**Section 1:**

**Setup the VS code**

**Theme**

*Dracula*

**Add-on**

*Angular language service*

*Auto Close Tag*

*Auto Rename Tag*

*Bracket Pair Colorizer 2*

*C#*

*C# Extensions*

*Es7 React/ Redux/ GraphQL/ React-Native snippets*

*Material Icon Theme*

*NugetPackage Manager*

*Prettier - code Formatter*

*SQLite*

*GitLens*

**DotNet CLI Command**

***dotnet --infor*** : hiển thị các vesion của dotnet core được cài vào máy và version nào đang được sử dụng

***dotnet -h*** : hiển thị tất cả các lệnh dotnet CLI

***dotnet new -h*** : hiển thị chỉ thị help cho lệnh dotnet new (hiển thị tất cả các loại template project mà ta muốn tạo).

***dotnet new sln*** : Tao file solution cho project nếu không chỉ ra tên thì tên solution sẽ là tên mặc đinh thư mục chứ file solution.

Ví dụ:

**C:\dev\ReactivitiesDemo> *dotnet new sln*** => **ReactivitiesDemo.sln**

**C:\dev\ReactivitiesDemo> *dotnet new sln -n TestingSolution*** => **TestingSolution.sln**

***dotnet new classlib -n Domain*** : Tạo một class library tên là Domain

***dotnet new classlib -n Application***: Tạo một class library tên là Application

***dotnet new classlib -n Persistence*** : Tạo một class library tên là Persistence

***dotnet new webapi -n API*** : Tạo một project Web API

***dotnet sln add .\Domain\*** : Add project Domain vào solution hiện tại (**ReactivitiesDemo.sln**)

***dotnet sln add .\Application\*** : Add project Application vào solution hiện tại (**ReactivitiesDemo.sln**)

***dotnet sln add .\Persistence\*** : Add project Persistence vào solution hiện tại (**ReactivitiesDemo.sln**)

***dotnet sln add .\API\*** : Add project API vào solution hiện tại (**ReactivitiesDemo.sln**)

***dotnet sln list* :** Hiển thị hết các project được reference vào solution hiện tại.



Add các reference và dependency của các solution với nhau:

**API**

-> Application

**Application**

-> Domain  
-> Persistence

**Persistence**

-> Domain

**C:\dev\ReactivitiesDemo\Application> *dotnet add reference ..\Domain\*** : Add reference của **Domain** vào **Application**

**C:\dev\ReactivitiesDemo\Application>** ***dotnet add reference ..\Persistence\*** :Add reference của **Persistence** vào **Application**

**C:\dev\ReactivitiesDemo\API>** ***dotnet add reference ..\Application\*** : Add reference của **Application** vào **API**

**C:\dev\ReactivitiesDemo\Persistence>** ***dotnet add reference ..\Domain\*** : Add reference của **Domain** vào **Persistence**

**C:\dev\ReactivitiesDemo>** ***code .*** : Mở tất cả project bằng VS Code

**C:\dev\ReactivitiesDemo>** ***dotnet run -p .\API\*** : Chạy project API

Lưu ý từ version dotnet core 3.0 trở lên ta phải cài vào dotnet tool.

**dotnet tool install --global dotnet-ef --version 3.0.0**

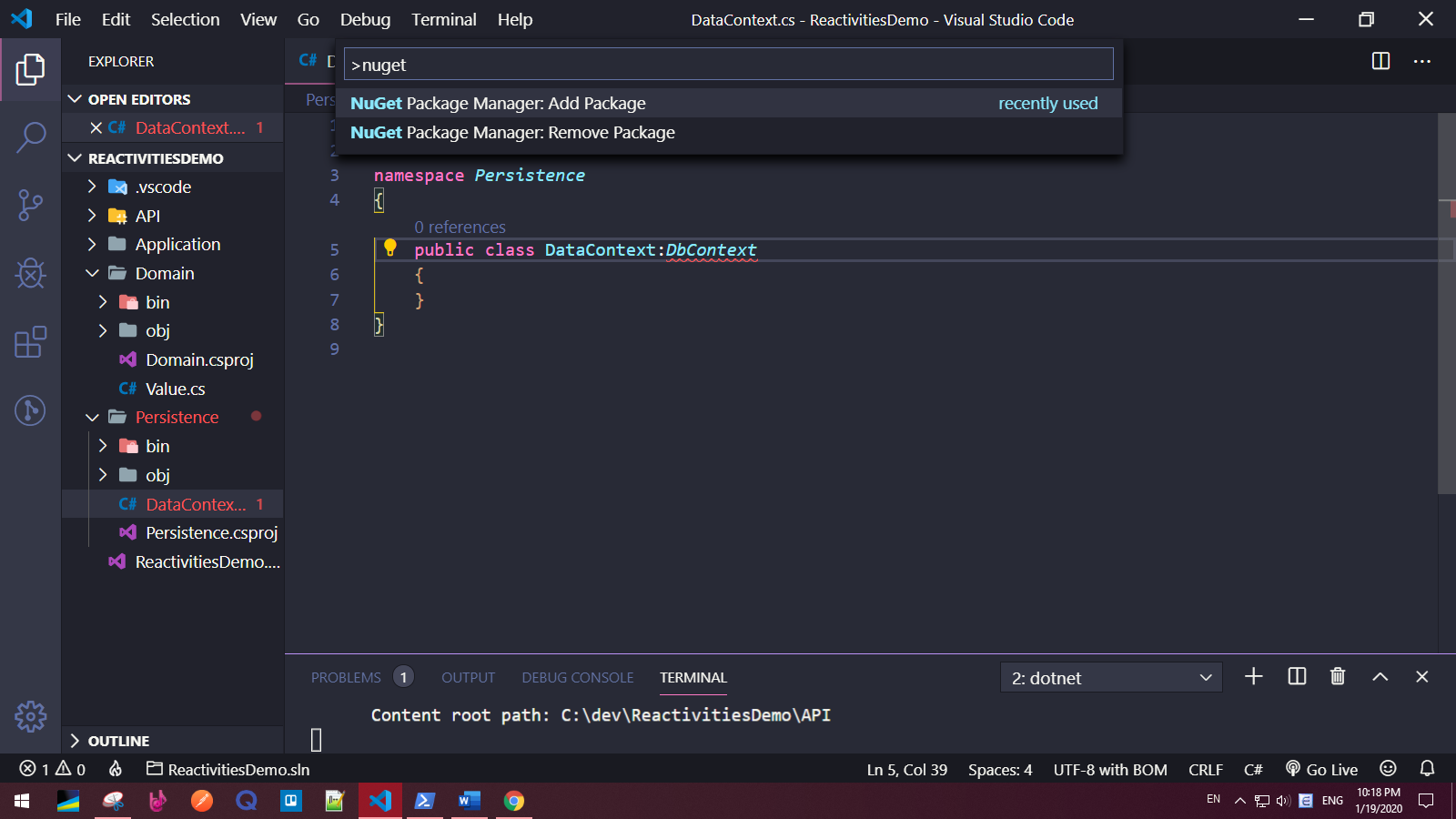
Để update

**dotnet tool update --global dotnet-ef --version 3.1.0**

Hoặc lệnh sau để update or reinstall bản latest

**dotnet tool update --global dotnet-ef**

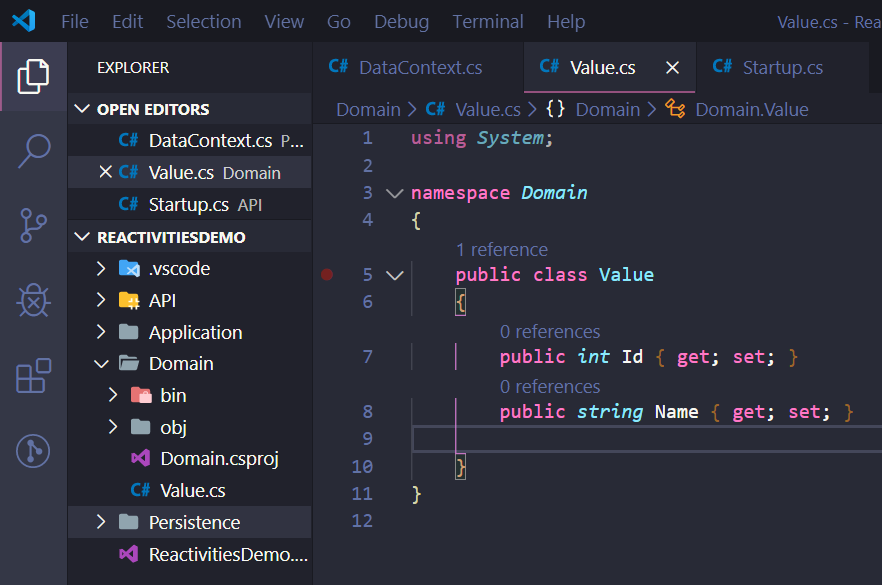
Để add thêm một NuGet Package trong VS Code ta nhấn CTR + SHIFT +P



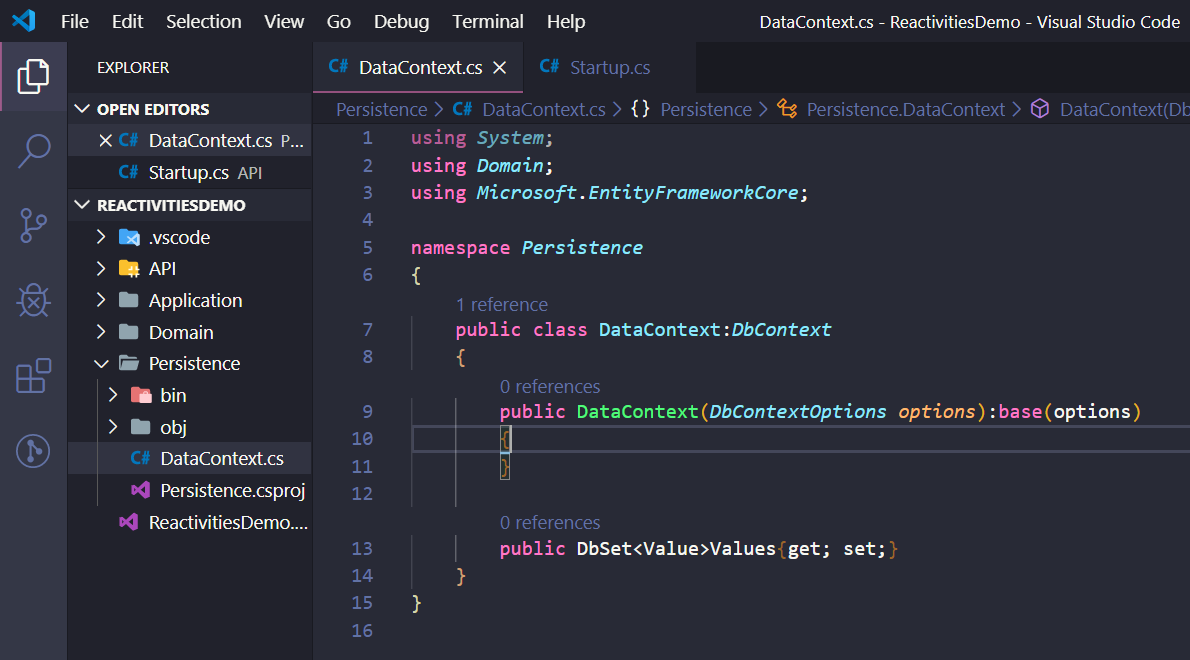
Sau đó ta gõ tên package muôn thêm vào sau đó chọn version và project muốn thêm vào.

Xây dựng entity trong Domain

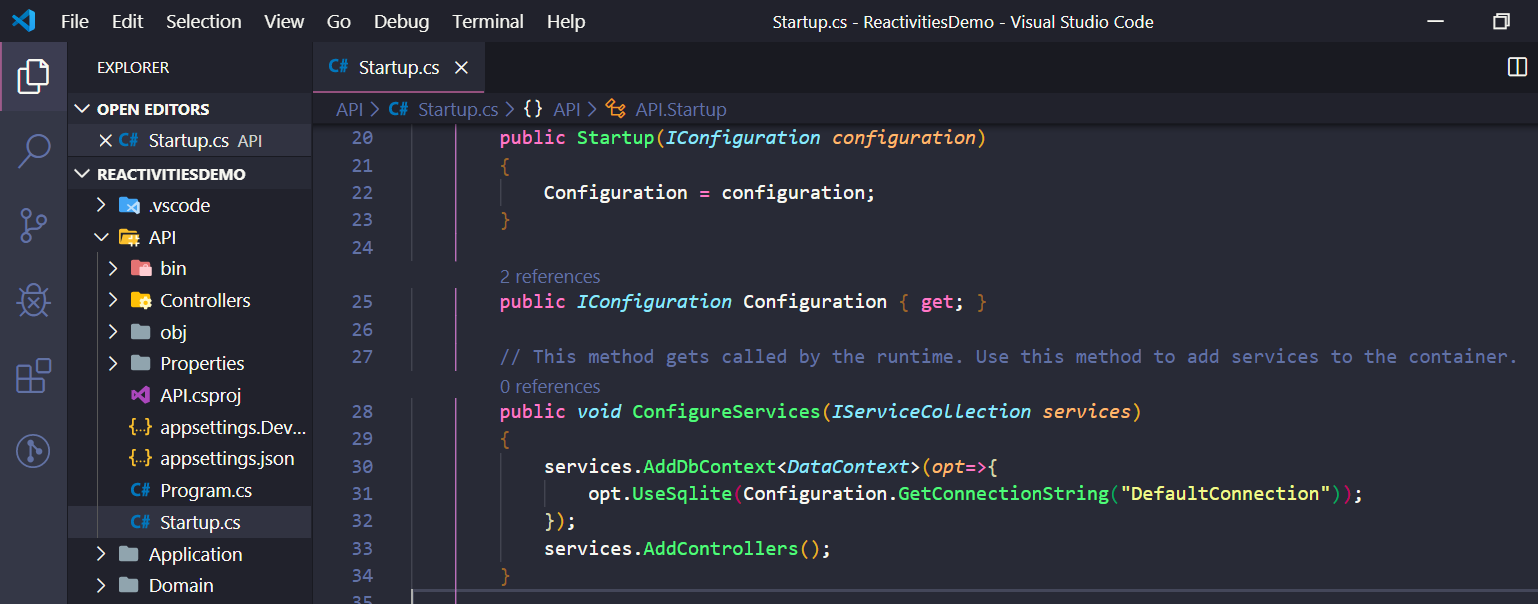
**Ta thêm một class tên là Value như sau:**



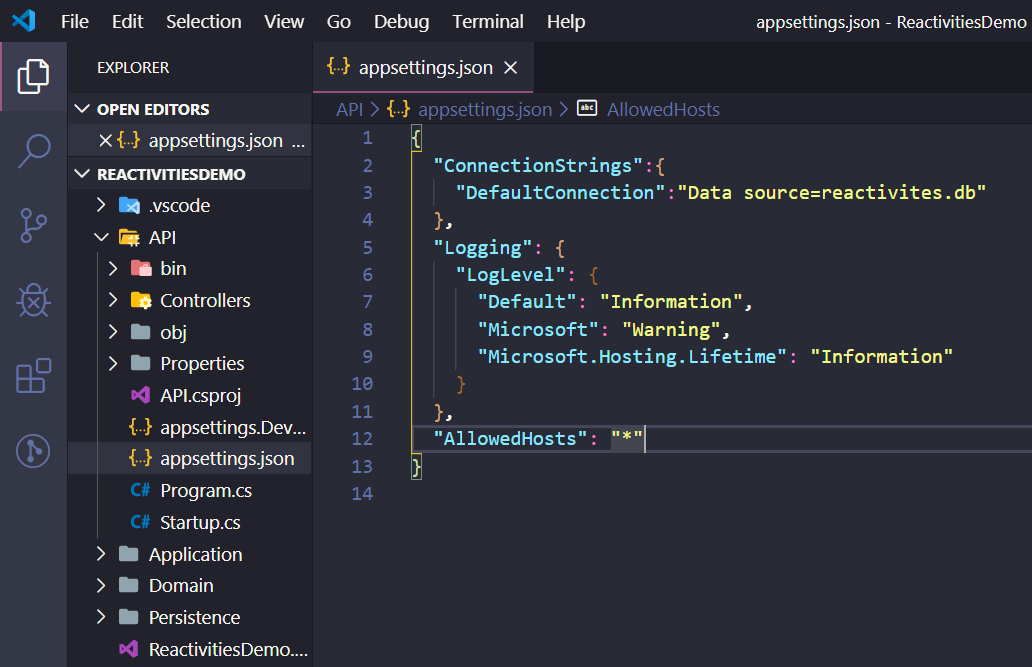
Tạo đối tượng DbContext trong Persistance như sau:



Trong Startup.cs của API ta đăng ký service DbContext như sau:



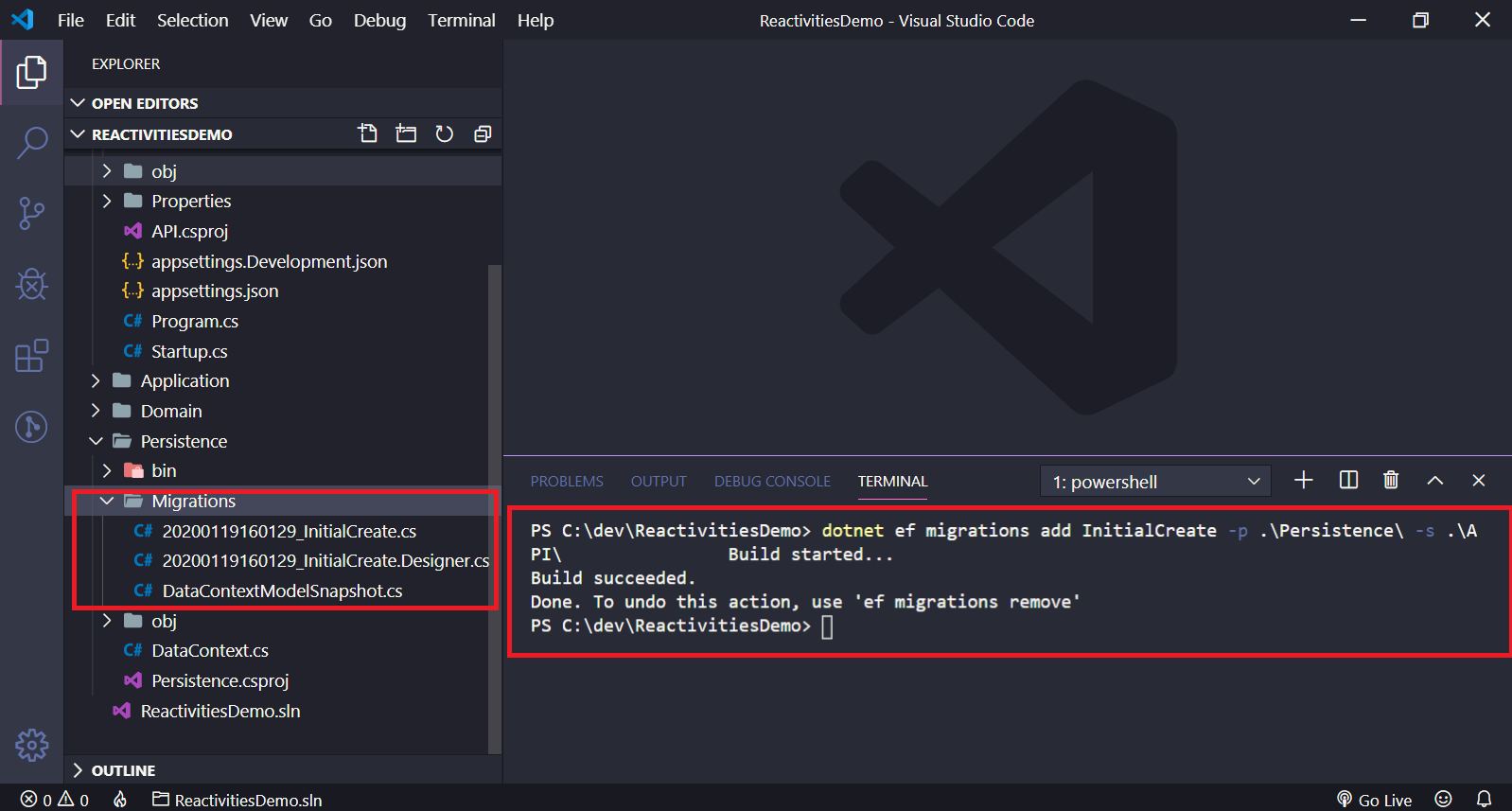
Ta khai báo connectionstring trong appsettings.jsong của API như sau:



