Лабораторна робота №4-6

Тема. Створення RESTful API

Мета. Ознайомитися з процесом створення клієнт-серверної архітектури. Навчитися створювати RESTful API для взаємодії між клієнтом і сервером. Розвинути вміння проєктувати та реалізовувати ендпойнти для типових CRUD-операцій.

ЗАВДАННЯ

Завдання 1 - Створити ASP.NET Core Minimal API.

Завдання 2 - Створити ASP.NET Core Web API.

Завдання 3 - Спроєктувати архітектури Controller – Service – Repository.

Завдання 4 - Додати MongoDB та реалізація CRUD через Repository/Service. Підключити MongoDB замість in-memory списків і реалізувати CRUD-операції на рівні архітектури Controller–Service–Repository.

Завдання 5 - Впровадити Dependency Injection (DI) для всіх шарів і налаштувати AutoMapper для автоматичного мапінгу моделей.

Завдання 6 – Authentication & Authorization

Хід роботи:

Завдання 1

Розроблено проект за допомогою C# ASP.Net Core.

Проект реалізує CRUD операції для власного API на тему: програма для керування підписками, надано три види ресурісів:

PeopleItems – модель, яка представляє користувача з властивостями: Id – унікальний ідентифікатор, Name – ім’я користувача та Email – електронна адреса користувача,

SubscribtionItems – модель, яка представляє об’єкт підписки з

					ЛР.ОК24.ПІ231.03.04		
Змін.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		Башук О. Ю.			Клієнт-серверна архітектура ПЗ. Створення RESTful API	*Літ.*	*Арк.* 1 / *Акрушів* 62
Перевір.		Жереб Д. В.				*Н*	
						ХПК	
Н. Контр.							
Затверд.							

властивостями: Id – унікальний ідентифікатор, OwnerId – Ідентифікатор власника підписки та Service – сервіс на якому зроблено підписку,

MessageItems – модель, яка представляє об’єкт повідомлення про підписку з властивостями: Id – унікальний ідентифікатор, Title – повідомлення, OwnerId – Ідентифікатор власника підписки та Service – сервіс на якому зроблено підписку, SubId – ідентифікатор підписки.

Лістинг:

Створено каталог Models із такими файлами:

People.cs:

```
namespace CRUDAPI.Models
{
    /// <summary>
    /// Моедль що представляє користувачів програми для управління під
писками
    /// </summary>
    public class PeopleItems
    {
        /// <summary>
        /// Унікальний ідентифікатор
        /// </summary>
        public int Id { get; set; }
        /// <summary>
        /// Ім'я або логін користувача
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Поштова адреса користувача
        /// </summary>
        public string Email { get; set; }
    }
}
```

Subscription.cs:

```
namespace CRUDAPI.Models
{
    /// <summary>
    /// Модель, яка описує підпску, яку може мати користувач
    /// </summary>
    public class SubscriptionItems
    {
        /// <summary>
```

```
        /// Унікальний індентифікатор підписки
        /// </summary>
        public int Id { get; set; }
         /// <summary>
        /// Ідентиікатор власника пдписки
        /// </summary>
        public int OwnerId { get; set; }
        /// <summary>
        /// Сервіс для кого виконана підпска
        /// </summary>
        public string Service { get; set; }
    }
}
```

Message.cs:

```
namespace CRUDAPI.Models
{
    /// <summary>
    /// Модель реалізує сповіщення, кі отримує користувач про підписку
    /// </summary>
    public class MessageItems
    {
        /// <summary>
        /// Унікальний ідентифікатор повідомлення
        /// </summary>
        public int Id { get; set; }
        /// <summary>
        /// Заголовок повідомлення
        /// </summary>
        public string Title { get; set; }
        /// <summary>
        /// Кому надсилаєься повідомлення
        /// </summary>
        public int OwnerId { get; set; }
        /// <summary>
        /// Індетифікатор підписки
        /// </summary>
        public int SubId { get; set; }
    }
}
```

З цьому каталозі також розміщено файл для валідації данних:

```
using CRUDAPI.Data;
using System.Text.RegularExpressions;
 namespace CRUDAPI.Models
{
    public static class Valid
```

```
        {
            const string compareField = @"^[\w-]+@[\w-]+\.\w+$";
            public static bool StringField(string field, int minLen,string nameField,
out string error)
            {
                if (string.IsNullOrEmpty(field))
                {
                    error = $"Текствове поле {nameField} пусте";
                    return false;
                }
                else if(field.Length < minLen){
                    error = $"Занадто коротке поле {nameField}";
                    return false;
                }
                error = string.Empty;
                return true;
            }
            public static bool Email(string email,out string error)
            {
                if (!Regex.IsMatch(email, compareField))
                {
                    error = "Невірна поштова адреса";
                    return false;
                }
                error = string.Empty;
                return true;
            }
            public static bool CheckOwner(int id)
            {
                var owner = Elements.peopleItems.FirstOrDefault(A => A.Id == id);
                if (owner is null)
                    return false;
                return true;
            }
            public static bool Check_Subscribe_Owner(int Subid, int OwnerId,out
string error)
            {
                var subscription = Elements.subsripptionItems.FirstOrDefault(A =>
A.Id == Subid);
                if (subscription is null)
                {
                    error = "Не знайдено вказаної підписки";
                    return false;
                }
                else if (subscription.OwnerId != OwnerId)
```

```
                {
                    error = "Неправильний власник підписки";
                    return false;
                }
                 error = string.Empty;
                return true;
            }
        }
    }
```

Створенно каталог Data, в якому містяться списки із елементів класів моделей:

```
using CRUDAPI.Models;


namespace CRUDAPI.Data
{
    public static class Elements
    {
public static List<PeopleItems> peopleItems = new List<PeopleItems>()
    {
        new PeopleItems() {Id = 1, Name = "Kate", Email = "Kate@Email.com" },
        new PeopleItems() {Id = 2, Name = "John", Email = "John@Email.com" },
        new PeopleItems() {Id = 3, Name = "Jane", Email = "Jane@Email.com" },
        new PeopleItems() {Id = 4, Name = "Steve", Email = "Steve@Email.com"
},
        new PeopleItems() {Id = 5, Name = "Alex", Email = "Alex@Email.com" }
    };
public static List<SubscriptionItems> subsripptionItems = new List<SubscriptionIt
ems>()
    {
        new SubscriptionItems(){Id = 1, OwnerId = 1, Service = "Netflix"},
        new SubscriptionItems(){Id = 2, OwnerId = 1, Service = "Xbox"},
        new SubscriptionItems(){Id = 3, OwnerId = 1, Service = "Amazon"},
         new SubscriptionItems(){Id = 4, OwnerId = 2, Service = "Steam"},
        new SubscriptionItems(){Id = 5, OwnerId = 2, Service = "Google"},
        new SubscriptionItems(){Id = 6, OwnerId = 2, Service = "Amazon"},
         new SubscriptionItems(){Id = 7, OwnerId = 3, Service = "YouTube"},
        new SubscriptionItems(){Id = 8, OwnerId = 3, Service = "Netflix"},
         new SubscriptionItems(){Id = 9, OwnerId = 4, Service = "YouTube"},
        new SubscriptionItems(){Id = 10, OwnerId = 4, Service = "Netflix"},
        new SubscriptionItems(){Id = 11, OwnerId = 4, Service = "Megogo"},
        new SubscriptionItems(){Id = 12, OwnerId = 4, Service = "AppleFilms"},
         new SubscriptionItems(){Id = 13, OwnerId = 5, Service = "AppleSerice"},
```

```
            new SubscriptionItems(){Id = 14, OwnerId = 5, Service = "Xbox"},
            new SubscriptionItems(){Id = 15, OwnerId = 5, Service = "AppleFilms"},
        };
public static List<MessageItems> messageItems = new List<MessageItems>()
                {
                    new MessageItems(){Id = 1,Title = "До оплати три дні", OwnerId =
 1, SubId = 1 },
                    new MessageItems(){Id = 2,Title = "До оплати два дні", OwnerId =
 1, SubId = 1 },
                    new MessageItems(){Id = 3,Title = "До оплати один день", OwnerI
d = 1, SubId = 1 },
                    new MessageItems(){Id = 4,Title = "Підписка прострочена", Own
erId = 2, SubId = 5 },
                    new MessageItems(){Id = 5,Title = "Підписку скасовано", OwnerI
d = 3, SubId = 7 },
                    new MessageItems(){Id = 6,Title = "Підписку просрочена", Owner
Id = 4, SubId = 9 },
                    new MessageItems(){Id = 7,Title = "Підпску продовжено", OwnerI
d = 4, SubId = 9 },
                    new MessageItems(){Id = 8,Title = "Підписка прострочена", Owne
rId = 4, SubId = 11 },
                    new MessageItems(){Id = 9,Title = "Підписку скасовано", OwnerId
 = 4, SubId = 11 },
                    new MessageItems(){Id = 10,Title = "До оплаи підписки тиждень",
 OwnerId = 5, SubId = 15 },
                };
            }
        }
```

Створено каталог EndPoints із класами для реалізації endpoints моделей:

PeopleEndPoints.cs:

```
using CRUDAPI.Models;
using CRUDAPI.Data;
```

```
namespace CRUDAPI.EndPoints
{
   public static class PeopleEndpoints
   {
     ///<summary>
   ///Реєструє всі маршрути до ресурсу People
   ///</summary>
   ///<param name="app">Екземпляр<see cref="WebApplication"/>.</para
m>
   public static void mapPeople(this WebApplication app)
      {
       app.MapGet("/peoples/{id:int}", (int id) =>
        {
           var peopleItem = Elements.peopleItems.FirstOrDefault(A => A.Id
== id);
            return peopleItem is null ? Results.NotFound() : Results.Ok(peopl
eItem);
         })
        .WithSummary("Отримати корисувача за Id")
        .Produces<PeopleItems>(StatusCodes.Status200OK)
        .Produces(StatusCodes.Status404NotFound)
        .WithName("GETPeople")
        .WithDescription("Отримати користувача за Id")
        .WithOpenApi();
       app.MapGet("/peoples", (string? name, string? email) =>
        {
           IEnumerable<PeopleItems> items = Elements.peopleItems;
            if (name is not null)
              items = items.Where(A => A.Name == name);
           if (email is not null)
              items = items.Where(A => A.Email == email);
            return items.Count() != 0 ? Results.Ok(items) : Results.NotFound(
);
        })
        .WithSummary("Отримати всіх корисувачів або конкретизувати з
а допомогою фільтрів")
        .Produces<PeopleItems>(StatusCodes.Status200OK)
        .Produces(StatusCodes.Status404NotFound)
        .WithName("GETPeoples")
        .WithDescription("Отримання всіх користувачів або конкреного з
а фільром")
        .WithOpenApi();
       app.MapPost("/peoples", (PeopleItems peopleItem) =>
      {
        if (!Valid.StringField(peopleItem.Name, 5, "Name", out var error1))
```

```
                return Results.BadRequest(error1);
             if(!Valid.Email(peopleItem.Email,out var error2))
                return Results.BadRequest(error2);
             peopleItem.Id = Elements.peopleItems.Max(A => A.Id) + 1;
             Elements.peopleItems.Add(peopleItem);
            return Results.Created($"/peoples/{peopleItem.Id}", peopleItem);
        })
            .WithSummary("Надіслати конкретного користувача в список")
            .Produces<PeopleItems>(StatusCodes.Status201Created)
            .Produces(StatusCodes.Status400BadRequest)
            .WithName("POSTPeople")
            .WithDescription("Відправити конкретного користувача")
            .WithOpenApi();
        app.MapPut("/peoples/{id:int}", (int id, PeopleItems updatePeople) =>
        {
            var people = Elements.peopleItems.Find(A => A.Id == id);
            if (people is null)
                return Results.NotFound();
            if (!Valid.StringField(updatePeople.Name, 5, "Name", out var error1
))
                return Results.BadRequest(error1);
            if (!Valid.Email(updatePeople.Email, out var error2))
                return Results.BadRequest(error2);
            people.Name = updatePeople.Name;
            people.Email = updatePeople.Email;
            return Results.Ok(people);
        })
            .WithSummary("Оновити повнсітю конкретного користувача")
            .Produces<PeopleItems>(StatusCodes.Status200OK)
            .Produces(StatusCodes.Status404NotFound)
            .Produces(StatusCodes.Status400BadRequest)
            .WithName("PUTPeople")
            .WithDescription("Оновити повністю конкреного кристувача")
            .WithOpenApi();
        app.MapPatch("/peoples/{id:int}", (int id, PeopleItems updatePeople) =
>      {
            var people = Elements.peopleItems.Find(A => A.Id == id);
            if (people is null)
                return Results.NotFound();
            if (updatePeople.Name is not null)
            {
              if (!Valid.StringField(updatePeople.Name, 5, "Name", out var error
1))
                  return Results.BadRequest(error1);
               people.Name = updatePeople.Name;
```

```
            }
            if (updatePeople.Email is not null)
            {
                if (!Valid.Email(updatePeople.Email, out var error2))
                    return Results.BadRequest(error2);
                people.Email = updatePeople.Email;
            }
            return Results.Ok(people);
        })
            .WithSummary("Оновити частково конкретного користувача")
            .Produces<PeopleItems>(StatusCodes.Status200OK)
            .Produces(StatusCodes.Status404NotFound)
            .Produces(StatusCodes.Status400BadRequest)
            .WithName("PATCHPeople")
            .WithDescription("Оновити частково конкреного кристувача")
            .WithOpenApi();
        app.MapDelete("/peoples/{id:int}", (int id) =>
        {
            var people = Elements.peopleItems.FirstOrDefault(A => A.Id == id);
            if (people is null)
                return Results.NotFound();
            Elements.peopleItems.Remove(people);
            return Results.NoContent();
        })
            .WithSummary("Видалити конкретного користувача")
            .Produces(StatusCodes.Status204NoContent)
            .Produces(StatusCodes.Status404NotFound)
            .WithName("DELETEPeople")
            .WithDescription("Видалити конкретного користувача")
            .WithOpenApi();
    }
    }
}
SuscriptionsEndPoints.cs:
using CRUDAPI.Models;
using CRUDAPI.Data;
 namespace CRUDAPI.EndPoints
{
    public static class SubscriptionsEndPoints
    {
        ///<summary>
        ///Реєструє всі маршрути до ресурсу Subscriptions
        ///</summary>
        ///<param name="app">Екземпляр<see cref="WebApplication"/>.</pa
ram>
```

```
        public static void mapSubscriptions(this WebApplication app)
        {
            app.MapGet("/subs/{id:int}", (int id) =>
            {
                var subItem = Elements.subsripptionItems.FirstOrDefault(A => A.
Id == id);
                return subItem is null ? Results.NotFound() : Results.Ok(subItem)
;
            })
                .WithSummary("Отримати підписку")
                .Produces<SubscriptionItems>(StatusCodes.Status200OK)
                .Produces(StatusCodes.Status404NotFound)
                .WithName("GETSub")
                .WithDescription("Отримати конкретну підписку за Id")
                .WithOpenApi();
            app.MapGet("/subs", (int? ownerId, string? service) =>
            {
                IEnumerable<SubscriptionItems> items = Elements.subsripptionIt
ems;
                if (ownerId is not null)
                    items = items.Where(A => A.OwnerId == ownerId);
                if (service is not null)
                    items = items.Where(A => A.Service == service);

                return items.Count() != 0 ? Results.Ok(items) : Results.NotFound(
);
            })
                .WithSummary("Отримати всі підписки або конкретні")
                .Produces<SubscriptionItems>(StatusCodes.Status200OK)
                .Produces(StatusCodes.Status404NotFound)
                .WithName("GETSubs")
                .WithDescription("Отримати всі підписки або конкретні за доп
омогою фільтрів")
                .WithOpenApi();
            app.MapPost("/subs", (SubscriptionItems subscription) =>
            {
                if (!Valid.StringField(subscription.Service, 3, "Service", out var er
ror))
                    return Results.BadRequest(error);
                if (!Valid.CheckOwner(subscription.OwnerId))
                    return Results.BadRequest("Немає вказаного 'ownerId'");
                subscription.Id = Elements.subsripptionItems.Max(A => A.Id) +
1;
                Elements.subsripptionItems.Add(subscription);
                return Results.Created($"/subs/{subscription.Id}", subscription);
```

```
                })
                  .WithSummary("Відправити підписку")
                  .Produces<SubscriptionItems>(StatusCodes.Status201Created)
                  .Produces(StatusCodes.Status400BadRequest)
                  .WithName("POSTSub")
                  .WithDescription("Відправити конкретну підписку")
                  .WithOpenApi();
                app.MapPut("/subs/{id:int}", (int id, SubscriptionItems updateSub)
=>
                {
                  var subscription = Elements.subsripptionItems.Find(A => A.Id ==
id);
                   if (subscription is null)
                     return Results.NotFound();
                   if (!Valid.CheckOwner(subscription.OwnerId))
                     return Results.BadRequest("Немає вказаного 'ownerId'");
                   if (!Valid.StringField(updateSub.Service, 3, "Service", out var err
or))
                     return Results.BadRequest(error);
                   subscription.OwnerId = updateSub.OwnerId;
                  subscription.Service = updateSub.Service;
                   return Results.Ok(subscription);
                })
                  .WithSummary("Оновити повністю підписку")
                  .Produces<SubscriptionItems>(StatusCodes.Status200OK)
                  .Produces(StatusCodes.Status404NotFound)
                  .Produces(StatusCodes.Status400BadRequest)
                  .WithName("PUTSub")
                  .WithDescription("Оновити повністю конкретну підписку за Id
")
                  .WithOpenApi();

                app.MapPatch("/subs/{id:int}", (int id, SubscriptionItems updateSub)
 =>
                {
                  var subscription = Elements.subsripptionItems.Find(A => A.Id ==
id);
                   if (subscription is null)
                     return Results.NotFound();
                   if (!Valid.StringField(updateSub.Service, 3, "Service", out var err
or))
                     return Results.BadRequest(error);
                   if (updateSub.OwnerId != 0)
                   {
                     if (!Valid.CheckOwner(subscription.OwnerId))
```

```
                    return Results.BadRequest("Немає вказаного 'ownerId'");
                subscription.OwnerId = updateSub.OwnerId;
            }
            if(updateSub.Service is not null)
                subscription.Service = updateSub.Service;
            return Results.Ok(subscription);
        })
            .WithSummary("Оновити частково підписку")
            .Produces<SubscriptionItems>(StatusCodes.Status200OK)
            .Produces(StatusCodes.Status404NotFound)
            .Produces(StatusCodes.Status400BadRequest)
            .WithName("PATCHSub")
            .WithDescription("Оновити частково конкретну підписку за Id"
)
            .WithOpenApi();
        app.MapDelete("/subs/{id:int}", (int id) =>
        {
            var sub = Elements.subsripptionItems.FirstOrDefault(A => A.Id =
= id);
        if (sub is null)
            return Results.NotFound();
            Elements.subsripptionItems.Remove(sub);
            return Results.NoContent();
        })
            .WithSummary("Видалити підписку")
            .Produces(StatusCodes.Status204NoContent)
            .Produces(StatusCodes.Status404NotFound)
            .WithName("DELETESub")
            .WithDescription("Видалити конкретну підписку за Id")
            .WithOpenApi();
    }
  }
}
MessageEndPoints.cs:
using CRUDAPI.Data;
using CRUDAPI.Models;
 namespace CRUDAPI.EndPoints
{
   public static class MessageEndPoints
   {
       ///<summary>
      ///Реєструє всі маршрути до ресурсу Subscriptions
      ///</summary>
      ///<param name="app">Екземпляр<see cref="WebApplication"/>.</pa
ram>
```

```
            public static void mapMessages(this WebApplication app)
            {
               app.MapGet("/messages/{id:int}", (int id) =>
               {
                  var mItem = Elements.messageItems.FirstOrDefault(A => A.Id ==
id);
                   return mItem is null ? Results.NotFound() : Results.Ok(mItem);
                })
                  .WithSummary("Отримати повідомлення")
                  .Produces<MessageItems>(StatusCodes.Status200OK)
                  .Produces(StatusCodes.Status404NotFound)
                  .WithName("GETmessage")
                  .WithDescription("Отримати конкретне повідомлення за Id")
                  .WithOpenApi();
                app.MapGet("/messages", (int? ownerId, int? subId,string? title) =>
                {
                  IEnumerable<MessageItems> items = Elements.messageItems;
                   if (ownerId is not null)
                      items = items.Where(A => A.OwnerId == ownerId);
                  if (subId is not null)
                      items = items.Where(A => A.SubId == subId);
                  if(title is not null)
                      items = items.Where(A => A.Title == title);
                   return items.Count() != 0 ? Results.Ok(items) : Results.NotFound(
);
                })
                   .WithSummary("Отримати всі повідомлення або конкретні")
                  .Produces<MessageItems>(StatusCodes.Status200OK)
                  .Produces(StatusCodes.Status404NotFound)
                  .WithName("GETmessages")
                  .WithDescription("Отримати всі повідомлення або використати
 фільри для конкретизації")
                  .WithOpenApi();
                app.MapPost("/messages", (MessageItems message) =>
                {
                  if (!Valid.StringField(message.Title, 10, "Title", out var error))
                      return Results.BadRequest(error);
                              if(!Valid.Check_Subscribe_Owner(message.SubId,mess
age.OwnerId,out string error1))
                    return Results.BadRequest(error1);
                   message.Id = Elements.messageItems.Max(A => A.Id) + 1;
                   Elements.messageItems.Add(message);
                   return Results.Created($"/messages/{message.Id}", message);
                })
                  .WithSummary("Зафіксувати повідомлення в колекції")
```

```
                .Produces<MessageItems>(StatusCodes.Status201Created)
                .Produces(StatusCodes.Status404NotFound)
                .WithName("POSTmessage")
                .WithDescription("Відправити повідомлення в колекцію")
                .WithOpenApi();
            app.MapPut("/messages/{id:int}", (int id, MessageItems updateM) =
>
            {
                var m = Elements.messageItems.Find(A => A.Id == id);
                if (m is null)
                    return Results.NotFound();
                if (!Valid.Check_Subscribe_Owner(updateM.SubId, updateM.Ow
nerId, out string error1))
                    return Results.BadRequest(error1);
                if (!Valid.StringField(updateM.Title, 10, "Title", out var error))
                    return Results.BadRequest(error);
                m.Title = updateM.Title;
                m.OwnerId = updateM.OwnerId;
                m.SubId = updateM.SubId;
                return Results.Ok(m);
            })
                .WithSummary("Оновити повністю повідомлення")
                .Produces<MessageItems>(StatusCodes.Status200OK)
                .Produces(StatusCodes.Status404NotFound)
                .Produces(StatusCodes.Status400BadRequest)
                .WithName("PUTmessage")
                .WithDescription("Оновити повністю повідомлення в колекції з
a Id")
                .WithOpenApi();
            app.MapPatch("/messages/{id:int}", (int id, MessageItems updateM)
 =>
            {
                var m = Elements.messageItems.Find(A => A.Id == id);
                if (m is null)
                    return Results.NotFound();
                int OId = updateM.OwnerId != 0? updateM.OwnerId : m.OwnerId
;
                int SId = updateM.SubId != 0? updateM.SubId : m.SubId;
                if (!Valid.Check_Subscribe_Owner(SId, OId, out string error1))
                    return Results.BadRequest(error1);
                if (updateM.OwnerId != 0)
                    m.OwnerId = updateM.OwnerId;
                if(updateM.SubId != 0)
                    m.SubId = updateM.SubId;
                if (updateM.Title is not null)
```

```
                {
                    if (!Valid.StringField(updateM.Title, 10, "Title", out var error))
                        return Results.BadRequest(error);
                    m.Title = updateM.Title;
                }
                return Results.Ok(m);
            })
                .WithSummary("Оновити частково повідомлення")
                .Produces<MessageItems>(StatusCodes.Status200OK)
                .Produces(StatusCodes.Status404NotFound)
                .Produces(StatusCodes.Status400BadRequest)
                .WithName("PATCHmessage")
                .WithDescription("Оновити частково повідомлення в колекції з
a Id")
                .WithOpenApi();
            app.MapDelete("/messages/{id:int}", (int id) =>
            {
                var m = Elements.messageItems.FirstOrDefault(A => A.Id == id);
                if (m is null)
                    return Results.NotFound();
                Elements.messageItems.Remove(m);
                return Results.NoContent();
            })
                .WithSummary("Видалити повідомлення")
                .Produces(StatusCodes.Status204NoContent)
                .Produces(StatusCodes.Status404NotFound)
                .WithName("DELETEmessage")
                .WithDescription("Видалити повідомлення з колекції за Id")
                .WithOpenApi();
        }
    }
}
```

Підключенно в program.cs всі endpoints:

```
using CRUDAPI.EndPoints;
 var builder = WebApplication.CreateBuilder(args);
builder.Services.AddEndpointsApiExplorer();
 builder.Services.AddSwaggerGen(oprions =>
{
    var xmlFile = $"{System.Reflection.Assembly.GetExecutingAssembly().
GetName().Name}.xml";
    var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
    oprions.IncludeXmlComments(xmlPath);
});
var app = builder.Build();
app.mapPeople();
```

```
app.mapSubscriptions();
app.mapMessages();
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
app.Run();
```

Протестовано роботу програму за допомогою swagger:

Запит1

Рисунок 1 – Отримання користувача за Id

Запит 2

Рисунок 2 – Отримання користувача за відсутнім Id

Запит 3

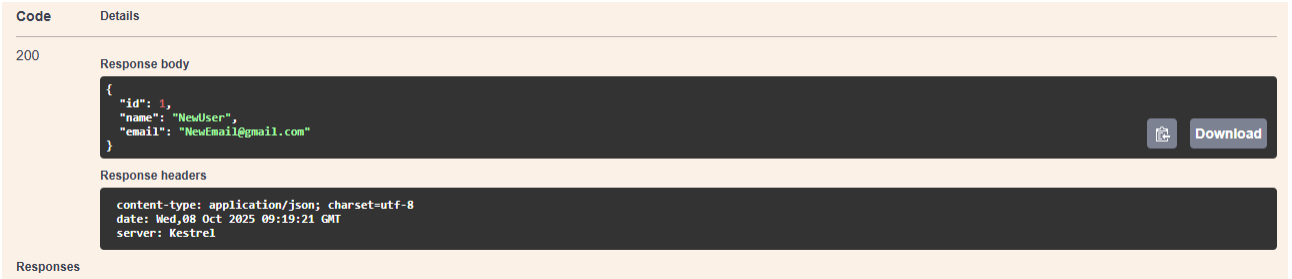


Рисунок 3 – Коректне оновлення користувача

Запит 4

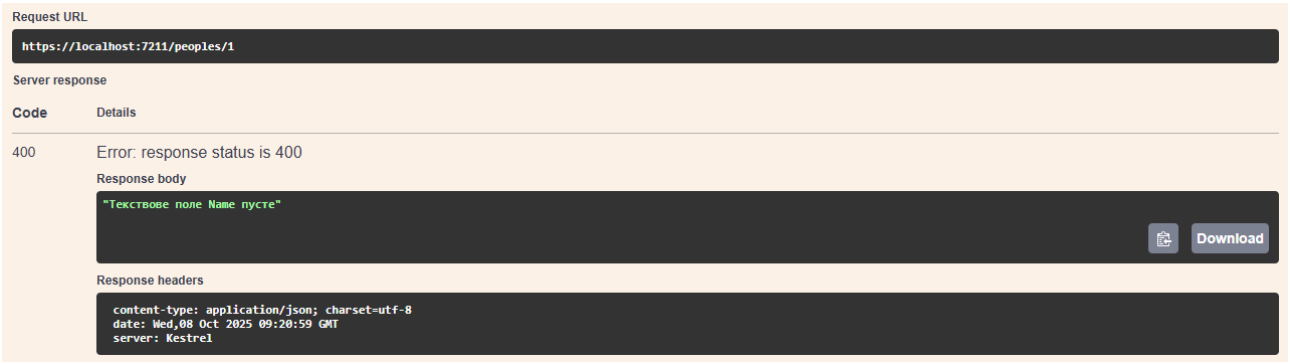


Рисунок 4 – Помилка під час оновлення

Запит5

Рисунок 5 – Помилка валідації – невірна електронна адреса

Запит 6

Рисунок 6 – Часткове оновлення користувача

Запит 7

Рисунок 7 – Видалення користувача

Запит 8

Request URL

```
https://localhost:7211/peoples
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 2,
    "name": "John",
    "email": "John@Email.com"
  },
  {
    "id": 3,
    "name": "Jane",
    "email": "NewEmail@gmail.com"
  },
  {
    "id": 4,
    "name": "Steve",
    "email": "Steve@Email.com"
  },
  {
    "id": 5,
    "name": "Alex",
    "email": "Alex@Email.com"
  }
]
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Wed,08 Oct 2025 09:26:36 GMT
server: Kestrel
```

