Documentacija ASP

Student: Nemanja Ranisavljevic 86/16

Predmet ASP.NET

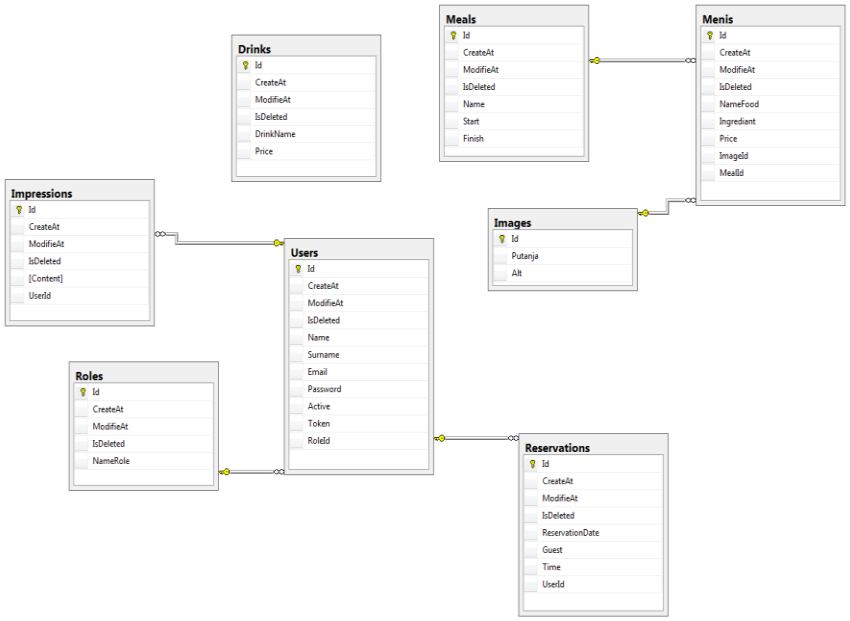
Uvod

Tema projekta je restoran koji daje mogucnost korisnicima da imaju uvid u jela restorana po dnevnim obrocima, sliku jela, sastojke jela I cenu, kartu pica koju nudi restoran, kao I mogucnost uvida u utiske zadovoljnih korisnika nase usluge. Korisnici koji se registruju dobijaju email dobrodoslice na sajt, nakon logovanja imaju mogucnost da rezervisu svoje mesto u restoranu unoseci datum, vreme, broj osoba. Nakon nase posete mogu da unsesu I svoj utisak koji kasnije vide ostali posetioci.Sve te mogucnosti su obezbedjene putem API projekta.

Web deo projekta pruza mogucnost korisnicima uvid u jelovnik restorana I slika svakog jela, kao I mogucnost unosenja utiska o restoranu I uvid u utiske. Admin ima mogucnost da brise, postavlja I menja jela restorana I utiske

NAPOMENA: Zbog problema sa dekripcijom tokena koji dobijem prilikom logovanja nisam stavio iznad metoda u kontolerima uloge koje mogu da pristupe. Jer neki kod dekriptuje, a neki ne.

Dizajn baze



Domain abstract Model

```
public abstract class Model
    {
        public int Id { get; set; }
        public DateTime? CreateAt { get; set; }
        public DateTime? ModifieAt { get; set; }
        public bool IsDeleted { get; set; }
    }
```

Drink

```
public class Drink:Model
    {
        public string DrinkName { get; set; }
        public double Price { get; set; }
    }
```

Image

```
public class Image
    {
        public int Id { get; set; }
        public string Putanja { get; set; }
        public string Alt { get; set; }

        public ICollection<Meni> Menis { get; set; }
    }
}
```

Impression

```
public class Impression:Model
    {
        public string Content { get; set; }
        public int UserId { get; set; }

        public User User { get; set; }
    }
```

Meal

```
    public class Meal:Model
    {
        public string Name { get; set; }
        public string Start { get; set; }
        public string Finish { get; set; }

        public ICollection<Meni> Menis { get; set; }
    }
```

Meni

```
public class Meni:Model
    {
        public string NameFood { get; set; }
        public string Ingrediant { get; set; }
        public string Price { get; set; }
        public int ImageId { get; set; }

        public Image Image { get; set; }

        public int MealId { get; set; }

        public Meal Meal { get; set; }
```

```
    }
```

Reservation

```
public class Reservation:Model
    {
        public DateTime ReservationDate { get; set; }
        public int Guest { get; set; }
        public string Time { get; set; }
        public int UserId { get; set; }

        public User User { get; set; }
    }
```

Role

```
public class Role:Model
    {
        public string NameRole { get; set; }

        public ICollection<User> Users { get; set; }
    }
```

User

```
public class User:Model
    {
        public string Name { get; set; }
        public string Surname { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }
        public bool Active { get; set; }
        public string Token { get; set; }
        public int RoleId { get; set; }

        public Role Role { get; set; }

        public ICollection<Impression> Impressions { get; set; }
        public ICollection<Reservation> Reservations { get; set; }
    }
```

EFDataAccess/Configurations

Drink

```
public class DrinkConfiguratin : IEntityTypeConfiguration<Drink>
    {
        public void Configure(EntityTypeBuilder<Drink> builder)
```

```
        {
            builder.Property(d => d.DrinkName)
                .HasMaxLength(50)
                .IsRequired();

            builder.Property(d => d.CreateAt).HasDefaultValueSql("GETDATE()");
            builder.Property(d => d.ModifieAt).HasDefaultValueSql("GETDATE()");
            builder.Property(d => d.IsDeleted).HasDefaultValue(false);
        }
    }
```

Image

```
public class ImageConfiguration : IEntityTypeConfiguration<Image>
    {
        public void Configure(EntityTypeBuilder<Image> builder)
        {
            builder.Property(i => i.Putanja).IsRequired();
            builder.Property(i => i.Alt).IsRequired().HasMaxLength(50);
        }
    }
```

Impression

```
public class ImpressionConfiguration : IEntityTypeConfiguration<Impression>
    {
        public void Configure(EntityTypeBuilder<Impression> builder)
        {
            builder.Property(i => i.Content)
                .IsRequired();

            builder.Property(i => i.CreateAt).HasDefaultValueSql("GETDATE()");
            builder.Property(i => i.ModifieAt).HasDefaultValueSql("GETDATE()");
            builder.Property(i => i.IsDeleted).HasDefaultValue(false);
        }
    }
```

Meal

```
public class MealConfiguration : IEntityTypeConfiguration<Meal>
    {
        public void Configure(EntityTypeBuilder<Meal> builder)
        {
            builder.Property(m => m.Name)
                .HasMaxLength(30)
                .IsRequired();

            builder.Property(m => m.Finish)
               .IsRequired();
```

```
            builder.Property(m => m.Start)
               .IsRequired();



            builder.Property(m => m.CreateAt).HasDefaultValueSql("GETDATE()");
            builder.Property(m => m.ModifieAt).HasDefaultValueSql("GETDATE()");
            builder.Property(m => m.IsDeleted).HasDefaultValue(false);
        }

    }
```

Meni

```
public class MeniConfiguration : IEntityTypeConfiguration<Meni>
    {
        public void Configure(EntityTypeBuilder<Meni> builder)
        {
            builder.Property(m => m.NameFood)
                .HasMaxLength(30)
                .IsRequired();

            builder.Property(m => m.Ingrediant)
               .IsRequired();

            builder.Property(m => m.Price)
               .IsRequired();



            builder.Property(u => u.CreateAt).HasDefaultValueSql("GETDATE()");
            builder.Property(u => u.ModifieAt).HasDefaultValueSql("GETDATE()");
            builder.Property(u => u.IsDeleted).HasDefaultValue(false);
        }


    }
}
```

Reservation

```
        public void Configure(EntityTypeBuilder<Reservation> builder)
        {
            builder.Property(r => r.ReservationDate)
                .HasMaxLength(50)
                .IsRequired();

            builder.Property(r => r.Guest)
                .IsRequired();

            builder.Property(r => r.Time)
                .IsRequired();
```

```
            builder.Property(r => r.CreateAt).HasDefaultValueSql("GETDATE()");
            builder.Property(r => r.ModifieAt).HasDefaultValueSql("GETDATE()");
            builder.Property(r => r.IsDeleted).HasDefaultValue(false);

        }
```