Để add một migration đầu tiên ta gõ lệnh sau:

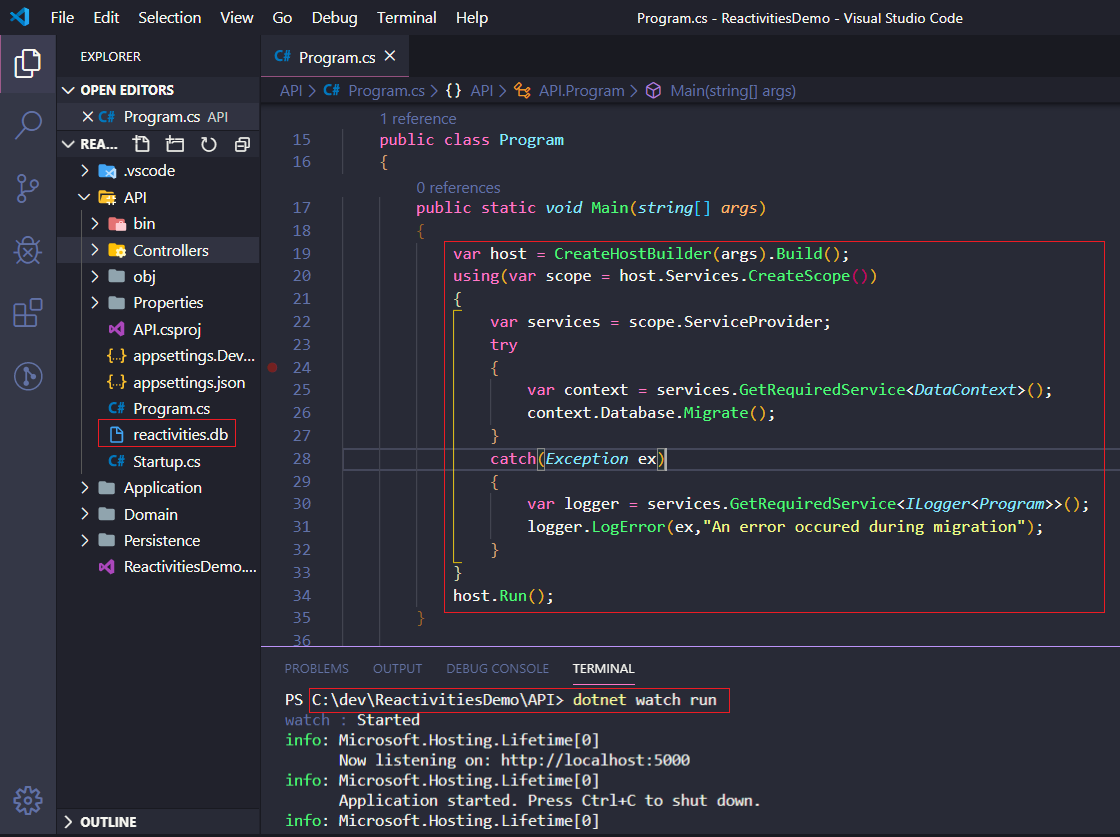
**dotnet ef migrations add InitialCreate -p .\Persistence\ -s .\API\**

Kết quả sẽ được như sau:

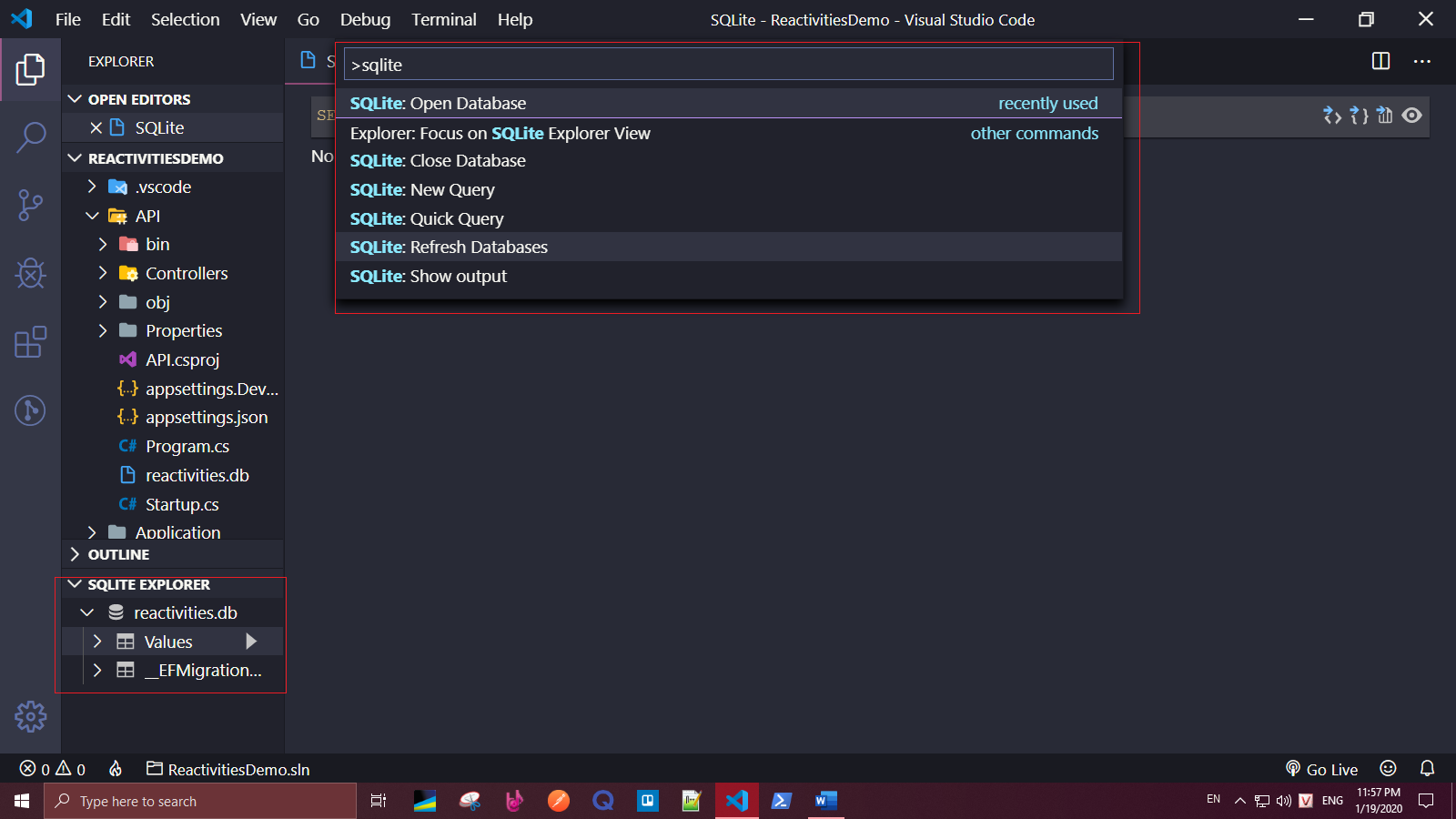


Thay vì dùng lệnh update database trong EF Core ta có thể dùng cách gọi method Migrate trong project startup như sau:

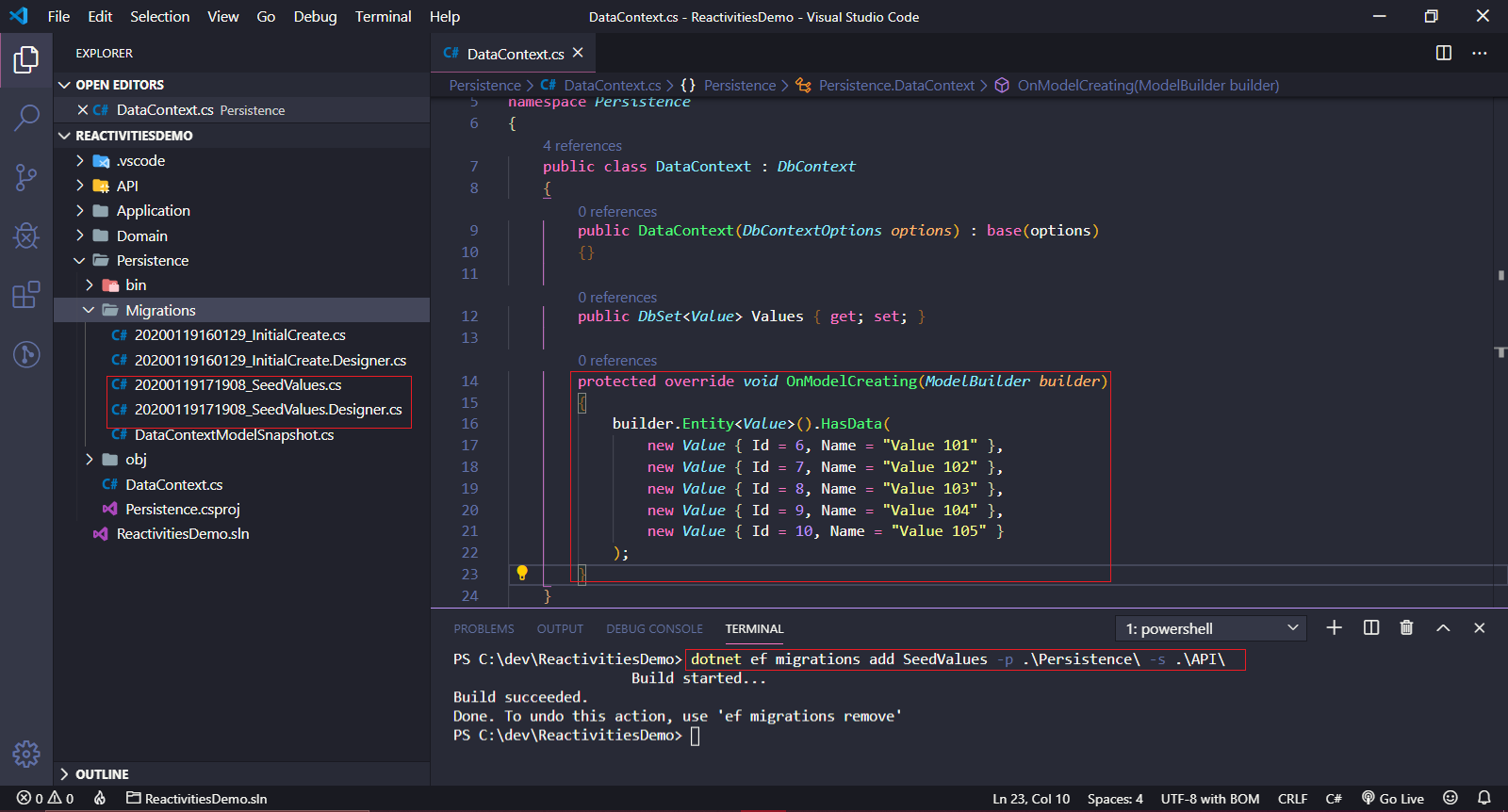
Với method này nó sẽ kiểm tra nếu database chưa có nó sẽ tạo mới.



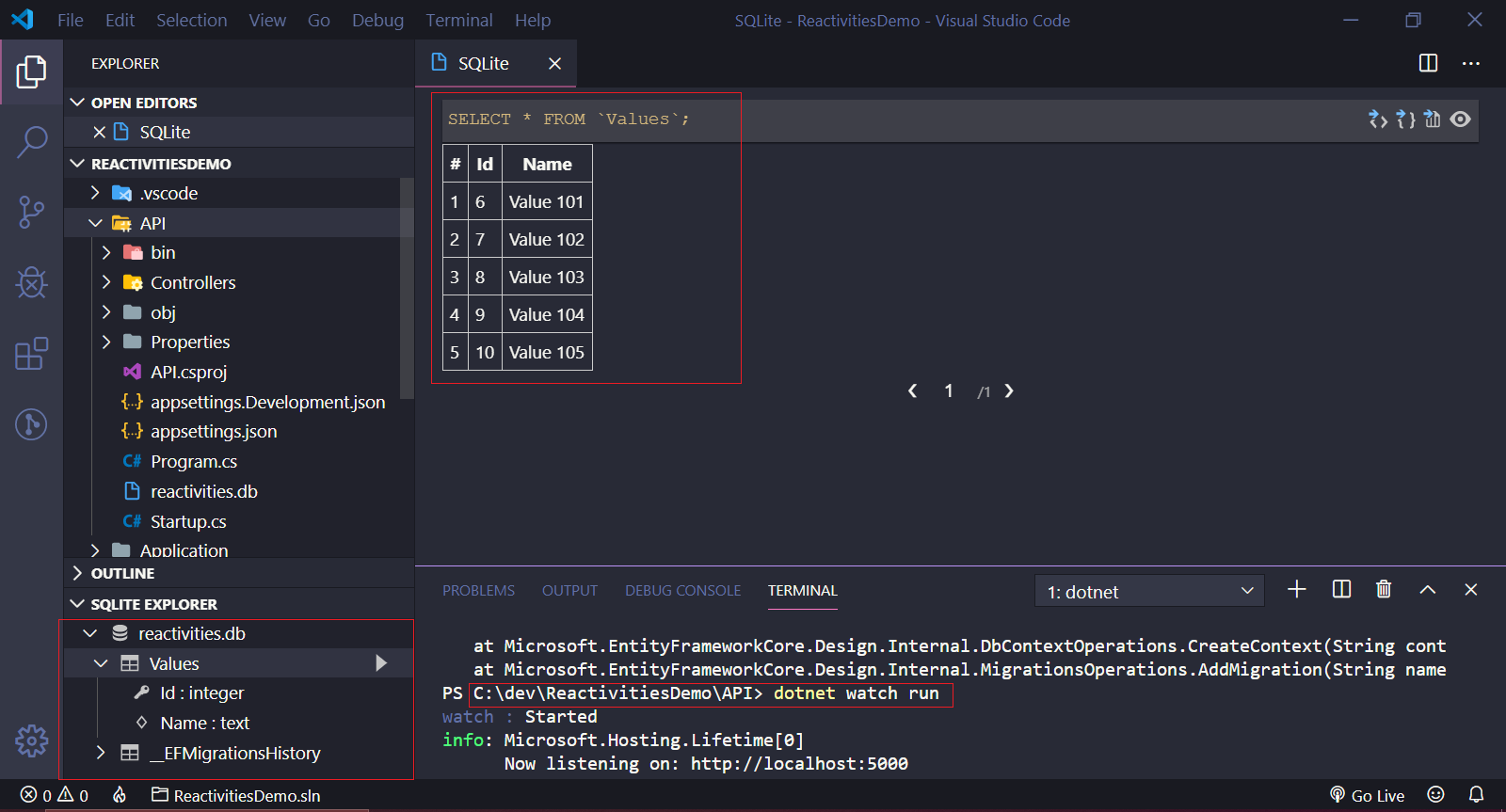
Để xem một database trong SQLite ta làm như sau:



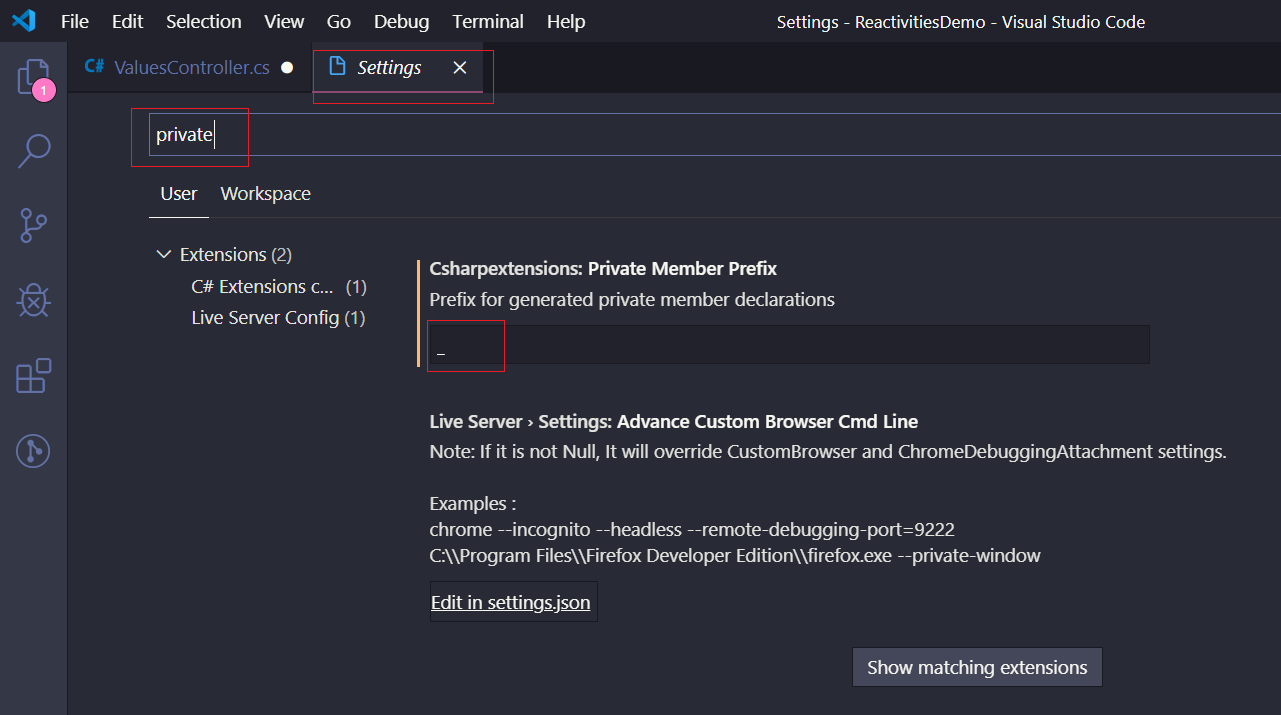
Seeding data using EF fluent configuration



Để execute migration cho việc seeding data ta làm như sau:



Để setup cho việc tự động tạo private member theo đúng code convention trong VS Code ta làm như sau:



Thêm một controller đơn giản vào API

namespace *API*.*Controllers*

{

    [*Route*("api/[controller]")]

    [*ApiController*]

    public class ValuesController : *ControllerBase*

    {

        private readonly *DataContext* \_context;

        public ValuesController(*DataContext* *context*)

        {

            \_context = context;

        }

        // GET api/values

        [*HttpGet*]

        public async *Task*<*ActionResult*<*IEnumerable*<*Value*>>> Get()

        {

            var values = await \_context.Values.ToListAsync();

            return Ok(values);

        }

        // GET api/values/5

        [*HttpGet*("{id}")]

        public async *Task*<*ActionResult*<*Value*>> Get(*int* *id*)

        {

            var value = await \_context.Values.FindAsync(id);

            return Ok(value);

        }

        // POST api/values

        [*HttpPost*]

        public *void* Post([*FromBody*] *string* *value*)

        {

        }

        // PUT api/values/5

        [*HttpPut*("{id}")]

        public *void* Put(*int* *id*, [*FromBody*] *string* *value*)

        {

        }

        // DELETE api/values/5

        [*HttpDelete*("{id}")]

        public *void* Delete(*int* *id*)

        {

        }

    }

}

**Section 2:**

**Create the Client**

Tạo một project react bằng npm và hỗ trợ typescript.

**C:\dev\ReactivitiesDemo>** ***npx create-react-app client-app --use-npm –typescript***

Gõ **npm start** để start client app

**Overview of Typescript**

let data : *string* | *number*;

// data sẽ có kiểu string hoặc number

interface *ICar*{

    color: *string*;

    model: *string*;

    topSpeed?:*number*;

    // Thuộc tính này có cũng được mà không có cũng không sao.