Рисунок 8 – Отримано список всіх користувачів

Запит 9

Request URL

```
https://localhost:7211/peoples
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 2,
    "name": "John",
    "email": "John@Email.com"
  },
  {
    "id": 3,
    "name": "Jane",
    "email": "NewEmail@gmail.com"
  },
  {
    "id": 4,
    "name": "Steve",
    "email": "Steve@Email.com"
  },
  {
    "id": 5,
    "name": "Alex",
    "email": "Alex@Email.com"
  }
]
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Wed,08 Oct 2025 09:26:36 GMT
server: Kestrel
```

Рисунок 9 – Отримано всіх користувачів

Запит 10

Request URL

```
https://localhost:7211/peoples?name=John
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 2,
    "name": "John",
    "email": "John@Email.com"
  }
]
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Wed,08 Oct 2025 09:27:53 GMT
server: Kestrel
```

Рисунок 10 – Отримано конкретного користувача

Запит 11

Рисунок 11 – Користувача не знайдено

Запит 12

Рисунок 12 – Створення нового користувача

Запит 13

Рисунок 13 – Спроба додавання користувача з невалідною адресою

Запит 14

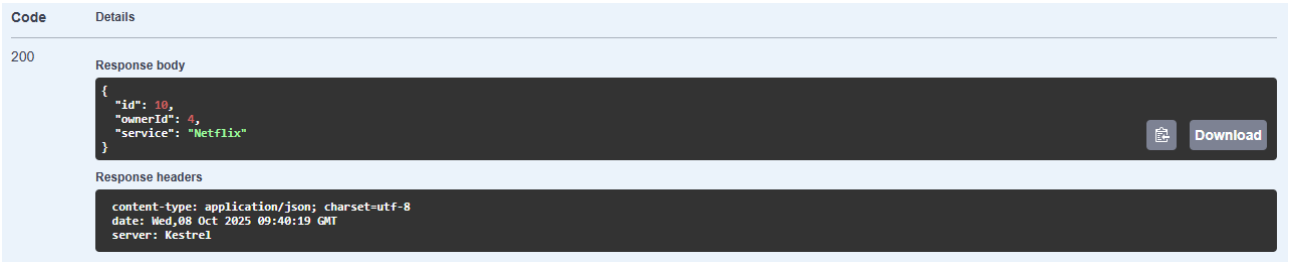


Рисунок 14 – Отримання конкретної підписки

Запит 15

Рисунок 15 – спроба отримати неіснуючу підписку

Запит 16

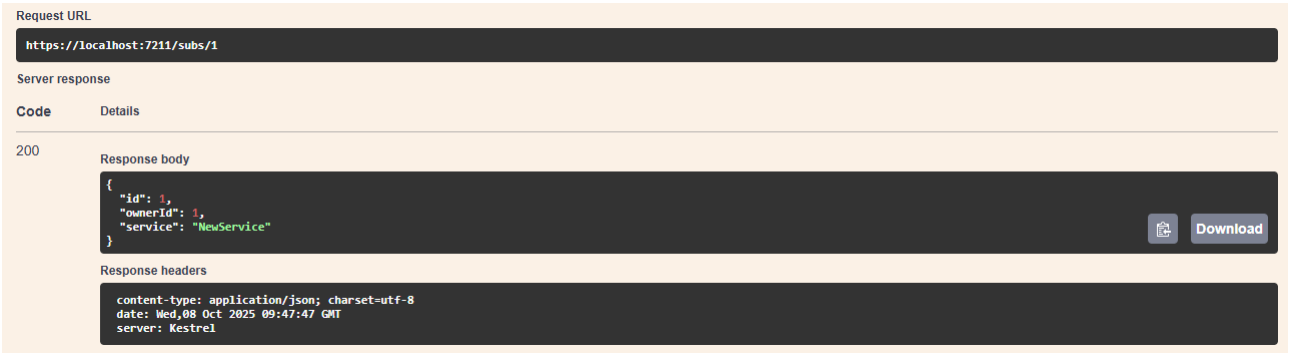


Рисунок 16 - Оновлення підписки

Запит 17

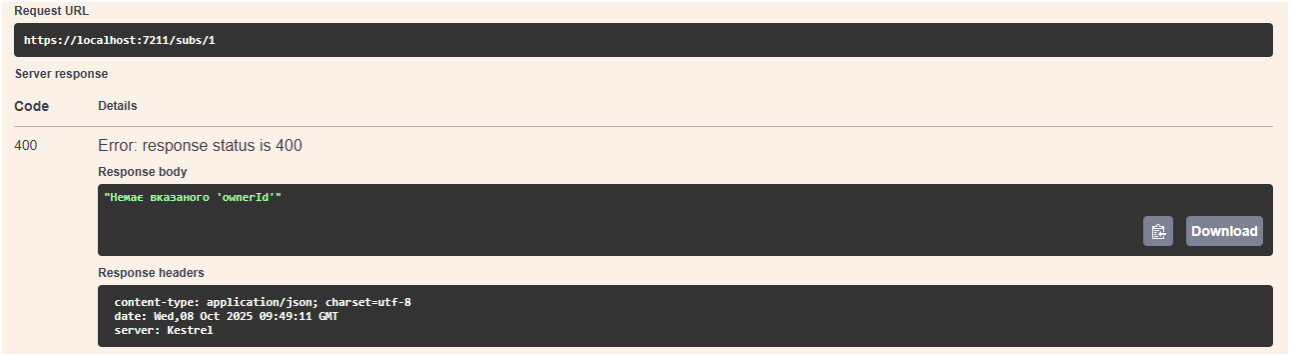


Рисунок 17 – Оновлення підписки з Id користувача, якого не існує

Запит 18

Рисунок 18 – Часткове оновлення користувача

Запит 19

Рисунок 19 – Спроба оновлення поля підписки на пусте

Запит 20

Рисунок 20 – Видалення підписки

Запит 21

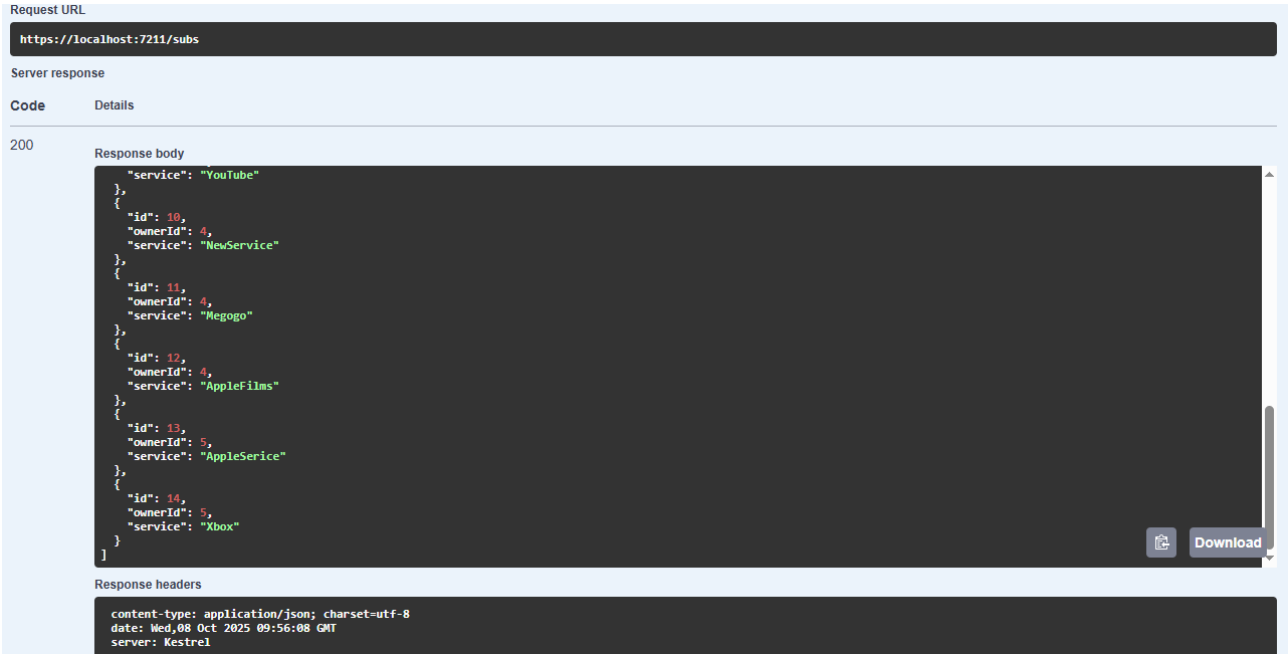


Рисунок 21 – Отримання списку всіх підписок

Запит 22

Рисунок 22 – Отримання підписки за вказаними полями

Запит 23

Рисунок 23 – Створення нової підписки

Запит 24

Рисунок 24 – Спроба додавання нової підписки з неіснуючм користувачем

Запит 25

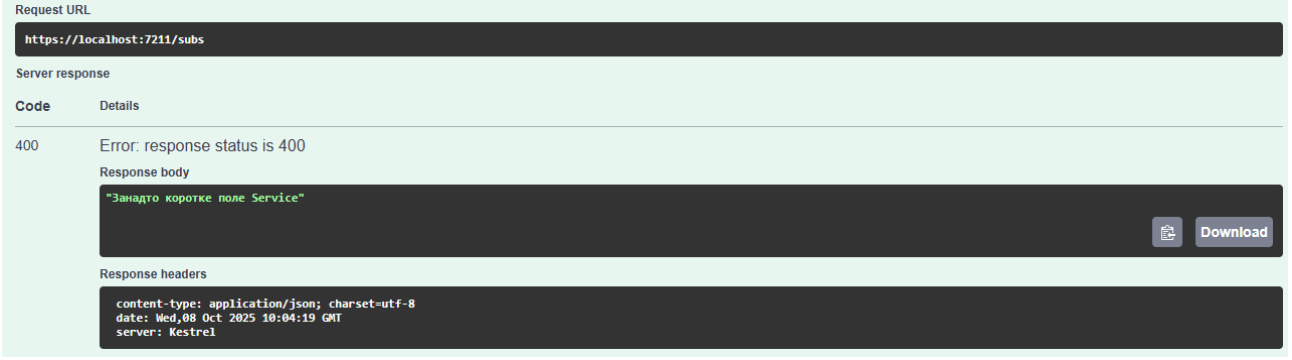


Рисунок 25 – Спроба створення підписки з невалідним сервісом

Запит 26

Рисунок 26 – Отримання конкретного повідомлення

Запит 27

Рисунок 27 – Спроба отримання неіснуючої підписки

Запит 28

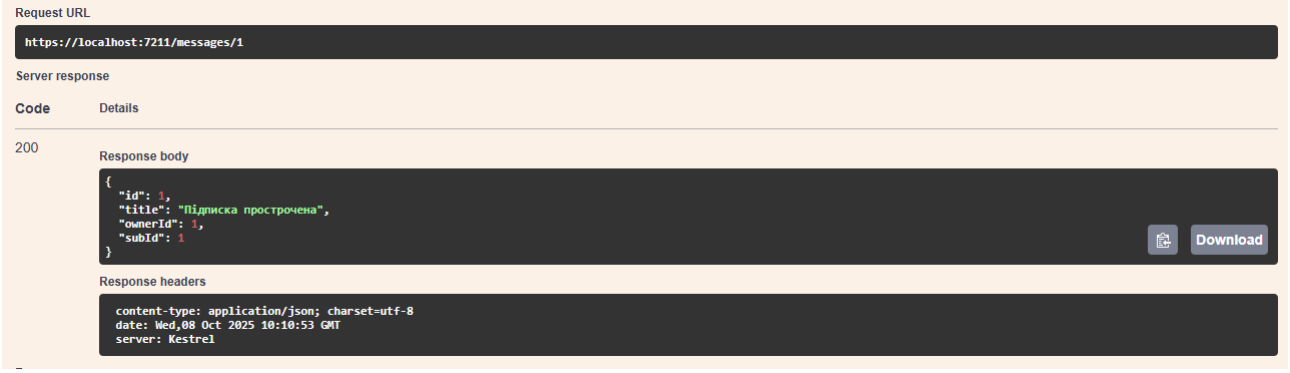


Рисунок 28 – Оновлення вмісту повідомлення

Запит 29

Рисунок 29 – Спроба відправит підписнику нагадування не про те повідолення

Запит 30

Рисунок 30 – Оновлення підписки в користувача

Запит 31

Рисунок 31 – Спроба оновлення невірної підписки користувача

Запит 32

Рисунок 32 – Видалення повідомленн про підписку

Запит 33

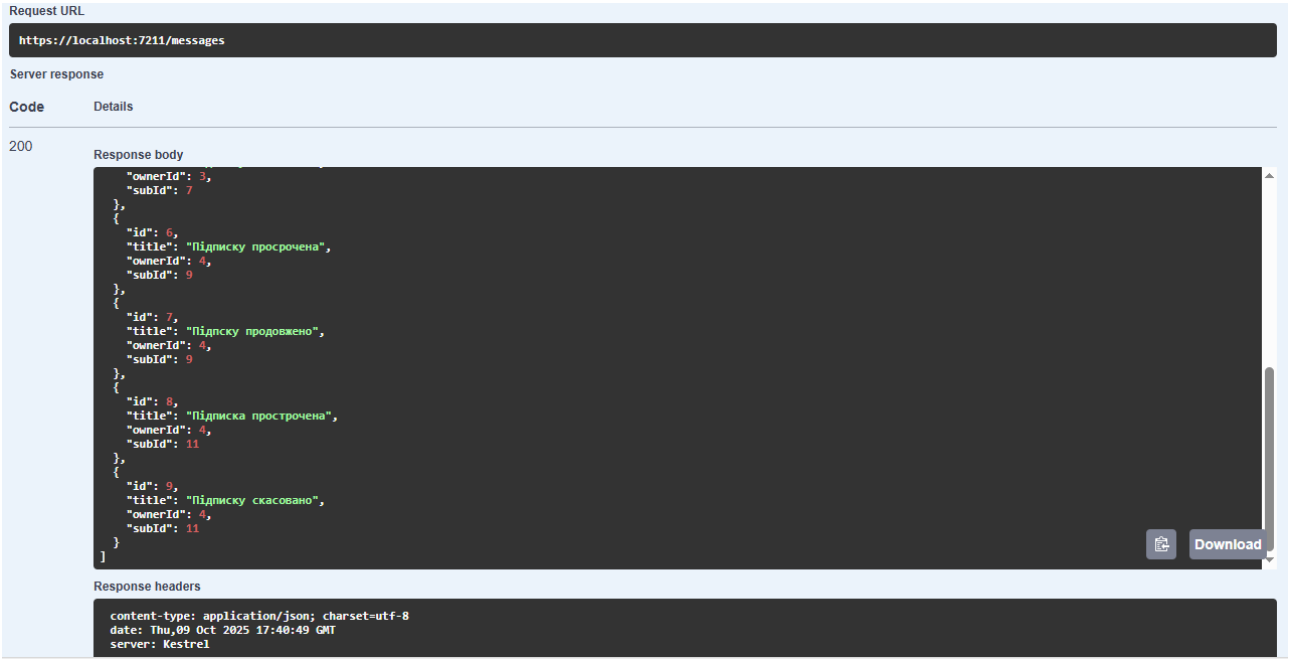


Рисунок 33 -Отримання списку усіх повідомлень

Запит 34

Рисунок 34 – Отримання всіх повідомлень до користувача

Запит 35

Рисунок 35 – Отримання всіх повідомлень про конретну підписку

Запит 36

Рисунок 36 – Спроба відправити пусте повідомлення

Запит 37

Рисунок 37 – Додано повідомлення для користувача та його підписки

Запит 38

Рисунок 38 – Спроба відправити повідомленн користувачу не про його підписку

Завдання 2

Розроблено проект за допомогою C# ASP.Net Core WebAPI.

Проект реалізує CRUD операції для власного API на тему: програма для керування підписками, надано три види ресурісів:

PeopleItems – модель, яка представляє користувача з властивостями: Id – унікальний ідентифікатор, Name – ім’я користувача та Email – електронна адреса користувача,

SubscribtionItems – модель, яка представляє об’єкт підписки з властивостями: Id – унікальний ідентифікатор, OwnerId – Ідентифікатор власника підписки та Service – сервіс на якому зроблено підписку, Status – перелік, для відображення станів підписки,

MessageItems – модель, яка представляє об’єкт повідомлення про підписку з властивостями: Id – унікальний ідентифікатор, Title – повідомлення, OwnerId – Ідентифікатор власника підписки та Service – сервіс на якому зроблено підписку, SubId – ідентифікатор підписки.

Реалізовано валідацію в класові PeopleItems за допомогою Data.Anotation, для класів SubscriptionItems, MessageItems за допомогою FluentValidator та винесено логіку в океремий каталог Validator.

Лістинг:

People.cs:

```
using System.ComponentModel.DataAnnotations;
 namespace LW_4_2.Models
{
   /// <summary>
   /// Моедль що представляє користувачів програми для управління під писками
   /// </summary>
   public class PeopleItems
   {
      /// <summary>
      /// Унікальний ідентифікатор
      /// </summary>
      public int Id { get; set; }
      /// <summary>
```

```
        /// Ім'я або логін користувача
        /// </summary>
        [Required(ErrorMessage = "Поле 'Name' Обов'язкове для заповнення
")]
        [RegularExpression(@"^[A-Z][a-z]{5,10}[0-
9]{0,5}$", ErrorMessage = "'Name' не відповідає вимогам:" +
          "Перші велика англійська літера, далі від 5 до 10 малих англійсь
ких символів та цифри за бажанням")]
        public string? Name { get; set; }
        /// <summary>
        /// Поштова адреса користувача
        /// </summary>
        [Required(ErrorMessage = "Поле 'Email' є обов'язковим")]
        [EmailAddress(ErrorMessage = "Не коректне значення 'Email'")]
        public string? Email { get; set; }
    }
}
```

Subscription.cs:

```
namespace LW_4_2.Models
{
    /// <summary>
    /// Перелік статусів підписки
    /// </summary>
    public enum SubStatus
    {
        /// <summary>
        /// Статус очікування - Expectation - 0
        /// </summary>
        Expectation,
        /// <summary>
```

```
    /// Статус активна - Active - 1
    /// </summary>
    Active,
    /// <summary>
    /// Статус прострочена - Overdue - 2
    /// </summary>
    Overdue,
    /// <summary>
    /// Статус Архівована - Archived - 3
    /// </summary>
    Archived
}
/// <summary>
/// Модель, яка описує підпску, яку може мати користувач
/// </summary>
public class SubscriptionItems
{
    /// <summary>
    /// Унікальний індентифікатор підписки
    /// </summary>
    public int? Id { get; set; }
    /// <summary>
    /// Ідентиікатор власника пдписки
    /// </summary>
    public int? OwnerId { get; set; }
    /// <summary>
    /// Сервіс для кого виконана підпска
    /// </summary>
    public string? Service { get; set; }
    /// <summary>
```

```
        /// Статус в якому перебуваєпідписка користувача
        /// </summary>
        public SubStatus? Status { get; set; }
    }
}
```

Message.cs:

```
namespace LW_4_2.Models
{
    /// <summary>
    /// Модель реалізує сповіщення, кі отримує користувач про підписку
    /// </summary>
    public class MessageItems
    {
        /// <summary>
        /// Унікальний ідентифікатор повідомлення
        /// </summary>
        public int Id { get; set; }
        /// <summary>
        /// Заголовок повідомлення
        /// </summary>
        public string? Title { get; set; }
        /// <summary>
        /// Кому надсилаєься повідомлення
        /// </summary>
        public int? OwnerId { get; set; }
        /// <summary>
        /// Індетифікатор підписки
        /// </summary>
        public int? SubId { get; set; }
    }
```

```
    }
    PeopleData.cs:
    using LW_4_2.Models;
    namespace LW_4_2.Data
    {
        public static class PeopleData
        {
            public static List<PeopleItems> peopleItems = new List<PeopleItems>(
)
            {
                    new PeopleItems() {Id = 1, Name = "Kate", Email = "Kate@Em
ail.com" },
                    new PeopleItems() {Id = 2, Name = "John", Email = "John@Em
ail.com" },
                    new PeopleItems() {Id = 3, Name = "Jane", Email = "Jane@Em
ail.com" },
                    new PeopleItems() {Id = 4, Name = "Steve", Email = "Steve@E
mail.com" },
                    new PeopleItems() {Id = 5, Name = "Alex", Email = "Alex@E
mail.com" }
            };
        }
    }
    SubData.cs:
    using LW_4_2.Models;
    namespace LW_4_2.Data
    {
        public static class SubData
        {
```

```
        public static List<SubscriptionItems> subsripptionItems = new List<Su
bscriptionItems>()
                {
                    new SubscriptionItems(){Id = 1, OwnerId = 1, Service = "Netfli
x", Status = SubStatus.Active},
                    new SubscriptionItems(){Id = 2, OwnerId = 1, Service = "Xbox
",Status = SubStatus.Active},
                    new SubscriptionItems(){Id = 3, OwnerId = 1, Service = "Amaz
on",Status = SubStatus.Active},
                    new SubscriptionItems(){Id = 4, OwnerId = 2, Service = "Steam
",Status = SubStatus.Active },
                    new SubscriptionItems(){Id = 5, OwnerId = 2, Service = "Googl
e", Status = SubStatus.Overdue},
                    new SubscriptionItems(){Id = 6, OwnerId = 2, Service = "Amaz
on", Status = SubStatus.Active},
                    new SubscriptionItems(){Id = 7, OwnerId = 3, Service = "YouT
ube",Status = SubStatus.Archived},
                    new SubscriptionItems(){Id = 8, OwnerId = 3, Service = "Netfli
x", Status = SubStatus.Active},
                    new SubscriptionItems(){Id = 9, OwnerId = 4, Service = "YouT
ube", Status = SubStatus.Active},
                    new SubscriptionItems(){Id = 10, OwnerId = 4, Service = "Netf
lix", Status = SubStatus.Active},
                    new SubscriptionItems(){Id = 11, OwnerId = 4, Service = "Meg
ogo", Status = SubStatus.Archived},
                    new SubscriptionItems(){Id = 12, OwnerId = 4, Service = "Appl
eFilms", Status = SubStatus.Active},


                    new SubscriptionItems(){Id = 13, OwnerId = 5, Service = "Appl
eSerice", Status = SubStatus.Active},
```

```
                    new SubscriptionItems(){Id = 14, OwnerId = 5, Service = "Xbo
x", Status = SubStatus.Active},
                    new SubscriptionItems(){Id = 15, OwnerId = 5, Service = "Appl
eFilms", Status = SubStatus.Active},
                };
        }
    }
    MessageData.cs:
    using LW_4_2.Models;
     namespace LW_4_2.Data
    {
        public static class MessageData
        {
            public static List<MessageItems> messageItems = new List<MessageIt
ems>()
            {
                new MessageItems(){Id = 1,Title = "До оплати три дні", OwnerId =
 1, SubId = 1 },
                new MessageItems(){Id = 2,Title = "До оплати два дні", OwnerId =
 1, SubId = 1 },
                new MessageItems(){Id = 3,Title = "До оплати один день", OwnerI
d = 1, SubId = 1 },
                 new MessageItems(){Id = 4,Title = "Підписка прострочена", Own
erId = 2, SubId = 5 },
                 new MessageItems(){Id = 5,Title = "Підписку скасовано", OwnerI
d = 3, SubId = 7 },
                new MessageItems(){Id = 6,Title = "Підписку просрочена", Owner
Id = 4, SubId = 9 },
                new MessageItems(){Id = 7,Title = "Підпску продовжено", OwnerI
d = 4, SubId = 9 },
```

```
                new MessageItems(){Id = 8,Title = "Підписка прострочена", Owne
rId = 4, SubId = 11 },
                new MessageItems(){Id = 9,Title = "Підписку скасовано", OwnerId
 = 4, SubId = 11 },

                new MessageItems(){Id = 10,Title = "До оплаи підписки тиждень",
 OwnerId = 5, SubId = 15 },
            };
        }
    }
    MessageValidator.cs:
    using FluentValidation;
    using LW_4_2.Models;
    using LW_4_2.Data;
    namespace LW_4_2.Validator
    {
        public class MessageValidator: AbstractValidator<MessageItems>
        {
            public MessageValidator()
            {
                RuleFor(x => x.Title)
                    .NotEmpty().WithMessage("Поле 'Title' не може бути порожнім
")
                    .Length(5, 100).WithMessage("Поле 'Title' має бути від трьох до
 ста символів");
                RuleFor(x => x.SubId)
                    .NotNull().WithMessage("Поле 'SubId' не може бути порожнім"
)
                    .Must(SubId => SubData.subsripptionItems.Any(s => s.Id == SubI
d))
```

```
                .WithMessage("Не знайдено 'SubId' із таким індексом");
            RuleFor(x => x.OwnerId)
                .NotEmpty().WithMessage("Поле 'OwnerId' не можу бу порожні
м")
                .Must(ownerId => PeopleData.peopleItems.Any(u => u.Id == own
erId))
                .WithMessage("Немає людина із таким значенням 'Id'")
                .Must((mes, ownerId) =>
                {
                    var sub = SubData.subsripptionItems.FirstOrDefault(s => s.Id =
= mes.SubId);
                    if (sub is null)
                        return false;
                    return sub.OwnerId == ownerId;
                }).WithMessage("Відсутня підписка за вказаним 'SubId' або для
 неї не існує користувача");
            }
        }
    }
    SubscriptionValidtor.cs:
    using LW_4_2.Models;
    using LW_4_2.Data;
    using FluentValidation;
     namespace LW_4_2.Validator
    {
        public class SubscriptionValidator: AbstractValidator<SubscriptionItems
>
        {
            public SubscriptionValidator()
            {
```

```
                RuleFor(x => x.OwnerId)
                    .NotEmpty().WithMessage("Поле 'OwnerId не може бути порож
нім'")
                    .Must(ownerID => SubData.subsripptionItems.Any(s => s.OwnerI
d == ownerID))
                    .WithMessage("Не знайдено користувача за вказаним 'OwnerId'
");


                RuleFor(x => x.Service)
                    .NotEmpty().WithMessage("Поле 'Service' не може бути порожн
ім")
                    .Length(3, 50).WithMessage("Поле 'Service' має бути в межах ві
д 3 до 50 символів");


                RuleFor(x => x.Status)
                    .NotEmpty().WithMessage("Поле 'Status' не можу бути порожні
м")
                    .IsInEnum().WithMessage("Поле 'Status' має бути вибране в зад
аних межах");
                }
            }
        }
        PeopleController.cs:
            using LW_4_2.Data;
            using LW_4_2.Models;
            using Microsoft.AspNetCore.Mvc;
        using System.Text.Json;
        namespace LW_4_2.Controllers
            {
                [ApiController]
```

```
[Route("[controller]")]
public class PeopleController : ControllerBase
{
    [HttpGet]
    public ActionResult<IEnumerable<PeopleItems>> GetAll(
        [FromQuery] string? Name, [FromQuery] string? Email)
    {
        IEnumerable<PeopleItems> peoples = PeopleData.peopleItems;
        if(Name is not null)
            peoples = peoples.Where(x => x.Name == Name);
        if(Email is not null)
            peoples = peoples.Where(x => x.Email == Email);
        if (!peoples.Any())
            return NotFound("Не знайдено елементів");
        return Ok(peoples);
    }
    [HttpGet("{id}")]
    public ActionResult<PeopleItems> GetById(int id)
    {
        var people = PeopleData.peopleItems.FirstOrDefault(x => x.Id ==
id);
        if (people is null)
            return NotFound($"Не знайдено людину із вказаним 'Id': {id}
");
        return Ok(people);
    }
    [HttpPost]
    public ActionResult<PeopleItems> Create(PeopleItems item)
    {
        item.Id = PeopleData.peopleItems.Max(x => x.Id) + 1;
```

```
            PeopleData.peopleItems.Add(item);
            return CreatedAtAction(nameof(GetById), new { Id = item.Id }, it
em);
        }
        [HttpPut("{id}")]
        public IActionResult Update(int id, PeopleItems upPeople)
        {
            var people = PeopleData.peopleItems.FirstOrDefault(p => p.Id ==
id);
            if (people is null)
                return NotFound($"Не знайдено людини із Id {id}");
            people.Name = upPeople.Name;
            people.Email = upPeople.Email;
            return NoContent();
        }
        [HttpPatch("{id}")]
        public IActionResult UpdatePart(int id, [FromBody] JsonElement up
People)
        {
            ModelState.ClearValidationState(nameof(upPeople));
            var people = PeopleData.peopleItems.FirstOrDefault(x => x.Id ==
id);
            if(people is null)
                return NotFound($"Не знайдено людини за вказаним Id {id}"
);
             var temp = new PeopleItems
            {
                Id = id,
                Email = people.Email,
                Name = people.Name
```

```
            }
                if(upPeople.TryGetProperty("name",out var name))
                temp.Name = name.GetString();
               if(upPeople.TryGetProperty("email",out var email))
                 temp.Email = email.GetString();
               if (!TryValidateModel(temp))
               return BadRequest(ModelState);
               people.Name = temp.Name;
               people.Email = temp.Email;
              return NoContent();
            }
            [HttpDelete("{id}")]
            public IActionResult DeletePart(int id)
            {
               var people = PeopleData.peopleItems.FirstOrDefault(p => p.Id ==
id);
               if (people is null)
                  return NotFound($"Не знайдено людину із вказаним Id {id}")
;
               PeopleData.peopleItems.Remove(people);
               return NoContent();
            }
         }
      }
```

