Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6

REST сервис

тема

|  |  |  |
| --- | --- | --- |
| Преподаватель |  | А.К. Погребников |
|  | подпись, дата | инициалы, фамилия |
| Студент КИ18-16б 031831229 |  | В.А. Прекель |
| номер группы, зачетной книжки | подпись, дата | инициалы, фамилия |

Красноярск 2020

# 1 Цель работы

В рамках данной практической работы необходимо реализовать REST сервис.

# 2 Общая постановка задачи

В рамках данной практической работы необходимо реализовать REST сервис. Разработайте демонстрационное приложение (используя любые изученные технологии), которое использует методы GET, POST, DELETE реализованного REST-API, а также способно отображать полученный через GET результат в виде таблицы.

# 3 Исходный код

Листинг 1 – MyStore.WebApi\Controllers\CartsController.fs

namespace MyStore.WebApi.Controllers  
  
open System.Collections.Generic  
open Microsoft.AspNetCore.Mvc  
open Microsoft.Extensions.Logging  
open MyStore.Data  
open MyStore.Data.Entity  
open MyStore.WebApi.Repository  
open MyStore.WebApi.Utils  
  
[<ApiController>]  
[<Route("[controller]")>]  
type CartsController(logger: ILogger<CartsController>, context: Context) =  
 inherit ControllerBase()  
  
 [<HttpGet("{id}")>]  
 member this.GetById(id) =  
 ActionResult.ofAsyncTA ActionResult<Cart>  
 <| async {  
 if Carts.exists context id  
 then return this.Ok(Carts.exactlyOne context id) :> \_  
 else return this.NotFound() :> \_  
 }  
  
  
 [<HttpGet("{id}/products")>]  
 member this.GetCartProducts(id) =  
 ActionResult.ofAsyncTA ActionResult<IEnumerable<Product>>  
 <| async {  
 match Carts.exists context id with  
 | true ->  
 return  
 this.Ok  
 ((Carts.exactlyOneIncludeProducts context id)  
 .Products) :> \_  
 | false -> return this.NotFound() :> \_  
 }  
  
  
 [<HttpPut("{id}/owner/{ownerId}")>]  
 member this.SetOwner(id, [<FromQuery>] ownerId) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 if (Carts.exists context id  
 && Customers.exists context ownerId) then  
 let cart = Carts.exactlyOne context id  
  
 cart.OwnerCustomerId <- ownerId  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.NoContent() :> \_  
 else  
 return this.NotFound() :> \_  
 }  
  
 [<HttpPut("{id}")>]  
 member this.Update(id, [<FromBody>] cart: Cart) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 if Carts.exists context id then  
 cart.CartId <- id  
  
 context.Carts.Update(cart) |> ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.NoContent() :> \_  
 else  
 return this.NotFound() :> \_  
 }  
  
 [<HttpPost>]  
 member this.Create([<FromBody>] cart: Cart) =  
 ActionResult.ofAsyncTA ActionResult<Cart>  
 <| async {  
 do! context.Carts.AddAsync(cart).AsTask()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.Created($"carts/{cart.CartId}", cart) :> \_  
 }  
  
 [<HttpPut("{id}/products/{productId}")>]  
 member this.AddProduct(id, productId) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 if (Carts.exists context id  
 && Products.exists context productId) then  
 let cart = Carts.exactlyOne context id  
  
 let product = Products.exactlyOne context productId  
  
 cart.Products.Add(product)  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.Ok() :> \_  
 else  
 return this.NotFound() :> \_  
 }

Листинг 2 – MyStore.WebApi\Controllers\CustomersController.fs

п»їnamespace MyStore.WebApi.Controllers  
  
open System  
open System.Collections.Generic  
open Microsoft.AspNetCore.Mvc  
open Microsoft.Extensions.Logging  
open MyStore.Data  
open MyStore.Data.Entity  
open MyStore.WebApi.Utils  
open MyStore.WebApi.Repository  
  
[<ApiController>]  
[<Route("[controller]")>]  
type CustomersController(logger: ILogger<CustomersController>, context: Context) =  
 inherit ControllerBase()  
  
 [<HttpGet>]  
 member this.GetOffset([<FromQuery>] start: Nullable<int>, [<FromQuery>] limit: Nullable<int>) =  
 ActionResult.ofAsyncTA ActionResult<IEnumerable<Customer>>  
 <| async {  
 return  
 this.Ok  
 (nullableLimitStartToSkipTake (start, limit)  
 |> Customers.skipTake context) :> \_  
 }  
  
