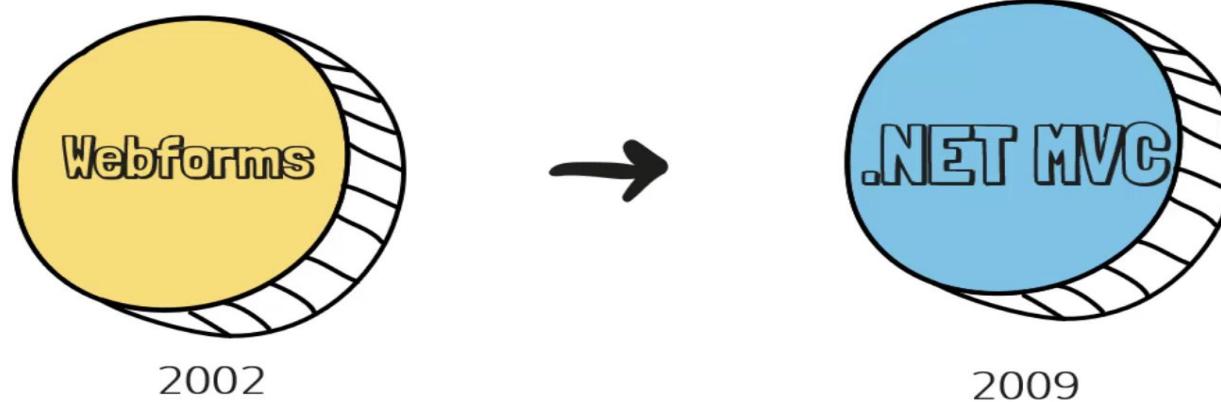


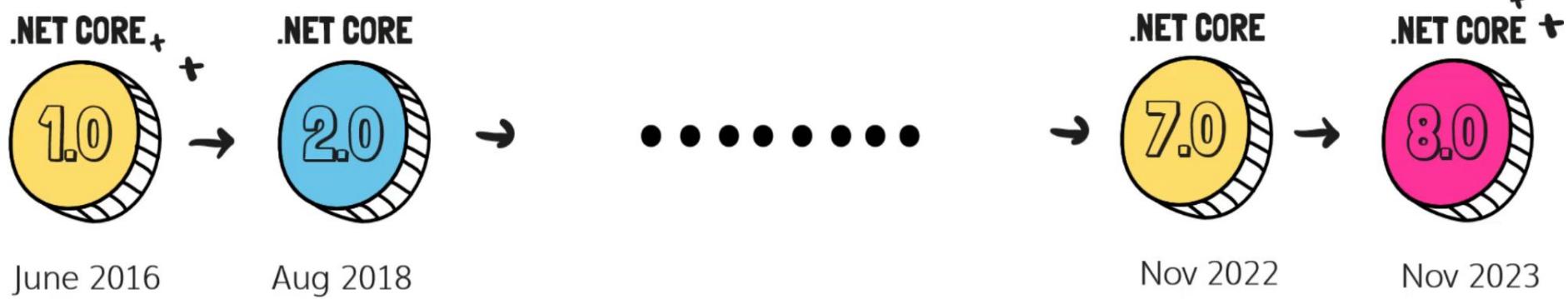


# .NET CORE ROADMAP





# .NET CORE ROADMAP



CROSS  
PLATFORM

BUILT IN  
DEPENDENCY  
INJECTION

EASY  
UPDATES

FAST AND  
OPEN  
SOURCE

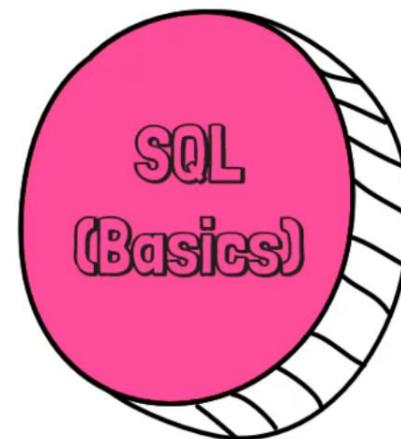
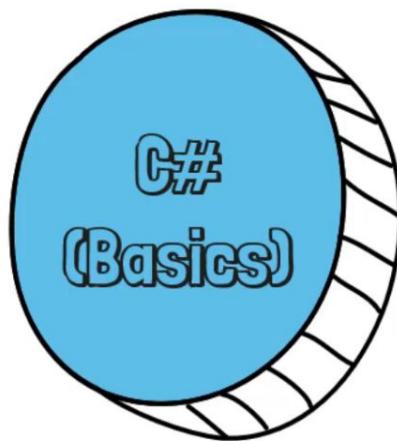
## ASP.NET CORE

CLOUD  
FRIENDLY

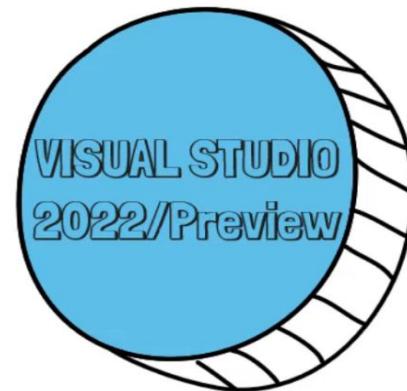
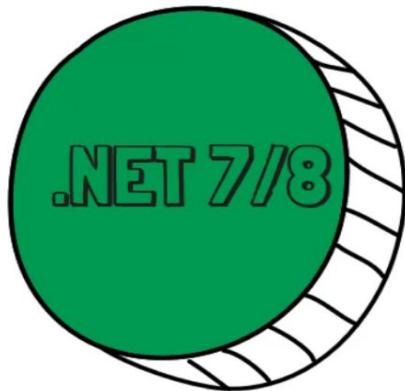
PERFORMANCE



# Prerequisites



# TOOLS NEEDED



# Create a new project

## Recent project templates

- ASP.NET Core Web App (Model-View-Controller) C#
- ASP.NET Core Web API C#
- ASP.NET Core Web App C#

mvc

Clear all

All languages All platforms All project ty...

### ASP.NET Core Web API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

C# Linux macOS Windows Cloud Service Web WebAPI

### ASP.NET Core Web App (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

C# Linux macOS Windows Cloud Service Web

### ASP.NET Core Web App (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

F# Linux macOS Windows Cloud Service Web

### ASP.NET Core Web API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

F# Linux macOS Windows Cloud Service Web WebAPI

Back

Next

launch...gs.json

Schema: <https://json.schemastore.org/launchsettings.json>

```
16     "environmentVariables": {  
17         "ASPNETCORE_ENVIRONMENT": "Development"  
18     }  
19 },  
20     "https": {  
21         "commandName": "Project",  
22         "dotnetRunMessages": true,  
23         "launchBrowser": true,  
24         "applicationUrl": "https://localhost:7001;http://localhost:5000",  
25         "environmentVariables": {  
26             "ASPNETCORE_ENVIRONMENT": "Development"  
27         }  
28 },  
29     "IIS Express": {  
30         "commandName": "IISExpress",  
31     }  
32 }
```

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) Bulky

Debug Any CPU https Bulky

BulkyW...erview

Overview Connected Services Publish

# ASP.NET Core

Learn about the .NET platform, create your first application and extend it to the cloud.

{ } Build Your App

ASP.NET Core documentation .NET application architecture

Cloud icon Connect To The Cloud

Upload icon Publish your app to Azure

Get started with ASP.NET on Azure

Keys icon Learn Your IDE

See our productivity guide Write code faster

Build

Rebuild

Clean

View

Analyze and Code Cleanup

Pack

Publish...

Configure Application Insights...

EF Core Power Tools

Overview

Scope to This

New Solution Explorer View

File Nesting

Edit Project File

- Add
- Manage NuGet Packages...
- Manage Client-Side Libraries...
- Manage User Secrets
- Remove Unused References...
- Sync Namespaces

Set as Startup Project

Debug

Git

Cut Ctrl+X

Remove Del

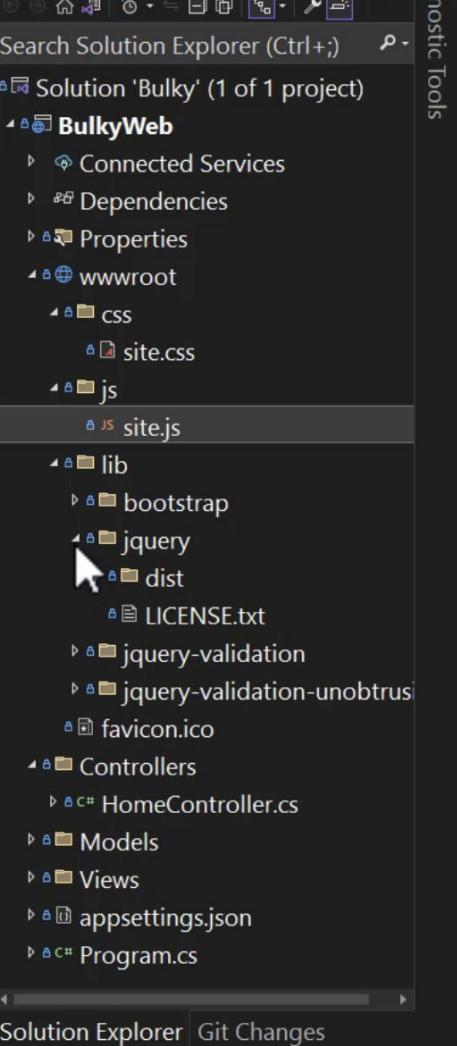
Rename F2

Unload Project

Load Direct Dependencies

Load Entire Dependency Tree

Copy Full Path



# Wwwroot and appsettings

Debu Any CPU https

Live Share

Toolbox

appset...on.json Schema: <No Schema Selected>

```
1 {  
2   "ConnectionStrings": {  
3     "DefaultConnection": "Server=(localdb)\\MSSQLLCLC  
4   }  
5 }
```

Solution Explorer

Search Solution Explorer (Ctrl+;) ▾

- Solution 'Bulky' (1 of 1 project)
  - BulkyWeb
    - Connected Services
    - Dependencies
    - Properties
      - launchSettings.json
    - wwwroot
    - Controllers
    - Models
    - Views
  - appsettings.json
    - appsettings.Development.json
    - + appsettings.Production.json
  - Program.cs

304% No issues found

Ln: 1 Ch: 1 SPC CRLF

Solution Explorer Git Changes

Program.cs

BulkyWeb

```
1 var builder = WebApplication.CreateBuilder(args)
2
3     // Add services to the container.
4     builder.Services.AddControllersWithViews();
5
6     var app = builder.Build();
7
8     // Configure the HTTP request pipeline.
9     if (!app.Environment.IsDevelopment())
10    {
11        app.UseExceptionHandler("/Home/Error");
12        // The default HSTS value is 30 days. You may want to change this for production scenarios.
13        app.UseHsts();
14    }
15
16    app.UseHttpsRedirection();
```

Package Manager Console Output

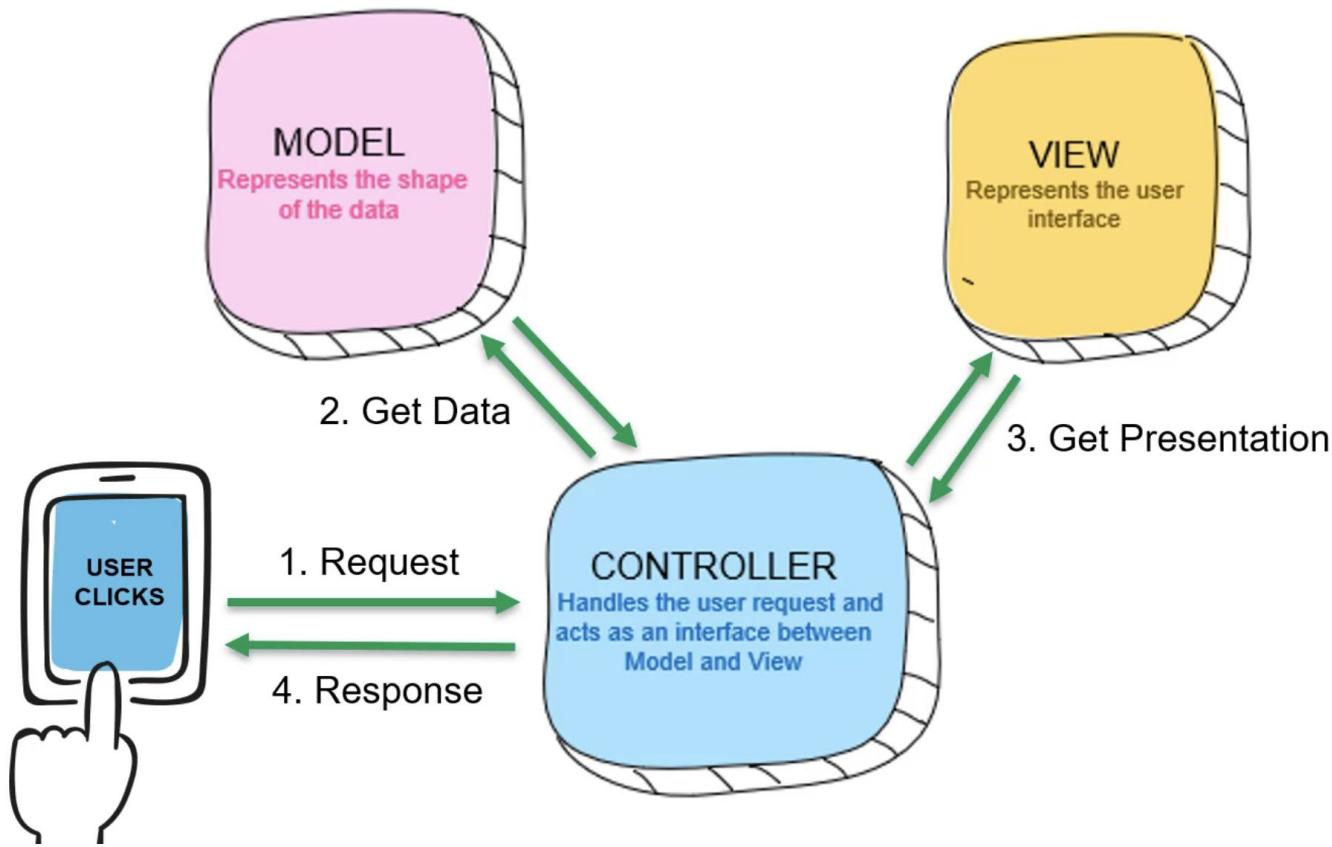
304% No issues found | Lln: 1 Ch: 1 SPC CRLF

Solution Explorer Git Changes

Solution Explorer

- Solution 'Bulky' (1 of 1 project)
  - BulkyWeb
    - Connected Services
    - Dependencies
    - Properties
      - launchSettings.json
    - wwwroot
    - Controllers
    - Models
    - Views
    - appsettings.json
  - Program.cs

# MVC ARCHITECTURE





# ROUTING IN MVC

The URL pattern for routing is considered after the domain name.

- `https://localhost:55555/Category/Index/3`
- `https://localhost:55555/{controller}/{action}/{id}`

URL	Controller	Action	Id
<code>https://localhost:55555/Category/Index</code>	Category	Index	Null
<code>https://localhost:55555/Category</code>	Category	Index	Null
<code>https://localhost:55555/Category/Edit/3</code>	Category	Edit	3
<code>https://localhost:55555/Product/Details/3</code>	Product	Details	3

Version

ASP.NET Core in .NET 9.0

ASP.NET Core documentation

- > Overview
- Get started
- > What's new
- ↳ Tutorials
- ↳ Web apps

Choose an ASP.NET Core UI

- > Razor Pages
- ↳ MVC
  - Get started
  - Add a controller
  - Add a view
  - Add a model
  - Work with a database
  - Controller actions and views
  - Add search
  - Add a new field

Learn / .NET / ASP.NET Core /



# Get started with ASP.NET Core MVC

Article • 07/30/2024 • 23 contributors

Feedback

## In this article

- Prerequisites
- Create a web app
- Visual Studio help

By [Rick Anderson](#)

This tutorial teaches ASP.NET Core MVC web development with controllers and views. If you're new to ASP.NET Core web development, consider the [Razor Pages](#) version of this tutorial, which provides an easier starting point. See [Choose an ASP.NET Core UI](#), which compares Razor Pages, MVC, and Blazor for UI development.

This is the first tutorial of a series that teaches ASP.NET Core MVC web development with controllers and views.

At the end of the series, you'll have an app that manages, validates, and displays movie data. You learn how to:

- ✓ Create a web app.
- ✓ Add and scaffold a model.
- ✓ Work with a database.
- ✓ Add search and validation.

## Additional resources

### Events

FabCon Vegas

Apr 1, 4 AM - Apr 3, 4 AM

The ultimate Power BI, Fabric, SQL, and AI community-led event. March 31 - April 2. Use code MSCUST for a \$150 discount. Prices go up Feb 11th.

[Register today](#)

### Training

Module

[Build your first ASP.NET Core web app - Training](#)

Learn how to build your first web app using ASP.NET Core.

Certification

[Microsoft Certified: Azure Developer Associate - Certifications](#)

Build end-to-end solutions in Microsoft Azure to create Azure Functions, implement and manage web apps, develop solutions utilizing Azure storage, and more.

### Documentation

[Part 3: add a view to an ASP.NET Core MVC app](#)

<https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-9.0&tabs=visual-studio>



# Introduction

1 minute

In this module, you create your first ASP.NET Core web app with .NET and C#.

## Example scenario

You're just getting started with ASP.NET Core. You want to understand how to quickly build your first web app and get familiar with the structure of a basic project. You want to understand how to run and serve a minimal web app on your local machine to view it in a browser.

## What will we be doing?

In this module, you:

- Review ASP.NET Core default project templates available in the .NET SDK.
- Create an ASP.NET Core web app project from a template.
- Examine the structure of the created project.
- Run your web app locally and view it in a browser.
- Review how the web app is served.
- Make code changes during local development.

<https://learn.microsoft.com/en-us/training/modules/build-your-first-aspnet-core-web-app/1-introduction>

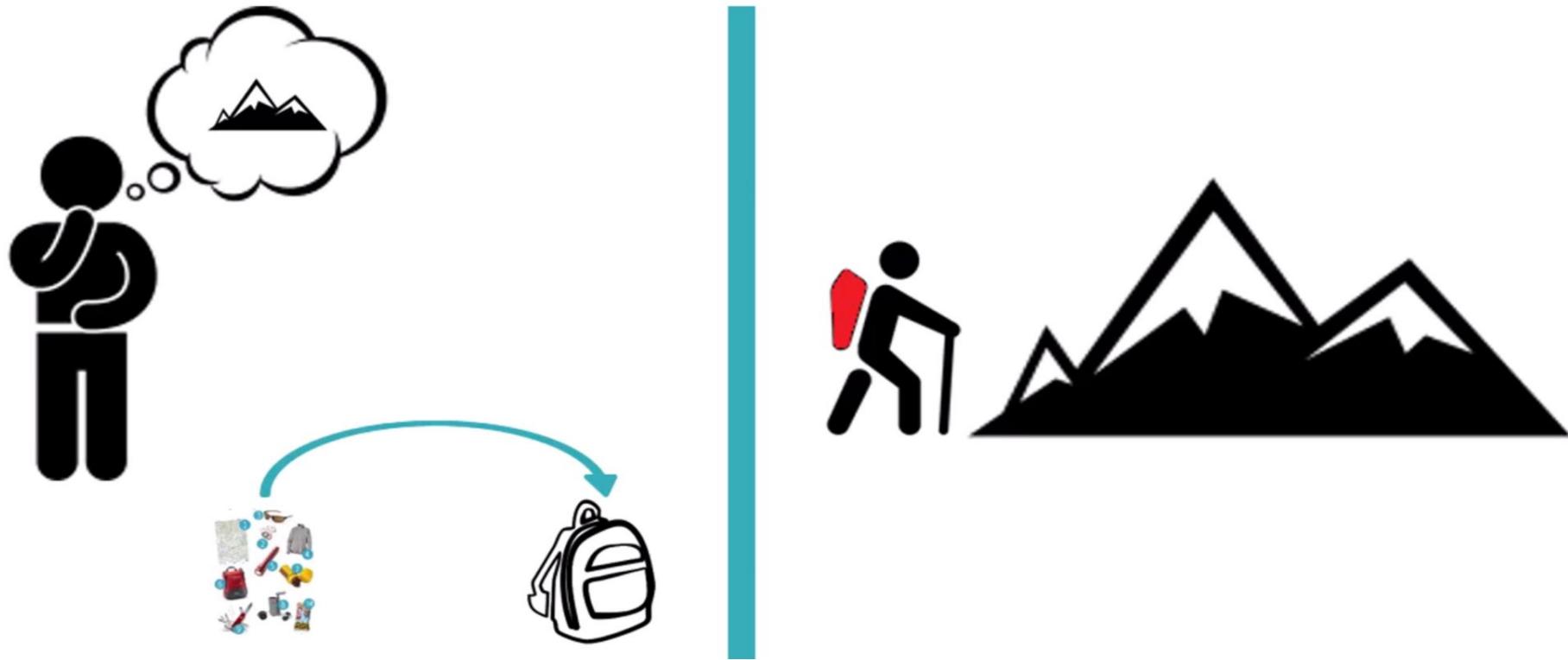
XXX



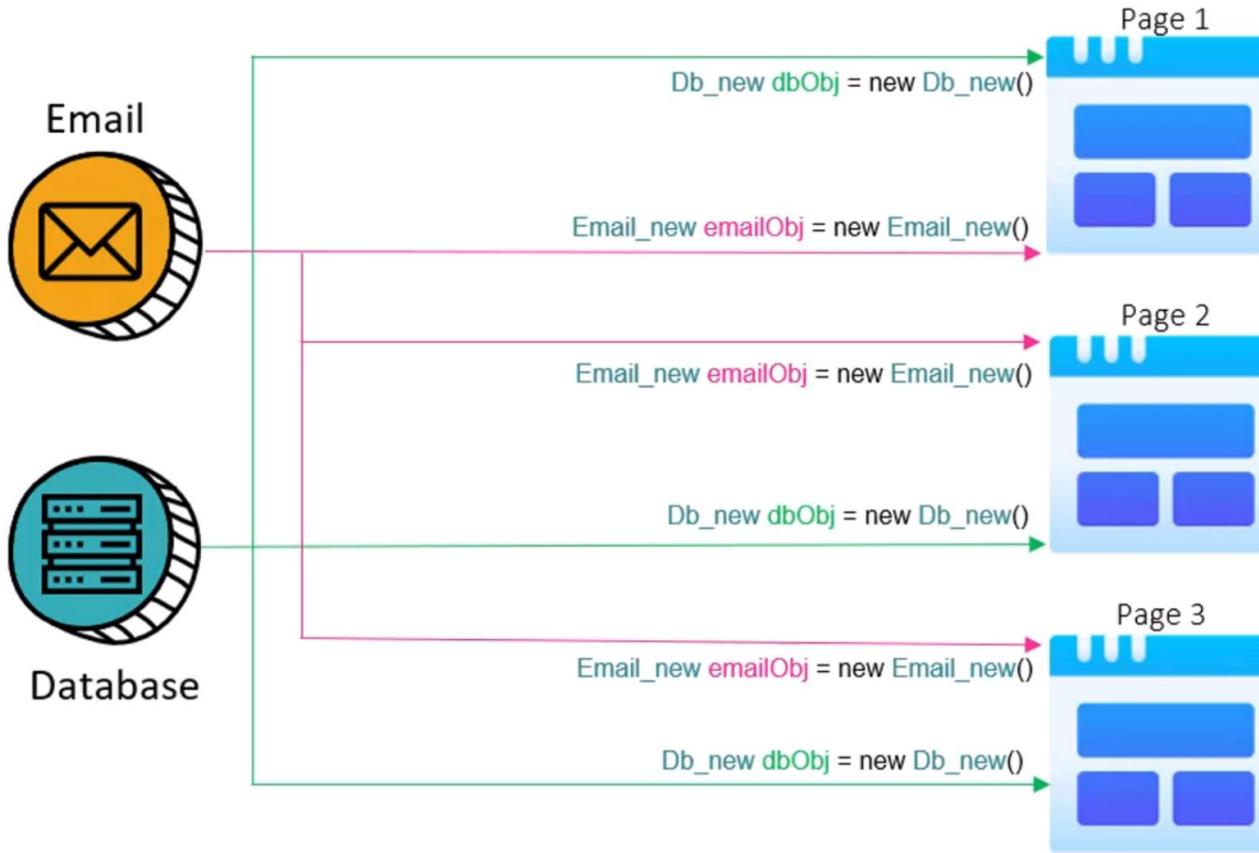
# DEPENDENCY INJECTION



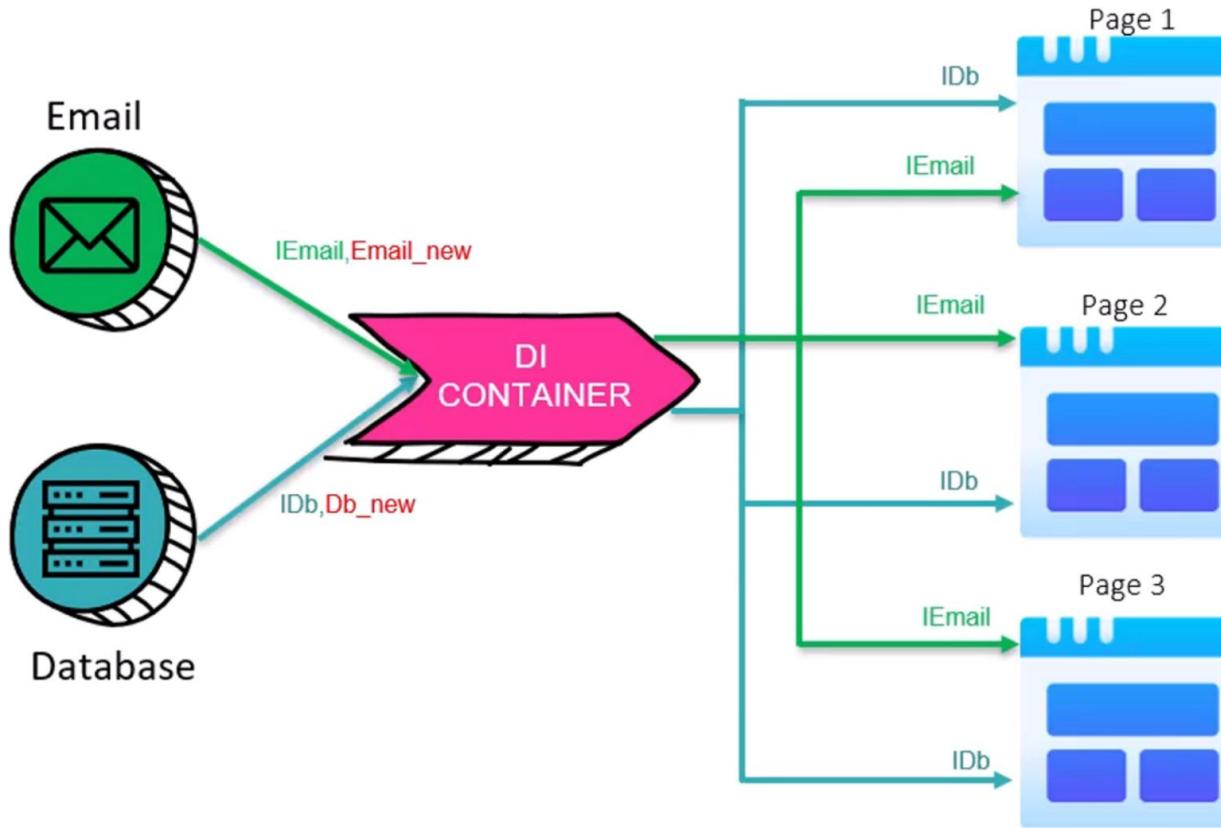
# Dependency Injection Example



# WITHOUT DEPENDENCY INJECTION



# WITH DEPENDENCY INJECTION



```
- namespace BulkyWeb.Models
{
    public class Category
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int DisplayOrder { get; set; }
    }
}
```

## Create Category Model

```
using System.ComponentModel.DataAnnotations;
```

```
namespace BulkyWeb.Models  
{  
    public class Category  
    {  
        [Key]  
        public int Id { get; set; }  
        [Required]  
        public string Name { get; set; }  
        public int DisplayOrder { get; set; }  
    }  
}
```

## Data Annotations

appsettings.json × Category.cs

Schema: <https://json.schemastore.org/appsettings.json>

```
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8    "AllowedHosts": "*",
9    "ConnectionStrings": {
10      "DefaultConnection": "Server: . ;Database=Bulky;Trusted_Connection=True;TrustServerCertifi
11    }
12  }
13 }
```

# Connection String

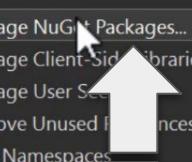
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) Bulky

Debu Any CPU https Category.cs

Schema: <https://json.schemastore.org/appsettings.json>

```
1  {
2      "Logging": {
3          "LogLevel": {
4              "Default": "Information",
5              "Microsoft.AspNetCore": "Warning"
6          }
7      },
8      "AllowedHosts": "*",
9      "ConnectionStrings": {
10         "DefaultConnection": "Server:.;Database=Bulky;Trusted_Connection=True;Tru"
11     }
12 }
13 }
```

Build Rebuild Clean View Analyze and Code Cleanup Pack Publish... Configure Application Insights... EF Core Power Tools Overview Scope to This New Solution Explorer View File Nesting Edit Project File Add Manage NuGet Packages... Manage Client-Side Libraries... Manage User Secrets... Remove Unused References... Sync Namespaces Set as Startup Project Debug Git Cut Ctrl+X Remove Del Rename F2 Unload Project Load Direct Dependencies



# Nuget packages for Entity Framework Core

NuGet....lkyWeb

Browse    Installed    Updates

Search (Ctrl+L)     Include prerelease

NuGet Package Manager: BulkyWeb

Package source: nuget.org

**Microsoft.EntityFrameworkCore** by Microsoft

Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries,...

**Microsoft.EntityFrameworkCore.SqlServer** by Microsoft

Microsoft SQL Server database provider for Entity Framework Core.

**Microsoft.EntityFrameworkCore.Tools** by Microsoft

Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

8.0.0-preview.1.23111.4    8.0.0-preview.1.23111.4    8.0.0-preview.1.23111.4

.NET Microsoft.EntityFrameworkCore. nuget.org

Installed: 8.0.0-preview.1.23111.4    Uninstall

Version: 8.0.0-preview.1.23111.4    Update

Options

Description

Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Enables these commonly used commands:

Add-Migration  
Bundle-Migration

# Install EntityFrameworkCore packages

BulkyWeb

```
</PropertyGroup>

<ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="8.0.0-preview.1.23111" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="8.0.0-preview.1.23111" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.0-preview.1.23111" />
        <PrivateAssets>all</PrivateAssets>
        <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
</ItemGroup>

</Project>
```

# Check Project File

Applica...text.cs

BulkyWeb

```
{\`1  using Microsoft.EntityFrameworkCore;
2
3  namespace BulkyWeb.Data
4
5  public class ApplicationDbContext : DbContext
6  {
7      public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
8      {
9      }
10 }
11 }
```

## Setup Data/ApplicationDbContext

```
// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<ApplicationDbContext>(options=>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

```

## Add Services to the Container

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) Bulky

Feature 'Diagnostic analyzer runner' is currently unavailable due to an internal error. Show Stack Trace

appsettings.json Program.cs Applica...text.cs

Schema: <https://json.schemastore.org/appsettings.json>

```
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8    "AllowedHosts": "*",
9  }
```

Run  
**“update-database”**  
command

```
PM> update-database
Build started...
Build succeeded.
```

Get Tools and Features...  
Manage Preview Features  
Connect to Database...  
Connect to Server...  
SQL Server  
Data Lake  
Code Snippets Manager... Ctrl+K, Ctrl+B  
Choose Toolbox Items...  
NuGet Package Manager  
Create GUID  
External Command 2  
External Tools...  
Theme  
Command Line  
Import and Export Settings...

