Social Network WebApplication

Budeme použivat - ASAP.NET + React.JS + MobX

Softwere bude cross-patform taže možeme použit akykolvek budeme chciet (od windowsu cez MacOs až po Linux)

Aplikacie -

Postman - <https://www.postman.com/>

.Net 8.0 - <https://dotnet.microsoft.com/en-us/download>

Note.js - <https://nodejs.org/en/download>

Visual Studio Code - <https://code.visualstudio.com/>

Extensions pre Visual Studio Code

* **SQLite**  ***v0.14.1***
* **NuGet Gallery**  ***v0.0.24***
* **Material Icon Theme**  ***v5.1.0***
* **C#**  ***v2.28.11***

Walking skeleton pre API -

Je to maly system ktory poskytuje end-end funkcionalitu, data budu v našej DB a budme ich zobrazovat cez našu API klientovi

[Social Network WebApplication 1](#_Toc1353188014)

[Vytvorenie dotnet projektu 3](#_Toc1548662640)

[Walking Skeleton - API 3](#_Toc1897184468)

[Domain Entity - 4](#_Toc150757043)

[Pridanie entity do DB framework 5](#_Toc1684249843)

[Pridanie API Controllera 9](#_Toc1757185015)

[React part 10](#_Toc13068925)

[React component 11](#_Toc1312292485)

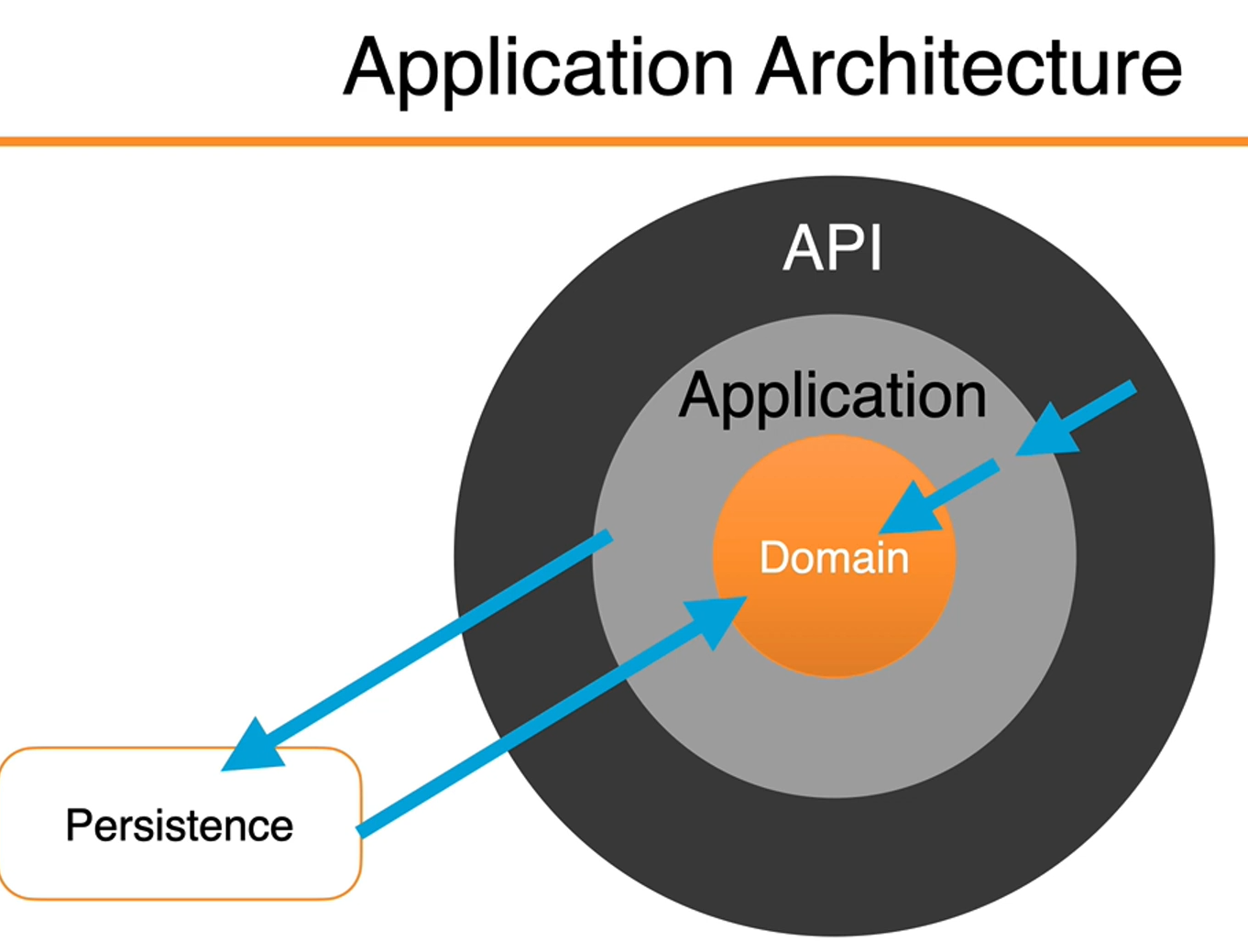
[JSX (JavaScript XML) 14](#_Toc1048698141)