Rule

```
ublic class RuleConfiguration : IEntityTypeConfiguration<Role>
    {
        public void Configure(EntityTypeBuilder<Role> builder)
        {
            builder.Property(r => r.NameRole)
                .IsRequired()
                .HasMaxLength(20);

            builder.Property(r => r.CreateAt).HasDefaultValueSql("GETDATE()");
            builder.Property(r => r.ModifieAt).HasDefaultValueSql("GETDATE()");
            builder.Property(r => r.IsDeleted).HasDefaultValue(false);
        }
    }
```

User

```
public class UserConfiguration : IEntityTypeConfiguration<User>
    {
        public void Configure(EntityTypeBuilder<User> builder)
        {
            builder.Property(u => u.Name)
                .HasMaxLength(50)
                .IsRequired();

            builder.Property(u => u.Surname)
                .HasMaxLength(50)
                .IsRequired();

            builder.Property(u => u.Email)
                .HasMaxLength(150)
                .IsRequired();
            builder.HasIndex(u => u.Email)
                .IsUnique();

            builder.Property(u => u.Active)
                .HasDefaultValue(false);

            builder.Property(u => u.CreateAt).HasDefaultValueSql("GETDATE()");
            builder.Property(u => u.ModifieAt).HasDefaultValueSql("GETDATE()");
            builder.Property(u => u.IsDeleted).HasDefaultValue(false);


        }
    }
```

```
}
```

DBContext

```
public class DBContext:DbContext
    {
        public DbSet<User> Users { get; set; }
        public DbSet<Role> Roles { get; set; }
        public DbSet<Reservation> Reservations { get; set; }

        public DbSet<Meni> Menis { get; set; }
        public DbSet<Meal> Meals { get; set; }

        public DbSet<Impression> Impressions { get; set; }
        public DbSet<Drink> Drinks { get; set; }
        public DbSet<Image> Images { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Data Source=USER-PC\SQLEXPRESS;Initial
Catalog=APIRestoran;Integrated Security=True");
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new DrinkConfiguratin());
            modelBuilder.ApplyConfiguration(new ImpressionConfiguration());
            modelBuilder.ApplyConfiguration(new MealConfiguration());
            modelBuilder.ApplyConfiguration(new MeniConfiguration());
            modelBuilder.ApplyConfiguration(new ReservationConfiguration());
            modelBuilder.ApplyConfiguration(new RuleConfiguration());
            modelBuilder.ApplyConfiguration(new UserConfiguration());
            modelBuilder.ApplyConfiguration(new ImageConfiguration());

        }
    }
```

Swagger

Restoran API v1

/swagger/v1/swagger.json

Auth

POST /api/Auth

GET /api/Auth/decode

Drink

GET /api/Drink

Parameters Try it out

Name	Description
DrinkName string (query)	
MinPrice number($double) (query)	
MaxPrice number($double) (query)	
PerPage integer($int32) (query)	
PageNumber integer($int32) (query)	
Id integer($int32) (query)	
Keyword string (query)	
IsDeleted boolean (query)	

Responses Response content type text/plain

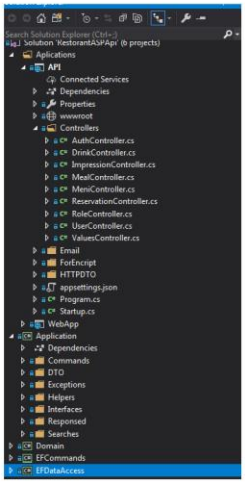
Code	Description
200	*Success*

Example Value | Model

```
[
  {
    "id": 0,
    "drinkName": "string",
    "price": 0
  }
]
```

POST /api/Drink

API controleri



STARTUP API

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
            services.AddDbContext<DBContext>();
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new Info { Title = "Restoran API", Version = "v1" });
            });


            services.AddTransient<IDrinkCreateCommand, EFCreateDrinkCommand>();
            services.AddTransient<IDeleteDrinkCommand, EFDeleteDrinkCommand>();
            services.AddTransient<IEditDrinkCommand, EFEditDrinkCommand>();
            services.AddTransient<IGetDrinksCommand, EFGetDrinksCommand>();
            services.AddTransient<IGetDrinkCommand, EFGetDrinkCommand>();



            services.AddTransient<IAddRolleCommand, EFAddRolleCommand>();
            services.AddTransient<IGetRolesCommand, EFGetRolesCommand>();
```

```
            services.AddTransient<IGetRoleCommand, EFGetRoleCommand>();
            services.AddTransient<IEditRoleCommand, EFEditRoleCommand>();
            services.AddTransient<IDeleteRoleCommand, EFDeleteRoleCommand>();



            services.AddTransient<IAddUserCommand, EFAddUserCommand>();
            services.AddTransient<IDeleteUserCommand, EFDeleteUserCommand>();
            services.AddTransient<IGetUserCommand, EFGetUserCommand>();
            services.AddTransient<IEditUserCommand, EFEditUserCommand>();
            services.AddTransient<IGetUsersCommand, EFGetUsersCommand>();



            services.AddTransient<IAddImpresssionCommand, EFAddImpressionCommand>();
            services.AddTransient<IDeleteImpressionCommand, EFDeleteImpressionCommand>();
            services.AddTransient<IGetImpressionCommand, EFGetImpressionCommand>();
            services.AddTransient<IEditImpressionCommand, EFEditImpressionCommand>();
            services.AddTransient<IGetImpressionsCommand, EFGetImpressionsCommand>();



            services.AddTransient<IAddReservationCommmand, EFAddReservationCommand>();
            services.AddTransient<IDeleteReservationCommand, EFDeleteReservationCommand>();
            services.AddTransient<IGetReservationCommand, EFGetReservationCommand>();
            services.AddTransient<IEditReservationCommand, EFEditReservationCommand>();
            services.AddTransient<IGetReservationsCommand, EFGetReservationsCommand>();


            services.AddTransient<IAddMealCommand, EFAddMealCommand>();
            services.AddTransient<IDeleteMealCommand, EFDeleteMealCommand>();
            services.AddTransient<IGetMealCommand, EFGetMealCommand>();
            services.AddTransient<IGetMealsCommand, EFGetMealsCommand>();
            services.AddTransient<IEditMealCommand, EFEditMealCommand>();


            services.AddTransient<IAddMeniCommand, EFAddMeniCommand>();
            services.AddTransient<IGetMeniCommand, EFGetMeniCommand>();
            services.AddTransient<IGetMeniesCommand, EFGetMeniesCommand>();
            services.AddTransient<IEditMeniCommand, EFEditMeniCommand>();
            services.AddTransient<IDeleteMeniCommand, EFDeleteMenicommand>();

            var section = Configuration.GetSection("Email");

            var sender =
                new SmtpEmailSender(section["host"], Int32.Parse(section["port"]),
section["fromaddress"], section["password"]);

            services.AddSingleton<IEmailSender>(sender);



            services.AddTransient<IHttpContextAccessor, HttpContextAccessor>();
            services.AddTransient<IAuthUserCommand, EFAuthUserCommand>();

            var key = Configuration.GetSection("Encryption")["key"];
```

```
            var enc = new Encryption(key);
            services.AddSingleton(enc);


            services.AddTransient(s => {
                var http = s.GetRequiredService<IHttpContextAccessor>();
                var value = http.HttpContext.Request.Headers["Authorization"].ToString();
                var encryption = s.GetRequiredService<Encryption>();

                try
                {
                    var decodedString = encryption.DecryptString(value);
                    decodedString = decodedString.Replace("\t", "");
                    var user = JsonConvert.DeserializeObject<LoggedUser>(decodedString);
                    user.IsLogged = true;
                    return user;
                }
                catch (Exception)
                {
                    return new LoggedUser
                    {
                        IsLogged = false
                    };
                }
            });

        }
```

AuthContoller

Kontroler koji omogucava enkripciju I dekrpiciju. Korisnik da bi se logovao mora da unese Email I Password.