  
 [<HttpGet("{id}")>]  
 member this.GetById(id) =  
 ActionResult.ofAsyncTA ActionResult<Customer>  
 <| async {  
 if (Customers.exists context id)  
 then return this.Ok(Customers.exactlyOne context id) :> \_  
 else return this.NotFound() :> \_  
 }  
  
 [<HttpDelete("{id}")>]  
 member this.DeleteById(id) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 if Customers.exists context id then  
 context.Customers.Remove(Customers.exactlyOne context id)  
 |> ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.NoContent() :> \_  
 else  
 return this.NotFound() :> \_  
 }  
  
  
 [<HttpPut("{id}")>]  
 member this.Update(id, [<FromBody>] customer: Customer) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 if Customers.exists context id then  
 customer.CustomerId <- id  
  
 context.Customers.Update(customer) |> ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.NoContent() :> \_  
 else  
 return this.NotFound() :> \_  
 }  
  
 [<HttpPost>]  
 member this.Add([<FromBody>] customer: Customer, [<FromQuery>] password) =  
 ActionResult.ofAsyncTA ActionResult<Customer>  
 <| async {  
 customer.PasswordSalt <- Crypto.GenerateSaltForPassword()  
 customer.PasswordHash <- Crypto.ComputePasswordHash(password, customer.PasswordSalt)  
  
 do! context.Customers.AddAsync(customer).AsTask()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
  
 return this.Created($"customers/{customer.CustomerId}", customer) :> \_  
 }

Листинг 3 – MyStore.WebApi\Controllers\OrdersController.fs

namespace MyStore.WebApi.Controllers  
  
open System  
open System.Collections.Generic  
open Microsoft.AspNetCore.Mvc  
open Microsoft.Extensions.Logging  
open MyStore.Data  
open MyStore.Data.Entity  
open MyStore.Data.Entity  
open MyStore.WebApi.Repository  
open MyStore.WebApi.Utils  
  
[<ApiController>]  
[<Route("[controller]")>]  
type OrdersController(logger: ILogger<OrdersController>, context: Context) =  
 inherit ControllerBase()  
  
 [<HttpGet("{id}")>]  
 member this.GetById(id) =  
 ActionResult.ofAsyncTA ActionResult<Order>  
 <| async {  
 match Orders.exists context id with  
 | true -> return this.Ok(Orders.exactlyOne context id) :> \_  
 | false -> return this.NotFound() :> \_  
 }  
  
  
 [<HttpGet>]  
 member this.GetOffset([<FromQuery>] start: Nullable<int>, [<FromQuery>] limit: Nullable<int>) =  
 ActionResult.ofAsyncTA ActionResult<IEnumerable<Order>>  
 <| async {  
 return  
 this.Ok  
 (nullableLimitStartToSkipTake (start, limit)  
 |> Orders.skipTake context) :> \_  
 }  
  
 [<HttpPost>]  
 member this.Create([<FromBody>] order) =  
 ActionResult.ofAsyncTA ActionResult<Order>  
 <| async {  
 do! context.Orders.AddAsync(order).AsTask()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
  
 return this.Created($"orders/{order.OrderId}", order) :> \_  
 }  
  
 [<HttpPut("{id}")>]  
 member this.Update(id, [<FromBody>] order: Order) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 if Orders.exists context id then  
 order.OrderId <- id  
  
 context.Orders.Update(order) |> ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.NoContent() :> \_  
 else  
 return this.NotFound() :> \_  
 }  
  
 [<HttpDelete("{id}")>]  
 member this.DeleteById(id) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 if Orders.exists context id then  
 context.Orders.Remove(Orders.exactlyOne context id)  
 |> ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.NoContent() :> \_  
 else  
 return this.NotFound() :> \_  
 }  
  
 [<HttpGet("{id}/orderedProducts")>]  
 member this.GetOrderedProducts(id) =  
 ActionResult.ofAsyncTA ActionResult<IEnumerable<OrderedProduct>>  
 <| async {  
 match Orders.exists context id with  
 | true ->  
 return  
 this.Ok  
 ((Orders.exactlyOneIncludeOrderedProducts context id)  
 .OrderedProducts) :> \_  
 | false -> return this.NotFound() :> \_  
 }  
  
