Provim Hyrje në Web Programim – Xhelal Jashari

1) Which of the following is TRUE?

a) Action method must be public method in a controller class.
b) Action method can be protected method in a controller class.
c) Action method can be private method in a controller class.
d) Action method can be static method in a controller class.

2) What is DRY principle in ASP.NET?

a) Both a and b.
b) Don't repeat yourself.
c) Don't revise yourself.

3) In an MVC web app, which attribute would you use to prohibit non-logged in users from accessing an action?

a) [Authenticated]
b) [Required]
c) [Authorize]
d) [LoggedIn]

4) In which file do we configure the web app, which contains the Configure and ConfigureServices methods? (Version .3,1)

a) Program.cs
b) Startup.cs
c) Middleware
d) Appsettings.json

5) Which of the following method of DbContext is used to save entities to the database?

a) Execute()
b) SaveChanges()
c) Save()
d) Add()

6) Name all the logical components (M,V,C) of the MVC architecture, and their responsibilities.

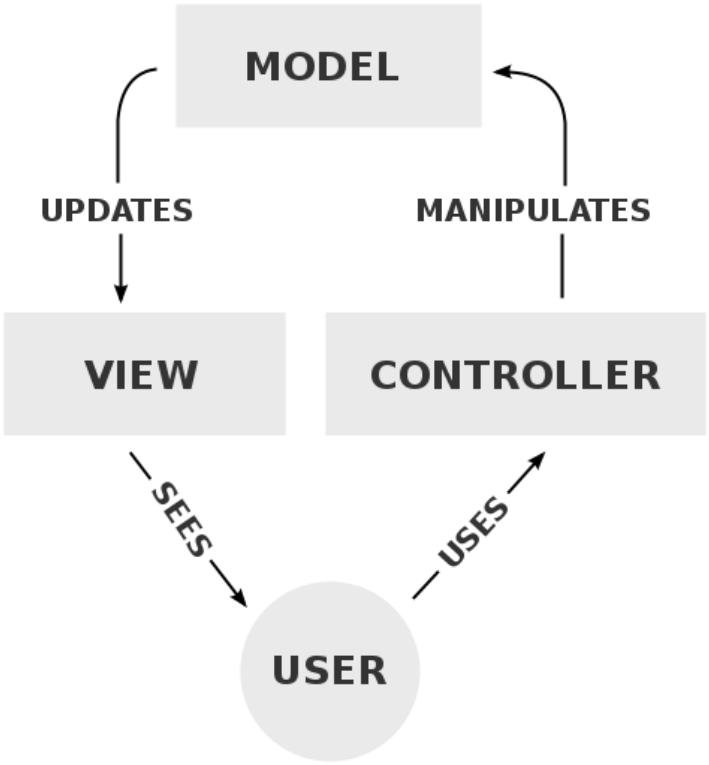
Arkitektura Model-View-Controller (MVC) është një model dizajni që përdoret zakonisht në inxhinierinë e softuerit. Tre komponentët logjikë të MVC dhe përgjegjësitë e tyre janë:

Model (M): Përfaqëson të dhënat dhe logjikën e biznesit. Kjo pjesë merret me ruajtjen dhe përpunimin e informacionit.

View (V): Është pjesa që shfaq informacionin dhe reagon ndaj veprimeve të përdoruesit. Kjo pjesë është përgjegjëse për paraqitjen grafike dhe interaktivitetin.

Controller (C): Është pjesa që menaxhon kërkesat e përdoruesve dhe përditëson modelin ose pamjen bazuar në ato kërkesa. Kontrolleri bashkon modelin dhe pamjen.

MVC ndihmon në ndarjen e detyrave të ndryshme në një aplikacion, duke e bërë më të lehtë zhvillimin dhe mirëmbajtjen e tij. Ky model lejon një qasje të strukturuar dhe efikase në krijimin e aplikacioneve të uebit.