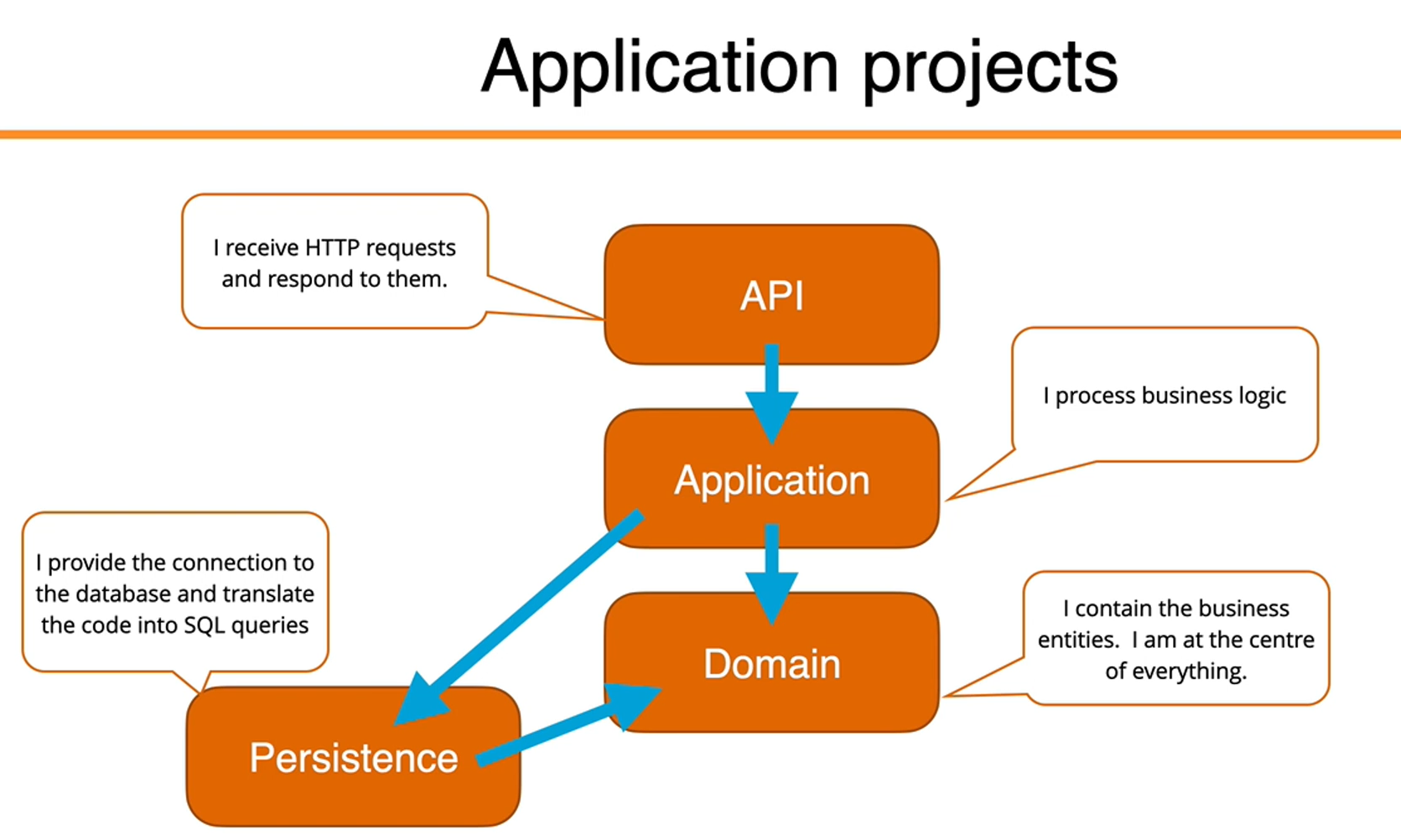
[CORS Policy 19](#_Toc1674089815)

[CRUD application 20](#_Toc637560293)

[Vysvetlenie kódu: 23](#_Toc1813550512)

[Zhrnutie: 24](#_Toc1746685929)





Vytvorenie dotnet projektu

# Walking Skeleton - API

V CMD zadame -

**dotnet new list**

Tento príkaz je užitočný, ak si nie ste istí, ktorú šablónu použiť pre svoj projekt, alebo ak chcete preskúmať dostupné možnosti pred vytvorením nového projektu. Poskytne vám rýchly prehľad šablón spolu s ich krátkymi názvami, podporovanými jazykmi a značkami, aby vám pomohol vybrať vhodnú šablónu pre vaše potreby.

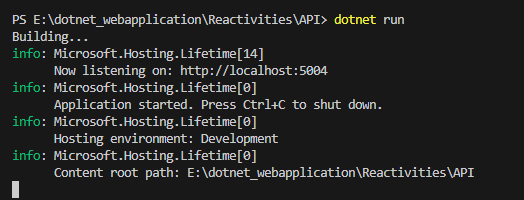
Scriptom - “create-projects-win.ps1”

Tento skript je skript prostredia PowerShell, ktorý sa používa na nastavenie základnej štruktúry projektu .NET pre aplikáciu web API spolu s podpornými knižnicami tried pre aplikačnú logiku, modely domény a perzistenciu údajov.

**dotnet run -**

(Tento príkaz sa bežne používa počas vývoja na rýchlu kompiláciu a testovanie aplikácií .NET bez toho, aby ste museli ručne spúšťať viacero príkazov na zostavenie a spustenie. Je obzvlášť užitočný pre konzolové aplikácie a webové aplikácie ASP.NET Core počas fázy vývoja.)

Otestujeme či naš projekt sa zostavil spravne



**Postman**

Nam bude služit na testovanie našje apliakcie (može sa použit aj swagger)

Pre dalšim pisanim kodu netreba zabudnut v configu nastavit

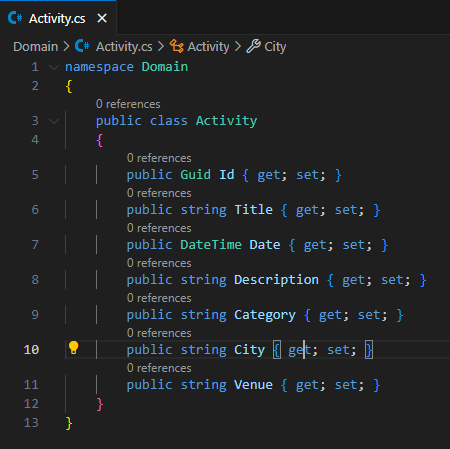
“**<Nullable>disable</Nullable>**”

je prvok XML, ktorý sa používa v súbore projektu alebo v konfiguračnom súbore na určenie toho, že nulovateľné referenčné typy sú pre daný projekt alebo jeho časť zakázané.

