is great for front-end development, SEO, and responsiveness.
  + **RESTful URLs:** MVC's routing system allows for clean, "hackable" URLs that are more user-friendly and better for search engine optimization.
  + **Lightweight:** It does not use ViewState, which makes the pages lighter and faster to load.

**3. How do you pass data from a Controller to a View in MVC?**

* **Answer:** There are several ways to pass data:
  + **ViewData:** A dictionary-like object that uses a string key. It's a dynamic type, so there's no compile-time checking. Data is cleared after the request is complete.
  + **ViewBag:** A dynamic object that is a wrapper around ViewData. It provides a more convenient syntax for accessing data (e.g., ViewBag.Name instead of ViewData["Name"]). It also has no compile-time checking.
  + **TempData:** A dictionary that stores data for one subsequent request. It's useful for passing data between actions, especially after a redirect. It uses session or cookies to persist data.
  + **Strongly Typed Views (ViewModel)**: This is the recommended approach. You create a C# class (a ViewModel) to hold the data required by the View. The Controller populates this object and passes it to the View, which is strongly typed to that ViewModel. This provides compile-time checking and is more maintainable.

**4. Explain the concept of Routing in ASP.NET MVC.**

* **Answer:** Routing is a system that maps incoming HTTP requests to specific controller actions. It allows for "pretty URLs" that don't have to correspond to physical files. The routes are defined in the Program.cs or Startup.cs file using the MapControllerRoute method. A default route often follows the pattern {controller}/{action}/{id?}, where the id is an optional parameter.

**5. What is a ViewModel, and why is it important?**

* **Answer:** A ViewModel is a C# class specifically designed to hold the data that a View needs. It is important because:
  + **Separation of Concerns:** It decouples the View from the domain model, preventing the exposure of unnecessary data or business logic.
  + **Strong Typing:** It provides type safety and compile-time checking.
  + **Data Shaping:** You can shape the data from multiple models or sources into a single object that the View can easily consume, simplifying the View's logic.

**.NET Core**

**1. What is .NET Core, and how is it different from the .NET Framework?**

* **Answer:** .NET Core is a modern, cross-platform, open-source framework for building web, cloud, and console applications. The key differences from the .NET Framework are:
  + **Cross-platform:** .NET Core runs on Windows, macOS, and Linux, while the .NET Framework is Windows-only.
  + **Open-Source:** .NET Core is open-source and community-driven, while the .NET Framework is primarily closed-source.
  + **Modular:** .NET Core is modular and lightweight. You only include the necessary components, which reduces the application's footprint.
  + **Performance:** .NET Core is generally faster and more performant than the .NET Framework.
  + **.NET 5+:** .NET 5 and newer versions unify .NET Core and .NET Framework into a single platform, with .NET 5+ being the future.

**2. Explain Middleware in ASP.NET Core.**

* **Answer:** Middleware components are software that are assembled into a request pipeline to handle requests and responses. Each middleware component can perform a specific task, such as logging, authentication, routing, or error handling. The order in which you add middleware to the pipeline is crucial, as it determines the order of execution.

**3. How does Dependency Injection (DI) work in .NET Core?**

* **Answer:** .NET Core has built-in support for Dependency Injection. It is a design pattern that allows you to inject dependencies (services or objects) into a class rather than having the class create them itself. This promotes loose coupling and makes the code more testable.
  + You register services with the built-in DI container in the Startup.cs or Program.cs file using methods like AddSingleton, AddScoped, and AddTransient.
  + The DI container is responsible for creating instances of these services and injecting them into the constructors of classes that request them.

**4. What is the difference between AddSingleton, AddScoped, and AddTransient in DI?**

* **Answer:** These are the three main service lifetimes:
  + **AddTransient:** A new instance of the service is created every time it's requested from the container. It's suitable for lightweight, stateless services.
  + **AddScoped:** A single instance of the service is created per client request (or scope). It's useful for services that need to maintain state for the duration of a single request.
  + **AddSingleton:** A single instance of the service is created for the entire application's lifetime. It's suitable for services that are stateless or need to be shared across the entire application.