7) What are the advantages of ASP.NET Core over ASP.NET? (List and compare)

1. High Performance

ASP.NET Core është i optimizuar për shpejtësi dhe përdorim të ulët të memorisë, duke e bërë efikase për trajtimin e një numri të madh të kërkesave të njëpasnjëshme në aplikacione web me performancë të lartë.

2. Cross-Platform Support

ASP.NET Core mundëson zhvillimin dhe implementimin në shumë sisteme operative (Windows, macOS, Linux), ofruar fleksibilitet dhe zgjedhje të platformës.

3. Less Code

ASP.NET Core promovon një përvojë zhvillimi të thjeshtuar me karakteristika si tag helpers dhe modelet moderne të zhvillimit, duke rezultuar në një kod me redundancë të ulët për produktivitet dhe lexueshmëri të përmiruar.

4. Cloud-Based App. Dev. Support

ASP.NET Core integrohet pa problem me shërbimet në cloud, duke mbështetur zhvillimin cloud-native me karakteristika si furnizuesit e konfiguracionit dhe implementimi i lehtë në platformat e ndryshme në cloud.

5. Containerization Support

Mbështetja e fortë e ASP.NET Core për containerization, veçanërisht me Docker, lejon zhvilluesit të përpilojnë aplikacionet dhe varësitë e tyre në kontejnerë të transportueshëm për konsistencë dhe implementim të thjeshtuar.

6. Built-in Dependency Injection

ASP.NET Core përfshin një sistem të integruar të injeksionit të varësive, që ndihmon në një kodbazë modulare dhe të qëndrueshme. Ky sistem lejon menaxhimin dhe injeksionin e lehtë të varësive, duke përmirësuar fleksibilitetin dhe testueshmërinë në procesin e zhvillimit.

8) What is the difference between a layout and a template? Give an example of usage?

Layout

Definimi: Një layout i referohet strukturës së një web faqe-je duke përfshirë organizimin e elementeve të përbashkëta si header, footers, navigation bars, dhe content areas. Layout definon pamjen dhe ndjesinë e përgjithshme të saj, duke ofruar një strukturë të njejtë nëpër faqe të shumta.

Shembull: Në ASP .NET Core, layout zakonisht definohet duke përdorur extension **.cshtml** (C# HTML) i cili tregon se është një Razor file**:**

```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title - My Website</title>
</head>
<body>
    <header>
        <!-- Header content goes here -->
    </header>

    <main>
        @RenderBody() <!-- Content from individual pages goes here -->
    </main>

    <footer>
        <!-- Footer content goes here -->
    </footer>
</body>
</html>
```

Template

Definimi: Një template mund të jetë një strukturë HTML që ri-përdoret si një pikë nisje për krijimin e faqeve të ngjashme.

Shembull: Një template i postimit në blog mund të përfshijë seksione për titullin e postimit, autorin, datën dhe përmbajtjen. Çdo herë që krijohet një postim i ri në blog, ky template mund të ri-përdoret me përmbajtje të ndryshme.

```
<article>
    <h2>{{ PostTitle }}</h2>
    <p>By: {{ Author }} | Date: {{ Date }}</p>
    <div>
        {{ Content }}
    </div>
</article>
```

9) Explain the concept of Scaffolding in ASP.NET MVC? (With Example)

Scaffolding në ASP.NET MVC është një framework për gjenerimin e kodit që automatizon krijimin e elementeve të zakonshme të aplikacionit në ueb. Ai gjeneron kod për controllers, views dhe modelet e të dhënave bazuar në një model të dhënash ose skemë të dhënash. Qëllimi është të krijohet shpejt një pikënisje funksionale për operacionet CRUD (Create, Read, Update, Delete) pa shkruar kod manualisht.