Domain Entity -

doménová entita sa zvyčajne vzťahuje na triedu alebo štruktúru, ktorá predstavuje koncept alebo entitu v rámci domény vašej aplikácie. "Doména" sa v tomto kontexte vzťahuje na problémový priestor alebo predmet, ktorým sa vaša aplikácia zaoberá.

Doménové entity sa často používajú pri návrhu riadenom doménou (Domain-Driven Design - DDD), čo je prístup k vývoju softvéru, ktorý kladie dôraz na modelovanie problémovej domény v softvéri, pričom sa zameriava na logiku a správanie domény.



Classa acitivity služi na getovanie a setovanie value ktore sa v nej nachadzaju (pre dalšie použivanie).

## Pridanie entity do DB framework

public class DataContext : DbContext

Táto syntax naznačuje, že trieda DataContext dedí od triedy DbContext (alebo ju rozširuje). V objektovo orientovanom programovaní dedičnosť umožňuje novej triede (odvodenej alebo dcérskej triede) dediť vlastnosti a správanie z existujúcej triedy (základnej alebo rodičovskej triedy).

Danej tirede vytovirme Constructor

public DataContext(DbContextOptions options) : base(options)

{

}

V kontexte konštruktora public DataContext(DbContextOptions options) : base(options) je DbContextOptions trieda v Entity Framework Core, ktorá sa používa na konfiguráciu správania DbContextu. Tento konštruktor sa zvyčajne používa, keď chcete vytvoriť inštanciu vlastnej triedy DbContext.

Options - Tento parameter je inštanciou DbContextOptions. Predstavuje konfiguračné nastavenia, ktoré bude DataContext používať.

Do Porgram.cs

Sme pridali conenction string lebo DB potrebuje conneciton string

builder.Services.AddDbContext<DataContext>(opt =>

{

opt.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection"));

}

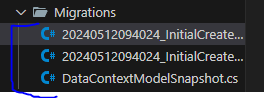
);

Dalej staihneem dotnet of - <https://www.nuget.org/packages/dotnet-ef>

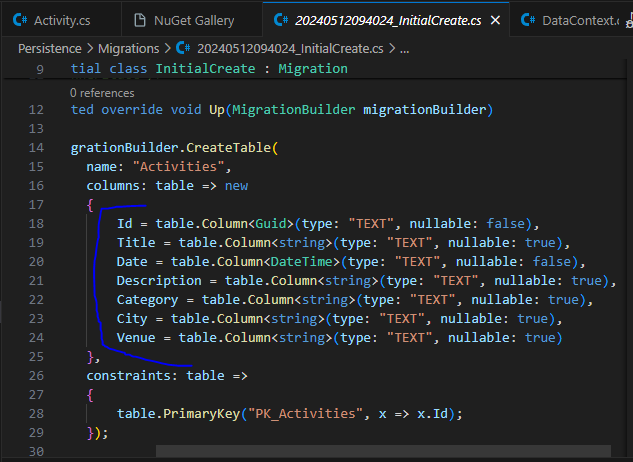
Ktory nam pomože s migraciou našho codu do DB

PS E:\dotnet\_webapplication\Reactivities> dotnet ef migrations add InitialCreate -s API -p Persistence

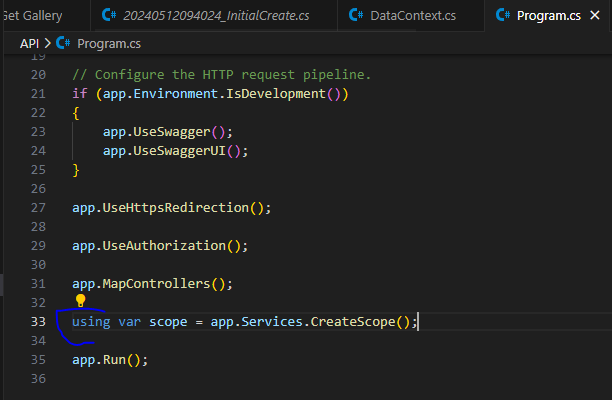
Ak sa pozrieme vytovirlo nam to 3 scripty



Hned v prvom si vieme skontrolovat stplce ktore nam to vytvori na zaklade prop activities ktore sme vytvorili (getere a setere)

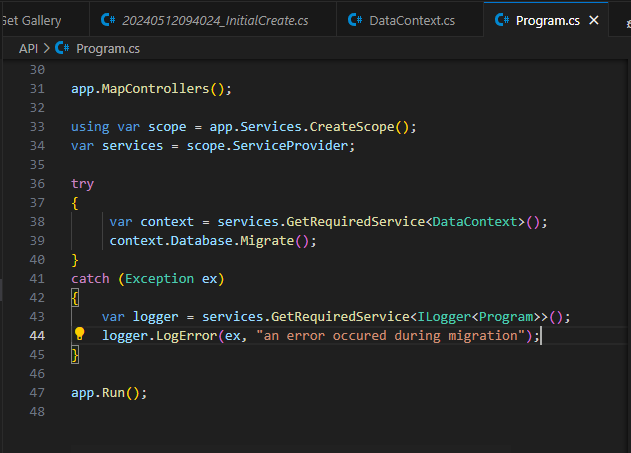
Create database

V clasee Program.cs red spustenym vytvorime



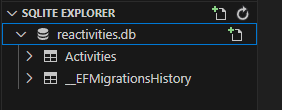
Takže keď vykonáte príkaz var scope = app.Services.CreateScope();, vytvoríte nový scope v kontajneri služieb aplikácie, ktorý potom môžete použiť na riešenie služieb špecifických pre tento scope. Toto sa často používa, keď potrebujete vyriešiť služby v rámci konkrétneho kontextu alebo rozsahu, napríklad v rámci požiadavky v aplikácii ASP.NET Core.