```
public class AuthController : ControllerBase
    {
        private readonly Encryption _enc;
        private IAuthUserCommand _authUserCommand;

        public AuthController(Encryption enc, IAuthUserCommand authUserCommand)
        {
            _enc = enc;
            _authUserCommand = authUserCommand;
        }

        // POST: api/Auth
        [HttpPost]
        public ActionResult Post([FromBody] UserAuthDTO request)
        {
```

```
            var user = _authUserCommand.Execute(request);

            var stringObjekat = JsonConvert.SerializeObject(user);

            var encrypted = _enc.EncryptString(stringObjekat);

            return Ok(new { token = encrypted });
        }

        [HttpGet("decode")]
        public IActionResult Decode(string value)
        {
            var decodedString = _enc.DecryptString(value);
            decodedString = decodedString.Replace("\u000e", "");
            var user = JsonConvert.DeserializeObject<LoggedUser>(decodedString);

            return null;
        }

    }
```

LoggedIn za autorizaciju klasa

```
private readonly string _role;
        public LoggedIn(string RoLeName)
        {
            _role = RoLeName;
        }

        public LoggedIn()
        {

        }
        //nakon akcije kontrolera se poziva
        public void OnResourceExecuted(ResourceExecutedContext context)
        {

        }
        //poziva se pre akcije kontrolera
        public void OnResourceExecuting(ResourceExecutingContext context)
        {
            var user = context.HttpContext.RequestServices.GetService<LoggedUser>();

            if (!user.IsLogged)
            {
                context.Result = new UnauthorizedResult();
            }
            else
            {
                if (_role != null)
                {
                    if (user.RoleName != _role)
                    {
```

```
                    context.Result = new UnauthorizedResult();
                }
            }
        }
    }
```

DrinkController

Pri dodavanju novog pica proverava se da li je cena manja od 0 ili je 0, ali ime pica vec postoji. Pri izmeni moguce je izmeniti bilo koji parametar posebno.
Pretraga I paginacija: Drink klasa ima mogucnost pretrage preko:

- DrinkName
- MaxPrice
- MinPrice
- Id
- PageNumber
- PerPage

```
public class DrinkController : ControllerBase
    {
        private IDrinkCreateCommand _createDrinkCommand;
        private IDeleteDrinkCommand _deleteDrinkCommand;
        private IEditDrinkCommand _editDrinkCommand;
        private IGetDrinksCommand _getDrinksCommand;
        private IGetDrinkCommand _getDrinkCommand;

        public DrinkController(IDrinkCreateCommand createDrinkCommand, IDeleteDrinkCommand
deleteDrinkCommand, IEditDrinkCommand editDrinkCommand, IGetDrinksCommand getDrinksCommand,
IGetDrinkCommand getDrinkCommand)
        {
            _createDrinkCommand = createDrinkCommand;
            _deleteDrinkCommand = deleteDrinkCommand;
            _editDrinkCommand = editDrinkCommand;
            _getDrinksCommand = getDrinksCommand;
```

```
            _getDrinkCommand = getDrinkCommand;
        }

        // GET: api/Drink
        [HttpGet]
        public ActionResult<IEnumerable<CreateDrinkDTO>> Get([FromQuery] DrinkSearch
drinkSearch)
        {
            try
            {
               var search = _getDrinksCommand.Execute(drinkSearch);
                return Ok(search);
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // GET: api/Drink/5
        [HttpGet("{id}", Name = "Get")]
        public ActionResult<IEnumerable<CreateDrinkDTO>> Get(int id)
        {
            try
            {
                var drink = _getDrinkCommand.Execute(id);
                return Ok(drink);
            }catch(NotFoundException)
            {
                return NotFound();
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // POST: api/Drink
        [HttpPost]
        public ActionResult Post([FromBody] CreateDrinkDTO value)
        {
            try
            {
                _createDrinkCommand.Execute(value);
                return StatusCode(201,"Create Drink is succesfuly");
            }
            catch (DataCanNotBeNull)
            {
               return StatusCode(422, "Price can not be null");
            }
            catch (AlredyExistException)
            {
                return StatusCode(422, "Drink name alredy exist");
            }
            catch (Exception)
            {
                return StatusCode(500, "Server errors, try later");
            }
        }
```

```
        // PUT: api/Drink/5
        [HttpPut("{id}")]
        public ActionResult Put(int id,[FromBody] CreateDrinkDTO value)
        {
            try
            {
                _editDrinkCommand.Execute(value);
                return NoContent();
            }catch(AlredyExistException)
            {
                return StatusCode(409, "Drink name Alredy Exist");
            }catch(DataCanNotBeNull)
            {
                return StatusCode(409, "Price can not be null");
            }catch(NotFoundException)
            {
                return NotFound();
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public ActionResult Delete(int id)
        {
            try
            {
                _deleteDrinkCommand.Execute(id);
                return NoContent();
            }
            catch (NotFoundException)
            {
                return NotFound();
            }
            catch (Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }
```

Aplication/CreateDrinkDTO

```
public class CreateDrinkDTO
    {
        public int Id { get; set; }
        [Required(ErrorMessage ="Name of drink is required")]
        [MinLength(3,ErrorMessage ="Name of drink can not be short than 3")]
        [MaxLength(20,ErrorMessage = "Name of drink can not be loger than 20")]
        public string DrinkName { get; set; }
        [Required(ErrorMessage = "Price of drink is required")]
        public double Price { get; set; }
       }
```

Aplication/Commands/DrinkCommand

```
public interface IDeleteDrinkCommand:ICommand<int>

public interface IDrinkCreateCommand:ICommand<CreateDrinkDTO>

public interface IEditDrinkCommand:ICommand<CreateDrinkDTO>

public interface IGetDrinkCommand:ICommand<int,IEnumerable<CreateDrinkDTO>>

public interface IGetDrinksCommand:ICommand<DrinkSearch,PagedRespone<CreateDrinkDTO>>
```

EFCommands/EFBaseCommadn

```
public abstract class EFBaseCommand
    {
        protected readonly DBContext _context;

        public EFBaseCommand(DBContext context)
        {
            this._context = context;
        }

    }
```

EFCommands/EFCreateDrinkCommand

```
public class EFCreateDrinkCommand : EFBaseCommand, IDrinkCreateCommand
    {
        public EFCreateDrinkCommand(DBContext context) : base(context)
        {
        }

        public void Execute(CreateDrinkDTO request)
        {
            if(_context.Drinks.Any(d => d.DrinkName == request.DrinkName))
            {
                throw new AlredyExistException();
            }

            if (request.DrinkName == null)
            {
                throw new DataCanNotBeNull();
            }

            if (request.Price <= 0)
            {
```

```
            throw new DataCanNotBeNull();
        }

        _context.Add(new Drink
        {
            DrinkName = request.DrinkName,
            Price = request.Price
        });

        _context.SaveChanges();


    }
}
```

EFDeleteDrinkCommand

```
public class EFDeleteDrinkCommand : EFBaseCommand, IDeleteDrinkCommand
    {
        public EFDeleteDrinkCommand(DBContext context) : base(context)
        {
        }

        public void Execute(int request)
        {
            var drink = _context.Drinks.Find(request);

            if(drink == null)
            {
                throw new NotFoundException();
            }

            _context.Drinks.Remove(drink);

            _context.SaveChanges();

        }
    }
```

EFEditDrinkCommand

```
public class EFEditDrinkCommand : EFBaseCommand, IEditDrinkCommand
    {
        public EFEditDrinkCommand(DBContext context) : base(context)
        {
        }

        public void Execute(CreateDrinkDTO request)
        {
            var drink = _context.Drinks.Find(request.Id);

            if(drink == null)
```

```
            {
                throw new NotFoundException();
            }

            if(_context.Drinks.Any(d => d.DrinkName == request.DrinkName))
            {
                throw new AlredyExistException();
            }

            if(request.Price <= 0)
            {
                throw new DataCanNotBeNull();
            }

            drink.DrinkName = request.DrinkName;
            drink.Price = request.Price;