Shembull: Nëse kemi modelin **Product**:

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
}
```

Tani për të krijuar CRUD operations për këtë model, në vend që të krijojmë controller dhe views manualisht, mund të përdorim scaffolding për t'a gjeneruar kodin bazik.

Krijo **Controller**:

1. Right-click on the Controllers folder in Visual Studio.
2. Choose "Add" -> "Controller."
3. In the dialog, select "MVC 5 Controller with views, using Entity Framework" or a similar option.
4. Choose the Product model class and a data context.
5. Click "Add."

Kjo do t'a gjenerojë controller me operacionet CRUD.

Krijo **Views**:

1. In the generated controller, you'll see methods like Index, Create, Edit, and Delete.
2. Right-click on the action methods in the controller.
3. Choose "Add View."
4. A dialog will appear; you can customize the view templates based on your needs.
5. Click "Add."

Kjo do t'i shtojë file-at me extension **.cshtml** dhe view për secilin operacion CRUD.

Përdorimi i scaffolding do të kursej shumë kohë, veçanërisht për detyrat të cilat përsëriten dhe ofron një pikënisje të shpejtë për ndërtimin e aplikacionit tuaj.

10) Which two main commands are used to add a new migration, an update the database?

Dy komandat që përdoren për të shtuar një migrim të ri dhe për t'a përditësuar databazën në ASP .NET Core janë:

Add-Migration: Kjo komandë përdoret për të krijuar një file të ri migrimi i cili përmban ndryshimet në database schema.

Update-Database: Kjo komandë përdoret për të aplikuar migrimet në pritje në databazë.

11) What is a tag helper in ASP.NET Core?(with example)

Tag Helpers në ASP.NET Core janë një mënyrë për të krijuar elementë ose atribute të personalizuara HTML që ofrojnë funksionalitet shtesë për një Razor view. Ato mund të përdoren për të përmbledhur kodin kompleks HTML dhe C# në komponentë të ripërdorshëm dhe të lehtë për t'u përdorur, të cilët mund të ndahen nëpër pamje të shumta.

Shembulli:

```
@model Person

<!DOCTYPE html>
<html>
<head>
    <title>Tag Helper Example</title>
</head>
<body>
    <h1>Person Details</h1>

    <div>
        <label asp-for="Name"></label>
        <input asp-for="Name" />
    </div>

    <div>
        <label asp-for="Age"></label>
        <input asp-for="Age" />
    </div>
</body>
</html>
```

Në këtë Razor file atributi **asp-for** në **<label>** dhe **<input>** është një tag helper dhe automatikisht gjeneron HTML e caktuar duke u bazuar në vetitë e specifikuara të modelit.

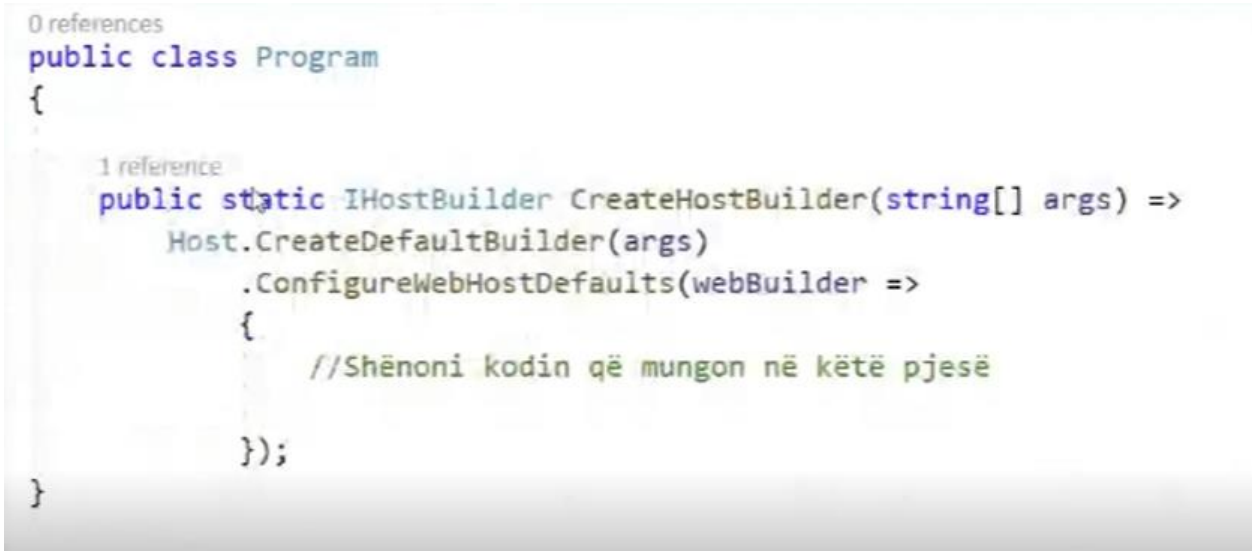
Kur bëhet render ky view me modelin 'Person', HTML i gjeneruar do të jetë kështu:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Tag Helper Example</title>
</head>
<body>
    <h1>Person Details</h1>

    <div>
        <label for="Name">Name</label>
        <input type="text" id="Name" name="Name" value="John Doe" />
    </div>

    <div>
        <label for="Age">Age</label>
        <input type="text" id="Age" name="Age" value="25" />
    </div>
</body>
</html>
```