Pridame try catch pre migraciu (ak by počas toho nastal problem)



Potom je fajn použit dotnet watch

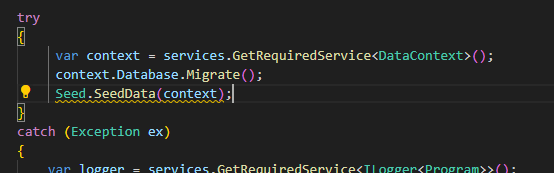
je nástroj príkazového riadka poskytovaný v súbore .NET Core SDK, ktorý umožňuje automatické prebudovanie a reštartovanie aplikácie .NET Core po zistení zmien v zdrojových súboroch. Je obzvlášť užitočný počas vývoja na urýchlenie procesu vývoja tým, že eliminuje potrebu ručného zastavenia, prebudovania a opätovného spustenia aplikácie po vykonaní zmien.



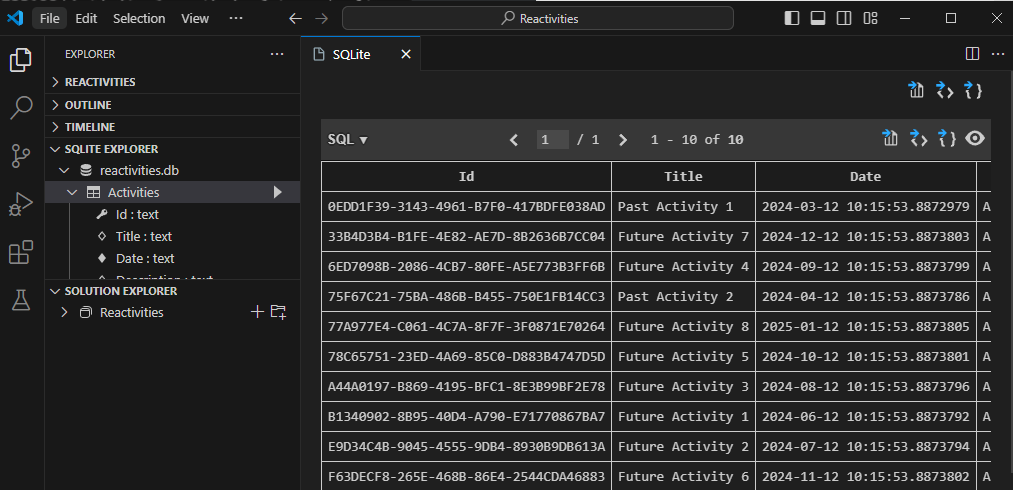
Toto použivame len pre development aby sme videli že to funguje

Seedovanie data je vytovrenie testovacih dat , na tot možeme použi “SeedData.txt”v ktorom sa nachazda cela trieda s tymito datami

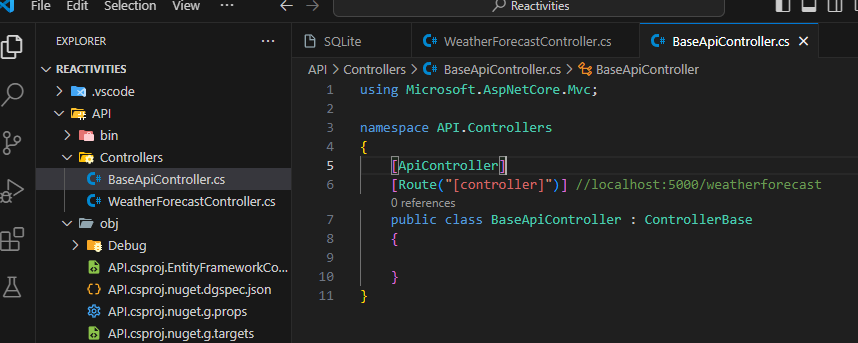
Po migrovani DB rovno do nej hodiem aj tieto Seedovane DATA (toto sme menili v program.cs)



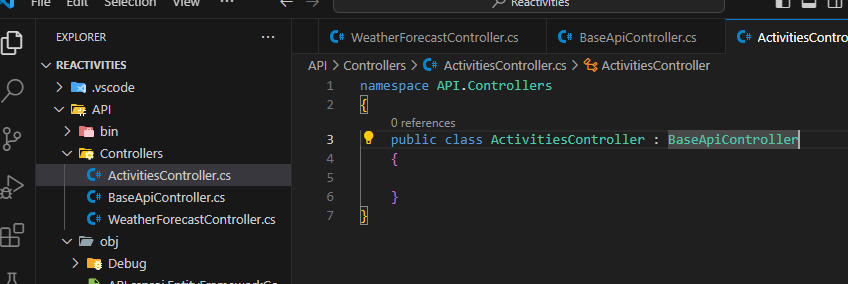
Skontrolovat si to možeme tak že vidime potom že v našej DB sa tieto udaje nachadzaju



## Pridanie API Controllera

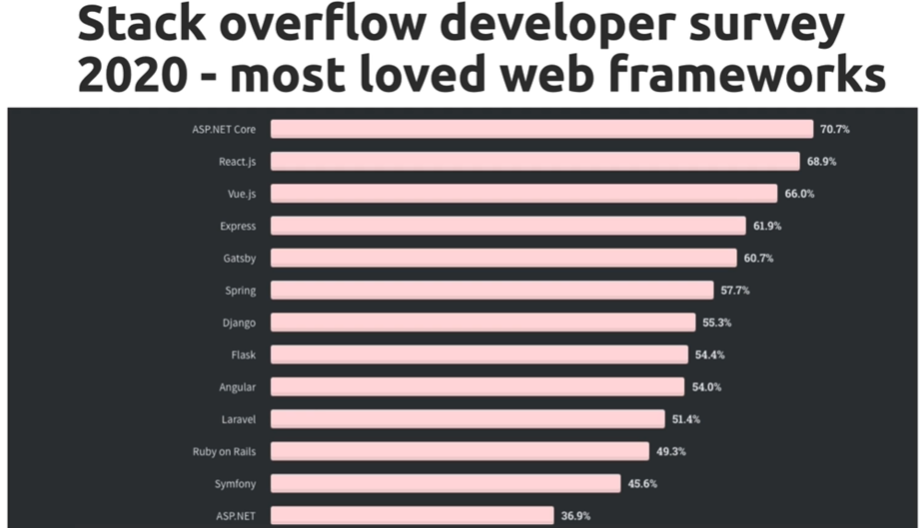
Tym že už mame vytvoreny jeden API contorller tak sme z neho zobrali priamrne časti s tym že dedili sme z ControllerBase

Keď teda vytvoríte novú triedu kontroléra API, ktorá dedí z triedy BaseApiController, automaticky zdedí atribút [ApiController] a šablónu trasy určenú pomocou [Route("[controller]")]. To pomáha udržiavať kód DRY (Neopakuj sa) centralizovaním spoločného správania a konfigurácie kontroléra API.