  
 [<HttpPost("{id}/orderedProducts/{productId}")>]  
 member this.AddProduct(id, productId) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 if (Orders.exists context id  
 && Products.exists context productId) then  
  
 let order = Orders.exactlyOne context id  
 let product = Products.exactlyOne context productId  
  
 let orderedProduct =  
 OrderedProduct(ProductId = product.ProductId, OrderId = order.OrderId, OrderedPrice = product.Price)  
  
 do! context  
 .OrderedProducts  
 .AddAsync(orderedProduct)  
 .AsTask()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 order.OrderedProducts.Add(orderedProduct)  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.Created($"{id}/orderedProducts/{productId}", orderedProduct) :> \_  
 else  
 return this.NotFound() :> \_  
 }  
  
 [<HttpGet("{id}/orderedProducts/{productId}")>]  
 member this.GetProduct(id, productId) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 match OrderedProducts.exists context id productId with  
 | true -> return this.Ok(OrderedProducts.exactlyOne context id productId) :> \_  
 | false -> return this.NotFound() :> \_  
 }

Листинг 4 – MyStore.WebApi\Controllers\ProductsController.fs

namespace MyStore.WebApi.Controllers  
  
open System  
open System.Collections.Generic  
open Microsoft.AspNetCore.Mvc  
open Microsoft.Extensions.Logging  
open MyStore.Data  
open MyStore.Data.Entity  
open MyStore.WebApi.Repository  
open MyStore.WebApi.Utils  
  
[<ApiController>]  
[<Route("[controller]")>]  
type ProductsController(logger: ILogger<ProductsController>, context: Context) =  
 inherit ControllerBase()  
  
 [<HttpGet("{id}")>]  
 member this.GetById(id) =  
 ActionResult.ofAsyncTA ActionResult<Product>  
 <| async {  
 match Products.exists context id with  
 | true -> return this.Ok(Products.exactlyOne context id) :> \_  
 | false -> return this.NotFound() :> \_  
 }  
  
  
 [<HttpGet>]  
 member this.GetOffset([<FromQuery>] start: Nullable<int>, [<FromQuery>] limit: Nullable<int>) =  
 ActionResult.ofAsyncTA ActionResult<IEnumerable<Product>>  
 <| async {  
 return  
 this.Ok  
 (nullableLimitStartToSkipTake (start, limit)  
 |> Products.skipTake context) :> \_  
 }  
  
 [<HttpPost>]  
 member this.Create([<FromBody>] product) =  
 ActionResult.ofAsyncTA ActionResult<Product>  
 <| async {  
 do! context.Products.AddAsync(product).AsTask()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
  
 return this.Created($"products/{product.ProductId}", product) :> \_  
 }  
  
 [<HttpPut("{id}")>]  
 member this.Update(id, [<FromBody>] product: Product) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 if Products.exists context id then  
 product.ProductId <- id  
  
 context.Products.Update(product) |> ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.NoContent() :> \_  
 else  
 return this.NotFound() :> \_  
 }  
  
 [<HttpDelete("{id}")>]  
 member this.DeleteById(id) =  
 ActionResult.ofAsyncTA ActionResult<unit>  
 <| async {  
 if Products.exists context id then  
 context.Products.Remove(Products.exactlyOne context id)  
 |> ignore  
  
 do! context.SaveChangesAsync()  
 |> Async.AwaitTask  
 |> Async.Ignore  
  
 return this.NoContent() :> \_  
 else  
 return this.NotFound() :> \_  
 }

Листинг 5 – MyStore.WebApi\Program.fs

namespace MyStore.WebApi  
  
open System  
open System.Collections.Generic  
open System.IO  
open System.Linq  
open System.Threading.Tasks  
open Microsoft.AspNetCore  
open Microsoft.AspNetCore.Hosting  
open Microsoft.Extensions.Configuration  
open Microsoft.Extensions.Hosting  
open Microsoft.Extensions.Logging  
  
module Program =  
 let exitCode = 0  
  
 let CreateHostBuilder args =  
 Host.CreateDefaultBuilder(args)  
 .ConfigureWebHostDefaults(fun webBuilder ->  
 webBuilder.UseStartup<Startup>() |> ignore  
 )  
  