**5. What is Kestrel, and what is its role?**

* **Answer:** Kestrel is a cross-platform web server for ASP.NET Core. It is the default and a high-performance web server that runs your ASP.NET Core application. It can be run as a standalone server or behind a more feature-rich reverse proxy server like IIS or Nginx.

**Entity Framework (EF)**

**1. What is Entity Framework (EF), and what is its purpose?**

* **Answer:** Entity Framework is an Object-Relational Mapping (ORM) framework for .NET. Its purpose is to allow developers to interact with a database using C# objects (entities) rather than writing raw SQL queries. It handles the "mapping" between the C# objects and the database tables, which simplifies data access and reduces boilerplate code.

**2. Explain the different development approaches in Entity Framework.**

* **Answer:**
  + **Database First:** You start with an existing database. EF reverse-engineers the database schema to generate the C# entity classes and a DbContext for you.
  + **Code First:** You start by defining your C# classes (the domain model). EF then generates the database schema from these classes. This is the most common approach in modern development.
  + **Model First (older approach):** You create a visual model of your entities and their relationships in the EDMX designer. EF then generates both the C# classes and the database from this model. This approach is less common now.

**3. What are Migrations in Entity Framework Core?**

* **Answer:** Migrations are a feature that allows you to manage and evolve your database schema over time. When you make changes to your C# entity classes (e.g., add a new property, change a data type), you can create a migration. This migration creates a C# class with code that describes how to update the database to match your new model. You can then apply this migration to your database, and EF will execute the necessary SQL commands to update the schema.

**4. Explain Lazy Loading vs. Eager Loading in Entity Framework.**

* **Answer:** These concepts relate to how related data (e.g., an Order and its related OrderItems) is loaded from the database.
  + **Lazy Loading:** Related entities are not loaded initially. They are loaded automatically and on-demand the first time a navigation property is accessed. This can result in many separate database queries.
  + **Eager Loading:** Related entities are loaded from the database along with the main entity in a single query. You use the Include() method to specify which related entities you want to load. This reduces the number of database round trips and is generally more efficient.

**5. What is the DbContext class, and what is the DbSet class?**

* **Answer:**
  + **DbContext:** The DbContext class is the primary class for interacting with the database. It represents a session with the database and provides a gateway to query and save entities. It tracks changes to the entities and manages transactions.
  + **DbSet<T>:** The DbSet<T> class represents a collection of a specific entity type in the database. For example, DbSet<Product> would represent the Products table. You use DbSet to perform CRUD (Create, Read, Update, Delete) operations.

Here are common interview questions and answers for Object-Oriented Programming (OOP) concepts. These questions are foundational and are often the first thing a technical interviewer will ask to gauge your understanding of programming paradigms.

**Core OOP Concepts**

**1. What is OOP (Object-Oriented Programming)?**

* **Answer:** OOP is a programming paradigm based on the concept of "objects." An object is a self-contained unit that combines data (state) and the procedures (behavior) that operate on that data. It aims to model real-world entities and their interactions, making code more modular, reusable, and easier to maintain.

**2. What are the four main pillars of OOP? Explain each one.**

* **Answer:** The four pillars are a foundational concept in OOP.
  1. **Encapsulation:** The bundling of data (attributes) and the methods (behaviors) that operate on that data into a single unit (a class). It also involves hiding the internal state of an object and requiring all interaction to be through the object's public methods. This prevents direct, unauthorized modification of an object's data.
  2. **Abstraction:** The process of simplifying complex systems by focusing on essential properties and behaviors while hiding unnecessary details. It allows you to create a "black box" that provides a clear, public interface for its users without revealing the intricate internal implementation. An abstract class or interface is a common way to achieve this.
  3. **Inheritance:** The mechanism by which one class acquires the properties and methods of another class. The class that is inherited from is called the "parent" or "base" class, and the class that inherits is called the "child" or "derived" class. Inheritance promotes code reuse and establishes a hierarchical relationship (IS-A relationship).
  4. **Polymorphism:** The ability of an object to take on many forms. It allows a single interface to be used for different types of objects, as long as they share a common base. This is often achieved through method overriding (runtime polymorphism) and method overloading (compile-time polymorphism).