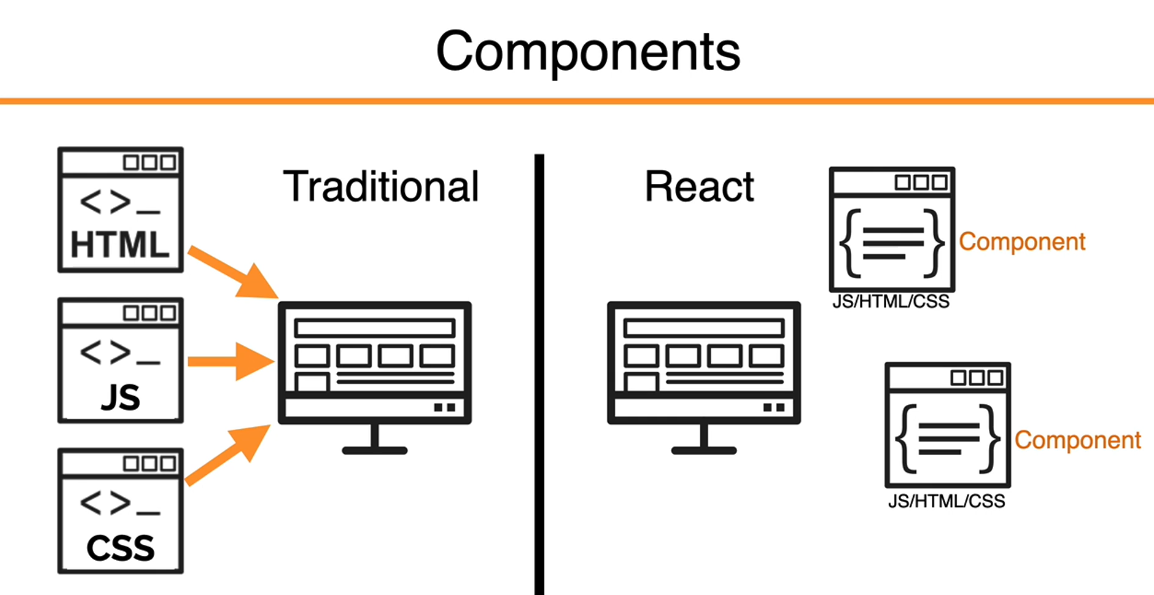
tento kontroler bude dedit z našho teraz vytovrnee controllera

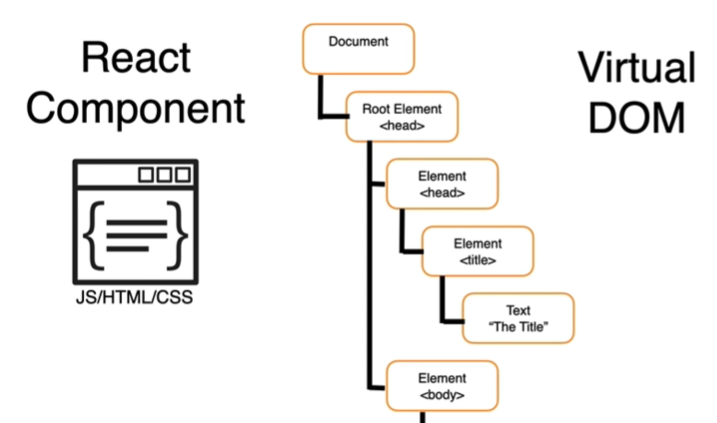
## React part

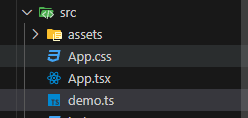
Spravna otazka prečo React ? Velke spoločnosti ako Facebook , netflix ,Airbnb použivaju React. Taktiež react bol hned druhy po ASP.NET. Developeri použivaju react lebo je velmi ryhcli , lahky na učenie.



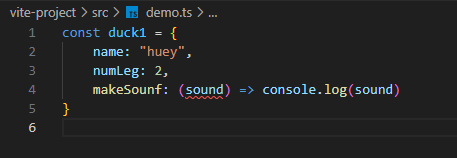
### React component

Virtual DOM

TypeScrpits -



Vytovrenie objektu -



Pre vytovrenie React projektu možmee použit bežnu react stranku

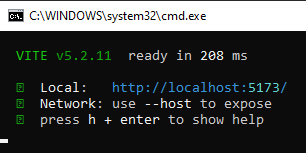
<https://react.dev/learn/start-a-new-react-project>

Avšak my sa zameriame skor na niečo ine a to bude

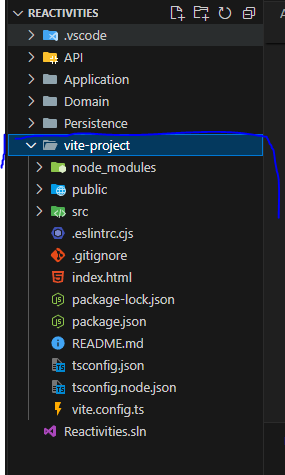
<https://vitejs.dev/guide/>

Ak všetko nainstalujeme podla postupu a spustime “npm run dev”