Package Manager Ctrl+P, Ctrl+M  
Manage NuGet Packages for Solution...  
Package Manager Settings

# Create Database

Category.cs appset...gs.json Program.cs Application.cs BulkyWeb.BulkyWeb.Data.ApplicationDbContext Categories

```
{\`1 1 using BulkyWeb.Models;
2 2     using Microsoft.EntityFrameworkCore;
3 3
4 4     namespace BulkyWeb.Data
5 5     {
6 6         public class ApplicationDbContext : DbContext
7 7         {
8 8             public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
9 9             {
10 10
11 11         }
12 12
13 13     public DbSet<Category> Categories { get; set; } \`13
14 14     }
15 15 }}
```

Add DbSet with Model Type

Package Manager Console  
Package source: All Default project: BulkyWeb

```
No migrations were applied. The database is already up to date.  
No migrations were applied. The database is already up to date.  
Done.  
PM> add-migration AddCategoryTableToDb
```

Run add-migration command

Confirm migration file and run “update-database” command

# Create Category Table

The screenshot illustrates the process of adding a new view to an ASP.NET Core application. On the left, the code editor shows the `CategoryController` class:

```
namespace BulkyWeb.Controllers
{
    public class CategoryController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Below the code editor is the "Add New Item - BulkyWeb" dialog, which lists various item types for C#:

- Class
- Interface
- Razor Component
- MVC Controller - Empty
- MVC Controller with read/write...
- API Controller - Empty
- API Controller with read/write ac...
- Razor Page - Empty
- Razor View - Empty
- EF Core Database First Wizard
- Razor Layout
- Assembly Information File
- Code File

The "Name:" field contains `Index.cshtml`. The "Add" button is highlighted with a cursor.

On the right, the browser window displays the application's home page with the title "Category List". The solution explorer on the far right shows the project structure:

- Solution 'Bulky' (1 of 1 project)
  - BulkyWeb
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
      - CategoryController.cs
      - HomeController.cs
    - Data
    - Migrations
    - Models
    - Views
      - Category
      - Home
        - Index.cshtml
        - Privacy.cshtml
      - Shared
        - \_ViewImports.cshtml
        - \_ViewStart.cshtml
- appsettings.json
- Program.cs

# Add Category Controller/View

```
Layout.cshtml • Index.cshtml CategoryController.cs
13 <navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3>
14 <div class="container-fluid">
15     <a href="#" class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">BulkyWeb</a>
16     <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse"
17             aria-expanded="false" aria-label="Toggle navigation">
18         <span class="navbar-toggler-icon"></span>
19     </button>
20     <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
21         <ul class="navbar-nav flex-grow-1">
22             <li class="nav-item">
23                 <a class="nav-link text-dark" asp-controller="Home" asp-action="Index">Home</a>
24             </li>
25             <li class="nav-item">
26                 <a class="nav-link text-dark" asp-controller="Category" asp-action="Index">Category</a>
27             </li>
28             <li class="nav-item">
29                 <a class="nav-link text-dark" asp-controller="Home" asp-action="Privacy">Privacy</a>
30             </li>
31         </ul>
32     </div>
```

# Add Category Link in Header

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
    {
    }

    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>().HasData(
            new Category { Id = 1, Name = "Action", DisplayOrder = 1 },
            new Category { Id = 2, Name = "SciFi", DisplayOrder = 2 },
            new Category { Id = 3, Name = "History", DisplayOrder = 3 });
    }
}
```

```
PM> add-migration SeedCategoryTable
Build started...
|
```

# Seed Category Data

```
PM> update-database
Build started...
|
```

```
namespace BulkyWeb.Controllers
{
    public class CategoryController : Controller
    {
        private readonly ApplicationDbContext _db;
        public CategoryController(ApplicationDbContext db)
        {
            _db= db;
        }
        public IActionResult Index()
        {
            List<Category> objCategoryList = _db.Categories.ToList();
            return View();
        }
    }
}
```

## Get All Categories

```
public IActionResult Index()
{
    List<Category> objCategoryList = _db.Categories.ToList();
    return View(objCategoryList);
}
```

```
@model List<Category>

<h1>Category List</h1>
<table class="table table-bordered table-striped">
    <thead>
        <tr>
            <th>
                Category Name
            </th>
            <th>
                Display Order
            </th>
        </tr>
    </thead>
    <tbody>
        @foreach(var obj in Model.OrderBy(u=>u.DisplayOrder))
        {
            <tr>
                <td>@obj.Name</td>
                <td>
                    @obj.DisplayOrder
                </td>
            </tr>
        }
    </tbody>
</table>
```

# Display All Categories

The screenshot shows the Bootswatch website with a grid of theme cards. Each card includes a preview button, a download dropdown, and a brief description. The themes displayed are:

- Litera**: The medium is the message. Includes a color palette: Primary, Secondary, Success, Info, Warning, Danger.
- Lumen**: Light and shadow. Includes a color palette: Primary, Secondary, Success, Info, Warning, Danger.
- LUX**: A touch of class. Includes a color palette: Primary, Secondary, Success, Info.
- Materia**: Material is the metaphor. Includes a color palette: Primary, Secondary, Success, Info, Warning, Danger.
- Minty**: A fresh feel. Includes a color palette: Primary, Secondary, Success, Info, Warning, Danger.
- Morph**: A neumorphic layer. Includes a color palette: Primary, Secondary, Success, Info.

**Copy css from downloaded theme and paste in lib/bootstrap/dist/css/bootstrap.css**

**Also change bootstrap.min.css to bootstrap.css in \_Layout.cshtml**

# Bootswatch Theme

The screenshot shows a web browser displaying the [Bootswatch Lux](https://bootswatch.com/lux/) theme page. The page features a dark header with the word "LUX" and a "THEMES" dropdown menu. Below the header, there are sections for "NAVBAR" and "BUTTONS", each showing a preview of the theme's design. To the right of these sections is a large "SOURCE CODE" area containing the HTML and CSS source code for the navbar. A "COPY CODE" button is located at the top right of this section. At the bottom right, there are links to "GITHUB" and "TWITTER". The background of the page is a dark grey gradient.

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarColor01">
      <ul class="navbar-nav me-auto">
        <li class="nav-item">
          <a class="nav-link active" href="#">Home
            <span class="visually-hidden">(current)</span>
          </a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Features</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Pricing</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">About</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" data-bs-toggle="dropdown" href="#">Dropdown
            <div class="dropdown-menu">
              <a class="dropdown-item" href="#">Action</a>
              <a class="dropdown-item" href="#">Another action</a>
              <a class="dropdown-item" href="#">Something else here</a>
              <div class="dropdown-divider"></div>
              <a class="dropdown-item" href="#">Separated link</a>
            </div>
          </li>
        </ul>
      </div>
    </div>
  </div>
</nav>
```

Check Source Code for theme and Make changes

← → C icons.getbootstrap.com G 🔍 ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

Docs Examples **Icons** Themes Blog

New in v1.10.0: 140+ new icons!

# Bootstrap Icons

Free, high quality, open source icon library with over 1,800 icons. Include them anyway you like—SVGs, SVG sprite, or web fonts. Use them with or without [Bootstrap](#) in any project.

\$ npm i bootstrap-icons  



Your new development career awaits. Check out the latest listings.  
ads via Carbon

Get CDN link and put inside `_Layout.cshtml` file.

Currently v1.10.3 • [Icons](#) • [Icon Sprite](#) • [Install](#) • [Usage](#) • [Styling](#) • [Accessibility](#) • [GitHub repo](#)

# Bootstrap Icons

```
@model List<Category>



## Category List



Create New Category



| Category Name |
|---------------|
|---------------|


```

# Design Category List

# WELCOME

Learn about [building Web apps with ASP.NET Core](#).



bootstrap.css    \_Layout.cshtml    Index.cshtml    Program.cs    CategoryController.cs    •

BulkyWeb

```
{ 1  using BulkyWeb.Data;
 2  using BulkyWeb.Models;
 3  using Microsoft.AspNetCore.Mvc;
 4
 5  namespace BulkyWeb.Controllers
 6  {
 7      public class CategoryController
 8      {
 9          private readonly ApplicationDbContext _db;
10          public CategoryController()
11          {
12              _db = db;
13          }
14          public IActionResult Index()
15          {
16              List<Category> objCategory = _db.Category.ToList();
17              return View(objCategory);
18          }
19
20          public IActionResult Create()
21          {
22              return View();
23          }

```

Add View...    Go To View    Quick Actions and Refactorings...    Rename...    Remove and Sort Usings    Peek Definition    Go To Definition    Go To Base    Go To Implementation    Find All References    View Call Hierarchy    Track Value Source    Create Unit Tests    Breakpoint    Run To Cursor    Force Run To Cursor    Execute in Interactive    Snippet    Cut    Copy    Paste

# Create Category Controller + View

```
@model Category
```

```
<form method="post">
    <div class="border p-3 mt-4">
        <div class="row pb-2">
            <h2 class="text-primary">Create Category</h2>
            <hr/>
        </div>
        <div class="mb-3">
            <label>Category Name</label>
            <input type="text" class="form-control"/>
        </div>
        <div class="mb-3">
            <label>Display Order</label>
            <input type="text" class="form-control" />
        </div>
        <div class="row">
            <div class="col-6">
                <button type="submit" class="btn btn-primary" >Create</button>
            </div>
            <div class="col-6">
                <a asp-controller="Category" asp-action="Index" class="btn btn-secondary" >
                    Back to List
                </a>
            </div>
        </div>
    </div>
</form>
```

# Design Create Category View

```
<div class="mb-3 row p-1">
    <label asp-for="Name" class="p-0"></label>
    <input asp-for="Name" class="form-control"/>
</div>
<div class="mb-3 row p-1">
    <label asp-for="DisplayOrder" class="p-0"></label>
    <input asp-for="DisplayOrder" class="form-control" />
</div>
```

```
public class Category
{
    [Key]
    public int Id { get; set; }
    [Required]
    [DisplayName("Category Name")]
    public string Name { get; set; }
    [DisplayName("Display Order")]
    public int DisplayOrder { get; set; }
```

# Input Tag Helpers

Category.cs Create.cshtml bootstrap.css \_Layout.cshtml Index.cshtml Program.cs CategoryController.cs

✉ BulkyWeb

```
13     }
14     public IActionResult Index()
15     {
16         List<Category> objCategoryList = _db.Categories.ToList();
17         return View(objCategoryList);
18     }
19
20     public IActionResult Create()
21     {
22         return View();
23     }
24     [HttpPost]
25     public IActionResult Create(Category obj)
26     {
27         _db.Categories.Add(obj);
28         _db.SaveChanges();
29         return RedirectToAction("Index");
30     }
31 }
32
33 }
```

# Create Category

```
namespace BulkyWeb.Models
{
    public class Category
    {
        [Key]
        public int Id { get; set; }
        [Required]
        [MaxLength(30)]
        [DisplayName("Category Name")]
        public string Name { get; set; }
        [DisplayName("Display Order")]
        [Range(1,100,ErrorMessage ="Display Order must be between 1-100")]
        public int DisplayOrder { get; set; }
    }
}
```

```
[HttpPost]
public IActionResult Create(Category obj)
{
    if (ModelState.IsValid)
    {
        _db.Categories.Add(obj);
        _db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View();
}
```

```
<div class="mb-3 row p-1">
    <label asp-for="Name" class="p-0"></label>
    <input asp-for="Name" class="form-control"/>
    <span asp-validation-for="Name" class="text-danger"></span>
</div>
<div class="mb-3 row p-1">
    <label asp-for="DisplayOrder" class="p-0"></label>
    <input asp-for="DisplayOrder" class="form-control" />
    <span asp-validation-for="DisplayOrder" class="text-danger"></span>
</div>
```

# Server Side Validations

```
[HttpPost]
public IActionResult Create(Category obj)
{
    if (obj.Name == obj.DisplayOrder.ToString())
    {
        ModelState.AddModelError("name", "The DisplayOrder cannot exactly match the Name.");
    }
    if (ModelState.IsValid)
    {
        _db.Categories.Add(obj);
        _db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View();
}
```

# Custom Validations

```
<div class="row pb-2">
    <h2 class="text-primary">Create Category</h2>
    <hr/>
</div>
<div asp-validation-summary="All"></div>
<div class="mb-3 row p-1">
    <label asp-for="Name" class="p-0"></label>
    <input asp-for="Name" class="form-control"/>
    <span asp-validation-for="Name" class="text-danger"></span>
</div>
```

## CREATE CATEGORY

- The Category Name field is required.
- The value " is invalid.

Category Name

The Category Name field is required.

Display Order

The value " is invalid.

CREATE

BACK TO LIST

# Asp Validation Summary

```
if (obj.Name == obj.DisplayOrder.ToString())
{
    ModelState.AddModelError("name", "The DisplayOrder cannot exactly match the Name.");
}
if (obj.Name!=null && obj.Name.ToLower() == "test")
{
    ModelState.AddModelError("", "Test is an invalid value");
}
```

```
<div asp-validation-summary="ModelOnly"></div>
```

## Asp Validation Summary

Create.cshtml • bootstrap.css \_Layout.cshtml CategoryController.cs

```
<span asp-validation-for="DisplayOrder" class="text-danger"></span>
</div>

<div class="row">
    <div class="col-6 col-md-3">
        <button type="submit" class="btn btn-primary form-control" >Create</button>
    </div>
    <div class="col-6 col-md-3">
        <a asp-controller="Category" asp-action="Index" class="btn btn-secondary" >Back to List</a>
    </div>
</div>

</div>
</form>

@section Scripts{
    @{
        <partial name="_ValidationScriptsPartial" />
    }
}
```

Solution Explorer

- Data
  - ApplicationDbContext
- Migrations
  - 20230124010956\_Ad
  - 20230124013649\_See
  - ApplicationDbContext
- Models
  - Category.cs
  - ErrorViewModel.cs
- Views
  - Category
    - Create.cshtml
    - Index.cshtml
  - Home
    - Index.cshtml
    - Privacy.cshtml
  - Shared
    - \_Layout.cshtml
    - \_ValidationScriptsP
    - Error.cshtml
    - \_ViewImports.cshtml
    - \_ViewStart.cshtml
- appsettings.json
- Program.cs

# Client Side Validation

Index.cshtml

```
Category.cs Create.cshtml bootstrap.css Layout.cshtml Categor...ller.cs
```

```
25 |         </th>
26 |         <th></th>
27 |     </tr>
28 | </thead>
29 | <tbody>
30 |     @foreach(var obj in Model.OrderBy(u=>u.DisplayOrder))
31 |     {
32 |         <tr>
33 |             <td>@obj.Name</td>
34 |             <td>
35 |                 @obj.DisplayOrder
36 |             </td>
37 |             <td>
38 |                 <div class="w-75 btn-group" role="group">
39 |                     <a asp-controller="Category" asp-action="Edit" class="btn btn-primary mx-2">
40 |                         <i class="bi bi-pencil-square"></i> Edit
41 |                     </a>
42 |                     <a asp-controller="Category" asp-action="Delete" class="btn btn-delete mx-2">
43 |                         <i class="bi bi-trash-fill"></i> Delete
44 |                     </a>
45 |                 </div>
46 |             </td>
47 |         </tr>
```

# Edit and Delete Button

```
public IActionResult Edit(int? id)
{
    if(id==null || id == 0)
    {
        return NotFound();
    }
    Category? categoryFromDb = _db.Categories.Find(id);
    //Category? categoryFromDb1 = _db.Categories.FirstOrDefault(u=>u.Id==id);
    //Category? categoryFromDb2 = _db.Categories.Where(u=>u.Id==id).FirstOrDefault();

    if (categoryFromDb == null)
    {
        return NotFound();
    }
    return View(categoryFromDb);
}
```

## Get Category Details to Edit

Edit.cshtml

```
16 <div class="mb-3 row p-1">
17     <label asp-for="DisplayOrder" class="p-0"></label>
18     <input asp-for="DisplayOrder" class="form-control" />
19     <span asp-validation-for="DisplayOrder" class="text-danger"></span>
20 </div>
21
22 <div class="row">
23     <div class="col-6 col-md-3">
24         <button type="submit" class="btn btn-primary form-control" >Update</button>
25     </div>
26     <div class="col-6 col-md-3">
27         <a asp-controller="Category" asp-action="Index" class="btn btn-secondary border form-control" href="#">Back to List</a>
28     </div>
29
30 </div>
31
32 </div>
33
34 </div>
35 </form>
36
37 @section Scripts{
38     @{
39         <partial name=" ValidationScriptsPartial" />
}
```

Copy and Paste Same Design from Create Category

```
[HttpPost]
public IActionResult Edit(Category obj)
{
    if (ModelState.IsValid)
    {
        _db.Categories.Update(obj);
        _db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View();
}
```

# Update Category

```
<form method="post">
    <input asp-for="Id" hidden />
    <div class="border p-3 mt-4">
        <div class="row pb-2">
            <h2 class="text-primary">Edit Category</h2>
            <hr/>
        </div>
        @*<div asp-validation-summary="ModelOnly" class="text-danger" role="alert"></div>
    <div class="mb-3 row p-1">
        <label asp-for="Name" class="p-0 col-form-label"></label>
        <input asp-for="Name" class="form-control" type="text" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="mb-3 row p-1">
        <label asp-for="DisplayOrder" class="p-0 col-form-label"></label>
        <input asp-for="DisplayOrder" class="form-control" type="text" />
        <span asp-validation-for="DisplayOrder" class="text-danger"></span>
    </div>

    <div class="row">
        <div class="col-6 col-md-3">
            <button type="submit" class="btn btn-primary form-control" >Update</button>
        </div>
    </div>

```

As we have “Id” field name so not required, but if we have any other name it must required, otherwise we won’t get that field in Controller when submit form.

Even though we don’t required now, but it is always good practice to have this hidden field.

# Update Category in Action

```
public IActionResult Delete(int? id)
{
    if (id == null || id == 0)
    {
        return NotFound();
    }
    Category? categoryFromDb = _db.Categories.Find(id);
    //Category? categoryFromDb1 = _db.Categories.FirstOrDefault(u=>u.Id==id);
    //Category? categoryFromDb2 = _db.Categories.Where(u=>u.Id==id).FirstOrDefault();

    if (categoryFromDb == null)
    {
        return NotFound();
    }
    return View(categoryFromDb);
}
```

```
[HttpPost, ActionName("Delete")]
public IActionResult DeletePOST(int? id)
{
    Category? obj = _db.Categories.Find(id);
    if (obj == null)
    {
        return NotFound();
    }
    _db.Categories.Remove(obj);
    _db.SaveChanges();
    return RedirectToAction("Index");
}
```

# Get and Post Action for Delete Category

```
<form method="post">
    <input asp-for="Id" hidden />
    <div class="border p-3 mt-4">
        <div class="row pb-2">
            <h2 class="text-primary">Delete Category</h2>
            <hr/>
        </div>
        <div class="mb-3 row p-1">
            <label asp-for="Name" class="p-0"></label>
            <input asp-for="Name" disabled class="form-control col"/>
        </div>
        <div class="mb-3 row p-1">
            <label asp-for="DisplayOrder" class="p-0"></label>
            <input asp-for="DisplayOrder" disabled class="form-control" />
        </div>
    </div>

    <div class="row">
        <div class="col-6 col-md-3">
            <button type="submit" class="btn btn-danger form-control" >Delete</button>
        </div>
        <div class="col-6 col-md-3">
            <a asp-controller="Category" asp-action="Index" class="btn btn-secondary border">
```

Copy and Paste Same Design from  
Edit Category.

Make all input field with disabled  
property.

# Delete Category in Action

```
[HttpPost]
public IActionResult Create(Category obj)
{
    if (obj.Name == obj.DisplayOrder.ToString())
    {
        ModelState.AddModelError("name", "The DisplayOrder cannot exactly match the Name.");
    }

    if (ModelState.IsValid)
    {
        _db.Categories.Add(obj);
        _db.SaveChanges();
        TempData["success"] = "Category created successfully";
        return RedirectToAction("Index");
    }

    return View();
}
```

```
[HttpPost]
public IActionResult Edit(Category obj)
{
    if (ModelState.IsValid)
    {
        _db.Categories.Update(obj);
        _db.SaveChanges();
        TempData["success"] = "Category updated successfully";
        return RedirectToAction("Index");
    }

    return View();
}
```

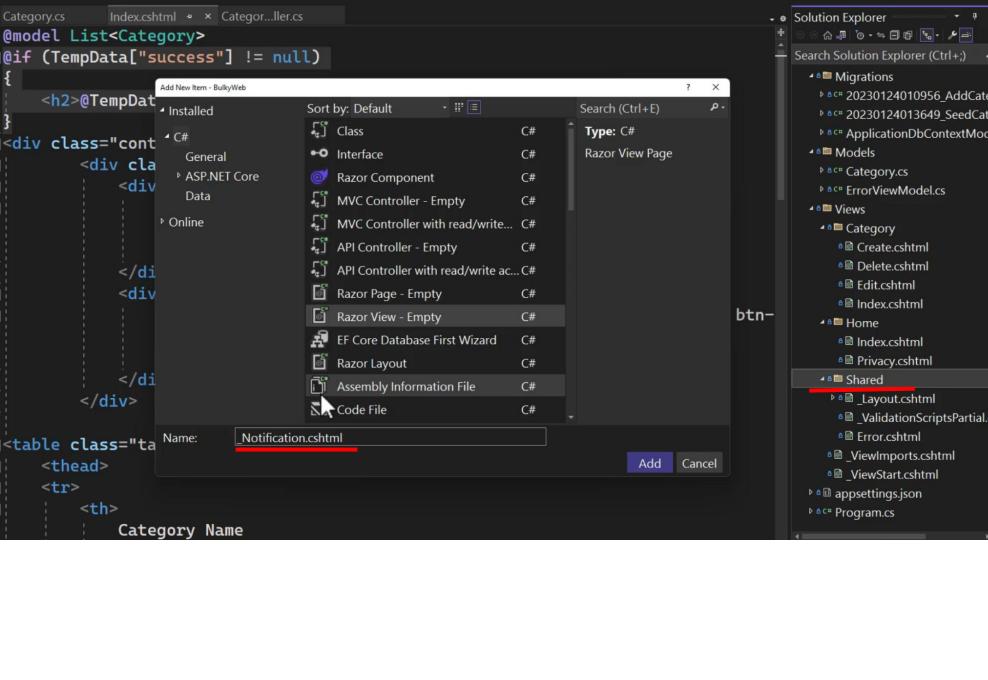
```
[HttpPost, ActionName("Delete")]
public IActionResult DeletePOST(int? id)
{
    Category? obj = _db.Categories.Find(id);
    if (obj == null)
    {
        return NotFound();
    }
    _db.Categories.Remove(obj);
    _db.SaveChanges();
    TempData["success"] = "Category deleted successfully";
    return RedirectToAction("Index");
}
```

The screenshot shows a code editor with two tabs: 'Delete.cshtml' and 'CategoryController.cs'. The 'Delete.cshtml' tab displays the following code:

```
1 @model List<Category>
2 @if (TempData["success"] != null)
3 {
4     <h2>@TempData["success"]</h2>
5 }
6 <div class="container">
7     <div class="row pt-4 pb-3">
8         <div class="col-6">
9             <h2 class="text-primary">
10                Category List
11            </h2>
12        </div>
13        <div class="col-6 text-end">
14            <a asp-controller="Category" asp-action="Create" class="btn btn-primary">Add New Category</a>
15        </div>
16    </div>
17 </div>
```

The 'CategoryController.cs' tab shows the implementation of the 'DeletePOST' action method.

# TempData



The screenshot shows the Visual Studio interface with the 'Add New Item' dialog open. The 'Type: C#' dropdown is selected, and the 'Shared' category is chosen. A new item named '\_Notification.cshtml' is being added. The 'Name:' field contains '\_Notification.cshtml'. The 'Add' button is highlighted.

```

Category.cs Index.cshtml × Category...ller.cs
@model List<Category>
@if (TempData["success"] != null)
{
    <h2>@ TempData["success"]
}
<div class="cont">
    <div cla>
        <div
            </di>
            <div
                </di>
                <di>
                    <div class="cont">
                        <div cla>
                            <div
                                </di>
                                <div
                                    </di>
                                    <di>
                                        <div class="cont">
                                            <div cla>
                                                <div
                                                    </di>
                                                    <div
                                                        </di>
                                                        <di>
                                                            <div class="cont">
                                                                <div cla>
                                                                    <div
                                                                        </di>
                                                                        <div
                                                                            </di>
                                                                            <di>
                                                                                <div class="cont">
                                                                                    <div cla>
                                                                                        <div
                                                                                            </di>
                                                                                            <div
                                                                                                </di>
                                                                                                <di>
                                                                                                    <div class="cont">
                                                                                                        <div cla>
                                                                                                            <div
                                                                                                                </di>
                                                                                                                <div
                                                                                                                    </di>
                                                                                                                    <di>
                                                                                                                        <div class="cont">
                                                                                                                            <div cla>
                                                                                                                                <div
                                                                                                                                    </di>
                                                                                                                                    <div
                                                                                                                                        </di>
                                                                                                                                        <di>
                                                                                                                                            <div class="cont">
                                                                                                                                                <div cla>
                                                                                                                                                    <div
                                                                ................................................................

```

```

Notifi...cshtml Delete.cshtml Category.cs Index.cshtml × Category...ller.cs
1  @if (TempData["success"] != null)
2  {
3      <h2>@ TempData["success"]</h2>
4  }
5  @if (TempData["error"] != null)
6  {
7      <h2>@ TempData["error"]</h2>
8  }

```

```

Notifi...cshtml Delete.cshtml Category.cs Index.cshtml × Category...ller.cs
1  @model List<Category>
2  @partial name="_Notification"
3  <div class="container">
4      <div class="row pt-4 pb-3">
5          <div class="col-6">
6              <h2 class="text-primary">
7                  Category List
8              </h2>
9          </div>

```

# Partial views

# Toastr

Simple javascript toast notifications



## / toastr

toastr is a Javascript library for Gnome / Growl type non-blocking notifications. jQuery is required. The goal is to create a simple core library that can be customized and extended.

### // Demo

Demo can be found at <http://codeseven.github.io/toastr/demo.html>

### // CDNs

Toastr is hosted at CDN JS

### /// Debug

- [//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.js](https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.js)
- [//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.css](https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.css)

### /// Minified

- [//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js](https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js)
- [//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css](https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css)

### // NuGet Gallery

<http://nuget.org/packages/toastr>



is maintained by [CodeSeven](#).

This page was generated by [GitHub Pages](#) using the Architect theme by [Jason Long](#).

# Toastr Notification

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6      <title>@ ViewData["Title"] - BulkyWeb</title>
7      <link rel="stylesheet" href "~/lib/bootstrap/dist/css/bootstrap.css" />
8      <link rel="stylesheet" href "~/css/site.css" asp-append-version="true" />
9      <link rel="stylesheet" href "~/BulkyWeb.styles.css" asp-append-version="true" />
10     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.2/font/bootstrap-icons.css" asp-append-version="true" />
11     <link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/toastr.min.css" asp-append-version="true" />
12 </head>
13 <body>
```

Also move <partial> tag from  
Category List (index.cshtml) to  
Layout.cshtml

```
_Notifi...cshtml Delete.cshtml Category.cs Index.cshtml Categor...ller.cs
@if (TempData["success"] != null)
{
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>
    <script type="text/javascript">
        toastr.success('@ TempData["success"]');
    </script>
}
@if (TempData["error"] != null)
{
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>
    <script type="text/javascript">
        toastr.error('@ TempData["error"]');
    </script>
}

-----
```

```
<div class="container">
    <main role="main" class="pb-3">
        <partial name="_Notification" />
        @RenderBody()
    </main>
</div>
```

# Toaster in Action

# N-Tier Architecture

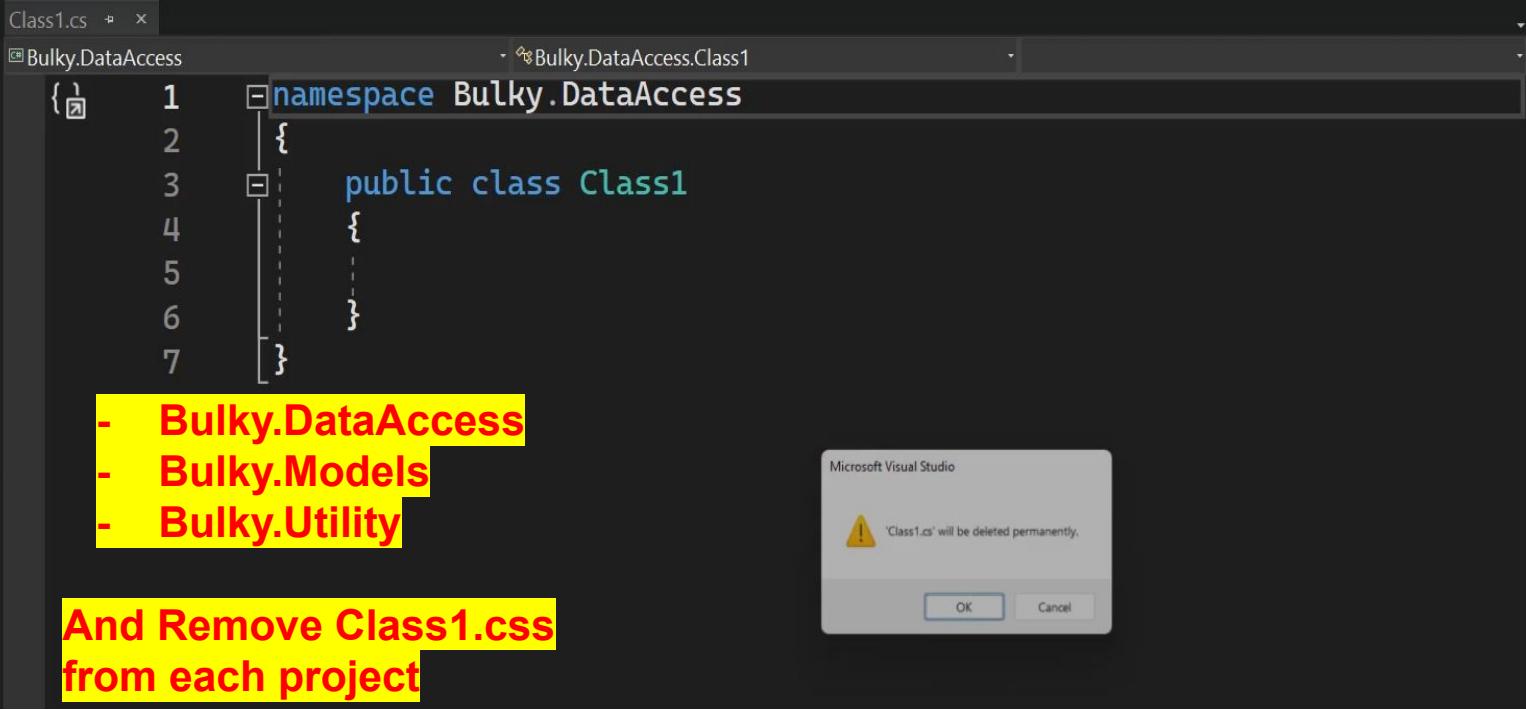
Class1.cs

Bulky.DataAccess

```
namespace Bulky.DataAccess
{
    public class Class1
    {
    }
}
```

- Bulky.DataAccess  
- Bulky.Models  
- Bulky.Utility

And Remove Class1.cs  
from each project



Microsoft Visual Studio

'Class1.cs' will be deleted permanently.

OK Cancel

# Create More Projects

Aplica..text.cs

```
1 using BulkyWeb.Models;
2 using Microsoft.EntityFrameworkCore;
3
4 namespace BulkyWeb.Data
5 {
6     public class ApplicationDbContext : DbContext
7     {
8         public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
9         {
10     }
11 }
```

**Move “Data” folder to “DataAccess” project**

```
12
13     public DbSet<Category> Categories { get; set; }
14
15     protected override void OnModelCreating(ModelBuilder modelBuilder)
16     {
17         modelBuilder.Entity<Category>().HasData(
18             new Category { Id = 1, Name = "Action", DisplayOrder = 1 },
19             new Category { Id = 2, Name = "SciFi", DisplayOrder = 2 },
20             new Category { Id = 3, Name = "History", DisplayOrder = 3 }
21         );
22 }
```

**Move “Models” to Models project**

```
23
24     public static string SD { get; set; } = Path.Combine("wwwroot", "Static");
25 }
```

**In Utility you can create Static file SD without code**

Solution Explorer

- Search Solution Explorer (Ctrl+Shift+F)
- Solution 'Bulky' (4 of 4 projects)
  - Bulky.DataAccess
    - Dependencies
    - Data
      - ApplicationContext.cs
      - Migrations
    - Bulky.Models
      - Dependencies
      - Models
    - Bulky.Utility
      - Dependencies
      - SD.cs
    - BulkyWeb
      - Connected Services
      - Dependencies
      - Properties
      - wwwroot
      - Controllers
      - Views
      - appsettings.json
      - Program.cs

# N-Tier Architecture

NuGet...olution x NuGet...Access Application.cs

Browse Installed Updates Consolidate

Search (Ctrl+L)  Include prerelease

Manage Packages for Solution

Package source: nuget.org

.NET Microsoft.EntityFrameworkCore by Microsoft 8.0.0-preview.1.23111.4  
Prerelease Entity Framework Core is a modern object-database mapper for .NET. It...

.NET Microsoft.EntityFrameworkCore.SqlServer by Microsoft 8.0.0-preview.1.23111.4  
Prerelease Microsoft SQL Server database provider for Entity Framework Core.

.NET Microsoft.EntityFrameworkCore.Tools by Microsoft 8.0.0-preview.1.23111.4  
Prerelease Entity Framework Core Tools for the NuGet Package Manager Console in...

.NET Microsoft.VisualStudio.Web.CodeGeneration.Design by M 8.0.0-preview.1.23117.2  
Prerelease Code Generation tool for ASP.NET Core. Contains the dotnet-aspnet-co...

.NET Microsoft.EntityFrameworkCore nuget.org  
Versions - 1

Project	Version
Bulky.DataAccess	<input checked="" type="checkbox"/>
Bulky.Models	<input type="checkbox"/>
Bulky.Utility	<input type="checkbox"/>

Installed: 8.0.0-preview.1.23111.4 Uninstall

Version: Latest prerelease 8.0.0-preview Install

Options

Description

Entity Framework Core is a modern object-database

Solution Explorer

Search Solution Explorer (Ctrl+)

Solution 'Bulky' (4 of 4 projects)

- Bulky.DataAccess
  - Dependencies
  - Data
    - ApplicationDbContext.cs
    - Migrations
  - Models
  - Utility
- BulkyWeb
  - Connected Services
  - Dependencies
  - Properties
  - wwwroot
  - Controllers
  - Views
  - appsettings.json
  - Program.cs

# Manage Nuget package for solution

```
ica..text.cs • Solution Explorer
ky.DataAccess          Bulky.DataAccess.Data.ApplicationDbContext      Categories
1  using Bulky.Models;
2  using Microsoft.EntityFrameworkCore;
3
4  namespace Bulky.DataAccess.Data
5  {
6      public class ApplicationDbContext : DbContext
7      {
8          public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : b:
9          {
10         }
11     }
12
13     public DbSet<Category> Categories { get; set; }
14
15     protected override void OnModelCreating(ModelBuilder modelBuilder)
16     {
17         modelBuilder.Entity<Category>().HasData(
18             new Category { Id = 1, Name = "Action", DisplayOrder = 1 },
19             new Category { Id = 2, Name = "SciFi", DisplayOrder = 2 },
20             new Category { Id = 3, Name = "History", DisplayOrder = 3 }
21         );
22     }
23 }
24
25
```

The screenshot shows a Visual Studio code editor with the file `Bulky.DataAccess.Data.ApplicationDbContext.cs` open. The code defines a `Category` entity with three initial entries: Action, SciFi, and History. The `Categories` property is annotated with a red squiggle underlining, indicating a potential issue or warning. The right side of the screen displays the Solution Explorer, showing the project structure for the 'Bulky' solution, which includes four projects: `Bulky.DataAccess`, `Bulky.Models`, `Bulky.Utility`, and `BulkyWeb`. The `Bulky.DataAccess` project is expanded, showing its contents: `Dependencies`, `Data` (containing `ApplicationDbContext.cs`), `Migrations` (containing two migration files: `20230124010956\_AddCateg` and `20230124013649\_SeedCateg`), and `DbContextModelK`.