 [<EntryPoint>]  
 let main args =  
 CreateHostBuilder(args).Build().Run()  
  
 exitCode

Листинг 6 – MyStore.WebApi\Repository.fs

namespace MyStore.WebApi.Repository  
  
open System.Linq  
open Microsoft.EntityFrameworkCore  
open MyStore.Data  
open MyStore.Data.Entity  
  
module Customers =  
 let exists (context: Context) id =  
 query {  
 for i in context.Customers do  
 exists (i.CustomerId = id)  
 }  
  
 let exactlyOne (context: Context) id =  
 query {  
 for i in context.Customers do  
 where (i.CustomerId = id)  
 exactlyOne  
 }  
  
 let skipTake (context: Context) (nskip, ntake) =  
 query {  
 for i in context.Customers do  
 sortBy i.CustomerId  
 select i  
 skip nskip  
 take ntake  
 }  
  
  
module Carts =  
 let exists (context: Context) id =  
 query {  
 for i in context.Carts do  
 exists (i.CartId = id)  
 }  
  
 let exactlyOne (context: Context) id =  
 query {  
 for i in context.Carts do  
 where (i.CartId = id)  
 exactlyOne  
 }  
  
 let exactlyOneIncludeProducts (context: Context) id =  
 query {  
 for i in context.Carts.Include(fun j -> j.Products) do  
 where (i.CartId = id)  
 exactlyOne  
 }  
  
 let skipTake (context: Context) (nskip, ntake) =  
 query {  
 for i in context.Carts do  
 sortBy i.CartId  
 select i  
 skip nskip  
 take ntake  
 }  
  
  
module Products =  
 let exists (context: Context) id =  
 query {  
 for i in context.Products do  
 exists (i.ProductId = id)  
 }  
  
 let exactlyOne (context: Context) id =  
 query {  
 for i in context.Products do  
 where (i.ProductId = id)  
 exactlyOne  
 }  
  
 let skipTake (context: Context) (nskip, ntake) =  
 query {  
 for i in context.Products do  
 sortBy i.ProductId  
 select i  
 skip nskip  
 take ntake  
 }  
  
module Orders =  
 let exists (context: Context) id =  
 query {  
 for i in context.Orders do  
 exists (i.OrderId = id)  
 }  
  
 let exactlyOne (context: Context) id =  
 query {  
 for i in context.Orders do  
 where (i.OrderId = id)  
 exactlyOne  
 }  
  
 let exactlyOneIncludeOrderedProducts (context: Context) id =  
 query {  
 for i in context.Orders.Include(fun i -> i.OrderedProducts) do  
 where (i.OrderId = id)  
 exactlyOne  
 }  
  
 let skipTake (context: Context) (nskip, ntake) =  
 query {  
 for i in context.Orders do  
 sortBy i.OrderId  
 select i  
 skip nskip  
 take ntake  
 }  
  
  
module OrderedProducts =  
 let exists (context: Context) orderId productId =  
 query {  
 for i in context.OrderedProducts do  
 exists (i.ProductId = productId && i.OrderId = orderId)  
 }  
  
 let exactlyOne (context: Context) orderId productId =  
 query {  
 for i in context.OrderedProducts do  
 where (i.ProductId = productId && i.OrderId = orderId)  
 exactlyOne  
 }

Листинг 7 – MyStore.WebApi\Startup.fs

namespace MyStore.WebApi  
  
open Microsoft.AspNetCore.Builder  
open Microsoft.AspNetCore.Hosting  
open Microsoft.Extensions.Configuration  
open Microsoft.Extensions.DependencyInjection  
open Microsoft.Extensions.Hosting  
open Microsoft.OpenApi.Models  
open MyStore.Data  
  
type Startup(configuration: IConfiguration) =  
 member this.Configuration = configuration  
  
 // This method gets called by the runtime. Use this method to add services to the container.  
 member this.ConfigureServices(services: IServiceCollection) =  
 // Add framework services.  
 services.AddControllers() |> ignore  
  
 services.AddSwaggerGen(fun c -> c.SwaggerDoc("v1", OpenApiInfo(Title = "MyStore.WebApi", Version = "v1")))  
 |> ignore  
  
 services.AddDbContext<Context>() |> ignore  
  
 services.AddCors() |> ignore  
  
 // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  
 member this.Configure(app: IApplicationBuilder, env: IWebHostEnvironment) =  
 if (env.IsDevelopment()) then  
 app.UseDeveloperExceptionPage() |> ignore  
 app.UseSwagger() |> ignore  
  
 app.UseSwaggerUI(fun c -> c.SwaggerEndpoint("/swagger/v1/swagger.json", "MyStore.WebApi v1"))  
 |> ignore  
  
 app  
 .UseHttpsRedirection()  
 .UseRouting()  
 //.UseAuthorization()  
 .UseCors(fun builder ->  
 builder.AllowAnyOrigin() |> ignore  
 builder.AllowAnyHeader() |> ignore  
 builder.AllowAnyMethod() |> ignore)  
 .UseEndpoints(fun endpoints -> endpoints.MapControllers() |> ignore)  
 |> ignore