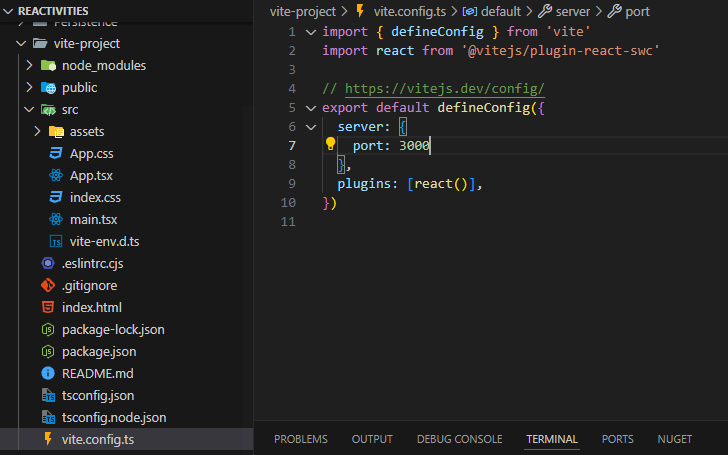
E:\dotnet\_webapplication\.Net---React---Social-Network-WebApplication\Reactivities\vite-project>



Instalalciou nam pribudlo viacero filov



Hned ako prvu zmenu je že vymažene všetky CSS veci s tym že chceme nastavitť aby naša apliakcia bežala na porte “3000” , to nastavime -



JSX (JavaScript XML)

je syntaxová nadstavba pre JavaScript, ktorá sa najčastejšie používa v rámci knižnice React na popis vzhľadu užívateľského rozhrania (UI). JSX umožňuje písať HTML prvky priamo v JavaScripte, čím poskytuje spôsob, ako štruktúrovať renderovacie logiky komponentov v syntaxe podobnej HTML.

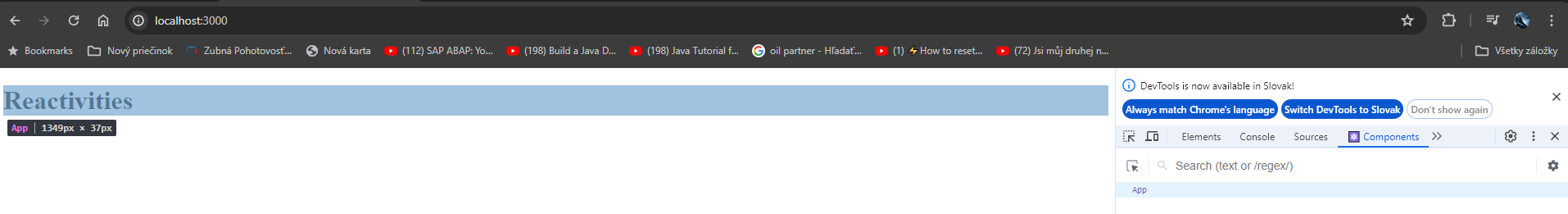
Vidim object a interface



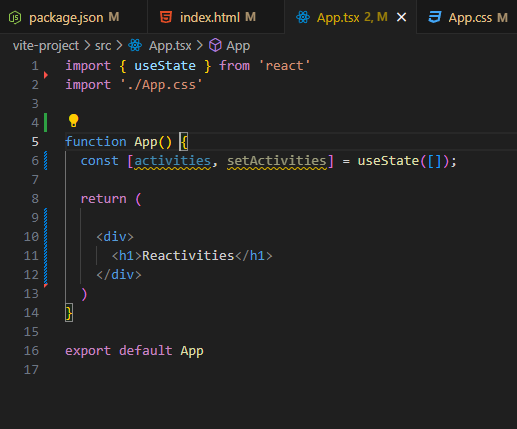
Objekty využivaju template interfacu a nasledne ich bližšie specifikujeme

Dalšie rozširenie ktore nam pomože bude

Budeme vidiet components a dalšie doležite veci

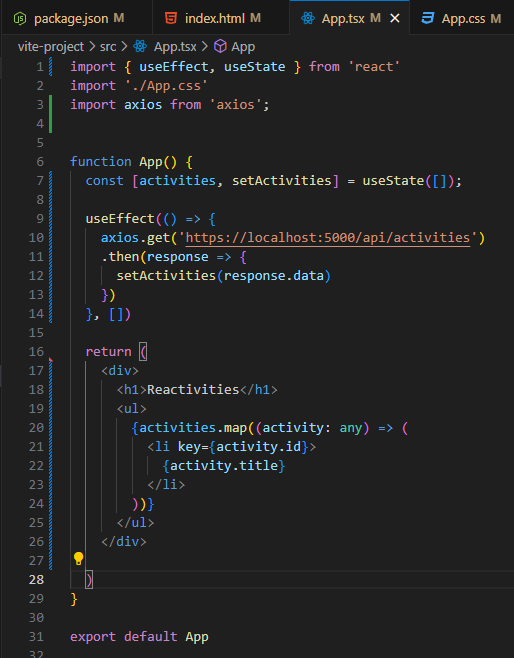
Ziskavanie dát z našej API

Vytovriem si vo funkcii app objekt ktory nam bude zbierat tieot data

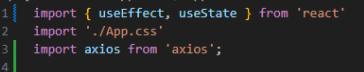


Vdaka tomu si možeme zapsiovat a pamata našu aktivitu, použijeme taktiež

Funkciu “useEffect” je jednou z kľúčových hookov v Reacte, čo je populárna JavaScriptová knižnica pre budovanie používateľských rozhraní. Táto funkcia umožňuje vykonávať vedľajšie účinky (side effects) v funkčných komponentoch. Vedľajšie účinky môžu zahŕňať rôzne operácie ako načítanie dát z API, pripojenie k webovým soketom, aktualizácia DOMu alebo nastavovanie časovačov.



**Importy:**

;

useEffect a useState sú hooky z Reactu.

./App.css je súbor so štýlmi, ktorý sa aplikuje na tento komponent.

axios je knižnica pre HTTP požiadavky.

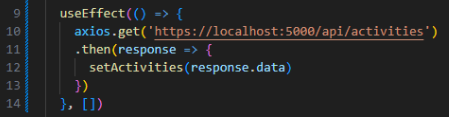
**Definícia komponentu App:**

;

Definuje sa funkčný komponent App.

useState inicializuje stavovú premennú activities ako prázdne pole. setActivities je funkcia na aktualizáciu tejto premennej.