Change namespace reference everywhere in Solution

view.cshtml Error.cshtml Program.cs HomeController.cs Category.cs

```
1 @model ErrorViewModel
2 @{
3     ViewData["Title"] = "Error";
4 }
5
6 <h1 class="text-danger">Error.</h1>
7 <h2 class="text-danger">An error occurred while processing your request.</h2>
```

Delete Database from SQL Server

```
8 [ModelBinder(BinderType=ModelBinders.Binders.HttpPost)]
9 public void ShowRequest()
10 {
11     <p>Request ID:<strong>1</strong></p>
12 }
```

Delete All Migrations Files

```
13 [ModelBinder(BinderType=ModelBinders.Binders.HttpPost)]
14 public void DeleteMigrations()
15 {
16     <p>Delete All Migrations</p>
17 }
```

Run “add-migration” command

```
18 [ModelBinder(BinderType=ModelBinders.Binders.HttpPost)]
19 public void AddMigration()
20 {
21     <p>Run “add-migration” command</p>
22 }
```

Make sure to select DataAccess project in PM console

```
23 [ModelBinder(BinderType=ModelBinders.Binders.HttpPost)]
24 public void SelectDataAccessProject()
25 {
26     <strong>The Development environment shouldn't be enabled for deployed applications. It can result in displaying sensitive information from exceptions to end users. For local debugging, enable the <strong>Development</strong> environment by setting the <code>ASPNETCORE_ENVIRONMENT</code> environment variable to <code>Development</code> and restarting the app.</strong>
27 }
```

Microsoft Visual Studio



OK Cancel

Migrations and all its contents will be deleted permanently.

Solution Explorer

- Solution 'Bulky' (4 of 4 projects)
  - Bulky.DataAccess
    - Dependencies
    - Data
  - Migrations
    - 20230124010956\_AddCategory.cs
    - 20230124013649\_SeedCategory.cs
    - ApplicationDbContextModelSnapshot.cs
  - Bulky.Models
  - Bulky.Utility
  - BulkyWeb

Search Solution Explorer (Ctrl+Shift+F)

Diagnostic Tools

# How to Reset Database

# Dependency Injection Service Lifetimes

- **Transient**      New Service - every time requested
- **Scoped**      New Service - once per request
- **Singleton**      New Service - once per application lifetime

```
namespace DI_Service_Lifetime.Services
{
    public interface IScopedGuidService
    {
        string GetGuid();
    }
}
```

```
namespace DI_Service_Lifetime.Services
{
    public class TransientGuidService : ITransientGuidService
    {
        private readonly Guid Id;

        public TransientGuidService()
        {
            Id = Guid.NewGuid();
        }

        public string GetGuid()
        {
            return Id.ToString();
        }
    }
}
```

Program.cs

```
DI_Service_Lifetime.Close (Ctrl+F4)
```

```
1 using DI_Service_Lifetime.Services;
2
3 var builder = WebApplication.CreateBuilder(args);
4
5 // Add services to the container.
6 builder.Services.AddControllersWithViews();
7 builder.Services.AddSingleton<ISingletonGuidService, SingletonGuidService>();
8 builder.Services.AddTransient<ITransientGuidService, TransientGuidService>();
9 builder.Services.AddScoped<IScopedGuidService, ScopedGuidService>();
10
11 var app = builder.Build();
12
13 // Configure the HTTP request pipeline.
14 if (!app.Environment.IsDevelopment())
15 {
16     app.UseExceptionHandler("/Home/Error");
17     // The default HSTS value is 30 days. You may want to change this for product
18     app.UseHsts();
19 }
```

Solution Explorer

- Search Solution Explorer (Ctrl+.)
- Solution 'DI\_Service\_Lifetime' (1 of 1)
  - DI\_Service\_Lifetime
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
    - Models
    - Services
      - IScopedGuidService.cs
      - ISingletonGuidService.cs
      - ITransientGuidService.cs
      - ScopedGuidService.cs
      - SingletonGuidService.cs
      - TransientGuidService.cs
    - Views
    - appsettings.json
  - Program.cs

```
public class HomeController : Controller
{
    private readonly IScopedGuidService _scoped1;
    private readonly IScopedGuidService _scoped2;

    private readonly ISingletonGuidService _singleton1;
    private readonly ISingletonGuidService _singleton2;

    private readonly ITransientGuidService _transient1;
    private readonly ITransientGuidService _transient2;
```

```
public HomeController(IScopedGuidService scoped1,
                      IScopedGuidService scoped2,
                      ISingletonGuidService singleton1,
                      ISingletonGuidService singleton2,
                      ITransientGuidService transient1,
                      ITransientGuidService transient2)
{
    _singleton1 = singleton1;
    _singleton2 = singleton2;
    _transient1 = transient1;
    _transient2 = transient2;
    _scoped1 = scoped1;
    _scoped2 = scoped2;
}
```

```
public IActionResult Index()
{
    StringBuilder messages = new StringBuilder();
    messages.Append($"Transient 1 : {_transient1.GetGuid()}\n");
    messages.Append($"Transient 2 : {_transient2.GetGuid()}\n\n");
    messages.Append($"Scoped 1 : {_scoped1.GetGuid()}\n");
    messages.Append($"Scoped 2 : {_scoped2.GetGustring IScopedGuidService.GetGuid()}\n");
    messages.Append($"Singleton 1 : {_singleton1.GetGuid()}\n");
    messages.Append($"Singleton 2 : {_singleton2.GetGuid()}\n\n");

    return Ok(messages.ToString());
}
```

# Repository Pattern

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;

namespace Bulky.DataAccess.Repository.IRepository
{
    internal interface IRepository<T> where T : class
    {
        //T - Category
        IEnumerable<T> GetAll();
        T Get(Expression<Func<T, bool>> filter);
        void Add(T entity);
        void Remove(T entity);
        void RemoveRange(IEnumerable<T> entity);
    }
}
```

# IRepository Interface

```
namespace Bulky.DataAccess.Repository
{
    public class Repository<T> : IRepository<T> where T : class
    {
        private readonly ApplicationDbContext _db;
        internal DbSet<T> dbSet;
        public Repository(ApplicationDbContext db)
        {
            _db = db;
            this.dbSet = _db.Set<T>();
            // _db.Categories == dbSet
        }

        public void Add(T entity)
        {
            dbSet.Add(entity);
        }

        public T Get(Expression<Func<T, bool>> filter)
        {
            IQueryables<T> query = dbSet;
            query = query.Where(filter);
            return query.FirstOrDefault();
        }

        public IEnumerable<T> GetAll()
        {
            IQueryables<T> query = dbSet;
            return query.ToList();
        }

        public void Remove(T entity)
        {
            dbSet.Remove(entity);
        }

        public void RemoveRange(IEnumerable<T> entity)
        {
            dbSet.RemoveRange(entity);
        }
    }
}
```

# Implement Repository Interface

```
namespace Bulky.DataAccess.Repository.IRepository
{
    public interface ICategoryRepository : IRepository<Category>
    {
        void Update(Category obj);
        void Save();
    }
}
```

```
namespace Bulky.DataAccess.Repository
{
    public class CategoryRepository : Repository<Category>, ICategoryRepository
    {
        private ApplicationDbContext _db;
        public CategoryRepository(ApplicationDbContext db) : base(db)
        {
            _db = db;
        }

        public void Save()
        {
            _db.SaveChanges();    [ ]
        }

        public void Update(Category obj)
        {
            _db.Categories.Update(obj);
        }
    }
}
```

# Implement ICategoryRepository and CategoryRepository

```

public class CategoryController : Controller
{
    private readonly ICategoryRepository _categoryRepo;
    public CategoryController(ICategoryRepository db)
    {
        _categoryRepo = db;
    }
    public IActionResult Index()
    {
        List<Category> objCategoryList = _categoryRepo.GetAll().ToList();
        return View(objCategoryList);
    }

    public IActionResult Edit(int? id)
    {
        if(id==null || id == 0)
        {
            return NotFound();
        }
        Category? categoryFromDb = _categoryRepo.Get(u=>u.Id==id);
        //Category? categoryFromDb1 = _db.Categories.FirstOrDefault();
        //Category? categoryFromDb2 = _db.Categories.Where(u=>u.Id=
    }

    builder.Services.AddScoped<ICategoryRepository, CategoryRepository>();

    var app = builder.Build();
}

```

```

[HttpPost]
public IActionResult Create(Category obj)
{
    if (obj.Name == obj.DisplayOrder.ToString())
    {
        ModelState.AddModelError("name", "The Display Order must be greater than the Id");
    }
    if (ModelState.IsValid)
    {
        _categoryRepo.Add(obj);
        _categoryRepo.Save();
        TempData["success"] = "Category created successfully";
        return RedirectToAction("Index");
    }
    return View();
}

[HttpPost]
public IActionResult Edit(Category obj)
{
    if (ModelState.IsValid)
    {
        _categoryRepo.Update(obj);
        _categoryRepo.Save();
        TempData["success"] = "Category updated successfully";
        return RedirectToAction("Index");
    }
}

```

# Replace DbContext with Category Repository

```
namespace BulkyBook.DataAccess.Repository.IRepository
{
    public interface IUnitOfWork
    {
        ICategoryRepository CategoryRepository { get; }

        void Save();
    }
}
```

```
namespace BulkyBook.DataAccess.Repository
{
    public class UnitOfWork : IUnitOfWork
    {
        private ApplicationDbContext _db;
        public ICategoryRepository Category { get; private set; }
        public UnitOfWork(ApplicationDbContext db)
        {
            _db = db;
            Category = new CategoryRepository(_db);
        }

        public void Save()
        {
            _db.SaveChanges();
        }
    }
}
```

# UnitOfWork Implementation

```
builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();
```

```
public class CategoryController : Controller
{
    private readonly IUnitOfWork _unitOfWork;
    public CategoryController(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
    }
    public IActionResult Index()
    {
        List<Category> objCategoryList = _unitOfWork.Category.GetAll().ToList();
        return View(objCategoryList);
    }
}
```

## UnitOfWork in Action

The screenshot shows the 'Add New Scaffolded Item' dialog box in Visual Studio. The left pane lists installed and common scaffolds: Installed includes MVC Area, MVC Controller - Empty, MVC Controller with read/write actions, MVC Controller with views, using Entity Framework, API Controller - Empty, API Controller with read/write actions, API Controller with actions, using Entity Framework, API with read/write endpoints, API with read/write endpoints, using Entity Framework, Razor View - Empty, Razor View, and Razor Component. Common includes API, MVC (selected), Razor Component, Razor Pages, Identity, and Layout. The right pane details the selected 'MVC Area' scaffold, which is described as an 'MVC area that can have its own models, views, controllers, and routes.' The bottom right buttons are 'Add' and 'Cancel'.

## Add 2 Areas : Admin and Customer

# Areas in .NET

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

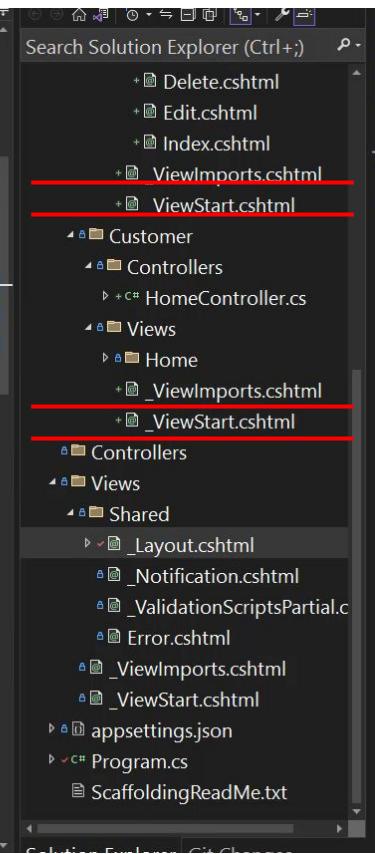
Data and Model folders we don't need so we can delete them from both areas.

Move “Home Controller” to Customer Area and “Category Controller” to Admin Area.

Tell controller belongs to which specific area?

```
[Area("Admin")]
public class CategoryController : Controller
```

```
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-dark bg-p
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-act
                <button class="navbar-toggler" type="button" data-bs-toggle="coll
                    aria-expanded="false" aria-label="Toggle navigation">
                        <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-con
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link" asp-area="Customer" asp-controller="Customer" asp-action="Index">Customer</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" asp-area="Admin" asp-controller="Admin" asp-action="Index">Admin</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" asp-area="Customer" asp-controller="Customer" asp-action="Index">Customer</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
```



# Update Links with asp-area



Validation

## Components

Accordion

Alerts

Badge

Breadcrumb

Buttons

Button group

Card

Carousel

Close button

Collapse

Dropdowns

List group

Modal

**NavBar**

Navs & tabs

```
<a class="navbar-brand" href="#">Navbar</a>
<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent">
  <span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarSupportedContent">
  <ul class="navbar-nav me-auto mb-2 mb-lg-0">
    <li class="nav-item">
      <a class="nav-link active" aria-current="page" href="#">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
    <li class="nav-item dropdown">
      <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" href="#">Dropdown
        </a>
      <ul class="dropdown-menu">
        <li><a class="dropdown-item" href="#">Action</a></li>
        <li><a class="dropdown-item" href="#">Another action</a></li>
        <li><hr class="dropdown-divider"></li>
        <li><a class="dropdown-item" href="#">Something else here</a></li>
      </ul>
    </li>
    <li class="nav-item">
      <a class="nav-link disabled" href="#">Disabled</a>
    </li>
  </ul>
</div>
```

## On this page

How it works

Supported content

Brand

Text

Image

Image and text

Nav

Forms

Text

Color schemes

Containers

Placement

Scrolling

Responsive behaviors

Toggler

External content

Offcanvas

CSS

Variables

# Dropdown in NavBar

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="d...
    Content Management
  </a>
  <ul class="dropdown-menu">
    <li class="nav-item">
      <a class="dropdown-item" asp-area="Admin" asp-controller="Category" ...
    </li>
    <li><hr class="dropdown-divider"></li>

  </ul>
</li>
```

# DropDown in Design

# Product CRUD

```
public class Product
{
    [Key]
    public int Id { get; set; }
    [Required]
    public string Title { get; set; }
    public string Description { get; set; }
    [Required]
    public string ISBN { get; set; }
    [Required]
    public string Author { get; set; }
    [Required]
    [Display(Name = "List Price")]
    [Range(1, 1000)]
    public double ListPrice { get; set; }
}
```

```
[Required]
[Display(Name = "Price for 1-50")]
[Range(1, 1000)]
public double Price { get; set; }

[Required]
[Display(Name = "Price for 50+")]
[Range(1, 1000)]
public double Price50 { get; set; }

[Required]
[Display(Name = "Price for 100+")]
[Range(1, 1000)]
public double Price100 { get; set; }
```

## Create Product Model

```
public DbSet<Category> Categories { get; set; }
public DbSet<Product> Products{ get; set; }

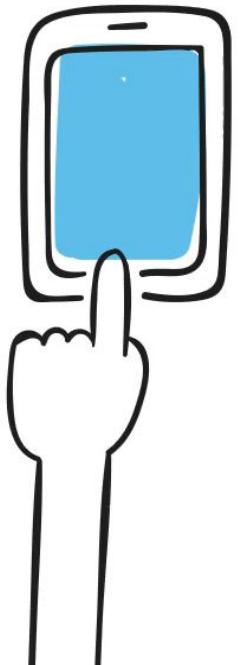
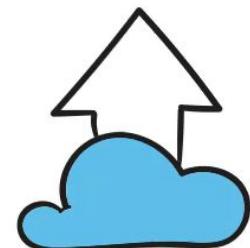
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Category>().HasData(
        new Category { Id = 1, Name = "Action", DisplayOrder = 1 },
        new Category { Id = 2, Name = "SciFi", DisplayOrder = 2 },
        new Category { Id = 3, Name = "History", DisplayOrder = 3 }
    );

    modelBuilder.Entity<Category>().HasData(
    );
}
```

## Seed Product

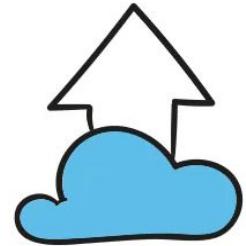
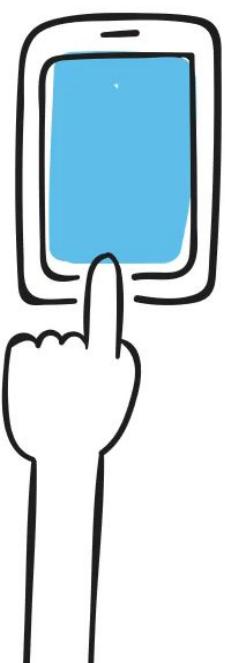


# ASSIGNMENT - 1



## Product Management

- Push Product to Database
- Implement Product Repository
- Configure Product Repository in UnitOfWork.



## ASSIGNMENT - 2

1. Create Product Controller & Action Methods  
(Create/Edit/Delete/Index)
2. Create Views for Create/Edit/Delete/Index
3. Add Client Side and Server-Side Validations.

```
[Required]
[Display(Name = "Price for 100+")]
[Range(1, 1000)]
public double Price100 { get; set; }

public int CategoryId { get; set; }
[ForeignKey("CategoryId")]
public Category Category { get; set; }
```

```
modelBuilder.Entity<Product>().HasData(
    new Product
    {
        Id = 1,
        Title = "Fortune of Time",
        Author = "Billy Spark",
        Description = "Praesent vitae soc",
        ISBN = "SWD9999001",
        ListPrice = 99,
        Price = 90,
        Price50 = 85,
        Price100 = 80,
        CategoryId = 1,
    },
    new Product
    {
        Id = 2,
        Title = "The Last King",
        Author = "John Smith",
        Description = "Inventoribus et ceteris",
        ISBN = "SWD9999002",
        ListPrice = 120,
        Price = 110,
        Price50 = 105,
        Price100 = 100,
        CategoryId = 2,
    }
);
```

Don't forget “add-migration” and “update-database” command...

## Add Foreign Key in EF Core

```
public int CategoryId { get; set; }  
[ForeignKey("CategoryId")]  
public Category Category { get; set; }  
public string ImageUrl { get; set; }
```

```
new Product  
{  
    Id = 1,  
    Title = "Fortune of Time",  
    Author = "Billy Spark",  
    Description = "Praesent  
    ISBN = "SWD9999001",  
    ListPrice = 99,  
    Price = 90,  
    Price50 = 85,  
    Price100 = 80,  
    CategoryId = 1, [   
        ImageUrl = ""  
    },
```

Don't forget “add-migration” and “update-database” command...

## Add Image Url Column

```
public IActionResult Create()
{
    IEnumerable<SelectListItem> CategoryList = _unitOfWork.Category
        .GetAll().Select(u => new SelectListItem
    {
        Text = u.Name,
        Value = u.Id.ToString()
    });
    return View();
}
```

ProductController -> Create action

## Projections in EF Core

# VIEWBAG

- ViewBag transfers data from the Controller to View, not vice-versa. Ideal for situations in which the temporary data is not in a model.
- ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0
- Any number of properties and values can be assigned to ViewBag
- The ViewBag's life only lasts during the current http request. ViewBag values will be null if redirection occurs.
- ViewBag is actually a wrapper around ViewData.

```
public IActionResult Create()
{
    IEnumerable< SelectListItem> CategoryList = _unitOfWork.Category
        .GetAll().Select(u => new SelectListItem
    {
        Text = u.Name,
        Value = u.Id.ToString()
    });

    ViewBag.CategoryList = CategoryList;
    return View();
}
```

```
<div class="form-floating py-2 col-12">
    <select asp-for="CategoryId" asp-items="ViewBag.CategoryList" class="form-select border-0 shadow">
        <option disabled selected>--Select Category--</option>
    </select>
    <label asp-for="CategoryId" class="ms-2"></label>
    <span asp-validation-for="CategoryId" class="text-danger"></span>
</div>
```

## ViewBag in Action

## VIEWBAG

- ViewBag transfers data from the Controller to View, not vice-versa. Ideal for situations in which the temporary data is not in a model.
- ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0
- Any number of properties and values can be assigned to ViewBag
- The ViewBag's life only lasts during the current http request. ViewBag values will be null if redirection occurs.
- ViewBag is actually a wrapper around ViewData.

ViewBag internally inserts data into ViewData dictionary. So the key of ViewData and property of ViewBag must **NOT** match

## VIEWDATA

- ViewData transfers data from the Controller to View, not vice-versa. Ideal for situations in which the temporary data is not in a model.
- ViewData is derived from ViewDataDictionary which is a dictionary type.
- ViewData value must be type cast before use.
- The ViewData's life only lasts during the current http request. ViewData values will be null if redirection occurs.

## TEMPDATA

- TempData can be used to store data between two consecutive requests.
- TempData internally uses Session to store the data. So think of it as a short lived session.
- TempData value must be type cast before use. Check for null values to avoid runtime error.
- TempData can be used to store only one time messages like error messages, validation messages.



## ViewData in Action

```
public IActionResult Create()
{
    IEnumerable<SelectListItem> CategoryList = _unitOfWork.Category
        .GetAll().Select(u => new SelectListItem
    {
        Text = u.Name,
        Value = u.Id.ToString()
    });

    //ViewBag.CategoryList = CategoryList;
    ViewData["CategoryList"] = CategoryList;

    return View();
}
```

```
<div class="form-group row" style="margin-bottom: 10px;">
    <label for="CategoryId" class="col-2 col-form-label">Category
    <div class="col-10">
        <select id="CategoryId" name="CategoryId" class="form-control">
            <option value="" disabled selected>--Select Category--</option>
            @for (var item in ViewData["CategoryList"])
            {
                <option value="@item.Value" @if(item.Value == Model.CategoryId){>selected</if>>>@item.Text</option>
            }
        </select>
    </div>

```

## ViewData in Action

The screenshot shows the 'Add New Item' dialog in Visual Studio. The 'Type: C# Items' dropdown is open, displaying various item templates. The 'Name:' field contains 'ProductVM'. The 'Add' button is highlighted with a mouse cursor.

Search Solution Explorer (Ctrl+;) Solution 'Bulky' (4 of 4 projects)

Add New Item - BulkyBook.Models

Sort by: Default

Type: C# Items

Name: ProductVM

Add Cancel

Installed

C# Items

- Code
- Data
- General
- Web
  - SQL Server
  - Storm Items
- Online

Class C# Items

- EF Core Database First Wizard C# Items
- Class for U-SQL C# Items
- Interface C# Items
- Component Class C# Items
- ADO.NET Entity Data Model C# Items
- Application Configuration File C# Items
- Application Manifest File (Win...) C# Items
- Assembly Information File C# Items
- Bitmap File C# Items
- Code Analysis Rule Set C# Items
- Code File C# Items
- Cursor File C# Items

Search (Ctrl+E) List

BulkyBook.DataAccess

BulkyBook.Models

- Dependencies
- ViewModels
  - Category.cs
  - ErrorViewModel.cs
  - Product.cs

BulkyBook.Utility

BulkyBookWeb

- Connected Services
- Dependencies
- Properties
- wwwroot

Areas

- Admin
- Controllers
  - CategoryController.cs
  - ProductController.cs
- Views
  - Category
    - Create.cshtml

```
namespace BulkyBook.Models.ViewModels
{
    public class ProductVM
    {
        public Product Product { get; set; }
        public IEnumerable< SelectListItem> CategoryList { get; set; }
    }
}
```

# View Models in Action

```
@model ProductVM
```

```
<div class="card shadow border-0 my-4">
    <div class="card-header bg-secondary bg-gradient ml-0 py-3">
        <div class="row">
            <div class="col-12 text-center">
                <h2 class="text-white py-2">Create Product</h2>
            </div>
        </div>
    </div>
    <div class="card-body p-4">
        <form method="post" class="row">
            <div class="border p-3">
                @*<div asp-validation-summary="ModelOnly"></div>*@
                <div class="form-floating py-2 col-12">
                    <input asp-for="Product.Title" class="form-control border-0 :>
                    <label asp-for="Title" class="ms-2"></label>
                    <span asp-validation-for="Title" class="text-danger"></span>
                </div>
```

```
        <div class="form-floating py-2 col-12">
            <select asp-for="@Model.Product.CategoryId" asp-items="@Model.CategoryList">
                <option disabled selected>--Select Category--</option>
            </select>
            <label asp-for="Product.CategoryId" class="ms-2"></label>
            <span asp-validation-for="Product.CategoryId" class="text-danger"></span>
        </div>
```

```
public IActionResult Create()
{
    ProductVM productVM = new()
    {
        CategoryList = _unitOfWork.Category
            .GetAll().Select(u => new SelectListItem
        {
            Text = u.Name,
            Value = u.Id.ToString()
        }),
        Product = new Product()
    };
    return View(productVM);
}
```

```
public IActionResult Create(ProductVM productVM)
{
    if (ModelState.IsValid)
    {
        _unitOfWork.Product.Add(productVM.Product);
        _unitOfWork.Save();
        TempData["success"] = "Product created successfully";
        return RedirectToAction("Index");
    }
    else
    {
        productVM.CategoryList = _unitOfWork.Category.GetAll().Select(u =>
        {
            Text = u.Name,
            Value = u.Id.ToString()
        });
        return View(productVM);
    }
}
```

```
public class ProductVM
{
    public Product Product { get; set; }
    [ValidateNever]
    public IEnumerable<SelectListItem> CategoryList { get; set; }
}
```

[ValidateNever] for CategoryList and Image column

```
<div class="form-floating py-2 col-12">
    <input type="file" class="form-control border-0 shadow" />
    <label asp-for="Product.ImageUrl" class="ms-2"></label>
</div>


<form method="post" class="row" enctype="multipart/form-data">
        <div class="border p-3">


```

# File Upload Input

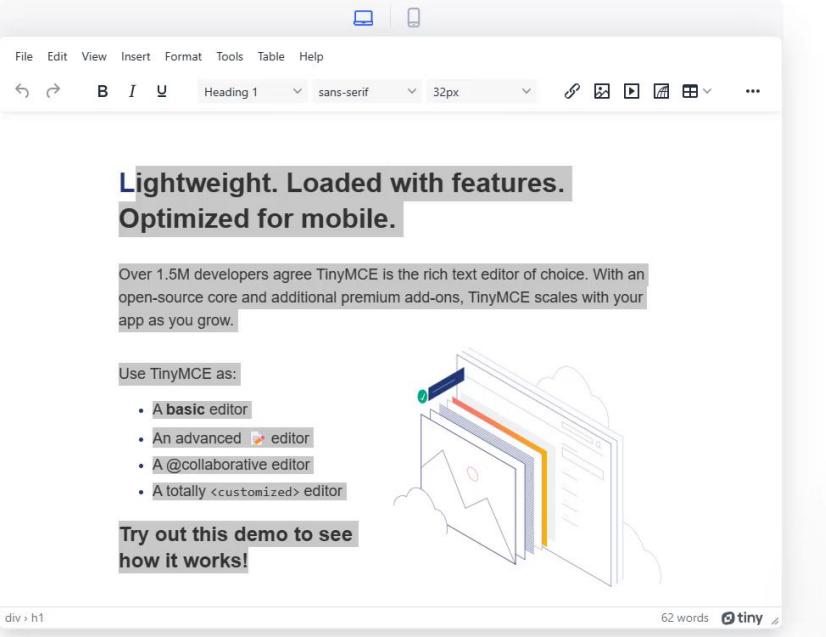
```
if (id == null || id == 0)
{
    //create
    return View(productVM);
}
else
{
    //update
    productVM.Product = _unitOfWork.Product.Get(u=>u.Id==id);
    return View(productVM);
}
```

```
[HttpPost]
public IActionResult Upsert(ProductVM productVM, IFormFile? file)
{
    if (ModelState.IsValid)
    {
        _unitOfWork.Product.Add(productVM.Product);
        _unitOfWork.Save();
        TempData["success"] = "Product created successfully";
        return RedirectToAction("Index");
    }
    else
    {
        productVM.CategoryList = _unitOfWork.Category.GetAll().Select(
        {
            Text = u.Name,
            Value = u.Id.ToString()
        });
        return View(productVM);
    }
}
```

# Combine Create and Edit Pages

START NOW: Free 14-day trial of Premium plugins

SIGN UP



The screenshot shows a demonstration of the TinyMCE rich text editor. At the top, there's a toolbar with icons for file operations, text styling (bold, italic, underline), headings (Heading 1, Heading 2), font family (sans-serif), font size (32px), and other tools like link, image, and table. Below the toolbar, the main content area displays the text: "Lightweight. Loaded with features. Optimized for mobile." A cursor arrow is visible on the left side of the editor window. To the right of the text, there's a section titled "Use TinyMCE as:" with a bulleted list: "A basic editor", "An advanced editor", "A @collaborative editor", and "A totally <customized> editor". Next to this list is a 3D-style illustration of a stack of documents or web pages. At the bottom left, there's a call-to-action button: "Try out this demo to see how it works!". At the very bottom of the editor window, it says "div > h1" and "62 words".

# Rich Text Editor

# Creating Web API

## Create a new project

### Recent project templates

Class Library

C#

Console App

C#

ASP.NET Core Web App (Model-View-Controller)

C#

ASP.NET Core Web API

C#

MVC

X

Clear all

All languages

All platforms

All project types



ASP.NET Core Web App (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core **MVC** Views and Controllers. This template can also be used for RESTful HTTP services.

C#

Linux

macOS

Windows

Cloud

Service

Web



ASP.NET Core Web App (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core **MVC** Views and Controllers. This template can also be used for RESTful HTTP services.

C#

Linux

macOS

Windows

Cloud

Service

Web



ASP.NET Core Web API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core **MVC** Views and Controllers.

F#

Linux

macOS

Windows

Cloud

Service

Web

Web API

Not finding what you're looking for?

[Install more tools and features](#)

Back

Next

# Create ASP.NET Core Web API



Swagger

Supported by SMARTBEAR

Select a definition

TestAPI v1



## TestAPI 1.0 OAS 3.0

<https://localhost:7286/swagger/v1/swagger.json>

### WeatherForecast



GET

/WeatherForecast



### Schemas



WeatherForecast >



# Run and Test Swagger

← → Home Workspaces API Network

Search Postman

Invite Settings Notifications Upgrade

Workspace New Import Overview GET Get Activities Dev GET Get Activity POST Create Activity PUT Edit Activity DEL Delete Activity GET Get Activities + Dev

Reactivities / Module 4 / Get Activities

Save Share

Send

Params Authorization Headers (7) Body Scripts Settings Cookies

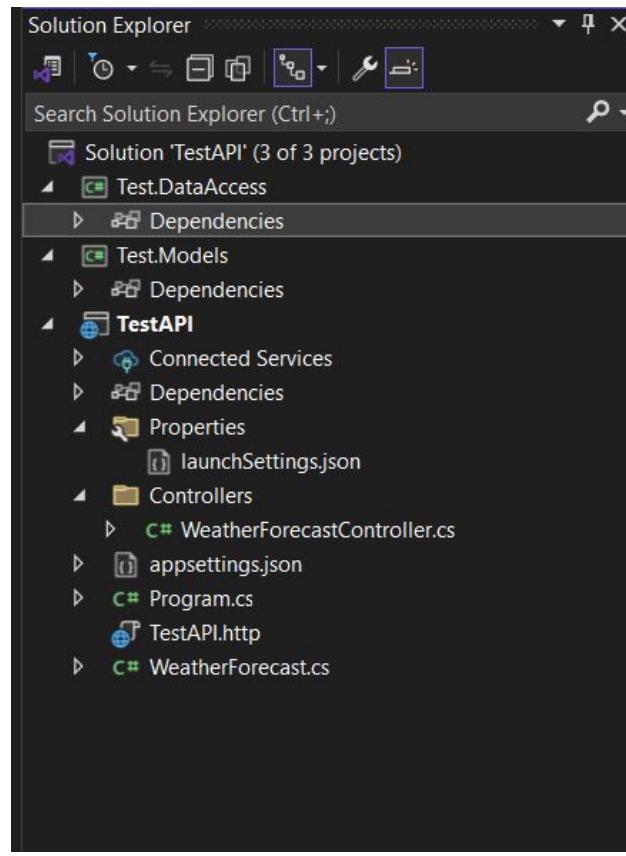
Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Response History

Click Send to get a response

# Postman



# Follow N-Tier Architecture Pattern

```
namespace Test.Models
{
    0 references
    public class Activity
    {
        0 references
        public Guid Id { get; set; }
        0 references
        public string Title { get; set; }
        0 references
        public DateTime Date { get; set; }
        0 references
        public string Description { get; set; }
        0 references
        public string Category { get; set; }
        0 references
        public string City { get; set; }
        0 references
        public string Venue { get; set; }
    }
}
```

# Create Activity Model Class

```
namespace Test.DataAccess
{
    1 reference
    public class ApplicationDbContext : DbContext
    {
        0 references
        public ApplicationDbContext(DbContextOptions options) : base(options)
        {
        }

        0 references
        public DbSet<Activity> Activities { get; set; }

        // Seed Data
        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {

            modelBuilder.Entity<Activity>().HasData(
                new Activity
                {
                    Id = new Guid(),
                    Title = "Past Activity 1",
                    Date = DateTime.UtcNow.AddMonths(-2),
                    Description = "Activity 2 months ago",
                    Category = "drinks",
                    City = "London",
                    Venue = "Pub",
                }
            );
        }
    }
}
```

# Create ApplicationDbContext class

```
builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddDbContext<ApplicationDbContext>(opt => opt.UseSqlServer(
    builder.Configuration.GetConnectionString("DefaultConnection"))
);

},
"ConnectionStrings": {
    "DefaultConnection": "Server=LAPTOP-TGF3N4T8;Database=WebApi;Trusted_Connection=True;TrustServerCertificate=True;"
},
```

## Add-migration and update-database

Add DbContext with program.cs file

```
namespace TestAPI.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class BaseApiController : ControllerBase
    {
    }
}
```

```
namespace TestAPI.Controllers
{
    1 reference
    public class ActivitiesController : BaseApiController
    {
        private readonly ApplicationDbContext _context;
        0 references
        public ActivitiesController(ApplicationDbContext context)
        {
            _context = context;
        }

        [HttpGet]
        0 references
        public async Task<ActionResult<List<Activity>>> GetActivities()
        {
            return await _context.Activities.ToListAsync();
        }

        [HttpGet("{id}")]
        0 references
        public async Task<ActionResult<Activity>> GetActivity(int id)
        {
            return await _context.Activities.FindAsync(id);
        }
    }
}
```

# Adding an API Controller

# Creating a React Project

# Vite

## Next Generation Frontend Tooling

Get ready for a development environment that can finally catch up with you.