SubController.cs:

```
using LW_4_2.Data;
using LW_4_2.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ApplicationParts;
using System.Text.Json;
```

```
namespace LW_4_2.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class SubController : ControllerBase
    {
        [HttpGet]
        public ActionResult<IEnumerable<SubscriptionItems>> GetAll(
            [FromQuery] int? ownerId, [FromQuery] string? service, [FromQuer
y] SubStatus? subStatus)
        {
            IEnumerable<SubscriptionItems> subItems = SubData.subsripptionIt
ems;
            if (ownerId is not null)
                subItems = subItems.Where(x => x.OwnerId == ownerId);
            if (service is not null)
                subItems = subItems.Where(x => x.Service == service);
            if (subStatus is not null)
                subItems = subItems.Where(x => x.Status == subStatus);
            if (subItems.Count() == 0)
                return NotFound("Не знайдено елементів за запитом");
            return Ok(subItems);
        }
        [HttpGet("{id}")]
        public ActionResult<SubscriptionItems> GetById(int id)
        {
            var sub = SubData.subsripptionItems.FirstOrDefault(x => x.Id == id)
;
            if (sub is null)
                return NotFound($"Не знайдено підписки з Id {id}");
```

```
            return Ok(sub);
        }
        [HttpPost]
        public ActionResult<SubscriptionItems> Create(SubscriptionItems ite
m)
        {
            item.Id = SubData.subsripptionItems.Max(x => x.Id) + 1;
            SubData.subsripptionItems.Add(item);
            return CreatedAtAction(nameof(GetById), new { Id = item.Id }, item
);
        }
        [HttpPut("{id}")]
        public IActionResult Update(int id, SubscriptionItems item)
        {
            var sub = SubData.subsripptionItems.FirstOrDefault(x => x.Id == id)
;
            if (sub is null)
                return NotFound($"Не знайдено підписки з Id {id}");
            sub.OwnerId = item.OwnerId;
            sub.Service = item.Service;
            sub.Status = item.Status;
            return NoContent();
        }
        [HttpPatch("{id}")]
        public IActionResult UpdatePart(int id, JsonElement item)
        {
            var sub = SubData.subsripptionItems.FirstOrDefault(x => x.Id == id)
;
            if (sub is null)
                return NotFound($"Не знайдено підписки з Id {id}");
```

```
            var temp = new SubscriptionItems
            {
                Id = sub.Id,
                OwnerId = sub.OwnerId,
                Service = sub.Service,
                Status = sub.Status
            };
            if(item.TryGetProperty("ownerId",out var upId))
                temp.OwnerId = upId.GetInt32();
            if (item.TryGetProperty("status",out var upStatus))
                temp.Status = (SubStatus)upStatus.GetInt32();
             if (item.TryGetProperty("service",out var upServ))
                temp.Service = upServ.GetString();
             TryValidateModel(temp);
             if (!ModelState.IsValid)
                return BadRequest(ModelState);
            sub.OwnerId = temp.OwnerId;
            sub.Service = temp.Service;
            sub.Status = temp.Status;
            return NoContent();
        }
        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            var sub = SubData.subsripptionItems.FirstOrDefault(x => x.Id == id)
;
            if (sub is null)
                return NotFound($"Не знайдено підписки з Id {id}");
            SubData.subsripptionItems.Remove(sub);
            return NoContent();
```

```
        }
      }
    }
    MessageController.cs:
    using LW_4_2.Data;
    using LW_4_2.Models;
    using Microsoft.AspNetCore.Mvc;
    using Microsoft.AspNetCore.Mvc.ModelBinding;
    using System.Text.Json;
     namespace LW_4_2.Controllers
    {
      [ApiController]
      [Route("[controller]")]
      public class MessageController: ControllerBase
      {
        [HttpGet]
        public ActionResult<IEnumerable<MessageItems>> GetAll(
          [FromQuery] int? ownerId,
          [FromQuery] int? subId,
          [FromQuery] string? title)
        {
          IEnumerable<MessageItems> messages = MessageData.messageIte
ms;
           if(ownerId is not null)
            messages = messages.Where(x => x.OwnerId == ownerId);
          if (subId is not null)
            messages = messages.Where(x => x.SubId == subId);
          if (title is not null)
            messages = messages.Where(x => x.Title == title);
           if (!messages.Any())
```

```
                return NotFound("Не знайдено елементів за вказаним запитом"
);
                return Ok(messages);
            }
            [HttpGet("{id}")]
            public ActionResult<MessageItems> GetById(int id)
            {
                var message = MessageData.messageItems.FirstOrDefault(x => x.Id
== id);
                if (message is null)
                    return NotFound($"Повідомлення із вказаним Id {id} не знайде
но");
                return Ok(message);
            }
            [HttpPost]
            public ActionResult<MessageItems> Create(MessageItems message)
            {
                message.Id = MessageData.messageItems.Max(x => x.Id) + 1;
                MessageData.messageItems.Add(message);
                return CreatedAtAction(nameof(GetById), new { Id = message.Id},
message);
            }
            [HttpPut("{id}")]
            public IActionResult Update(int id, MessageItems message)
            {
                var mes = MessageData.messageItems.FirstOrDefault(x => x.Id == i
d);
                if (mes is null)
                    return NotFound($"Повідомлення із вказаним Id {id} не знайде
но");
```

```
            mes.OwnerId = message.OwnerId;
            mes.SubId = message.SubId;
            mes.Title = message.Title;
            return NoContent();
        }
        [HttpPatch("{id}")]
        public IActionResult UpdatePart(int id,JsonElement message)
        {
            var mes = MessageData.messageItems.FirstOrDefault(x => x.Id == i
d);
            if (mes is null)
                return NotFound($"Повідомлення із вказаним Id {id} не знайде
но");
            var temp = new MessageItems()
            {
                Id = id,
                OwnerId = mes.OwnerId,
                SubId = mes.SubId,
                Title = mes.Title,
            };
            if(message.TryGetProperty("ownerId",out var upO))
                temp.OwnerId = upO.GetInt32();
            if(message.TryGetProperty("subId", out var upS))
                temp.SubId = upS.GetInt32();
            if (message.TryGetProperty("title", out var upT))
                temp.Title = upT.GetString();
            TryValidateModel(temp);
            if(!ModelState.IsValid)
                return BadRequest(ModelState);
            mes.OwnerId = temp.OwnerId;
```

```
                mes.SubId = temp.SubId;
                mes.Title = temp.Title;
                return NoContent();
            }
            [HttpDelete("{id}")]
            public IActionResult Delete(int id)
            {
                var mes = MessageData.messageItems.FirstOrDefault(x => x.Id == i
d);
                if (mes is null)
                    return NotFound($"Повідомлення із вказаним Id {id} не знайде
но");
                MessageData.messageItems.Remove(mes);
                return NoContent();
            }
        }
    }
    Program.cs:
    using FluentValidation;
    using FluentValidation.AspNetCore;
    using LW_4_2.Models;
    using LW_4_2.Validator;
    var builder = WebApplication.CreateBuilder(args);
    // Add services to the container.
    builder.Services.AddControllers().AddFluentValidation();
    // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetc
ore/swashbuckle
    builder.Services.AddEndpointsApiExplorer();
    builder.Services.AddSwaggerGen(oprions =>
    {
```

```
        var xmlFile = $"{System.Reflection.Assembly.GetExecutingAssembly().
GetName().Name}.xml";
        var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
        oprions.IncludeXmlComments(xmlPath);
    });
    builder.Services.AddValidatorsFromAssemblyContaining<SubscriptionVali
dator>();
    builder.Services.AddValidatorsFromAssemblyContaining<MessageValidato
r>();
    var app = builder.Build();
    // Configure the HTTP request pipeline.
    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }
    app.UseAuthorization();
    app.MapControllers();
    app.Run();
```

Протестовано роботу програми за допомогою swagger:

Запит 1

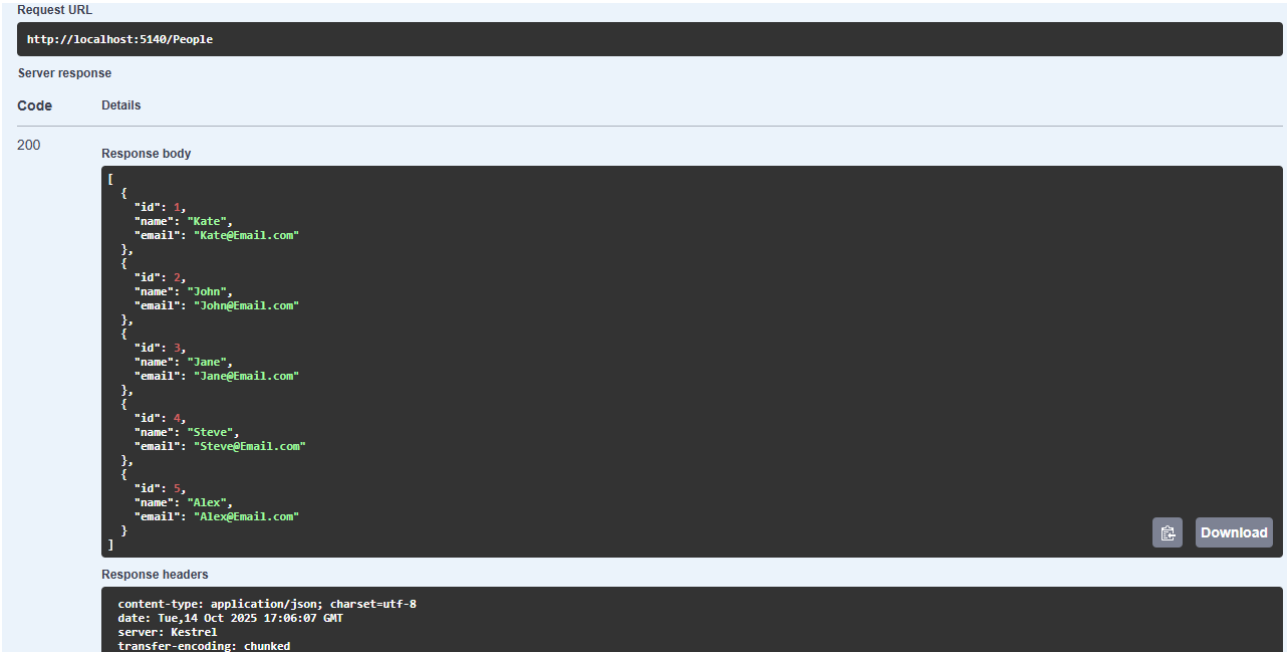


Рисунок 39 – Отримання всіх користувачів

Запит 2

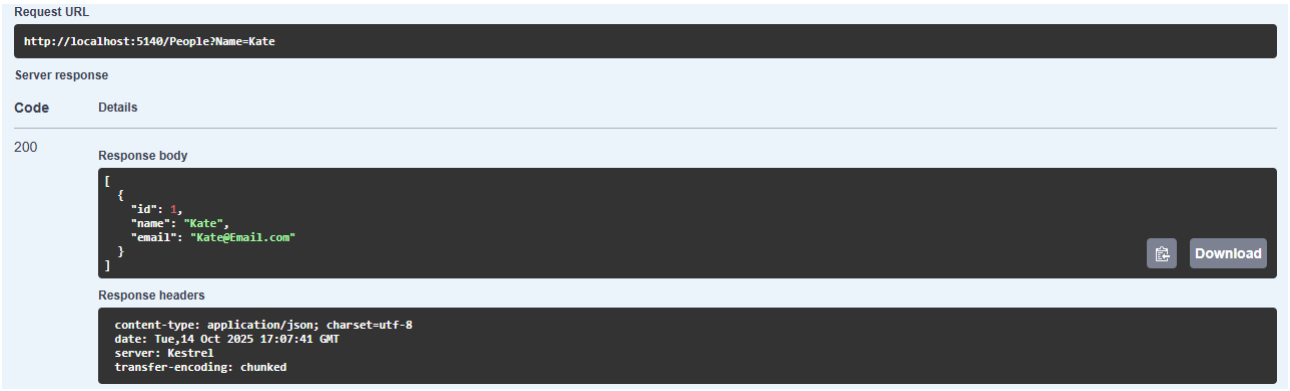


Рисунок 40 – Отримання користувачів за одним параметром

Запит 3

Рисунок 41 -Отримання користувачів за двома параметрами

Запит 4

Рисунок 42 -Спроба отримати користувачів за хибним параметром запиту

Запит 5

Рисунок 43 – Отримання користувача за Id

Запит 6

Рисунок 44 – Спроба отримати неіснуючого користувача

Запит 7

Рисунок 45 – Додавання нового користувача

Запит 8

Рисунок 46 – Спроба додавання користуваа із некоректним логінок

Запит 9

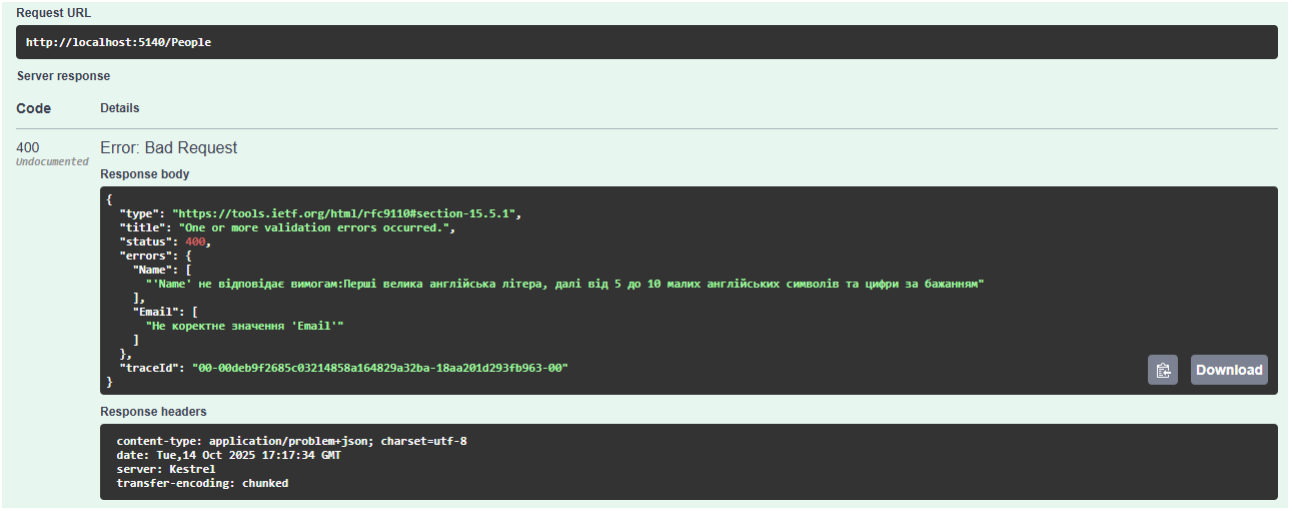


Рисунок 47 – Спроа додавання користувача із некоректними полями

Запит 10

Рисунок 48 – Спроба відправлення пустого поля

Запит 11

Рисунок 49 – Оновлення інформації про першого користувача

Запит 12

Рисунок 50 – Спроа оновлення з використання некоректного логіну

Запит 13

Рисунок 51 – Спроба оновленн на пусті поля

Запит 14

Рисунок 52 – Часткове оновлення користувача

Запит 15

Рисунок 53 – Спроба оновлення на пусте поле

Запит 16

Рисунок 54 – Видалення користувача

Запит 17

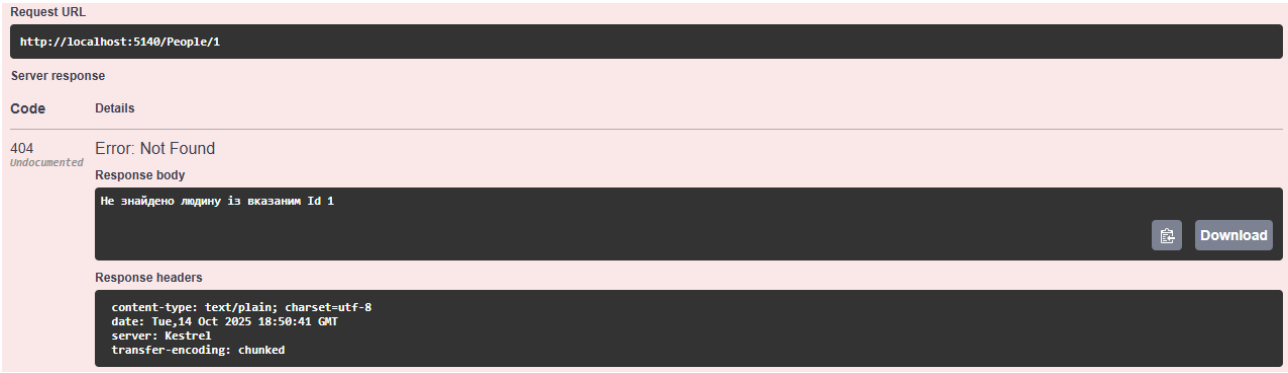


Рисунок 55 – Спроба видалення користувача

Запит 18

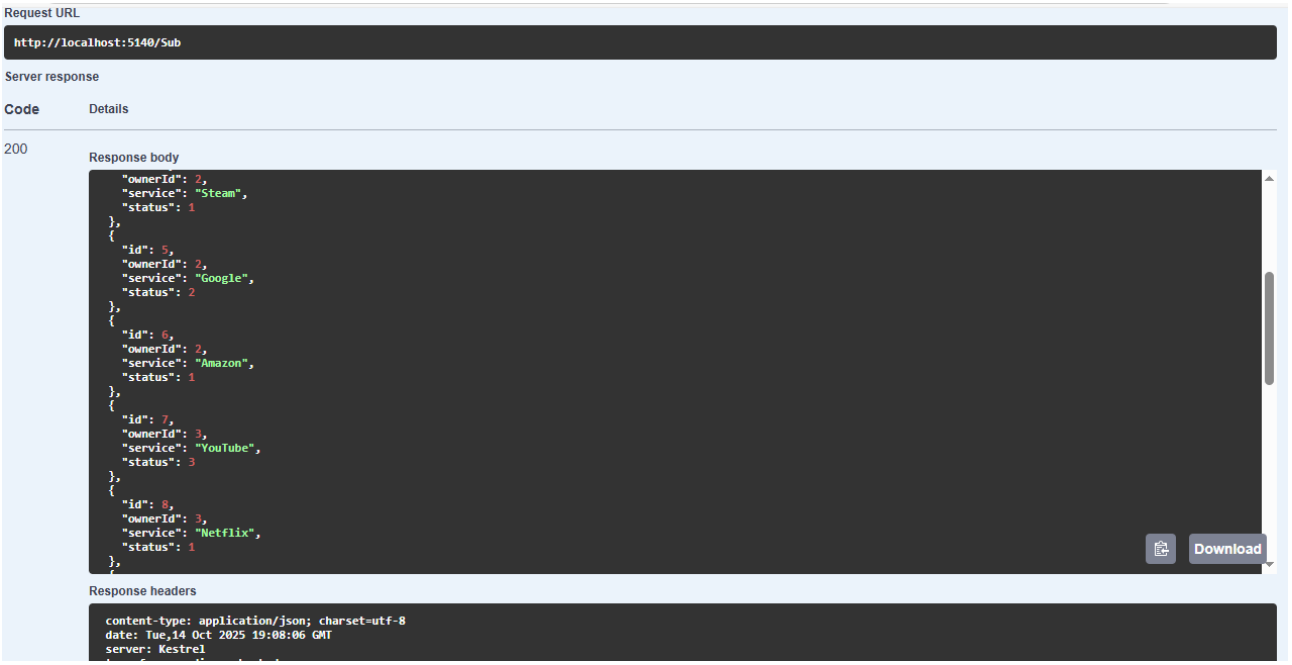


Рисунок 56 – Отримано список всіх підписок

Запит 19

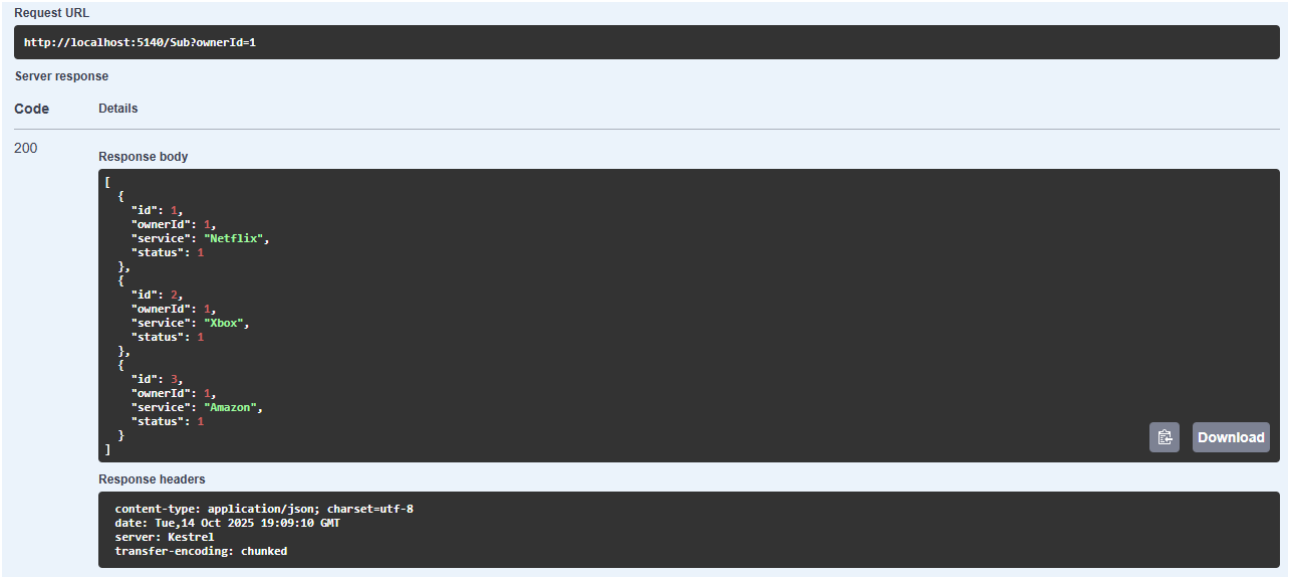


Рисунок 57 – Отримання підписок з використанням параметра

Запит 20

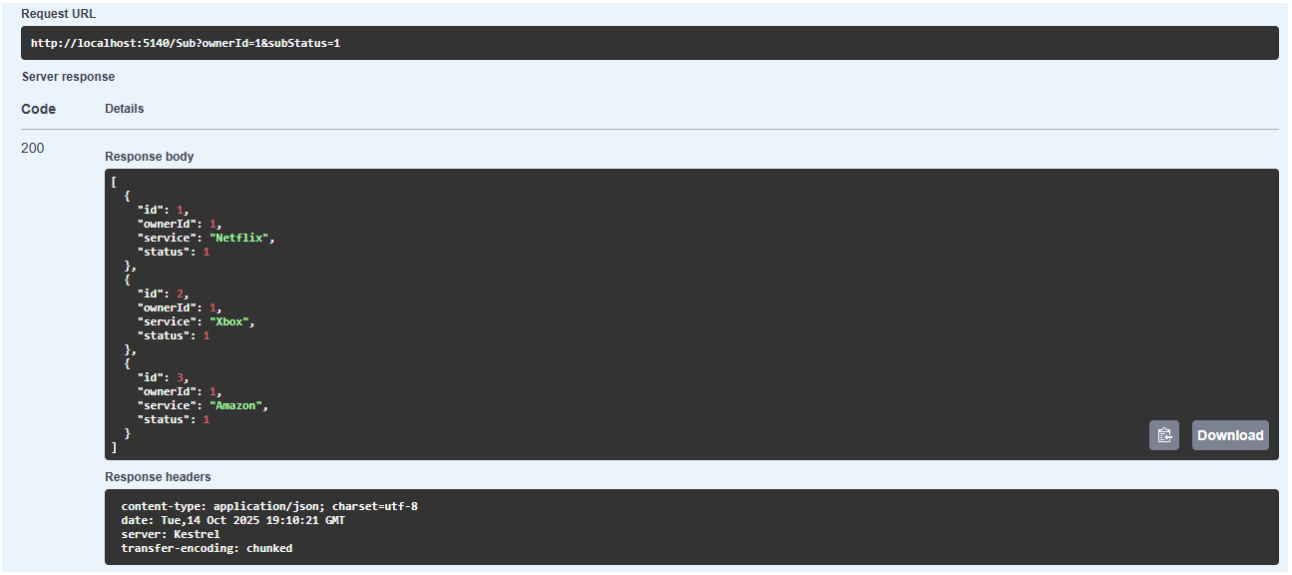


Рисунок 58 – Отримання підписок за деількома параметрами

Запит 21

Рисунок 59 – Спроба отримати підписки за відсутнім параметром

Запит 22

Рисунок 60 – Отримання підписки за Id

Запит 23

Рисунок 61 – Спроба отримання неіснуючої підписки за Id

Запит 24

Рисунок 62 – Додавання нової підписки

Запит 25

Рисунок 64 – Спроа додавання підписки із неіснуючим користувачем

Запит 26

Рисунок 65 – Ще одна спроба додати невірну інормацію про підписку

Запит 27

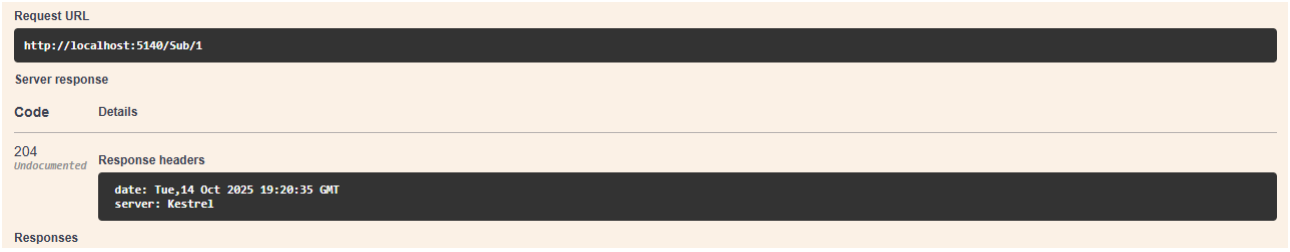


Рисунок 66 – Оновлення інформації про підпску

Запит 28

Рисунок 67 – Спроа некоректного оновлення підписки

Запит 29

Рисунок 68 – Часткове оновлення власника підписки

Запит 30

Рисунок 69 – Спроба часткового оновлення підписки невірним користувачем

Запит 31

Рисунок 70 – Видалення підписки за Id

Запит 32

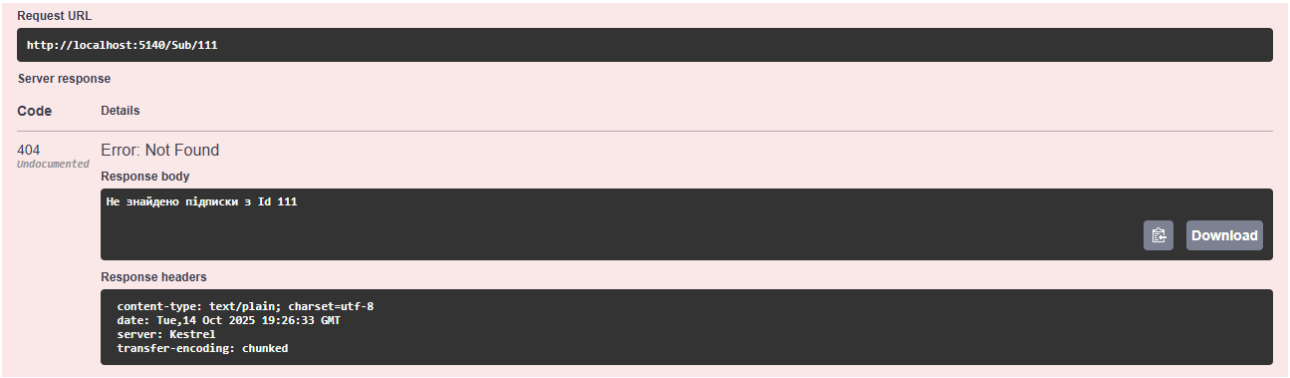


Рисунок 71 – Спроба видалення неіснуючої підписки

Запит 33

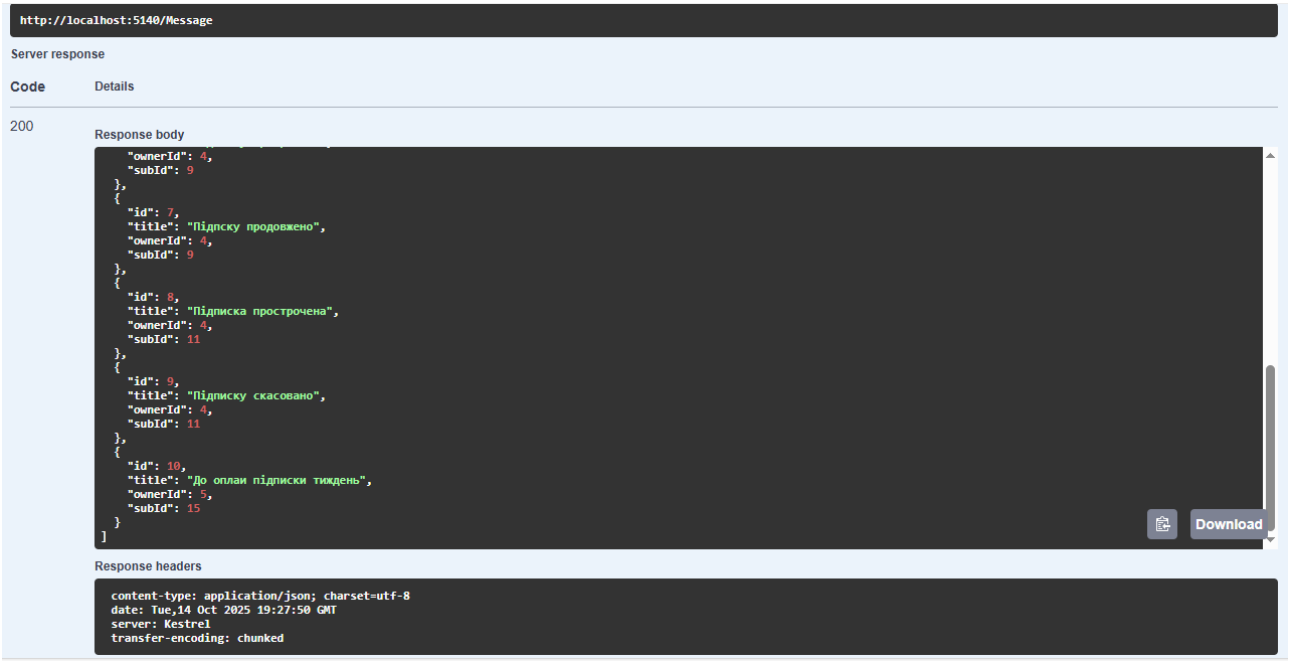


Рисунок 72 – Отримання списку всіх повідомлень

Запит 34

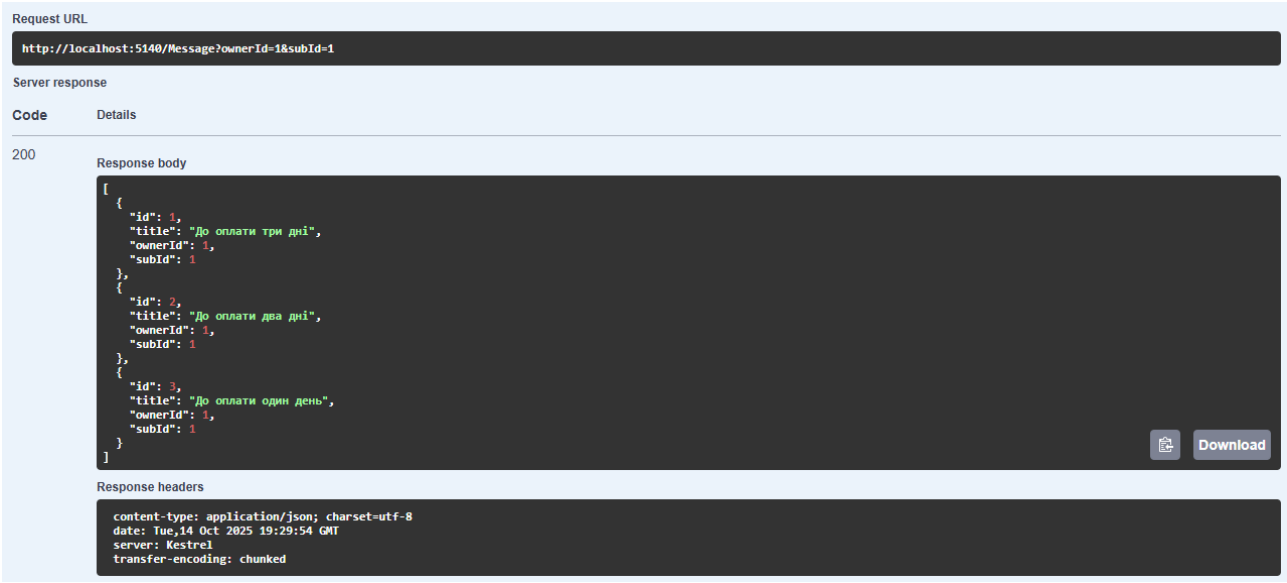


Рисунок 73 – Отримання повідомлень за вказаними парамерами

Запит 35

Рисунок 74 – Спроба отримати неіснуюче повідомлення

Запит 36

Рисунок 75 – Отримання повідомлення за Id

Запит 37

Рисунок 76 – Спроба отримання неіснуючого повідомлення за Id

Запит 38

Рисунок 77 – Додавання нової підписки

Запит 39

Request URL

```
http://localhost:5140/Message
```

Server response

Code	Details
400 *Undocumented*	Error: Bad Request

Response body


```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
  "title": "One or more validation errors occurred.",
  "status": 400,
  "errors": {
    "Title": [
      "Поле 'Title' не може бути порожнім",
      "Поле 'Title' має бути від трьох до ста символів"
    ],
    "OwnerId": [
      "Відсутня підписка за вказаним 'SubId' або для неї не існує користувача"
    ]
  },
  "traceId": "00-62fe2d8dd958ab132e24aa9f06b4e7f0-17af6db3c1e9b08b-00"
}
```


Download

Response headers

```
content-type: application/problem+json; charset=utf-8
date: Tue,14 Oct 2025 19:35:51 GMT
server: Kestrel
transfer-encoding: chunked
```

Рисунок 78 – Спроба додавання повідомлення про підписку не тому користувачеві та з порожніс текстом

Запит 40

Request URL

```
http://localhost:5140/Message/1
```

Server response

Code	Details
204 *Undocumented*	Response headers

```
date: Tue,14 Oct 2025 19:37:42 GMT
server: Kestrel
```

Рисунок 79 – Коректне оновлення повідомлення

Запит 41

400 *Undocumented* Error: Bad Request

Response body


```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
  "title": "One or more validation errors occurred.",
  "status": 400,
  "errors": {
    "Title": [
      "Поле 'Title' не може бути порожнім"
    ],
    "OwnerId": [
      "Відсутня підписка за вказаним 'SubId' або для неї не існує користувача"
    ]
  },
  "traceId": "00-a186d7ff4bbbdc3c3d32e132d7ddf193-ac9107b4cd152008-00"
}
```


Download

Response headers

```
content-type: application/problem+json; charset=utf-8
date: Tue,14 Oct 2025 19:38:38 GMT
server: Kestrel
transfer-encoding: chunked
```

Рисунок 80 – Спроба оновити повідомлення, присвоївши підписку не ому користувачеві та з порожнім підписом

Запит 42

Request URL

```
http://localhost:5140/Message/1
```

Server response

Code	Details
204 *Undocumented*	Response headers

```
date: Tue,14 Oct 2025 19:44:11 GMT
server: Kestrel
```

Рисунок 81 – Коректне часткове оновлення

Запит 43

Рисунок 82 – Спроба часткового оновлення повідомлення пустим тектом

Запит 44

Рисунок 83 – Видалення повідомлення

Запит 45

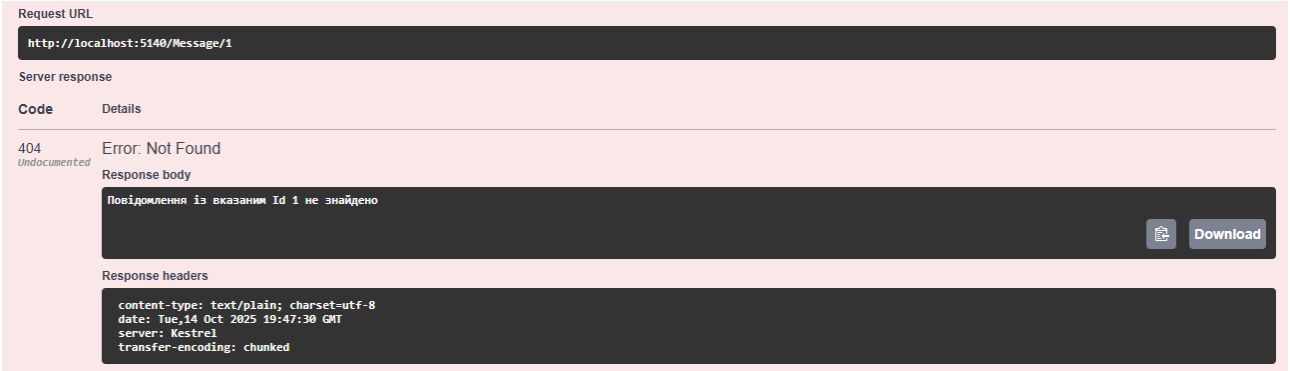


Рисунок 84 – Спроба видалення повідолення ще раз

Завдання 3

Створено діаграму компонентів для програми по відображенню підписок користувача. Діаграма складається Controller Layer – відповідає за прийняття та відправлення http – запитів, валідацію вхідної інформації, Service Layer – відповідає за бізнес логіку, Repository Layer – відповідає за логіку

звернення до клієнта БД, DB Layer – відповідає за створення клієнта БД із потрібним вмістом.

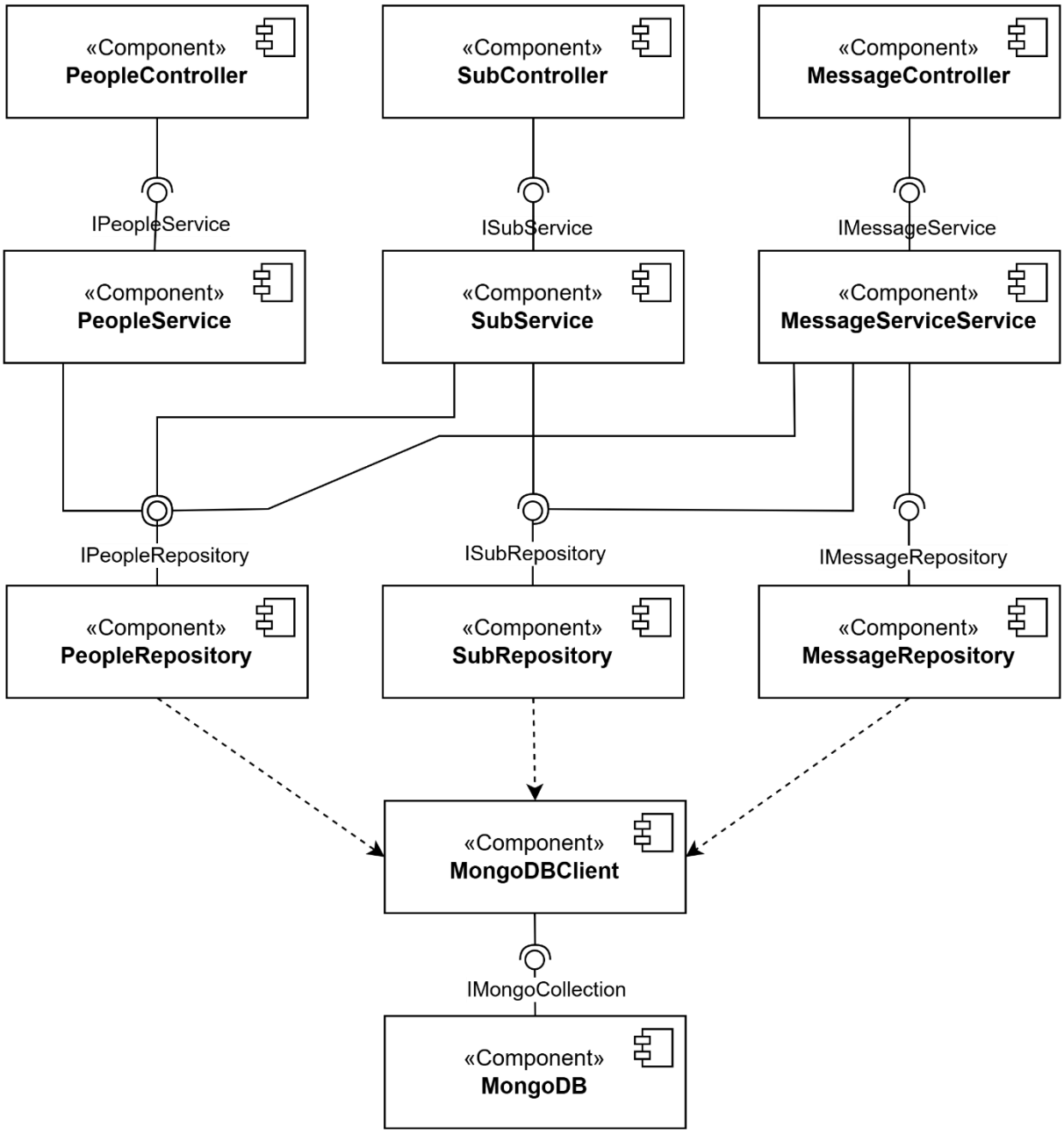


Рисунок 85 – Component Diagram

Controller Layer – надає клієнтові методи для звернення до ресурсів за вказаними ендпоінтами. Він складається із трьох компонентів(контролер для користувачів, підписок, повідомлень), кожний із який надає можливість виконання CRUD – операцій, приклади методів Create – створення сутності, Get – отримання, Update – Оновлення, Delete – Видалення.

Service Layer – виконує бізнес логіку для запитів/об’єктів від контролерів. Також складається із трьох компонентів (сервіс для користувачів, підписок, повідомлень), реалізує відповідний для нього інтерфейс із набором методів, які має виконати сервіс. Наприклад має реалізувати такі методи: CreateAsync – створення, GetAsync – отримання, UpdateAsync – Оновлення, DeleteAsync – Видалення.

Repository Layer – виконує логіку звернення до клієнта БД для отримання елементів БД чи оновлення. Також складається із трьох компонентів (репозиторій для користувачів, підписок, повідомлень), реалізує відповідний для нього інтерфейс із набором методів, які має виконати репозторій. Наприклад має реалізувати такі методи: CreateAsync – створення, GetAsync – отримання, UpdateAsync – Оновлення, DeleteAsync – Видалення.

DB Layer – реалізує логіку клієнта для отримання колекції/таблиці елементів із БД. Наприклад реалізує метод GetCollection, за яким можна отримати колекцію потрібних елементів.

Завдання 4

Вдосконалено проект із другого завдання, використовуючи компонентну діаграму із третього завдання. В проектові створено підключення до MongoDB, оновлено моделі, створено шар репозиторіїв, які реалізують відповідні до них інтерфейси, та використовують MongoDBClient, створено шар сервісів, які реалізують відповідні їм інтерфейси, та які використовують репозиторії, також оновлено шар контролерів, які використовують сервіси

Виконано CRUD операції із використанням MongoDB:

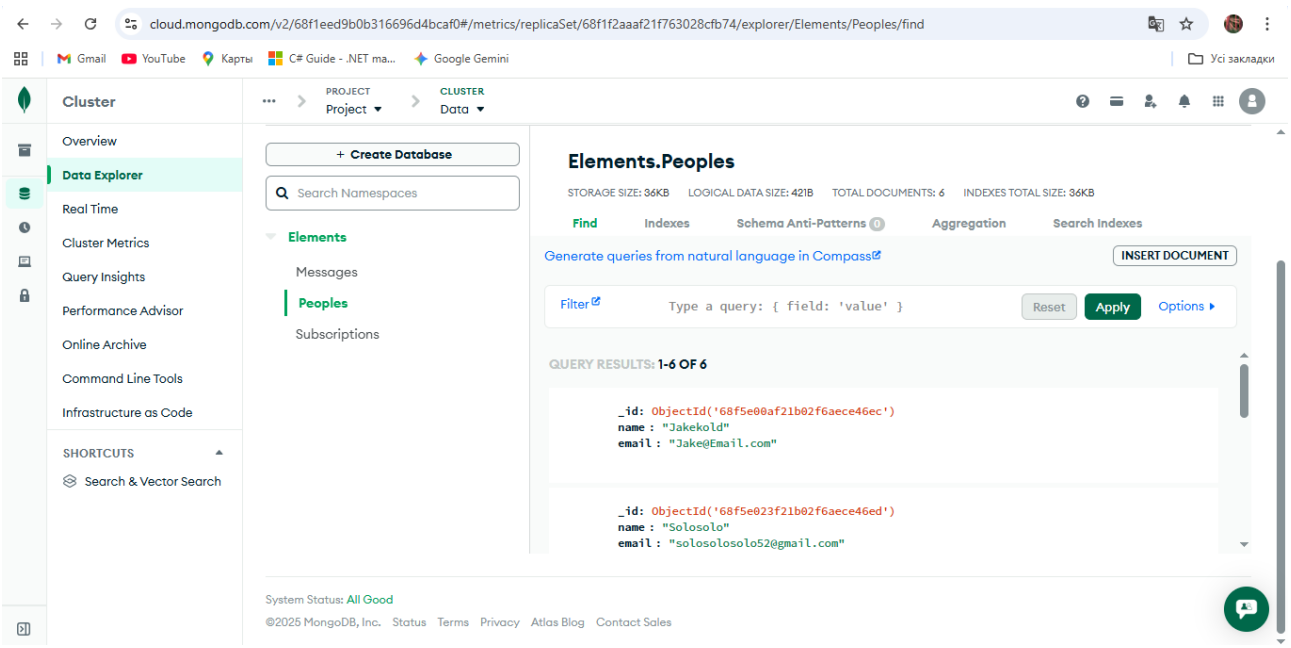


Рисунок 86 – Початковий вміст БД

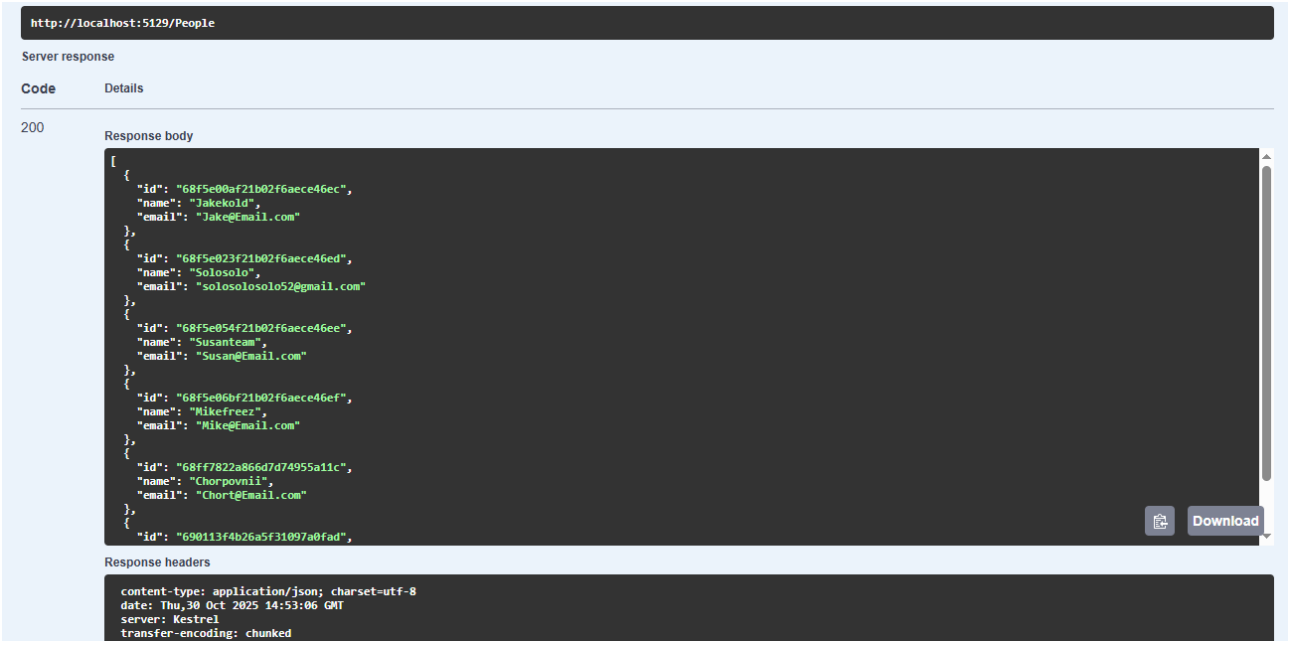


Рисунок 87 – Отримання всіх користувачів

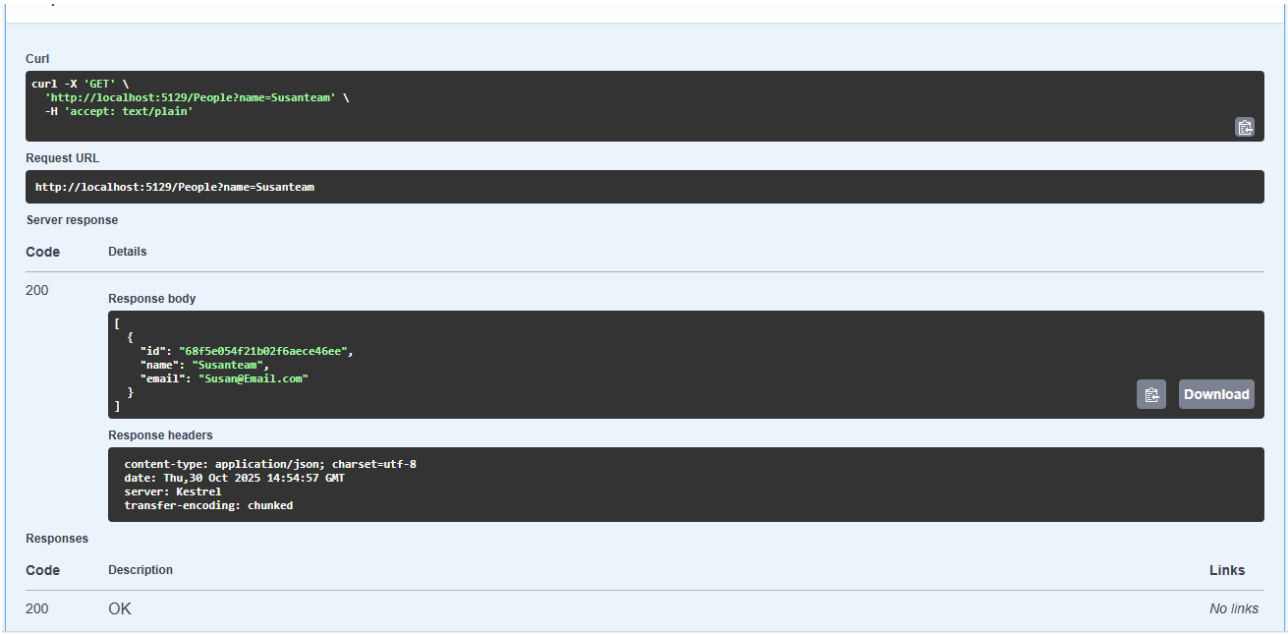


Рисунок 88 – Отримно користувача з використанням фільтру

Рисунок 89 – Отримано користувача за Id

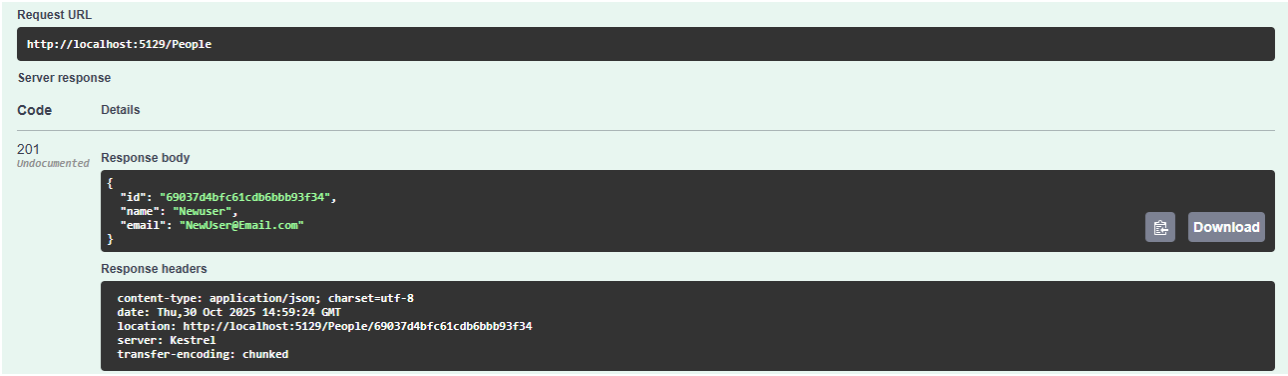


Рисунок 90 – Створено нового користувача

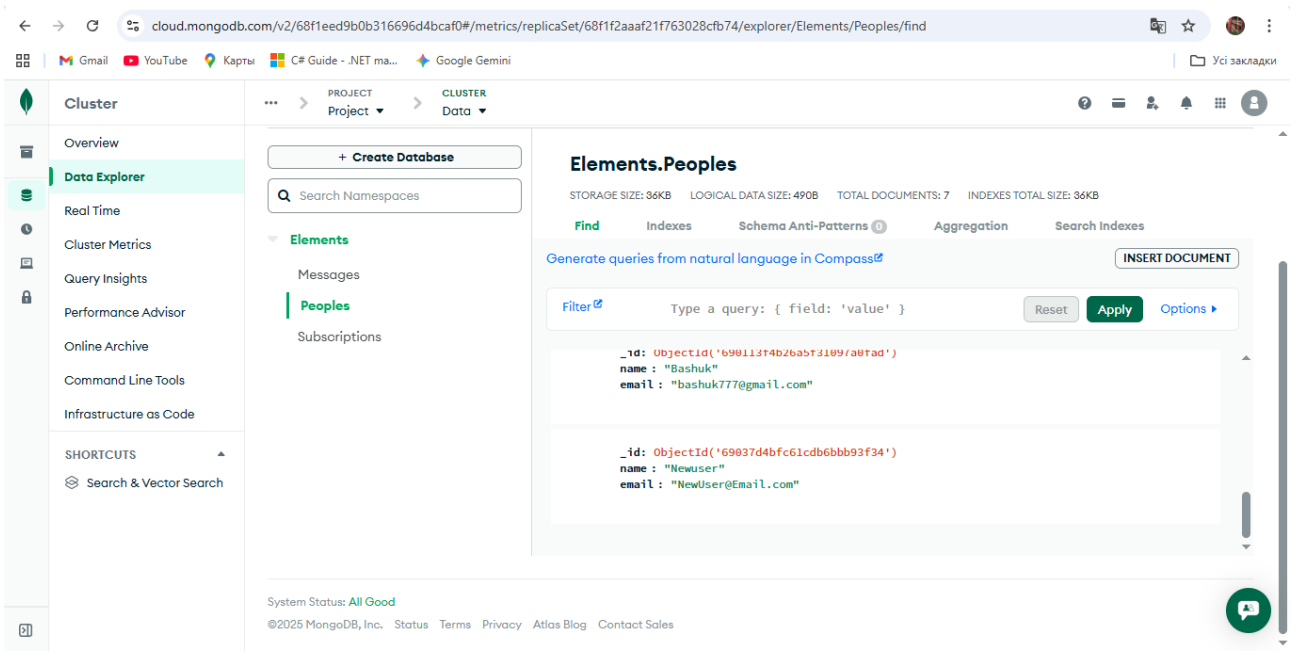


Рисунок 91 – Вміст бази даних після додавання нового користувача

Рисунок 92 – Повне оновлення користувача

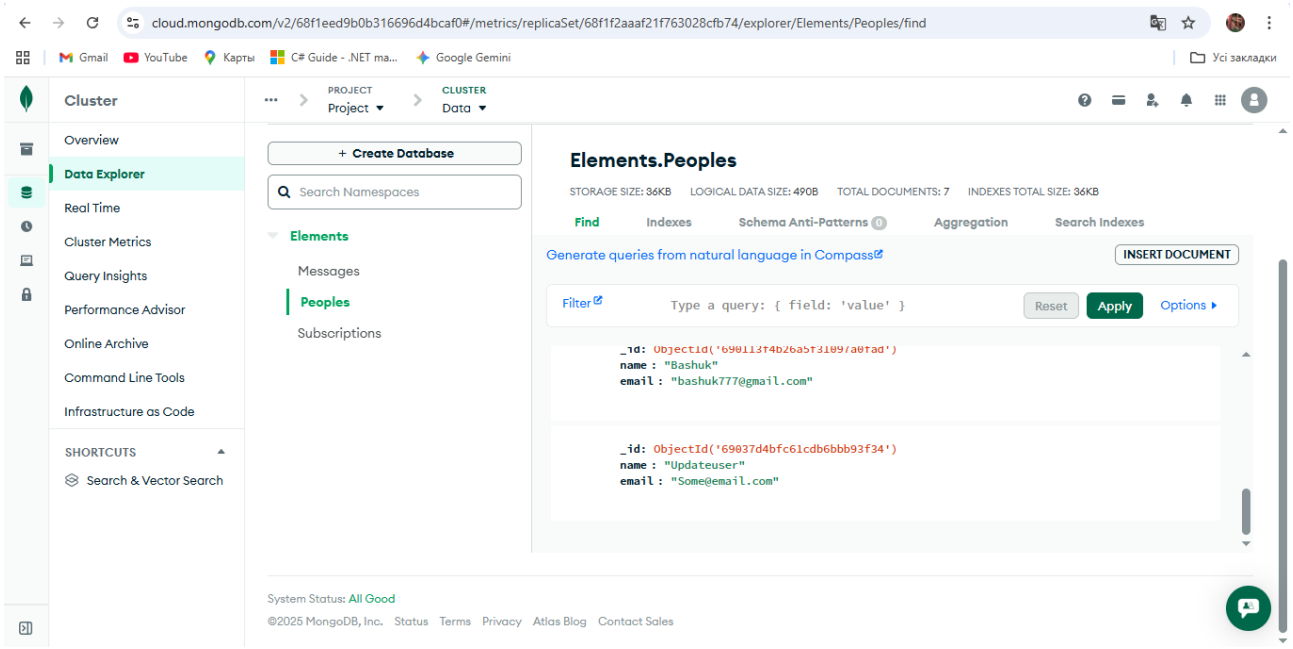


Рисунок 93 – Вміст після повного оновлення

Рисунок 94 – Часткове оновлення користувача

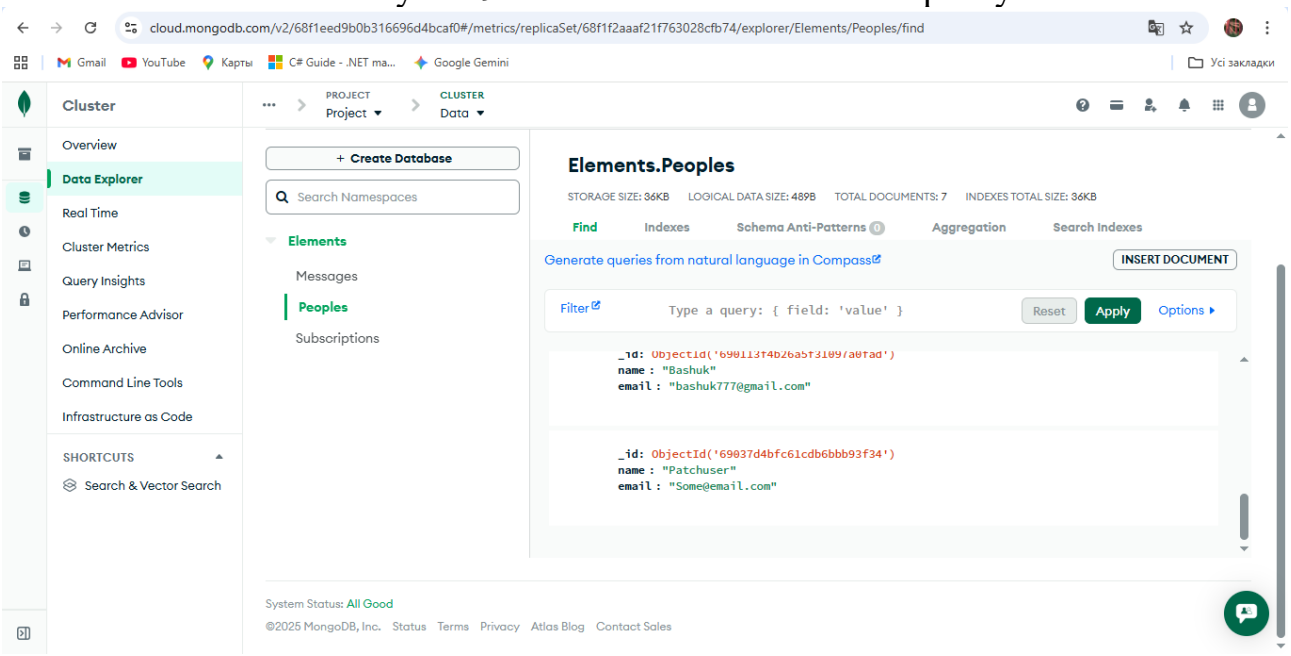


Рисунок 95 – Вміст бази даних після часткового оновлення

Рисунок 96 – Видалення користувача

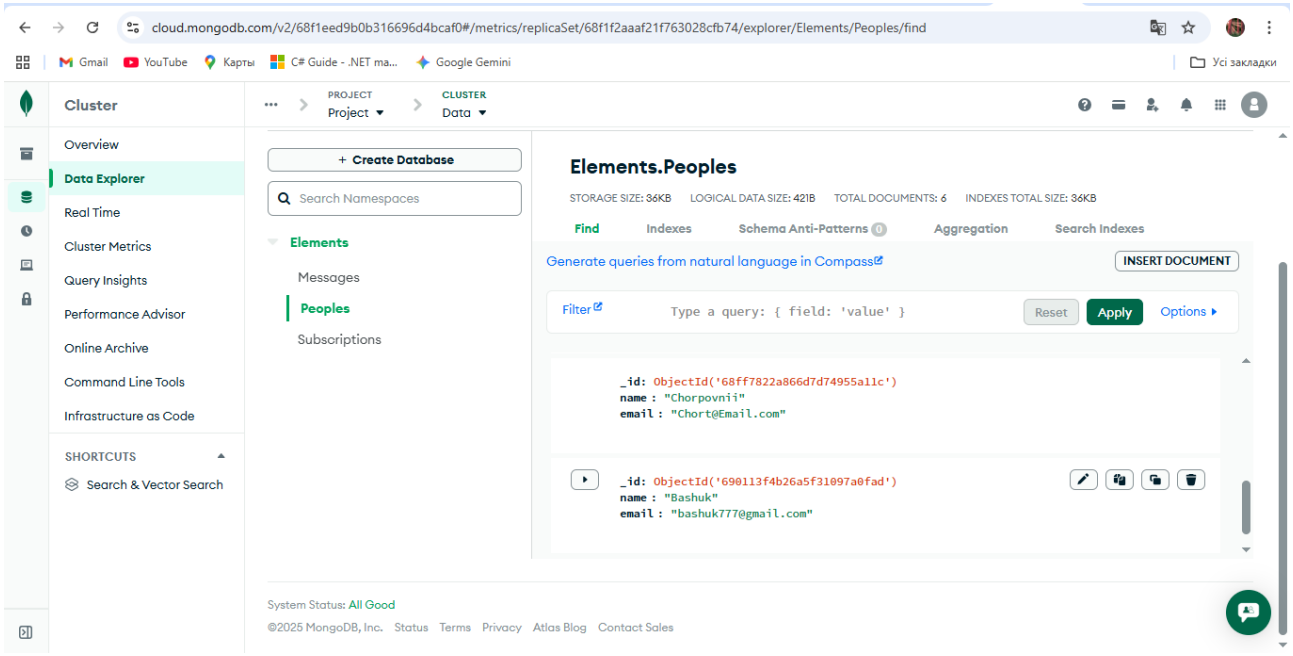


Рисунок 97 – Вміст бази даних після видалення

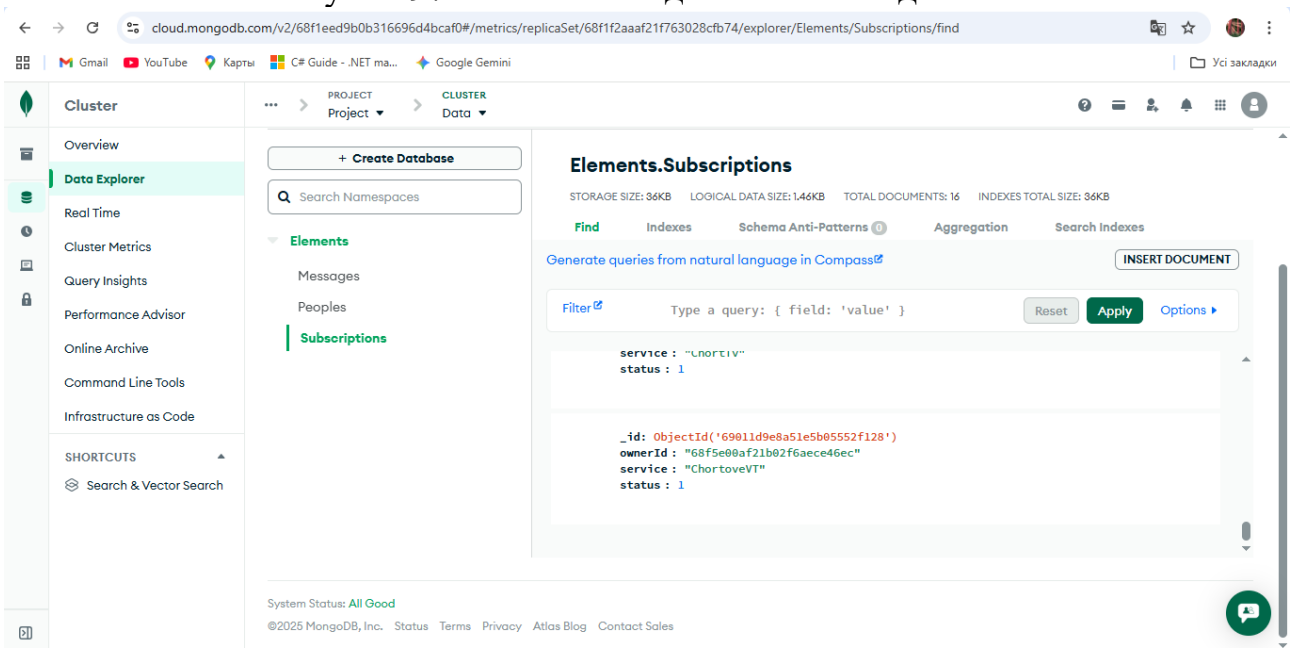


Рисунок 98 – Вміст колекції підписок

http://localhost:5129/Sub

Server response

Code Details

200

Response body

Response headers

```
content-type: application/json; charset=utf-8
date: Thu,30 Oct 2025 15:09:10 GMT
server: Kestrel
transfer-encoding: chunked
```

Рисунок 99 – Отримано всі підписки користувачів

Request URL

http://localhost:5129/Sub?service=XBox&status=1

Server response

Code Details

200

Response body

Response headers

```
content-type: application/json; charset=utf-8
date: Thu,30 Oct 2025 15:10:06 GMT
server: Kestrel
transfer-encoding: chunked
```

Рисунок 100 – Отримано підписки за вказаними фільтрами

Request URL

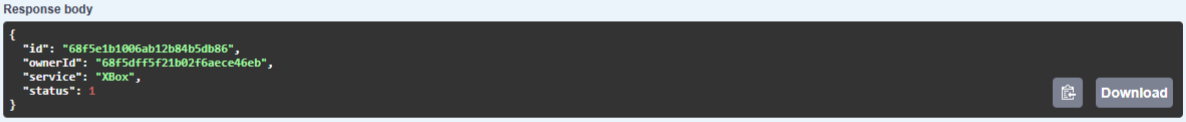
http://localhost:5129/Sub/68f5e1b1006ab12b84b5db86

Server response

Code Details

200

Response body



Response headers

```
content-type: application/json; charset=utf-8
date: Thu,30 Oct 2025 15:10:48 GMT
server: Kestrel
transfer-encoding: chunked
```