}

// Cả hai khai báo sau điều hợp lệ

const car1: *ICar* = {

    color:'blue',

    model: 'BMW'

}

const car2: *ICar* = {

    color: 'red',

    model: 'Mercedes',

    topSpeed: 100

}

// Trả về kiểu number

const multiply = (*x*:*number*, *y*:*number*) => {

    return (x\*y);

}

// Trả về kiểu string

const multiply2 = (*x*:*number*, *y*:*number*):*string* => {

    return (x\*y).toString();

}

// Trả về kiểu void

const multiply3 = (*x*:*number*, *y*:*number*):*void* => {

   (x\*y);

}

// Nhưng Typescript nó ngầm định hiểu kiểu trả về do vậy ta

// sẽ không cần chỉ ra kiểu trả về là void khi không có

// chỉ thị return

const multiply4 = (*x*:*number*, *y*:*number*) => {

    (x\*y);

}

**Addind the first component in react**

**Step 1:** Ta mở file App.tsx

**Step 2:** Ta thêm đoạn code như sau

**Step 3:** Ta thực hiện run và sẽ được kết quả như sau:

import React,{Component} from 'react';

import logo from './logo.svg';

import './App.css';

class App extends *Component* {

  // Khi load lần đầu tiên thì state sẽ rỗng.

  state = {

    values:[]

  }

  // Sau khi component được mount thì nó sẽ được gán giá trị

  componentDidMount(){

*this*.setState({

      values: [{id:1, name:'Value 101'},{id:2,name:'Value 102'}]

    })

  }

  render(){

    return (

      <div *className*="App">

        <header *className*="App-header">

          <img *src*={logo} *className*="App-logo" *alt*="logo" />

          <ul>

            {*this*.state.values.map((*value*:*any*)=>(

              <li>{value.name}</li>

            ))}

          </ul>

        </header>

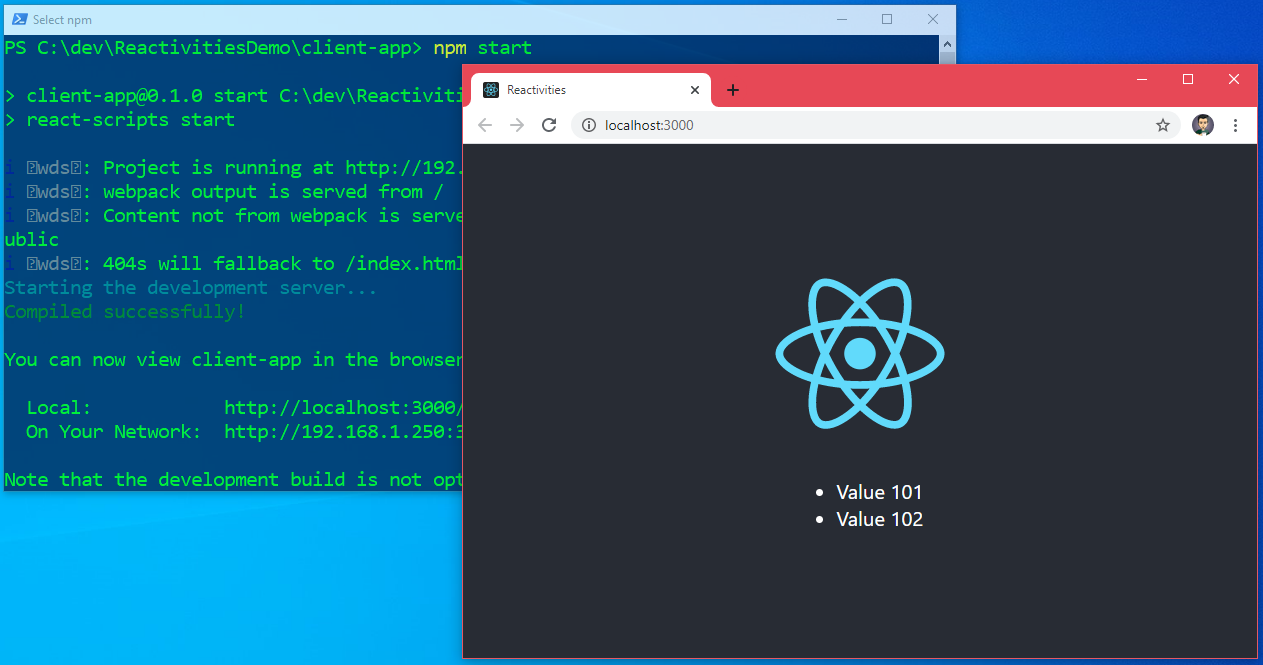
      </div>

    );

  }

}

export default App;



**Fetching data from the API**

**Step 1**: Install axios để dùng http request của thằng này (cài đặt vào project client)

**PS C:\dev\ReactivitiesDemo\client-app> npm install axios**

**Step 2**: Ta thêm các dòng code sau

import React,{Component} from 'react';

import logo from './logo.svg';

import './App.css';

import axios from 'axios';

class App extends *Component* {

  state = {

    values:[]

  }

  componentDidMount(){

    axios.get('http://localhost:5000/api/values')

    .then((*response*)=>{

*console*.log(response);

*this*.setState({

        values: response.data

      })

    })

  }

  render(){

    return (

      <div *className*="App">

        <header *className*="App-header">

          <img *src*={logo} *className*="App-logo" *alt*="logo" />

          <ul>

            {*this*.state.values.map((*value*:*any*)=>(

              <li>{value.name}</li>

            ))}

          </ul>

        </header>

      </div>

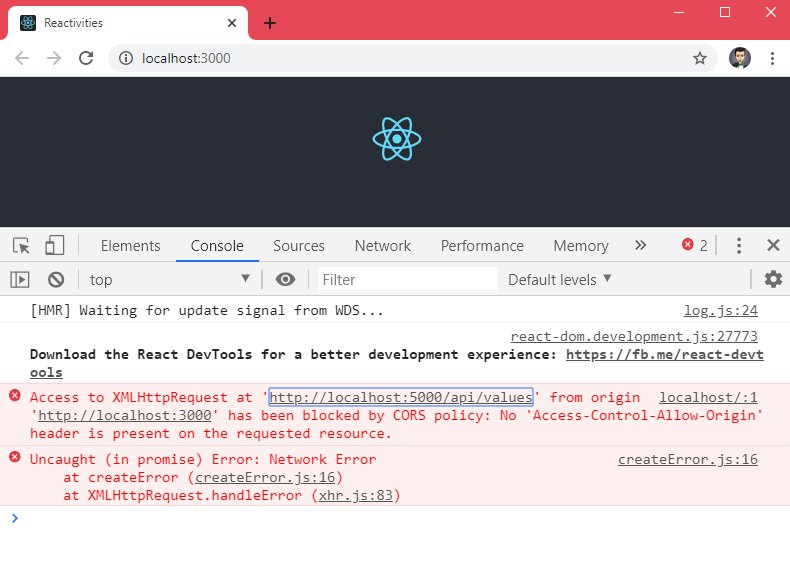
    );

  }

}

export default App;

**Step 3**: Khi ta thực thi đoạn chương trình, ta sẽ **thấy đoạn lỗi như sau** và ta sẽ resolve nó trong phần kế tiếp.



**Adding CORS support to the API**

Step 1: Trong file Startup.cs của solution API ta thêm 2 đoạn code sau:

public class Startup

{

    public Startup(*IConfiguration* *configuration*)

    {

        Configuration = configuration;

    }

    public *IConfiguration* Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.

    public *void* ConfigureServices(*IServiceCollection* *services*)

    {

        services.AddDbContext<*DataContext*>(*opt*=>{

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

        });

        // Begin of Adding CORS Support to the API

        services.AddCors(*opt*=>{

            opt.AddPolicy("CorsPolicy",*policy*=>{

                policy.AllowAnyHeader().AllowAnyMethod().WithOrigins("http://localhost:3000");

            });

        });

        // End of Adding CORS Support to the API

        services.AddControllers();

    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

    public *void* Configure(*IApplicationBuilder* *app*, *IWebHostEnvironment* *env*)

    {

        if (env.IsDevelopment())

        {

            app.UseDeveloperExceptionPage();

        }

        // Begin of Adding CORS support to the API

        app.UseCors("CorsPolicy");

        // End of Adding CORS support to the API

        app.UseRouting();

        app.UseEndpoints(*endpoints* =>

        {

            endpoints.MapControllers();

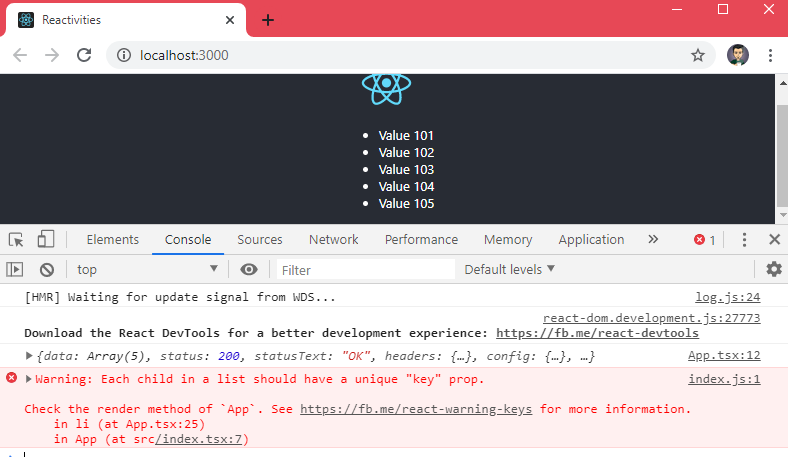
        });

    }

}

**Step 2:** Khi ta thực hiện re-build và re-run projects ta sẽ thấy ta nhận được results mong muốn nhưng vẫn còn một lỗi.

Lỗi này phát sinh vì thành phần thẻ li của ta khai báo trong component không có thuộc tính key, nên react nó không happy với việc này.



**Step 3:** Fix lỗi li không có key. Ta thêm đoạn code cho thẻ li như bên dưới, và khi ta thực hiện run lại sẽ không còn nhận lỗi nào nữa.

class App extends *Component* {

  state = {

    values:[]

  }

  componentDidMount(){

    axios.get('http://localhost:5000/api/values')

    .then((*response*)=>{

*console*.log(response);

*this*.setState({

        values: response.data

      })

    })

  }

  render(){

    return (

      <div *className*="App">

        <header *className*="App-header">

          <img *src*={logo} *className*="App-logo" *alt*="logo" />

          <ul>

            {*this*.state.values.map((*value*:*any*)=>(

              <li *key*={value.id}>{value.name}</li>

            ))}

          </ul>

        </header>

      </div>

    );

  }

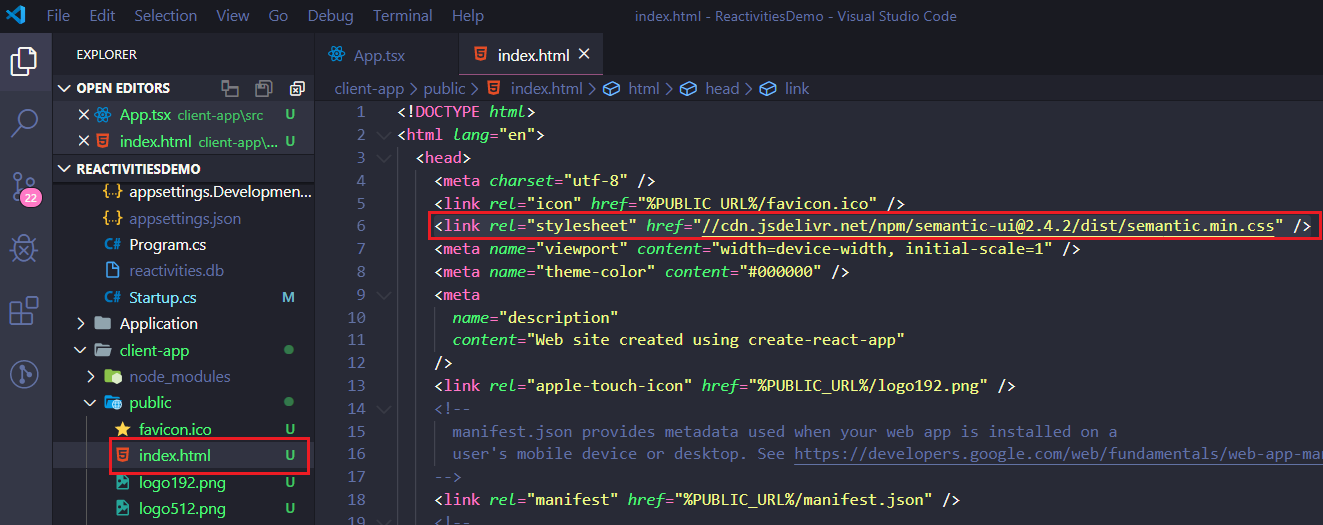
}

**Adding Semantic UI to our app**

**Step 1:** install semantic UI

PS C:\dev\ReactivitiesDemo\client-app> npm install semantic-ui-react

**Step 2:** Add CSS reference cho index.html.



**Step 3** : Refactor trang App.tsx như sau:

import React,{Component} from 'react';

import { Header, List } from 'semantic-ui-react'

import './App.css';

import axios from 'axios';

class App extends *Component* {

  state = {

    values:[]

  }

  componentDidMount(){

    axios.get('http://localhost:5000/api/values')

    .then((*response*)=>{

*console*.log(response);

*this*.setState({

        values: response.data

      })

    })

  }

  render(){

    return (

      <div>

        <*Header* *as*='h2' *icon*='users' *content*='Reactivities ' />

        <*List*>

          {*this*.state.values.map((*value*:*any*)=>(

              <*List.Item* *key*={value.id}>{value.name}</*List.Item*>

            ))}

        </*List*>

      </div>

    );

  }

}

export default App;

**Section 3:**

**Building a CRUD application in .net core using CQRS + Mediator pattern**

**Adding the Activity entity**

**Step 1:** Thêm mới một model gọi là Activity

using *System*;

namespace *Domain*

{

    public class Activity

    {

        public *Guid* Id { get; set; }

        public *string* Title { get; set; }

        public *string* Description { get; set; }

        public *string* Category { get; set; }

        public *DateTime* Date { get; set; }

        public *string* City { get; set; }

        public *string* Venue { get; set; }

    }

}

**Step 2:** Trong DataContext ta thêm Activity vào DbSet như sau:

using *System*;

using *Domain*;

using *Microsoft*.*EntityFrameworkCore*;

namespace *Persistence*

{

    public class DataContext : *DbContext*

    {

        public DataContext(*DbContextOptions* *options*) : base(options)

        {}

        public *DbSet*<*Value*> Values { get; set; }

        public *DbSet*<*Activity*> Activities {get; set;}

        protected override *void* OnModelCreating(*ModelBuilder* *builder*)

        {

            builder.Entity<*Value*>().HasData(

                new *Value* { Id = 6, Name = "Value 101" },

                new *Value* { Id = 7, Name = "Value 102" },

                new *Value* { Id = 8, Name = "Value 103" },

                new *Value* { Id = 9, Name = "Value 104" },

                new *Value* { Id = 10, Name = "Value 105" }

            );

        }

    }

}

**Step 3:** Thực hiện lệnh migration để thêm mới entity Activity như sau:

**PS C:\dev\ReactivitiesDemo> dotnet ef migrations add "ActivityEntityAdded" -p .\Persistence\ -s .\API\**

**Step 4:** Vì ta đã implement việc update database sau khi migrations trong file Startup.cs của API do vậy ta chỉ việc restart lại application.

**PS C:\dev\ReactivitiesDemo\API> dotnet watch run**

**Seeding Activity data**

**Step 1:** Ta thêm một class tên là Seed trong Persistence như sau:

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *Domain*;

namespace *Persistence*

{

    public class Seed

    {

        public static *void* SeedData(*DataContext* *context*){

            if(!context.Activities.Any()){

                var activities = new *List*<*Activity*>{

                    new *Activity*

                    {

                        Title = "Past Activity 1",

                        Date = DateTime.Now.AddMonths(-2),

                        Description = "Activity 2 months ago",

                        Category = "drinks",

                        City = "London",

                        Venue = "Pub",

                    },

                    new *Activity*

                    {

                        Title = "Past Activity 2",

                        Date = DateTime.Now.AddMonths(-1),

                        Description = "Activity 1 month ago",

                        Category = "culture",

                        City = "Paris",

                        Venue = "Louvre",

                    },

                    new *Activity*

                    {

                        Title = "Future Activity 1",

                        Date = DateTime.Now.AddMonths(1),

                        Description = "Activity 1 month in future",

                        Category = "culture",

                        City = "London",

                        Venue = "Natural History Museum",

                    },

                    new *Activity*

                    {

                        Title = "Future Activity 2",

                        Date = DateTime.Now.AddMonths(2),

                        Description = "Activity 2 months in future",

                        Category = "music",

                        City = "London",

                        Venue = "O2 Arena",

                    },

                    new *Activity*

                    {

                        Title = "Future Activity 3",

                        Date = DateTime.Now.AddMonths(3),

                        Description = "Activity 3 months in future",

                        Category = "drinks",

                        City = "London",

                        Venue = "Another pub",

                    },

                    new *Activity*

                    {

                        Title = "Future Activity 4",

                        Date = DateTime.Now.AddMonths(4),

                        Description = "Activity 4 months in future",

                        Category = "drinks",

                        City = "London",

                        Venue = "Yet another pub",

                    },

                    new *Activity*

                    {

                        Title = "Future Activity 5",

                        Date = DateTime.Now.AddMonths(5),

                        Description = "Activity 5 months in future",

                        Category = "drinks",

                        City = "London",

                        Venue = "Just another pub",

                    },

                    new *Activity*

                    {

                        Title = "Future Activity 6",

                        Date = DateTime.Now.AddMonths(6),

                        Description = "Activity 6 months in future",

                        Category = "music",

                        City = "London",

                        Venue = "Roundhouse Camden",

                    },

                    new *Activity*

                    {

                        Title = "Future Activity 7",

                        Date = DateTime.Now.AddMonths(7),

                        Description = "Activity 2 months ago",

                        Category = "travel",

                        City = "London",

                        Venue = "Somewhere on the Thames",

                    },

                    new *Activity*

                    {

                        Title = "Future Activity 8",

                        Date = DateTime.Now.AddMonths(8),

                        Description = "Activity 8 months in future",

                        Category = "film",

                        City = "London",

                        Venue = "Cinema",

                    }

                };

            context.Activities.AddRange(activities);

            context.SaveChanges();

            }

        }

    }

}

**Step 2:** Trong file Program.cs của API, method Main ta thêm vào dòng code sau để thực hiện việc seeding data cho table Activities.

public static *void* Main(*string*[] *args*)

{

    var host = CreateHostBuilder(args).Build();

    using(var scope = host.Services.CreateScope())

    {

        var services = scope.ServiceProvider;

        try

        {

            var context = services.GetRequiredService<*DataContext*>();

            context.Database.Migrate();

            Seed.SeedData(context);

        }

        catch(*Exception* ex)

        {

            var logger = services.GetRequiredService<*ILogger*<*Program*>>();

            logger.LogError(ex,"An error occured during migration");