            _context.SaveChanges();
        }
    }
```

EFGetDrinkCommand

```
public class EFGetDrinkCommand : EFBaseCommand,IGetDrinkCommand
    {
        public EFGetDrinkCommand(DBContext context) : base(context)
        {
        }

        public IEnumerable<CreateDrinkDTO> Execute(int id)
        {
            var query = _context.Drinks.AsQueryable();

            if(_context.Drinks.Any(d => d.Id == id))
            {
                query = query.Where(d => d.Id == id);
            }
            else
            {
                throw new NotFoundException();
            }

            return query.Select(d => new CreateDrinkDTO
            {
                Id = d.Id,
                DrinkName = d.DrinkName,
                Price = d.Price
            });

        }
    }
```

EFGetDrinksCommand

```
public class EFGetDrinksCommand : EFBaseCommand, IGetDrinksCommand
  {
      public EFGetDrinksCommand(DBContext context) : base(context)
      {
      }

      public PagedRespone<CreateDrinkDTO> Execute(DrinkSearch request)
      {
          var query = _context.Drinks.AsQueryable();

          if (request.DrinkName != null)
          {
              var drinName = request.DrinkName.ToLower();

              query = query.Where(d => d.DrinkName.ToLower().Contains(drinName));

          }

          if (request.MaxPrice.HasValue)
          {
              query = query.Where(d => d.Price >= request.MaxPrice);
          }

          if (request.MinPrice.HasValue)
          {
              query = query.Where(d => d.Price <= request.MinPrice);
          }
          if (request.Id.HasValue)
          {
              query = query.Where(d => d.Id == request.Id);
          }

          var TotalCount = query.Count();

          query = query
              .Skip((request.PageNumber - 1) * request.PerPage)
              .Take(request.PerPage);

          var pageCount = (int)Math.Ceiling((double)TotalCount / request.PerPage);

          var response = new PagedRespone<CreateDrinkDTO>
          {
              CurrentPage = request.PageNumber,
              TotalCount = TotalCount,
              PageCount = pageCount,
              Data = query
              .Select(d => new CreateDrinkDTO
              {
                  Id = d.Id,
                  DrinkName = d.DrinkName,
                  Price = d.Price
```

```
                })
        };

            return response;
        }
    }
```

ImpressionController

Pretraga I paginacija: Pretraga se moze odraditi preko

- Content
- PerPage
- CountPage

```
public class ImpressionController : ControllerBase
    {
        private IAddImpresssionCommand _addImpressionCommand;
        private IDeleteImpressionCommand _deleteImpressionCommand;
        private IGetImpressionCommand _getImpressionCommand;
        private IEditImpressionCommand _editImpressionCommand;
        private IGetImpressionsCommand _getImpressionsCommand;
        private readonly LoggedUser _user;

        public ImpressionController(IAddImpresssionCommand addImpressionCommand,
IDeleteImpressionCommand deleteImpressionCommand, IGetImpressionCommand
getImpressionCommand, IEditImpressionCommand editImpressionCommand, IGetImpressionsCommand
getImpressionsCommand, LoggedUser user)
        {
            _addImpressionCommand = addImpressionCommand;
            _deleteImpressionCommand = deleteImpressionCommand;
            _getImpressionCommand = getImpressionCommand;
            _editImpressionCommand = editImpressionCommand;
            _getImpressionsCommand = getImpressionsCommand;
            _user = user;
        }



        // GET: api/Impression

        [HttpGet]
        public ActionResult<IEnumerable<ImpressionDTO>> Get([FromQuery] ImpressSearch
request)
        {
            try
            {
                var search = _getImpressionsCommand.Execute(request);
                return Ok(search);
```

```
    }catch(NotFoundException)
    {
        return NotFound();
    }catch(Exception)
    {
        return StatusCode(500, "Server error, try later");
    }
}

// GET: api/Impression/5
[HttpGet("{id}")]
public ActionResult<ImpressionDTO> Get(int id)
{
    try
    {
        var res = _getImpressionCommand.Execute(id);
        return Ok(res);
    }catch(NotFoundException)
    {
        return NotFound();
    }catch(Exception)
    {
        return StatusCode(500, "Server error, try later");
    }
}

// POST: api/Impression
[HttpPost]
public ActionResult Post([FromBody] ImpressionDTO value)
{
    try
    {
        _addImpressionCommand.Execute(value);
        return StatusCode(201, "Impression is create");
    }
    catch (NotFoundException)
    {
        return NotFound();
    }
    catch (Exception)
    {
        return StatusCode(500, "Server error, try later");
    }
}

// PUT: api/Impression/5
[HttpPut("{id}")]
public ActionResult Put(int id, [FromBody] ImpressionEditDTO value)
{
    try
    {
        _editImpressionCommand.Execute(value);
        return NoContent();
    }catch(NotFoundException)
    {
        return NotFound();
    }catch(Exception)
```

```
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public ActionResult Delete(int id)
        {
            try
            {
                _deleteImpressionCommand.Execute(id);
                return NoContent();
            }catch(NotFoundException)
            {
                return NotFound();
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }
    }
```

ImpressionDTO

```
public class ImpressionDTO
    {
        public int Id { get; set; }
        [MinLength(5, ErrorMessage = "Content can not be short than 3")]

        [Required(ErrorMessage ="Contetn is required")]
        public string Content { get; set; }

        [Required(ErrorMessage = "User ID is required")]
        public int UserId { get; set; }
        public string NameSurname { get; set; }
    }
```

ImpressionEditDTO

```
public class ImpressionEditDTO
    {
        public int Id { get; set; }
        [MinLength(5, ErrorMessage = "Content can not be short than 3")]
        [MaxLength(100, ErrorMessage = "Max character for Content is 100")]
        [Required(ErrorMessage = "Contetn is required")]
        public string Content { get; set; }
    }
```

Aplication/Commands/ImpressionCommand

```
public interface IAddImpresssionCommand : ICommand<ImpressionDTO>

public interface IDeleteImpressionCommand : ICommand<int>

public interface IEditImpressionCommand:ICommand<ImpressionEditDTO>

public interface IGetAllImpressionWebCommand : ICommand<ImpressionSearchWeb,
IEnumerable<ImpressionDTO>>

public interface IGetImpressionCommand : ICommand<int,ImpressionDTO>

public interface IGetImpressionsCommand :
ICommand<ImpressSearch,PagedRespone<ImpressionDTO>>

public  interface IGetImpressionWebCommand : ICommand<int, ImpressionEditDTO>
```

EFAddImpressionCommand

```
public class EFAddImpressionCommand : EFBaseCommand, IAddImpresssionCommand
    {
        public EFAddImpressionCommand(DBContext context) : base(context)
        {
        }

        public void Execute(ImpressionDTO request)
        {
            if (_context.Users.Any(u => u.Id == request.UserId))
            {

            }
            else
            {
                throw new NotFoundException();
            }

            _context.Impressions.Add(new Impression {
                Content = request.Content,
                UserId = request.UserId
            });

            _context.SaveChanges();
        }
    }
```

EFDeleteIC

```
public class EFDeleteImpressionCommand : EFBaseCommand, IDeleteImpressionCommand
    {
        public EFDeleteImpressionCommand(DBContext context) : base(context)
        {
        }

        public void Execute(int id)
        {
            var search = _context.Impressions.Find(id);

            if(search == null)
            {
                throw new NotFoundException();
            }

            _context.Impressions.Remove(search);

            _context.SaveChanges();
        }
    }
```

EFEDIT

```
public class EFEditImpressionCommand : EFBaseCommand, IEditImpressionCommand
    {
        public EFEditImpressionCommand(DBContext context) : base(context)
        {
        }

        public void Execute(ImpressionEditDTO request)
        {
            var searches = _context.Impressions.Find(request.Id);

            if(searches == null)
            {
                throw new NotFoundException();
            }

            searches.Content = request.Content;