Рисунок 101 – Отримано підписку за конкретним Id

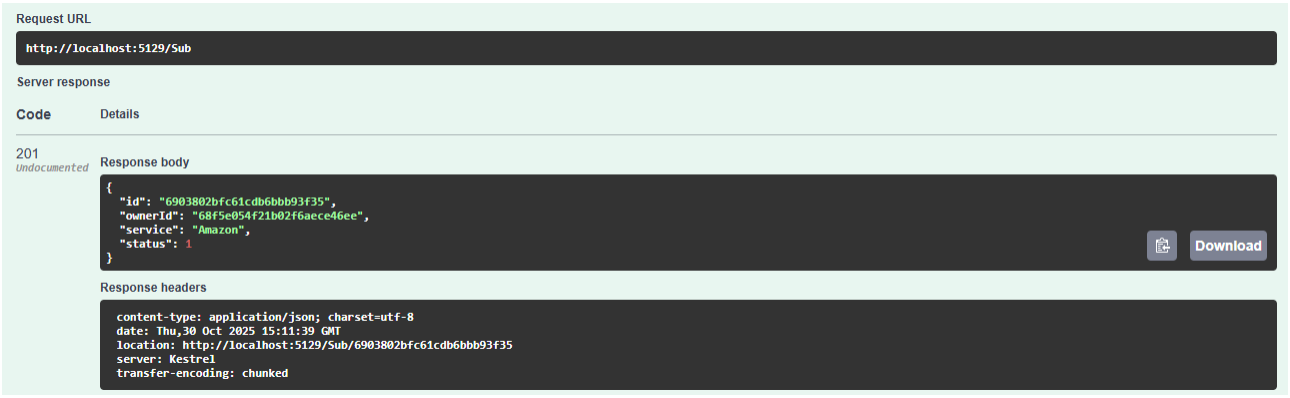


Рисунок 102 – Створено нову підписку для користувача

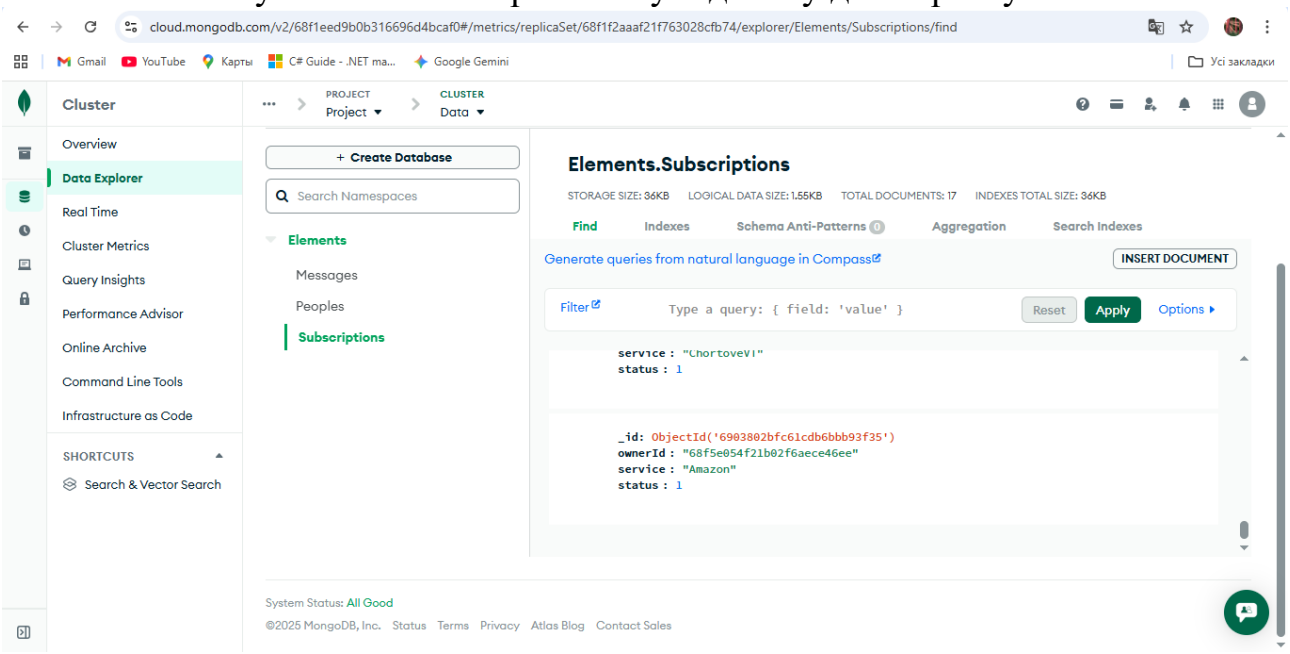


Рисунок 103 – Вміст бази даних після створення підписки

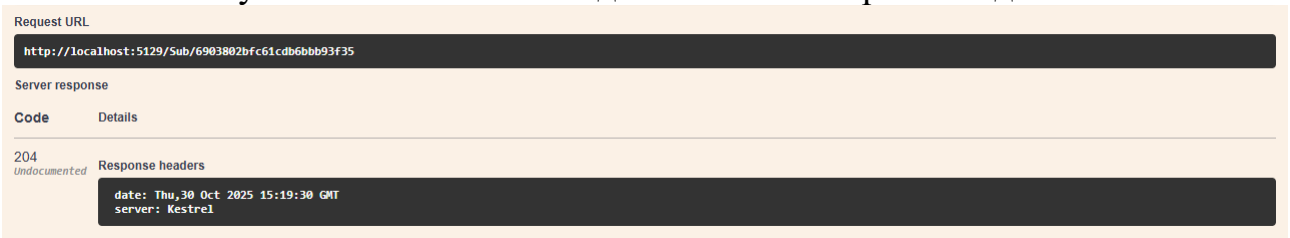


Рисунок 104 – Повне оновлення підписки

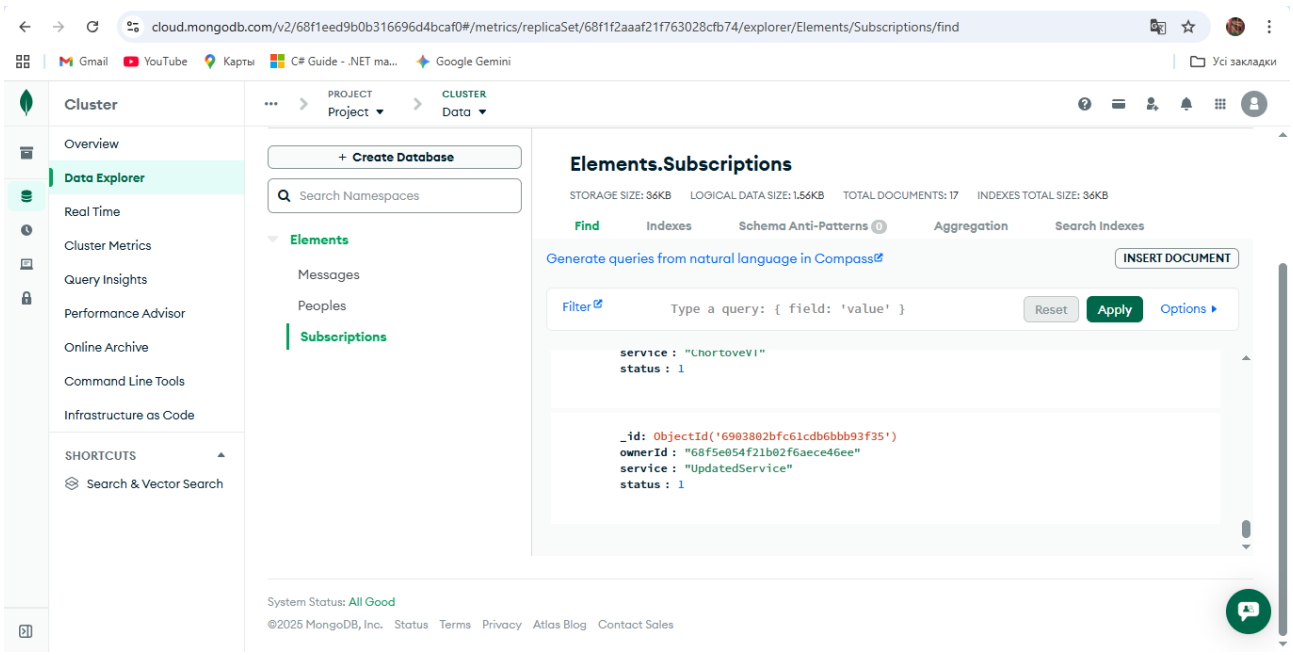


Рисунок 105 – Вміст колекції після повного оновлення

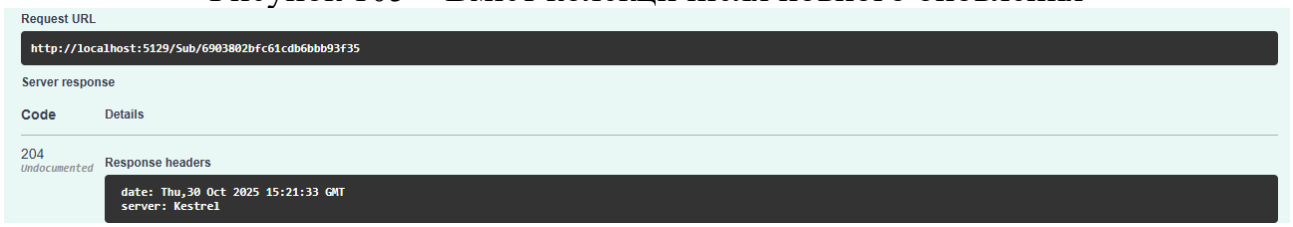


Рисунок 106 – Часткове оновлення підписки

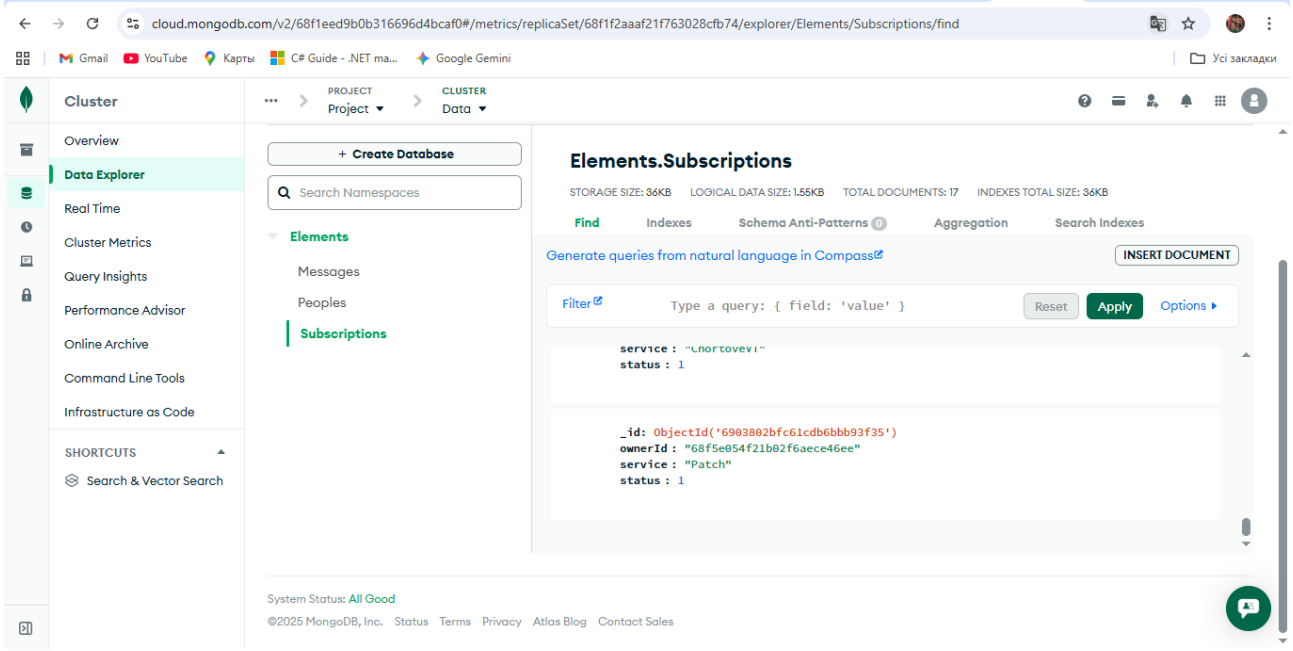


Рисунок 107 – Вміст колекції після часткового оновлення

Рисунок 108 – Видалення підписки

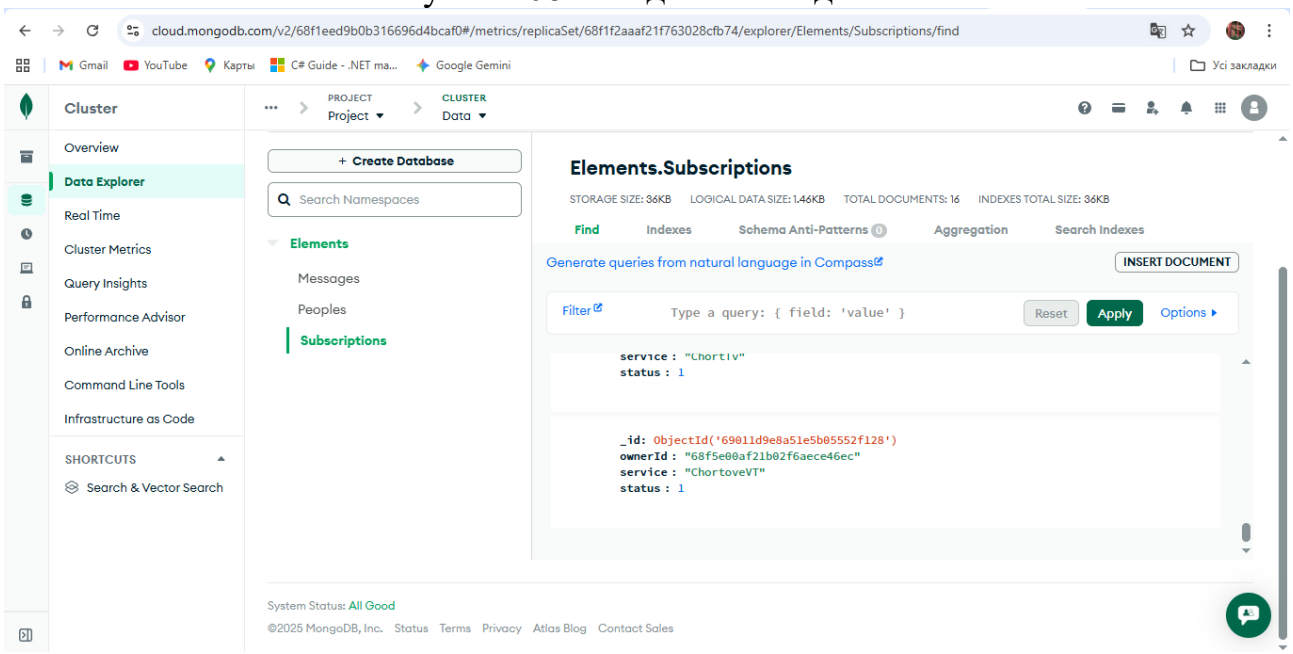


Рисунок 109 – Вміст колекції після видалення підписки

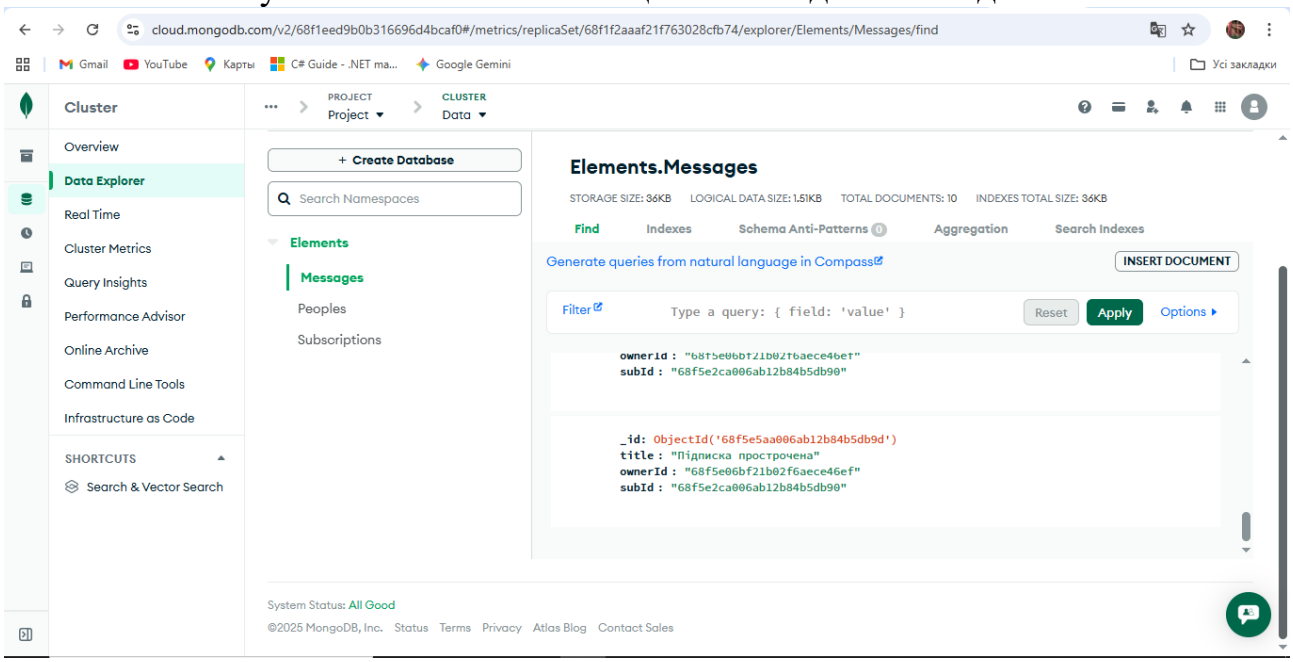


Рисунок 110 – Вміст колекції повідомлень

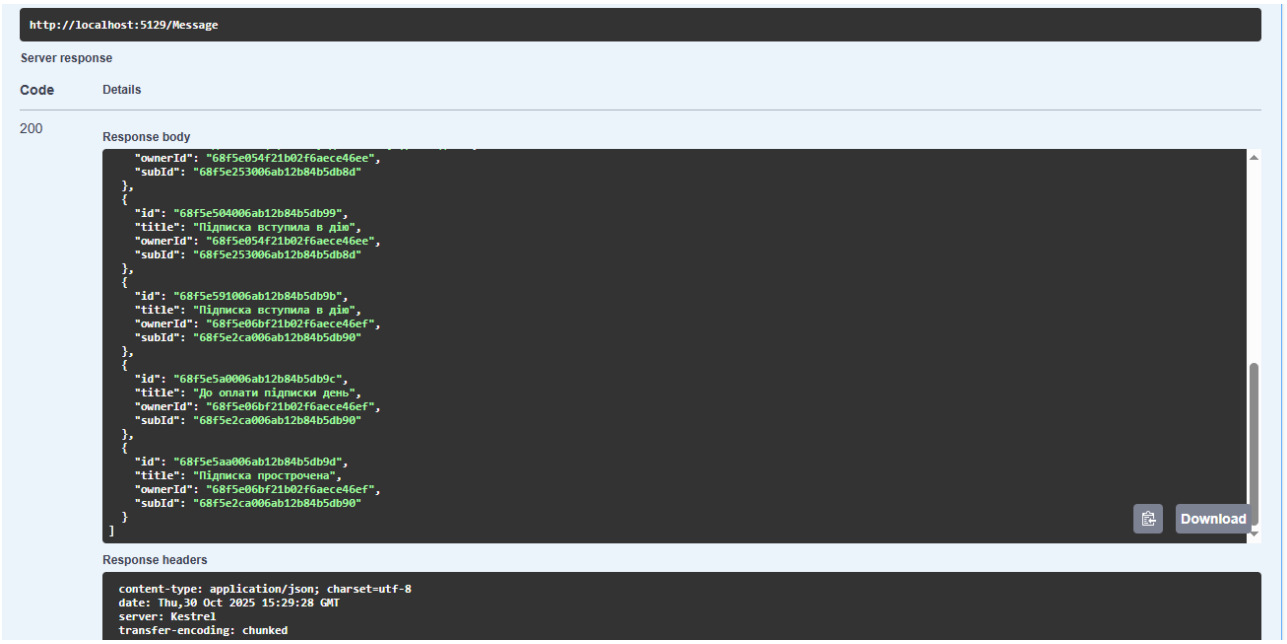


Рисунок 111 – Отримання списку всіх повідомлень

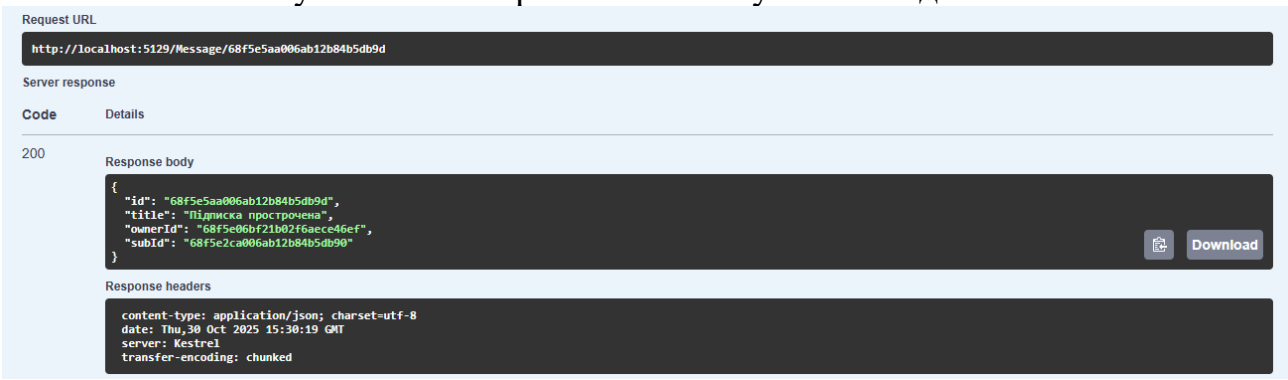


Рисунок 112 – Отримання конкретного повідомлення

Рисунок 113 – Створення нового повідомлення

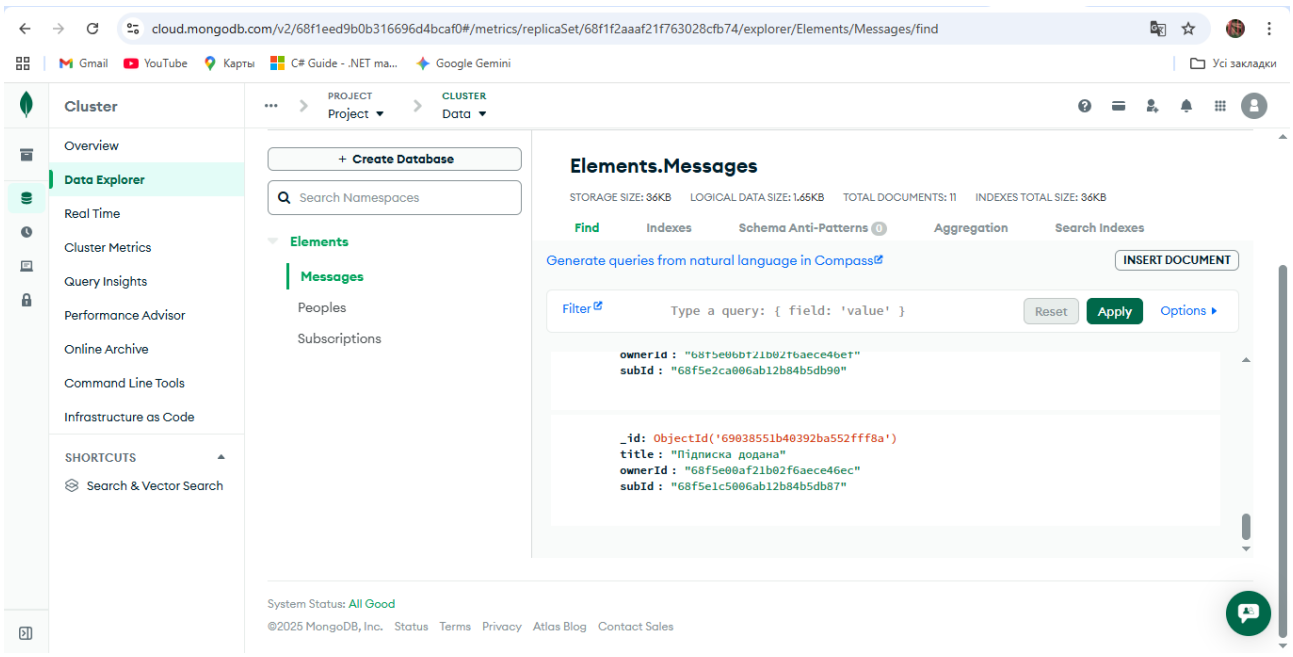


Рисунок 114 – Вміст колекції після додавання

Рисунок 115 – Повне оновлення повідомлення

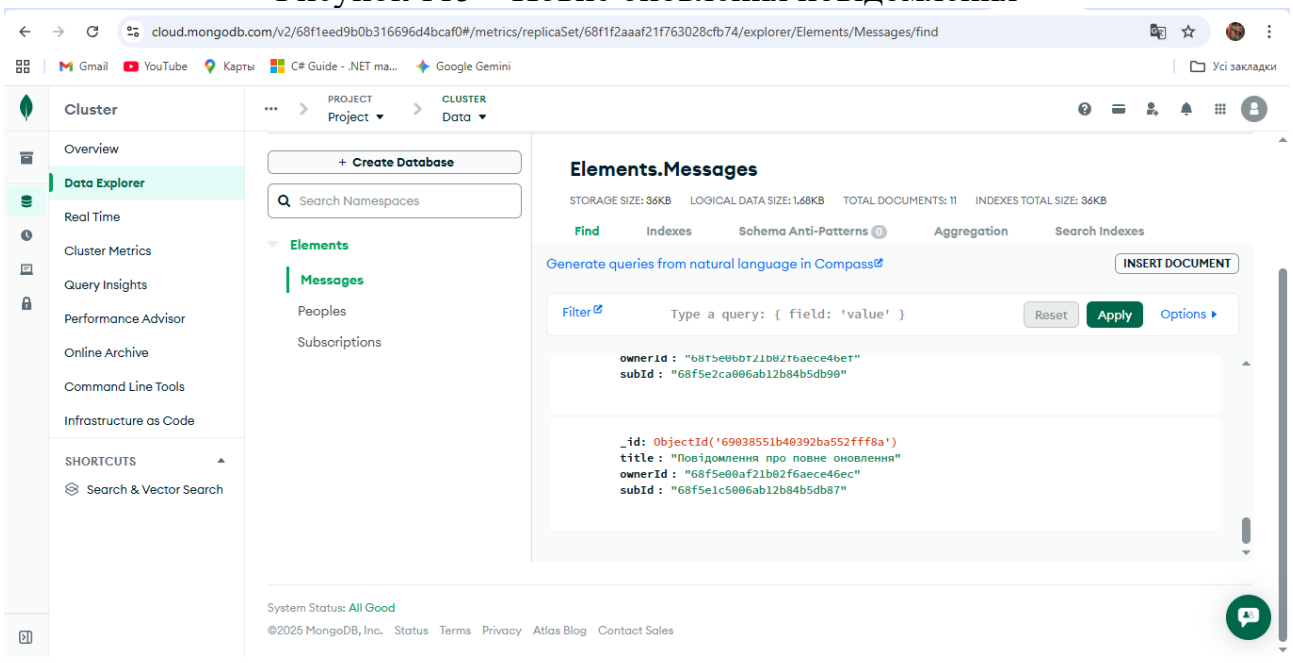


Рисунок 116 – Вміст колекції після повного оновлення

Рисунок 117 – Часткове оновлення повідомлення

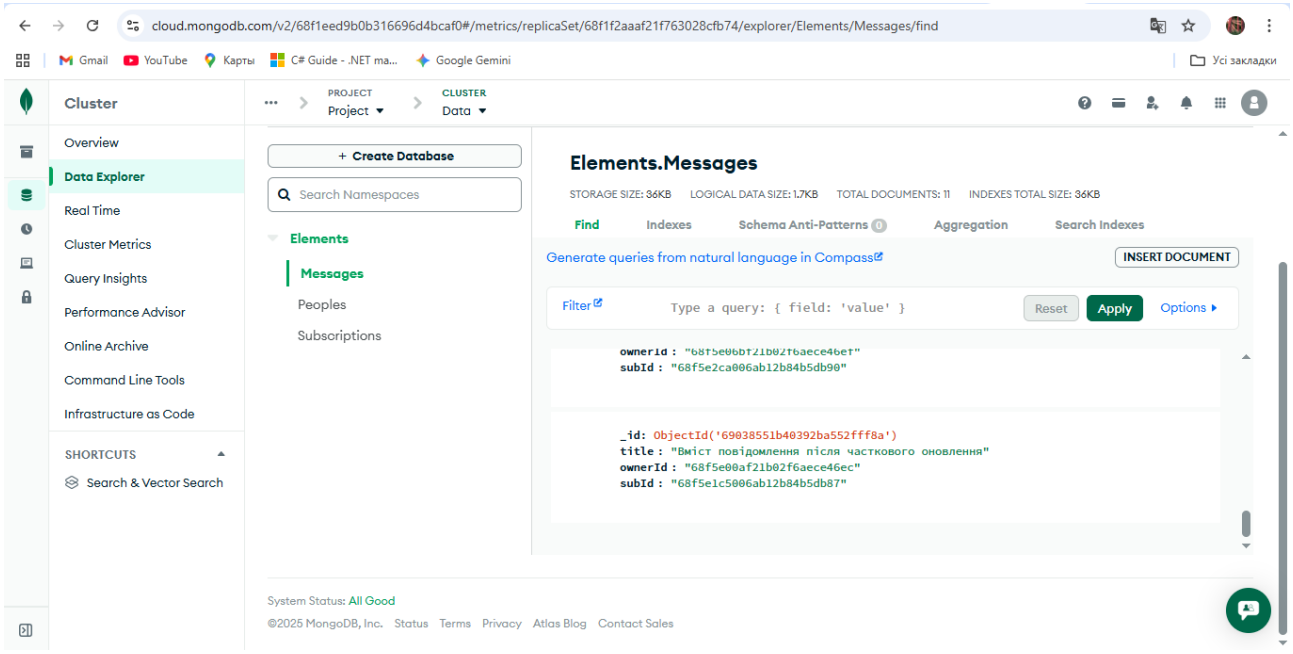


Рисунок 118 – Вміст колекції після часткового оновлення повідомлення

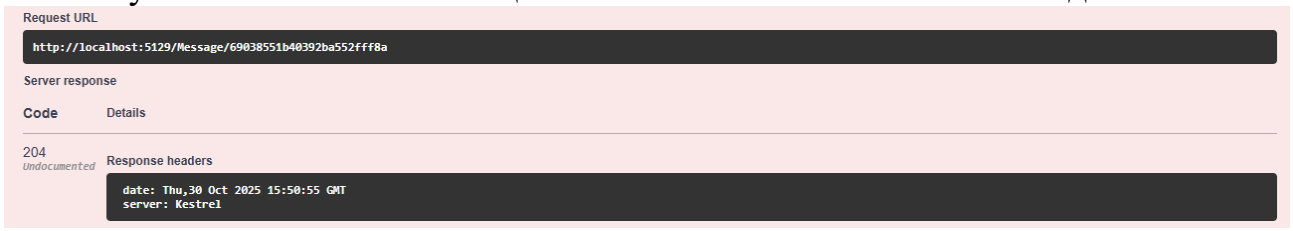


Рисунок 119 – Видалення повідомлення

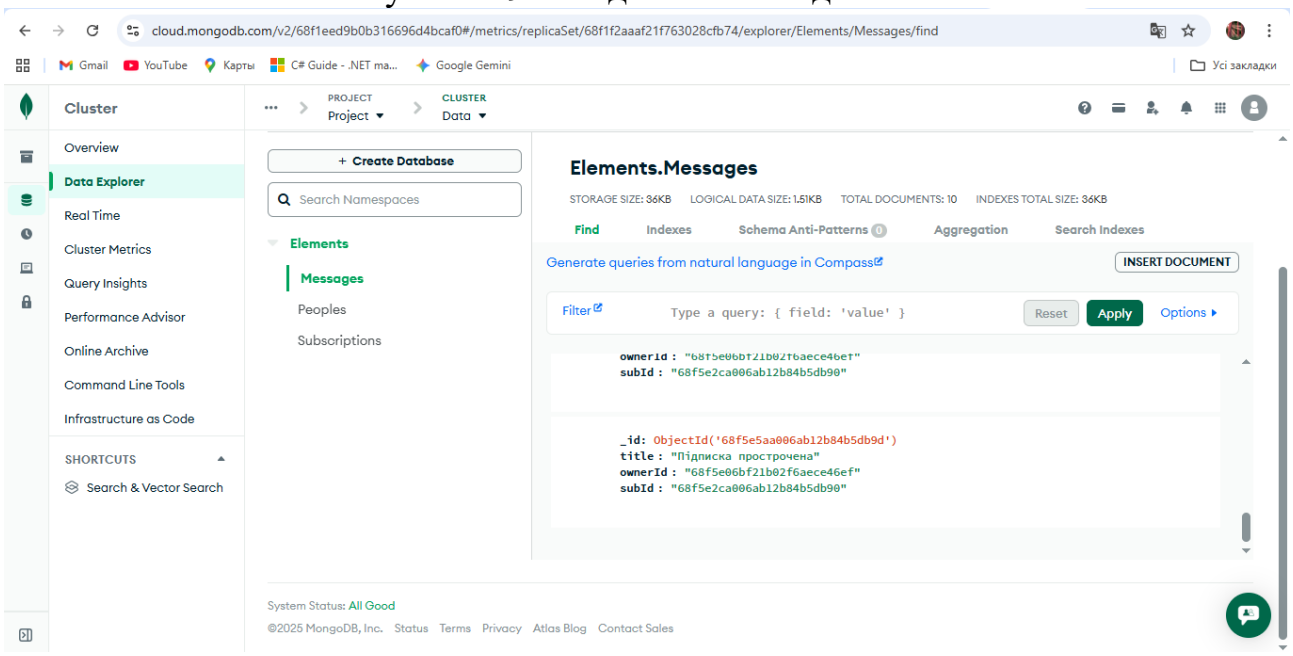


Рисунок 120 – Вміст колекції після оновлення

Завдання 5

В проект із попереднього завдання було додано нові моделі, автомаминг, налаштовано використання репозиторіїв, сервісів, автомаперів за допомогою DI.