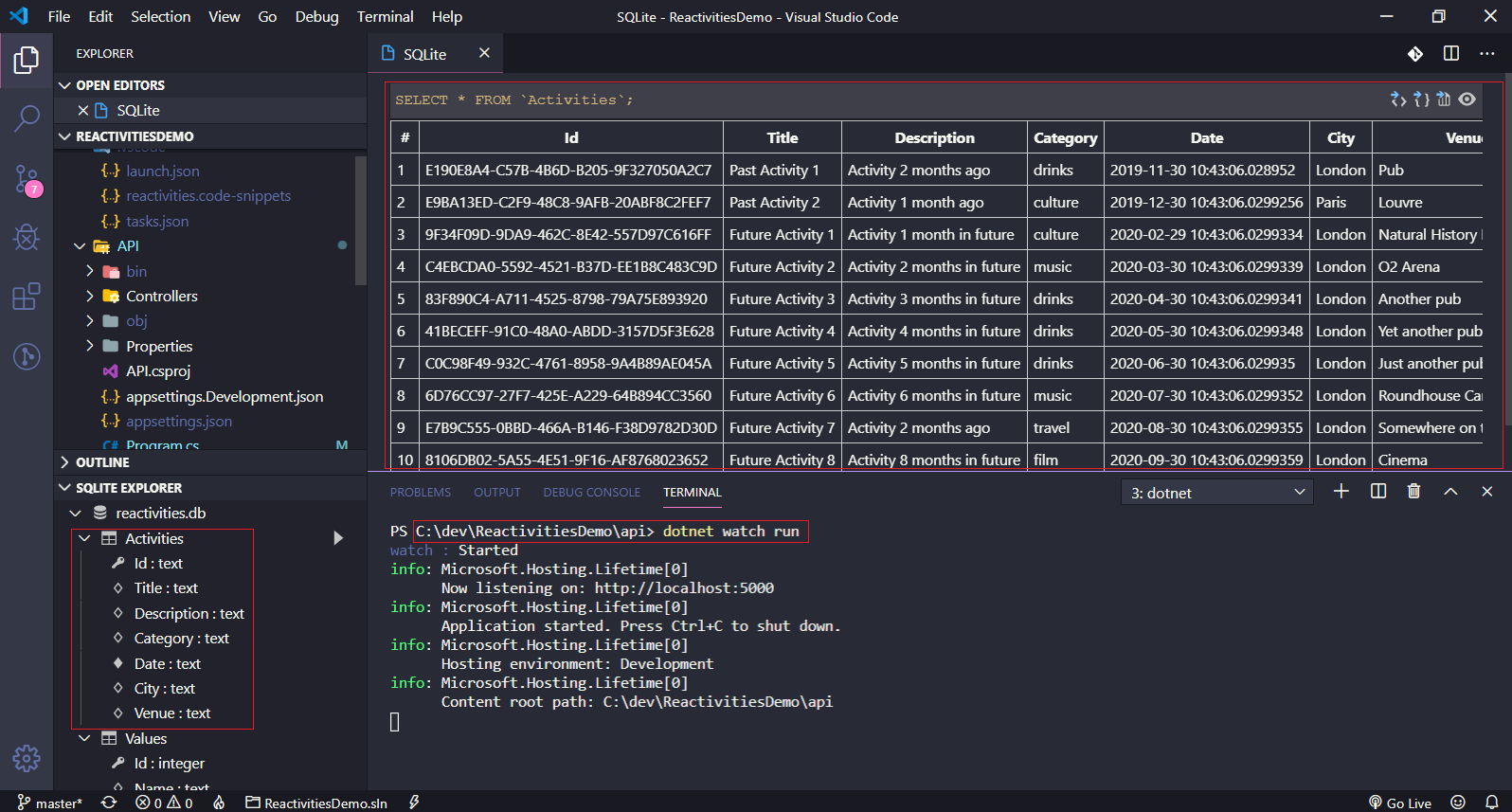
        }

    }

    host.Run();

}

**Step 3**: Khi ta thực hiện restart lại ứng dụng ta sẽ thấy table Activities sẽ được thêm các data như sau:



**Command and Queries -CQRS.**

Command

- Does something

- Modifies state

- Should not return a value

Query

- Answers a question

- Does not modify state

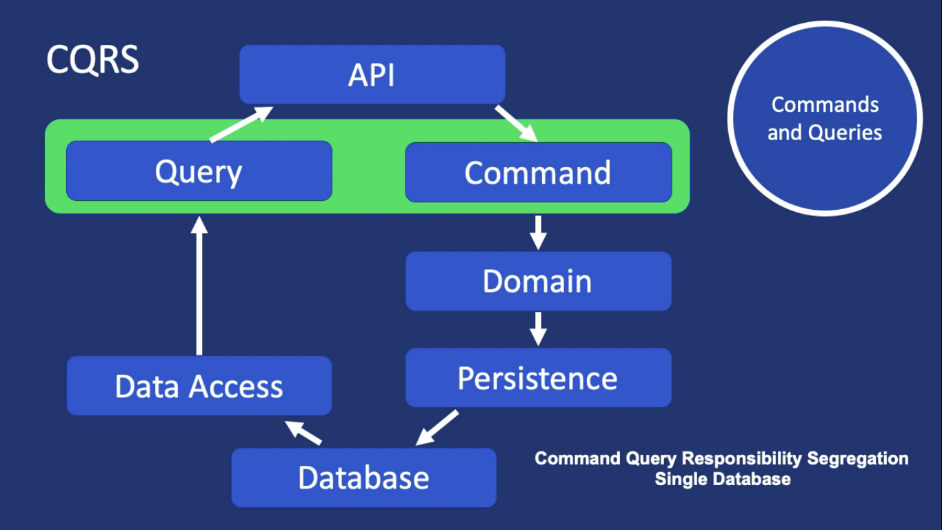
- Should return a value

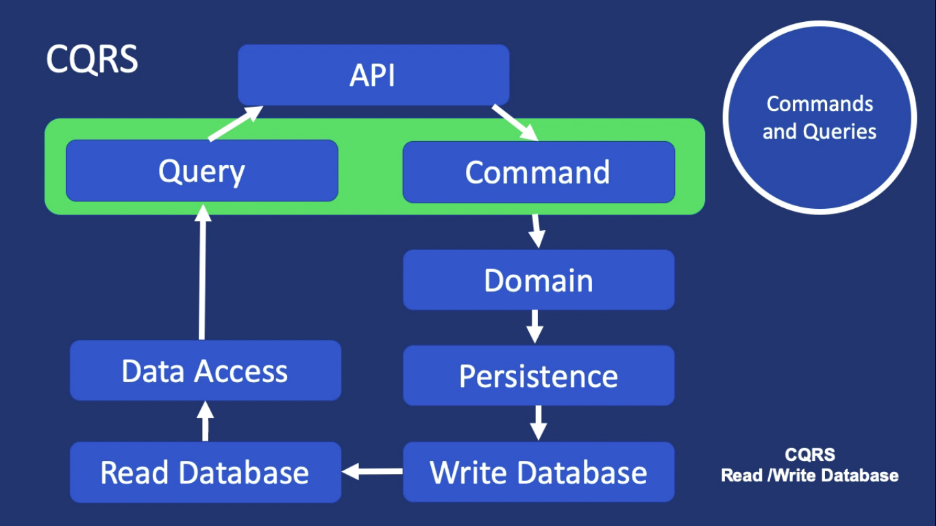
Commands use write DB

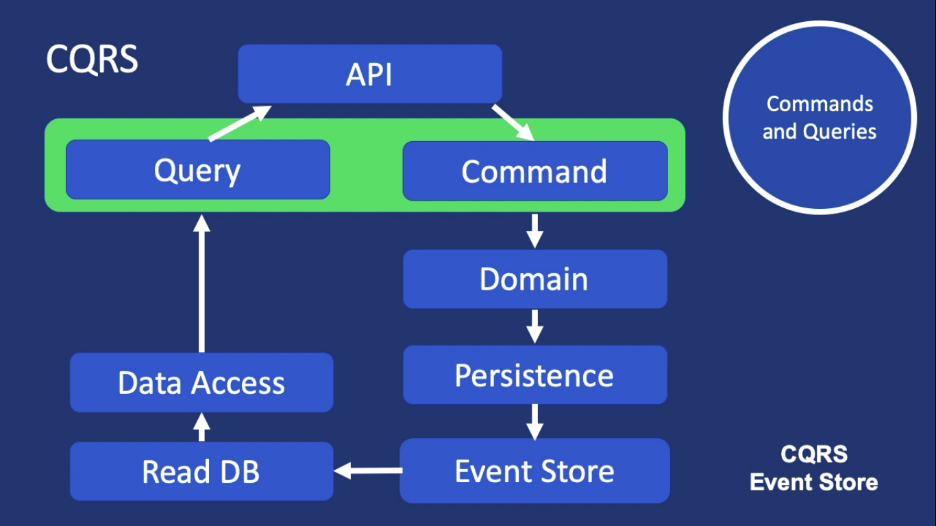
Queries use read DB

Eventual consistence

Can be faster.

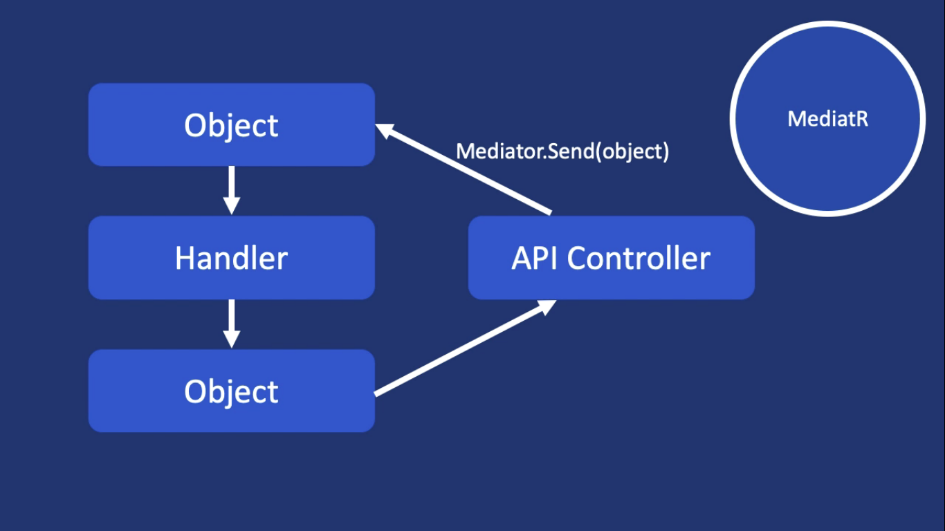


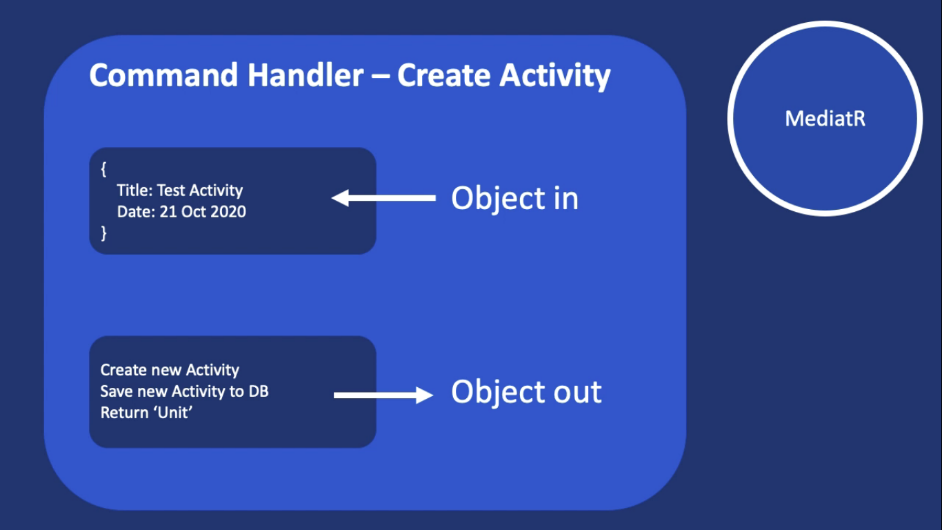


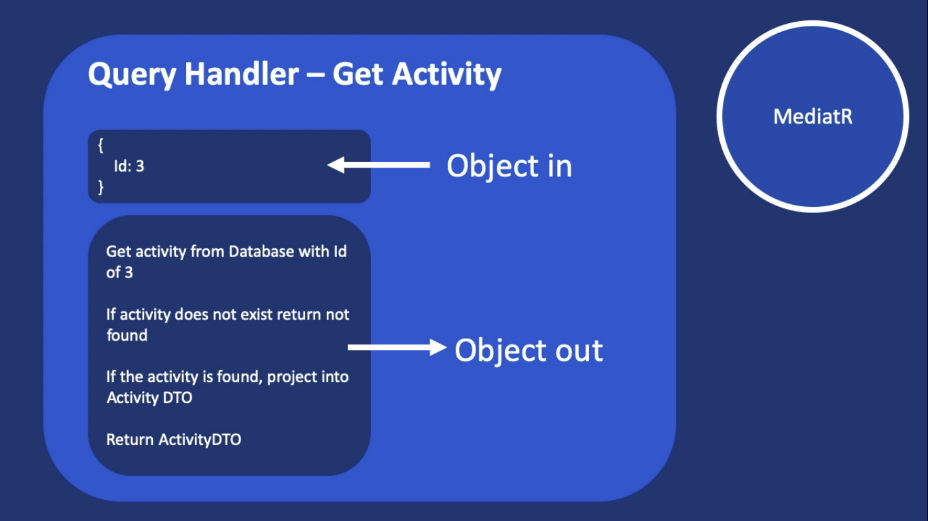




Introduction to MediatR







**Creating our first query handler**

**Step 1:** Cài đặt extension tên là MediatR.Extensions.Microsoft.DependencyInjection vào solution Application

**Step 2:** Tạo một thư mục Activities trong solution Application và thêm vào một class là List.cs như sau:

using *System*.*Collections*.*Generic*;

using *System*.*Threading*;

using *System*.*Threading*.*Tasks*;

using *Domain*;

using *MediatR*;

using *Microsoft*.*EntityFrameworkCore*;

using *Persistence*;

namespace *Application*.*Activities*

{

    public class List

    {

        public class Query : *IRequest*<*List*<*Activity*>> { }

        public class Handler : *IRequestHandler*<*Query*, *List*<*Activity*>>

        {

            private readonly *DataContext* \_context;

            public Handler(*DataContext* *context*)

            {

                \_context = context;

            }

            public async *Task*<*List*<*Activity*>> Handle(*Query* *request*, *CancellationToken* *cancellationToken*)

            {

                var activities = await \_context.Activities.ToListAsync();

                return activities;

            }

        }

    }

}

**Creating the activities API Controller.**

**Step 1:** Trong project API ta thêm mới một controller tên là ActivitiesController.cs như sau:

using *System*.*Collections*.*Generic*;

using *System*.*Threading*.*Tasks*;

using *Application*.*Activities*;

using *Domain*;

using *MediatR*;

using *Microsoft*.*AspNetCore*.*Mvc*;

namespace *API*.*Controllers*

{

    [*Route*("api/[controller]")]

    [*ApiController*]

    public class ActivitiesController : *ControllerBase*

    {

        private readonly *IMediator* \_mediator;

        public ActivitiesController(*IMediator* *mediator*)

        {

            \_mediator = mediator;

        }

        [*HttpGet*]

        public async *Task*<*ActionResult*<*List*<*Activity*>>> List()

        {

            return await \_mediator.Send(new *List*.*Query*());

        }

    }

}

**Step 2:** Ta thực hiện việc đăng ký trong Startup.cs của project API như sau:

public *void* ConfigureServices(*IServiceCollection* *services*)

{

    services.AddDbContext<*DataContext*>(*opt*=>{

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

    });

    // Begin of Adding CORS Support to the API

    services.AddCors(*opt*=>{

        opt.AddPolicy("CorsPolicy",*policy*=>{

            policy.AllowAnyHeader().AllowAnyMethod().WithOrigins("http://localhost:3000");

        });

    });

    // End of Adding CORS Support to the API

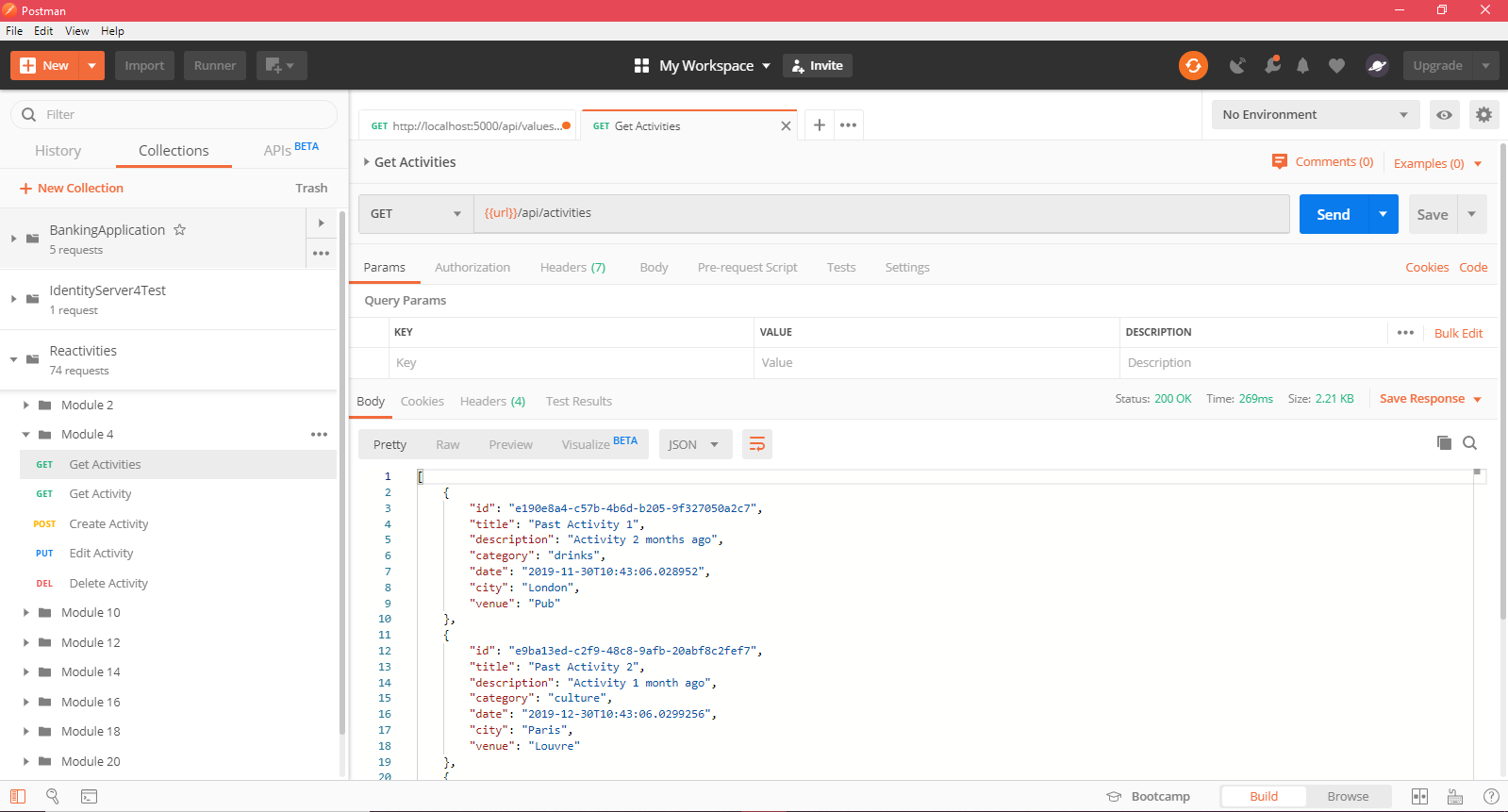
    // Adding mediatR

    services.AddMediatR(typeof(*List*.*Handler*).Assembly);

    services.AddControllers();

}

**Step 3:** Ta thực hiện build và run ứng dụng và test **api/activities** sẽ được kết quả như sau:



**Adding the details hander**

**Step 1**: Trong thư mục Activities ta thêm vào một class tên là Detail.cs như sau:

using *System*;

using *System*.*Threading*;

using *System*.*Threading*.*Tasks*;

using *Domain*;

using *MediatR*;

using *Persistence*;

namespace *Application*.*Activities*

{

    public class Details

    {

        public class Query:*IRequest*<*Activity*>

        {

            public *Guid* Id {get; set;}

        }

        public class Handler : *IRequestHandler*<*Query*, *Activity*>

        {

            private readonly *DataContext* \_context;

            public Handler(*DataContext*  *context*)

            {

                \_context = context;

            }

            public async *Task*<*Activity*> Handle(*Query* *request*, *CancellationToken* *cancellationToken*)

            {

                var activity = await \_context.Activities.FindAsync(request.Id);

                return activity;

            }

        }

    }

}

**Step 2:** Trong ActivitiesController ta thêm vào một phương thức Details như sau:

[*HttpGet*("{id}")]