[Get Started](#)[Why Vite?](#)[View on GitHub](#)[ViteConf 23!](#)

### Instant Server Start

On demand file serving over native ESM, no bundling required!



### Lightning Fast HMR

Hot Module Replacement (HMR) that stays fast regardless of app size.



### Rich Features

Out-of-the-box support for TypeScript, JSX, CSS and more.

# Why React?





NETFLIX



# React is fast

# Easy to learn

# It's just a library

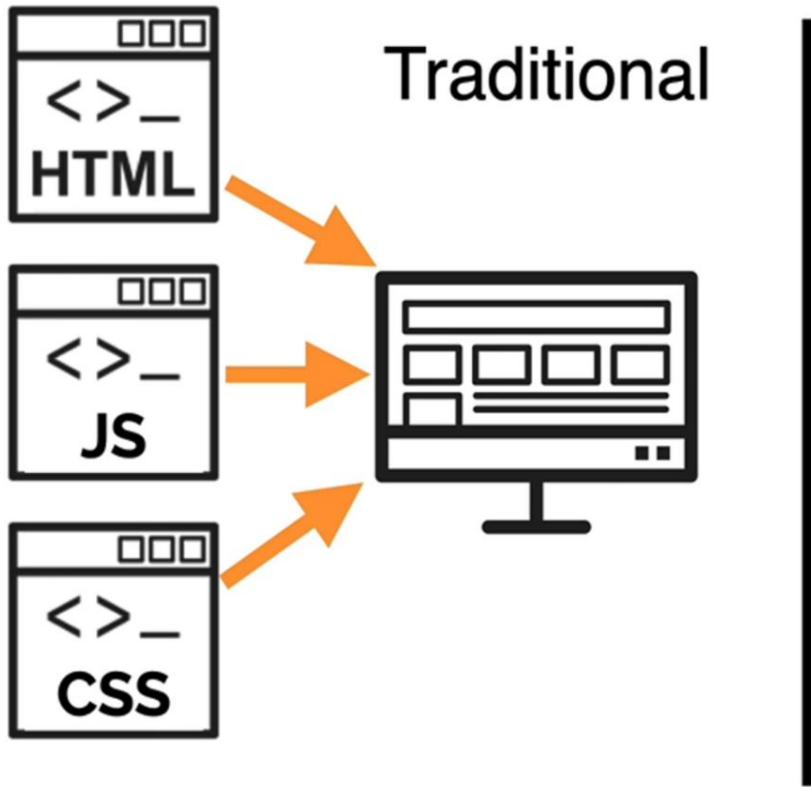
# It's just javascript\*

\*or typescript

# React Components

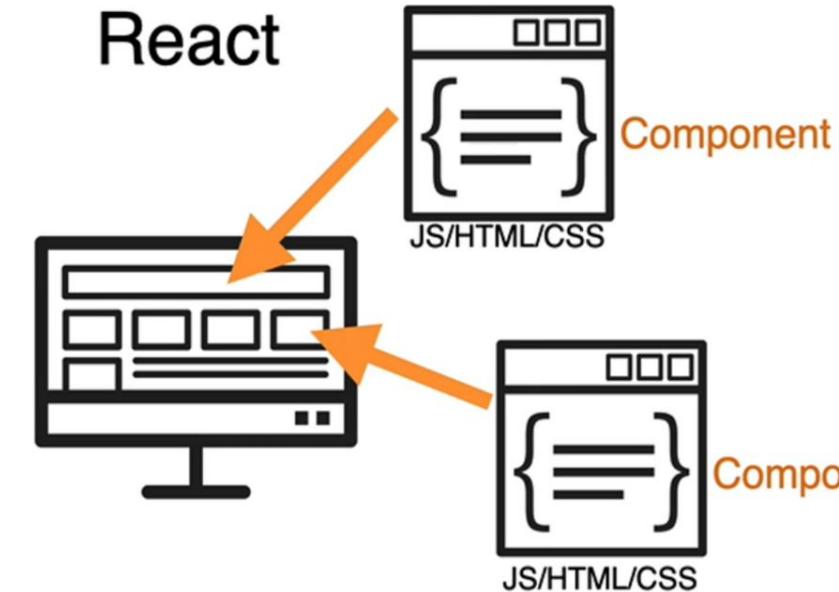


# Components



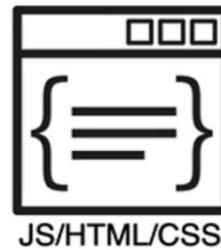
Traditional

React



# Components

Component



JS/HTML/CSS

State

Component



JS/HTML/CSS

Props

Component



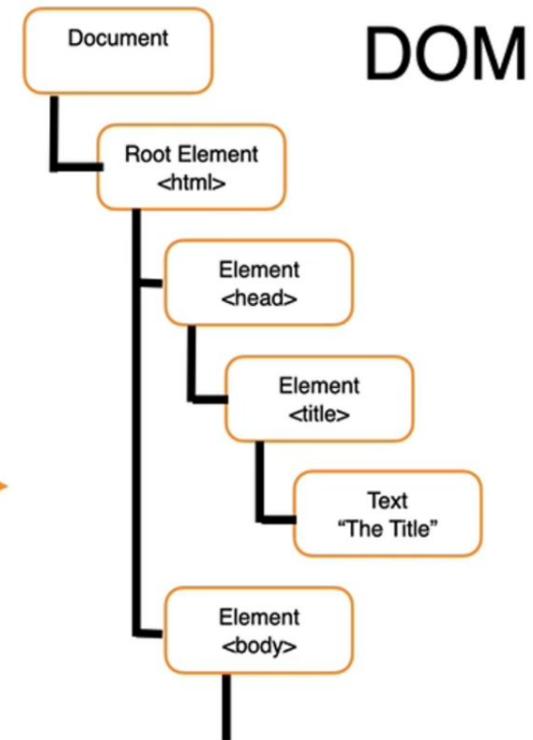
JS/HTML/CSS

Props

# Virtual DOM



Updates DOM  
directly



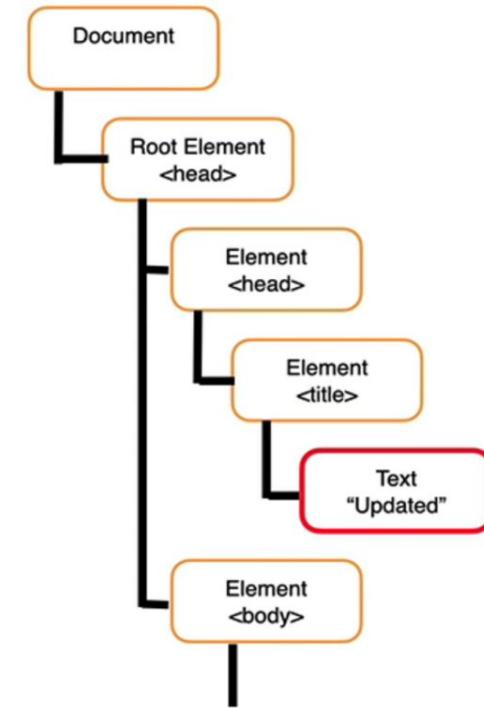
# Virtual DOM

## React Component

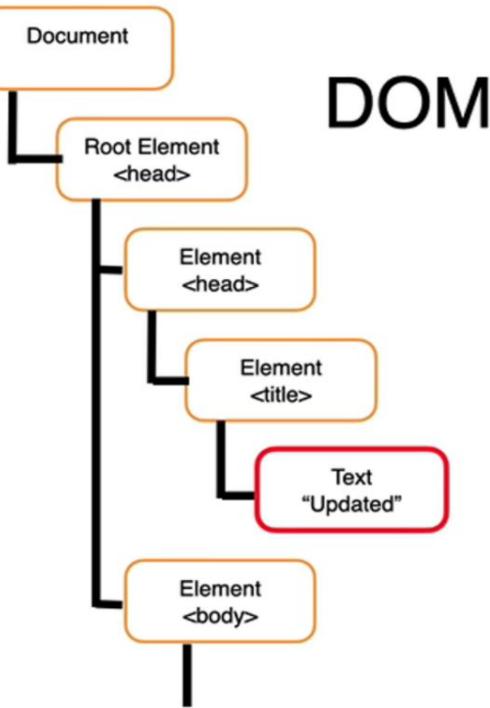


JS/HTML/CSS

Title = "Updated"



## Virtual DOM

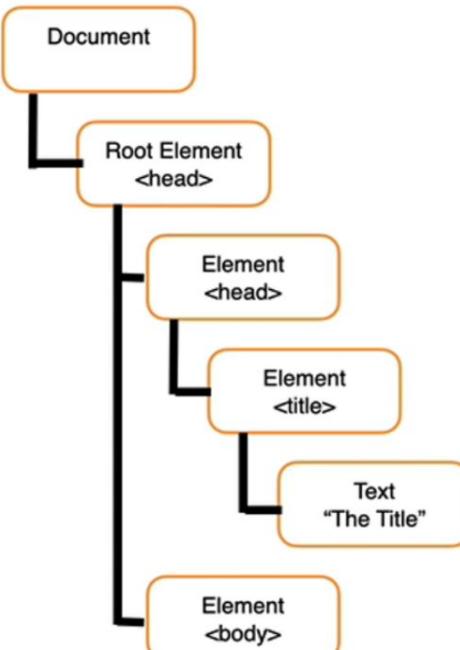
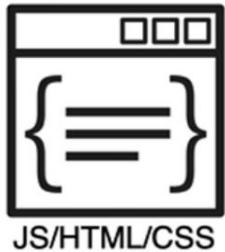


Compute diff

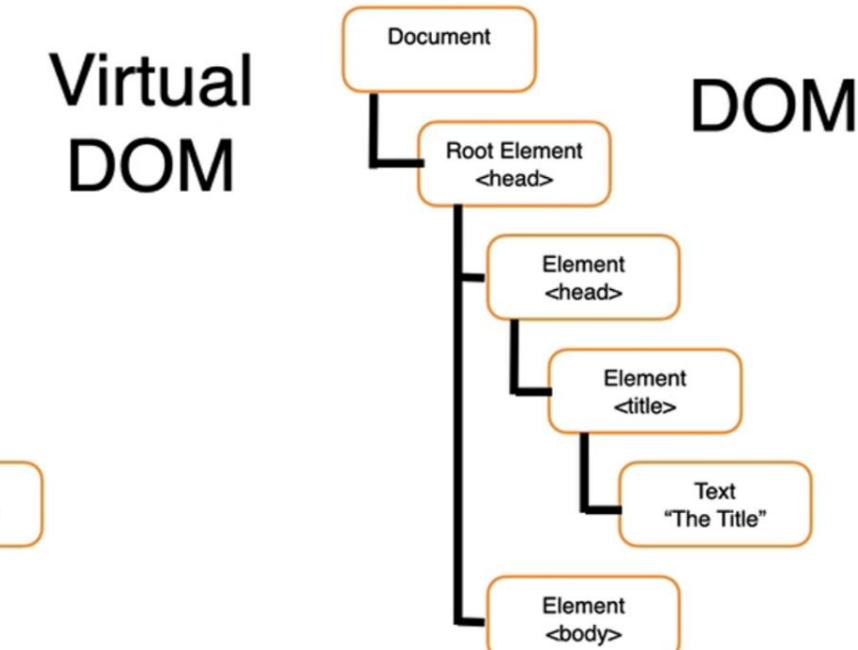
Re-render

# One way binding

## React Component



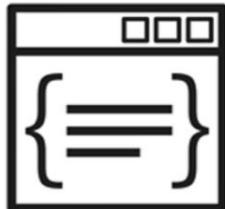
## Virtual DOM



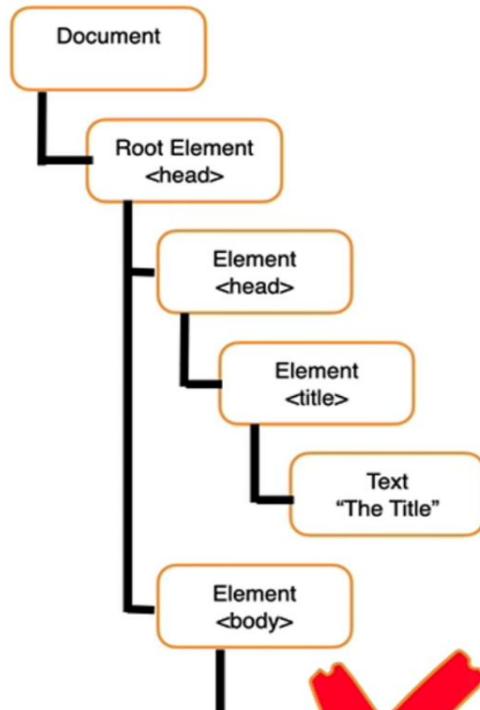
DOM

# One way binding

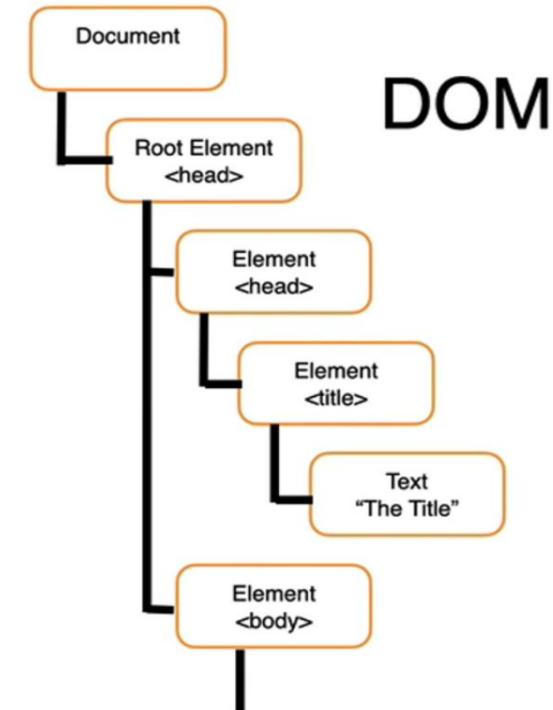
React  
Component



JS/HTML/CSS



Virtual  
DOM



DOM

# JSX

App.tsx X

client-app > src > App.tsx > ...

```
1 import React from 'react';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <h1>Reactivities</h1>
8     </div>
9   );
10 }
11
12 export default App;
13
```

# JSX

App.tsx X

client-app > src > App.tsx > ...

```
1 import React from 'react';
2 import './App.css';
3
4 function App() {
5   return (
6     React.createElement('div', {className: 'app'}),
7     React.createElement('h1', null, 'Reactivities')
8   );
9 }
10
11 export default App;
12 |
```

# React Hooks



## useState()

```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

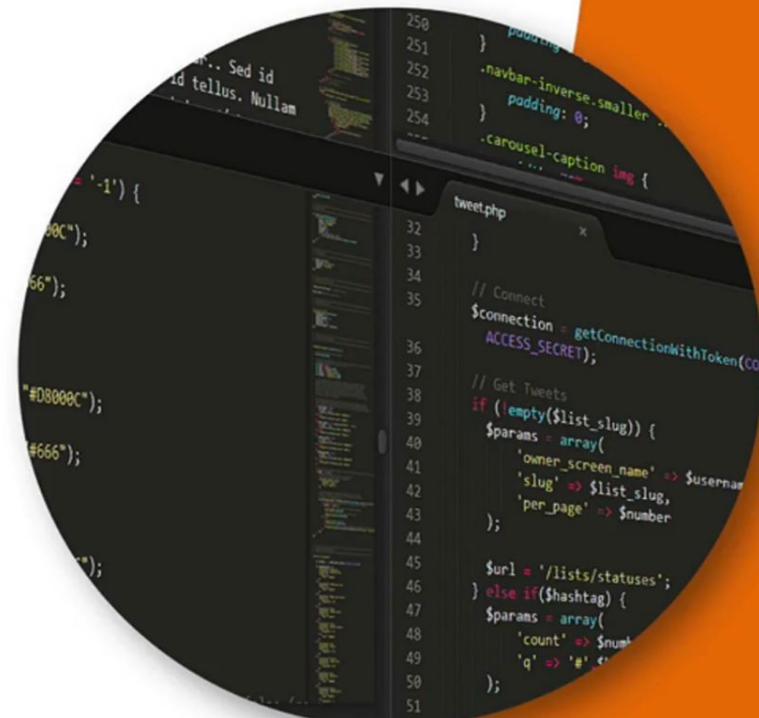
  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```



## useEffect()

# TypeScript



## TypeScript Rocks!

- Strong Typing
- Object orientated
- Better intellisense
- Access modifiers
- Future JS Features
- Catches silly mistakes in dev
- 3rd Party libraries
- Easy to learn if you know JS
- Much improved in React

## TypeScript is annoying!

- More upfront code
- 3rd Party libraries
- Strict mode is... strict!

# TypeScript demo

TypeScript Download Docs Handbook Community Playground Tools

# TypeScript is JavaScript with syntax for types.

TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale.

Try TypeScript Now

Online or via npm



Editor Checks Auto-complete Interfaces JSX

```
const user = {  
    firstName: "Angela",  
    lastName: "Davis",  
    role: "Professor",  
}  
  
console.log(user.name)  
  
Property 'name' does not exist on type '{ firstName: string;  
lastName: string; role: string; }'.
```



TypeScript 5.7 is now available, 5.8 is currently in beta.

## What is TypeScript?

JavaScript and More

A Result You Can Trust

Safety at Scale



## React Developer Tools

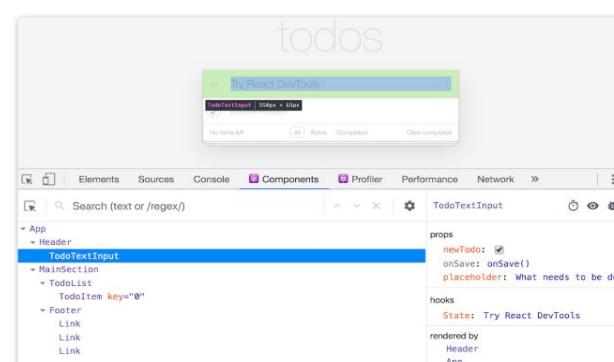
Featured 4.0 ★ (1.6K ratings)

Extension

Developer Tools

4,000,000 users

Add to Chrome



## Overview

Adds React debugging tools to the Chrome Developer Tools.

Created from revision c7c68ef842 on 10/15/2024.

React Developer Tools is a Chrome DevTools extension for the open-source React JavaScript library. It allows you to inspect the React component hierarchies in the Chrome Developer Tools.



English

## Getting Started

### Introduction

Example

POST Requests

### Axios API

Axios API

The Axios Instance

Request Config

Response Schema

Config Defaults

Interceptors

Handling Errors

Cancellation

URL-Encoding Bodies

Multipart Bodies

### Other

Notes

### Contributors

Sponsoring Axios

Code of Conduct

Collaborator Guide



# Getting Started

Promise based HTTP client for the browser and node.js

## What is Axios?

Axios is a [promise-based](#) HTTP Client for [node.js](#) and the browser. It is [isomorphic](#) (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js [http](#) module, while on the client (browser) it uses XMLHttpRequests.

## Features

- Make [XMLHttpRequests](#) from the browser
- Make [http](#) requests from node.js
- Supports the [Promise](#) API
- Intercept request and response
- Transform request and response data
- Cancel requests
- Timeouts
- Query parameters serialization with support for nested entries
- Automatic request body serialization to:
  - JSON ([application/json](#))
  - Multipart / FormData ([multipart/form-data](#))
  - URL encoded form ([application/x-www-form-urlencoded](#))
- Posting HTML forms as JSON
- Automatic JSON data handling in response
- Progress capturing for browsers and node.js with extra info (speed rate, remaining time)
- Setting bandwidth limits for node.js
- Compatible with spec-compliant FormData and Blob (including [node.js](#))
- Client side support for protecting against [XSRF](#)

## Installing

Using npm:

# Fetching Data From API

```

function App() {
  const [activities, setActivities] = useState([]);

  useEffect(() => {
    axios.get('http://localhost:5000/api/activities')
      .then(response => {
        setActivities(response.data)
      })
  }, []);

  return (
    <div>
      <h1>Reactivities</h1>
      <ul>
        {activities.map((activity: any) => (
          <li key={activity.id}>
            {activity.title}
          </li>
        ))}
      </ul>
    </div>
  )
}

```

localhost:3000

## Reactivities

The screenshot shows the Network tab in the Chrome DevTools developer console. It lists several network requests made by the browser to the endpoint 'http://localhost:5000/api/activities'. Most of these requests are blocked due to CORS policy issues, resulting in 'ERR\_FAILED 200 OK' status codes. One request is successful ('OK'). The requests are as follows:

- Access to XMLHttpRequest at 'http://localhost:5000/api/activities' from origin 'http://localhost:3000' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
- GET http://localhost:5000/api/activities net::ERR\_FAILED 200 (OK)
- Uncaught (in promise) > AxiosError {message: "Network Error", name: "AxiosError", code: "ERR\_NETWORK", config: {...}, request: XMLHttpRequest, ...}
- Access to XMLHttpRequest at 'http://localhost:5000/api/activities' from origin 'http://localhost:3000' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
- GET http://localhost:5000/api/activities net::ERR\_FAILED 200 (OK)
- Uncaught (in promise) > AxiosError {message: "Network Error", name: "AxiosError", code: "ERR\_NETWORK", config: {...}, request: XMLHttpRequest, ...}

# CORS Error

# CORS Policy

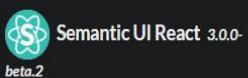
```
builder.Services.AddCors(opt => {
    opt.AddPolicy("CorsPolicy", policy =>
    {
        policy.AllowAnyHeader().AllowAnyMethod().WithOrigins("http://localhost:3000");
    });
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseCors("CorsPolicy");
app.UseAuthorization();
```



Semantic UI React 3.0.0

beta.2

GitHub

CHANGELOG

## Getting Started

### Introduction

Get Started

Composition

Shorthand Props

Theming

Layout examples

Prototypes

Migration guide to v3

Press "/" to filter components



### Elements

Button

Flag

Container



Build your website for just  
\$3.88/mth. More value and  
performance with Namecheap.

ADS VIA CARBON



# Semantic UI React

The official Semantic-UI-React integration.

## Introduction

Semantic UI React is the official React integration for [Semantic UI](#).

- ✓ jQuery Free
- ✓ Declarative API
- ✓ Augmentation
- ✓ Shorthand Props
- ✓ Sub Components
- ✓ Auto Controlled State

Installation instructions are provided in the [Usage](#) section.

Instead of Semantic UI  
we can also use  
Material UI

## jQuery Free

jQuery is a DOM manipulation library. It reads from and writes to the DOM. React uses a virtual DOM (a JavaScript representation of the real DOM). React



v19

Search

Ctrl K

Learn

Reference

Community

Blog



react@19

Overview

Hooks

useActionState

useCallback

useContext

useDebugValue

useDeferredValue

useEffect

useId

useImperativeHandle

useInsertionEffect

useLayoutEffect

Is this page useful?



Hooks. You can't call it inside loops or conditions. If you need that, extract a new component and move the state into it.

- If you're not trying to synchronize with some external system, [you probably don't need an Effect](#).
- When Strict Mode is on, React will run one extra development-only setup+cleanup cycle before the first real setup. This is a stress-test that ensures that your cleanup logic "mirrors" your setup logic and that it stops or undoes whatever the setup is doing. If this causes a problem, implement the cleanup function.
- If some of your dependencies are objects or functions defined inside the component, there is a risk that they will [cause the Effect to re-run more often than needed](#). To fix this, remove unnecessary [object](#) and [function](#) dependencies. You can also [extract state updates](#) and [non-reactive logic](#) outside of your Effect.
- If your Effect wasn't caused by an interaction (like a click), React will generally let the browser [paint the updated screen first before running your Effect](#). If your Effect is doing something visual (for example, positioning a tooltip), and the delay is noticeable (for example, it flickers), replace [useEffect](#) with [useLayoutEffect](#).
- If your Effect is caused by an interaction (like a click), [React may run your Effect before the browser paints the updated screen](#). This ensures that the result of the Effect can be observed by the event system. Usually, this works as expected. However, if you must defer the work until after paint, such as an `alert()`, you can use `setTimeout`. See [reactwg/react-18/128](#) for more information.

ON THIS PAGE

Overview

Reference

[useEffect\(setup, dependencies?\)](#)

Usage

Connecting to an external system

Wrapping Effects in custom Hooks

Controlling a non-React widget

Fetching data with Effects

Specifying reactive dependencies

Updating state based on previous state from an Effect

Removing unnecessary object dependencies

Removing unnecessary function dependencies



# CQRS pattern

<https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs>

# MediatR

<https://www.nuget.org/packages/mediatr/>

```
<PackageReference Include="MediatR" Version="12.4.1" />
```

# List.cs

```
namespace Application.Activities
{
    2 references
    public class List
    {
        3 references
        public class Query : IRequest<List<Activity>> { }

        2 references
        public class Handler : IRequestHandler<Query, List<Activity>>
        {
            private readonly DataContext _context;
            0 references
            public Handler(DataContext context)
            {
                _context = context;
            }
            0 references
            public async Task<List<Activity>> Handle(Query request, CancellationToken cancellationToken)
            {
                return await _context.Activities.ToListAsync();
            }
        }
    }
}
```

# Add/Register MediatR Service

```
builder.Services.AddMediatR(cfg =>  
    cfg.RegisterServicesFromAssembly(typeof(List.Handler).Assembly));
```

# Change ActivitiesController and Test

```
namespace WebApi.Controllers
{
    1 reference
    public class ActivitiesController : BaseApiController
    {
        private readonly IMediator _mediator;
        0 references
        public ActivitiesController(IMediator mediator)
        {
            _mediator = mediator;
        }

        [HttpGet]
        0 references
        public async Task<ActionResult<List<Activity>>> GetActivities()
        {
            return await _mediator.Send(new List.Query());
        }
    }
}
```

# Change BaseApiController

```
namespace WebApi.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class BaseApiController : ControllerBase
    {
        private IMediator _mediator;
        3 references
        protected IMediator Mediator
        {
            get
            {
                return _mediator ??= HttpContext.RequestServices.GetService<IMediator>();
            }
        }
    }
}
```

# Change Activities Controller and Test

```
namespace WebApi.Controllers
{
    0 references
    public class ActivitiesController : BaseApiController
    {
        /*private readonly IMediator _mediator;
        public ActivitiesController(IMediator mediator)
        {
            _mediator = mediator;
        }*/
        [HttpGet]
        0 references
        public async Task<ActionResult<List<Activity>>> GetActivities() {
            return await Mediator.Send(new List.Query());
        }
    }
}
```

# Details.cs

```
namespace Application.Activities
{
    1 reference
    public class Details
    {
        3 references
        public class Query : IRequest<Activity> {
            2 references
            public int Id { get; set; }
        }

        1 reference
        public class Handler : IRequestHandler<Query, Activity>
        {
            private readonly DataContext _context;
            0 references
            public Handler(DataContext context)
            {
                _context = context;
            }
            0 references
            public async Task<Activity> Handle(Query request, CancellationToken cancellationToken)
            {
                return await _context.Activities.FindAsync(request.Id);
            }
        }
    }
}
```

# Activities Controller with HttpGet by ID

```
[HttpGet("{id}")]
0 references
public async Task<ActionResult<Activity>> GetActivity(int id) {
    return await Mediator.Send(new Details.Query { Id = id });
}
```

# Create.cs

```
namespace Application.Activities
{
    1 reference
    public class Create
    {
        3 references
        public class Command : IRequest {
            2 references
            public Activity Activity { get; set; }
        }

        1 reference
        public class Handler : IRequestHandler<Command>
        {
            private readonly DataContext _context;
            0 references
            public Handler(DataContext context)
            {
                _context = context;
            }
            0 references
            public async Task Handle(Command request, CancellationToken cancellationToken)
            {
                await _context.Activities.AddAsync(request.Activity);
                _context.SaveChanges();
                return;
            }
        }
    }
}
```

# Activities Controller with HttpPost

```
[HttpPost]  
0 references  
public async Task<IActionResult> CreateActivity(Activity activity) {  
    await Mediator.Send(new Create.Command{ Activity = activity });  
    return Ok();  
}
```

# Edit.cs

```
namespace Application.Activities
{
    1 reference
    public class Edit
    {
        3 references
        public class Command : IRequest
        {
            3 references
            public Activity Activity { get; set; }
        }

        1 reference
        public class Handler : IRequestHandler<Command>
        {
            private readonly DataContext _context;

            0 references
            public Handler(DataContext context, IMapper mapper)
            {
                _context = context;
            }

            0 references
            public async Task Handle(Command request, CancellationToken cancellationToken)
            {
                var activity = await _context.Activities.FindAsync(request.Activity.Id);
                activity.Title = request.Activity.Title ?? activity.Title;
                await _context.SaveChangesAsync();
            }
        }
    }
}
```

# Adding AutoMapper

## In Application Project

```
<PackageReference  
Include="AutoMapper.Extensions.  
Microsoft.DependencyInjection"  
Version="12.0.0" />
```

```
namespace Application.Core  
{  
    2 references  
    public class MappingProfiles : Profile  
    {  
        0 references  
        public MappingProfiles()  
        {  
            CreateMap<Activity, Activity>();  
        }  
    }  
}
```

```
builder.Services.AddMediatR(cfg => cfg.RegisterServicesFromAssembly(typeof(List.Handler).Assembly));  
builder.Services.AddAutoMapper(typeof(MappingProfiles).Assembly);
```

# Edit.cs with AutoMapper

```
0 references
public async Task Handle(Command request, CancellationToken cancellationToken)
{
    var activity = await _context.Activities.FindAsync(request.Activity.Id);
    //activity.Title = request.Activity.Title ?? activity.Title;

    // AutoMapper - AutoMapper.Extensions.Microsoft.DependencyInjection
    _mapper.Map(request.Activity, activity);

    await _context.SaveChangesAsync();
}
```

# Activities Controller with HttpPut

```
[HttpPut("{id}")]
0 references
public async Task<IActionResult> EditActivity(int id, Activity activity)
{
    activity.Id = id;
    await Mediator.Send(new Edit.Command { Activity = activity });
    return Ok();
}
```

# Delete.cs

```
namespace Application.Activities
{
    1 reference
    public class Delete
    {
        3 references
        public class Command : IRequest
        {
            2 references
            public int Id { get; set; }
        }

        1 reference
        public class Handler : IRequestHandler<Command>
        {
            private readonly DataContext _context;
            0 references
            public Handler(DataContext context)
            {
                _context = context;
            }
            0 references
            public async Task Handle(Command request, CancellationToken cancellationToken)
            {
                var activity = await _context.Activities.FindAsync(request.Id);
                _context.Remove(activity);
                await _context.SaveChangesAsync();
            }
        }
    }
}
```

# Activities Controller with HttpDelete

```
[HttpDelete("{id}")]
0 references
public async Task<IActionResult> DeleteActivity(int id)
{
    await Mediator.Send(new Delete.Command { Id = id });
    return Ok();
}
```

# Startup class housekeeping

Program.cs

```
using WebApi.Extensions;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
builder.Services.AddApplicationServices(builder.Configuration);

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

```
namespace WebApi.Extensions
{
    0 references
    public static class ApplicationServiceExtensions
    {
        1 reference
        public static IServiceCollection AddApplicationServices(this IServiceCollection services,
            IConfiguration config)
        {
            // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
            services.AddEndpointsApiExplorer();
            services.AddSwaggerGen();

            services.AddDbContext<DataContext>(opt => opt.UseSqlServer(
                config.GetConnectionString("DefaultConnection"))
            );

            services.AddCors(opt => {
                opt.AddPolicy("CorsPolicy", policy =>
                {
                    policy.AllowAnyHeader().AllowAnyMethod().WithOrigins("http://localhost:3000");
                });
            });

            services.AddMediatR(cfg => cfg.RegisterServicesFromAssembly(typeof(List.Handler).Assembly));
            services.AddAutoMapper(typeof(MappingProfiles).Assembly);

            return services;
        }
    }
}
```

# Cancellation tokens ( List.cs )

```
2 references
public class Handler : IRequestHandler<Query, List<Activity>>
{
    private readonly DataContext _context;
    private readonly ILogger<List> _logger;
0 references
    public Handler(DataContext context, ILogger<List> logger)
    {
        _logger = logger;
        _context = context;
    }
0 references
    public async Task<List<Activity>> Handle(Query request, CancellationToken cancellationToken)
    {
        try
        {
            for (var i = 0; i < 10; i++)
            {
                cancellationToken.ThrowIfCancellationRequested();
                await Task.Delay(1000, cancellationToken);
                _logger.LogInformation($"Task {i} has completed");
            }
        }
        catch (System.Exception)
        {
            _logger.LogInformation("Task ws cancelled");
        }
    }
    return await _context.Activities.ToListAsync();
}
```

# Activities Controller Pass CancellationToken

```
[HttpGet]  
0 references  
public async Task<ActionResult<List<Activity>>> GetActivities(CancellationToken ct) {  
    return await Mediator.Send(new List.Query(), ct);  
}
```

After Testing remove Code we have added for testing purpose in Edit.cs.

We don't have any request currently which will take much time to response.