Приклади використання мапінгу в програмі:

PeopleController.cs:

```
using AutoMapper;
using LW4_task_3.Interfaces;
using LW4_task_3.Models.Entities;
using LW4_task_3.Models.Request;
using LW4_task_3.Models.Response;
using LW4_task_3.Validators;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.Text.Json;
 namespace LW4_task_3.Controllers
{
   [Route("[controller]")]
   [ApiController]
   public class PeopleController : ControllerBase
   {
      private readonly IPeopleService _peopleService;
     // Створення змінної для подальшого вкористання за допомогою DI
      private readonly IMapper _mapper;
       public PeopleController(IPeopleService peopleService, IMapper mapp
er)
      {
         _peopleService = peopleService;
         _mapper = mapper;
      }
       [HttpGet]
      public async Task<ActionResult<IEnumerable<PeopleResponse>>> G
et(
         [FromQuery] string? name, [FromQuery] string? email)
```

```
        {
            try
            {
                var peoplesItems = await _peopleService.GetPeoplesItemsAsync(n
ame, email);
        // Перетворення DTO  в Entity
                var peoples = _mapper.Map<IEnumerable<PeopleResponse>>(pe
oplesItems);
                 return Ok(peoples);
            }
            catch (KeyNotFoundException kex)
            {
                return NotFound(kex.Message);
            }
        }
            [HttpGet("{id}")]
        public async Task<ActionResult<PeopleResponse>> GetById(string id
)
        {
            try
            {
                ValidElement.ValidId(id);
                var peopleItem = await _peopleService.GetByIdAsync(id);
        // Перетворення DTO  в Entity
                var people = _mapper.Map<PeopleResponse>(peopleItem);
                return Ok(people);
            }
            catch (KeyNotFoundException kex)
            {
                return NotFound(kex.Message);
```

```
            }
            catch(ArgumentException aex)
            {
                return BadRequest(aex.Message);
            }
        }
        [HttpPost]
        public async Task<IActionResult> Create(PeopleRequest item)
        {
    // Перетворення DTO  в Entity
            var element = _mapper.Map<PeopleItem>(item);
            await _peopleService.CreateAsync(element);
     // Перетворення Entity в DTO
            var resp = _mapper.Map<PeopleResponse>(element);
            return CreatedAtAction(nameof(GetById), new { Id = resp.Id }, resp
);
        }
        [HttpPut("{id}")]
        public async Task<IActionResult> Update(string id, PeopleRequest ite
m)
        {
            try
            {
                ValidElement.ValidId(id);
                // Перетворення DTO  в Entity
                var element = _mapper.Map<PeopleItem>(item);
                await _peopleService.UpdateAsync(id, element);
                return NoContent();
            }
            catch (KeyNotFoundException kex)
```

```
            {
                return NotFound(kex.Message);
            }
            catch (ArgumentException aex)
            {
                return BadRequest(aex.Message);
            }
        }
        [HttpPatch("{id}")]
        public async Task<IActionResult> UpdatePart(string id, JsonElement e
lement)
        {
            try
            {
                ValidElement.ValidId(id);
                 var upPeople = await _peopleService.GetByIdAsync(id);
    // Перетворення Entity в DTO
                var temp = _mapper.Map<PeopleRequest>(upPeople);
                 if (element.TryGetProperty("name", out var name))
                    temp.Name = name.GetString();
                if (element.TryGetProperty("email", out var email))
                    temp.Email = email.GetString();
                 if (!TryValidateModel(temp))
                    return BadRequest(ModelState);
            //Маппінг DTO в Entity
                _mapper.Map(temp, upPeople);
                await _peopleService.UpdateAsync(id, upPeople);
                return NoContent();
            }
            catch (KeyNotFoundException kex)
```

```
            {
                return NotFound(kex.Message);
            }
            catch(ArgumentException aex)
            {
                return BadRequest(aex.Message);
            }
        }
    }
}
```

Створено профілі для маппінгу кожного типу моделей:

```
namespace LW4_task_3.Mapping
{
    public class MessageProfile: Profile
    {
        public MessageProfile()
        {
            CreateMap<MessageRequest, MessageItem>().ReverseMap();
            CreateMap<MessageItem, MessageResponse>();
        }
    }
}
namespace LW4_task_3.Mapping
{
    public class PeopleProfile: Profile
    {
        public PeopleProfile()
        {
            CreateMap<PeopleRequest, PeopleItem>().ReverseMap();
            CreateMap<PeopleItem, PeopleResponse>();
```

```
            }
        }
    }
    namespace LW4_task_3.Mapping
    {
        public class SubProfile: Profile
        {
            public SubProfile()
            {
                CreateMap<SubscriptionRequest, SubscriptionItem>()
                    .ForMember(d => d.Status, o => o.MapFrom(s => Enum.Parse<Su
bStatus>(s.Status,true)));
                CreateMap<SubscriptionItem, SubscriptionRequest>()
                    .ForMember(d => d.Status, o => o.MapFrom(s => s.Status.ToStrin
g()));
                CreateMap<SubscriptionItem, SubscriptionResponse>()
                    .ForMember(d => d.Status, o => o.MapFrom(s => s.Status.ToStrin
g()));
            }
        }
    }
```

Додано до конструкторів конролерів, сервісів прив’язку до інтерфейсів, а не до об’єктів.

Виконано запити для перевірки роботи DI та Mapping

MessageController:

```
private readonly IMessageService _messageService;
            private readonly IMapper _mapper;
             public MessageController(IMessageService messageService, IMapp
er mapper)
            {
```

```
            _messageService = messageService;
            _mapper = mapper;
        }
```

SubController:

```
private readonly ISubService _subService;
    private readonly IMapper _mapper;
    public SubController(ISubService subService, IMapper mapper)
    {
        _subService = subService;
        _mapper = mapper;
    }
```

PeopleController:

```
private readonly IPeopleService _peopleService;
    private readonly IMapper _mapper;
    public PeopleController(IPeopleService peopleService, IMapper mappe
r)
    {
        _peopleService = peopleService;
        _mapper = mapper;
    }
```

MessageService:

```
private readonly IMessageRepository _messageRepository;
    private readonly ISubRepository _subRepository;
    private readonly IPeopleRepository _peopleRepository;
    public MessageService(IMessageRepository messageRepository, ISub
Repository subRepository, IPeopleRepository peopleRepository)
    {
        _messageRepository = messageRepository;
        _subRepository = subRepository;
        _peopleRepository = peopleRepository;
```

```
        }
```

SubSrvice:

```
private readonly ISubRepository _subRepository;
        private readonly IPeopleRepository _peopleRepository;
        public SubService(ISubRepository subRepository, IPeopleRepository p
eopleRepository)
        {
            _subRepository = subRepository;
            _peopleRepository = peopleRepository;
        }
```

PeopleService:

```
private readonly IPeopleRepository _peopleRepository;
        public PeopleService(IPeopleRepository peopleRepository)
        {
            _peopleRepository = peopleRepository;
        }
```

В Program.cs додано прив’язку між інтерфейсом та класом для реалізації DI:

```
using FluentValidation;
using FluentValidation.AspNetCore;
using LW4_task_3.Models;
using LW4_task_3.Validators;
using LW4_task_3.Services;
using LW4_task_3.Interfaces;
using LW4_task_3.InterfacesRepository;
using LW4_task_3.Repositories;
using LW4_task_3.Mapping;
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
builder.Services.AddControllers().AddFluentValidation();
```

```
builder.Services.AddScoped<IPeopleRepository,PeopleRepository>();
builder.Services.AddScoped<ISubRepository,SubRepository>();
builder.Services.AddScoped<IMessageRepository,MessageRepository>();
builder.Services.AddScoped<IPeopleService, PeopleService>();
builder.Services.AddScoped<ISubService, SubService>();
builder.Services.AddScoped<IMessageService, MessageService>();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(oprions =>
{
    var xmlFile = $"{System.Reflection.Assembly.GetExecutingAssembly().GetName().Name}.xml";
    var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
    oprions.IncludeXmlComments(xmlPath);
});
builder.Services.AddValidatorsFromAssemblyContaining<PeopleValidator>();
builder.Services.AddValidatorsFromAssemblyContaining<SubValidator>();
builder.Services.AddValidatorsFromAssemblyContaining<MessageValidator>();
builder.Services.AddAutoMapper(typeof(PeopleProfile));
builder.Services.AddAutoMapper(typeof(SubProfile));
builder.Services.AddAutoMapper(typeof(MessageProfile));
var app = builder.Build();
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
```

```
}
app.UseAuthorization();
app.MapControllers();
app.Run();
```

В програмі DI працює за допомогою заміни створення об’єктів у конструкторах класів на прив’язку до інтерфейсів, які підставляються автоматично за допомогою реєстрації в Program.cs. AutoMapper також використовується за допомогою DI, клас має об’єкт типу IMapper за допомогою якого робить маппінг, зареєстрований в класові – профілі.

Здійснено перевірку роботи CRUD – операцій після оновлення проекту.

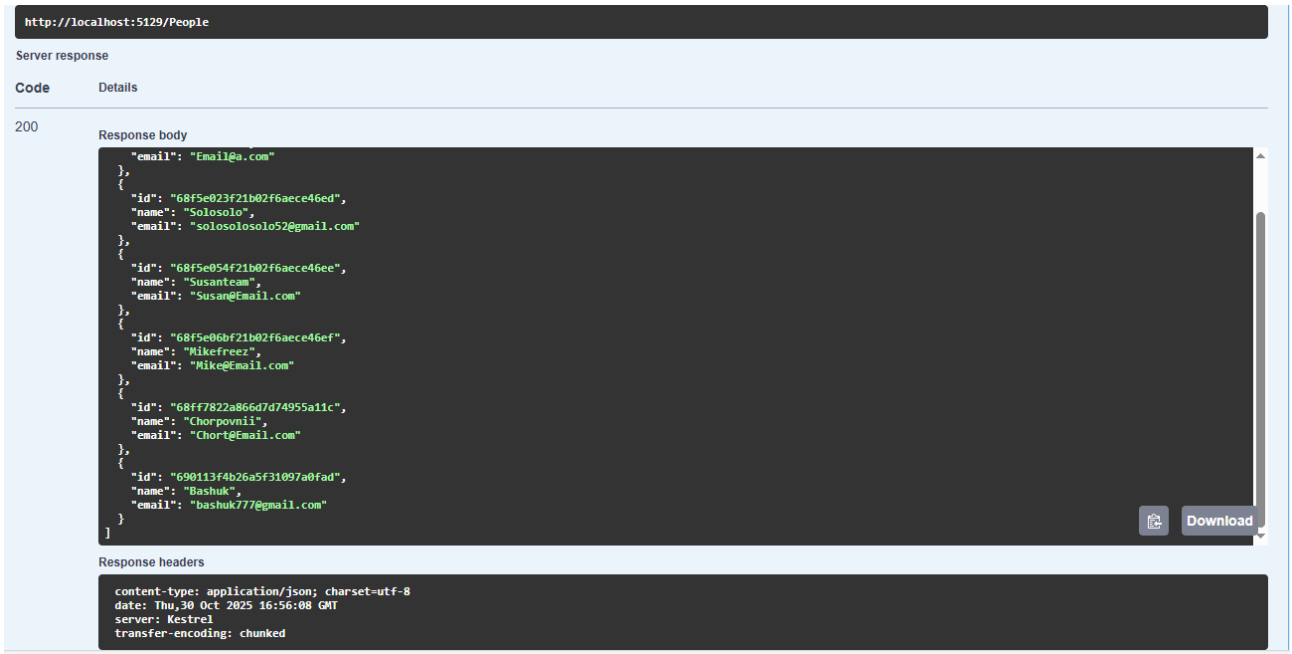


Рисунок 121 – Отримано список всіх користувачів

Рисунок 122 – Отримано користувачів за вказаним фільтром

Рисунок 123 – Отримано користувача за Id

Рисунок 124 – Створено нового користувача

Рисунок 125 – Оновлено повністю користувача

Рисунок 126 – Здійснено часткове оновлення

Рисунок 127 – Видалено користувача

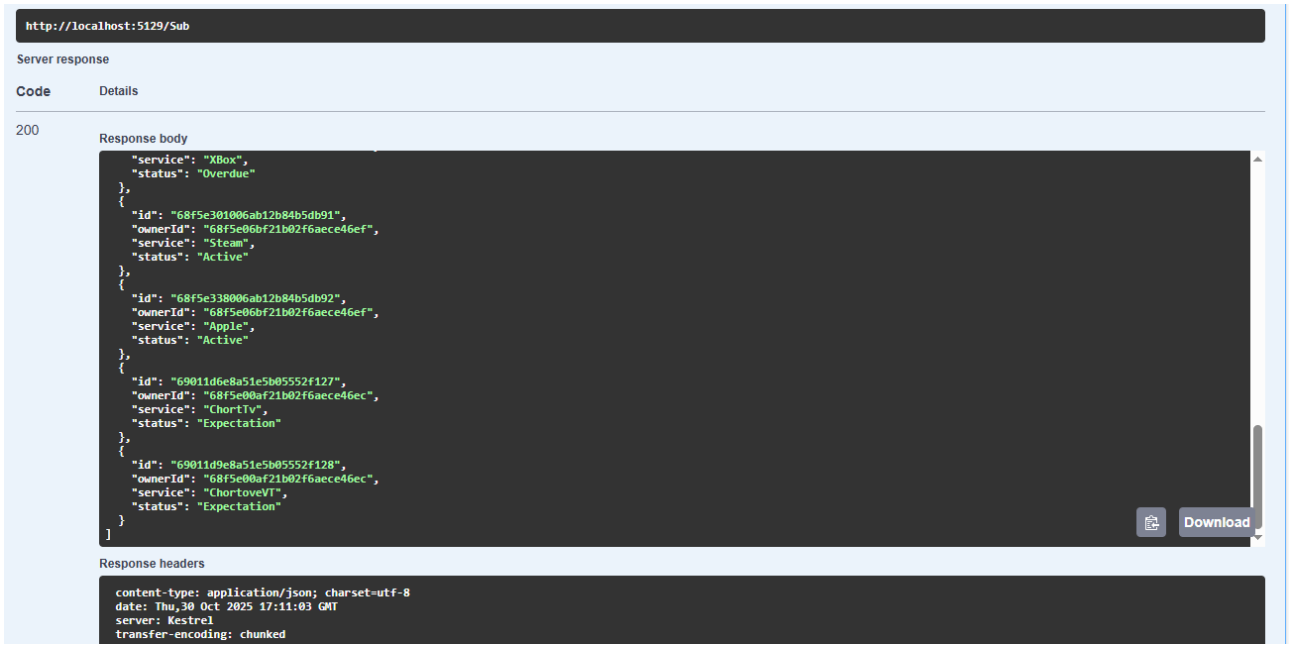


Рисунок 128 – Отримано список всіх підписок

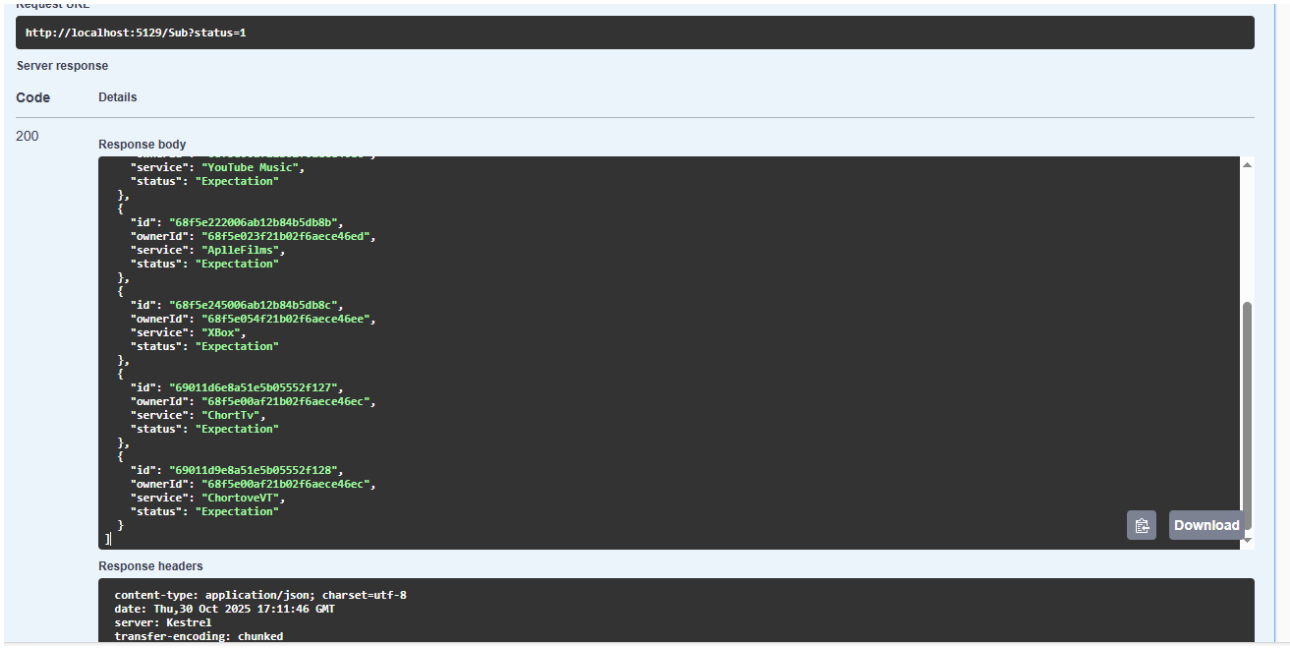


Рисунок 129 – Отримано список підписок, що очікують на активацію

Рисунок 130 – Отримано підписку за Id

Рисунок 130 – Створено нову підписку для користувача

Рисунок 131 – Здійснено повне оновлення підписки

Рисунок 132 – Здійснено часткове оновлення підписки

Рисунок 133 – Видалено підписку

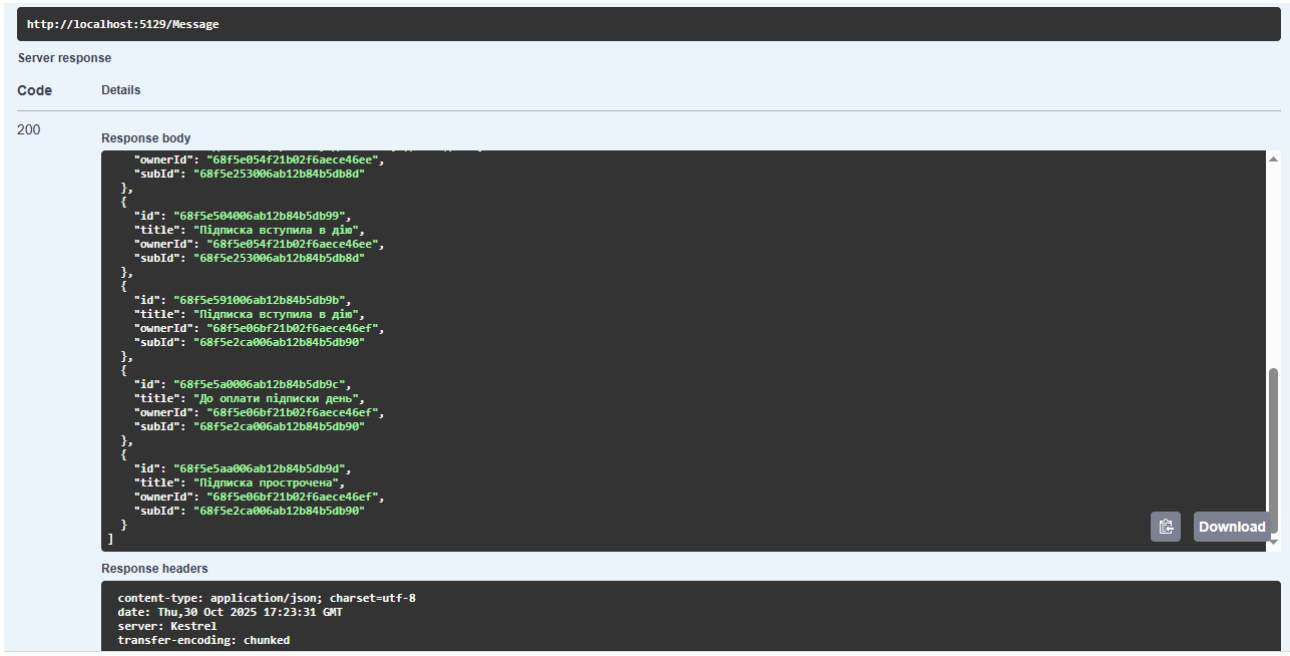


Рисунок 134 – Отримано список всіх повідомлень

Рисунок 135 – Отримано повідомлення за Id

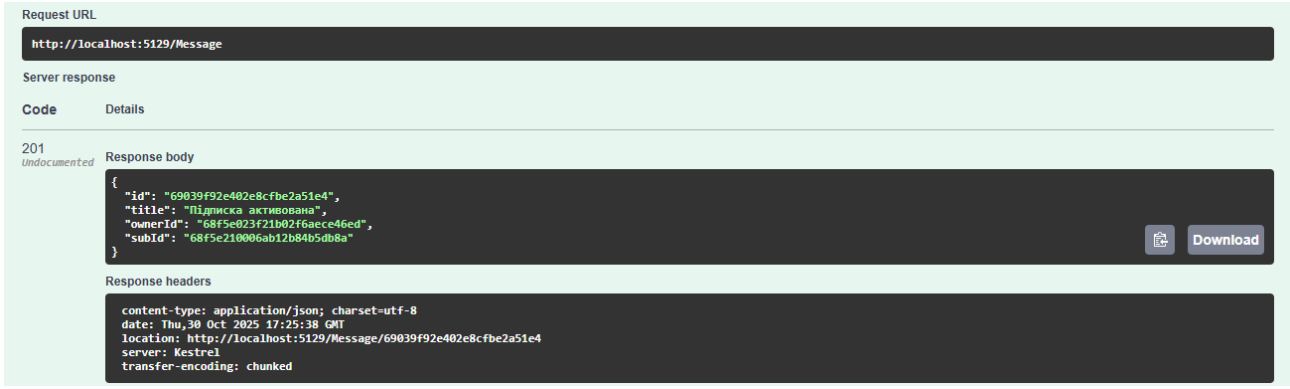


Рисунок 136 – Створено нове повідомлення

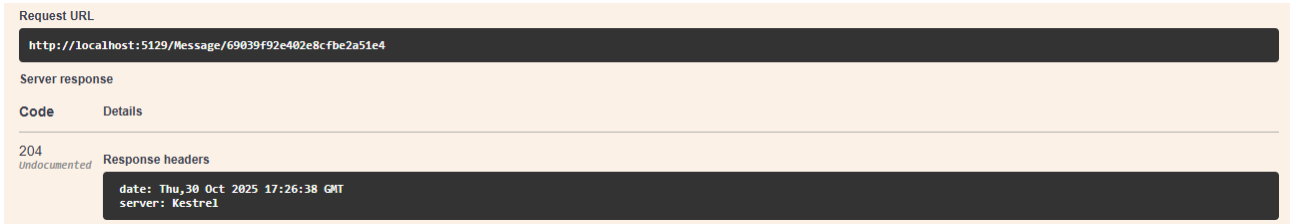


Рисунок 137 – Повністю оновлено повідомлення

Рисунок 138 – Частково оновлено повідомлення

Рисунок 139 – Видалено повідомлення

Завдання 6

Встановлено необхідні пакети та налаштовано appsetting.json:

```
{
 "Logging": {
  "LogLevel": {
   "Default": "Information",
   "Microsoft.AspNetCore": "Warning"
  }
 },
 "AllowedHosts": "*",
 "JwtSettings": {
  "SecretKey": "My_Super_Secret_Key_2w22w3_4w52q2728=93+_some",
  "Issuer": "http://localhost:5129",
  "Audience": "http://localhost:5129",
  "AccessTokenExpirationMinutes": 60,
  "RefreshTokenExpirationDays":  30
 }
}
```

Створено модель JwtSetting та сутність для роботи із БД, і DTO моделі для логіну та реєстрації:

```
namespace LW4_task_3.Models
{
   public class JwtSettings
   {
      public string SecretKey { get; set; }
      public string Issuer { get; set; }
      public string Audience { get; set; }
      public int AccessTokenExpirationMinutes { get; set; }
      public int RefreshTokenExpirationDays { get; set; }
   }
}
using LW4_task_3.Enums;
using Microsoft.AspNetCore.Identity;
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;
namespace LW4_task_3.Models.Entities
{
   public class UserItem
   {
      [BsonId]
      [BsonRepresentation(BsonType.ObjectId)]
      public string? Id { get; set; }
      [BsonElement("username")]
      public string? UserName { get; set; }
      [BsonElement("email")]
      public string? Email { get; set; }
       [BsonElement("paswordHash")]
      public string? Password { get; set; }
```

```
        [BsonElement("refreshToken")]
        public string? RefreshToken { get; set;}
        [BsonElement("refreshTokenExpireTime")]
        public DateTime? RefreshTokenExpireTime { get; set; }
        [BsonElement("role")]
        public UserRole Role { get; set; } = UserRole.User;
    }
}
namespace LW4_task_3.Models.Request
{
    public class LoginModel
    {
        public string Email { get; set; }
        public string Password { get; set; }
    }
}
namespace LW4_task_3.Models.Request
{
    public class UserRegistration
    {
        public string? UserName { get; set; }
        public string? Email { get; set; }
        public string? Password { get; set; }
    }
}
```

Створено сервіс для хешування та сервіс для генерування accesstoken, refreshtoken:

```
using LW4_task_3.Interface.Interfaces;
using System.Security.Cryptography;
using System.Text;
```

```
namespace LW4_task_3.Services
{
    public class PaswordHasher : IPasswordHasher
    {
        public string Hash(string password)
        {
            using var sha256 = SHA256.Create();
            var bytes = Encoding.UTF8.GetBytes(password);
            var hash = sha256.ComputeHash(bytes);
            return Convert.ToBase64String(hash);
        }
        public bool Verify(string password, string hash)
        {
            var hashPas = Hash(password);
            return hashPas == hash;
        }
    }
}
using LW4_task_3.Models;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using System.Security.Claims;
using System.IdentityModel.Tokens.Jwt;
using LW4_task_3.Models.Entities;
using System.Security.Cryptography;
using LW4_task_3.Enums;
 namespace LW4_task_3.Services
{
    public class JwtTokenGenerator
```

```
{
    private readonly JwtSettings _jwtSettings;
    public JwtTokenGenerator(IOptions<JwtSettings> options)
    {
        _jwtSettings = options.Value;
    }
    public string GenerateToken(UserItem user)
    {
        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_j
wtSettings.SecretKey));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacS
ha256);
        var claims = new List<Claim>
        {
            new Claim(JwtRegisteredClaimNames.Sub, user.Id),
            new Claim(JwtRegisteredClaimNames.Email,user.Email),
            new Claim(JwtRegisteredClaimNames.Name, user.UserName)
        };
        foreach(UserRole role in Enum.GetValues(typeof(UserRole)))
        {
            if (role == UserRole.None)
                continue;
            if (user.Role.HasFlag(role))
                claims.Add(new Claim(ClaimTypes.Role, role.ToString()));
        }
        claims.Add(new Claim("rolesValue", ((int)user.Role).ToString()));
        var token = new JwtSecurityToken(
            issuer: _jwtSettings.Issuer,
            audience: _jwtSettings.Audience,
            claims: claims,
```

```
                expires: DateTime.UtcNow.AddMinutes(_jwtSettings.AccessToke
nExpirationMinutes),
                signingCredentials: creds
            );
                return new JwtSecurityTokenHandler().WriteToken(token);
        }
        public string GeneretaRefreshToken()
        {
            var randomNumbers = new byte[32];
            using var random = RandomNumberGenerator.Create();
            random.GetBytes(randomNumbers);
            return Convert.ToBase64String(randomNumbers);
        }
        public ClaimsPrincipal? GetClaimsPrincipalFromExpiredToken(string t
oken)
        {
            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_j
wtSettings.SecretKey));
            var tokenValidationParametr = new TokenValidationParameters
            {
                ValidateAudience = true,
                ValidateIssuer = true,
                ValidateIssuerSigningKey = true,
                ValidIssuer = _jwtSettings.Issuer,
                ValidAudience = _jwtSettings.Audience,
                IssuerSigningKey = key,
                ValidateLifetime = false
            };

            var tokenHandler = new JwtSecurityTokenHandler();
```

```
                try
                {
                    var principal = tokenHandler.ValidateToken(token, tokenValidatio
nParametr, out var securityToken);
                    if (securityToken is not JwtSecurityToken jwtSecurityToken ||
                        !jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.Hma
cSha256, StringComparison.InvariantCultureIgnoreCase))
                        return null;

                    return principal;
                }
                catch
                {
                    return null;
                }
            }
        }
    }
```