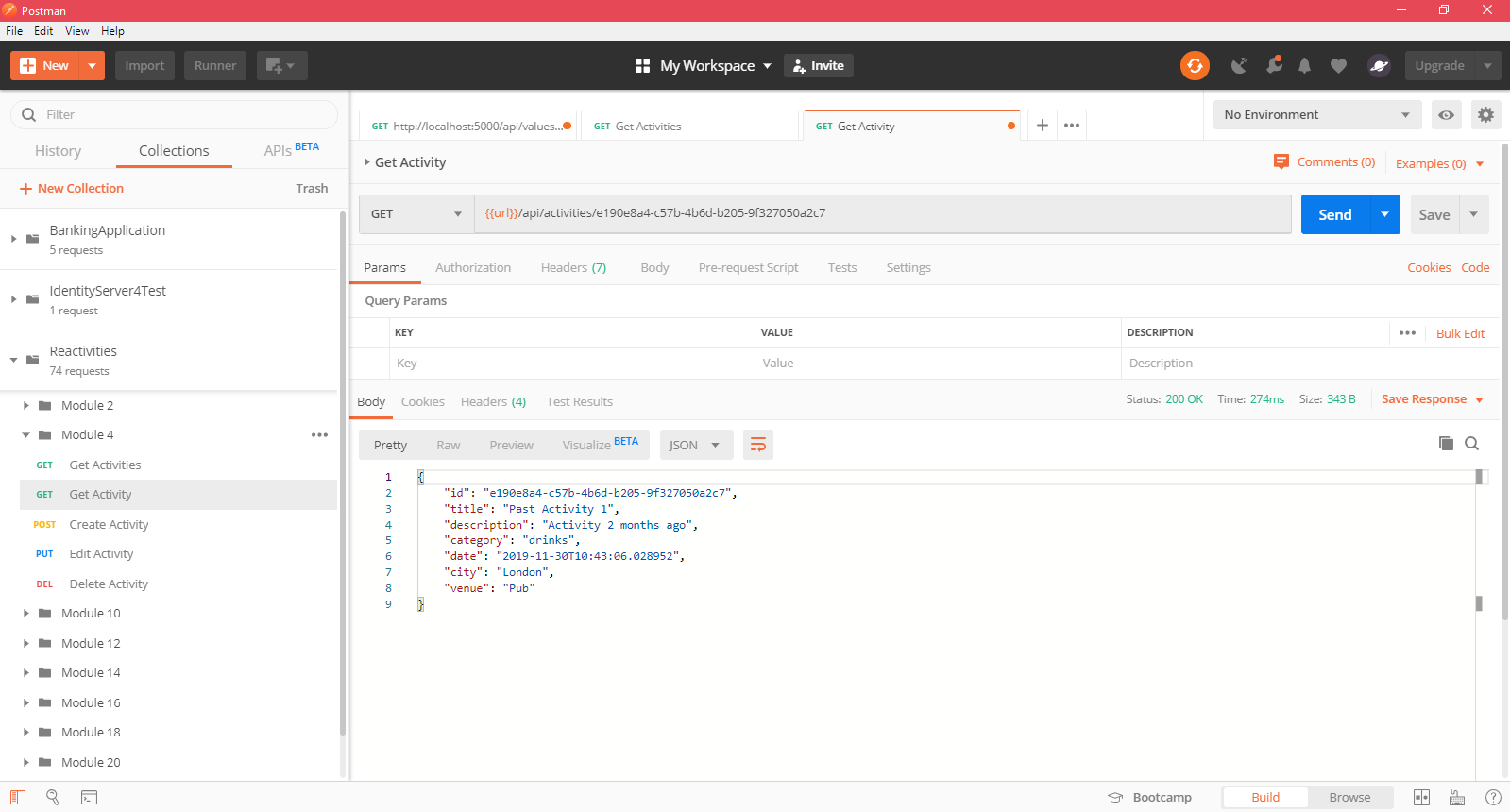
public async *Task*<*ActionResult*<*Activity*>> Details(*Guid* *id*)

{

    return await \_mediator.Send(new *Details*.*Query*{Id = id});

}

**Step 3:** Khi ta test get deail một activity bằng Postman ta sẽ được kết quả như sau:



**Cancellation tokens**

Trong phần này ta sẽ làm một demo về việc sử dụng cancellation token. Ngữ cảnh là khi ta lấy tất cả các item của đối tượng Activity, trong lúc đang lấy data, ta thực hiện cancel request đó.

**Step 1:** Trong file List.cs ta implement như sau;

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Threading*;

using *System*.*Threading*.*Tasks*;

using *Domain*;

using *MediatR*;

using *Microsoft*.*EntityFrameworkCore*;

using *Microsoft*.*Extensions*.*Logging*;

using *Persistence*;

namespace *Application*.*Activities*

{

    public class List

    {

        public class Query : *IRequest*<*List*<*Activity*>> { }

        public class Handler : *IRequestHandler*<*Query*, *List*<*Activity*>>

        {

            private readonly *DataContext* \_context;

            private readonly *ILogger*<*List*> \_logger;

            public Handler(*DataContext* *context*, *ILogger*<*List*> *logger*)

            {

                \_logger = logger;

                \_context = context;

            }

            public async *Task*<*List*<*Activity*>> Handle(*Query* *request*, *CancellationToken* *cancellationToken*)

            {

                //Demo using CancellationToken

                try

                {

                    for (*int* i = 0; i < 10; i++)

                    {

                        cancellationToken.ThrowIfCancellationRequested();

                        await Task.Delay(1000, cancellationToken);

                        \_logger.LogInformation($"Task {i} has completed");

                    }

                }

                catch (*Exception* ex) when (ex is *TaskCanceledException*)

                {

                    \_logger.LogInformation("Task was cancelled");

                }

                var activities = await \_context.Activities.ToListAsync(cancellationToken);

                return activities;

            }

        }

    }

}

**Step 2:** trong file ActivitiesController.cs ta implement thêm các đoạn code sau:

[*HttpGet*]

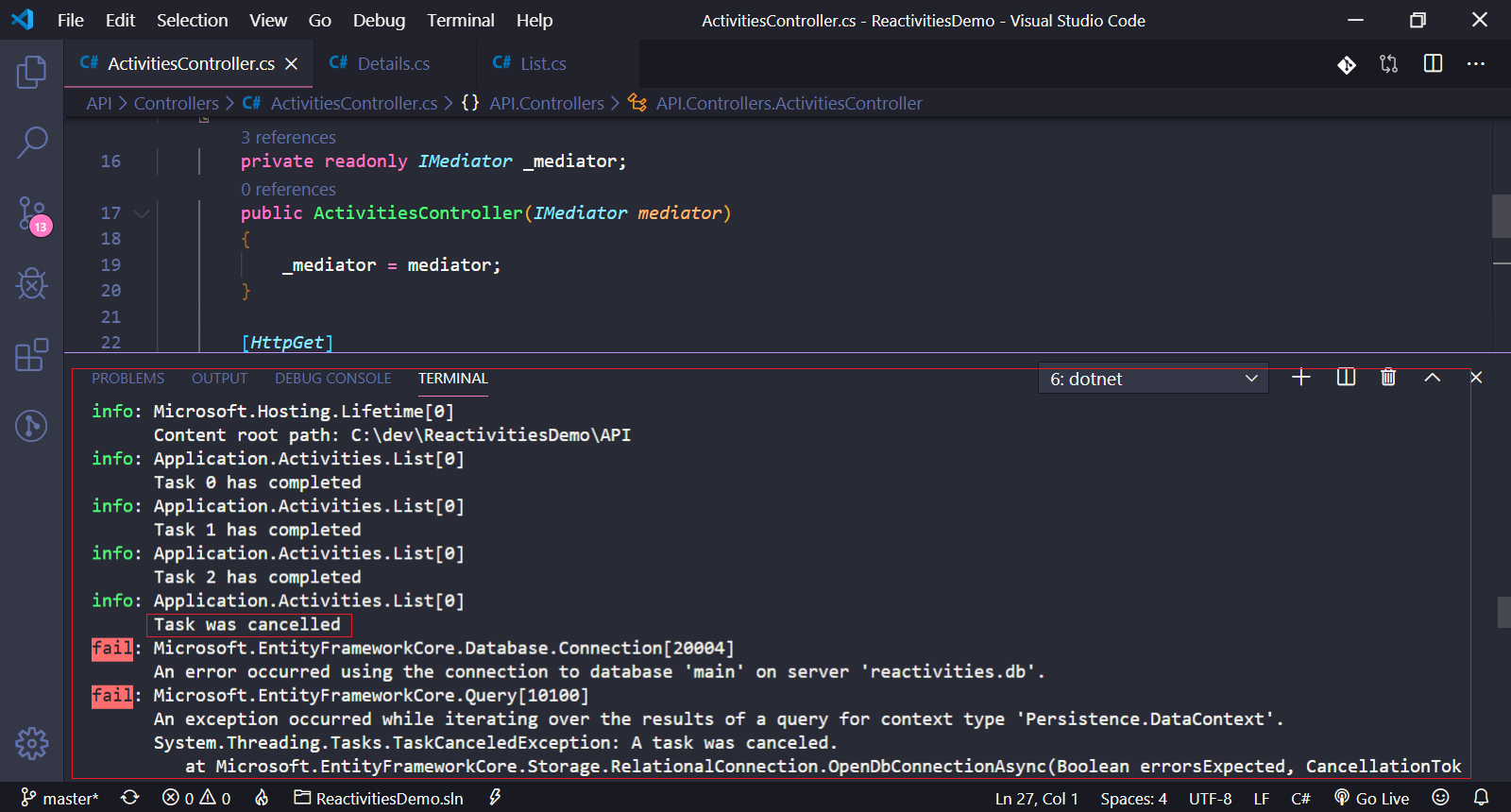
public async *Task*<*ActionResult*<*List*<*Activity*>>> List(*CancellationToken* *ct*)

{

    return await \_mediator.Send(new *List*.*Query*(),ct);

}

**Step 3:** Khi ta gởi một request để lấy hết các activities, liền sau đó ta cancel request đó thì ta sẽ thấy restult như sau:



**Adding the create handler**

**Step 1:** Trong folder Activities của Application ta thêm mới một class tên là Create như sau

using *System*;

using *System*.*Threading*;

using *System*.*Threading*.*Tasks*;

using *Domain*;

using *MediatR*;

using *Persistence*;

namespace *Application*.*Activities*

{

    public class Create

    {

        public class Command : *IRequest*

        {

            public *Guid* Id { get; set; }

            public *string* Title { get; set; }

            public *string* Description { get; set; }

            public *string* Category { get; set; }

            public *DateTime* Date { get; set; }

            public *string* City { get; set; }

            public *string* Venue { get; set; }

        }

        public class Handler : *IRequestHandler*<*Command*>

        {

            private readonly *DataContext* \_context;

            public Handler(*DataContext* *context*)

            {

                \_context = context;

            }

            public async *Task*<*Unit*> Handle(*Command* *request*, *CancellationToken* *cancellationToken*)

            {

                var activity = new *Activity*

                {

                    Id = request.Id,

                    Title = request.Title,

                    Description = request.Description,

                    Category = request.Category,

                    Date = request.Date,

                    City = request.City,

                    Venue = request.Venue

                };

                \_context.Activities.Add(activity);

                var success = await \_context.SaveChangesAsync()>0;

                if(success) return Unit.Value;

                throw new *Exception*("Problem saving changges");

            }

        }

    }

}

**Step 2:** Trong ActivitiesController ta thêm vào một method để handle request thêm mới từ client.

[*HttpPost*]

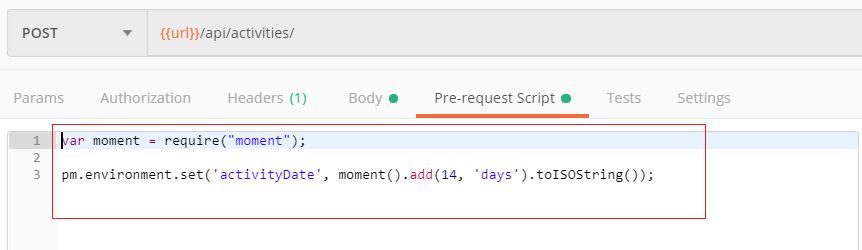
public async *Task*<*ActionResult*<*Unit*>> Create (*Create*.*Command* *command*)

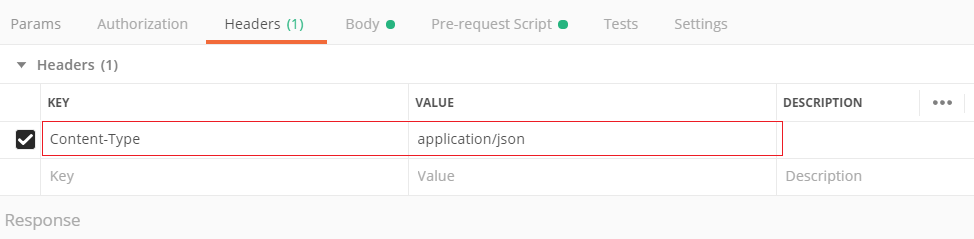
{

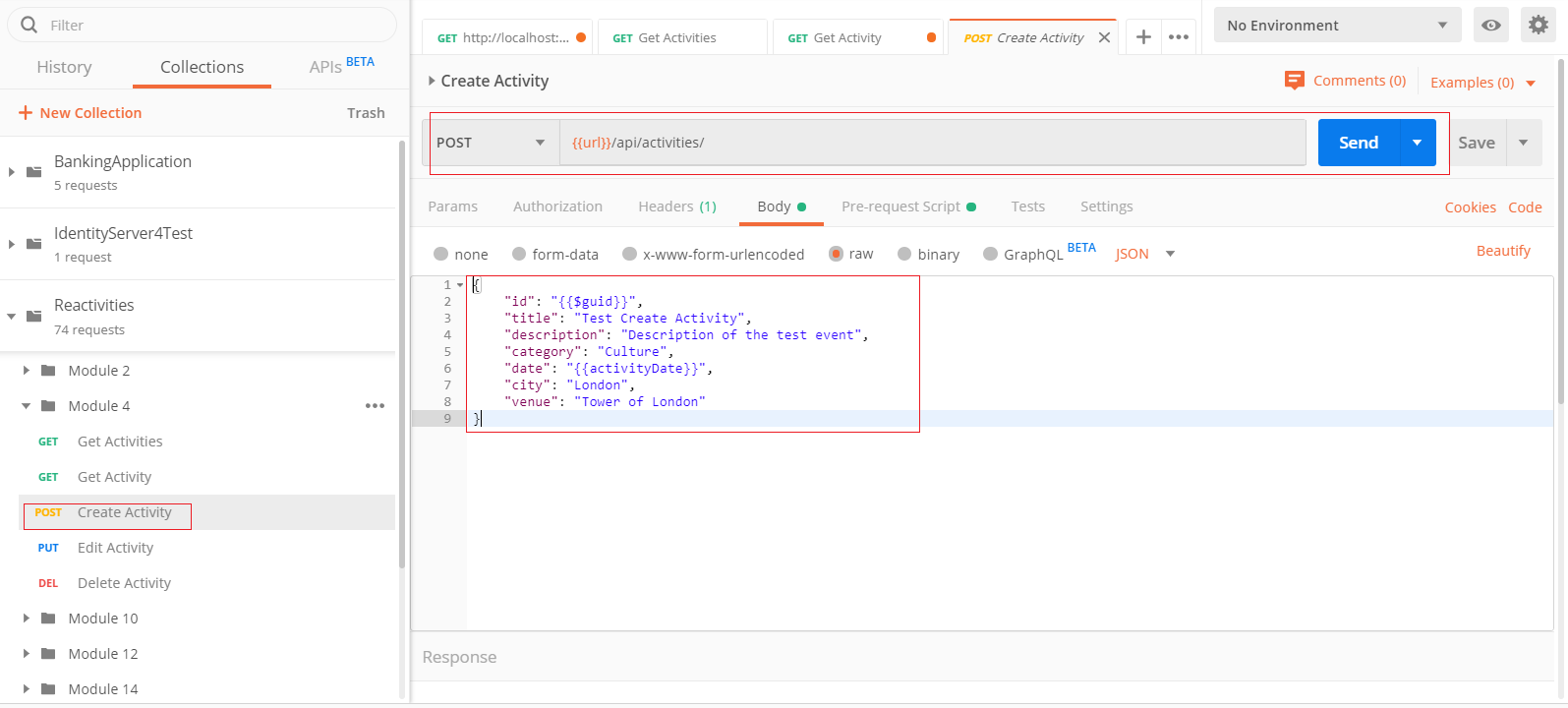
    return await \_mediator.Send(command);

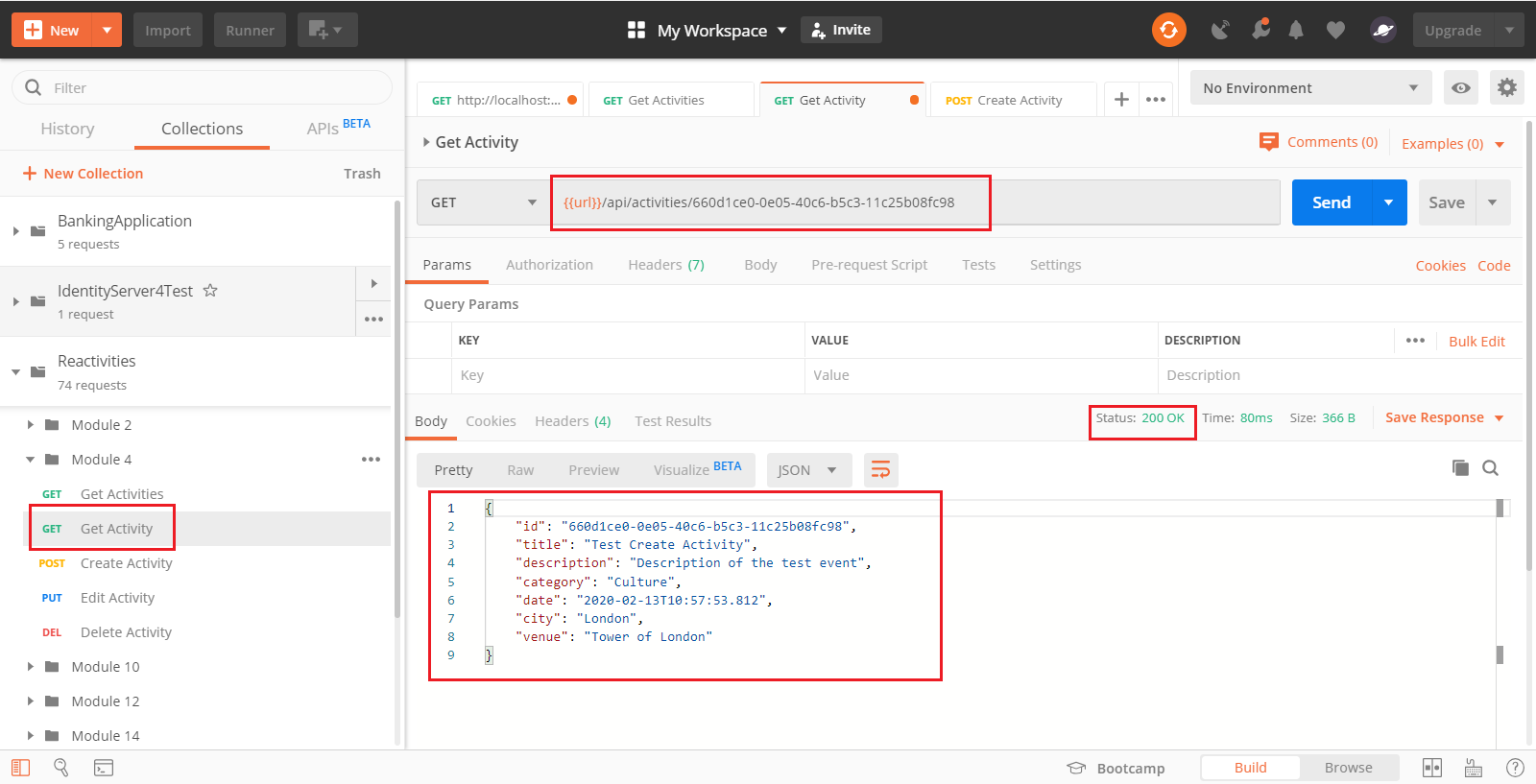
}

**Step 3:** Thực hiện việc testing ứng dụng.





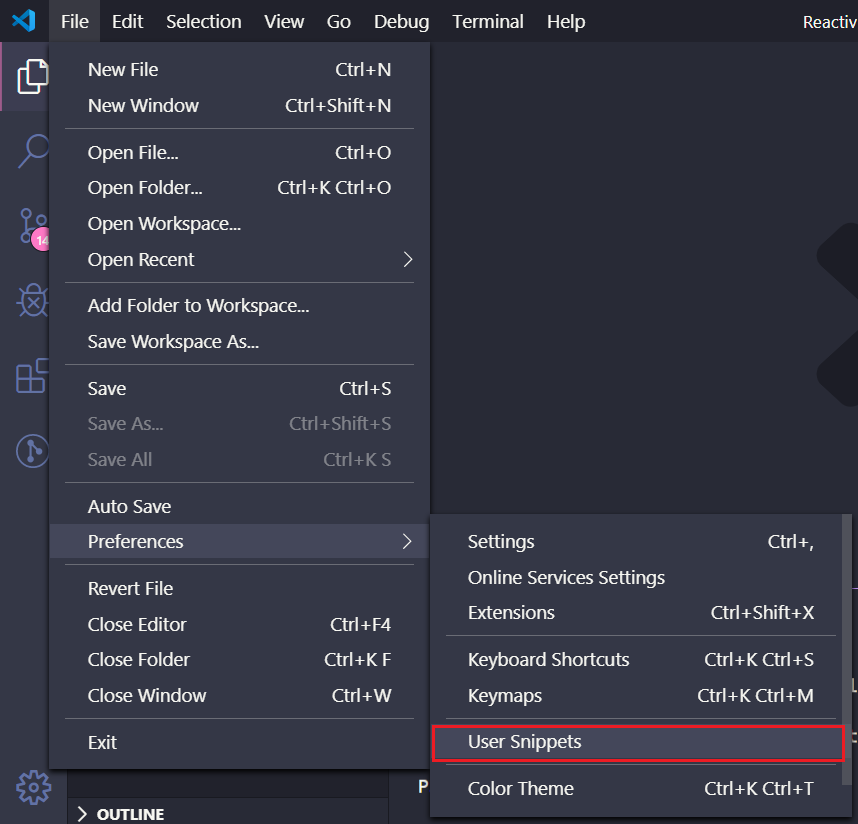




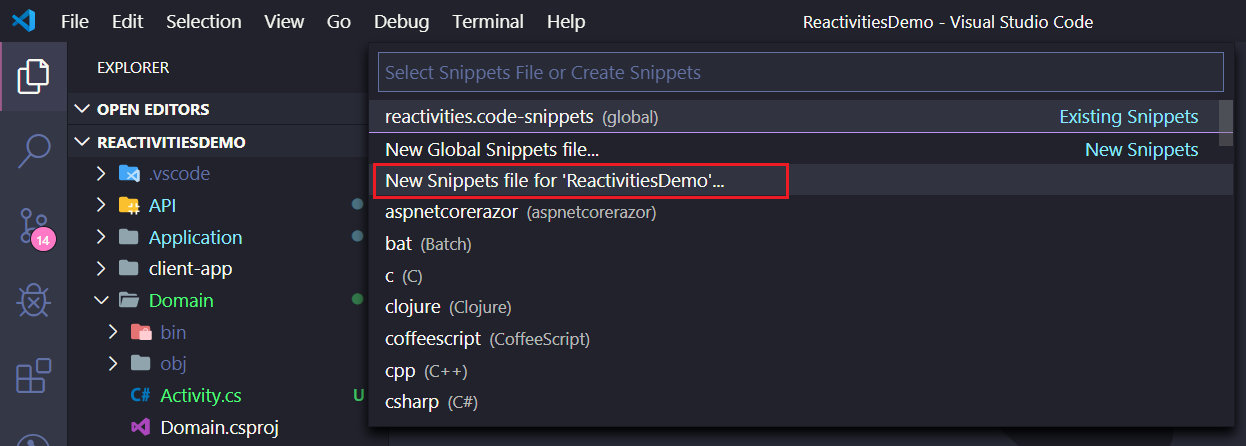
**Dealing with boilerplate code in our handlers**