**Detailed Questions on Each Pillar**

**3. Explain Encapsulation with a real-world example.**

* **Answer:** Think of a car. The car's internal workings (the engine, transmission, etc.) are complex and hidden from the driver. The driver only interacts with the car through a simple public interface: the steering wheel, accelerator, and brake pedal. You don't need to know how the engine works to drive the car. This is encapsulation. In code, you would have a Car class with private fields like speed and fuelLevel and public methods like accelerate() and brake().

**4. What is the difference between Abstraction and Encapsulation?**

* **Answer:**
  + **Abstraction** focuses on hiding the complexity and presenting a simplified interface to the user. It's about "what" the object does.
  + **Encapsulation** focuses on bundling data and methods together and restricting access to the internal state. It's about "how" the object is structured and protected.
  + **Analogy:** Abstraction is the blueprint of a house (showing the rooms but not the plumbing details). Encapsulation is the walls that keep the internal wiring and plumbing from being exposed.

**5. What is the difference between Method Overloading and Method Overriding?**

* **Answer:** This is a key polymorphism question.
  + **Method Overloading (Compile-time Polymorphism):** Occurs when a class has multiple methods with the same name but different parameters (number, type, or order of parameters). The compiler decides which method to call at compile time based on the arguments provided.
  + **Method Overriding (Runtime Polymorphism):** Occurs when a child class provides a specific implementation for a method that is already defined in its parent class. The overridden method must have the same name, same parameters, and same return type. The decision of which method to call is made at runtime based on the actual type of the object.

**6. What is an IS-A relationship vs. a HAS-A relationship?**

* **Answer:** These are two fundamental ways to represent relationships between objects.
  + **IS-A Relationship (Inheritance):** A child class is a more specific type of the parent class. For example, a Dog IS-A Animal. It is implemented using inheritance.
  + **HAS-A Relationship (Composition/Aggregation):** An object contains another object as a part of its state. For example, a Car HAS-A Engine. It is implemented using object composition, where one class holds a reference to an object of another class.

**Advanced Concepts**

**7. What is an Abstract Class? What is an Interface? What is the difference?**

* **Answer:**
  + **Abstract Class:** A class that cannot be instantiated directly. It's a blueprint that can contain both abstract methods (without a body) and concrete methods (with a body). Abstract classes are used when you have a common base for a group of related classes that need some shared functionality and some unique functionality.
  + **Interface:** A blueprint of a class that can only contain method signatures (no implementation). A class that implements an interface must provide an implementation for all of its methods. Interfaces are used to define a contract or a set of behaviors that a class must adhere to.
  + **Key Differences:**
    - **Implementation:** An abstract class can have method implementations; an interface cannot (in pre-Java 8 and C# 8, interfaces could have default implementations).
    - **Inheritance:** A class can inherit from only one abstract class but can implement multiple interfaces.
    - **Fields:** Abstract classes can have instance fields; interfaces cannot.

**8. What are static members? How are they different from instance members?**

* **Answer:**
  + **Static Members:** Belong to the class itself, not to any specific instance of the class. There is only one copy of a static member, shared by all objects of the class. They are accessed using the class name (ClassName.staticMethod()).
  + **Instance Members:** Belong to an instance (object) of the class. Each object has its own copy of an instance member. They are accessed using a specific object reference (myObject.instanceMethod()).
  + **Use Cases:** static members are often used for utility methods or data that doesn't depend on a specific object's state, such as a math helper function or a counter for the number of objects created.

**9. What is a constructor? What are its rules?**

* **Answer:** A constructor is a special type of method that is automatically called when an object of a class is created. Its primary purpose is to initialize the object's state.
  + **Rules:**
    - It must have the same name as the class.
    - It does not have a return type, not even void.
    - It can be overloaded.
    - If you don't provide a constructor, the compiler will provide a default, parameterless constructor.

**10. What is the difference between a class and an object?**

* **Answer:**
  + **Class:** A blueprint or a template for creating objects. It defines the properties (fields) and behaviors (methods) that an object of that type will have.
  + **Object:** An instance of a class. It's a concrete entity created from the class blueprint, with its own unique set of values for the properties defined in the class.

These questions cover a broad range of OOP concepts, from the fundamentals to more specific, practical applications. A strong answer for each will demonstrate a solid grasp of the paradigm's principles.