# Creating a CRUD application in React



# Folder Structure

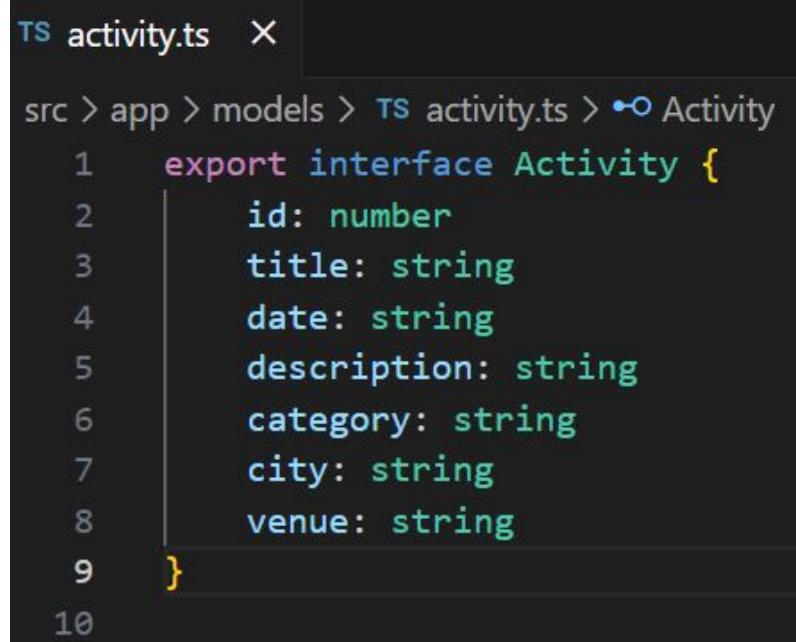
<https://legacy.reactjs.org/docs/faq-structure.html>

```
✓ REACTIVITIES
  > .vscode
  > node_modules
  > public
  ✓ src
    ✓ app
      ✓ layout
        App.tsx
        # styles.css
      > features
        main.tsx
        TS vite-env.d.ts
        .gitignore
        eslint.config.js
        index.html
        {} package-lock.json
        {} package.json
        i README.md
        {} tsconfig.app.json
        TS tsconfig.json
        {} tsconfig.node.json
        ⚡ vite.config.ts
```

# Adding Activity Interface

<https://transform.tools/json-to-typescript>

<https://blog.logrocket.com/types-vs-interfaces-typescript/>



The screenshot shows a code editor window with a dark theme. The file is named 'activity.ts' and is located at 'src > app > models > activity.ts'. The code defines an interface 'Activity' with properties: id (number), title (string), date (string), description (string), category (string), city (string), and venue (string). The code is numbered from 1 to 10. A yellow cursor is positioned at the end of the closing brace on line 9.

```
TS activity.ts ×
src > app > models > TS activity.ts > Activity
1  export interface Activity {
2    id: number
3    title: string
4    date: string
5    description: string
6    category: string
7    city: string
8    venue: string
9  }
10
```

# App.tsx update using Activity interface

```
function App() {  
  const [activities, setActivities] = useState<Activity[]>([]);  
  
  useEffect(() => {  
    axios.get<Activity[]>("https://localhost:5000/api/Activities")  
      .then(response => setActivities(response.data));  
  }, [])  
  
  return (  
    <>  
      <Header as="h2" icon={"users"} content="Activities"></Header>  
      <List>  
        {activities.map((activity: Activity) => (  
          <List.Item key={activity.id}>{activity.title}</List.Item>  
        ))}  
      </List>  
    </>  
  )  
}
```

# Adding Nav bar

<https://react.semantic-ui.com/collections/menu/>

```
src > app > layout > ⚘ NavBar.tsx > [!] NavBar
  1  import { Container, Menu } from "semantic-ui-react"
  2
  3  const NavBar = () => {
  4    return (
  5      <Menu inverted fixed="top">
  6        <Container>
  7          <Menu.Item header>
  8            
  9            Reactivities
 10        </Menu.Item>
 11        <Menu.Item name="Activities" />
 12        <Menu.Item name="Create Activity">
 13        </Menu.Item>
 14      </Container>
 15    </Menu>
 16  )
 17}
 18
 19 export default NavBar
```

# Create Activity List with Navbar

localhost:3000

Reactivities Activities Create Activity

**Past Activity 1**  
2020-09-30T13:44:08.311074  
Activity 2 months ago  
London, Pub

drinks View

**Past Activity 2**  
2020-10-30T13:44:08.321595  
Activity 1 month ago  
Paris, Louvre

culture View

**Future Activity 1**  
2020-12-30T13:44:08.321598  
Activity 1 month in future  
London, Natural History Museum

culture View

**Future Activity 2**  
2021-01-30T13:44:08.321598  
Activity 2 months in future  
London, O2 Arena

music View

# Creating Activity Details

```
export default function ActivityDashboard({activities}: Props) {
  return (
    <Grid>
      <Grid.Column width='10'>
        <ActivityList activities={activities} />
      </Grid.Column>
      <Grid.Column width='6'>
        {activities[0] &&
          <ActivityDetails activity={activities[0]} />}
      </Grid.Column>
    </Grid>
  )
}
```

The screenshot shows a web application running on localhost:3000. The top navigation bar includes a logo, 'Reactivities', 'Activities', and a green 'Create Activity' button. Below the navigation, there are three activity cards:

- Past Activity 1**  
2020-09-30T13:44:08.311074  
Activity 2 months ago  
London, Pub  
drinks View
- Past Activity 2**  
2020-10-30T13:44:08.321595  
Activity 1 month ago  
Paris, Louvre  
culture View
- Future Activity 1**  
2020-12-30T13:44:08.321598  
Activity 1 month in future  
London, Natural History Museum  
culture View

A modal window is open on the right side, showing a preview of 'Past Activity 1'. It includes a close button, a photo thumbnail of people at a pub, and a preview of the activity details.

# Creating Activity Form

```
src
  app
  features
    activities
      dashboard
        ActivityDashboard.tsx
        ActivityList.tsx
      details
        ActivityDetails.tsx
      form
        ActivityForm.tsx
        index.tsx
        react-app-env.ts
        reportWebVitals.ts
```

The screenshot displays a React application interface. At the top, there is a navigation bar with tabs: 'Reactivities' (highlighted), 'Activities', and 'Create Activity'. Below the navigation, there are four activity cards:

- Past Activity 2**  
2020-10-30T13:44:08.321595  
Activity 1 month ago  
Paris, Louvre  
culture View
- Future Activity 1**  
2020-12-30T13:44:08.321598  
Activity 1 month in future  
London, Natural History Museum  
culture View
- Future Activity 2**  
2021-01-30T13:44:08.321598  
Activity 2 months in future  
London, O2 Arena  
music View
- Future Activity 3**  
2021-02-28T13:44:08.321598  
Activity 3 months in future  
London, Another pub  
drinks View

On the right side, a modal window is open for 'Past Activity 1'. The modal has fields for 'sdssd' (City), 'sdsd' (Venue), and 'd' (Category). It includes 'Edit' and 'Cancel' buttons at the top and 'Submit' and 'Cancel' buttons at the bottom.

# Selecting Activity To View

```
function App() {
  const [activities, setActivities] = useState<Activity[]>([]);
  const [selectedActivity, setSelectedActivity] = useState<Activity | undefined>(undefined);

  useEffect(() => {
    axios.get<Activity[]>('http://localhost:5000/api/activities').then(response => {
      setActivities(response.data);
    })
  }, [])

  function handleSelectedActivity(id: string) {
    setSelectedActivity(activities.find(x => x.id === id));
  }

  function handleCancelSelectActivity() {
    setSelectedActivity(undefined);
  }
}
```

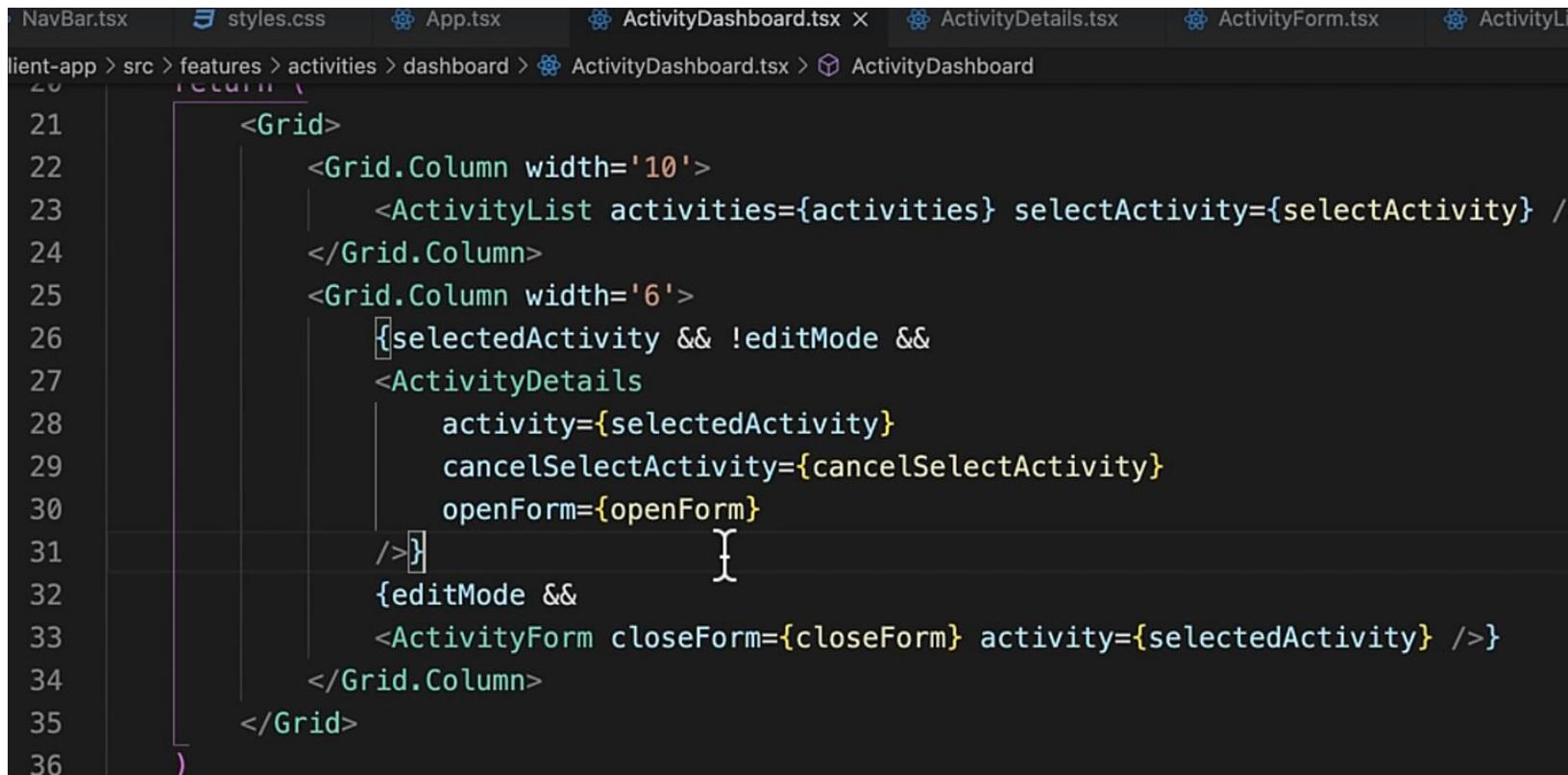
# Display Create / Edit Form

```
function handleFormOpen(id?: string) {
  id ? handleSelectActivity(id) : handleCancelSelectActivity();
  setEditMode(true);
}

function handleFormClose() {
  setEditMode(false);
}

return (
  <>
  <NavBar />
  <Container style={{marginTop: '7em'}}>
    <ActivityDashboard
      activities={activities}
      selectedActivity={selectedActivity}
      selectActivity={handleSelectActivity}
      cancelSelectActivity={handleCancelSelectActivity}
      editMode={editMode}
      openForm={handleFormOpen}
      closeForm={handleFormClose}
    />
  </Container>
)
```

# Toggle Activity Details when Activity Form Showing



The screenshot shows a code editor with a dark theme. The file being edited is `ActivityDashboard.tsx`. The code implements a dashboard layout with two columns. The left column has a width of 10 units and contains an `ActivityList` component. The right column has a width of 6 units and contains logic to toggle between displaying `ActivityDetails` and `ActivityForm` based on the `selectedActivity` state and `editMode` prop.

```
NavBar.tsx styles.css App.tsx ActivityDashboard.tsx X ActivityDetails.tsx ActivityForm.tsx ActivityLi
lient-app > src > features > activities > dashboard > ActivityDashboard.tsx > ActivityDashboard
21     <Grid>
22         <Grid.Column width='10'>
23             <ActivityList activities={activities} selectActivity={selectActivity} />
24         </Grid.Column>
25         <Grid.Column width='6'>
26             {selectedActivity && !editMode &&
27             <ActivityDetails
28                 activity={selectedActivity}
29                 cancelSelectActivity={cancelSelectActivity}
30                 openForm={openForm}>
31             />}>
32             {editMode &&
33             <ActivityForm closeForm={closeForm} activity={selectedActivity} />}>
34         </Grid.Column>
35     </Grid>
36 }
```

# Form input Handling in React

```
const [activity, setActivity] = useState(initialState);

function handleSubmit() {
  console.log(activity);
}

function handleInputChange(event: ChangeEvent<HTMLInputElement>) {
  const {name, value} = event.target;
  setActivity({...activity, [name]: value})
}
```

```
    autoComplete='off'>
    title' value={activity.title} name='title' onChange={handleInputChange} />
    ='Description' value={activity.description} name='description' onChange={handleInputChange} />
    category' value={activity.category} name='category' onChange={handleInputChange} />
    date' value={activity.date} name='date' onChange={handleInputChange} />
    city' value={activity.city} name='city' onChange={handleInputChange}/>
    venue' value={activity.venue} name='venue' onChange={handleInputChange}/>
    <input type='submit' content='Submit' />
  </div>
  <button floated='right' type='button' content='Cancel' />
```

# Handle Create and Edit Submission

```
function handleCreateOrEditActivity(activity: Activity) {  
    activity.id  
    ? setActivities([...activities.filter(x => x.id !== activity.id), activity])  
    : setActivities([...activities, activity]);  
    setEditMode(false);  
    setSelectedActivity(activity);  
}
```

# Using Guid for activity id

The screenshot shows the npmjs.com package page for 'uuid'. The page header includes the URL 'npmjs.com/package/uuid', a red navigation bar with 'Sponsorships', 'Pricing', and 'Documentation' links, and a search bar with the placeholder 'Search packages'. Below the header, the package details are shown: 'uuid' (version 11.0.5, published a month ago, public). There are three tabs: 'Readme' (selected), 'Code' (Beta), and '0 Dependencies'. The 'Code' tab displays a snippet of TypeScript code for handling activity creation or editing:

```
function handleCreateOrEditActivity(activity: Activity) {
  activity.id
    ? setActivities([...activities.filter(x => x.id !== activity.id), activity])
    : setActivities([...activities, {...activity, id: uuid()}]);
  setEditMode(false);
  setSelectedActivity(activity);
}
```

Below the code, a note states: 'For the creation of [RFC9562](#) (formerly [RFC4122](#))'.

# Deleting an Activity

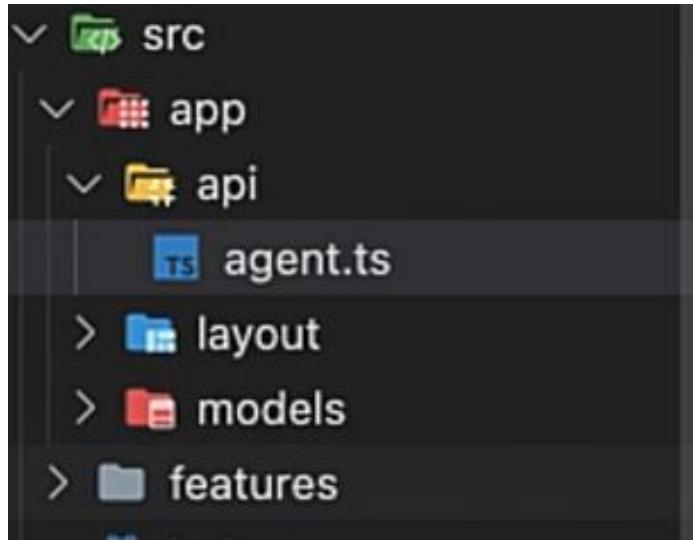
```
function handleDeleteActivity(id: string) {  
  setActivities([...activities.filter(x => x.id !== id)])  
}
```

```
<ActivityDashboard  
  activities={activities}  
  selectedActivity={selectedActivity}  
  selectActivity={handleSelectActivity}  
  cancelSelectActivity={handleCancelSelectActivity}  
  editMode={editMode}  
  openForm={handleFormOpen}  
  closeForm={handleFormClose}  
  createOrEdit={handleCreateOrEditActivity}  
  deleteActivity={handleDeleteActivity}  
/>
```

# Axios



# Setting up Axios



```
import axios, { AxiosResponse } from 'axios';

axios.defaults.baseURL = 'http://localhost:5000/api';

const responseBody = (response: AxiosResponse) => response.data;

const requests = {
  get: (url: string) => axios.get(url).then(responseBody),
  post: (url: string, body: {}) => axios.post(url, body).then(responseBody),
  put: (url: string, body: {}) => axios.put(url, body).then(responseBody),
  del: (url: string) => axios.delete(url).then(responseBody),
}

const Activities = {
  list: () => requests.get('/activities')
}

useEffect(() => {
  agent.Activities.list().then(response => {
    setActivities(response);
  })
}, [])

const agent = [
  Activities
]

export default agent;
```

# Axios types

```
const responseBody = <T> (response: AxiosResponse<T>) => response.data;

const requests = {
  get: <T> (url: string) => axios.get<T>(url).then(responseBody),
  post: <T> (url: string, body: {}) => axios.post<T>(url, body).then(responseBody),
  put: <T> (url: string, body: {}) => axios.put<T>(url, body).then(responseBody),
  del: <T> (url: string) => axios.delete<T>(url).then(responseBody),
}

const Activities = {
  list: () => requests.get<Activity[]>('/activities')
}

const agent = {
  Activities
}
```

# Date Handle

```
<Form onSubmit={handleSubmit} autoComplete='off'>  
  <Form.Input placeholder='Title' value={activity.title} name='title' />  
  <Form.TextArea placeholder='Description' value={activity.description} name='description' />  
  <Form.Input placeholder='Category' value={activity.category} name='category' />  
  <Form.Input type='date' placeholder='Date' value={activity.date} name='date' />  
  <Form.Input placeholder='City' value={activity.city} name='city' />  
  <Form.Input placeholder='Venue' value={activity.venue} name='venue' />  
  <Button floated='right' positive type='submit' content='Submit' />  
  <Button onClick={closeForm} floated='right' type='button' content='Close' />  
</Form>
```

```
useEffect(() => {  
  agent.Activities.list().then(response => {  
    let activities: Activity[] = [];  
    response.forEach(activity => {  
      activity.date = activity.date.split('T')[0];  
      activities.push(activity);  
    })  
    setActivities(activities);  
  })  
}, [])
```

Past Activity 2

Activity 1 month ago

culture

30/10/2020

Paris

Louvre

Cancel

Submit

# Adding Loading Indicator

```
const sleep = (delay: number) => {
  return new Promise((resolve) => {
    setTimeout(resolve, delay)
  })
}

axios.defaults.baseURL = 'http://localhost:5000/api';

axios.interceptors.response.use(async response => {
  try {
    await sleep(1000);
    return response;
  } catch (error) {
    console.log(error);
    return await Promise.reject(error);
  }
})
```

```
src > app > layout > LoadingComponent.tsx > +o Props > ↗ content
import { Dimmer, Loader } from 'semantic-ui-react';

interface Props {
  inverted?: boolean;
  content?: string;
}

export default function LoadingComponent({inverted = true, content = 'Loading...'}: Props) {
  return (
    <Dimmer active={true} inverted={inverted}>
      <Loader content={content} />
    </Dimmer>
  )
}
```

```
if (loading) return <LoadingComponent content='Loading app' />

return (
  <>
    <NavBar openForm={handleFormOpen} />
    <Container style={{marginTop: '7em'}}>
```

# Posting Data to Server

```
const Activities = {
  list: () => requests.get<Activity[]>('/activities'),
  details: (id: string) => requests.get<Activity>(`/activities/${id}`),
  create: (activity: Activity) => axios.post<void>('/activities', activity),
  update: (activity: Activity) => axios.put<void>(`/activities/${activity.id}`, activity),
  delete: (id: string) => axios.delete<void>(`/activities/${id}`) }
```

**Use `requests.post`, `requests.put` and `requests.delete` instead of `axios`**

```
function handleCreateOrEditActivity(activity: Activity) {
  setSubmitting(true);
  if (activity.id) {
    agent.Activities.update(activity).then(() => {
      setActivities([...activities.filter(x => x.id !== activity.id), activity])
      setSelectedActivity(activity);
      setEditMode(false);
      setSubmitting(false);
    })
  } else {
    activity.id = uuid();
    agent.Activities.create(activity).then(() => {
      setActivities([...activities, activity])
      setSelectedActivity(activity);
      setEditMode(false);
      setSubmitting(false);
    })
  }
}
```

# Deleting Activity on Server

```
function handleDeleteActivity(id: string) {
  setSubmitting(true);
  agent.Activities.delete(id).then(() => {
    setActivities([...activities.filter(x => x.id !== id)]);
    setSubmitting(false);
  })
}

<Item.Extra>
  <Button onClick={() => selectActivity(activity.id)} floated='right' name={activity.id} loading={submitting && target === activity.id} onClick={(e) => handleActivityDelete(e, activity.id)} floated='right' content='Delete' color='red' />
  <Label basic content={activity.category} />
</Item.Extra>
```

<https://mobx.js.org/react-integration.html>

# MobX



# MobX Core functions

- Observables
- Actions
- Computed properties
- Reactions
- AutoRun



# MobX Observable

```
1 import { makeObservable, observable } from "mobx";
2 import { createContext } from "react";
3
4 class DemoStore {
5     firstName = 'Bob';
6     lastName = 'Smith';
7
8     constructor() {
9         makeObservable(this, {
10             firstName: observable,
11             lastName: observable
12         })
13     }
14 }
15
16 export default createContext(new DemoStore());
```

# MobX Action

```
1 import { action, makeObservable, observable } from "mobx";
2 import { createContext } from "react";
3
4 class DemoStore {
5     firstName = 'Bob';
6     lastName = 'Smith';
7
8     constructor() {
9         makeObservable(this, {
10             firstName: observable,
11             lastName: observable,
12             setFirstName: action
13         })
14     }
15
16     setFirstName = (name: string) => {
17         this.firstName = name;
18     }
19 }
20
21 export default createContext(new DemoStore());
```

# MobX Computed

```
1 import { action, computed, makeObservable, observable } from "mobx";
2 import { createContext } from "react";
3
4 class DemoStore {
5   firstName = 'Bob';
6   lastName = 'Smith';
7
8   constructor() {
9     makeObservable(this, {
10       firstName: observable,
11       lastName: observable,
12       setFirstName: action,
13       fullName: computed
14     })
15   }
16
17   get fullName() {
18     return this.firstName + ' ' + this.lastName;
19   }
20
21   // omitted
```

# MakeAutoObservable

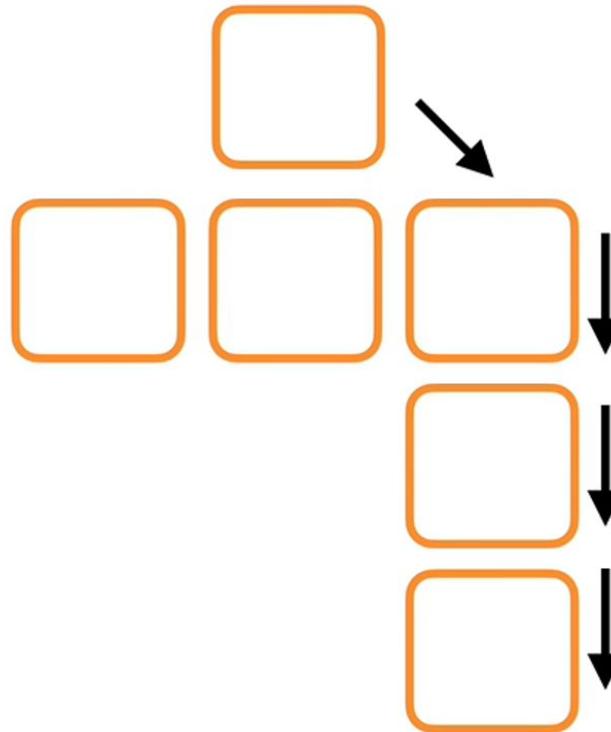
```
1 import { makeAutoObservable } from "mobx";
2 import { createContext } from "react";
3
4 class DemoStore {
5   firstName = 'Bob';
6   lastName = 'Smith';
7
8   constructor() {
9     makeAutoObservable(this)
10 }
11
12 get fullName() {
13   return this.firstName + ' ' + this.lastName;
14 }
15
16 setFirstName = (name: string) => {
17   this.firstName = name;
18 }
19 }
20
21 export default createContext(new DemoStore());
```

# MobX Reaction

```
1 import { makeAutoObservable, reaction} from "mobx";
2 import { createContext } from "react";
3
4 class DemoStore {
5   firstName = 'Bob';
6   lastName = 'Smith';
7
8   constructor() {
9     makeAutoObservable(this);
10
11   reaction(
12     () => this.firstName,
13     (firstName) => console.log(firstName)
14   )
15 }
16
17 setFirstName = (name: string) => {
18   this.firstName = name;
19 }
20
21 // omitted
```

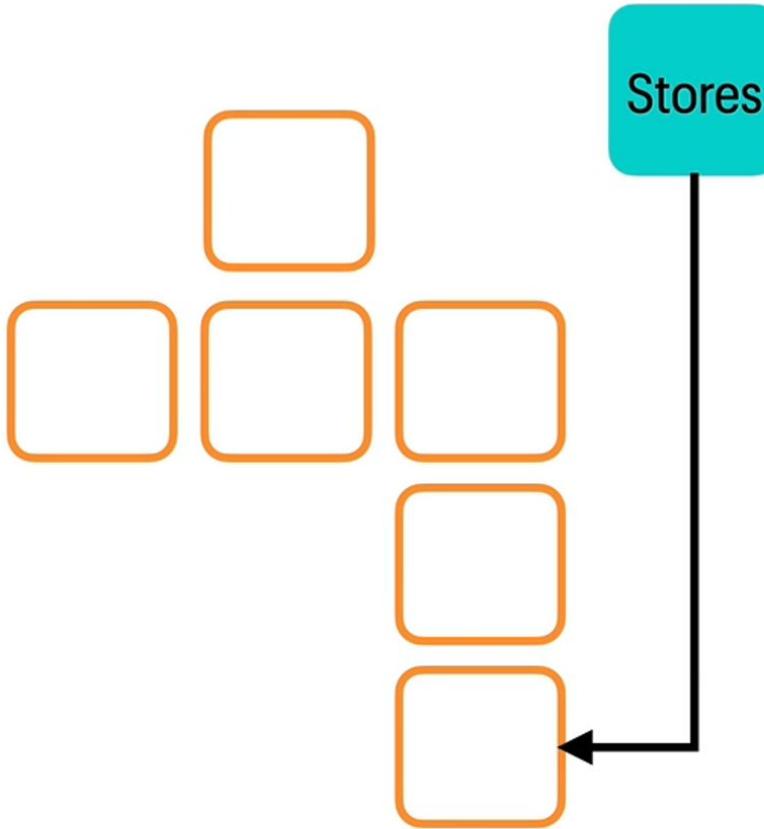
# React Context

---



# React Context

---



# React Context

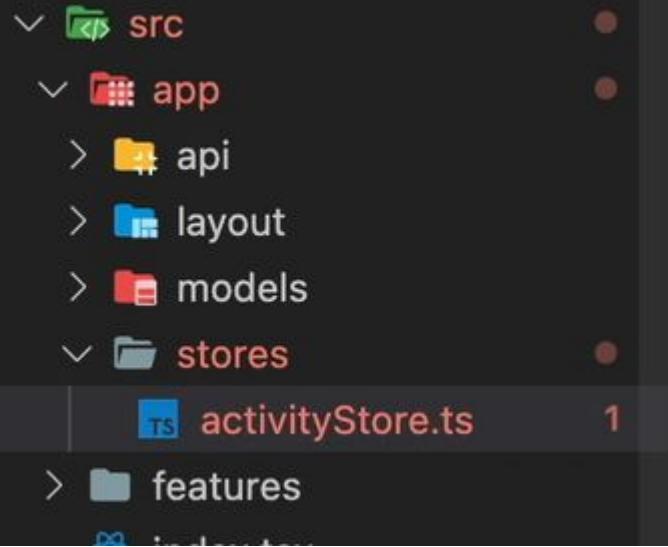
```
1 import React, {useContext} from 'react';
2 import DemoStore from '../app/demoStore';
3
4 export default function Demo() {
5     const demoStore = useContext(DemoStore);
6     const {fullName} = demoStore;
7
8     return (
9         <div>
10            <h1>Hello {fullName}</h1>
11        </div>
12    )
13 }
```

# MobX React Lite

```
1 import { observer } from 'mobx-react-lite';
2 import React, {useContext} from 'react';
3 import DemoStore from '../app/demoStore';
4
5 export default observer (function Demo() {
6     const demoStore = useContext(DemoStore);
7     const {fullName} = demoStore;
8
9     return (
10         <div>
11             <h1>Hello {fullName}</h1>
12         </div>
13     )
14 })
```

# Setting up Mobx - ActivityStore

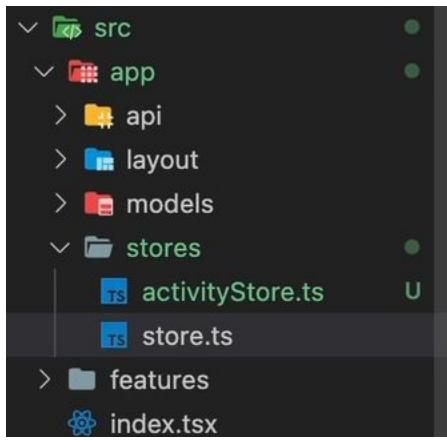
```
npm install mobx mobx-react-lite
```



```
import { makeObservable, observable } from "mobx";
export default class ActivityStore {
  title = 'Hello from MobX!';

  constructor() {
    makeObservable(this, [
      title: observable
    ])
  }
}
```

# Setting up Mobx - store.ts



```
> src > app > stores > ts store.ts > ts store
import ActivityStore from "./activityStore";
import {createContext, useContext} from "react";

interface Store {
    activityStore: ActivityStore
}

export const store: Store = {
    activityStore: new ActivityStore()
}

export const StoreContext = createContext(store);

export function useStore() {
    return useContext(StoreContext);
}
```

# Setting up Mobx - main.tsx & app.tsx

```
ReactDOM.createRoot(document.getElementById('root')!).render(  
  <React.StrictMode>  
    <StoreContext.Provider value={store}>  
      <App />  
    </StoreContext.Provider>  
  </React.StrictMode>,  
)
```

```
function App() {  
  const {activityStore} = useStore();  
  
  <Container style={{marginTop: '7em'}}>  
    <h2>{activityStore.title}</h2>
```

# MobX Actions

```
export default class ActivityStore {  
    title = 'Hello from MobX!';  
  
    constructor() {  
        makeObservable(this, {  
            title: observable,  
            setTitle: action  
        })  
    }  
  
    setTitle = () => {  
        this.title = this.title + '!';  
    }  
}
```

```
<Container style={{marginTop: '7em'}}>  
    <h2>{activityStore.title}</h2>  
    <Button content='Add exclamation!' positive onClick={activityStore.setTitle} />  
    <ActivityDashboard  
        title={activityStore.title}  
    </ActivityDashboard>  
</Container>
```

```
export default observer(App);
```

```
export default class ActivityStore {  
    title = 'Hello from MobX!';  
  
    constructor() {  
        makeAutoObservable(this)  
    }  
  
    setTitle = () => {  
        this.title = this.title + '!';  
    }  
}
```

# Refactoring the app to use MobX

```
export default class ActivityStore {
    activities: Activity[] = [];
    selectedActivity: Activity | null = null;
    editMode = false;
    loading = false;
    loadingInitial = false;

    constructor() {
        makeAutoObservable(this)
    }

    loadActivities = async () => {
        this.loadingInitial = true;
        try {
            const activities = await agent.Activities.list();
            activities.forEach(activity => {
                activity.date = activity.date.split('T')[0];
                this.activities.push(activity);
            })
            this.loadingInitial = false;
        } catch (error) {
            console.log(error);
            this.loadingInitial = false;
        }
    }
}
```

```
useEffect(() => {
    activityStore.loadActivities();
}, [activityStore])
```

```
if (!activityStore.loadingInitial) return <LoadingComponent content='Loading app' />
```

# MobX strict mode

<https://mobx.js.org/actions.html#asynchronous-actions>

```
try {
  const activities = await agent.Activities.list();
  runInAction(() => {
    activities.forEach(activity => {
      activity.date = activity.date.split('T')[0];
      this.activities.push(activity);
    })
    this.loadingInitial = false;
  })
} catch (error) {
  console.log(error);
  runInAction(() => {
    this.loadingInitial = false;
  })
}
```

```
loadActivities = async () => {
  this.setLoadingInitial(true);
  try {
    const activities = await agent.Activities.list();
    activities.forEach(activity => {
      activity.date = activity.date.split('T')[0];
      this.activities.push(activity);
    })
    this.setLoadingInitial(false);
  } catch (error) {
    console.log(error);
    this.setLoadingInitial(false);
  }
}

setLoadingInitial = (state: boolean) => {
  this.loadingInitial = state;
}
```

# Selecting an Activity using MobX

```
selectActivity = (id: string) => {
  this.selectedActivity = this.activities.find(a => a.id === id);
}

cancelSelectedActivity = () => [
  this.selectedActivity = undefined;
]

openForm = (id?: string) => {
  id ? this.selectActivity(id) : this.cancelSelectedActivity();
  this.editMode = true;
}

closeForm = () => {
  this.editMode = false;
}
```

```
export default function NavBar() {
  const {activityStore} = useStore();

  return (
    <Menu inverted fixed='top'>
      <Container>
        <Menu.Item header>
          
          Reactivities
        </Menu.Item>
        <Menu.Item name='Activities' />
        <Menu.Item>
          <Button onClick={() => activityStore.openForm()} positive content='Create Acti' />
        </Menu.Item>
    </Container>
  );
}

export default function ActivityDashboard({activities, deleteActivity,
  createOrEdit, submitting}: Props) {

  const {activityStore} = useStore();
  const {selectedActivity, editMode} = activityStore;
```

# Creating an Activity using MobX

```
createActivity = async (activity: Activity) => {
    this.loading = true;
    activity.id = uuid();
    try {
        await agent.Activities.create(activity);
        runInAction(() => {
            this.activities.push(activity);
            this.selectedActivity = activity;
            this.editMode = false;
            this.loading = false;
        })
    } catch (error) {
        console.log(error);
        runInAction(() => {
            this.loading = false;
        })
    }
}
```

```
updateActivity = async (activity: Activity) => {
    this.loading = true;
    try {
        await agent.Activities.update(activity);
        runInAction(() => {
            this.activities = [...this.activities.filter(a => a.id !== activity.id), activity];
            this.selectedActivity = activity;
            this.editMode = false;
            this.loading = false;
        })
    } catch (error) {
        console.log(error);
    }
}
```

```
function handleSubmit() {
    activity.id ? updateActivity(activity) : createActivity(activity);
}
```

# Deleting an Activity using MobX

```
deleteActivity = async (id: string) => {
  this.loading = true;
  try {
    await agent.Activities.delete(id);
    runInAction(() => {
      this.activities = [...this.activities.filter(a => a.id !== id)];
      if (this.selectedActivity?.id === id) this.cancelSelectedActivity();
      this.loading = false;
    })
  } catch (error) {
    console.log(error);
    runInAction(() => {
      this.loading = false;
    })
  }
}
```

```
function handleActivityDelete(e: SyntheticEvent<HTMLButtonElement>, id: string) {
  setTarget(e.currentTarget.name);
  deleteActivity(id);
}
```

# Using javascript map object to store the Activities

The screenshot shows a browser window displaying the MDN Web Docs page for the `Map` object. The URL in the address bar is `developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map#Instance_methods`. The page title is "Map". The main content area starts with a heading "Jump to section" followed by a list of links: "Description", "Constructor", "Static properties", "Instance properties", "Instance methods", "Examples", and "Specifications". To the right of this list, there is a "Description" section with the following text:

The `Map` object holds key-value pairs and remembers the original insertion order of the keys. Any value (both objects and primitive values) may be used as either a key or a value.