Trong phần này ta sẽ biết cách tạo một code snippet trong VS Code, điều này sẽ giúp ta đỡ phải gõ lại nhiều lần những đoạn code giống nhau.

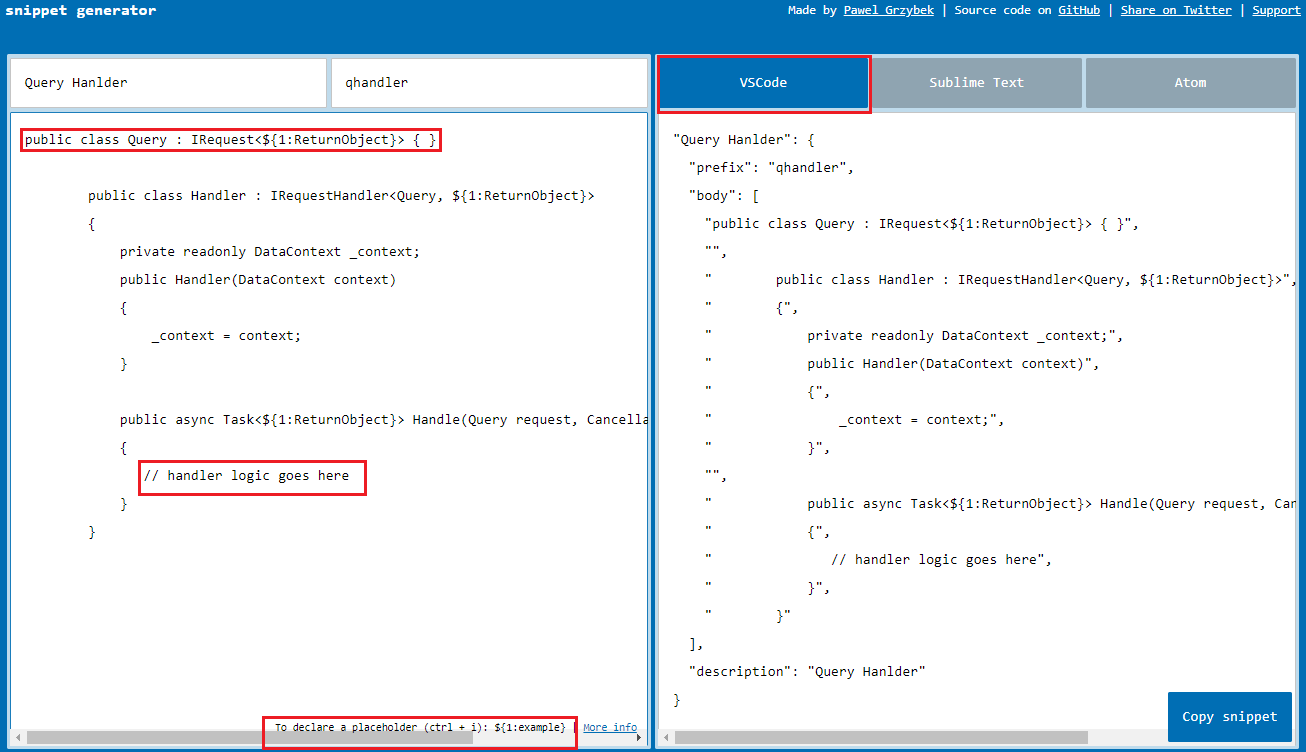
**Step 1:**

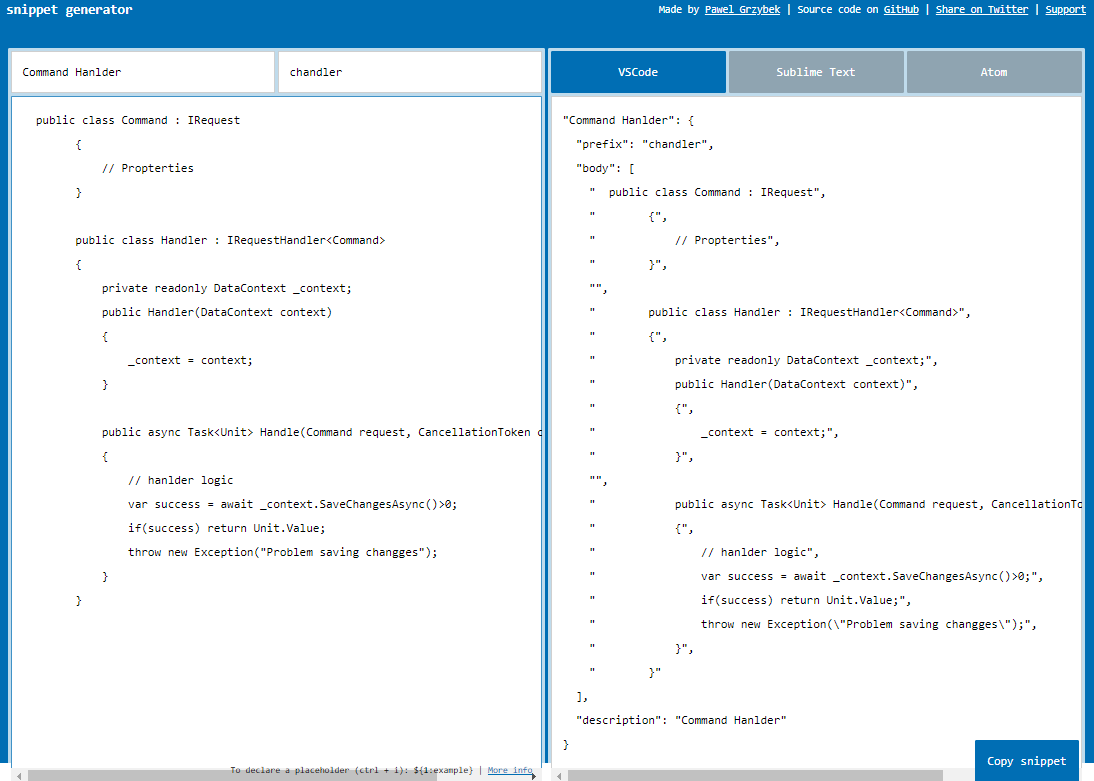


**Step 2:**

****

**Step 3**: Visit<https://snippet-generator.app/>

****



Ta được kết quả như sau:

{

    "Query Hanlder": {

        "prefix": "qhandler",

        "body": [

          "public class Query : IRequest<${1:ReturnObject}> { }",

          "",

          "        public class Handler : IRequestHandler<Query, ${1:ReturnObject}>",

          "        {",

          "            private readonly DataContext \_context;",

          "            public Handler(DataContext context)",

          "            {",

          "                \_context = context;",

          "            }",

          "",

          "            public async Task<${1:ReturnObject}> Handle(Query request, CancellationToken cancellationToken)",

          "            {",

          "               // handler logic goes here",

          "            }",

          "        }"

        ],

        "description": "Query Hanlder"

      },

      "Command Hanlder": {

        "prefix": "chandler",

        "body": [

          "  public class Command : IRequest",

          "        {",

          "            // Propterties",

          "        }",

          "",

          "        public class Handler : IRequestHandler<Command>",

          "        {",

          "            private readonly DataContext \_context;",

          "            public Handler(DataContext context)",

          "            {",

          "                \_context = context;",

          "            }",

          "",

          "            public async Task<Unit> Handle(Command request, CancellationToken cancellationToken)",

          "            {",

          "                // hanlder logic",

          "                var success = await \_context.SaveChangesAsync()>0;",

          "                if(success) return Unit.Value;",

          "                throw new Exception(\"Problem saving changges\");",

          "            }",

          "        }"

        ],

        "description": "Command Hanlder"

      }

}

**Adding an Edit handler**

**Step 1:** Trong folder Activities của solution Application ta thêm một class tên là Edit như sau:

using *System*;

using *System*.*Threading*;

using *System*.*Threading*.*Tasks*;

using *MediatR*;

using *Persistence*;

namespace *Application*.*Activities*

{

    public class Edit

    {

        public class Command : *IRequest*

        {

            public *Guid* Id { get; set; }

            public *string* Title { get; set; }

            public *string* Description { get; set; }

            public *string* Category { get; set; }

            public *DateTime*? Date { get; set; }

            public *string* City { get; set; }

            public *string* Venue { get; set; }

        }

        public class Handler : *IRequestHandler*<*Command*>

        {

            private readonly *DataContext* \_context;

            public Handler(*DataContext* *context*)

            {

                \_context = context;

            }

            public async *Task*<*Unit*> Handle(*Command* *request*, *CancellationToken* *cancellationToken*)

            {

                var activity = await \_context.Activities.FindAsync(request.Id);

                if(activity ==null)

                {

                        throw new *Exception*("Could not find activity");

                }

                activity.Title = request.Title??activity.Title;

                activity.Description = request.Description??activity.Description;

                activity.Category = request.Category??activity.Category;

                activity.Date = request.Date??activity.Date;

                activity.City = request.City??activity.City;

                activity.Venue = request.Venue??activity.Venue;

                var success = await \_context.SaveChangesAsync() > 0;

                if (success) return Unit.Value;

                throw new *Exception*("Problem saving changes");

            }

        }

    }

}

**Step 2:** Trong ActivitiesController ta thêm một method như sau:

[*HttpPut*("{id}")]

public async *Task*<*ActionResult*<*Unit*>> Edit(*Guid* *id*, *Edit*.*Command* *command*)

{

    command.Id = id;

    return await \_mediator.Send(command);

}

**Addin a Delete handler**

**Step 1**: Trong thư mục Activities của solution Application ta thêm mới một class Delete như sau:

using *System*;

using *System*.*Threading*;

using *System*.*Threading*.*Tasks*;

using *MediatR*;

using *Persistence*;

namespace *Application*.*Activities*

{

    public class Delete

    {

        public class Command : *IRequest*

        {

            public *Guid* Id { get; set; }

        }

        public class Handler : *IRequestHandler*<*Command*>

        {

            private readonly *DataContext* \_context;

            public Handler(*DataContext* *context*)

            {

                \_context = context;

            }

            public async *Task*<*Unit*> Handle(*Command* *request*, *CancellationToken* *cancellationToken*)

            {

                var activity = await \_context.Activities.FindAsync(request.Id);

                if(activity == null)

                    throw new *Exception*("Could not find activity");

                \_context.Remove(activity);

                var success = await \_context.SaveChangesAsync() > 0;

                if (success) return Unit.Value;

                throw new *Exception*("Problem saving changges");

            }

        }

    }

}

**Step 2**: Trong ActivitiesController ta thêm mới một method như sau:

[*HttpDelete*("{id}")]

public async *Task*<*ActionResult*<*Unit*>> Delete(*Guid* *id*)

{

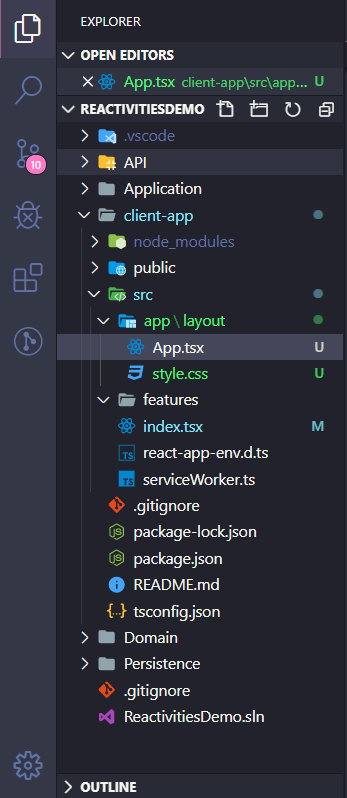
    return await \_mediator.Send(new *Delete*.*Command*{Id = id});

}

**Section 4:**

**Building a CRUD application in React**

**Folder stucture in React**



**Getting a list of activities from the API**

**Step 1:** trong App.tsx ta edit lại như sau

***Lưu ý rằng đối tượng activity ở đây ta đang định kiểu là any, do vậy khi tra truy cập các thuộc tính mà thằng Activity không có thì nó vẫn happy không báo lỗi, chỉ là nó không hiển gì ra mà thôi***

import React,{Component} from 'react';

import { Header, List } from 'semantic-ui-react'

import axios from 'axios';

class App extends *Component* {

  state = {

    activities:[]

  }

  componentDidMount(){

    axios.get('http://localhost:5000/api/activities')

    .then((*response*)=>{

*this*.setState({

        activities: response.data

      })

    })

  }

  render(){

    return (

      <div>

        <*Header* *as*='h2' *icon*='users' *content*='Reactivities ' />

        <*List*>

          {*this*.state.activities.map((*activity*:*any*)=>(

              <*List.Item* *key*={activity.id}>{activity.title}</*List.Item*>

            ))}

        </*List*>

      </div>

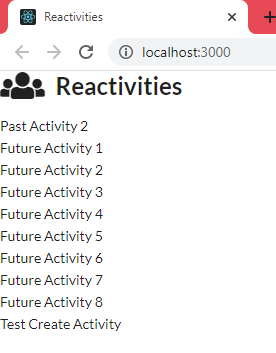
    );

  }

}

export default App;

**Step 2**: Ta thực hiện run lại ứng dụng sẽ được kết quả sau:



**Adding an Activity interface in Typescript**

**Step 1**: Ta thêm một interface tên là activity như sau



**Step 2:** trong file App.tsx ta edit lại như sau

Tới lúc này activity của ta là một strongly type không còn là kiểu any nữa, và nó sẽ giúp ta dễ dàng phát hiện các silly mistake.

import React,{Component} from 'react';

import { Header, List } from 'semantic-ui-react'

import axios from 'axios';

import { IActivity } from '../models/activity';

interface *IState*{

  activities: *IActivity*[];

}

class App extends *Component*<{},*IState*> {

  // best practice to set it readonly

  readonly state: *IState* = {

    activities:[]

  }

  componentDidMount(){

    // axious will return a array of activities

    axios.get<*IActivity*[]>('http://localhost:5000/api/activities')

    .then((*response*)=>{

*this*.setState({

        activities: response.data

      })

    })

  }

  render(){

    return (

      <div>

        <*Header* *as*='h2' *icon*='users' *content*='Reactivities ' />

        <*List*>

          {*this*.state.activities.map((*activity*)=>(

              <*List.Item* *key*={activity.id}>{activity.title}</*List.Item*>

            ))}

        </*List*>

      </div>

    );

  }

}

export default App;

**Refactoring our class component to use React Hooks**

*- Nếu bạn đã quen thuộc với method của react lifecycle thì useEffect được xem là sự kết hợp của 3 đứa là componentDidMount, componentDidUpdate, vaf componentWillUnmount.*

*- Ở bước này ta sẽ chuyển một commponent ban đầu viết theo kiểu hướng class qua dạng hướng function, ở react mới thì sẽ viết theo kiểu hướng function.*

**Step 1:** Trong App.tsx ta refactoring lại như sau

import React, { useState, useEffect } from "react";

import { Header, List } from "semantic-ui-react";

import axios from "axios";

import { IActivity } from "../models/activity";

const App = () => {

  const [activities, setActivities] = useState<*IActivity*[]>([]);

  useEffect(() => {

    axios

      .get<*IActivity*[]>("http://localhost:5000/api/activities")

      .then(*response* => {

        setActivities(response.data);

      });

  },[]); // [] cái này cực kỳ quan trong,

        // ta hai báo một empty arry để chỉ rằng ta sử dụng effect  một lần thôi

        // bởi vì mỗi lần một component render thì nó sẽ gọi lại thằng effect này

  return (

    <div>

      <*Header* *as*="h2" *icon*="users" *content*="Reactivities " />

      <*List*>

        {activities.map(*activity* => (

          <*List.Item* *key*={activity.id}>{activity.title}</*List.Item*>

        ))}

      </*List*>

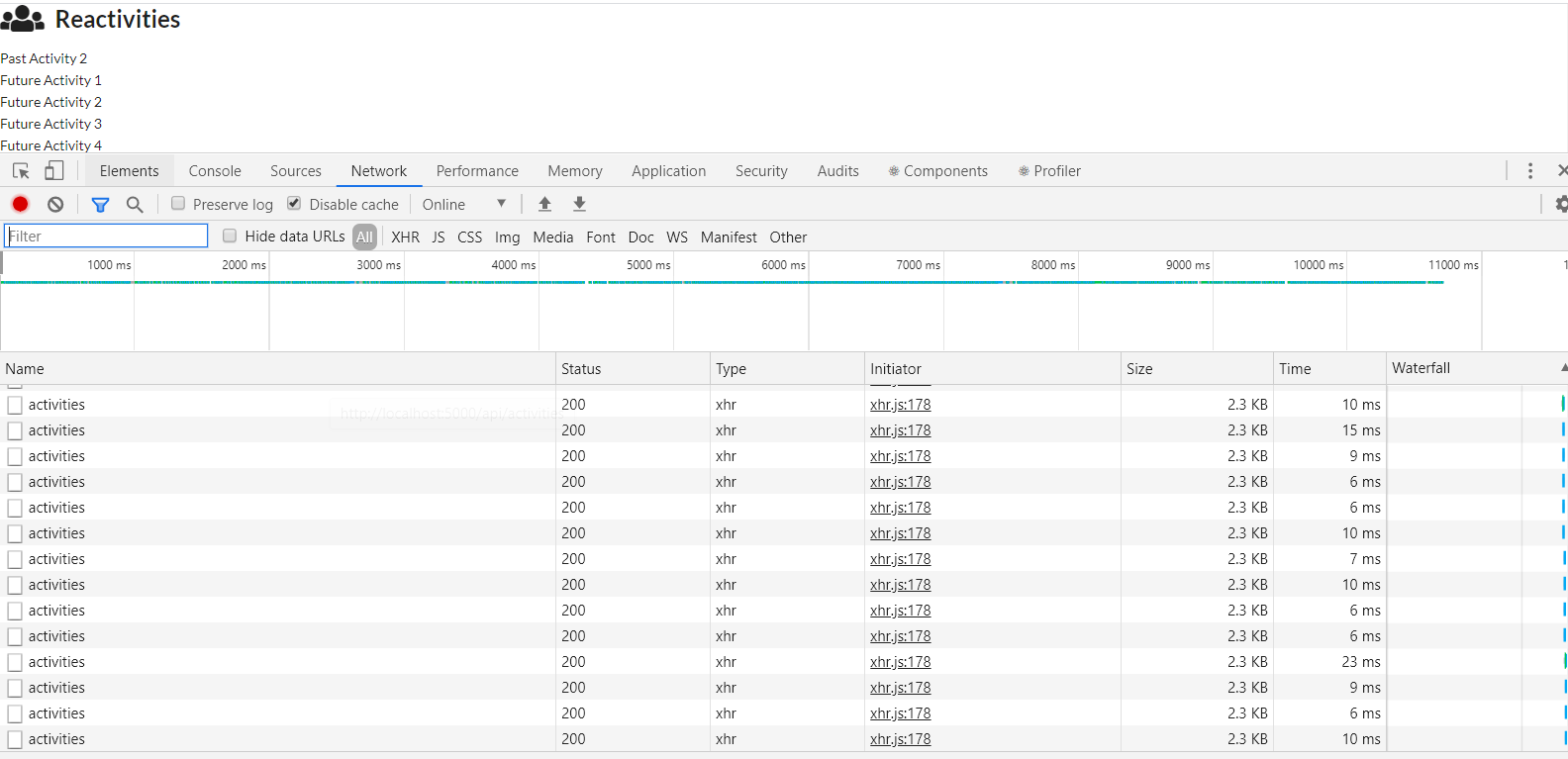
    </div>

  );