Листинг 8 – MyStore.WebApi\Utils.fs

module MyStore.WebApi.Utils  
  
open System  
open Microsoft.AspNetCore.Mvc  
  
let nullableLimitStartToSkipTake (start: Nullable<int>, limit: Nullable<int>) =  
 let nskip =  
 match Option.ofNullable (start) with  
 | Some (x) -> x  
 | None -> 0  
  
 let ntake =  
 match Option.ofNullable (limit) with  
 | Some (x) -> x  
 | None -> Int32.MaxValue  
  
 (nskip, ntake)  
  
module ActionResult =  
 let ofAsync (res: Async<IActionResult>) = res |> Async.StartAsTask  
  
 let ofAsyncT (res: Async<ActionResult<'T>>) = res |> Async.StartAsTask  
  
 let ofAsyncTA (n: ActionResult -> ActionResult<'T>) (res: Async<IActionResult>) =  
 async {  
 let! t = res  
 return downcast t |> n  
 }  
 |> Async.StartAsTask

Листинг 9 – MyStore.Fable\src\App.fs

module App  
  
open Fable.React  
open Feliz  
open Feliz.Router  
open Feliz.UseElmish  
open Customers  
  
type State = { CurrentUrl: string list }  
type Msg = UrlChanged of string list  
  
let init () =  
 { CurrentUrl = Router.currentUrl () }, Elmish.Cmd.none  
  
let update (UrlChanged segments) state =  
 { state with CurrentUrl = segments }, Elmish.Cmd.none  
  
let router =  
 FunctionComponent.Of(fun () ->  
 let state, dispatch = React.useElmish (init, update, [||])  
  
 React.router [ router.onUrlChanged (UrlChanged >> dispatch)  
  
 router.children [ match state.CurrentUrl with  
 | [] -> Html.h1 "Home"  
 | [ "users" ] -> Html.p "123"  
 | [ "users"; Route.Int userId ] -> Html.h1 (sprintf "User ID %d" userId)  
 | [ "customers" ] -> CustomersPage()  
 | \_ -> Html.h1 "Not found" ] ])  
  
[<ReactComponent>]  
let HelloWorld () = React.fragment [ router () ]

Листинг 10 – MyStore.Fable\src\Extensions.fs

[<AutoOpen>]  
module Extensions  
  
open System  
open Fable.Core  
open Fable.Core.JsInterop  
  
[<RequireQualifiedAccess>]  
module StaticFile =  
  
 /// Function that imports a static file by it's relative path.  
 let inline import (path: string) : string = importDefault<string> path  
  
[<RequireQualifiedAccess>]  
module Config =  
 /// Returns the value of a configured variable using its key.  
 /// Retursn empty string when the value does not exist  
 [<Emit("process.env[$0] ? process.env[$0] : ''")>]  
 let variable (key: string) : string = jsNative  
  
 /// Tries to find the value of the configured variable if it is defined or returns a given default value otherwise.  
 let variableOrDefault (key: string) (defaultValue: string) =  
 let foundValue = variable key  
 if String.IsNullOrWhiteSpace foundValue  
 then defaultValue  
 else foundValue  
  
// Stylesheet API  
// let private stylehsheet = Stylesheet.load "./fancy.css"  
// stylesheet.["fancy-class"] which returns a string  
module Stylesheet =  
  
 type IStylesheet =  
 [<Emit "$0[$1]">]  
 abstract Item : className:string -> string  
  
 /// Loads a CSS module and makes the classes within available  
 let inline load (path: string) = importDefault<IStylesheet> path

Листинг 11 – MyStore.Fable\src\Global.fs

[<AutoOpen>]  
module Global  
  
let baseUrl = "http://localhost:5000"