## Description

A `Map` object iterates its elements in insertion order — a `for...of` loop returns an array of `[key, value]` for each iteration.

```
export default class ActivityStore {
  activities: Activity[] = [];
  activityRegistry = new Map<string, Activity>();
  selectedActivity: Activity | undefined = undefined;
```

```
const activities = await agent.Activities.list();
activities.forEach(activity => {
  activity.date = activity.date.split('T')[0];
  this.activityRegistry.set(activity.id, activity);
})
```

```
selectActivity = (id: string) => {
  this.selectedActivity = this.activityRegistry.get(id);
}
```

```
await agent.Activities.update(activity);
runInAction(() => {
  this.activityRegistry.set(activity.id, activity);
  this.selectedActivity = activity;
```

```
await agent.Activities.delete(id);
runInAction(() => {
  this.activityRegistry.delete(id);
```

```
get activitiesByDate() {
  return Array.from(this.activityRegistry.values()).sort((a, b) =>
    Date.parse(a.date) - Date.parse(b.date));
}
```

```
<Item.Group divided>
{activitiesByDate.map(activity => (
  <Item key={activity.id}>
    <Item.Content>
```

# Routing



# Why do we need a router?

---

- SPAs need routers
- We only have one page (index.html)
- Small apps we could just conditionals
- Our app will get more complex
- React Router - most popular choice

# React Router API

---

<BrowserRouter>

<Route>

<Link>

<NavLink>

<Redirect>

# React Router hooks

---

`useHistory`

`useLocation`

`useParams`

`useRouteMatch`

# History

- Keeps track of current location
- Re-renders when change detected

```
Props
  ▶ history: {...}
    action: "POP"
    ▶ block: block()
    ▶ createHref: createHref()
    ▶ go: go()
    ▶ goBack: goBack()
    ▶ goForward: goForward()
    length: 2
    ▶ listen: listen()
    ▶ location: {...}
    ▶ push: push()
    ▶ replace: replace()
  ▶ location: {...}
    hash: ""
    pathname: "/events"
    search: ""
  ▶ match: {...}
```



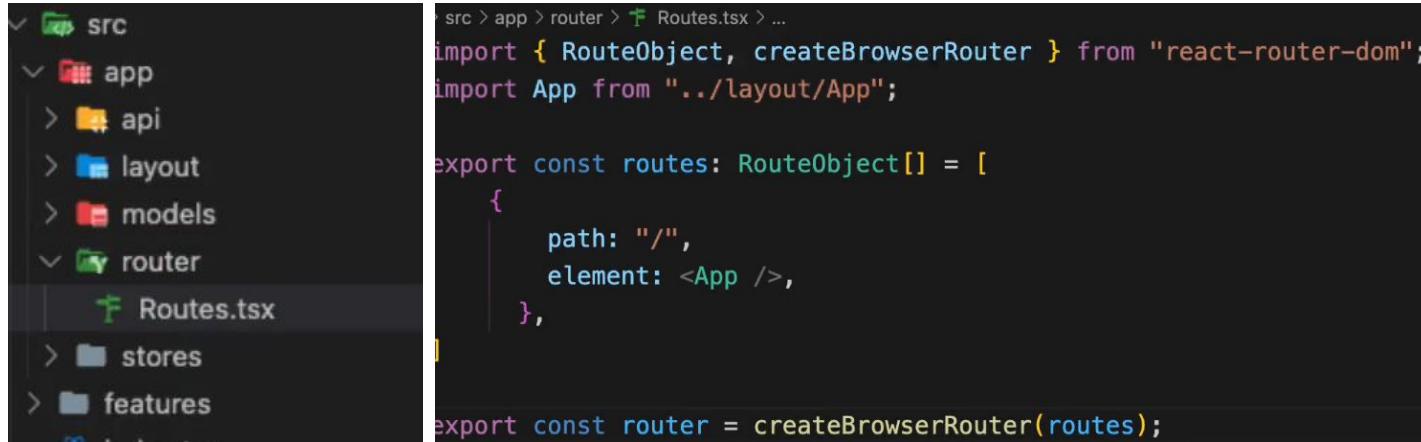
# Installing React Router

The screenshot shows the React Router website at [reactrouter.com/en/main](https://reactrouter.com/en/main). The page has a navigation bar with a logo, a search bar, and links for 'main' and 'dark mode'. On the left, there's a sidebar with sections for 'Getting Started' (Feature Overview, Tutorial, Examples, FAQs, Main Concepts), 'Upgrading' (Upgrading from v5, Migrating from @reach/router), and 'Routers' (Picking a Router). The main content area features several cards:

- What's New in 6.4?**: A card with a green icon of a bar chart. It says: "v6.4 is our most exciting release yet with new data abstractions for reads, writes, and navigation hooks to easily keep your UI in sync with your data. The new feature overview will catch you up."
- I'm New**: A card with a blue icon of a person. It says: "Start with the tutorial. It will quickly introduce you to the primary features of React Router: from creating routes, to loading and mutating data, to rendering optimistic UI."
- I'm on v5**: A card with a purple icon of a bar chart. It says: "The migration guide will help you migrate incrementally and keep shipping along the way. Or, do it all in one yolo commit! Either way, we've got you covered to start using the new features right away."
- I'm Stuck!**: A card with an orange icon of a person. It says: "Running into a problem? Chances are you're not alone! Explore common questions about React Router v6."

```
client-app git:(main) ✘ npm install react-router-dom
```

# RouterProvider



The image shows a file tree on the left and the content of `Routes.tsx` on the right.

**File Tree:**

- src
- app
- api
- layout
- models
- router
- Routes.tsx
- stores
- features

**Routes.tsx Content:**

```
src > app > router > Routes.tsx > ...
import { RouteObject, createBrowserRouter } from "react-router-dom";
import App from "../layout/App";

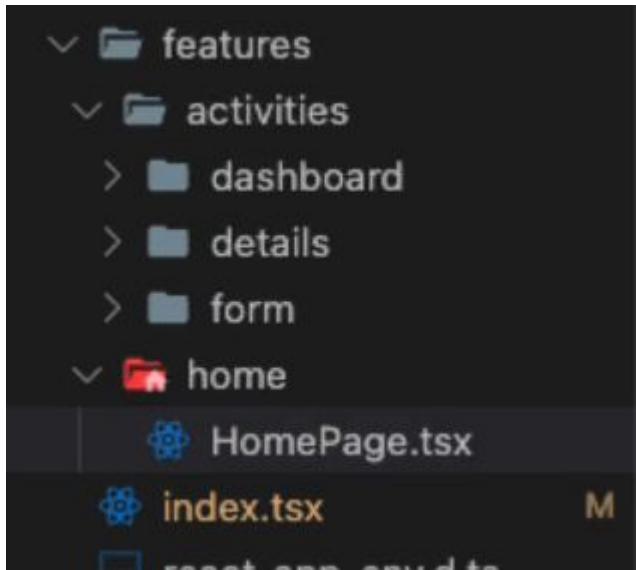
export const routes: RouteObject[] = [
  {
    path: "/",
    element: <App />,
  },
]

export const router = createBrowserRouter(routes);
```

```
import { RouterProvider } from 'react-router-dom'
import { router } from './app/router/Routes'

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <StoreContext.Provider value={store}>
      <RouterProvider router={router} />
    </StoreContext.Provider>
  </React.StrictMode>,
)
```

# Home Page



```
o > src > features > home > HomePage.tsx > HomePage > marginTop  
import { Container } from "semantic-ui-react";  
  
export default function HomePage() {  
    return (  
        <Container style={{marginTop: '7em'}}>  
            <h1>Home page</h1>  
        </Container>  
    )  
}
```

# Adding Routes

```
export const routes: RouteObject[] = [
  {
    path: '/',
    element: <App />,
    children: [
      {path: '', element: <HomePage />},
      {path: 'activities', element: <ActivityDashboard />},
      {path: 'createActivity', element: <ActivityForm />},
    ]
]
```

```
return (
  <>
  <NavBar />
  <Container style={{ marginTop: '7em' }}>
    <Outlet />
  </Container>
</>
);
```

# Adding nav links

```
<Menu inverted fixed='top'>
  <Container>
    <Menu.Item as={NavLink} to='/' header>
      <img src='/assets/logo.png' alt='logo' style={{ma
      Reactivities
    </Menu.Item>
    <Menu.Item as={NavLink} to='/activities' name='Activit
    <Menu.Item>
      <Button as={NavLink} to='/createActivity' positive
    </Menu.Item>
  </Container>
</Menu>
```

# Adding Details Link

```
export const routes: RouteObject[] = [
  {
    path: '/',
    element: <App />,
    children: [
      {path: '', element: <HomePage />},
      {path: 'activities', element: <ActivityDashboard />},
      {path: 'activities/:id', element: <ActivityDetails />},
      {path: 'createActivity', element: <ActivityForm />},
    ]
  }
]
```

```
<Item.Extra>
  <Button as={Link} to={`/activities/${activity.id}`} floated='right'>
```

# Getting an Individual Activity

```
loadActivity = async (id: string) => {
  let activity = this.getActivity(id);
  if (activity) this.selectedActivity = activity;
  else {
    this.setLoadingInitial(true);
    try {
      activity = await agent.Activities.details(id);
      this.setActivity(activity);
      this.selectedActivity = activity;
      this.setLoadingInitial(false);
    } catch (error) {
      console.log(error);
      this.setLoadingInitial(false);
    }
  }
}
```

```
private setActivity = (activity: Activity) => [
  activity.date = activity.date.split('T')[0];
  this.activityRegistry.set(activity.id, activity);
]
```

# Remove Methods

```
selectActivity = (id: string) => {
  this.selectedActivity = this.activityRegistry.get(id);
}

cancelSelectActivity = () => {
  this.selectedActivity = undefined;
}

openForm = (id?: string) => {
  id ? this.selectActivity(id) : this.cancelSelectActivity();
  this.editMode = true;
}

closeForm = () => {
  this.editMode = false;
}
```

# Using Route Parameters

```
export default observer(function ActivityDetails() {
  const {activityStore} = useStore();
  const {selectedActivity: activity, loadActivity, loadingInitial} = activityStore;
  const {id} = useParams();

  useEffect(() => {
    if (id) loadActivity(id);
  }, [id, loadActivity])

  if (loadingInitial || !activity) return <LoadingComponent />;
}
```

# Adding edit Activity Route

```
export const routes: RouteObject[] = [
  {
    path: '/',
    element: <App />,
    children: [
      {path: '', element: <HomePage />},
      {path: 'activities', element: <ActivityDashboard />},
      {path: 'activities/:id', element: <ActivityDetails />},
      {path: 'createActivity', element: <ActivityForm />},
      {path: 'manage/:id', element: <ActivityForm />},
    ],
  },
<Card.Content extra>
  <Button.Group widths='2'>
    <Button as={Link} to={`/manage/${activity.id}`} basic color='blue' content='Ed
    <Button as={Link} to='/activities' basic color='grey' content='Cancel' />
  </Button.Group>
</Card.Content>
```

# Edit Activity Form

```
loading, loadActivity, loadingInitial} = activityStore;
const {id} = useParams();

const [activity, setActivity] = useState<Activity>({
  id: '',
  title: '',
  category: '',
  description: '',
  date: '',
  city: '',
  venue: ''
});

useEffect(() => {
  if (id) loadActivity(id).then(activity => setActivity(activity!))
}, [id, loadActivity]);
```

```
loadActivity = async (id: string) => {
  let activity = this.getActivity(id);
  if (activity) {
    this.selectedActivity = activity;
    return activity;
  }
  else {
    this.setLoadingInitial(true);
    try {
      activity = await agent.Activities.details(id);
      this.setActivity(activity);
      this.selectedActivity = activity;
      this.setLoadingInitial(false);
      return activity;
    } catch (error) {
      console.log(error);
      this.setLoadingInitial(false);
    }
  }
}
```

# Adding Key to the route

beta.reactjs.org/learn/preserving-and-resetting-state#option-2-resetting-state-with-a-key

BETA

## Option 2: Resetting state with a key

API

There is also another, more generic, way to reset a component's state.

You might have seen `key`s when rendering lists. Keys aren't just for lists! You can use keys to make React distinguish between any components. By default, React uses order within the parent ("first counter", "second counter") to discern between components. But keys let you tell React that this is not just a *first* counter, or a *second* counter, but a specific counter—for example, *Taylor's* counter. This way, React will know *Taylor's* counter wherever it appears in the tree!

In this example, the two `<Counter />`s don't share state even though they appear in the same place in JSX:

App.js

```
1 import { useState } from 'react';
2
3 export default function Scoreboard() {
4   const [isPlayerA, setIsPlayerA] = useState(true);
5   return (
6     <div>
7       {isPlayerA ? (
```

```
import const routes: RouteObject[] = [
  {
    path: '/',
    element: <App />,
    children: [
      {path: '', element: <HomePage />},
      {path: 'activities', element: <ActivityDashboard />},
      {path: 'activities/:id', element: <ActivityDetails />},
      {path: 'createActivity', element: <ActivityForm key='create' />},
      {path: 'manage/:id', element: <ActivityForm key='manage' />},
    ]
  }
]
```

# Redirecting after submission

```
export default observer(function ActivityForm() {
  const {activityStore} = useStore();
  const {selectedActivity, createActivity, updateActivity, loading, loadActivity, loadingInitial} = activityStore;
  const {id} = useParams();
  const navigate = useNavigate();
```

```
function handleSubmit() {
  if (!activity.id) {
    activity.id = uuid();
    createActivity(activity).then(() => navigate(`/activities/${activity.id}`))
  } else {
    updateActivity(activity).then(() => navigate(`/activities/${activity.id}`))
  }
}
```

# Moving the Home page outside nav

```
function App() {
  const location = useLocation();

  return (
    <>
      {location.pathname === '/' ? <HomePage /> : (
        <>
          <NavBar />
          <Container style={{ marginTop: '7em' }}>
            <Outlet />
          </Container>
        </>
      )}
    <>
  
```

```
export default function HomePage() {
  return (
    <Container style={{marginTop: '7em'}}>
      <h1>Home page</h1>
      <h3>Go to <Link to='/activities'>Activities</Link></h3>
    </Container>
  }
}
```

# Grouping Activities by date

```
get groupedActivities() {
    return Object.entries(
        this.activitiesByDate.reduce((activities, activity) => {
            const date = activity.date;
            activities[date] = activities[date] ? [...activities[date], activity] : [activity];
            return activities;
        }, {} as {[key: string]: Activity[]})
    )
}

return (
    <>
        {groupedActivities.map(([group, activities]) => (
            <Fragment key={group}>
                <Header sub color='teal'>
                    {group}
                </Header>
                <Segment>
                    <Item.Group divided>
                        {activities.map(activity => (
                            <ActivityListItem key={activity.id} activity={activity} />
                        ))}
                    </Item.Group>
                </Segment>
            </Fragment>
        ))}
    </>
)
```

# Design for Activity List Item

2021-01-10



**Test mobx with redirect**  
Hosted by Bob

⌚ 2021-01-10📍 Test

Attendees go here

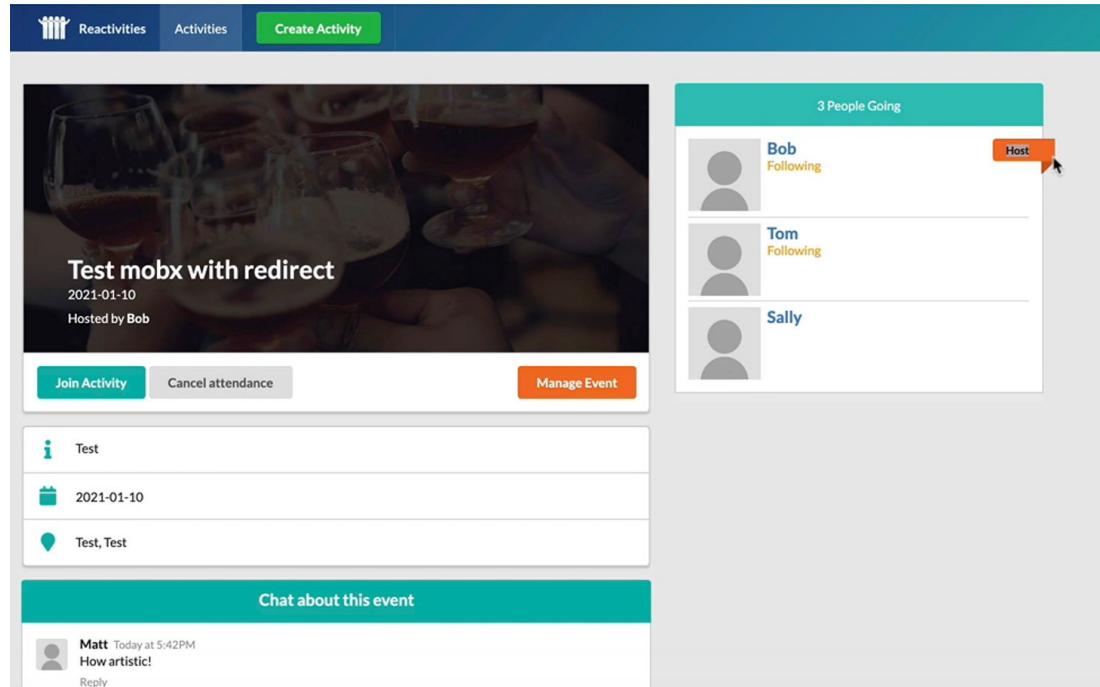
Test

**View**



# Design for Activity Details

```
return (  
  <Grid>  
    <Grid.Column width={10}>  
      <ActivityDetailedHeader />  
      <ActivityDetailedInfo />  
      <ActivityDetailedChat />  
    </Grid.Column>  
    <Grid.Column width={6}>  
      <ActivityDetailedSidebar />  
    </Grid.Column>  
  </Grid>  
)
```



Create components like  
`ActivityDetailsHeader`,  
`ActivityDetailsChat`,  
`ActivityDetailsInfo`,  
`ActivityDetailsSidebar`

# Adding Activity Filter component

2021-01-10



**Test mobx with redirect**  
Hosted by Bob

⌚ 2021-01-10 🎙 Test

Attendees go here

Test

[View](#)

2021-01-30



**Future Activity 2**  
Hosted by Bob

⌚ 2021-01-30 🎙 O2 Arena

 Filters

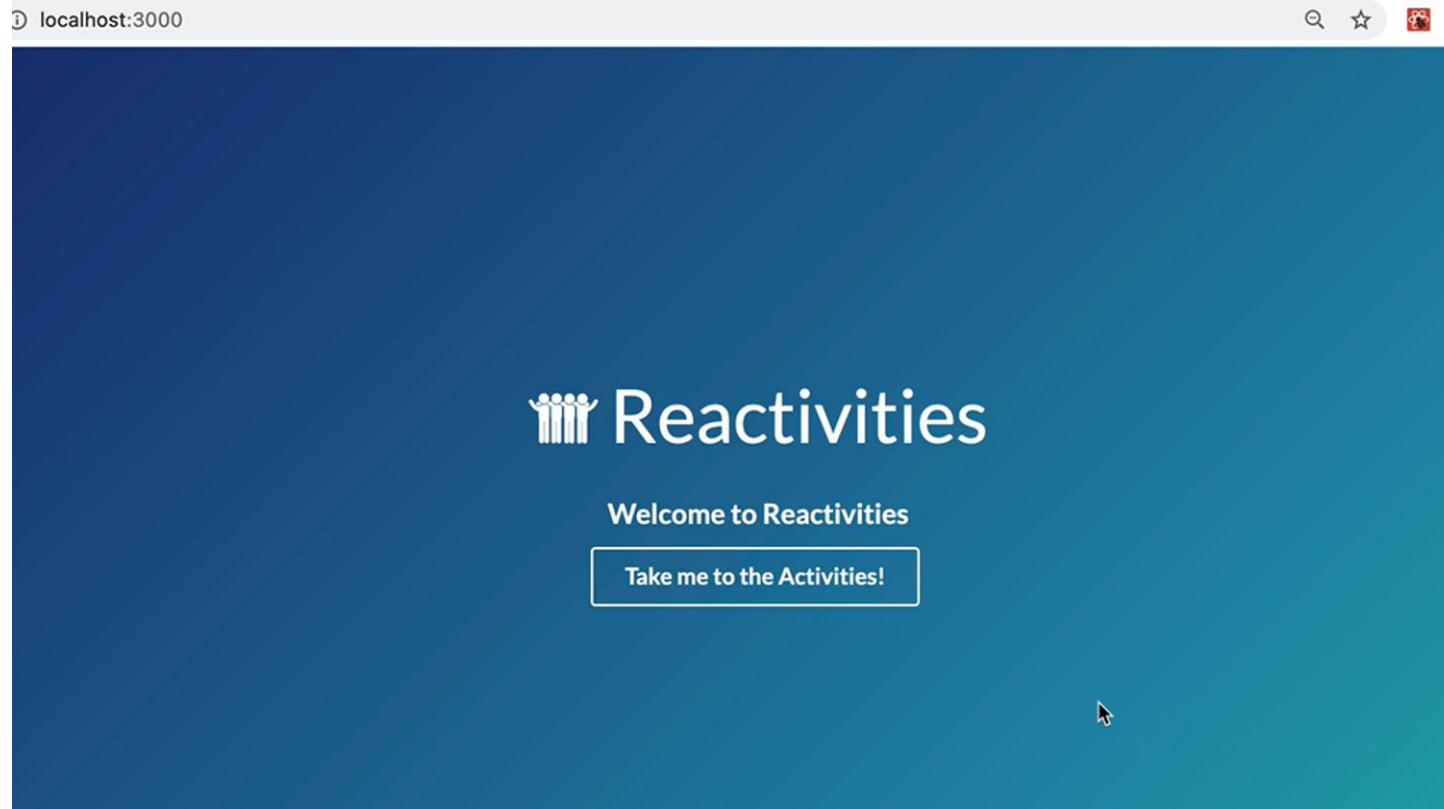
All Activities

I'm going

I'm hosting

January 2021						
MON	TUE	WED	THU	FRI	SAT	SUN
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

# Styling the home page



# Error handling



- Validation
- Handling HTTP Error Responses
- Handling Exceptions
- Custom middleware
- Using Axios interceptors



# HTTP Error responses

---

200 - OK

400 - Bad Request

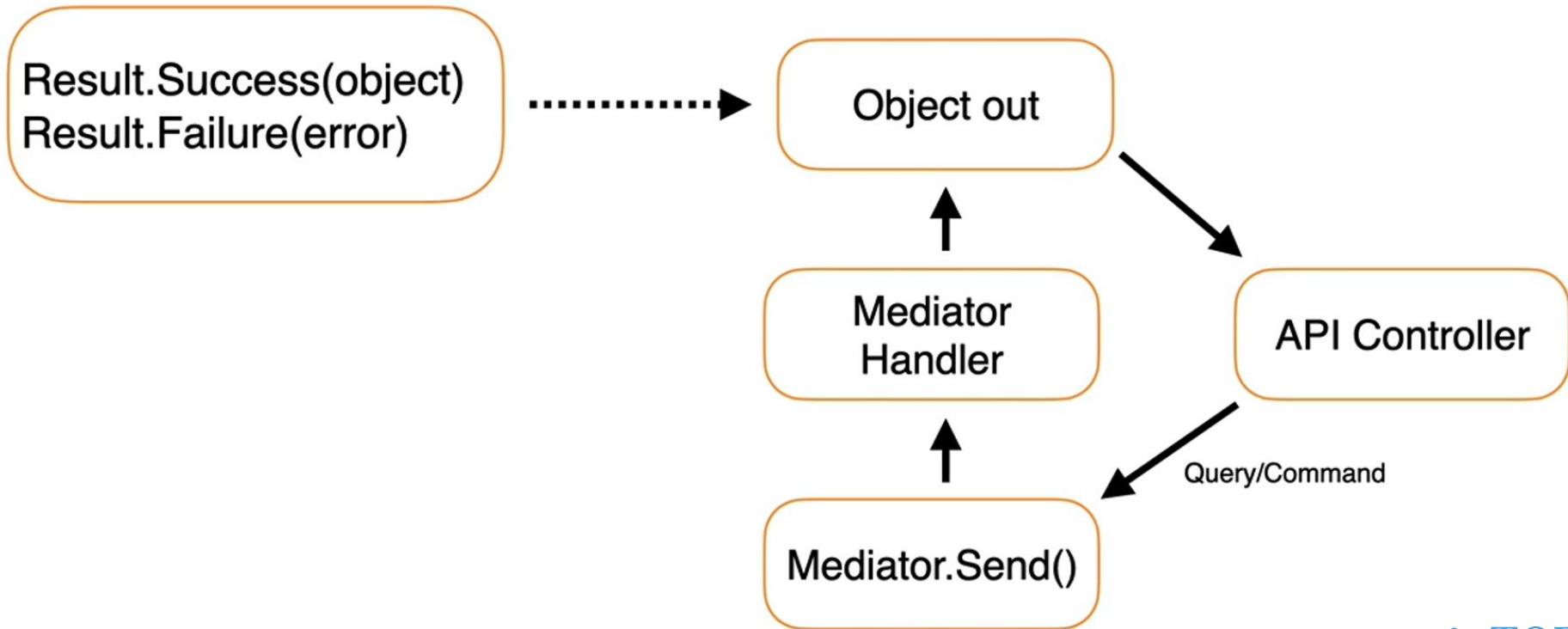
401 - Unauthorised

403 - Forbidden

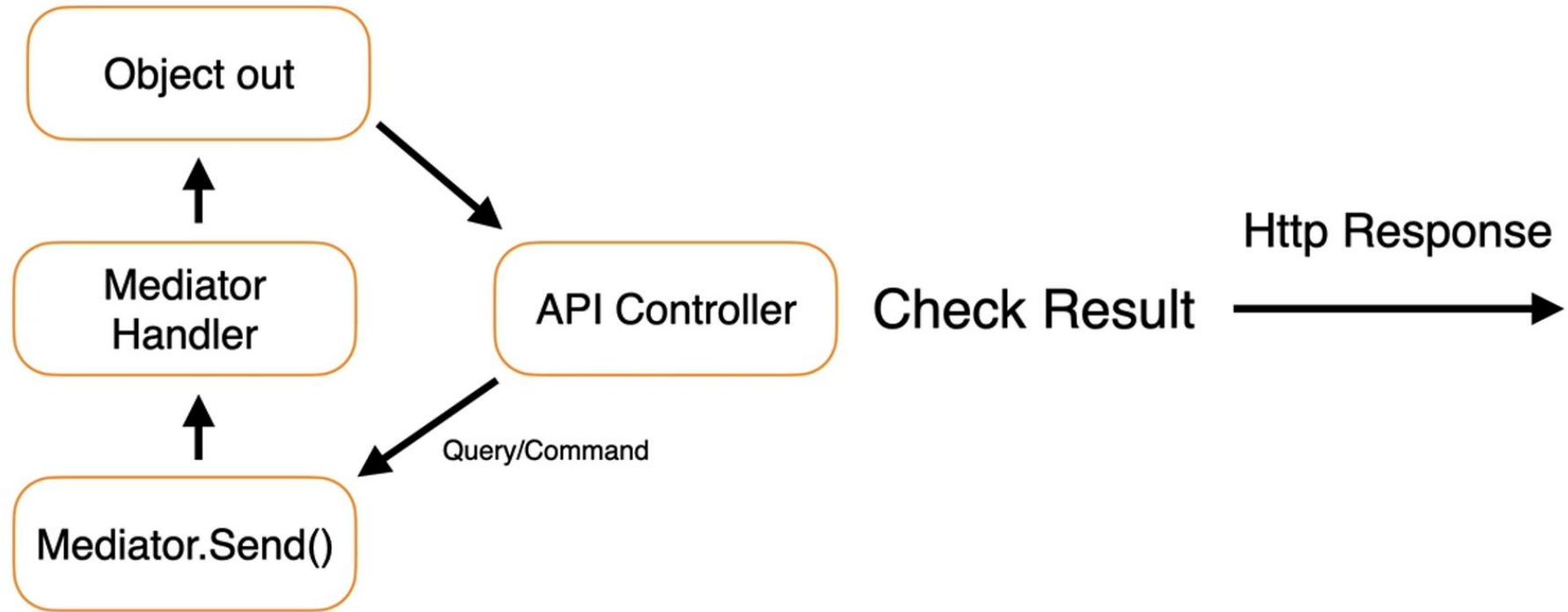
404 - Not Found

500 - Server Error

# Result Object



# Result Object



# Validation with Data annotations

```
public class Activity
{
    2 references
    public Guid Id { get; set; }

    [Required]
    10 references
    public string Title { get; set; }
    10 references
    public DateTime Date { get; set; }
```

# Handling API Error Response - Solution 1

```
[HttpGet("{id}")]
0 references
public async Task<ActionResult<Activity>> GetActivity(Guid id)
{
    var activity = await Mediator.Send(new Details.Query{Id = id});

    if (activity == null) return NotFound();

    return activity;
}
```

**Handle API Error responses in Controller is not good practice**

# Handling API Error Response - Solution 2

```
0 references
public async Task<Activity> Handle(Query request, CancellationToken cancellationToken)
{
    var activity = await _context.Activities.FindAsync(request.Id);

    if (activity == null) throw new Exception("Activity not found");

    return activity;
}
```

**Exception cost lot more than API responses , Exceptions are heavy**

# Result Object

```
> Core > C# Result.cs > (7) Application.Core > Application.Core.Result<T> > Failure(string error)
namespace Application.Core
{
    4 references
    public class Result<T>
    {
        2 references
        public bool IsSuccess { get; set; }
        1 reference
        public T Value { get; set; }
        1 reference
        public string Error { get; set; }

        0 references
        public static Result<T> Success(T value) => new Result<T> {IsSuccess = true, Value = value};
        0 references
        public static Result<T> Failure(string error) => new Result<T> {IsSuccess = false, Error = e
    }
}
```

# Result Object

```
0 references
public async Task<Result<Activity>> Handle(Query request, CancellationToken cancellationToken)
{
    var activity = await _context.Activities.FindAsync(request.Id);

    return Result<Activity>.Success(activity);
}
```

```
[HttpGet("{id}")]
0 references
public async Task<IActionResult> GetActivity(Guid id)
{
    var result = await Mediator.Send(new Details.Query{Id = id});

    if (result.IsSuccess && result.Value != null)
        return Ok(result.Value);
    if (result.IsSuccess && result.Value == null)
        return NotFound();
    return BadRequest(result.Error);
}
```

# Move Logic To BaseAPIController

```
0 references
protected ActionResult HandleResult<T>(Result<T> result)
{
    if (result.IsSuccess && result.Value != null)
        return Ok(result.Value);
    if (result.IsSuccess && result.Value == null)
        return NotFound();
    return BadRequest(result.Error);
}
```

```
[HttpGet("{id}")]
0 references
public async Task<IActionResult> GetActivity(Guid id)
{
    return HandleResult(await Mediator.Send(new Details.Query{Id = id}));
}
```

# List and Create Activity

```
public async Task<Result<List<Activity>>> Handle(Query request, CancellationTokenTok
return Result<List<Activity>>.Success(await _context.Activities.ToListAsync()
```

```
public async Task<Result<Unit>> Handle(Command request, CancellationToken c
{
    _context.Activities.Add(request.Activity);

    var result = await _context.SaveChangesAsync() > 0;

    if (!result) return Result<Unit>.Failure("Failed to create activity");

    return Result<Unit>.Success(Unit.Value);
}
```

# Edit and Delete

```
public async Task<Result<Unit>> Handle(Command request, CancellationToken cancelationToken)
{
    var activity = await _context.Activities.FindAsync(request.Activity.Id);

    if (activity == null) return null;

    _mapper.Map(request.Activity, activity);

    var result = await _context.SaveChangesAsync() > 0;

    if (!result) return Result<Unit>.Failure("Failed to update activity");

    return Result<Unit>.Success(Unit.Value);
}
```

```
protected ActionResult HandleResult<T>(Result<T> result)
{
    if (result == null) return NotFound();
    if (result.IsSuccess && result.Value != null)
        return Ok(result.Value);
    if (result.IsSuccess && result.Value == null)
        return NotFound();
    return BadRequest(result.Error);
}
```