Створено сервіс та репозиторій для зберігання та роботи із сутностями в MongoDB:

```
    using LW4_task_3.Clients;
    using LW4_task_3.Interface.InterfacesRepository;
    using LW4_task_3.Models.Entities;
    using MongoDB.Driver;
    namespace LW4_task_3.Repositories
    {
        public class UserRepository : IUserRepository
        {
            IMongoCollection<UserItem> _users;
```

```
            public UserRepository()
            {
                _users = MongoDBClient.Instance.GetCollection<UserItem>("Users
");
            }
            public async Task CreateAsync(UserItem element) =>
                await _users.InsertOneAsync(element);
            public async Task DeleteAsync(string id) =>
                await _users.DeleteOneAsync(x => x.Id == id);
            public async Task<List<UserItem>> GetAllAsync() =>
                await _users.Find(x => true).ToListAsync();
            public async Task<UserItem> GetByEmailAsync(string email) =>
                await _users.Find(x => x.Email == email).FirstOrDefaultAsync();
            public async Task<UserItem> GetByIdAsync(string id) =>
                await _users.Find(x => x.Id == id).FirstOrDefaultAsync();
            public async Task<bool> IsExist(string id) =>
                await _users.Find(x => x.Id == id).AnyAsync();
            public async Task UpdateAsync(string id, UserItem element) =>
                await _users.ReplaceOneAsync(x => x.Id == id, element);
        }
    }
```

Створено TestCotroller для роботи перевірки роботи автентифікації:

```
using LW4_task_3.Enums;
using LW4_task_3.Interface.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.SignalR;
using System.Security.Claims;
```

```
namespace LW4_task_3.Controllers
{
    [Route("[controller]")]
    [ApiController]
    public class TestController : ControllerBase
    {
        IUserService _userService;
        public TestController(IUserService userService)
        {
            _userService = userService;
        }
        [Authorize]
        [HttpGet("private")]
        public async Task<IActionResult> GetIdEmail()
        {
            var email = User?.Claims.FirstOrDefault(c => c.Type == ClaimType
s.Email)?.Value;
            var user = await _userService.GetByEmailAsync(email);
            return Ok(new
            {
                UserId = user.Id,
                Email = email,
            });
        }
    }
}
```

Створено контролер для автентифікації, логіну та оновлення токену:

```
using AutoMapper;
using LW4_task_3.Interface.Interfaces;
using LW4_task_3.Models.Entities;
```

```
using LW4_task_3.Models.Request;
using LW4_task_3.Services;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration.UserSecrets;
using System.Security.Claims;
namespace LW4_task_3.Controllers
{
   [Route("[controller]")]
   [ApiController]
   public class AuthController : ControllerBase
   {
      private readonly IUserService _userService;
      private readonly IPasswordHasher _passwordHasher;
      private readonly JwtTokenGenerator _jwtGenerator;
      private readonly IMapper _mapper;
      public AuthController(IUserService userService,
         IPasswordHasher passwordHasher,
         JwtTokenGenerator JwtTokenGenerator, IMapper mapper)
      {
         _userService = userService;
         _passwordHasher = passwordHasher;
         _jwtGenerator = JwtTokenGenerator;
         _mapper = mapper;
      }
        [HttpPost("register")]
      public async Task<IActionResult> Reg(UserRegistration user)
      {
         var userItem = _mapper.Map<UserItem>(user);
         try
```

```
        {
            await _userService.CreateAsync(userItem);
            return Ok("Користувача зареєстровано");
        }
        catch (ArgumentException aex)
        {
            return BadRequest(aex.Message);
        }
    }
    [HttpPost("login")]
    public async Task<IActionResult> LogIn(LoginModel loginModel)
    {
        string message = "Невірна пошта чи пароль";
        UserItem user;
        try
        {
            user = await _userService.GetByEmailAsync(loginModel.Email);
        }
        catch (KeyNotFoundException)
        {
            return Unauthorized(message);
        }
        if(!_passwordHasher.Verify(loginModel.Password,user.Password))
            return BadRequest(message);
        var token = _jwtGenerator.GenerateToken(user);
        user.RefreshToken= _jwtGenerator.GeneretaRefreshToken();
        user.RefreshTokenExpireTime = DateTime.UtcNow.AddDays(7);
        await _userService.UpdateAsync(user.Id, user);
        return Ok(new
        {
```

```
                Token = token,
                TokenExpireTime = DateTime.UtcNow.AddMinutes(60),
                ResreshToken = user.RefreshToken,
                ResreshTokenExpireTime = user.RefreshTokenExpireTime
            });
        }
        [HttpPost("refresh")]
        public async Task<IActionResult> Refresh([FromHeader(Name = "Aut
horization")] string authHead, string refreshToken)
        {
            if (string.IsNullOrEmpty(authHead) || !authHead.StartsWith("Bearer
"))
                return BadRequest("Немає або неправильний заголовок");
            var token = authHead.Substring("Bearer ".Length);
            var princial = _jwtGenerator.GetClaimsPrincipalFromExpiredToken(
token);
            if (princial is null)
                return Unauthorized("Невірний token");
            var id = princial?.Claims.FirstOrDefault(c => c.Type == ClaimTypes
.NameIdentifier)?.Value;
            UserItem user;
            try
            {
                user = await _userService.GetByIdAsync(id);
            }
            catch(KeyNotFoundException kex)
            {
                return NotFound(kex.Message);
            }
            if (user.RefreshToken != refreshToken
```

```
            || user.RefreshTokenExpireTime <= DateTime.UtcNow)
            return Unauthorized("Невірний RefreshToken");
        var newToken = _jwtGenerator.GenerateToken(user);
        var newrefreshToken = _jwtGenerator.GeneretaRefreshToken();
        user.RefreshToken = newrefreshToken;
        user.RefreshTokenExpireTime = DateTime.UtcNow.AddDays(7);
        await _userService.UpdateAsync(id, user);
        return Ok(new
        {
            Token = token,
            TokenExpireTime = DateTime.UtcNow.AddMinutes(60),
            ResreshToken = user.RefreshToken,
            ResreshTokenExpireTime = user.RefreshTokenExpireTime
        });
    }
  }
}
```

Створено контролер для роботи із User:

```
using AutoMapper;
using LW4_task_3.Enums;
using LW4_task_3.Interface.Interfaces;
using LW4_task_3.Models.Entities;
using LW4_task_3.Models.Request;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Protocols.OpenIdConnect;
using System.Data;
using ZstdSharp.Unsafe;
namespace LW4_task_3.Controllers.A_A
```

```
    {
        [Route("[controller]")]
        [ApiController]
        [Authorize(Roles = nameof(UserRole.Admin))]
        public class UserController : ControllerBase
        {
            private readonly IUserService _userService;
            private readonly IMapper _mapper;
            private readonly IPasswordHasher _passwordHasher;
            public UserController(IUserService userService,IMapper mapper, IPass
wordHasher passwordHasher)
            {
                _userService = userService;
                _mapper = mapper;
                _passwordHasher = passwordHasher;
            }
            [Authorize(Roles = $"{nameof(UserRole.Admin)},{nameof(UserRole.
Manager)}")]
            [HttpGet]
            public async Task<IActionResult> GetAllUser()
            {
                try
                {
                    var users = await _userService.GetAllAsync();
                    return Ok(users);
                }
                catch (KeyNotFoundException kex)
                {
                    return NotFound(kex.Message);
                }
```

```
        }
        [Authorize(Roles = $"{nameof(UserRole.Admin)},{nameof(UserRole.
Manager)}")]
        [HttpGet("{id}")]
        public async Task<IActionResult> GetOneUser(string id)
        {
            try
            {
                var user = await _userService.GetByIdAsync(id);
                return Ok(user);
            }
            catch(KeyNotFoundException kex)
            {
                return NotFound(kex.Message);
            }
        }
        [HttpPut("{id}")]
        public async Task<IActionResult> UpdateUser(string id, UserRegistrat
ion userRegistration)
        {
            try
            {
                var updateUser = await _userService.GetByIdAsync(id);
                var emailUser = await _userService.GetByEmailAsync(userRegist
ration.Email);
                if (emailUser is not null && emailUser.Id != updateUser.Id)
                    return BadRequest("Ця пошта вже використовується");
                userRegistration.Password = _passwordHasher.Hash(userRegistrat
ion.Password);
                _mapper.Map(userRegistration, updateUser);
```

```
            await _userService.UpdateAsync(id, updateUser);
            return NoContent();
        }
        catch (KeyNotFoundException kex)
        {
            return NotFound(kex.Message);
        }
    }
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteUser(string id)
    {
        try
        {
            await _userService.DeleteAsync(id);
            return NoContent();
        }
        catch (KeyNotFoundException kex)
        {
            return NotFound(kex.Message);
        }
    }
    [HttpPost("setRole")]
    public async Task<IActionResult> SetRole(string id, UserRole role)
    {
        try
        {
            var user = await _userService.GetByIdAsync(id);
            user.Role = role;
            await _userService.UpdateAsync(id, user);
```

```
                return Ok(new { UserName = user.UserName, Email = user.Email
, Role = (int)user.Role });
            }
            catch (KeyNotFoundException kex)
            {
                return NotFound(kex.Message);
            }
        }
        [HttpPost("addRole")]
        public async Task<IActionResult> AddRole(string id, UserRole role)
        {
            try
            {
                var user = await _userService.GetByIdAsync(id);
                user.Role |= role;
                await _userService.UpdateAsync(id, user);
                return Ok(new { UserName = user.UserName, Email = user.Email
, Role = (int)user.Role });
            }
            catch (KeyNotFoundException kex)
            {
                return NotFound(kex.Message);
            }
        }
        [HttpPost("removeRole")]
        public async Task<IActionResult> RemoveRole(string id, UserRole rol
e)
        {
            try
            {
```

```
                var user = await _userService.GetByIdAsync(id);
                 user.Role &= ~role;
                 await _userService.UpdateAsync(id, user);
                return Ok(new { UserName = user.UserName, Email = user.Email
, Role = (int)user.Role });
            }
            catch (KeyNotFoundException kex)
            {
                return NotFound(kex.Message);
            }
        }
    }
}
```

Протестовано роботу програми з різними ролями:

Рисунок 140 – Реєстрація нового користувача

Рисунок 141 – Невірно введений пароль

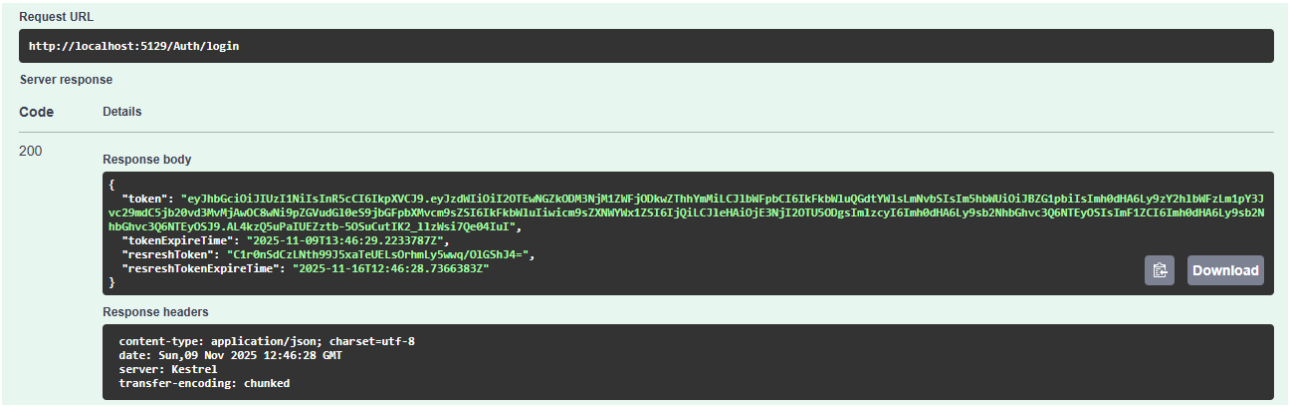


Рисунок 142 – Вхід із роллю Admin

Рисунок 143 – Запит на TestController

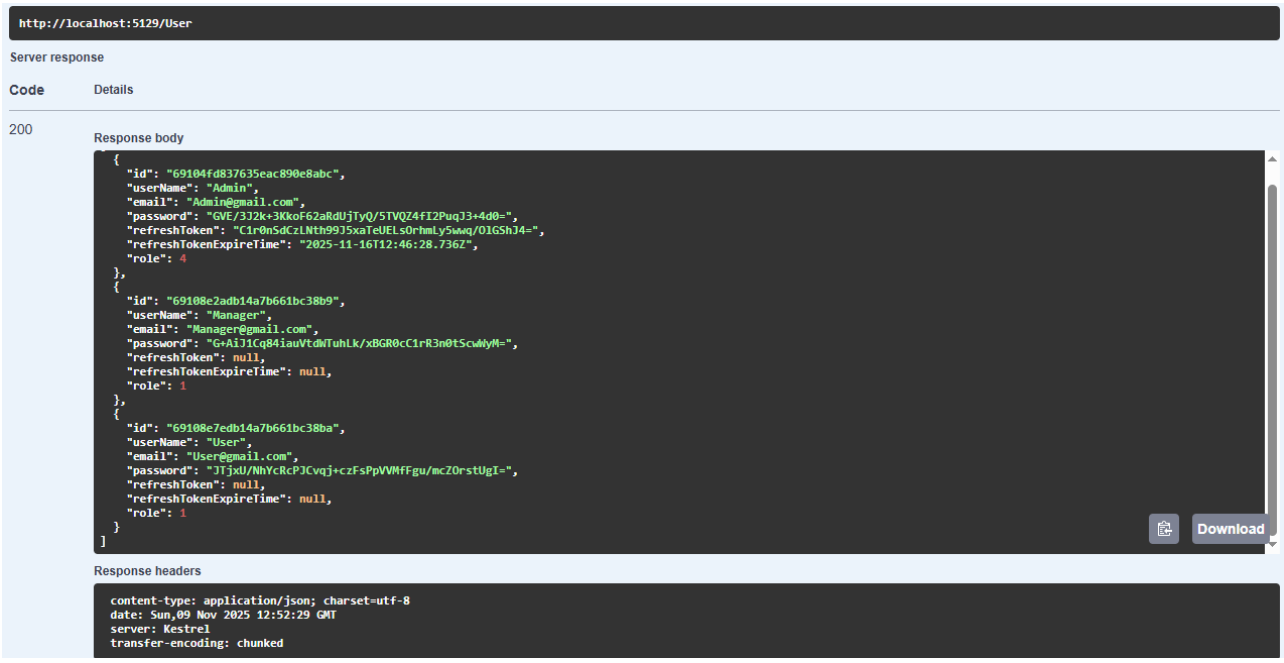


Рисунок 144 – Переглянути список всіх User-ів

Рисунок 145 – Отримано інформацію про конкретного User-а

Рисунок 146 – Вкористано /setRole для зміни ролі менеджера

Рисунок 147 – Додано роль менеджера через /addRole

Рисунок 148 – Видалено роль User через /removeRole

Рисунок 149 – Оновлено ім’я для Manager

Рисунок 150 – Результат оновлення

Рисунок 151 – Видалено оновленого користувача

Рисунок 152 – Спроба отримання видаленогокористувача

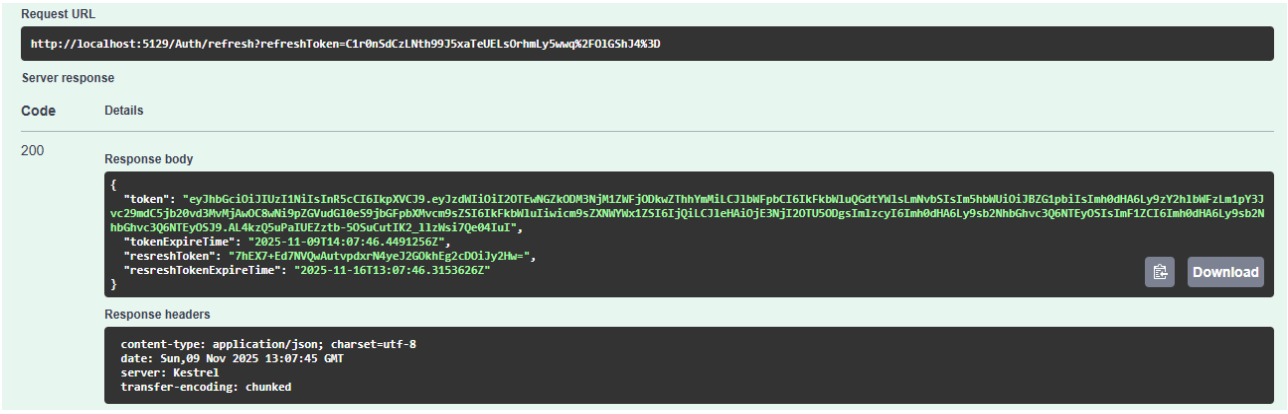


Рисунок 154 – Оновлення токену

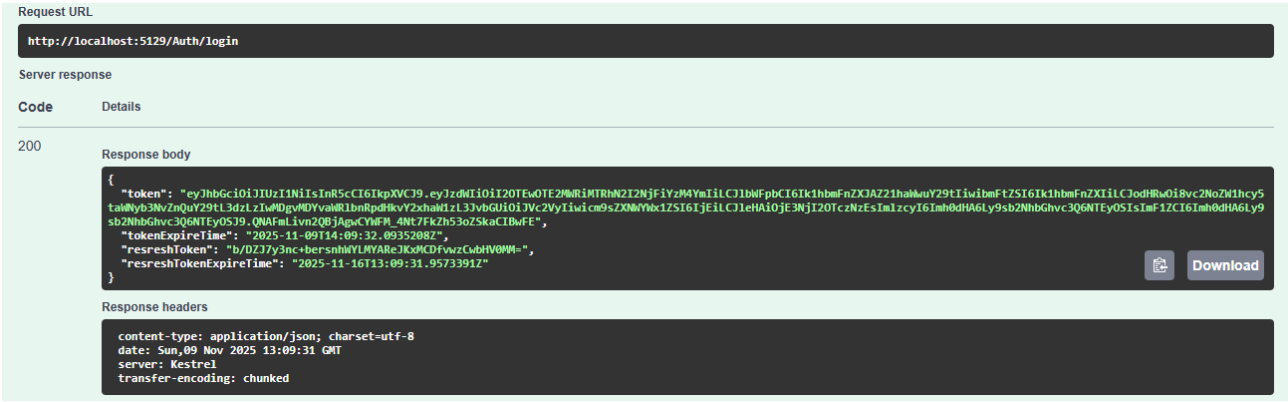


Рисунок 155 – Вхід з роллю Manager

Рисунок 156 – Виконання запиту за допомогою TestController

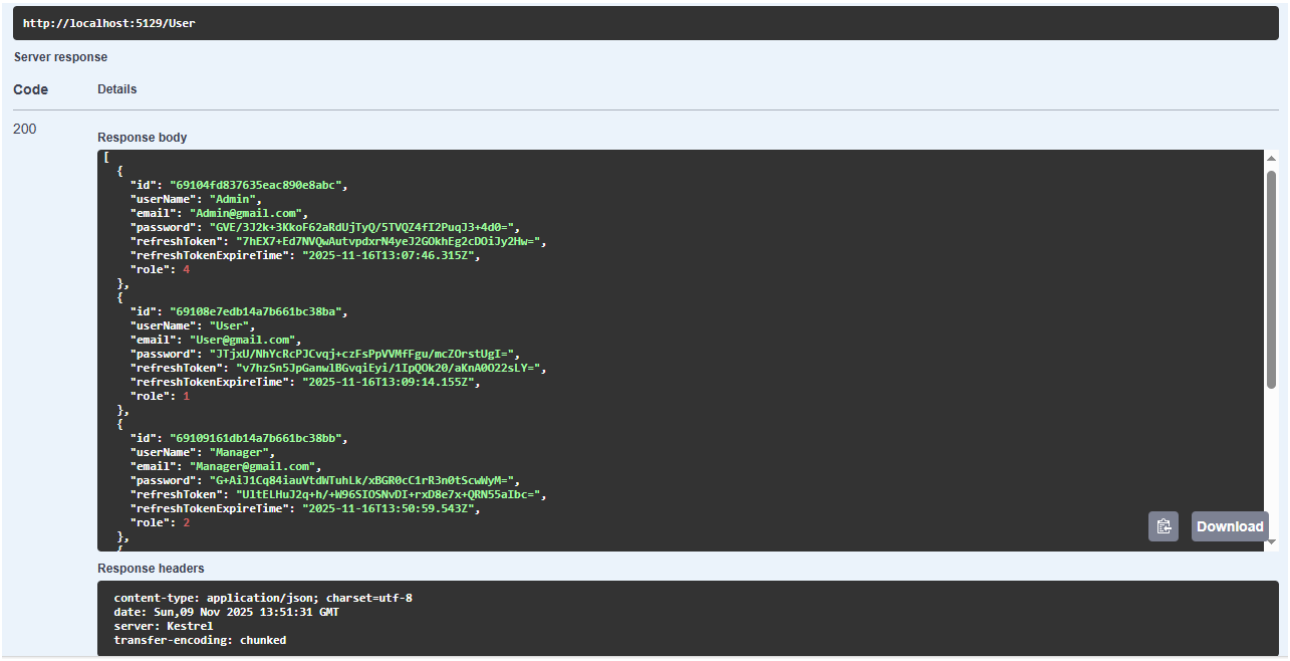


Рисунок 157 – Отримано список всіх User-ів

Рисунок 158 – Отримано конкретного User-а

Рисунок 159 – Спроба видалити User-а із роллю Manager

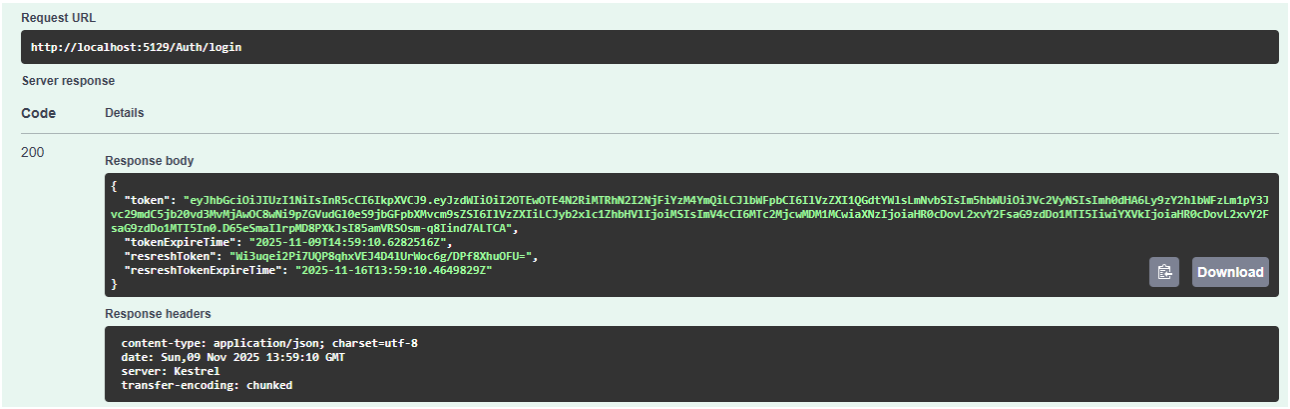


Рисунок 160 – Здійснено вхід із роллю User

Рисунок 161 – Виконано запит на TestController

Рисунок 162 – Спроба отримати список User-ів з невідповідною роллю

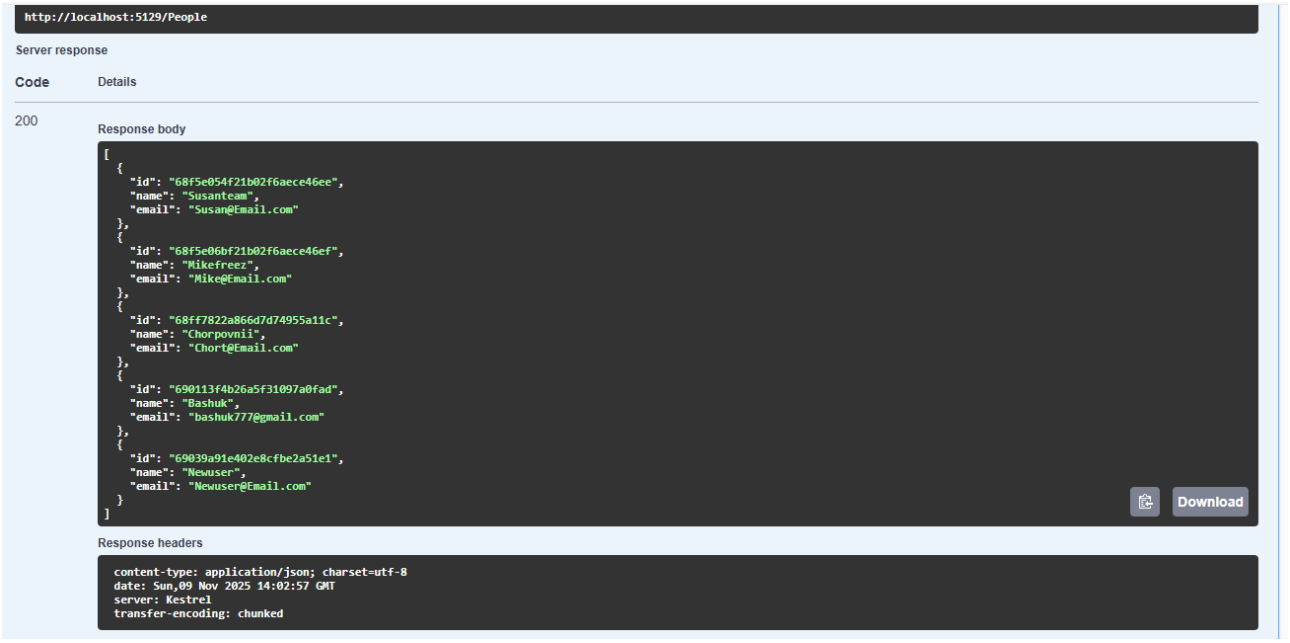


Рисунок 163 – Здійснено запит для отримання списку користувачів підписок

Рисунок 164 – Отримано конкретного користувача підписки

Рисунок 165 – Споба видалення користувача підписки

Висновок: на даній лабораторній роботі я реалізував CRUD – операції у C# ASP.Net Web – API, Minimal API та навчився працювати із ними, також здобув практичні навики по створенню RESTful API для взаємодії із клієнтом. Також засвоїв як реалізувати та працювати з тришаровою архітектурою, як використовувати DI. Також я засвоїв роботу із автентифікацією та авторизацією