Листинг 12 – MyStore.Fable\src\Main.fs

module Main  
  
open Feliz  
open Browser.Dom  
open Fable.Core.JsInterop  
  
importAll "./styles/global.scss"  
  
ReactDOM.render (App.HelloWorld(), document.getElementById "feliz-app")

Листинг 13 – MyStore.Fable\src\Models.fs

module Models  
  
open System  
  
type Customer =  
 { CustomerId: int  
 FirstName: string  
 LastName: string option  
 Honorific: string option  
 Email: string  
 CurrentCartId: int option }  
  
type Cart =  
 { CartId: int  
 IsPublic: bool  
 OwnerCustomerId: int option }  
  
type Order =  
 { OrderId: int  
 CustomerId: int  
 CreateTimeOffset: DateTimeOffset }  
  
type Product =  
 { ProductId: int  
 Name: string  
 Description: string  
 Price: decimal }  
  
type OrderedProduct =  
 { ProductId: int  
 OrderId: int  
 OrderedPrice: decimal }

Листинг 14 – MyStore.Fable\src\Customers.fs

module Customers  
  
open System  
open Feliz  
open Feliz.UseListener  
open Thoth  
open Thoth.Fetch  
open Thoth.Json  
open Models  
  
let getCustomerById (id: int) =  
 promise {  
 let url = $"{baseUrl}/Customers/{id}"  
  
 return! Fetch.tryGet<\_, Customer> (url, caseStrategy = CamelCase)  
 }  
  
let putCustomerById (id: int) customer =  
 promise {  
 let url = $"{baseUrl}/Customers/{id}"  
  
 return! Fetch.tryPut<Customer, unit> (url, data = customer, caseStrategy = CamelCase)  
 }  
  
let postCustomerById customer password =  
 promise {  
 let url =  
 $"{baseUrl}/Customers?password={password}"  
  
 return! Fetch.tryPost<Customer, Customer> (url, data = customer, caseStrategy = CamelCase)  
 }  
  
let deleteCustomerById (id: int) =  
 promise {  
 let url = $"{baseUrl}/Customers/{id}"  
  
 return! Fetch.tryDelete<\_, unit> (url, caseStrategy = CamelCase)  
 }  
  
let getSkipTake start limit =  
 promise {  
 let url =  
 $"{baseUrl}/Customers?start={start}&limit={limit}"  
  
 return! Fetch.tryGet<\_, Customer list> (url, caseStrategy = CamelCase)  
 }  
  
let CustomerForm =  
 React.functionComponent<{| Customer: Customer |}> (fun props ->  
 let customer, setCustomer = React.useState (props.Customer)  
 let error, setError = React.useState<FetchError option> (None)  
 let status, setStatus = React.useState ("Не было запроса")  
 let password, setPassword = React.useState ("")  
  
 (React.useEffect (fun () ->  
 match error with  
 | Some x -> setStatus (x.ToString())  
 | None -> ()),  
 [| error |])  
 |> ignore  
  