            _context.SaveChanges();
        }
    }
```

EFGetAllImpressionWebCommand

```
public class EFGetAllImpressionWebCommand : EFBaseCommand, IGetAllImpressionWebCommand
    {
        public EFGetAllImpressionWebCommand(DBContext context) : base(context)
        {
        }

        public IEnumerable<ImpressionDTO> Execute(ImpressionSearchWeb request)
        {
            var query = _context.Impressions.AsQueryable();

            if (request.Content != null)
            {
                var content = request.Content.ToLower();

                query = query.Where(i => i.Content.ToLower().Contains(content));
            }

            return query.Select(i => new ImpressionDTO
            {
                Id = i.Id,
                Content = i.Content,
                UserId = i.UserId,
                NameSurname = i.User.Name + " " + i.User.Surname
            });

        }
    }
```

EFGetImpression

```
public class EFGetImpressionCommand : EFBaseCommand, IGetImpressionCommand
    {
        public EFGetImpressionCommand(DBContext context) : base(context)
        {
        }

        public ImpressionDTO Execute(int id)
        {
            var obj = _context.Impressions.Find(id);

            if (obj == null)
            {
                throw new NotFoundException();
            }

            var user = _context.Users.Find(obj.UserId);



            return new ImpressionDTO
            {
                Id = obj.Id,
                Content = obj.Content,
                UserId = obj.UserId,
                NameSurname = user.Name + " " + user.Surname
```

```
            };


        }
    }
```

EFGetImpressionsCommand

```
public class EFGetImpressionsCommand : EFBaseCommand, IGetImpressionsCommand
    {
        public EFGetImpressionsCommand(DBContext context) : base(context)
        {
        }

        public PagedRespone<ImpressionDTO> Execute(ImpressSearch request)
        {
            var query = _context.Impressions.AsQueryable();

           if(request.Content != null)
            {
                var content = request.Content.ToLower();

                query = query.Where(i => i.Content.ToLower().Contains(content));
            }

            var totalCount = query.Count();

            query = query.Skip((request.PageNumber - 1) *
request.PerPage).Take(request.PerPage);

            var pageCount = (int)Math.Ceiling((double)totalCount / request.PerPage);

            var response = new PagedRespone<ImpressionDTO>
            {
                TotalCount = totalCount,
                CurrentPage = request.PageNumber,
                PageCount = pageCount,
                Data = query.Include(u => u.User)
                       .Select(i => new ImpressionDTO
                       {
                            Id = i.Id,
                            Content = i.Content,
                            UserId = i.UserId,
                            NameSurname = i.User.Name + " " + i.User.Surname
                       })
            };


            return response;

        }
    }
```

EFGetImpressionWeCommand

```
    public class EFGetImpressionWebCommand : EFBaseCommand, IGetImpressionWebCommand
    {
        public EFGetImpressionWebCommand(DBContext context) : base(context)
        {
        }

        public ImpressionEditDTO Execute(int id)
        {
            var obj = _context.Impressions.Find(id);

            if (obj == null)
            {
                throw new NotFoundException();
            }


            return new ImpressionEditDTO
            {
                Id = obj.Id,
                Content = obj.Content

            };

        }
    }
```

MealController

Pretraga se vrsi preko Name

```
public class MealSearch
    {
        public string Name { get; set; }
        public int PerPage { get; set; } = 4;
        public int PageNumber { get; set; } = 1;
    }
public class MealController : ControllerBase
    {
        private IAddMealCommand _addMealCommand;
        private IDeleteMealCommand _deleteMealCommand;
        private IGetMealCommand _getMealCommand;
        private IGetMealsCommand _getMealsCommand;
        private IEditMealCommand _editMealCommand;

        public MealController(IAddMealCommand addMealCommand,
            IDeleteMealCommand deleteMealCommand,
            IGetMealCommand getMealCommand,
```

```
    IGetMealsCommand getMealsCommand,
    IEditMealCommand editMealCommand)
{
    _addMealCommand = addMealCommand;
    _deleteMealCommand = deleteMealCommand;
    _getMealCommand = getMealCommand;
    _getMealsCommand = getMealsCommand;
    _editMealCommand = editMealCommand;
}

// GET: api/Meal
[HttpGet]
public ActionResult<IEnumerable<MealGetDTO>> Get([FromQuery] MealSearch request)
{
    try
    {
        var meals = _getMealsCommand.Execute(request);
        return Ok(meals);
    }catch(Exception)
    {
        return StatusCode(500, "Server error, try later");
    }
}

// GET: api/Meal/5
[HttpGet("{id}")]
public ActionResult<IEnumerable<MealGetDTO>> Get(int id)
{
    try
    {
        var meal = _getMealCommand.Execute(id);
        return Ok(meal);
    }catch(NotFoundException)
    {
        return StatusCode(404, "Meal not found or meal is deleted");
    }catch(Exception)
    {
        return StatusCode(500, "Server error, try later");
    }
}

// POST: api/Meal
[HttpPost]
public ActionResult Post([FromBody] MealCreateDTO value)
{
    try
    {
        _addMealCommand.Execute(value);
        return StatusCode(201, "Meal is succesfuly create");
    }catch(AlredyExistException)
    {
        return StatusCode(422, "Name of meal alredy exist");

    }catch(Exception)
    {
        return StatusCode(500, "Server error, try later");
    }
```

```
        }

        // PUT: api/Meal/5
        [HttpPut("{id}")]
        public ActionResult Put(int id, [FromBody] MealGetDTO value)
        {
            try
            {
                _editMealCommand.Execute(value);
                return NoContent();
            }catch(NotFoundException)
            {
                return NotFound();
            }catch(AlredyExistException)
            {
                return StatusCode(422, "Name alredy exist");
            }
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public ActionResult Delete(int id)
        {
            try
            {
                _deleteMealCommand.Execute(id);
                return NoContent();
            }catch(NotFoundException)
            {
                return NotFound();
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }
    }
```

MealGetDTO

```
public class MealGetDTO
    {
        public int Id { get; set; }
        public string Name { get; set; }

        public string Start { get; set; }

        public string Finish { get; set; }

        public bool IsDeleted { get; set; }
    }
```

MealCreateDTO

```
public class MealCreateDTO
    {
        [Required(ErrorMessage = "Name is required")]
        [MaxLength(10, ErrorMessage = "Max char for Name is 10")]
        [MinLength(3, ErrorMessage = "Min char for Name is 3")]
        public string Name { get; set; }
        [Required(ErrorMessage = "Start is required")]
        [MaxLength(2, ErrorMessage = "Max char for Start is 3")]
        [Range(6, 23, ErrorMessage = "Range for Start is 6 - 23 h")]
        public string Start { get; set; }
        [Required(ErrorMessage = "Finish is required")]
        [MaxLength(2, ErrorMessage = "Max char for Finish is 3")]
        [Range(6, 23, ErrorMessage = "Range for Finish is 6 - 23 h")]
        public string Finish { get; set; }
    }
```

EFAddMeal

```
public class EFAddMealCommand : EFBaseCommand, IAddMealCommand
    {
        public EFAddMealCommand(DBContext context) : base(context)
        {
        }

        public void Execute(MealCreateDTO request)
        {
            if(_context.Meals.Any(m => m.Name == request.Name))
            {
                throw new AlredyExistException();
            }

            _context.Meals.Add(new Meal {
                Name = request.Name,
                Start = request.Start,
                Finish = request.Finish
            });

            _context.SaveChanges();
        }
    }
```

EFDeleteMeal

```
public class EFDeleteMealCommand : EFBaseCommand, IDeleteMealCommand
    {
        public EFDeleteMealCommand(DBContext context) : base(context)
        {
```

```
        }

        public void Execute(int id)
        {
            var meal = _context.Meals.Find(id);

            if(meal == null)
            {
                throw new NotFoundException();
            }

            meal.IsDeleted = true;
            _context.SaveChanges();
        }
    }
```

EFEditMeal