12) What is missing code in the class Program (see figure below) (Version .3,1)?




```
webBuilder.UseStartup<Startup>();
```

13) Write an example of the default route pattern in MVC?

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
app.MapRazorPages();
```

14) Define a class "Drejtimi" and DbSet with the following properties and requirements:

- ID (Integer)
- Name (String)
- Address (String)
- Web (Double)
- Email (String)
- Status (Boolean)
- CreationDate (DateTime)

Only FullName is required and must be at least 7 characters.

```
public class Drejtimi
{
   public int ID { get; set; }
   [Required]
   [MinLength(7)]
   public string FullName { get; set; }
   public string Address { get; set; }
   public double Web { get; set; }
   public string Email { get; set; }
   public bool Status { get; set; }
   public DateTime CreationDate { get; set; }
}

public DbSet<Drejtimi> Drejtimet { get; set; }
```

15) Write an API / GET method of the controller "select by ID" for the "Afati" model, (e.g. GET:api/Afati/2)

```
// GET BY ID
[HttpGet("api/Afati/{id}")]
public ActionResult<Afati> GetById(int id)
{
   var afati = _context.Afati.Find(id);
```

```
    if (afati == null)
    {
        return NotFound();
    }
    return afati;
}
// GET
[HttpGet("api/Afati")]
public ActionResult<IEnumerable<Afati>> GetAll()
{
    return _context.Afati.ToList();
}
// CREATE
[HttpPost("api/Afati")]
public ActionResult<Afati> Create(Afati afati)
{
    if (afati.Name.Length < 4)
    {
        return BadRequest("Name must be at least 4 characters.");
    }
    _context.Afati.Add(afati);
    _context.SaveChanges();
    return CreatedAtAction(nameof(GetById), new { id = afati.ID }, afati);
}
// UPDATE
[HttpPut("api/Afati/{id}")]
public IActionResult Update(int id, Afati afati)
{
    if (id != afati.ID)
```

```
    {
        return BadRequest();
    }
    if (afati.Name.Length < 4)
    {
        return BadRequest("Name must be at least 4 characters.");
    }
    _context.Entry(afati).State = EntityState.Modified;
    _context.SaveChanges();
    return NoContent();
}
// DELETE
[HttpDelete]
public IActionResult Delete(int id)
{
    var afati = _context.Afati.FirstOrDefault(a => a.ID == id);
    if (afati == null)
    {
        return NotFound();
    }
    _context.Afati.Remove(afati);
 _context.SaveChanges();
    return NoContent();
}
```