**Použitie useEffect:**

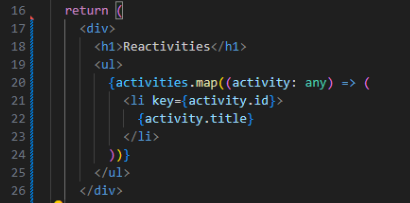
}, [])

useEffect sa spustí po prvej renderi komponentu (prázdne pole závislostí znamená, že sa spustí len raz).

axios.get vykoná HTTP GET požiadavku na URL <https://localhost:5000/api/activities>.

Keď požiadavka úspešne dokončí, vráti odpoveď (response), z ktorej sa získa dátová časť (response.data) a tá sa uloží do activities pomocou setActivities.

**Renderovanie komponentu:**

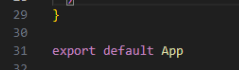


Komponent renderuje div s nadpisom Reactivities.

Potom renderuje ul (neusporiadaný zoznam), ktorý obsahuje položky zoznamu (li).

Pre každú aktivitu v activities sa vytvorí li s kľúčom activity.id a textom activity.title.

**Exportovanie komponentu:**



App je exportovaný ako predvolený export z modulu, čo znamená, že ho možno importovať a použiť v iných častiach aplikácie.

**Shrnutie**

Tento kód definuje komponent, ktorý:

Načíta dáta z API pri prvom zobrazení komponentu pomocou axios.

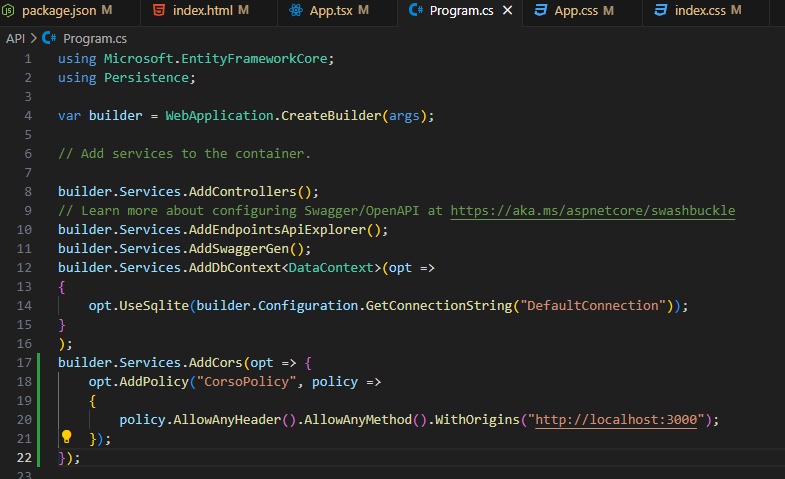
Uloží načítané dáta do stavovej premennej activities.

Vykreslí zoznam týchto aktivít na stránku.

Je to typický príklad toho, ako v Reacte vykonávať vedľajšie účinky (ako načítanie dát z API) a ako spravovať a zobrazovať stav komponentu.

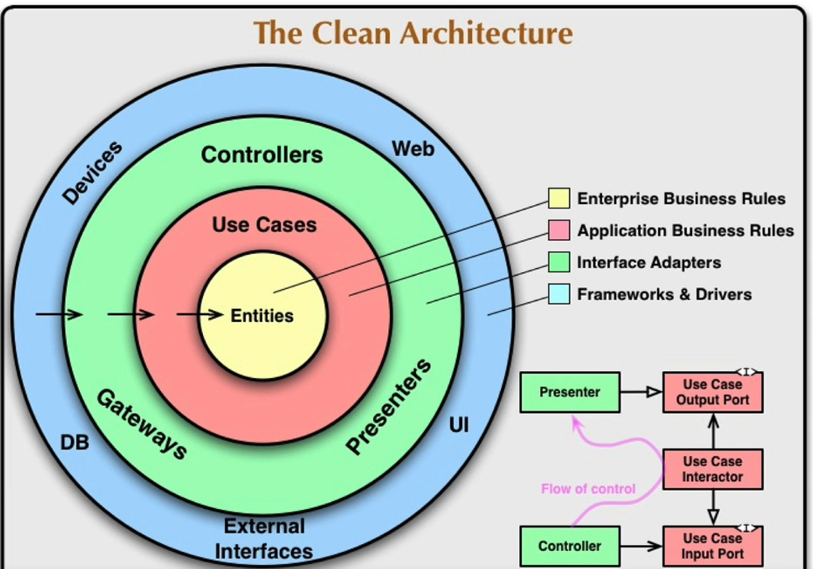
### CORS Policy

Nasledne treba pridat CORS policy aby sa nam neblokovala stranka



# CRUD application

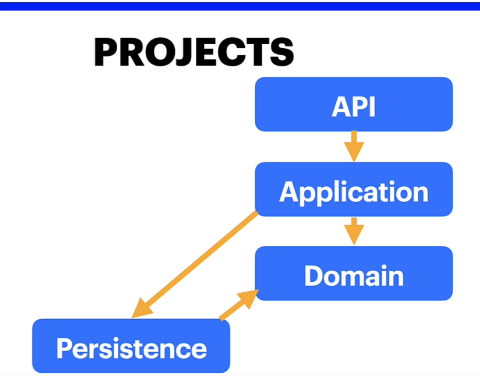
CRUD je skratka, ktorá sa často používa v oblasti vývoja softvéru, a znamená Create (Vytvoriť), Read (Čítať), Update (Aktualizovať) a Delete (Zmazať). Tieto štyri základné operácie sú kľúčové pri práci s dátami v databázových aplikáciách. V kontexte .NET aplikácií sa CRUD operácie často implementujú pomocou rôznych technológií a frameworkov poskytovaných .NET ekosystémom.