```
public class EFEditMealCommand : EFBaseCommand, IEditMealCommand
    {
        public EFEditMealCommand(DBContext context) : base(context)
        {
        }

        public void Execute(MealGetDTO request)
        {
            var search = _context.Meals.Find(request.Id);

            if(search == null)
            {
                throw new NotFoundException();
            }

            if(request.Name != null)
            {
                if(!_context.Meals.Any(m => m.Name == request.Name))
                {
                    search.Name = request.Name;

                }else
                {
                    throw new AlredyExistException();
                }
            }

            if(request.Start != null)
            {
                if(search.Start != request.Start)
                {
                    search.Start = request.Start;
                }

            }

            if (request.Finish != null)
```

```
            {
                if (search.Finish != request.Finish)
                {
                    search.Finish = request.Finish;
                }

            }

            if (search.IsDeleted != request.IsDeleted)
            {
                search.IsDeleted = request.IsDeleted;
            }



            _context.SaveChanges();
        }
    }
```

GetMealCommand

```
public class EFGetMealCommand : EFBaseCommand, IGetMealCommand
    {
        public EFGetMealCommand(DBContext context) : base(context)
        {
        }

        public IEnumerable<MealGetDTO> Execute(int id)
        {
            var query = _context.Meals.AsQueryable();

            if (_context.Meals.Any(m => m.Id == id && m.IsDeleted == false))
            {
                query = query.Where(m => m.Id == id);
            }
            else
            {
                throw new NotFoundException();
            }

            return query.Select(m => new MealGetDTO {
                Id = m.Id,
                Name = m.Name,
                Start = m.Start + " h",
                Finish = m.Finish + " h",
                IsDeleted = m.IsDeleted
            });
        }
    }
```

GetMealsCommand

```
public class EFGetMealsCommand : EFBaseCommand, IGetMealsCommand
    {
        public EFGetMealsCommand(DBContext context) : base(context)
        {
        }

        public PagedRespone<MealGetDTO> Execute(MealSearch request)
        {
            var query = _context.Meals.AsQueryable();



            if (request.Name != null)
            {
                var name = request.Name.ToLower();
                query = query.Where(m => m.Name.ToLower().Contains(name));
            }

            var totalCount = query.Count();

            query = query.Skip((request.PageNumber - 1) * request.PerPage)
                .Take(request.PerPage);

            var pageCount = (int)Math.Ceiling((double)totalCount / request.PerPage);

            var response = new PagedRespone<MealGetDTO>
            {

                CurrentPage = request.PageNumber,
                TotalCount = totalCount,
                PageCount = pageCount,
                Data = query.Select(m => new MealGetDTO
                {
                    Id = m.Id,
                    Name = m.Name,
                    Start = m.Start + " h",
                    Finish = m.Finish + " h",
                    IsDeleted = m.IsDeleted
                })
            };
            return response;
        }
    }
```

EFGetWebMealCommand

```
public class EFGetWebMealsCommand : EFBaseCommand, IGetWebMealsCommand
    {
        public EFGetWebMealsCommand(DBContext context) : base(context)
        {
        }

        public IEnumerable<MealGetDTO> Execute(ClassForNullObj request)
        {
            var obj = _context.Meals.AsQueryable();
```

```
            return obj.Select(m => new MealGetDTO {
                Id = m.Id,
                 Name = m.Name
            });
        }
    }
```

MeniController

Moze se pretraziti putem NameFood. Slika je obavezna pri dodavanju I izmeni.

```
public class MeniSearch
    {
        public string NameFood { get; set; }
        public int PerPage { get; set; } = 4;
        public int PageNumber { get; set; } = 1;
    }

public class MeniController : ControllerBase
    {
        private IAddMeniCommand _addMeniCommand;
        private IGetMeniCommand _getMeniCommand;
        private IGetMeniesCommand _getMeniesCommand;
        private IEditMeniCommand _editMeniCommand;
        private IDeleteMeniCommand _deleteMeniCommand;

        public MeniController(IAddMeniCommand addMeniCommand,
            IGetMeniCommand getMeniCommand,
            IGetMeniesCommand getMeniesCommand,
            IEditMeniCommand editMeniCommand,
            IDeleteMeniCommand deleteMeniCommand)
        {
            _addMeniCommand = addMeniCommand;
            _getMeniCommand = getMeniCommand;
            _getMeniesCommand = getMeniesCommand;
            _editMeniCommand = editMeniCommand;
            _deleteMeniCommand = deleteMeniCommand;
        }



        // GET: api/Meni
        [HttpGet]
        public ActionResult<IEnumerable<MeniGetDTO>> Get([FromQuery] MeniSearch request)
        {
            try
            {
                var menies = _getMeniesCommand.Execute(request);
                return Ok(menies);
```

```
            }catch(Exception)
            {
                return StatusCode(500, "Server error try later");
            }
        }

        // GET: api/Meni/5
        [HttpGet("{id}")]
        public ActionResult<IEnumerable<MeniGetDTO>> Get(int id)
        {
            try
            {
                var search = _getMeniCommand.Execute(id);
                return Ok(search);
            }catch(NotFoundException)
            {
                return NotFound();
            }
            catch (Exception)
            {
                return StatusCode(500, "Server error try later");
            }
        }

        // POST: api/Meni
        [HttpPost]
        public ActionResult Post([FromForm] HttpSlikaDTO p)
        {

                var ext = Path.GetExtension(p.Image.FileName); //daje ekstenziju .jpg

                if (!FileUpload.AllowExtensions.Contains(ext))
                {
                    return UnprocessableEntity("Image extension is not allowed.");
                }


            try
            {
                  var newFileName = Guid.NewGuid().ToString() + "_" +
p.Image.FileName;//unique za ime fajla

                    var filePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot",
"uploads", newFileName);

                    p.Image.CopyTo(new FileStream(filePath, FileMode.Create));


                var dto = new MeniAddDTO
                {
                    NameFood = p.NameFood,
                    Ingradiant = p.Ingradiant,
                    FileName = newFileName,
                    Price = p.Price,
                    MealId = p.MealId
                };
```

```
                _addMeniCommand.Execute(dto);
                return NoContent();
            }
            catch (NotFoundException)
            {
                return StatusCode(404, "Meal of food not found");
            }catch(AlredyExistException)
            {
                return StatusCode(422, "Name of food alredy exist");
            }
            catch (Exception)
            {
                return StatusCode(500,"Server error try later");
            }

        }

        // PUT: api/Meni/5
        [HttpPut("{id}")]
        public ActionResult Put(int id, [FromForm] HttpSlikaUpdateDTO p)
        {
            var ext = Path.GetExtension(p.Image.FileName); //daje ekstenziju .jpg

            if (!FileUpload.AllowExtensions.Contains(ext))
            {
                return UnprocessableEntity("Image extension is not allowed.");
            }

            try
            {
                var newFileName = Guid.NewGuid().ToString() + "_" +
p.Image.FileName;//unique za ime fajla

                var filePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot",
"uploads", newFileName);

                p.Image.CopyTo(new FileStream(filePath, FileMode.Create));


                var dto = new MeniAddDTO
                {
                    Id = p.Id,
                    NameFood = p.NameFood,
                    Ingradiant = p.Ingradiant,
                    FileName = newFileName,
                    Price = p.Price,
                    MealId = p.MealId
                };

                _editMeniCommand.Execute(dto);
                return NoContent();
            }
            catch (NotFoundException)
            {
                return StatusCode(404, "Meal id not found or id of meni");
            }
            catch (AlredyExistException)
```

```
            {
                return StatusCode(422, "Name of food alredy exist");
            }
            catch (Exception)
            {
                return StatusCode(500, "Server error try later");
            }
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public ActionResult Delete(int id)
        {
            try
            {
                _deleteMeniCommand.Execute(id);
                return NoContent();
            }catch(NotFoundException)
            {
                return NotFound();
            }catch(Exception)
            {
                return StatusCode(500, "Server error try later");
            }
        }
    }
```

MeniEditDTO