# Exception Handler Middleware - AppException Class

<https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware>

```
namespace Application.Core
{
    0 references
    public class AppException
    {
        0 references
        public AppException(int statusCode, string message, string details = null)
        {
            StatusCode = statusCode;
            Message = message;
            Details = details;
        }

        1 reference
        public int StatusCode { get; set; }
        1 reference
        public string Message { get; set; }
        1 reference
        public string Details { get; set; }
    }
}
```

# ExceptionMiddleware

```
namespace API.Middleware
{
    2 references
    public class ExceptionMiddleware
    {
        1 reference
        private readonly RequestDelegate _next;
        1 reference
        private readonly ILogger<ExceptionMiddleware> _logger;
        1 reference
        private readonly IHostEnvironment _env;
        0 references
        public ExceptionMiddleware(RequestDelegate next, ILogger<ExceptionMiddleware> logger,
            IHostEnvironment env)
        {
            _env = env;
            _logger = logger;
            _next = next;
        }
    }
}
```

# InvokeAsync in ExceptionMiddleware

```
public async Task InvokeAsync(HttpContext context)
{
    try
    {
        await _next(context);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, ex.Message);
        context.Response.ContentType = "application/json";
        context.Response.StatusCode = (int) HttpStatusCode.InternalServerError;

        var response = _env.IsDevelopment()
            ? new AppException(context.Response.StatusCode, ex.Message, ex.StackTrace?.ToString())
            : new AppException(context.Response.StatusCode, "Internal Server Error");

        var options = new JsonSerializerOptions{PropertyNamingPolicy = JsonNamingPolicy.Came
        var json = JsonSerializer.Serialize(response, options);

        await context.Response.WriteAsync(json);
    }
}
```

## Setup Middleware in program.cs

```
// Configure the HTTP request pipeline.  
app.UseMiddleware<ExceptionMiddleware>();  
  
if (app.Environment.IsDevelopment())  
{  
    app.UseSwagger();  
    app.UseSwaggerUI();  
}
```

# Error Handling in Client App

```
namespace API.Controllers
{
    public class BuggyController : BaseApiController
    {
        [HttpGet("not-found")]
        public ActionResult GetNotFound()
        {
            return NotFound();
        }

        [HttpGet("bad-request")]
        public ActionResult GetBadRequest()
        {
            return BadRequest("This is a bad request");
        }

        [HttpGet("server-error")]
        public ActionResult GetServerError()
        {
            throw new Exception("This is a server error");
        }

        [HttpGet("unauthorised")]
        public ActionResult GetUnauthorised()
        {
            return Unauthorized();
        }
    }
}
```

```
export default function TestErrors() {
    const baseUrl = 'http://localhost:5000/api/'

    function handleNotFound() {
        axios.get(baseUrl + 'buggy/not-found').catch(err => console.log(err.response));
    }

    function handleBadRequest() {
        axios.get(baseUrl + 'buggy/bad-request').catch(err => console.log(err.response));
    }

    function handleServerError() {
        axios.get(baseUrl + 'buggy/server-error').catch(err => console.log(err.response));
    }

    function handleUnauthorised() {
        axios.get(baseUrl + 'buggy/unauthorised').catch(err => console.log(err.response));
    }

    function handleBadGuid() {
        axios.get(baseUrl + 'activities/notaguid').catch(err => console.log(err.response));
    }

    function handleValidationError() {
        axios.post(baseUrl + 'activities', {}).catch(err => console.log(err.response));
    }
}
```

# Install react-toastify

<https://www.npmjs.com/package/react-toastify>

```
function App() {
  const location = useLocation();

  return (
    <>
      <ToastContainer position='bottom-right' hideProgressBar theme='colored' />
      {location.pathname === '/' ? <HomePage /> : (
        <>
          <NavBar />
          <Container style={{ marginTop: '7em' }}>
            <Outlet />
          </Container>
        </>
      )}
    </>
  );
}
```

# Axios interceptor

```
axios.interceptors.response.use(async response =>
  await sleep(1000);
  return response;
}, (error: AxiosError) => {
  const {data, status} = error.response!;
  switch (status) {
    case 400:
      toast.error('bad request')
      break;
    case 401:
      toast.error('unauthorised')
      break;
    case 403:
      toast.error('forbidden')
      break;
    case 404:
      toast.error('not found');
      break;
    case 500:
      toast.error('server error');
      break;
  }
  return Promise.reject(error);
})
```

# Adding a not found component

```
src > features > errors > NotFound.tsx > NotFound
import { Link } from "react-router-dom";
import { Button, Header, Icon, Segment } from "semantic-ui-react";

export default function NotFound() {
  return (
    <Segment placeholder>
      <Header icon>
        <Icon name='search' />
        Oops – we've looked everywhere but could not find what you are looking for
      </Header>
      <Segment.Inline>
        <Button as={Link} to='/activities'>
          Return to activities page
        </Button>
      </Segment.Inline>
    </Segment>
  )
}
```

```
export const routes: RouteObject[] = [
  {
    path: '/',
    element: <App />,
    children: [
      {path: 'activities', element: <ActivityDashboard />},
      {path: 'activities/:id', element: <ActivityDetails />},
      {path: 'createActivity', element: <ActivityForm key='create' />},
      {path: 'manage/:id', element: <ActivityForm key='manage' />},
      {path: 'errors', element: <TestErrors />},
      {path: 'not-found', element: <NotFound />},
      {path: '*', element: <Navigate replace to='/not-found' />},
    ]
  }
]
```

# router.navigate() when 404 error

```
import { router } from '../router/Routes';
```

```
    break;
  case 404:
    router.navigate('/not-found');
    break;
  case 500:
    toast.error('server error');
    break;
```

# Handling 400 errors

```
case 400:  
  if (data.errors) {  
    const modalStateErrors = [];  
    for (const key in data.errors) {  
      if (data.errors[key]) {  
        modalStateErrors.push(data.errors[key])  
      }  
    }  
    throw modalStateErrors.flat();  
  } else {  
    toast.error(data);  
  }  
  break;
```

```
function handleValidationError() {  
  axios.post(baseUrl + 'activities', {}).catch(err => console.log(err));  
}
```

# ValidationError Component

```
interface Props {
  errors: string[];
}

export default function ValidationError({errors}: Props) {
  return (
    <Message error>
      {errors && (
        <Message.List>
          {errors.map((err: string, i) => (
            <Message.Item key={i}>{err}</Message.Item>
          )))
        </Message.List>
      )}
    </Message>
  )
}
```

## Test Error component

Not Found	Bad Request	Validation Error	Server Error	Unauthorised	Bad Guid
-----------	-------------	------------------	--------------	--------------	----------

- 'City' must not be empty.
- 'Date' must not be empty.
- 'Title' must not be empty.
- 'Venue' must not be empty.
- 'Category' must not be empty.
- 'Description' must not be empty.

# Handling 500 Error on Client

```
export default class CommonStore {
  error: ServerError | null = null;

  constructor() {
    makeAutoObservable(this);
  }

  setServerError(error: ServerError) {
    this.error = error;
  }
}
```

```
export default observer(function ServerError() {
  const {commonStore} = useStore();
  return (
    <Container>
      <Header as='h1' content='Server Error' />
      <Header sub as='h5' color="red" content={commonStore.error?.message} />
      {commonStore.error?.details && (
        <Segment>
          <Header as='h4' content='Stack trace' color="teal" />
          <code style={{marginTop: '10px'}}>{commonStore.error.details}</code>
        </Segment>
      )}
    </Container>
  )
})
```

```
case 500:  
    store.commonStore.setServerError(data);  
    router.navigate('/server-error');  
    break;
```

```
        {path: 'errors', element: <ServerError />},  
        {path: 'not-found', element: <NotFound />},  
        {path: 'server-error', element: <ServerError />},  
        {path: '*', element: <Navigate replace to='/not-found' />},
```

# Handling error from an Invalid GUID

```
case 400:  
  if (config.method === 'get' && data.errors.hasOwnProperty('id')) {  
    router.navigate('/not-found');  
  }  
  if (data.errors) {  
    const modalStateErrors = [];  
    const errors = data.errors.id || data.errors;  
    errors.forEach(error => {  
      modalStateErrors.push({  
        id: error.id,  
        message: error.message  
      });  
    });  
    setModalState({  
      errors: modalStateErrors,  
      title: 'Error'  
    });  
  }  
}
```

OR...

```
if (config.method === 'get' && Object.prototype.hasOwnProperty.call(data.errors, 'id'))  
  router.navigate('/not-found');  
}  
if (data.errors) {  
  const modalStateErrors = [];
```

# Forms



Comment existing handleChange and handleSubmit methods

# Setting up Formik

<https://formik.org/docs/overview>

```
return (
  <Segment clearing>
    <Formik enableReinitialize initialValues={activity} onSubmit={values => console.log(values)}
      &{({ values: activity, handleChange, handleSubmit }) => (
        <Form onSubmit={handleSubmit} autoComplete='off'>
          <Form.Input placeholder='Title' value={activity.title} name='title' onChange={handleChange}>
          <Form.TextArea placeholder='Description' value={activity.description} name='description' onChange={handleChange}>
          <Form.Input placeholder='Category' value={activity.category} name='category' onChange={handleChange}>
          <Form.Input type='date' placeholder='Date' value={activity.date} name='date' onChange={handleChange}>
          <Form.Input placeholder='City' value={activity.city} name='city' onChange={handleChange}>
          <Form.Input placeholder='Venue' value={activity.venue} name='venue' onChange={handleChange}>
          <Button loading={loading} floated='right' positive type='submit' content='Submit' onClick={handleSubmit}>
          <Button as={Link} to='/activities' floated='right' type='button' content='Cancel' onClick={()=>clearing()}/>
        </Form>
      )}
    </Formik>
)
```

No need of handleChange and value props with Formik's Form and Field , it will handled automatically

# Formik with less code

```
import { Formik, Form, Field } from 'formik';

<Formik enableReinitialize initialValues={activity} onSubmit={values => console.log(values)}>
  <{& handleSubmit} = {({ handleSubmit }) => (
    <Form className='ui form' onSubmit={handleSubmit} autoComplete='off'>
      <Field type='text' placeholder='Title' name='title' />
      <Field placeholder='Description' name='description' />
      <Field placeholder='Category' name='category' />
      <Field type='date' placeholder='Date' name='date' />
      <Field placeholder='City' name='city' />
      <Field placeholder='Venue' name='venue' />
      <Button loading={loading} floated='right' positive type='submit' color='green'>Submit</Button>
      <Button as={Link} to='/activities' floated='right' type='button' color='blue'>Cancel</Button>
    </Form>
  )}
</Formik>
```

# Validation in Formik

<https://formik.org/docs/guides/validation>

```
const validationSchema = Yup.object({
  title: Yup.string().required('The activity title is required')
})
```

```
<Formik
  validationSchema={validationSchema}
  enableReinitialize
  initialValues={activity}
  onSubmit={values => console.log(values)}>
  {({ handleSubmit }) => (
    <Form className='ui form' onSubmit={handleSubmit} autoComplete='off'>
      <FormField>
        <Field placeholder='Title' name='title' />
        <ErrorMessage name='title' />
      </FormField>
    </Form>
  )}
</Formik>
```

# Creating reusable text input

<https://formik.org/docs/api/useField>

```

interface Props {
  placeholder: string;
  name: string;
  label?: string;
}

export default function MyTextInput(props: Props) {
  const [field, meta] = useField(props.name);
  return (
    <Form.Field error={meta.touched && !meta.error}>
      <label>{props.label}</label>
      <input {...field} {...props} />
      {meta.touched && meta.error ? (
        <Label basic color='red'>{meta.error}</Label>
      ) : null}
    </Form.Field>
  )
}

```

```

{({ handleSubmit }) => (
  <Form className='ui form' onSubmit={handleSubmit} autoComplete='off'>
    <MyTextInput name='title' placeholder='Title' />
    <MyTextInput placeholder='Description' name='description' />
    <MyTextInput placeholder='Category' name='category' />
    <MyTextInput type='date' placeholder='Date' name='date' />
  </Form>
)
}

```

# Creating reusable textarea

```
client-app > src > app > common > form > MyTextArea.tsx > Props > rows
1 import { useField } from 'formik';
2 import React from 'react';
3 import { Form, Label } from 'semantic-ui-react';
4
5 interface Props {
6   placeholder: string;
7   name: string;
8   rows: number;
9   label?: string;
10 }
11
12 export default function MyTextArea(props: Props) {
13   const [field, meta] = useField(props.name);
14   return (
15     <Form.Field error={meta.touched && !meta.error}>
16       <label>{props.label}</label>
17       <textarea {...field} {...props} />
18       {meta.touched && meta.error ? (
19         <Label basic color='red'>{meta.error}</Label>
20       ) : null}
21     </Form.Field>
22   )
23 }
```

```
<Form className='ui form' onSubmit={handleSubmit} autoComplete='off'>
  <MyTextInput name='title' placeholder='Title' />
  <MyTextArea rows={3} placeholder='Description' name='description' />
  <MyTextInput placeholder='Category' name='category' />
```

```
interface Props {
  placeholder: string;
  name: string;
  options: {text: string, value: string}[];
  label?: string;
}
```

```
export default function MySelectInput(props: Props) {
  const [field, meta, helpers] = useField(props.name);
  return (
    <Form.Field error={meta.touched && !meta.error}>
      <label>{props.label}</label>
      <Select
        clearable
        options={props.options}
        value={field.value || null}
        onChange={(e, d) => helpers.setValue(d.value)}
        onBlur={() => helpers.setTouched(true)}
        placeholder={props.placeholder}
      />
      {meta.touched && meta.error ? (
        <Label basic color='red'>{meta.error}</Label>
      ) : null}
    </Form.Field>
```

# Creating reusable select input

```
> src > app > common > options > categoryOptions.ts > categoryOptions.ts
export const categoryOptions = [
  {text: 'Drinks', value: 'drinks'},
  {text: 'Culture', value: 'culture'},
  {text: 'Film', value: 'film'},
  {text: 'Food', value: 'food'},
  {text: 'Music', value: 'music'},
  {text: 'Travel', value: 'travel'},
]
```

```
Form className='ui form' onSubmit={handleSubmit} autoComplete='off'>
  <MyTextInput name='title' placeholder='Title' />
  <MyTextArea rows={3} placeholder='Description' name='description' />
  <MySelectInput options={categoryOptions} placeholder='Category' name='category' />
```

# Reusable Date input

<https://www.npmjs.com/package/react-datepicker>

```
<MyDateInput
  ...
  placeholderText='Date'
  name='date'
  showTimeSelect
  timeCaption='time'
  dateFormat='MMMM d, yyyy h:mm aa'
/>
```

```
import DatePicker, {ReactDatePickerProps} from 'react-datepicker';

export default function MyDateInput(props: Partial<ReactDatePickerProps>) {
  const [field, meta, helpers] = useField(props.name!);
  return (
    <Form.Field error={meta.touched && !meta.error}>
      <DatePicker
        {...field}
        {...props}
        selected={(field.value && new Date(field.value)) || null}
        onChange={value => helpers.setValue(value)}
      />
      {meta.touched && meta.error ? (
        <Label basic color='red'>{meta.error}</Label>
      ) : null}
    </Form.Field>
  )
}
```

# The Date strategy

```
private setActivity = (activity: Activity) => {
  activity.date = new Date(activity.date!);
  this.activityRegistry.set(activity.id, activity);
}
```

```
get activitiesByDate() {
  return Array.from(this.activityRegistry.values()).sort((a, b) =>
  a.date!.getTime() - b.date!.getTime());
}
```

```
get groupedActivities() {
  return Object.entries(
    this.activitiesByDate.reduce((activities, activity) => [
      const date = activity.date!.toISOString().split('T')[0];
      activities[date] = activities[date] ? [...activities[date]
```

```
category: Yup.string().required(),
date: Yup.string().required('Date is required').nullable(),
venue: Yup.string().required(),
```

```
export interface Activity {
  id: string;
  title: string;
  date: Date | null;
  description: string;
  category: string;
  city: string;
  venue: string;
}
```

# Using Date-FNS - format Date wherever date displaying

<https://www.npmjs.com/package/date-fns>

```
<Segment>
  <span>
    <Icon name='clock' /> {format(activity.date!, 'dd MMM yyyy h:mm aa')}
    <Icon name='marker' /> {activity.venue}
  </span>
</Segment>
```

```
get groupedActivities() {
  return Object.entries(
    this.activitiesByDate.reduce((activities, activity) => {
      const date = format(activity.date!, 'dd MMM yyyy');
      activities[date] = activities[date] ? [...activities[date], a
```

# Hooking up form submission to Formik

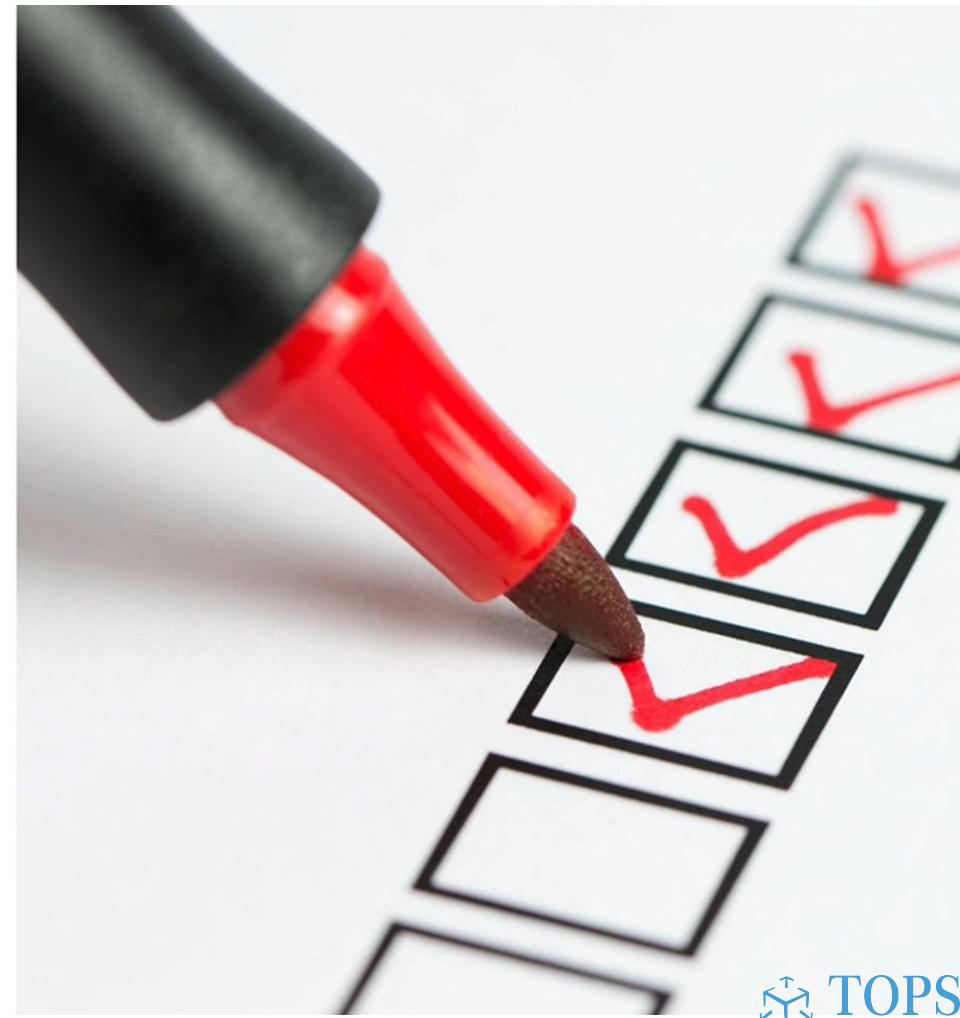
```
function handleFormSubmit(activity: Activity) {
  if (activity.id.length === 0) {
    let newActivity = {
      ...activity,
      id: uuid()
    };
    createActivity(newActivity).then(() =>
  } else {
    updateActivity(activity).then(() => history.push('/'))
  }
}
```

```
<Formik
  validationSchema={validationSchema}
  enableReinitialize
  initialValues={activity}
  onSubmit={values => handleFormSubmit(values)}>
  {({ handleSubmit, isValid, isSubmitting, dirty }) => (
    <Form className='ui form' onSubmit={handleSubmit} autoComplete='off'>
      <MyTextInput name='title' placeholder='Title' />
      <MyTextInput placeholder='Venue' name='venue' />
      <Button
        disabled={isSubmitting || !dirty || !isValid}
        loading={loading} floated='right'
        positive type='submit' content='Submit' />
    </Form>
  )}
</Formik>
```

# Identity



- ASP.NET Core Identity
- JWT Token Authentication
- Login/Register
- Authenticated requests



# ASPNET Core Identity

---

- Membership system
- Supports login stored in Identity
- Supports external providers
- Comes with default user stores
- UserManager
- SignInManager

# Password Hashing + Salting

```
{  
  "email": "sally@test.com",  
  "username": "sally",  
  "displayName": "Sally",  
  "password": "Pa$$w0rd"  
}
```

```
{  
  "email": "bob@test.com",  
  "username": "bob",  
  "displayName": "Bob",  
  "password": "Pa$$w0rd"  
}
```

Pa\$\$w0rd + hash + salt

PasswordHash

AQAAAAEAAACcQAAAAEeZXDOXy3SkBU/ThRX1D0lb8ZrizCOvBAa5t+QN79M61/E9j8ZN78kyIGoKK+52Zw==

AQAAAAEAAACcQAAAEEFRYYJBW99s8+vCicmd4XG0vqc2Ys9Ru0693Mask34Hc/A+RjzIgiL9sGLiivesZg==

# PasswordHasher.cs

---

```
11  namespace Microsoft.AspNetCore.Identity
12  {
13      /// <summary>
14      /// Implements the standard Identity password hashing.
15      /// </summary>
16      /// <typeparam name="TUser">The type used to represent a user.</typeparam>
17      public class PasswordHasher<TUser> : IPasswordHasher<TUser> where TUser : class
18  {
19      /* =====
20       * HASHED PASSWORD FORMATS
21       * =====
22      *
23      * Version 2:
24      * PBKDF2 with HMAC-SHA1, 128-bit salt, 256-bit subkey, 1000 iterations.
25      * (See also: SDL crypto guidelines v5.1, Part III)
26      * Format: { 0x00, salt, subkey }
27      *
28      * Version 3:
29      * PBKDF2 with HMAC-SHA256, 128-bit salt, 256-bit subkey, 10000 iterations.
30      * Format: { 0x01, prf (UInt32), iter count (UInt32), salt length (UInt32), salt, subkey }
31      * (All UInt32s are stored big-endian.)
32      */
33 }
```

Install nuget package then create a model

# Adding User Identity

**Microsoft.AspNetCore.Identity.EntityFrameworkCore**

```
namespace Domain
{
    0 references
    public class AppUser : IdentityUser
    {
        0 references
        public string DisplayName { get; set; }
        0 references
        public string Bio { get; set; }
    }
}
```

Other properties available from **IdentityUser** like  
**Email, UserName, Password, etc.**

# Adding an IdentityDbContext

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace Persistence
{
    15 references
    public class DataContext : IdentityDbContext<AppUser>
    {
}
```

**Run Migration command and check migration file**

# Configuring Identity in the startup class

```
namespace API.Extensions
{
    0 references
    public static class IdentityServiceExtensions
    {
        0 references
        public static IServiceCollection AddIdentityServices(this IServiceCollection services,
            IConfiguration config)
        {
            services.AddIdentityCore<AppUser>(opt =>
            {
                opt.Password.RequireNonAlphanumeric = false;
            })
            .AddEntityFrameworkStores<DataContext>();

            services.AddAuthentication();
        }

        return services;
    }
}

builder.Services.AddApplicationServices(builder.Configuration);
builder.Services.AddIdentityServices(builder.Configuration);
```

# Adding Seed Users

```
+> Persistence > Persistence.Seed > Seed(DataContext context, UserManager<AppUser> userManager)
```

```
1 reference  
public static async Task Seed(DataContext context, UserManager<AppUser> userManager)
```

```
{  
    if (!userManager.Users.Any())  
    {  
        var users = new List<AppUser>  
        {  
            new AppUser{DisplayName = "Bob", UserName = "bob", Email = "bob@test.com"},  
            new AppUser{DisplayName = "Tom", UserName = "tom", Email = "tom@test.com"},  
            new AppUser{DisplayName = "Jane", UserName = "jane", Email = "jane@test.com"},  
        };  
  
        foreach (var user in users)  
        {  
            await userManager.CreateAsync(user, "Pa$$w0rd");  
        }  
    }  
}  
  
if (context.Activities.Any()) return;
```

```
var activities = new List<Activity>
```

```
using var scope = app.Services.CreateScope();  
var services = scope.ServiceProvider;  
  
try  
{  
    var context = services.GetRequiredService<DataContext>();  
    var userManager = services.GetRequiredService<UserManager<AppUser>>();  
    await context.Database.MigrateAsync();  
    await Seed.SeedData(context, userManager);  
}  
catch (Exception ex)  
{  
    var logger = services.GetRequiredService<ILogger<Program>>();  
    logger.LogError(ex, "An error occurred during migration");  
}
```

# Comparison of Approaches

Approach	OnModelCreating()	UserManager in Program.cs
Can hash passwords?	<input checked="" type="checkbox"/> Yes (using PasswordHasher )	<input checked="" type="checkbox"/> Yes (handled by UserManager )
Can manage roles?	<input checked="" type="checkbox"/> Yes (static seed)	<input checked="" type="checkbox"/> Yes (dynamic seed)
Can assign roles to users?	<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes
Works at runtime?	<input type="checkbox"/> No (only design-time)	<input checked="" type="checkbox"/> Yes
Requires migrations?	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No

# Creating user DTOs

```
namespace API.DTOs
{
    public class UserDto
    {
        public string DisplayName { get; set; }
        public string Token { get; set; }
        public string Image { get; set; }
        public string Username { get; set; }
    }

    namespace API.DTOs
    {
        public class LoginDto
        {
            public string Email { get; set; }
            public string Password { get; set; }
        }
    }

    public class RegisterDto
    {
        public string Email { get; set; }
        public string Password { get; set; }
        public string DisplayName { get; set; }
        public string Username { get; set; }
    }
}
```

## Why Use DTOs?

1. **Security** – Prevents exposing sensitive fields from your database models.
2. **Performance** – Transfers only the required fields, reducing payload size.
3. **Decoupling** – Separates domain models from API response formats.
4. **Validation** – DTOs allow custom validation before processing the data.

# Adding Account Controller

```
[HttpPost("login")]
0 references
public async Task<ActionResult<UserDto>> Login(LoginDto loginDto)
{
    var user = await _userManager.FindByEmailAsync(loginDto.Email);

    if (user == null) return Unauthorized();

    var result = await _userManager.CheckPasswordAsync(user, loginDto.Password);

    if (result)
    {
        return new UserDto
        {
            DisplayName = user.DisplayName,
            Image = null,
            Token = "this will be a token",
            Username = user.UserName
        };
    }

    return Unauthorized();
}
```

```
[ApiController]
[Route("api/[controller]")]
0 references
public class AccountController : ControllerBase
{
    1 reference
    private readonly UserManager<AppUser> _userManager;
    0 references
    public AccountController(UserManager<AppUser> userManager)
    {
        _userManager = userManager;
    }
}
```

# JSON Web Tokens (JWT)



# JWT Tokens



JUUT

Debugger   Libraries   Introduction   Ask   Get a T-shirt!

Crafted by  Auth0

```
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ  
uYW1laWQiOiJib2IiLCJuYmYiOjE1NjU0OTU5NzE  
sImV4cCI6MTU2NjEwMDc3MSwiaWF0IjoxNTY1NDK  
10TcxQ.MmWSa-jZMERUmLWops1ZbbQ4U-  
EQwX6FtypUaSz62QVJg4_ISqi5eResCmLGbejXPB  
iuRjSzf8gNYpjRKjWfDA|
```

## HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS512",  
  "typ": "JWT"  
}
```

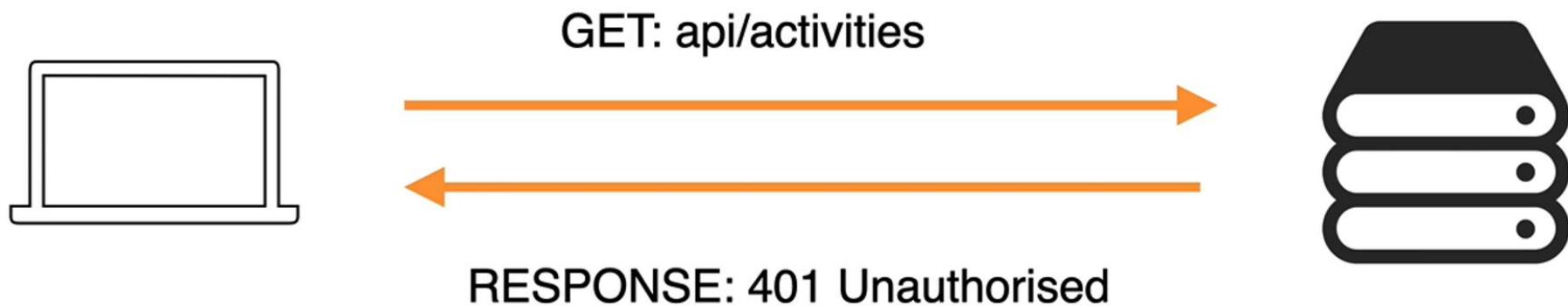
## PAYOUT: DATA

```
{  
  "nameid": "bob",  
  "nbf": 1565495971,  
  "exp": 1566100771,  
  "iat": 1565495971  
}
```

## VERIFY SIGNATURE

```
HMACSHA512(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```

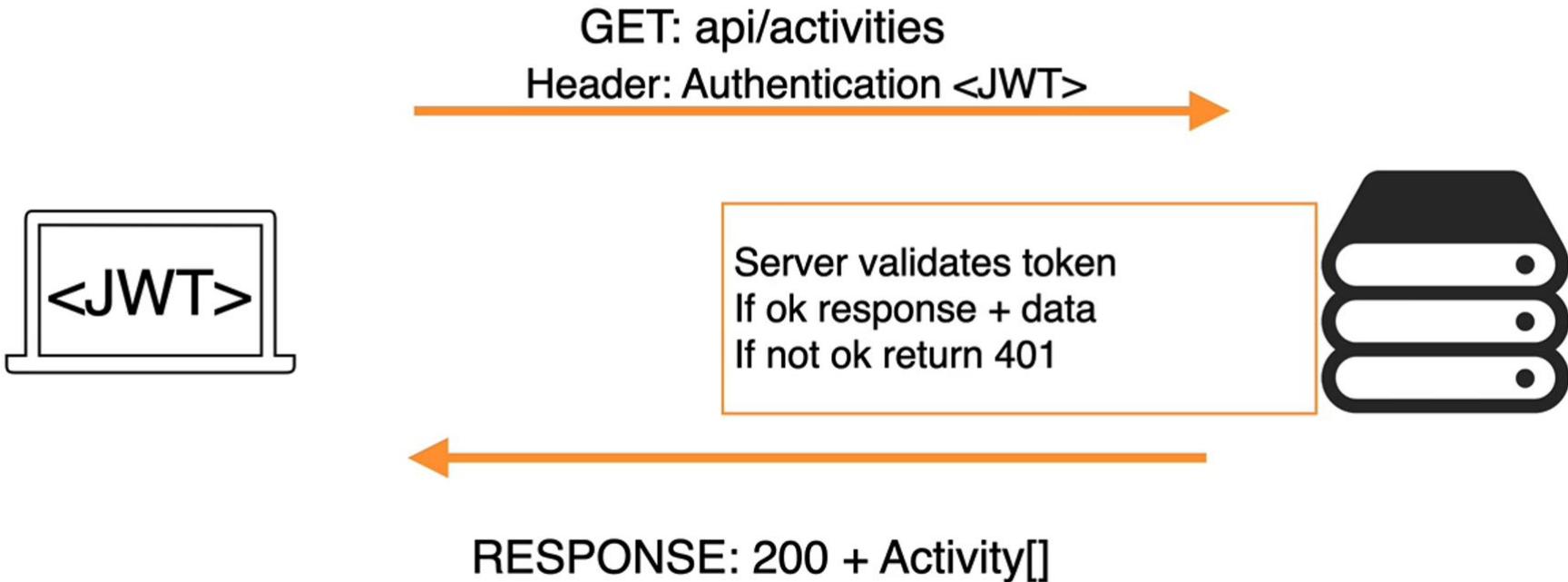
# Authenticating to the server with JWT



# Authenticating to the server with JWT



# Authenticating to the server with JWT



Install nuget package then create a Service

# Creating a token service

```
namespace API.Services
{
    0 references
    public class TokenService
    {
        0 references
        public string CreateToken(AppUser user)
        {
            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, user.UserName),
                new Claim(ClaimTypes.NameIdentifier, user.Id),
                new Claim(ClaimTypes.Email, user.Email),
            };

            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("super secret key"));
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(claims),
                Expires = DateTime.UtcNow.AddDays(7),
                SigningCredentials = creds
            };
            var tokenHandler = new JwtSecurityTokenHandler();
            var token = tokenHandler.CreateToken(tokenDescriptor);
        }
    }
}
```

System.IdentityModel.Tokens.Jwt

Register Service in IdentityServiceExtension

```
services.AddAuthentication();
services.AddScoped<TokenService>();
```

```
private readonly UserManager<AppUser> _userManager;
1 reference
private readonly TokenService _tokenService;
0 references
public AccountController([UserManager<AppUser> userManager, TokenService tokenService])
{
    _tokenService = tokenService;
    _userManager = userManager;
}

if (result)
{
    return new UserDto
    {
        DisplayName = user.DisplayName,
        Image = null,
        Token = _tokenService.CreateToken(user),
        Username = user.UserName
    };
}
```

# If Get Any Error ???

<https://passwordsgenerator.net/>

