**Plan: ASP.NET Core Introduction**

**Objective**

In this lecture, the students should be introduced to ASP.NET Core, the MVC Pattern, ASP.NET Core MVC, Creating an ASP.NET Core MVC App, Controllers and Actions, Overview of Routing and Static Files, MVC Views and Razor, Overview of Dependency Injection, Overview of Model Binding and Model Validation.

**Motivation**

ASP.NET Core is a cross-platform, open-source framework for building modern, cloud-enabled, Internet-connected apps. With ASP.NET Core, you can: build web apps and services, Internet of Things (IoT) apps and mobile backends. The MVC pattern is another important concept from software development for structuring Web back-end apps. The ASP.NET Core MVC framework is a lightweight, open-source, highly testable presentation framework optimized for use with ASP.NET Core. ASP.NET Core MVC provides a patterns-based way to build dynamic websites that enables a clean separation of concerns.

**Content**

### ASP.NET Core (~15 min)

* **.NET Core Overview – Bird's eye view**
* **ASP.NET Core Overview**
  + **ASP.NET Core** is a **cross-platform open-source back-end development framework for C#**
    - ASP.NET Core **MVC**: build server-side Web apps
    - ASP.NET Core **Web** **API**: build Web services and REST APIs
    - ASP.NET Core **Web** **Pages**: build simple Web apps
  + Great documentation: <https://docs.microsoft.com/en-us/aspnet>
    - **ASP.NET Core** provides
      * Integration of modern client-side frameworks (Angular, React, Blazor, etc.)
      * Development workflows (MVC, WebAPI, Razor Pages, SignalR)
  + **ASP.NET Core** applications run both on **.NET Core** and **.NET Framework**
* **ASP.NET Core Main Features**
  + A **unified** **framework** for building web UI and web APIs, architected for testability
  + Ability to develop and run on **Windows**, **macOS** and **Linux**
    - Ability to host on IIS, Nginx, Apache, Docker, or self-host in your process
  + Built-in **dependency injection**
  + A lightweight, high-performance, and modular HTTP request pipeline (**middlewares**)
  + **Razor** **Pages** is a page-based programming model that makes building web UI easier
  + **Blazor** lets you use C# in the browser and share server-side and client-side app logic
  + **Razor** **markup** provides a syntax for Razor Pages, **MVC** views and Tag Helpers
  + Cloud-ready, environment-based configuration system
  + Side-by-side app versioning
  + Tooling that simplifies modern web development (Visual Studio, VS Code, CLI)

### Q&A [Sli.do] (~5 min)

### The MVC Pattern (~20 min)

* **The MVC Pattern**
  + **Model–View–Controller** (MVC) is a **software architectural pattern**
  + Originally formulated in the late 1970s by Trygve Reenskaug as part of the **Smalltalk** (object-oriented programming language)
  + **Code reusability** and **separation of concerns**
  + Originally developed for **desktop**, then adapted for **internet applications**
* **The Model-View-Controller (MVC) Pattern**
  + The **Model-View-Controller** (**MVC**) pattern
  + **Controller**
    - Handles user actions
    - Updates the model
    - Renders the view (UI)
  + **Model**
    - Holds app data
  + **View**
    - Displays the UI, based on the model data
* **Controller**
  + The **Controller** in **MVC** represents
    - **Processes user's actions** and produces a response
    - Process the requests with the help of **Views** and **Models**
    - A set of classes that handles
      * Communication from the user
      * Overall application flow
      * Application-specific logic
    - Every **Controller** has one or more "**Actions**"
* **View**
  + The **View** in **MVC** represents
    - Defines how the application's user interface (**UI**) will be displayed
    - May support **Master Views** (layouts) and **Sub-Views** (**partial views** or controls)
    - In Web apps: template to dynamically generate HTML
* **Model**
  + The **Model** in **MVC** represents
    - A **set of classes** that describes the **data** we display in the UI
    - May contain **data validation rules**
  + Two types of models
    - **View model / binding model**
      * Maps the UI of the Web page to C# class
      * Part of the **MVC** architecture
    - **Database model / domain model**
      * Maps database table to C# class (using ORM)
* **MVC Steps**
  + Incoming **Request** routed to **Controller**
  + **Controller** processes **Request** and creates a **Model** (view model)
    - Controller also selects **appropriate result** (for example: **View**)
  + **Model** is passed to the **View**
  + **The View** transforms **Model** into appropriate output format (HTML)
  + **Response** is rendered (**HTTP Response**)
* **Web MVC Frameworks**
  + **Web MVC frameworks** are used to build Web applications
    - It provides the MVC **structure** and **engine** to build Web apps
    - **Controllers** handle HTTP GET / POST requests and render a view
    - **Views** display HTML + CSS, based on the models
    - **Models** hold app data for views, prepared by controllers
  + Examples of Web MVC frameworks:
  + **ASP.NET Core MVC** (C#), **Spring MVC** (Java), **Express** (JS), **Django** (Python), **Laravel** (PHP), **Ruby on Rails** (Ruby), **Revel** (Go), …