};

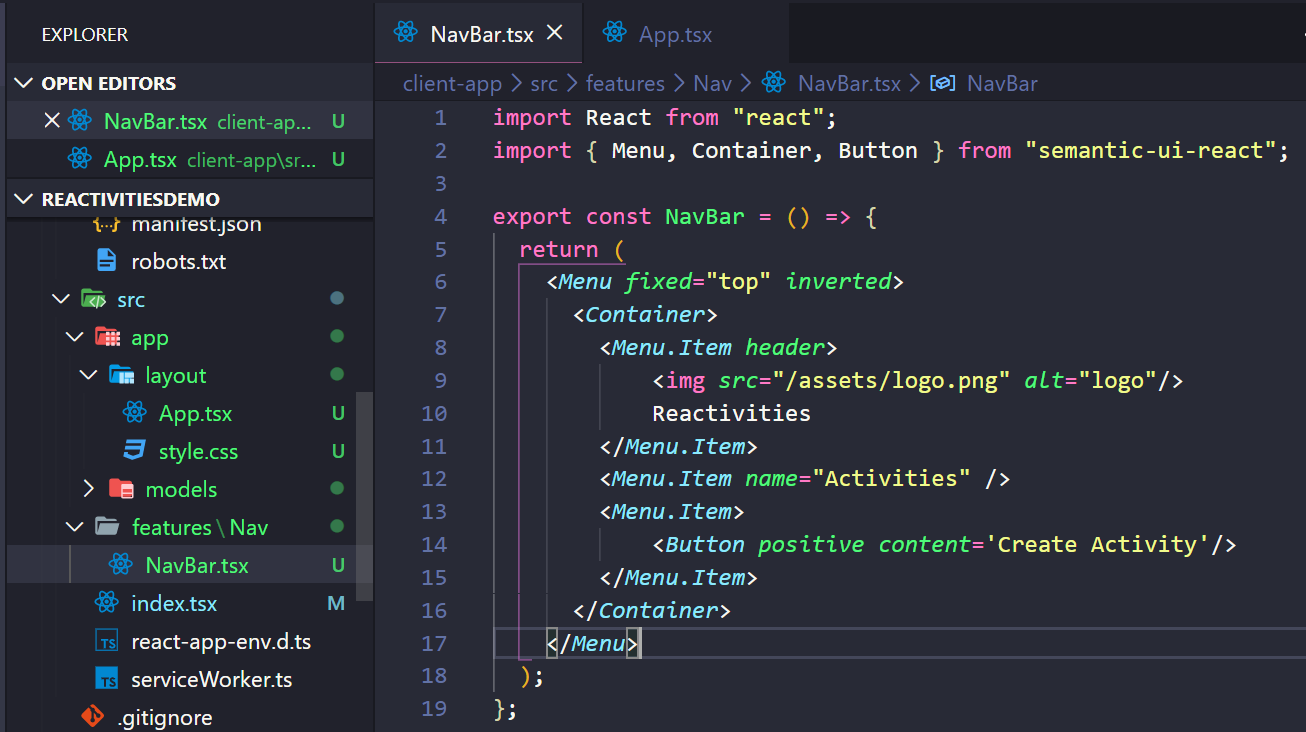
export default App;

Trong **useEffect** nếu thằng thông số thứ 2 ( **[]** ) không được khai báo thì nó sẽ liên tục gởi request về server để get dữ liệu như sau:



**Adding the navigation bar**

**Step 1:** Trong thư mục features ta tạo một thư mục Nav trong thư mục mới này ta thêm một component tên NavBar.tsx như sau:



**Step 2**: Trong App.tsx ta modify lại function return như sau: (ta thêm component NavBar thay cho header cũ trước đó).

return (

    <div>

      <*NavBar*/>

      <*List*>

        {activities.map(*activity* => (

          <*List.Item* *key*={activity.id}>{activity.title}</*List.Item*>

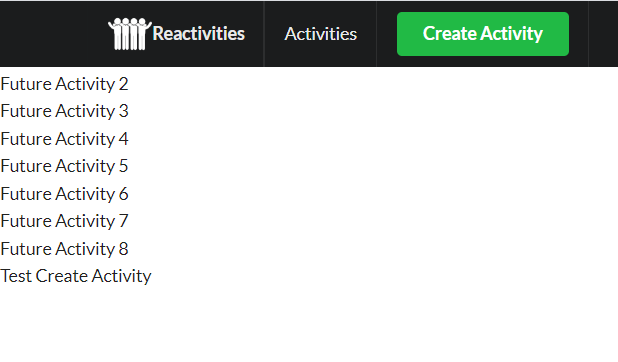
        ))}

      </*List*>

    </div>

  );

**Step 3:** Khi re-run lại ứng dụng ta sẽ thấy được giao diện mới như sau:



**Styling React components**

**Step 1:** Trong NavBar.tsx ta modify lại như sau:

import React from "react";

import { Menu, Container, Button } from "semantic-ui-react";

export const NavBar = () => {

  return (

    <*Menu* *fixed*="top" *inverted*>

      <*Container*>

        <*Menu.Item* *header*>

            <img *src*="/assets/logo.png" *alt*="logo" *style*={{marginRight:'10px'}}/>

            Reactivities

        </*Menu.Item*>

        <*Menu.Item* *name*="Activities" />

        <*Menu.Item*>

            <*Button* *positive* *content*='Create Activity'/>

        </*Menu.Item*>

      </*Container*>

    </*Menu*>

  );

};

**Step 2:** Trong App.tsx ta cũng modify lại như sau

import React, { useState, useEffect,Fragment } from "react";

import { List, Container } from "semantic-ui-react";

import axios from "axios";

import { IActivity } from "../models/activity";

import { NavBar } from "../../features/Nav/NavBar";

const App = () => {

  const [activities, setActivities] = useState<*IActivity*[]>([]);

  useEffect(() => {

    axios

      .get<*IActivity*[]>("http://localhost:5000/api/activities")

      .then(*response* => {

        setActivities(response.data);

      });

  }, []);

  return (

    <*Fragment*>

      <*NavBar* />

      <*Container* *style*={{marginTop:'7em'}}>

        <*List*>

          {activities.map(*activity* => (

            <*List.Item* *key*={activity.id}>{activity.title}</*List.Item*>

          ))}

        </*List*>

      </*Container*>

    </*Fragment*>

  );

};

export default App;

**Step 3:** Trong file style.css mà ta thêm vào ta thêm các styles như sau (bằng cách gõ snippet S\_5.6)

body {

  background-color: rgb(234, 234, 234) !important;

}

*.ui.inverted.top.fixed.menu* {

  background-image: linear-gradient(

    135deg,

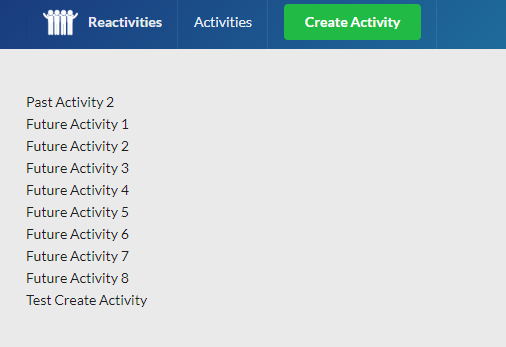
    rgb(24, 42, 115) 0%,

    rgb(33, 138, 174) 69%,

    rgb(32, 167, 172) 89%

  ) !important;

**Step 4:** Thực hiện run và xem kết quả



**Adding the activity Dashboard**

**Step 1**: trong App.tsx ta modify lại như sau:

import React, { useState, useEffect,Fragment } from "react";

import { List, Container } from "semantic-ui-react";

import axios from "axios";

import { IActivity } from "../models/activity";

import { NavBar } from "../../features/Nav/NavBar";

import { ActivityDashboard } from "../../features/activities/dashboard/ActivityDashboard";

const App = () => {

  const [activities, setActivities] = useState<*IActivity*[]>([]);

  useEffect(() => {

    axios

      .get<*IActivity*[]>("http://localhost:5000/api/activities")

      .then(*response* => {

        setActivities(response.data);

      });

  }, []);

  return (

    <*Fragment*>

      <*NavBar* />

      <*Container* *style*={{marginTop:'7em'}}>

        <*ActivityDashboard* *activities*={activities}></*ActivityDashboard*>

      </*Container*>

    </*Fragment*>

  );

};

export default App;

**Step 2**: Ta thêm mới một file ActivityDashboard.tsx như sau:



**Creating the Activity list**

**Step 1**: Trong file ActivityDashboard.tsx ta modify lại như sau:

import React from "react";

import { Grid, List } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

import { ActivityList } from "./ActivityList";

interface *IProps*{

    activities:*IActivity*[]

}

export const ActivityDashboard: *React*.*FC*<*IProps*> = ({*activities*}) => {

  return (

    <*Grid*>

      <*Grid.Column* *width*={10}>

        <*ActivityList* *activities*={activities}/>

      </*Grid.Column*>

    </*Grid*>

  );

};

**Step 2**: Trong thư mục dashboard ta thêm file ActivityList.tsx như sau:

import React from "react";

import { Item, Button, Label, Segment } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

interface *IProps* {

  activities: *IActivity*[];

}

export const ActivityList: *React*.*FC*<*IProps*> = ({ *activities* }) => {

  return (

    <*Segment* *clearing*>

      <*Item.Group* *divided*>

        {activities.map(*activity* => (

          <*Item* *key*={activity.id}>

            <*Item.Content*>

              <*Item.Header* *as*="a">{activity.title}</*Item.Header*>

              <*Item.Meta*>{activity.date}</*Item.Meta*>

              <*Item.Description*>

                <div>{activity.description}</div>

                <div>

                  {activity.city}, {activity.venue}

                </div>

              </*Item.Description*>

              <*Item.Extra*>

                <*Button* *floated*="right" *content*="View" *color*="blue" />

                <*Label* *basic* *content*={activity.category} />

              </*Item.Extra*>

            </*Item.Content*>

          </*Item*>

        ))}

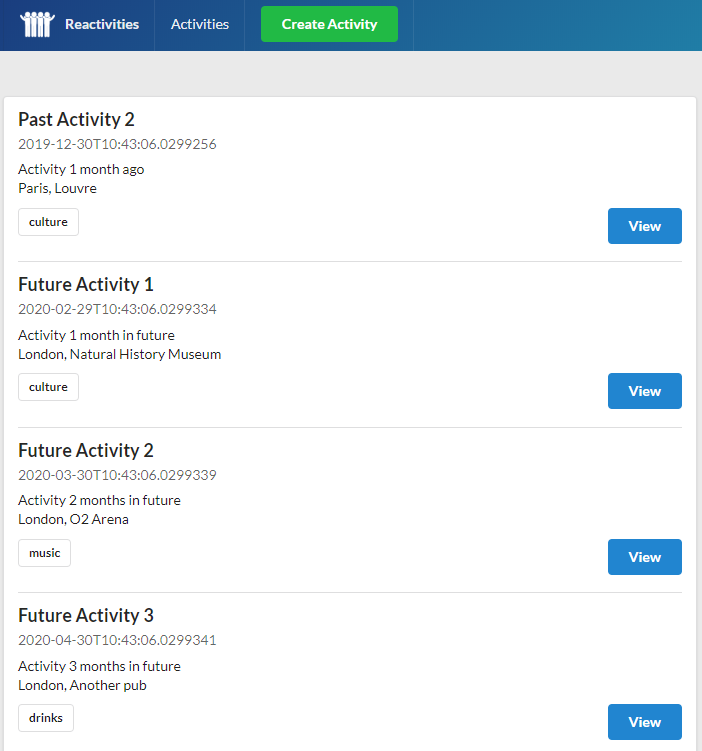
      </*Item.Group*>

    </*Segment*>

  );

};

**Step 3**: Khi thực thi lại chương trình ta sẽ thấy kết quả như sau:



**Adding the activity Details component**

**Step 1:** Trong file ActivityDashboard.tsx ta modify lại như sau:

import React from "react";

import { Grid, List } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

import { ActivityList } from "./ActivityList";

import { ActivityDetails } from "../details/ActivityDetails";

interface *IProps*{

    activities:*IActivity*[]

}

export const ActivityDashboard: *React*.*FC*<*IProps*> = ({*activities*}) => {

  return (

    <*Grid*>

      <*Grid.Column* *width*={10}>

        <*ActivityList* *activities*={activities}/>

      </*Grid.Column*>

      <*Grid.Column* *width*={6}>

        <*ActivityDetails*/>

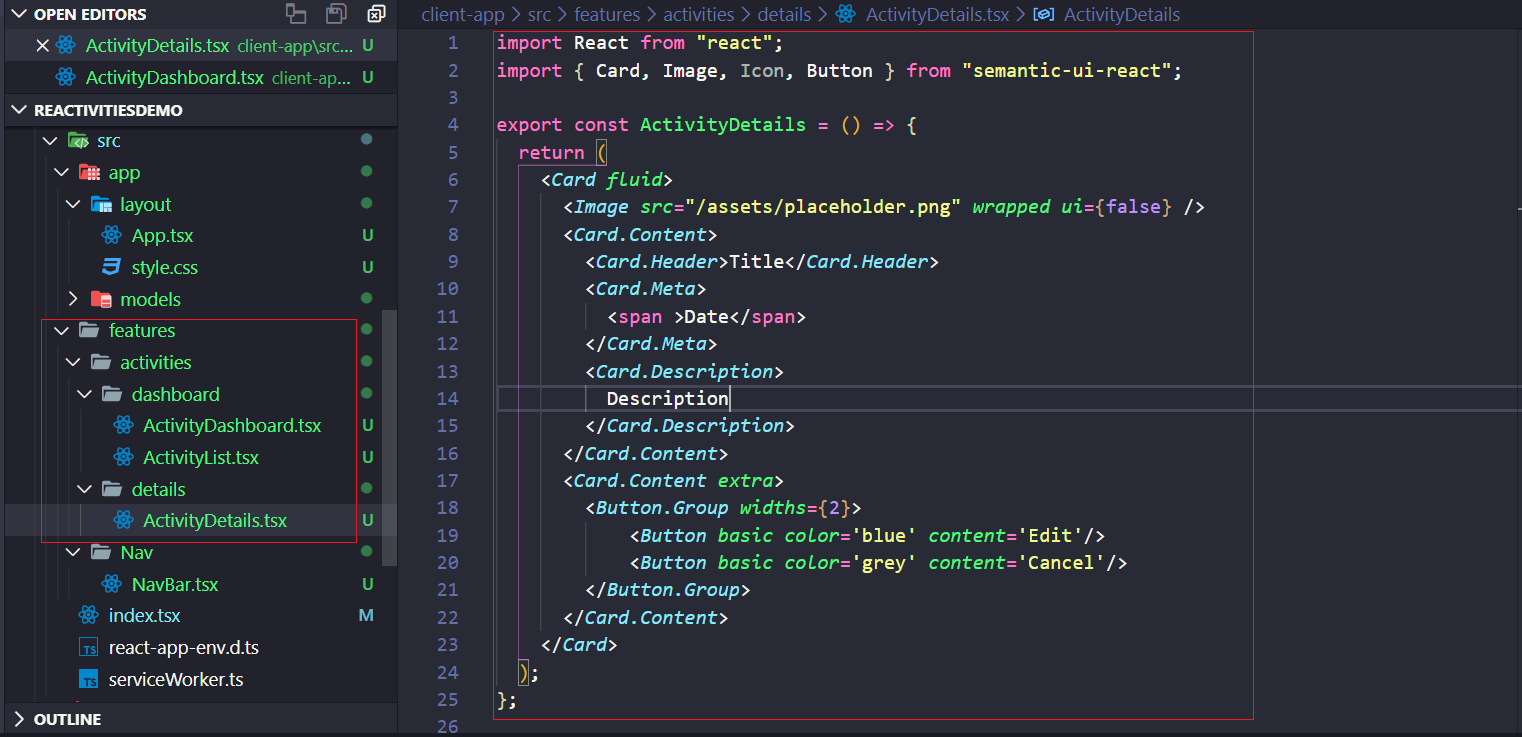
      </*Grid.Column*>

    </*Grid*>

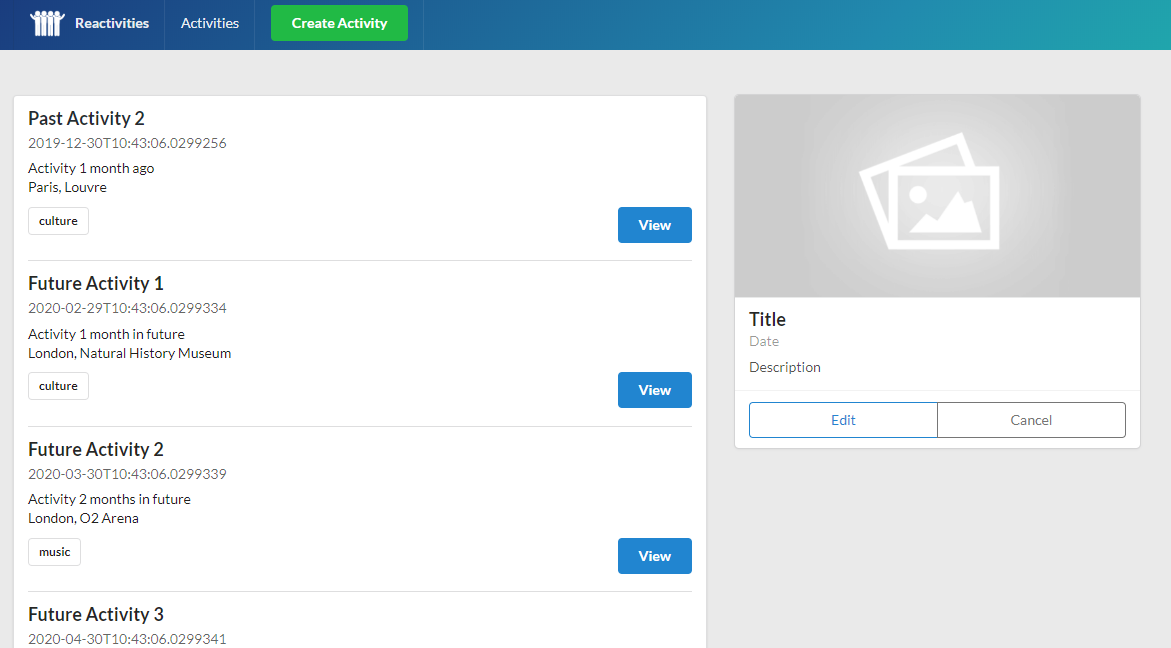
  );

