Writer: Kiran Kumar J

The document is to provide Frequently asked question in interviews and in meeting about ASP.NET MVC , Web APIS Entity Framework Data Access and syntaxes

asp.net mvc WEBapis entity frameworks FAQS

ASP.NET MVC WEBAPI EF DATA ACCESS FAQS

[ASP.NET MVC Web API EF Data Access FAQ’s 1](#_Toc205036716)

[MVC Fundamentals 1](#_Toc205036717)

[What is the Model-View-Controller (MVC) architectural pattern 1](#_Toc205036718)

[What are the advantages of using MVC for web application development? 3](#_Toc205036719)

[Core MVC Concepts 5](#_Toc205036720)

[Explain the role of Controllers in an ASP.NET MVC application? 5](#_Toc205036721)

[What is an Action Method in MVC? 5](#_Toc205036722)

[How does Routing work in ASP.NET Core, an open-source web development framework | .NET MVC? 5](#_Toc205036723)

# ASP.NET MVC Web API EF Data Access FAQ’s

# MVC Fundamentals

## What is the Model-View-Controller (MVC) architectural pattern

MVC is an architectural pattern that separates an application into three main logical components:

**Model** (data and business logic),

**View** (user interface), and

**Controller** (handles user input, interacts with Model and View).

It promotes separation of concerns and improves maintainability.

Controller is commutes between Model and View , When Client want to see specific data in a page meaning view we need to access first its controller then its action

The action method controls the logic that binds the data with the database for specific request and prepares the data in that methods sends back to the client view as response

Example Client want to see the customer information

Then need to access the url with CustomersController then what kind of action he want to execute whether its list , add , edit or delete customer information then specific action method will be executed through controllers action methods example client wants to see all the list of customers in his company then it should be List and type of method is Get .However MVC default creates the list of customers in Index ActionsResults

The List method will read all the list of customers with ORM model bind with all its table properties and make it as ActionResult or JsonResult or XMLResult and sendbacks to the View to display the list

View reads that Model and binds to the form here we need to use html helpers to bind data according to the requirement

Code Example :

Student Model

Represents the application's data (e.g., a Student):

public class Student

{

public int Id { get; set; }

public string Name { get; set; }

}

Student Controller

Handles incoming HTTP requests, fetches data, and selects the View to render

using System.Collections.Generic;

using System.Web.Mvc;

public class StudentController : Controller

{

public ActionResult Index()

{

// In real apps, fetch from database. Here, hardcoded list for illustration.

var students = new List<Student>

{

new Student {Id = 1, Name = "Kiran"},

new Student {Id = 2, Name = "Kumar"}

};

return View(students);

}

}

Student List View

Displays the student data (e.g., in Views/Student/Index.cshtml):

@model IEnumerable<Student>

<h2>Student List</h2>

<ul>

@foreach(var student in Model)

{

<li>@student.Name</li>

}

</ul>

Test

<http://localhost:25/Student/Index>

## What are the advantages of using MVC for web application development?

Advantages include

1. Separation of concerns,
2. Improved testability,
3. Better organization of code,
4. Easier maintenance, and
5. Support for asynchronous development.

Using the MVC (Model-View-Controller) pattern for web application development offers several advantages.

Following, each advantage is explained with C# ASP.NET MVC code snippets to illustrate how MVC delivers these benefits:

// 1. Separation of Concerns

MVC cleanly separates data (Model), presentation (View), and input logic (Controller), making each part easier to manage and modify independently

// Model

public class Product {

public int Id { get; set; }

public string Name { get; set; }

}

// Controller

public class ProductController : Controller {

public ActionResult Details(int id) {

var product = ProductRepository.GetById(id);

return View(product);

}

}

// View (Details.cshtml)

@model Product

@Model.Name

We can change how Product is displayed without altering the business logic or controller.

//2. Improved Testability

Since business logic and UI are separated, automated tests can focus on individual components (especially models and controllers),leading to more reliable applications

// Controller Action

public class ProductController : Controller {

private readonly IProductService \_service;

public ProductController(IProductService service) {

\_service = service;

}

public ActionResult Index() {

var products = \_service.GetAll();

return View(products);

}

}

// Test (using mock)

[TestMethod]

public void Index\_ReturnsAViewWithProducts() {

var mockService = new Mock();

mockService.Setup(s => s.GetAll()).Returns(new List());

var controller = new ProductController(mockService.Object);

var result = controller.Index() as ViewResult;

Assert.IsNotNull(result);

}

We can mock dependencies and test the controller in isolation.

//3. Better Organization of Code

MVC encourages structuring projects in clear directories and class hierarchies (Models, Views, Controllers),

making it easy to locate and manage code as the project grows

//\*\*Project Structure Example:\*\*

/Models/Product.cs

/Views/Product/Details.cshtml

/Controllers/ProductController.cs

> Large projects remain manageable and scalable.

//4. Easier Maintenance

Because models, views, and controllers are independent, you can update one part (like adding a new field in the model or changing the UI) without major changes elsewhere. This reduces bugs and makes updates faster

// Adding a new property only in the product model

public class Product {

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; } // New field

}

// Update the view to show Description. No controller logic changes needed.

//5. Support for Asynchronous Development

MVC frameworks (like ASP.NET MVC) natively support asynchronous operations, which is vital for scalable web apps

public async Task Index()

{

var products = await productService.GetAllAsync();

return View(products);

}

> This allows the application to handle more users efficiently, with better performance for operations like API calls or database queries.

# Core MVC Concepts

## Explain the role of Controllers in an ASP.NET MVC application?

Controllers are classes that handle incoming user requests.

They process user input, interact with the Model to retrieve or update data, and then decide which View to render to the user.

## What is an Action Method in MVC?

An Action Method is a public method within a Controller class that handles a specific incoming HTTP request (e.g., GET, POST).

It's the entry point for handling user interactions and returning a result (like a View, JSON, or redirect).

## How does Routing work in ASP.NET Core, an open-source web development framework | .NET MVC?

Routing is the mechanism that maps incoming URL requests to specific Controller action methods.

It defines URL patterns and associates them with controller-action pairs, allowing for clean, user-friendly URLs without needing actual physical files.