### ASP.NET Core MVC (~15 min)

* **ASP.NET Core MVC Overview**
  + **ASP.NET Core MVC** provides features for building web APIs and web apps
    - Uses the **Model-View-Controller (MVC)** design pattern
    - Lightweight, open source, testable, good tooling
    - **Razor** markup for Razor Pages and MVC views
    - RESTful services with **ASP.NET Core Web API**
      * Built-in support for multiple data formats, content negotiation and CORS
    - Achieve high-quality architecture design, optimizing developer work
      * **Convention over Configuration**
    - **Model binding** automatically maps data from HTTP requests
    - **Model validation** with client-side and server-side validation
    - Often combined with **Entity** **Framework** for **ORM**
* **ASP.NET Core MVC** provides features for building web APIs and web apps
  + Uses the **Model-View-Controller (MVC)** design pattern
  + Lightweight, open source, testable, good tooling
  + **Razor** markup for Razor Pages and MVC views
  + RESTful services with **ASP.NET Core Web API**
  + Built-in support for multiple data formats, content negotiation and CORS
  + Achieve high-quality architecture design, optimizing developer work
  + **Convention** **over** **Configuration**
  + **Model binding** automatically maps data from HTTP requests
  + **Model** **validation** with client-side and server-side validation
  + Often combined with **Entity Framework** for **ORM**
* **ASP.NET Core MVC Features**
  + **Routing** for mapping requests
  + **Dependency injection** for injecting components at runtime
  + Strongly-typed views with the **Razor view engine**
  + **Model binding** automatically maps data from HTTP requests
  + **Model** **validation** with client-side and server-side validation
  + **Tag helpers** enable server-side code in HTML elements
  + Filters, Areas, Middlewares
  + Built-in security features
  + **Identity** with users and roles
  + And many more…

### Q&A [Sli.do] (~5 min)

### BREAK: 15 min

### Creating an ASP.NET Core MVC App (~20 min)

* Explain which workload must be installed in VS2022
* Show how to create a new project and which template should be chosen upon creating the app
* Brief overview of what's inside a Web app

### Controllers and Actions (~25 min)

* Controllers
  + All controllers should be in the "**Controllers**" folder
  + Controller naming standard should be **{name}Controller**
  + Every controller should inherit the **Controller** class
    - Access to **Request**, **Response**, **HttpContext**, **RouteData**, **TempData**, etc.
  + Routes select Controllers in every request ers
* **Actions**
  + **Actions** are the ultimate **Request** destination
    - Public controller methods
    - Non-static
    - No return value restrictions
  + Actions typically return an **IActionResult**
* **Action Results**
  + **Action result** == controller's response to a browser request
    - Represent various **HTTP status codes**
  + Inherit from the base **ActionResult** class
* **Action Selectors**
  + **ActionName**(string name)
  + **AcceptVerbs**
    - **HttpPost**
    - **HttpGet**
    - **HttpDelete**
    - **HttpOptions**
    - …
  + **NonAction**
  + **RequireHttps**
  + **etc.**
* **Action Parameters**
  + **ASP.NET Core** maps the **data** from the **HTTP request** to action parameters in few ways:
    - **Routing engine** can pass parameters to actions
      * **Routing pattern**: Users/{**username**}
    - URL query string can contain parameters
      * /Users/ByUsername?**username**=**NikolayIT**
    - HTTP post data can also contain parameters