```
{  
    "statusCode": 500,  
    "message": "IDX10720: Unable to create KeyedHashAlgorithm for algorithm 'http://www.w3.org/2001/04/xmldsig-more#hmac-sha512', the key size must be greater than: '512' bits. [key has '128' bits. (Parameter 'keyBytes')]  
    ",  
    "details": "    at Microsoft.IdentityModel.Tokens.CryptoProviderFactory.ValidateKeySize(Byte[] keyBytes, String  
        algorithm, Int32 expectedNumberOfBytes)\n    at Microsoft.IdentityModel.Tokens.CryptoProviderFactory.  
        CreateKeyedHashAlgorithm(Byte[] keyBytes, String algorithm)\n    at Microsoft.IdentityModel.Tokens.
```

**With new version of Jwt, Secret Key must be 64 character long**

**Use <https://passwordsgenerator.net/> to generate Secret Key and replace “super secret key”**

**You can decode token on jwt.ms and check payload**

Install nuget package  
Microsoft.AspNetCore.Authentication.JwtBearer

# Authenticating to the App

```
var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("super secret key"));

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(opt =>
{
    opt.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = key,
        ValidateIssuer = false,
        ValidateAudience = false
    };
});

[Authorize]
[HttpGet("{id}")]
0 references
public async Task<ActionResult<Activity>> GetActivity(Guid id)
{
    return HandleResult(await Mediator.Send(new Details.Query { Id = id }));
}
```

app.UseCors("CorsPolicy");  
app.UseAuthentication();  
app.UseAuthorization();

# Storing secrets in development

<https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-9.0&tabs=windows>

```
"ConnectionStrings": {  
    "DefaultConnection": "Data Source=reacti  
},  
"TokenKey": "super secret key"
```

```
1 reference  
private readonly IConfiguration _config;  
0 references  
public TokenService(IConfiguration config)  
{  
    _config = config;  
}
```

```
var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["TokenKey"]));  
var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);
```

No need to write [Authorize] annotation with every API end points

## Creating an auth policy

```
builder.Services.AddControllers(opt =>
{
    var policy = new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();
    opt.Filters.Add(new AuthorizeFilter(policy));
});
```

```
[AllowAnonymous]
[ApiController]
[Route("api/[controller]")]
0 references
public class AccountController : ControllerBase
{
    3 references
    private readonly UserManager<AppUser> _userManager;
```

# Registering new user

```
[HttpPost("register")]
0 references
public async Task<ActionResult<UserDto>> Register(RegisterDto registerDto)
{
    if (await _userManager.Users.AnyAsync(x => x.UserName == registerDto.Username))
    {
        return BadRequest("Username is already taken");
    }

    var user = new AppUser
    {
        DisplayName = registerDto.DisplayName,
        Email = registerDto.Email,
        UserName = registerDto.Username
    };

    var result = await _userManager.CreateAsync(user, registerDto.Password);

    if (result.Succeeded)
    {
        return new UserDto
        {
            DisplayName = user.DisplayName,
            Image = null,
            Token = _tokenService.CreateToken(user),
            Username = user.UserName
        };
    }

    return BadRequest(result.Errors);
}
```

# Replace BadRequest Response ValidationProblem

```
if (await _userManager.Users.AnyAsync(x => x.Email == registerDto.Email))
{
    ModelState.AddModelError("email", "Email taken");
    return ValidationProblem();
}
if (await _userManager.Users.AnyAsync(x => x.UserName == registerDto.Username))
{
    ModelState.AddModelError("username", "Username taken");
    return ValidationProblem();
}
```

# Validating the Registration of Users

```
public class RegisterDto
{
    [Required]
    1 reference
    public string DisplayName { get; set; }

    [Required]
    [EmailAddress]
    2 references
    public string Email { get; set; }

    [Required]
    [RegularExpression("(?=.*\\d)(?=.*[a-z])(?=.*[A-Z]).{4,8}$", ErrorMessage = "Password must contain at least one uppercase letter, one lowercase letter, and one digit, and be between 4 and 8 characters long")]
    1 reference
    public string Password { get; set; }

    [Required]
    2 references
    public string Username { get; set; }
}
```

# Getting the current user

```
[Authorize]
[HttpGet]
0 references
public async Task<ActionResult<UserDto>> GetCurrentUser()
{
    var user = await _userManager.FindByEmailAsync(User.FindFirstValue(ClaimTypes.Email));

    return CreateUserObject(user);
}
0 references
private UserDto CreateUserObject([AppUser]User)
{
    return new UserDto
    {
        DisplayName = user.DisplayName,
        Image = null,
        Token = _tokenService.CreateToken(user),
        Username = user.UserName
    };
}
```

# Client Side login and registration



- Axios Interceptors
- MobX reactions
- Form submission errors
- Modals



# Create a Login Form

```
{path: 'login', element: <LoginForm />},  
  
port default function HomePage() {  
  return (  
    <Segment inverted textAlign='center' vertical  
      <Container text>  
        <Header as='h1' inverted>  
          <Image size='massive' src='/' alt='Reactivities logo' />  
          Reactivities  
        </Header>  
        <Header as='h2' inverted content='Welcome back!' />  
        <Button as={Link} to='/login' size='large' style={{marginBottom: 10}}>  
          Login!  
        </Button>  
      </Container>  
    </Segment>
```

```
export default function LoginForm() {  
  return (  
    <Formik  
      initialValues={{email: '', password: ''}}  
      onSubmit={values => console.log(values)}  
    >  
      <{({handleSubmit}) => (  
        <Form className='ui form' onSubmit={handleSubmit} autoComplete='off'>  
          <MyTextInput placeholder='Email' name='email' />  
          <MyTextInput placeholder='Password' name='password' type='password' />  
          <Button positive content='Login' type='submit' fluid />  
        </Form>  
      )}>  
    </Formik>
```

# Creating interfaces and methods

```
const Account = {  
    current: () => requests.get<User>('/account'),  
    login: (user: UserFormValues) => requests.post<User>('/account/login', user),  
    register: (user: UserFormValues) => requests.post<User>('/account/register', user)  
}
```

```
const agent = {  
    Activities,  
    Account|
```

```
> src > app > models > user.ts > UserFormValues  
export interface User {  
    username: string;  
    displayName: string;  
    token: string;  
    image?: string;  
}  
  
export interface UserFormValues {
```

```
    email: string;  
    password: string;  
    displayName?: string;  
    username?: string;
```

# Creating a User store

```
export default observer(function LoginForm() {
  const {userStore} = useStore();
  return (
    <Formik
      initialValues={{email: '', password: ''}}
      onSubmit={values => userStore.login(values)}
    >
      {(handleSubmit, isSubmitting) => (
        <Form className='ui form' onSubmit={handleSubmit} autoComplete='off'>
          <MyTextInput name='email' placeholder='Email' />
          <MyTextInput name='password' placeholder='Password' type='password' />
          <Button loading={isSubmitting} positive content='Login' type='submit' fluid />
        </Form>
      )}
    </Formik>
  )
})
```

```
export default class UserStore {
  user: User | null = null;

  constructor() {
    makeAutoObservable(this)
  }

  get isLoggedIn() {
    return !!this.user;
  }

  login = async (creds: UserFormValues) => {
    const user = await agent.Account.login(creds);
    console.log(user);
  }
}
```

# Displaying errors in the form

```
<Formik
  initialValues={{email: '', password: '', error: null}}
  onSubmit={(values, {setErrors}) => userStore.login(values).catch(error =>
    setErrors({error: 'Invalid email or password'}))}
>
  {(handleSubmit, isSubmitting, errors) =>
    <Form className='ui form' onSubmit={handleSubmit} autoComplete='off'>
      <MyTextInput name='email' placeholder='Email' />
      <MyTextInput name='password' placeholder='Password' type='password' />
      <ErrorMessage
        name='error' render={() =>
          <Label style={{marginBottom: 10}} basic color='red' content={errors.error}/>
        />
      <Button loading={isSubmitting} positive content='Login' type='submit' fluid />
    </Form>
  }
</Formik>
```

# Setting the token upon login

```
login = async (creds: UserFormValues) => {
  try {
    const user = await agent.Account.login(creds);
    store.commonStore.setToken(user.token);
    runInAction(() => this.user = user);
    router.navigate('/activities');
  } catch (error) {
    throw error;
  }
}
```

User store

```
logout = () => [
  store.commonStore.setToken(null),
  localStorage.removeItem('jwt'),
  this.user = null,
  router.navigate('/');
]
```

```
setToken = (token: string | null) => {
  if (token) localStorage.setItem('jwt', token);
  this.token = token;
}
```

Common store

```
setAppLoaded = () => {
  this.appLoaded = true;
}
```

# Updating Home Page and nav bar

```
{userStore.isLoggedIn ? (
    <>
        <Header as='h2' inverted content='Welcome to Reactivities' />
        <Button as={Link} to='/activities' size='huge' inverted>
            Go to Activities!
        </Button>
    </>
) : (
    <Button as={Link} to='/login' size='huge' inverted>
        Login!
    </Button>
)}
```

Home Page

```
const {userStore: {user, logout}} = useStore();
<Menu.Item position='right'>
    <Image src={user?.image || '/assets/user.png'} avatar spaced='right' />
    <Dropdown pointing='top left' text={user?.displayName}>
        <Dropdown.Menu>
            <Dropdown.Item as={Link} to={`/profile/${user?.username}`}
                text='My Profile' icon='user' />
            <Dropdown.Item onClick={logout} text='Logout' icon='power' />
        </Dropdown.Menu>
    </Dropdown>
</Menu.Item>
```

Navbar

# Persisting login

```
getUser = async () => {
  try {
    const user = await agent.Account.current();
    runInAction(() => this.user = user);
  } catch (error) {
    console.log(error)
  }
}
```

UserStore

```
export default class CommonStore {
  error: ServerError | null = null;
  token: string | null = localStorage.getItem('jwt');
  appLoaded = false;
```

```
function App() {
  const location = useLocation();
  const {commonStore, userStore} = useStore();
```

```
useEffect(() => {
  if (commonStore.token) {
    userStore.getUser().finally(() => commonStore.setAppLoaded())
  } else {
    commonStore.setAppLoaded()
  }
}, [commonStore, userStore])
```

```
if (!commonStore.appLoaded) return <LoadingComponent content='Loading ...'>
```

# Sending up the token with request

```
axios.interceptors.request.use(config => {
  const token = store.commonStore.token;
  if (token && config.headers) config.headers.Authorization = `Bearer ${token}`;
  return config;
})
```

# Adding Register Form

```
<Formik
  initialValues={{ displayName: '', username: '', email: '', password: '', error: null }}
  onSubmit={({ values, { setErrors } }) =>
    userStore.register(values).catch(error => setErrors({ error: 'Invalid email or password' }))
  }
  validationSchema={Yup.object({
    displayName: Yup.string().required(),
    username: Yup.string().required(),
    email: Yup.string().required(),
    password: Yup.string().required(),
  })}
}

{{ handleSubmit, isSubmitting, errors, isValid, dirty }} => (
  <Form className='ui form' onSubmit={handleSubmit} autoComplete='off'>
    <Header as='h2' content='Sign up to Reactivities' color="teal" textAlign="center" />
    <MyTextInput placeholder="Display Name" name='displayName' />
    <MyTextInput placeholder="Username" name='username' />
    <MyTextInput placeholder="Email" name='email' />
    <MyTextInput placeholder="Password" name='password' type='password' />
    <ErrorMessage name='error' render={() =>
      <Label style={{ marginBottom: 10 }} basic color='red' content={errors.error} />
    }
    <Button
      disabled={!isValid || !dirty || isSubmitting}
      loading={isSubmitting}
      positive content='Register'
      type="submit" fluid
    />
  </Form>
)
```

```
register = async (creds: UserFormValues) => {
  try {
    const user = await agent.Account.register(creds);
    store.commonStore.setToken(user.token);
    runInAction(() => this.user = user);
    router.navigate('/activities');
    store.modalStore.closeModal();
  } catch (error) {
    throw error;
  }
}
```

# Handling validation errors in registration form

```
<Formik
  initialValues={[displayName: '', username: '', email: '', password: '', error: null]}
  onSubmit={(values, {setErrors}) => userStore.register(values).catch(error =>
    setErrors({error}))}
  validationSchema={Yup.object({
    displayName: Yup.string().required()
  })}
```

```
<MyTextInput placeholder='Password' name='password' type='password' />
<ErrorMessage
  name='error' render={() =>
    <ValidationErrors errors={errors.error as unknown as string[]} />}
}>
<Button
  disabled={!isValid || !dirty || isSubmitting}
  loading={isSubmitting} positive content='Register' type='submit' fluid />
```

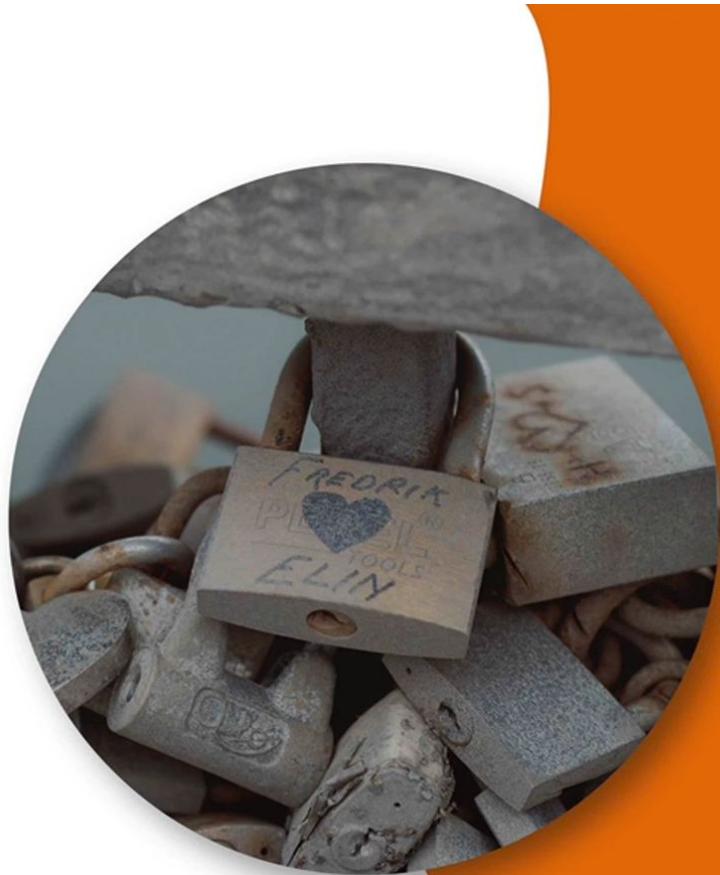
➤ **Storing JWT token in local storage is insecure!**

It can be. Weaknesses include:

1. XSS attacks. Javascript can access localStorage. This requires the attacker to execute malicious javascript on your application.
2. Local machine privileges. Anyone who has admin access can access localStorage for another user account.

If an attacker can run malicious javascript on your application then a JWT stored in localStorage is the least of your problems. This is what we will address later.

# EF Relationships



# One to One

---



# One to Many

---



# Many to Many

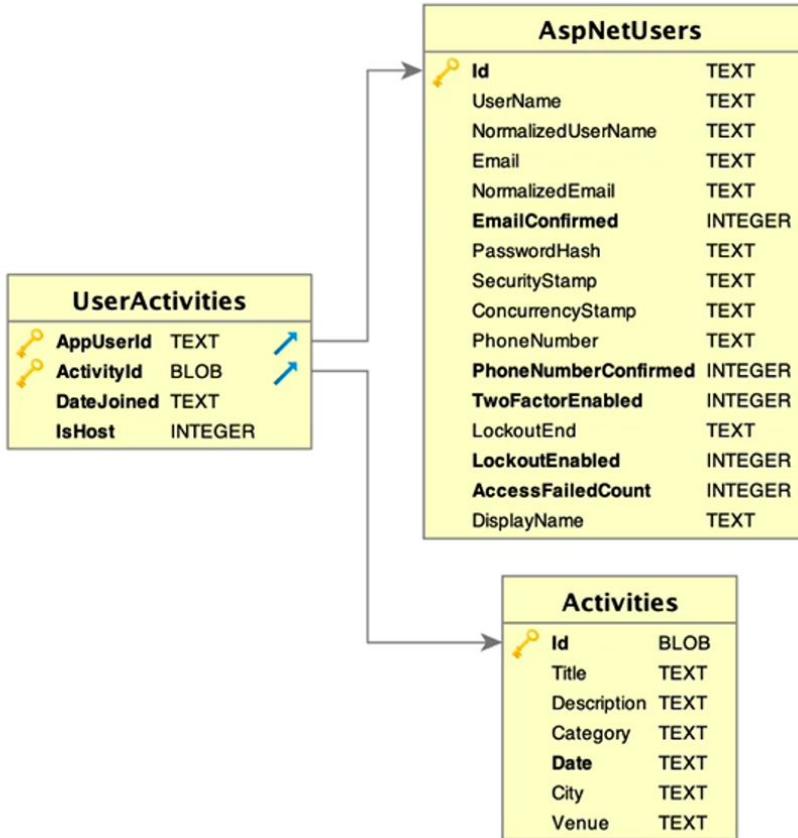
---



# Many to Many



# Many to Many join table



# EF Core convention for Many to Many Relationship

```
public string Venue { get; set; }  
0 references  
public ICollection<AppUser> Attendees { get; set; }
```

```
public string Bio { get; set; }  
0 references  
public ICollection<Activity> Activities { get; set; }
```

**Add Migration and check migration file**

**New Table will created for relationship**

**But it will not work if want additional column like  
to check if user is the host for the activity**

**So go for remove-migration**

# Configuring new Relationship model class

```
public class ActivityAttendee
{
    0 references
    public string AppUserId { get; set; }
    0 references
    public AppUser AppUser { get; set; }
    0 references
    public Guid ActivityId { get; set; }
    0 references
    public Activity Activity { get; set; }
    0 references
    public bool IsHost { get; set; }
}
```

```
0 references
public string Bio { get; set; }
0 references
public ICollection<ActivityAttendee> Activities { get; set; }
```

```
1 reference
public string Venue { get; set; }
2 references
public ICollection<ActivityAttendee> Attendees { get; set; } = new List<ActivityAttendee>();
```

# Configuring Primary and Foreign Key

```
0 references
public DbSet<ActivityAttendee> ActivityAttendees { get; set; }

protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);

    builder.Entity<ActivityAttendee>(x => x.HasKey(aa => new {aa.AppUserId, aa.ActivityId}));

    builder.Entity<ActivityAttendee>()
        .HasOne(u => u.AppUser)
        .WithMany(a => a.Activities)
        .HasForeignKey(aa => aa.AppUserId);

    builder.Entity<ActivityAttendee>()
        .HasOne(u => u.Activity)
        .WithMany(a => a.Attendees)
        .HasForeignKey(aa => aa.ActivityId);
}
```

Add Migration and check migration file

# Infrastructure project

---



UserAccessor

GetUsername()



Implementation

IUserAccessor

GetUsername()



Abstraction

```
namespace Infrastructure.Security
```

```
{  
    0 references  
    public class UserAccessor : IUserAccessor  
    {  
        2 references  
        private readonly IHttpContextAccessor _httpContextAccessor;  
        0 references  
        public UserAccessor(IHttpContextAccessor httpContextAccessor)  
        {  
            _httpContextAccessor = httpContextAccessor;  
        }  
  
        0 references  
        public string GetUsername()  
        {  
            return _httpContextAccessor.HttpContext.User.FindFirstValue(ClaimTypes.Name);  
        }  
    }  
}
```

```
namespace Application.Interfaces
```

```
{  
    0 references  
    public interface IUserAccessor  
    {  
        0 references  
        string GetUsername();  
    }  
}
```

```
services.AddValidatorsFromAssemblyContaining<Create>();  
services.AddHttpContextAccessor();  
services.AddScoped<IUserAccessor, UserAccessor>();  
  
return services;
```

# Updating Create activity handler

```
private readonly DataContext _context;  
  
private readonly IUserAccessor _userAccessor;  
0 references  
public Handler(DataContext context, IUserAccessor userAccessor)  
{  
    _userAccessor = userAccessor;  
    _context = context;  
}
```

```
public async Task<Result<Unit>> Handle(Command request, C  
{  
    var user = await _context.Users.FirstOrDefaultAsync(x  
        x.UserName == _userAccessor.GetUsername());  
  
    var attendee = new ActivityAttendee  
    {  
        AppUser = user,  
        Activity = request.Activity,  
        IsHost = true  
    };  
  
    request.Activity.Attendees.Add(attendee);  
  
    _context.Activities.Add(request.Activity);  
}
```

# Getting related data

<https://learn.microsoft.com/en-us/ef/core/querying/related-data/>

```
public async Task<Result<List<Activity>>> Handle(Query request, CancellationToken cancellationToken)
{
    var activities = await _context.Activities
        .Include(a => a.Attendees)
        .ThenInclude(u => u.AppUser)
        .ToListAsync(cancellationToken);

    return Result<List<Activity>>.Success(activities);
}
```

```
{  
    "statusCode": 500,  
    "message": "A possible object cycle was detected. This can either be due to a cycle or if the object depth is larger than the maximum allowed depth of 32. Consider using ReferenceHandler.Preserve on JsonSerializerOptions to support cycles.",  
    "details": "    at System.Text.Json.ThrowHelper.ThrowJsonException_SerializerCycleDetected(Int32 maxDepth)\n    at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)\n    at System.Text.Json.JsonPropertyInfo`1.GetMemberAndWriteJson(Object obj, WriteStack& state, Utf8JsonWriter writer)\n    at System.Text.Json.Serialization.Converters.ObjectDefaultConverter`1.OnTryWrite(Utf8JsonWriter writer, Object value)
```

# Shaping the Related Data

```
Application > Activities > C# ActivityDto.cs > {} Application.Activities > Application.Activities.ActivityDto
 7   public class ActivityDto
 8   {
 9     public Guid Id { get; set; }
10    public string Title { get; set; }
11    public DateTime Date { get; set; }
12    public string Description { get; set; }
13    public string Category { get; set; }
14    public string City { get; set; }
15    public string Venue { get; set; }
16    public string HostUsername { get; set; }
17    public ICollection<Profile> Profiles { get; set; }
18 }
```

```
namespace Application.Profiles
{
 0 references
  public class Profile
  {
    0 references
    public string Username { get; set; }
    0 references
    public string DisplayName { get; set; }
    0 references
    public string Bio { get; set; }
    0 references
    public string Image { get; set; }
  }
}
```

```
public MappingProfiles()
{
    CreateMap<Activity, Activity>();
    CreateMap<Activity, ActivityDto>()
        .ForMember(d => d.HostUsername, o => o.MapFrom(s => s.Attendees
            .FirstOrDefault(x => x.IsHost).AppUser.UserName));
    CreateMap<ActivityAttendee, Profiles.Profile>()
        .ForMember(d => d.DisplayName, o => o.MapFrom(s => s.AppUser.DisplayName))
        .ForMember(d => d.Username, o => o.MapFrom(s => s.AppUser.UserName))
        .ForMember(d => d.Bio, o => o.MapFrom(s => s.AppUser.Bio));
}
```

```
public async Task<Result<List<ActivityDto>>> Handle(Query request)
{
    var activities = await _context.Activities
        .ProjectTo<ActivityDto>(_mapper.ConfigurationProvider)
        .ToListAsync(cancellationToken);

    return Result<List<ActivityDto>>.Success(activities);
}
```

# Details.cs

```
using System;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using ActivityManagementSystem.Dtos;
using ActivityManagementSystem.Entities;
using ActivityManagementSystem.Repositories;
using ActivityManagementSystem.Services;
using AutoMapper;
using ConfigurationProvider;

namespace ActivityManagementSystem.Handlers
{
    public class DetailsQueryHandler : IQueryHandler<GetActivityQuery, Result<ActivityDto>>
    {
        private readonly IRepository<Activity> _context;
        private readonly IMapper _mapper;
        private readonly IConfigurationProvider _configurationProvider;
        private readonly ILogger<DetailsQueryHandler> _logger;

        public DetailsQueryHandler(IRepository<Activity> context, IMapper mapper, IConfigurationProvider configurationProvider, ILogger<DetailsQueryHandler> logger)
        {
            _context = context;
            _mapper = mapper;
            _configurationProvider = configurationProvider;
            _logger = logger;
        }

        public async Task<Result<ActivityDto>> Handle(Query request, CancellationToken cancellationToken)
        {
            var activity = await _context.Activities
                .ProjectTo<ActivityDto>(_mapper.ConfigurationProvider)
                .FirstOrDefaultAsync(x => x.Id == request.Id);

            return Result<ActivityDto>.Success(activity);
        }
    }
}
```

# Adding Attendance handler

```
public bool IsCancelled { get; set; }  
0 references  
public ICollection<Profile> Attendees { get; set; }
```

In both class Activity and ActivityDto

Then Add migration

Add New CQRS command class

Activities/UpdateAttendance

```
public class UpdateAttendance  
{  
    1 reference  
    public class Command : IRequest<Result<Unit>>  
    {  
        0 references  
        public Guid Id { get; set; }  
    }  
  
    0 references  
    public class Handler : IRequestHandler<Command, Result<Unit>>  
    {  
        }  
}
```

```
private readonly DataContext _context;
private readonly IUserAccessor _userAccessor;
0 references
public Handler(DataContext context, IUserAccessor userAccessor)
{
    _userAccessor = userAccessor;
    _context = context;
}

public async Task<Result<Unit>> Handle(Command request, CancellationToken cancellationToker
{
    var activity = await _context.Activities
        .Include(a => a.Attendees).ThenInclude(u => u.AppUser)
        .SingleOrDefaultAsync(x => x.Id == request.Id);

    if (activity == null) return null;

    var user = await _context.Users.FirstOrDefaultAsync(x =>
        x.UserName == _userAccessor.GetUsername());

    if (user == null) return null;

    var hostUsername = activity.Attendees.FirstOrDefault(x => x.IsHost)?.AppUser?.UserName;
```

```
var attendance = activity.Attendees.FirstOrDefault(x => x.AppUser.UserName == user.UserName);

if (attendance != null && hostUsername == user.UserName)
    activity.IsCancelled = !activity.IsCancelled;

if (attendance != null && hostUsername != user.UserName)
    activity.Attendees.Remove(attendance);

if (attendance == null)
{
    attendance = new ActivityAttendee
    {
        AppUser = user,
        Activity = activity,
        IsHost = false
    };

    activity.Attendees.Add(attendance);
}

var result = await _context.SaveChangesAsync() > 0;

return result ? Result<Unit>.Success(Unit.Value) : Result<Unit>.Failure("Problem updating attendance record")
```

# ActivitiesController

```
[HttpPost("{id}/attend")]
0 references
public async Task<IActionResult> Attend(Guid id)
{
    return HandleResult(await Mediator.Send(new UpdateAttendance.Command{Id = id}));
}
```

# Adding Custom auth policy

```
namespace Infrastructure.Security
{
    2 references
    public class IsHostRequirement : IAuthorizationRequirement
    {
    }

    0 references
    public class IsHostRequirementHandler : AuthorizationHandler<IsHostRequirement>
    {

        protected override Task HandleRequirementAsync(AuthorizationHandlerContext context)
        {
            throw new System.NotImplementedException();
        }
    }
}

private readonly DataContext _dbContext;
1 reference
private readonly IHttpContextAccessor _httpContextAccessor;
0 references
public IsHostRequirementHandler(DataContext dbContext,
    IHttpContextAccessor httpContextAccessor)
{
    _httpContextAccessor = httpContextAccessor;
    _dbContext = dbContext;
}
```

```
protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, Is
{
    var userId = context.User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null) return Task.CompletedTask;

    var activityId = Guid.Parse(_httpContextAccessor.HttpContext?.Request.RouteValues
        .SingleOrDefault(x => x.Key == "id").Value?.ToString());

    var attendee = _dbContext.ActivityAttendees
        .FindAsync(userId, activityId).Result;

    if (attendee == null) return Task.CompletedTask;

    if (attendee.IsHost) context.Succeed(requirement);

    return Task.CompletedTask;
}
```

```
services.AddAuthorization(opt =>
{
    opt.AddPolicy("IsActivityHost", policy =>
    {
        policy.Requirements.Add(new IsHostRequirement());
    });
});

services.AddTransient<IAuthorizationHandler, IsHostRequirementHandler>();
services.AddScoped<TokenService>();
```

```
[Authorize(Policy = "IsActivityHost")]
[HttpPut("{id}")]
0 references
public async Task<IActionResult> EditActivity(Guid id, Activity activity)
{
    activity.Id = id;
    return HandleResult(await Mediator.Send(new Edit.Command { Id = id, Activity = activity }));
}
```

```
[Authorize(Policy = "IsActivityHost")]
[HttpDelete("{id}")]
0 references
public async Task<IActionResult> DeleteActivity(Guid id)
```

# Feature - client side attendance



# Adding the attendees component - Add Model

```
> src > app > models > profile.ts > Pro  
export interface Profile {  
    username: string;  
    displayName: string;  
    image?: string;  
    bio?: string;  
}
```

```
export interface Activity {  
    id: string;  
    title: string;  
    date: Date | null;  
    description: string;  
    category: string;  
    city: string;  
    venue: string;  
    hostUsername?: string;  
    isCancelled?: boolean;  
    attendees?: Profile[]  
}
```

# Adding the attendees component - with ActivityList Item

```
<Segment secondary>
  <ActivityListItemAttendee attendees={activity.attendees!} />
</Segment>
```

```
interface Props {
  attendees: Profile[];
}

export default observer(function ActivityListItemAttendee({ attendees }: Props) {
  return (
    <List horizontal>
      {attendees.map(attendee => (
        <List.Item key={attendee.username} as={Link} to={`/profiles/${attendee.username}`}>
          <Image size='mini' circular src={attendee.image || '/assets/user.png'} />
        </List.Item>
      ))}
    </List>
  )
})
```

# Updating details component

```
interface Props {
  attendees: Profile[];
}

export default observer(function ActivityDetailedSidebar ({attendees}: Props) {
  return (
    >
      {attendees.length} {attendees.length === 1 ? 'Person' : 'People'} going
    </Segment>
  )
  <Grid>
    <Grid.Column width={10}>
      <ActivityDetailedHeader activity={activity} />
      <ActivityDetailedInfo activity={activity} />
      <ActivityDetailedChat />
    </Grid.Column>
    <Grid.Column width={6}>      (JSX attribute) Props.attendees: P
      <ActivityDetailedSidebar attendees={activity.attendees!} />
    </Grid.Column>
  </Grid>
  <List relaxed divided>
    {attendees.map(attendee => (
      <Item style={{ position: 'relative' }} key={attendee.username}>
        <Label
          style={{ position: 'absolute' }}
          color='orange'
          ribbon='right'
        >
          Host
        </Label>
        <Image size='tiny' src={attendee.image || '/assets/user.png'} />
        <Item.Content verticalAlign='middle'>
          <Item.Header as='h3'>
            <Link to={`/profiles/${attendee.username}`}>{attendee.dis
        </Item.Content>
      </Item>
    ))
  </List>

```

# Conditional Rendering the Buttons

```
private setActivity = (activity: Activity) => {
  const user = store.userStore.user;
  if (user) {
    activity.isGoing = activity.attendees!.some(
      a => a.username === user.username
    );
    activity.isHost = activity.hostUsername === user.username;
    activity.host = activity.attendees?.find(x => x.username === activity.hostUsername);
  }
  activity.date = new Date(activity.date!);
  this.activityRegistry.set(activity.id, activity);
}
```

```
export interface Activity {
  id: string;
  title: string;
  date: Date | null;
  description: string;
  category: string;
  city: string;
  venue: string;
  hostUsername?: string;
  isCancelled?: boolean;
  isGoing?: boolean;
  isHost?: boolean;
  host?: Profile;
  attendees?: Profile[]
}
```

# ActivityListItem

```
<Item.Description>Hosted by {activity.host?.displayName}</Item.Description>
{activity.isHost && (
    <Item.Description>
        <Label basic color='orange'>
            You are hosting this activity
        </Label>
    </Item.Description>
)}
{activity.isGoing && !activity.isHost && (
    <Item.Description>
        <Label basic color='green'>
            You are going to host this activity
        </Label>
    </Item.Description>
)}
```

# Activity Details Header

```
<Segment clearing attached='bottom'>
  {activity.isHost ? (
    <Button as={Link} to={`/manage/${activity.id}`} color='orange' floated='right'
      Manage Event
    </Button>
  ) : activity.isGoing ? (
    <Button>Cancel attendance</Button>
  ) : (
    <Button color='teal'>Join Activity</Button>
  )}
</Segment>
```

# Adding store method to attend - Async

```
const Activities = {
  list: () => requests.get<Activity[]>('/activities'),
  attend: (id: string) => requests.post<void>(`/activities/${id}/attend`, {})
```

```
export interface IProfile {
  username: string;
  displayName: string;
  image?: string;
  bio?: string;
}

export class Profile implements IProfile {
  constructor(user: User) {
    this.username = user.username;
    this.displayName = user.displayName;
    this.image = user.image
  }

  username: string;
  displayName: string;
  image?: string;
  bio?: string;
}
```

```
const user = store.userStore.user;
this.loading = true;
try {
    await agent.Activities.attend(this.selectedActivity!.id);
    runInAction(() => {
        if (this.selectedActivity?.isGoing) {
            this.selectedActivity.attendees =
                this.selectedActivity.attendees?.filter(a => a.username !== user?.username)
            this.selectedActivity.isGoing = false;
        } else {
            const attendee = new Profile(user!);
            this.selectedActivity?.attendees?.push(attendee);
            this.selectedActivity!.isGoing = true;
        }
        this.activityRegistry.set(this.selectedActivity!.id, this.selectedActivity!)
    })
} catch (error) {
    console.log(error);
} finally {
    runInAction(() => this.loading = false);
}
```

# Hosting

## Dot Net Hosting

<https://codewithmukesh.com/blog/hosting-aspnet-core-with-smarteraspnet/#:~:text=Here%20is%20how%20to%20Host,the%2060%20Days%20FREE%20Trial.>

## React Hosting

<https://medium.com/choreo-tech-blog/deploy-your-react-app-with-choreo-in-just-5-minutes-ca6ac0a0933e>