```
public class EditMeniDTO
    {
        public int Id { get; set; }
        [MaxLength(30, ErrorMessage = "Max lenght for name food is 30")]
        [MinLength(5, ErrorMessage = "min lenght for name food is 5")]
        public string NameFood { get; set; }

        [MinLength(5, ErrorMessage = "Min lenght for Ingradiant is 5")]
        public string Ingradiant { get; set; }


        public string Price { get; set; }
        [Required(ErrorMessage = "Meal of food is required")]
        public int MealId { get; set; }

        public IFormFile Image { get; set; }

    }
```

MeniAddDTO

```
public class MeniAddDTO
```

```
    {
        public int Id { get; set; }
        public string NameFood { get; set; }
        public string Ingradiant { get; set; }
        public string Price { get; set; }
        public string FileName { get; set; }

        public int MealId { get; set; }
```

MeniGetDTO

```
public class MeniGetDTO
    {
        public int Id { get; set; }
        public string NameFood { get; set; }
        public string Ingradiant { get; set; }
        public string Price { get; set; }
        public string FileName { get; set; }
        public string MealName { get; set; }
        public string Alt { get; set; }


    }
```

HttpSlika DTO za dodavanje

```
public class HttpSlikaDTO
    {
        [Required(ErrorMessage ="Name of food is required")]
        [MaxLength(30,ErrorMessage ="Max lenght for name food is 30")]
        [MinLength(5,ErrorMessage = "min lenght for name food is 5")]
        public string NameFood { get; set; }
        [Required(ErrorMessage = "Ingradiant of food is required")]
        [MinLength(5, ErrorMessage = "Min lenght for Ingradiant is 5")]
        public string Ingradiant { get; set; }
        [Required(ErrorMessage = "Price of food is required")]

        public string Price { get; set; }
        [Required(ErrorMessage = "Meal of food is required")]
        public int MealId { get; set; }

        public IFormFile Image { get; set; }
    }
}
```

HttpSlikaZaEDitDTO

```
public class HttpSlikaUpdateDTO
    {

        public int Id { get; set; }
```

```
        [MaxLength(30, ErrorMessage = "Max lenght for name food is 30")]
        [MinLength(5, ErrorMessage = "min lenght for name food is 5")]
        public string NameFood { get; set; }

        [MinLength(5, ErrorMessage = "Min lenght for Ingradiant is 5")]
        public string Ingradiant { get; set; }


        public string Price { get; set; }
        [Required(ErrorMessage = "Meal of food is required")]
        public int MealId { get; set; }

        public IFormFile Image { get; set; }
    }
}
```

EFAddMEni

```
public class EFAddMeniCommand : EFBaseCommand, IAddMeniCommand
    {
        public EFAddMeniCommand(DBContext context) : base(context)
        {
        }

        public void Execute(MeniAddDTO request)
        {
            if(_context.Menis.Any(m => m.NameFood == request.NameFood))
            {
                throw new AlredyExistException();
            }

            if(_context.Meals.Any(m => m.Id == request.MealId))
            {

            }else
            {
                throw new NotFoundException();
            }


            var image = new Image
            {
                Putanja = request.FileName,
                Alt = request.NameFood
            };

            _context.Add(new Meni {
                NameFood = request.NameFood,
                Ingrediant = request.Ingradiant,
                Price = request.Price,
                MealId = request.MealId,
                Image = image
                });
```

```
            _context.SaveChanges();
        }
    }
```

EFDelete

```
public class EFDeleteMenicommand : EFBaseCommand, IDeleteMeniCommand
    {
        public EFDeleteMenicommand(DBContext context) : base(context)
        {
        }

        public void Execute(int id)
        {
            var meni = _context.Menis.Find(id);

            if(meni == null)
            {
                throw new NotFoundException();
            }

            var idSlike = meni.ImageId;
            var slika = _context.Images.Find(idSlike);

            _context.Remove(slika);
            _context.Remove(meni);
            _context.SaveChanges();

        }
    }
```

EFEdit

```
public class EFEditMeniCommand : EFBaseCommand, IEditMeniCommand
    {
        public EFEditMeniCommand(DBContext context) : base(context)
        {
        }

        public void Execute(MeniAddDTO request)
        {
            var meni = _context.Menis.Find(request.Id);

            if(meni == null)
            {
                throw new NotFoundException();
            }

            if (request.NameFood != null)
            {
```

```
            if (!_context.Menis.Any(m => m.NameFood == request.NameFood))
            {
                meni.NameFood = request.NameFood;
            }
            else
            {
                throw new AlredyExistException();
            }
        }

        if (request.Ingradiant != null)
        {
            if(meni.Ingrediant != request.Ingradiant)
            {
                meni.Ingrediant = request.Ingradiant;
            }

        }

        if (request.Price != null)
        {
            if (meni.Price != request.Price)
            {
                meni.Price = request.Price;
            }

        }

        if (request.FileName != null)
        {
            var idSlike = meni.ImageId;
            var slika = _context.Images.Find(idSlike);
            slika.Putanja = request.FileName;
        }

        if(meni.MealId != request.MealId)
        {
            if(_context.Meals.Any(m => m.Id == request.MealId))
            {
                meni.MealId = request.MealId;
            }
            else
            {
                throw new NotFoundException();
            }
        }

        _context.SaveChanges();

    }
  }
}
```

EFGetMeniesWeb

```
public class EFGetAllMeniesCommandWeb : EFBaseCommand, IGetAllMeniesCommandWeb
    {
        public EFGetAllMeniesCommandWeb(DBContext context) : base(context)
        {
        }

        public IEnumerable<MeniGetDTO> Execute(MeniSearchWeb request)
        {
            var query = _context.Menis.AsQueryable();

            if (request.NameFood != null)
            {
                var name = request.NameFood.ToLower();

                query = query.Where(m => m.NameFood.ToLower()
                .Contains(name)
                && m.IsDeleted == false);

            }

            return query.Select(m => new MeniGetDTO
            {
                Id = m.Id,
                NameFood = m.NameFood,
                Ingradiant = m.Ingrediant,
                Price = m.Price,
                FileName = m.Image.Putanja,
                Alt = m.Image.Alt,
                MealName = m.Meal.Name
            });
        }
    }
}
```

GetEditWeb

```
ublic class EFGetEditMeniCommand : EFBaseCommand, IGetEditMeniCommand
    {
        public EFGetEditMeniCommand(DBContext context) : base(context)
        {
        }

        public HttpEditMeni Execute(int request)
        {
            var obj = _context.Menis.Find(request);
            if(obj == null)
            {
                throw new NotFoundException();
            }
```

```
            return new HttpEditMeni
            {
                Id = obj.Id,
                NameFood = obj.NameFood,
                Ingradiant = obj.Ingrediant,
                Price = obj.Price,
                MealId = obj.MealId
            };
        }
    }
```

EFGetMeniCommand

```
public class EFGetMeniCommand : EFBaseCommand, IGetMeniCommand
    {
        public EFGetMeniCommand(DBContext context) : base(context)
        {
        }

        public IEnumerable<MeniGetDTO> Execute(int id)
        {
            var query = _context.Menis.AsQueryable();

            if (_context.Menis.Any(m => m.Id == id))
            {
                query = query.Where(m => m.Id == id);
            }
            else
            {
                throw new NotFoundException();
            }

            return query.Select(m => new MeniGetDTO {
                Id = m.Id,
                NameFood = m.NameFood,
                Ingradiant = m.Ingrediant,
                Price = m.Price,
                FileName = m.Image.Putanja,
                Alt = m.Image.Alt,
                MealName = m.Meal.Name
            });
        }
    }
}
```

EFgetMeniesCommand

```
public class EFGetMeniesCommand : EFBaseCommand, IGetMeniesCommand
    {
        public EFGetMeniesCommand(DBContext context) : base(context)
        {
```