 React.fragment [ Html.button [ prop.text "Get"  
 prop.onClick (fun \_ ->  
 (getCustomerById customer.CustomerId)  
 .``then``(fun result ->  
 match result with  
 | Ok (value) ->  
 setError (None)  
 setCustomer (value)  
 setStatus ("Выполнен get")  
 | Error (error) -> setError (Some error))  
 |> ignore) ]  
 Html.button [ prop.text "Put"  
 prop.onClick (fun \_ ->  
 (putCustomerById customer.CustomerId customer)  
 .``then``(fun result ->  
 match result with  
 | Ok (value) ->  
 setError (None)  
 setStatus ("Выполнен put")  
 | Error (error) -> setError (Some error))  
 |> ignore) ]  
 Html.button [ prop.text "Post"  
 prop.onClick (fun \_ ->  
 (postCustomerById customer password)  
 .``then``(fun result ->  
 match result with  
 | Ok (value) ->  
 setError (None)  
 setCustomer (value)  
 setStatus ("Выполнен post")  
 | Error (error) -> setError (Some error))  
 |> ignore) ]  
 Html.button [ prop.text "Delete"  
 prop.onClick (fun \_ ->  
 (deleteCustomerById customer.CustomerId)  
 .``then``(fun result ->  
 match result with  
 | Ok (value) ->  
 setError (None)  
 setStatus ("Выполнен delete")  
 | Error (error) -> setError (Some error))  
 |> ignore) ]  
 Html.label [ prop.text status ]  
 Html.br []  
 Html.label [ prop.text "Id" ]  
 Html.input [ prop.value customer.CustomerId  
 prop.onChange (fun (s: string) ->  
 setCustomer  
 ({ customer with  
 CustomerId = Int32.Parse(s) })) ]  
 Html.br []  
 Html.label [ prop.text "Имя" ]  
 Html.input [ prop.value customer.FirstName  
 prop.onChange (fun (s: string) -> setCustomer ({ customer with FirstName = s })) ]  
 Html.br []  
 Html.label [ prop.text "Фамилия" ]  
 Html.input [ prop.value  
 (match customer.LastName with  
 | Some (x) -> x  
 | None -> "")  
 prop.onChange (fun (s: string) ->  
 setCustomer ({ customer with LastName = Some s })) ]  
 Html.br []  
 Html.label [ prop.text "Обращение" ]  
 Html.input [ prop.value  
 (match customer.Honorific with  
 | Some (x) -> x  
 | None -> "")  
 prop.onChange (fun (s: string) ->  
 setCustomer ({ customer with Honorific = Some s })) ]  
 Html.br []  
 Html.label [ prop.text "E-mail" ]  
 Html.input [ prop.value customer.Email  
 prop.onChange (fun (s: string) -> setCustomer ({ customer with Email = s })) ]  
 Html.br []  
 Html.label [ prop.text "Номер корзины" ]  
 Html.input [ prop.value  
 (match customer.CurrentCartId with  
 | Some (x) -> x.ToString()  
 | None -> "")  
 prop.onChange (fun (s: string) ->  
 setCustomer  
 ({ customer with  
 CurrentCartId =  
 match s with  
 | "" -> None  
 | x -> Int32.Parse(x) |> Some })) ]  
 Html.br []  
 Html.label [ prop.text "Пароль" ]  
 Html.input [ prop.value password  
 prop.type' "password"  
 prop.onChange (fun (s: string) -> setPassword (s)) ]  
 Html.br [] ])  
  
let Customers =  
 React.functionComponent<{| Customers: Customer list |}> (fun props ->  
 React.fragment [ Html.table [ yield Html.th [ Html.p "Id" ]  
 yield Html.th [ Html.p "Имя" ]  
 yield Html.th [ Html.p "Фамилия" ]  
 yield Html.th [ Html.p "Обращение" ]  
 yield Html.th [ Html.p "E-mail" ]  
 yield Html.th [ Html.p "Номер корзины" ]  
 for c in props.Customers do  
 yield  
 Html.tr [ Html.td [ Html.p c.CustomerId ]  
 Html.td [ Html.p c.FirstName ]  
 Html.td [ Html.p  
 (match c.LastName with  
 | Some (x) -> x  
 | None -> "") ]  
 Html.td [ Html.p  
 (match c.Honorific with  
 | Some (x) -> x  
 | None -> "") ]  
 Html.td [ Html.p c.Email ]  
 Html.td [ Html.p  
 (match c.CurrentCartId with  
 | Some (x) -> x.ToString()  
 | None -> "") ] ] ] ])  
  
let CustomersPage =  
 React.functionComponent (fun () ->  
 let id, setId = React.useState (2)  
  
 let customers, setCustomers =  
 React.useState<Customer list>  
 ([ { CustomerId = 021  
 FirstName = "state1.ToString()"  
 LastName = Some "123"  
 Honorific = Some "123"  
 Email = ""  
 CurrentCartId = Some 12 } ])  
  
 let error, setError = React.useState<FetchError option> (None)  
  
 React.fragment [ Html.button [ prop.text "Показать"  
 prop.onClick (fun \_ ->  
 (getSkipTake id 10)  
 .``then``(fun result ->  
 match result with  
 | Ok (value) -> setCustomers (value)  
 | Error (error) -> setError (Some error))  
 |> ignore) ]  
 Html.input [ prop.onChange (fun (e: string) -> setId (Int32.Parse(e))) ]  
 Customers {| Customers = customers |}  
 CustomerForm  
 {| Customer =  
 match customers |> List.tryHead with  
 | Some x -> x  
 | None ->  
 { CustomerId = 021  
 FirstName = "state1.ToString()"  
 LastName = Some "123"  
 Honorific = Some "123"  
 Email = ""  
 CurrentCartId = Some 12 } |} ])