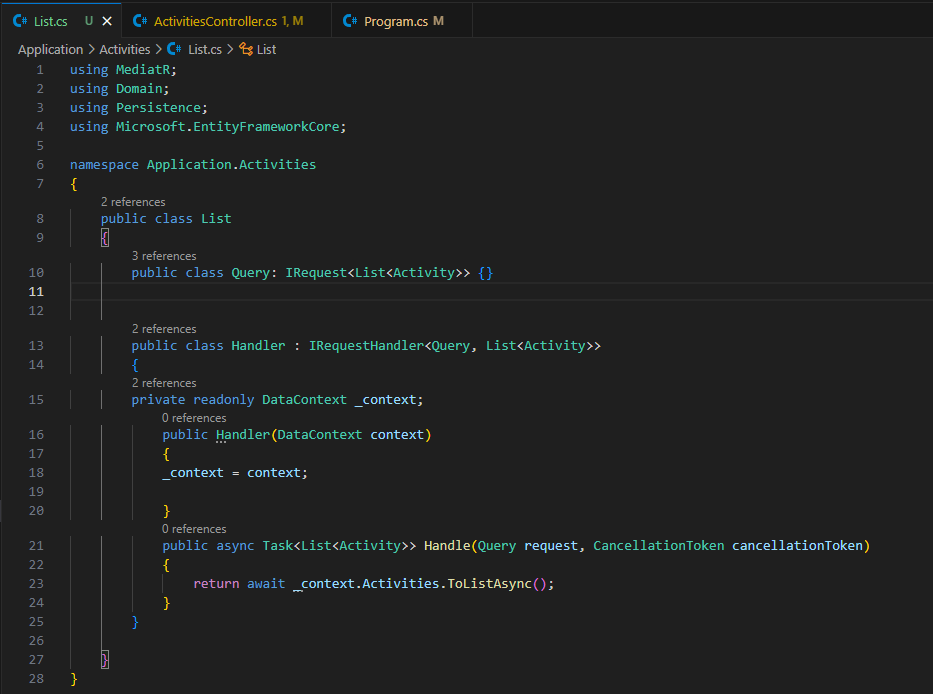
CQRS

(Command Query Responsibility Segregation) je architektonický vzor, ktorý oddeľuje operácie čítania (query) a zápisu (command) do samostatných modelov. Tento prístup sa často používa na zvýšenie škálovateľnosti a výkonu aplikácie tým, že umožňuje optimalizáciu oboch typov operácií nezávisle od seba.

V kontexte vývoja webových aplikácií v .NET môže byť implementácia CQRS rozdelená na nasledujúce kroky:

1. Definovanie príkazov (Commands): Príkazy reprezentujú operácie, ktoré menia stav systému. Každý príkaz by mal byť jedinečnou akciou, ako napríklad vytvorenie nového používateľa, aktualizácia objednávky alebo zmazanie záznamu.
2. Definovanie dotazov (Queries): Dotazy sú operácie, ktoré načítavajú údaje z databázy bez toho, aby menili jej stav. Napríklad, získanie zoznamu používateľov alebo detailov objednávky.
3. Spracovatelia príkazov (Command Handlers): Spracovatelia sú zodpovední za vykonávanie príkazov. Každý príkaz by mal mať svojho spracovateľa, ktorý vykoná potrebné akcie na zmenu stavu systému.
4. Spracovatelia dotazov (Query Handlers): Spracovatelia sú zodpovední za spracovanie dotazov a načítanie údajov z databázy.
5. Použitie MediatR na mediáciu medzi príkazmi/dotazmi a spracovateľmi: Knižnica MediatR je v .NET často používaná na implementáciu CQRS, pretože poskytuje jednoduchý spôsob na mediáciu medzi príkazmi/dotazmi a ich spracovateľmi.



CQRS (Command Query Responsibility Segregation) v .NET, konkrétne v kontexte aplikácie používajúcej MediatR na mediáciu medzi požiadavkami a spracovateľmi. Tento kód konkrétne definuje dotaz (Query) na získanie zoznamu aktivít a jeho spracovateľa (Handler).

### **Vysvetlenie kódu:**

1. **Using Statements**:

Tieto direktívy importujú potrebné namespaces. **MediatR** je knižnica, ktorá umožňuje mediáciu medzi požiadavkami a spracovateľmi. **Domain** a **Persistence** sú pravdepodobne namespaces definované v projekte, ktoré obsahujú entity a kontext databázy. **Microsoft.EntityFrameworkCore** je použité pre prácu s Entity Framework Core, ORM pre prístup k databáze.

1. **Definícia triedy List v namespace Application.Activities**:

Táto trieda obsahuje definíciu dotazu a jeho spracovateľa pre získanie zoznamu aktivít.

1. **Definícia dotazu (Query)**:

Trieda **Query** dedí z **IRequest<List<Activity>>**, čo znamená, že tento dotaz bude očakávať odpoveď typu **List<Activity>**. Tento dotaz neobsahuje žiadne vlastnosti, pretože jeho účelom je jednoducho získať všetky aktivity.

1. **Definícia spracovateľa (Handler)**:
   * **Handler** implementuje **IRequestHandler<Query, List<Activity>>**, čo znamená, že spracováva dotazy typu **Query** a vracia zoznam aktivít (**List<Activity>**).
   * **DataContext** je kontext databázy, ktorý je injektovaný do spracovateľa pomocou konštruktora. Tento kontext je pravdepodobne definovaný v **Persistence** namespace a je použitý na prístup k databázovým údajom.
   * **Handle** metóda je implementovaná na spracovanie dotazu. V tomto prípade jednoducho načíta všetky aktivity z databázy a vráti ich ako zoznam (**List<Activity>**).

### **Zhrnutie:**

Tento kód definuje dotaz a jeho spracovateľa pre načítanie zoznamu aktivít z databázy pomocou CQRS vzoru v .NET aplikácii. **Query** je požiadavka, ktorá očakáva zoznam aktivít, a **Handler** je zodpovedný za spracovanie tejto požiadavky a vrátenie výsledkov. Použitie MediatR zjednodušuje mediáciu medzi požiadavkami a spracovateľmi, čo vedie k lepšiemu oddeleniu zodpovedností a čitateľnosti kódu.