};

**Step 2:** Trong thư mục Dashboard ta tạo mới một thư mục là details với một file tên là ActivityDetails.tsx như sau:



**Step 3:** Ta xem kết quả sẻ như sau:



**Adding the activity form component**

**Step 1:** trong file ActivityDashboard.tsx ta modify như sau:

import React from "react";

import { Grid, List } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

import { ActivityList } from "./ActivityList";

import { ActivityDetails } from "../details/ActivityDetails";

import { ActivityForm } from "../form/ActivityForm";

interface *IProps*{

    activities:*IActivity*[]

}

export const ActivityDashboard: *React*.*FC*<*IProps*> = ({*activities*}) => {

  return (

    <*Grid*>

      <*Grid.Column* *width*={10}>

        <*ActivityList* *activities*={activities}/>

      </*Grid.Column*>

      <*Grid.Column* *width*={6}>

        <*ActivityDetails*/>

        <*ActivityForm*/>

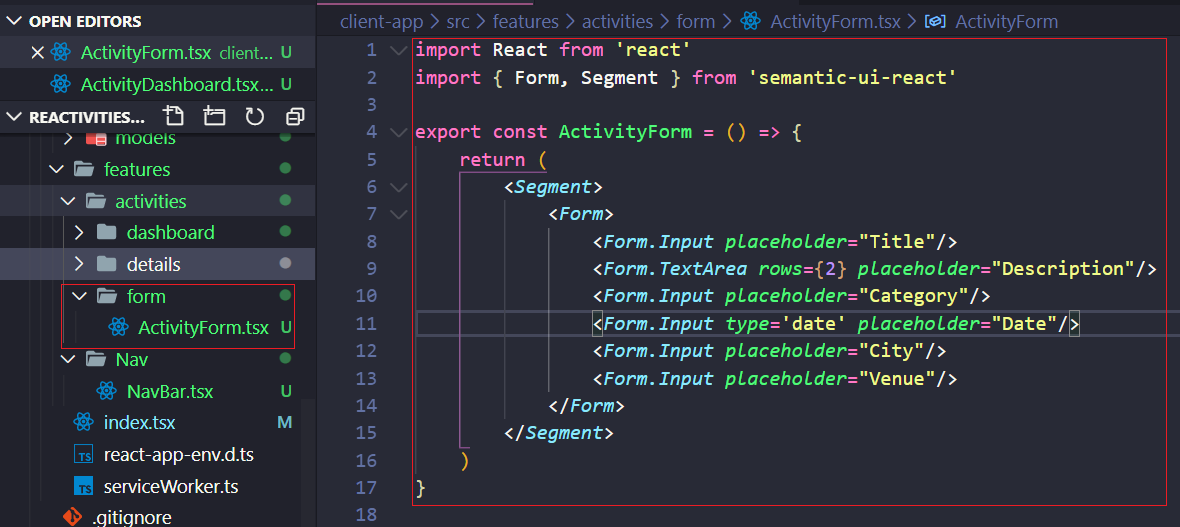
      </*Grid.Column*>

    </*Grid*>

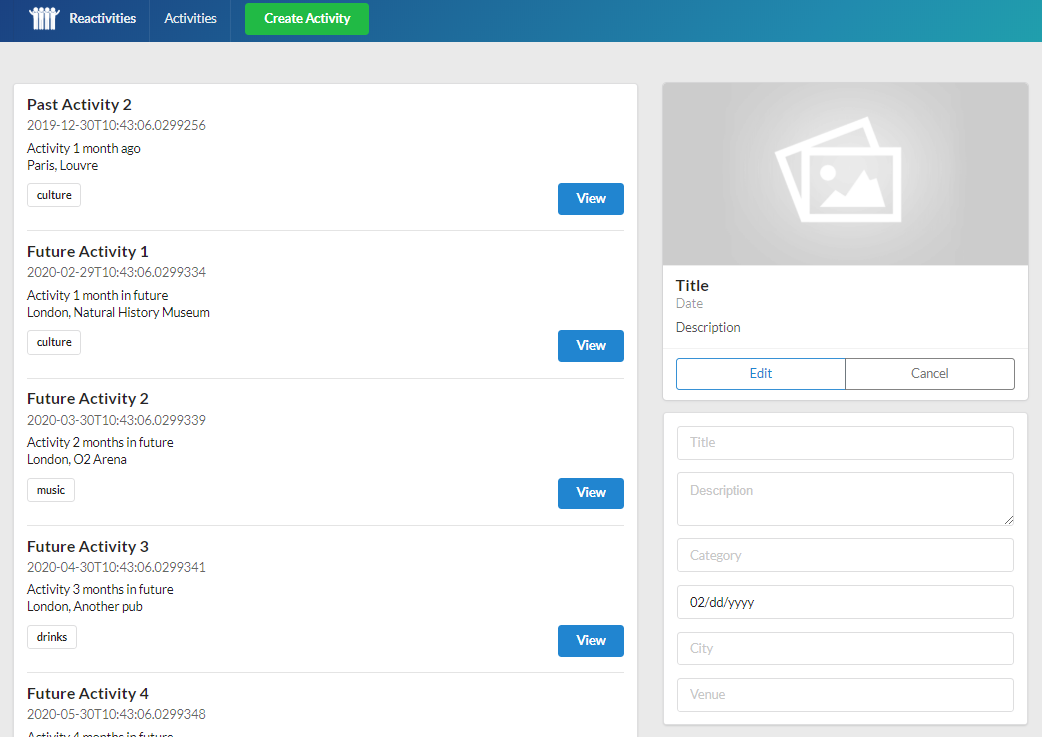
  );

};

**Step 2:** Trong thư mục activities ta thêm mới một thư mục là form và một file tên là ActivityForm.tsx như sau:



**Step 3:** Ta sẽ được giao diện như sau



**Selecting an individual Activity**

**Step 1:** trong App.tsx ta modify như sau:

import React, { useState, useEffect, Fragment } from "react";

import { List, Container } from "semantic-ui-react";

import axios from "axios";

import { IActivity } from "../models/activity";

import { NavBar } from "../../features/Nav/NavBar";

import { ActivityDashboard } from "../../features/activities/dashboard/ActivityDashboard";

const App = () => {

  const [activities, setActivities] = useState<*IActivity*[]>([]);

  const [selectedActivity, setSelectedActivity] = useState<*IActivity* | *null*>(

    null

  );

  const handleSelectedActivity = (*id*: *string*) => {

    setSelectedActivity(activities.filter(*a* => a.id === id)[0]);

  };

  useEffect(() => {

    axios

      .get<*IActivity*[]>("http://localhost:5000/api/activities")

      .then(*response* => {

        setActivities(response.data);

      });

  }, []);

  return (

    <*Fragment*>

      <*NavBar* />

      <*Container* *style*={{ marginTop: "7em" }}>

        <*ActivityDashboard*

*activities*={activities}

*selectActivity*={handleSelectedActivity}

*selectedActivity*={selectedActivity!}

          // dấu ! phía trên để nói rằng nó có thể là null

          // Ta có thể khai bao phía trên như sau

          // selectedActivity: IActivity | null;

        ></*ActivityDashboard*>

      </*Container*>

    </*Fragment*>

  );

};

export default App;

**Step 2:** trong ActivityDashBoard.tsx ta modify như sau:

import React from "react";

import { Grid, List } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

import { ActivityList } from "./ActivityList";

import { ActivityDetails } from "../details/ActivityDetails";

import { ActivityForm } from "../form/ActivityForm";

interface *IProps* {

  activities: *IActivity*[];

  selectActivity: (*id*: *string*) => *void*;

  selectedActivity: *IActivity*;

}

export const ActivityDashboard: *React*.*FC*<*IProps*> = ({

  activities,

  selectActivity,

  selectedActivity

}) => {

  return (

    <*Grid*>

      <*Grid.Column* *width*={10}>

        <*ActivityList* *activities*={activities} *selectActivity*={selectActivity} />

      </*Grid.Column*>

      <*Grid.Column* *width*={6}>

        {selectedActivity && <*ActivityDetails* *activity*={selectedActivity} />}

        <*ActivityForm* />

      </*Grid.Column*>

    </*Grid*>

  );

};

**Step 3:** Trong file ActivityList.tsx ta modify như sau:

import React from "react";

import { Item, Button, Label, Segment } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

interface *IProps* {

  activities: *IActivity*[];

  selectActivity: (*id*: *string*) => *void*;

}

export const ActivityList: *React*.*FC*<*IProps*> = ({

  activities,

  selectActivity

}) => {

  return (

    <*Segment* *clearing*>

      <*Item.Group* *divided*>

        {activities.map(*activity* => (

          <*Item* *key*={activity.id}>

            <*Item.Content*>

              <*Item.Header* *as*="a">{activity.title}</*Item.Header*>

              <*Item.Meta*>{activity.date}</*Item.Meta*>

              <*Item.Description*>

                <div>{activity.description}</div>

                <div>

                  {activity.city}, {activity.venue}

                </div>

              </*Item.Description*>

              <*Item.Extra*>

                <*Button*

*onClick*={() => selectActivity(activity.id)}

*floated*="right"

*content*="View"

*color*="blue"

                />

                <*Label* *basic* *content*={activity.category} />

              </*Item.Extra*>

            </*Item.Content*>

          </*Item*>

        ))}

      </*Item.Group*>

    </*Segment*>

  );

};

**Step 4:** Trong ActivityDetails.tsx ta modify như sau:

import React from "react";

import { Card, Image, Icon, Button } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

interface *IProps* {

  activity: *IActivity*;

}

export const ActivityDetails: *React*.*FC*<*IProps*> = ({ *activity* }) => {

  return (

    <*Card* *fluid*>

      <*Image*

*src*={`/assets/categoryImages/${activity.category}.jpg`}

*wrapped*

*ui*={false}

      />

      <*Card.Content*>

        <*Card.Header*>{activity.title}</*Card.Header*>

        <*Card.Meta*>

          <span>{activity.date}</span>

        </*Card.Meta*>

        <*Card.Description*>{activity.description}</*Card.Description*>

      </*Card.Content*>

      <*Card.Content* *extra*>

        <*Button.Group* *widths*={2}>

          <*Button* *basic* *color*="blue" *content*="Edit" />

          <*Button* *basic* *color*="grey" *content*="Cancel" />

        </*Button.Group*>

      </*Card.Content*>

    </*Card*>

  );

};

**Adding an edit mode to display the form**

Step 1: trong App.tsx ta modify lại như sau:

import React, { useState, useEffect, Fragment } from "react";

import { List, Container } from "semantic-ui-react";

import axios from "axios";

import { IActivity } from "../models/activity";

import { NavBar } from "../../features/Nav/NavBar";

import { ActivityDashboard } from "../../features/activities/dashboard/ActivityDashboard";

const App = () => {

  const [activities, setActivities] = useState<*IActivity*[]>([]);

  const [selectedActivity, setSelectedActivity] = useState<*IActivity* | *null*>(

    null

  );

  const [editMode, setEditMode] = useState(false);

  const handleSelectedActivity = (*id*: *string*) => {

    setSelectedActivity(activities.filter(*a* => a.id === id)[0]);

  };

  useEffect(() => {

    axios

      .get<*IActivity*[]>("http://localhost:5000/api/activities")

      .then(*response* => {

        setActivities(response.data);

      });

  }, []);

  return (

    <*Fragment*>

      <*NavBar* />

      <*Container* *style*={{ marginTop: "7em" }}>

        <*ActivityDashboard*

*activities*={activities}

*selectActivity*={handleSelectedActivity}

*selectedActivity*={selectedActivity!}

*editMode* = {editMode}

*setEditMode* = {setEditMode}

        ></*ActivityDashboard*>

      </*Container*>

    </*Fragment*>

  );

};

export default App;

**Step 2:** Trong file ActivityDashBoard.tsx ta modify lại như sau:

import React from "react";

import { Grid, List } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

import { ActivityList } from "./ActivityList";

import { ActivityDetails } from "../details/ActivityDetails";

import { ActivityForm } from "../form/ActivityForm";

interface *IProps* {

  activities: *IActivity*[];

  selectActivity: (*id*: *string*) => *void*;

  selectedActivity: *IActivity*;

  editMode: *boolean*;

  setEditMode: (*editMode*: *boolean*) => *void*;

}

export const ActivityDashboard: *React*.*FC*<*IProps*> = ({

  activities,

  selectActivity,

  selectedActivity,

  editMode,

  setEditMode

}) => {

  return (

    <*Grid*>

      <*Grid.Column* *width*={10}>

        <*ActivityList* *activities*={activities} *selectActivity*={selectActivity} />

      </*Grid.Column*>

      <*Grid.Column* *width*={6}>

        {selectedActivity && !editMode && (

        <*ActivityDetails* *activity*={selectedActivity} *setEditMode*={setEditMode} />)}

        {editMode && <*ActivityForm* />}

      </*Grid.Column*>

    </*Grid*>

  );

};

**Step 3:** Trong ActivityDetail.tsx ta modify lại như sau:

import React from "react";

import { Card, Image, Icon, Button } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

interface *IProps* {

  activity: *IActivity*;

  setEditMode: (*editMode*: *boolean*)=>*void*;

}

export const ActivityDetails: *React*.*FC*<*IProps*> = ({ *activity*, *setEditMode* }) => {

  return (

    <*Card* *fluid*>

      <*Image*

*src*={`/assets/categoryImages/${activity.category}.jpg`}

*wrapped*

*ui*={false}

      />

      <*Card.Content*>

        <*Card.Header*>{activity.title}</*Card.Header*>

        <*Card.Meta*>

          <span>{activity.date}</span>

        </*Card.Meta*>

        <*Card.Description*>{activity.description}</*Card.Description*>

      </*Card.Content*>

      <*Card.Content* *extra*>

        <*Button.Group* *widths*={2}>

          <*Button* *onClick*={()=>setEditMode(true)} *basic* *color*="blue" *content*="Edit" />

          <*Button* *basic* *color*="grey" *content*="Cancel" />

        </*Button.Group*>

      </*Card.Content*>

    </*Card*>

  );

};

**Adding a create activity mode**

**Step 1:** Trong App.tsx ta modify lại như sau:

import React, { useState, useEffect, Fragment } from "react";

import { List, Container } from "semantic-ui-react";

import axios from "axios";

import { IActivity } from "../models/activity";

import { NavBar } from "../../features/Nav/NavBar";

import { ActivityDashboard } from "../../features/activities/dashboard/ActivityDashboard";

const App = () => {

  const [activities, setActivities] = useState<*IActivity*[]>([]);

  const [selectedActivity, setSelectedActivity] = useState<*IActivity* | *null*>(

    null

  );

  const [editMode, setEditMode] = useState(false);

  const handleSelectedActivity = (*id*: *string*) => {

    setSelectedActivity(activities.filter(*a* => a.id === id)[0]);

  };

  const handleOpenCreateForm = () => {

    setSelectedActivity(null);

    setEditMode(true);

  }

  useEffect(() => {

    axios

      .get<*IActivity*[]>("http://localhost:5000/api/activities")

      .then(*response* => {

        setActivities(response.data);

      });

  }, []);

  return (

    <*Fragment*>

      <*NavBar* *openCreateForm*={handleOpenCreateForm} />

      <*Container* *style*={{ marginTop: "7em" }}>

        <*ActivityDashboard*

*activities*={activities}

*selectActivity*={handleSelectedActivity}

*selectedActivity*={selectedActivity!}

*editMode* = {editMode}

*setEditMode* = {setEditMode}

*setSelectedActivity* = {setSelectedActivity}

        ></*ActivityDashboard*>

      </*Container*>

    </*Fragment*>

  );

};

export default App;

**Step 2**: Trong NavBar.tsx ta modify lại như sau:

import React from "react";

import { Menu, Container, Button } from "semantic-ui-react";

interface *IProps*{

  openCreateForm: () => *void*;

}

export const NavBar: *React*.*FC*<*IProps*> = ({*openCreateForm*}) => {

  return (

    <*Menu* *fixed*="top" *inverted*>

      <*Container*>

        <*Menu.Item* *header*>

            <img *src*="/assets/logo.png" *alt*="logo" *style*={{marginRight:'10px'}}/>

            Reactivities

        </*Menu.Item*>

        <*Menu.Item* *name*="Activities" />

        <*Menu.Item*>

            <*Button* *onClick*={openCreateForm} *positive* *content*='Create Activity'/>

        </*Menu.Item*>

      </*Container*>

    </*Menu*>

  );

};

**Step 3:** Trong file ActivityForm.tsx ta modify lại như sau:

import React from 'react'

import { Form, Segment, Button } from 'semantic-ui-react'

interface *IProps*{

    setEditMode: (*editMode*: *boolean*) => *void*;

}

export const ActivityForm: *React*.*FC*<*IProps*> = ({*setEditMode*}) => {

    return (

        <*Segment* *clearing*>

            <*Form*>

                <*Form.Input* *placeholder*="Title"/>

                <*Form.TextArea* *rows*={2} *placeholder*="Description"/>

                <*Form.Input* *placeholder*="Category"/>

                <*Form.Input* *type*='date' *placeholder*="Date"/>

                <*Form.Input* *placeholder*="City"/>

                <*Form.Input* *placeholder*="Venue"/>

                <*Button* *floated* ='right' *positive* *type*='submit' *content*='Submit'/>

                <*Button* *onClick*={()=>setEditMode(false)} *floated* ='right' *type*='button' *content*='Cancel'/>

            </*Form*>

        </*Segment*>

    )

}

**Step 4:** trong ActivityDashboard.tsx ta modify lại như sau:

import React from "react";

import { Grid, List } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

import { ActivityList } from "./ActivityList";

import { ActivityDetails } from "../details/ActivityDetails";

import { ActivityForm } from "../form/ActivityForm";

interface *IProps* {

  activities: *IActivity*[];

  selectActivity: (*id*: *string*) => *void*;

  selectedActivity: *IActivity*;

  editMode: *boolean*;

  setEditMode: (*editMode*: *boolean*) => *void*;

  setSelectedActivity: (*activity*: *IActivity* | *null*) => *void*;

}

export const ActivityDashboard: *React*.*FC*<*IProps*> = ({

  activities,

  selectActivity,

  selectedActivity,

  editMode,

  setEditMode,

  setSelectedActivity

}) => {

  return (

    <*Grid*>

      <*Grid.Column* *width*={10}>

        <*ActivityList* *activities*={activities} *selectActivity*={selectActivity} />

      </*Grid.Column*>

      <*Grid.Column* *width*={6}>

        {selectedActivity && !editMode && (

        <*ActivityDetails*

*activity*={selectedActivity}

*setEditMode*={setEditMode}

*setSelectedActivity* = {setSelectedActivity} />)}

        {editMode && <*ActivityForm* *setEditMode*={setEditMode} />}

      </*Grid.Column*>

    </*Grid*>

  );

};

**Step 5:** Trong ActivityDetails.tsx ta modify lại như sau:

import React from "react";

import { Card, Image, Icon, Button } from "semantic-ui-react";

import { IActivity } from "../../../app/models/activity";

interface *IProps* {

  activity: *IActivity*;

  setEditMode: (*editMode*: *boolean*)=>*void*;

  setSelectedActivity: (*activity*: *IActivity* | *null*) => *void*;

}

export const ActivityDetails: *React*.*FC*<*IProps*> = ({ *activity*, *setEditMode*, *setSelectedActivity* }) => {

  return (

    <*Card* *fluid*>

      <*Image*

*src*={`/assets/categoryImages/${activity.category}.jpg`}

*wrapped*

*ui*={false}

      />

      <*Card.Content*>

        <*Card.Header*>{activity.title}</*Card.Header*>

        <*Card.Meta*>

          <span>{activity.date}</span>

        </*Card.Meta*>

        <*Card.Description*>{activity.description}</*Card.Description*>

      </*Card.Content*>

      <*Card.Content* *extra*>

        <*Button.Group* *widths*={2}>

          <*Button* *onClick*={()=>setEditMode(true)} *basic* *color*="blue" *content*="Edit" />

          <*Button* *onClick*={()=>setSelectedActivity(null)} *basic* *color*="grey" *content*="Cancel" />

        </*Button.Group*>

      </*Card.Content*>

    </*Card*>

  );

};