### Q&A [Sli.do] (~5 min)

### BREAK: 15 min

### Views and Razor View Engine (~20 min)

* **Views**
  + **Views** render the **HTML** **code** for the invoked action
  + View naming standard is **{ActionName}.cshtml**
  + Views should be placed in folder "**/Views/{ControllerName}**"
  + A lot of **view** **engines** available
    - View engines execute code and provide HTML
    - Provide a lot of helpers to easily generate HTML
    - The most popular is **Razor** View Engine
* **Razor View Engine**
  + **Razor** is a markup syntax which helps us write **HTML** and **server-side code** using **C#**
  + **Razor View Engine**: use **Razor** with **MVC** to produce HTML
    - **Code blocks** start with a **@** character and don't require explicit closing
* **Razor View Engine: Example**
* **Passing Data to a View – Weakly Typed**
  + With **ViewBag** (**dynamic** type):
    - Action: **ViewBag**.**Message** = "**Hello World!**";
    - View: **@ViewBag**.**Message**
  + With **ViewData** (dictionary)
    - Action: **ViewData**["**message**"] = "**Hello World!**";
    - View: **@ViewData**["**message**"]
* **ViewBag – Example**
* **Passing Data to a View – Strongly Typed – Example**

### Q&A [Sli.do] (~5 min)

### ASP.NET Core MVC Routing (~10 min)

* **ASP.NET Core MVC Routing**
  + **ASP.NET Core MVC** uses a **middleware** for **Routing** on client requests.
    - **Routes** describe how request URL paths should be mapped to **Controller** **Actions**.
    - There are 2 types of Action routing
      * Conventional
      * Attribute
  + Called **Conventional** because it establishes a **convention** for URL paths.
* **Conventional Routing (Used by Default)**
  + Called **Conventional** because it establishes a **convention** for URL paths
  + Will match a route like "**/Cats/Show/1**"
  + Will extract the route values

### Static files (~10 min)

* **Static Files**
  + **Static** **files** are a necessity for a web application to work
  + Files such as HTML, CSS, JS and different Assets can be served directly to Clients with **ASP.NET Core**

### Q&A [Sli.do] (~5 min)

### BREAK: 10 min

### Dependency Injection (~10 min)

* **What is Dependency Injection?**
  + **Dependency injection** injects objects at **runtime**
    - **Register** some service class in the **Startup** class
    - Later, **inject** the registered class in your controllers

### Model Binding (~10 min)

* **Model Binding (1)**
  + **Model binding** in ASP.NET Core MVC maps data from **HTTP** **requests** to **action** **method** **parameters**
    - The parameters may be primitive types or complex types
    - Implemented abstractly, paving the way for reusability in different apps
  + The framework binds request data to action parameters by **name**
    - The value of each parameter will be searched, using the **parameter** **name**
    - Classes are mapped using the names of the **public** **settable** **properties**
  + **Model binding** can look through **several data sources** perRequest
    - **Form values** – POST Request parameters
    - **Route values** – The set of Route values provided by the Routing
    - **Query strings** – The query string parameters in the URL
    - Even in headers, cookies, session, etc. in custom model binders
    - Data from these sources are stored as **name-value** pairs
  + The framework checks each of the data sources for a parameter value
    - If there is no parameter in the data source, the next in order is checked
    - The data sources are checked in the **order** specified above
  + If binding **fails**, the framework does **not** throw an **error**
    - Every action, accepting **user input**, should check if binding was successful
    - This is done through the **ModelState.IsValid** property
  + Each entry in the **controller**'s **ModelState** property is a **ModelStateEntry**
    - Each **ModelStateEntry** contains an **Errors** property
    - It's rarely necessary to query this collection, though
  + Default **Model binding** works great for most development scenarios
    - It is also extensible, and you can customize the built-in behavior
  + You can easily **iterate** over the errors in the **ModelState**
* **Incoming Request to MVC**

### Q&A [Sli.do] (~5 min)

### Model Validation (~10 min)

* **Model Validation**
  + **Validation** is absolutely necessary before persisting data
    - There may be potential security threats
    - There may be malformed data (**type**, **size**, **data** **constraints**)
  + In **ASP.NET Core MVC**, validation happens both on **client** and **server**
  + .NET provides us an abstracted validation through **attributes**
    - Some attributes configure model validation by **constraint**
      * Similar to validation on **database fields**
    - Other apply patterns to data to enforce **business rules**
      * **Credit Cards**, **Phone Numbers**, **Email Addresses** etc.
  + Validation attributes make enforcing these requirements simple
    - They are specified at the **property** or **parameter** level
  + Some of the most popular built-in validation attributes

### Q&A [Sli.do] (~5 min)

**Overview**

* **ASP.NET Core** is a great platform for developing Web apps
* MVC **Controllers** and **Actions**
* MVC **Views** and **Razor**
* **Routing**
* **Static Files**
* Dependency Injection
* **Model** **Binding** and **Model** **Validation**

**Exercise**

Give tasks and examples that include creating ASP.NET Core Projects, Controllers & Actions, Routing, Static Files.

**Evaluation & Exam**

The lecture will be included in the exam.