```
        }

        public PagedRespone<MeniGetDTO> Execute(MeniSearch request)
        {
            var query = _context.Menis.AsQueryable();

            if (request.NameFood != null)
            {
                var name = request.NameFood.ToLower();

                query = query.Where(m => m.NameFood.ToLower()
                .Contains(name)
                && m.IsDeleted == false);
            }

            var totalCount = query.Count();

            query = query.Skip((request.PageNumber - 1) * request.PerPage)
                .Take(request.PerPage);

            var pageCount = (int)Math.Ceiling((double)totalCount / request.PerPage);


            var response = new PagedRespone<MeniGetDTO>
            {
                CurrentPage = request.PageNumber,
                TotalCount = totalCount,
                PageCount = pageCount,
                Data = query.Select(m => new MeniGetDTO {
                    Id = m.Id,
                    NameFood = m.NameFood,
                    Ingradiant = m.Ingrediant,
                    Price = m.Price,
                    FileName = m.Image.Putanja,
                    Alt = m.Image.Alt,
                    MealName = m.Meal.Name
                })

            };

            return response;
        }
    }
```

EFGetMeniWebCommand

```
public class EFGetMeniWebCommand : EFBaseCommand, IGetMeniWebCommand
    {
        public EFGetMeniWebCommand(DBContext context) : base(context)
        {
        }

        public MeniGetDTO Execute(int request)
        {
            var obj = _context.Menis.Find(request);
```

```
            if (obj == null)
            {
                throw new NotFoundException();
            }
            var img = _context.Images.Find(obj.ImageId);
            var meal = _context.Meals.Find(obj.MealId);

            return new MeniGetDTO
            {
                Id = obj.Id,
                NameFood = obj.NameFood,
                Ingradiant = obj.Ingrediant,
                Price = obj.Price,
                FileName = img.Putanja,
                Alt = img.Alt,
                MealName = meal.Name

            };
        }
    }
}
```

ReservationController

Rezervacija moze da se pretrazi preko Name tj imena usera koji je kreirao

```
public class ReservationController : ControllerBase
    {
        private IAddReservationCommmand _addReservationCommand;
        private IDeleteReservationCommand _deleteReservationCommand;
        private IGetReservationCommand _getReservationCommand;
        private IEditReservationCommand _editReservationCommand;
        private IGetReservationsCommand _getReservationsCommmand;

        public ReservationController(IAddReservationCommmand addReservationCommand,
IDeleteReservationCommand deleteReservationCommand, IGetReservationCommand
getReservationCommand, IEditReservationCommand editReservationCommand,
IGetReservationsCommand getReservationsCommmand)
        {
            _addReservationCommand = addReservationCommand;
            _deleteReservationCommand = deleteReservationCommand;
            _getReservationCommand = getReservationCommand;
            _editReservationCommand = editReservationCommand;
            _getReservationsCommmand = getReservationsCommmand;
        }



        // GET: api/Reservation
```

```
        [HttpGet]
        public ActionResult<IEnumerable<ReservationGetDTO>> Get([FromQuery]
ReservationSearch request)
        {
            try
            {
                var response = _getReservationsCommmand.Execute(request);
                return Ok(response);
            }
            catch
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // GET: api/Reservation/5
        [HttpGet("{id}")]
        public ActionResult<IEnumerable<ReservationGetDTO>> Get(int id)
        {
            try
            {
                var res = _getReservationCommand.Execute(id);
                return Ok(res);
            }catch(NotFoundException)
            {
                return NotFound();
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // POST: api/Reservation
        [HttpPost]
        public ActionResult Post([FromBody] ReservationDTO value)
        {
            try
            {
                _addReservationCommand.Execute(value);
                return StatusCode(201, "Reservation create");
            }catch(DataCanNotBeNull)
            {
                return StatusCode(422, "Guest can not be null");
            }catch(NotFoundException)
            {
                return StatusCode(404, "User not found");
            }
            catch (Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // PUT: api/Reservation/5
        [HttpPut("{id}")]
        public ActionResult Put(int id, [FromBody] ReservationEditDTO value)
        {
```

```
            try
            {
                _editReservationCommand.Execute(value);
                return NoContent();
            }catch(AlredyExistException)
            {
                return StatusCode(422, "Any of parameters alredy exit value");

            }
            catch(NotFoundException)
            {
                return NotFound();
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public ActionResult Delete(int id)
        {
            try
            {
                _deleteReservationCommand.Execute(id);
                return NoContent();
            }catch(NotFoundException)
            {
                return NotFound();
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }
    }
```

ReservationDTO

```
public class ReservationDTO
    {
        public int Id { get; set; }
        [Required(ErrorMessage = "Reservation date is required")]
        public DateTime ReservationDate { get; set; }
        [Required(ErrorMessage = "Numper guest is required")]
        public int Guest { get; set; }
        [Required(ErrorMessage = "Time date is required")]
        [MaxLength(2,ErrorMessage ="Max char for Time is 3")]
        [Range(6,23,ErrorMessage ="Range for time is 6 - 23 h")]
        public string Time { get; set; }
        [Required(ErrorMessage = "User is required")]
        public int UserId { get; set; }
    }
```

EFAdd

```
public class EFAddReservationCommand : EFBaseCommand, IAddReservationCommmand
    {
        public EFAddReservationCommand(DBContext context) : base(context)
        {
        }

        public void Execute(ReservationDTO request)
        {
            if(request.Guest <= 0)
            {
                throw new DataCanNotBeNull();
            }



            if(!_context.Users.Any(u => u.Id == request.UserId))
            {
                throw new NotFoundException();
            }

            _context.Reservations.Add(new Reservation
            {
                ReservationDate = request.ReservationDate,
                Guest = request.Guest,
                Time = request.Time,
                UserId = request.UserId
            });

            _context.SaveChanges();
        }
    }
```

EFDelete

```
public class EFDeleteReservationCommand : EFBaseCommand, IDeleteReservationCommand
    {
        public EFDeleteReservationCommand(DBContext context) : base(context)
        {
        }

        public void Execute(int id)
        {
            var res = _context.Reservations.Find(id);

            if(res == null)
            {
                throw new NotFoundException();
            }

            _context.Remove(res);
```

```
            _context.SaveChanges();
        }
    }
}
```

EFEditReservation

```
public class EFEditReservationCommand : EFBaseCommand, IEditReservationCommand
    {
        public EFEditReservationCommand(DBContext context) : base(context)
        {
        }

        public void Execute(ReservationEditDTO request)
        {
            var reservation = _context.Reservations.Find(request.Id);

            if (reservation == null)
            {
                throw new NotFoundException();
            }

            if(request.ReservationDate != null)
            {
                if (reservation.ReservationDate != request.ReservationDate)
                {
                    reservation.ReservationDate = request.ReservationDate;
                }
                else
                {
                    throw new AlredyExistException();
                }
            }

            if (request.Guest != null)
            {
                if (reservation.Guest != request.Guest)
                {
                    reservation.Guest = (int)request.Guest;
                }
                else
                {
                    throw new AlredyExistException();
                }
            }

            if (request.Time != null)
            {
                if (reservation.Time != request.Time)
                {
                    reservation.Time = request.Time;
                }
```

```
                else
                {
                    throw new AlredyExistException();
                }
            }

            _context.SaveChanges();

        }
    }
}
```

EFGetReservationCommand

```
public class EFGetReservationCommand : EFBaseCommand, IGetReservationCommand
    {
        public EFGetReservationCommand(DBContext context) : base(context)
        {
        }

        public IEnumerable<ReservationGetDTO> Execute(int id)
        {
            var query = _context.Reservations.AsQueryable();

            if (_context.Reservations.Any(r => r.Id == id))
            {
                query = query.Where(r => r.Id == id);
            }
            else
            {
                throw new NotFoundException();
            }

            return query.Include(u => u.User)
                .Select(r => new ReservationGetDTO
                {
                    Id = r.Id,
                    ReservationDate = r.ReservationDate,
                    Guest = r.Guest,
                    Time = r.Time + " h",
                    UserName = r.User.Name + " " + r.User.Surname
                });
        }
    }
}
```

EFGetReservationsCommand

```
public class EFGetReservationsCommand : EFBaseCommand, IGetReservationsCommand
    {
        public EFGetReservationsCommand(DBContext context) : base(context)
        {
```

```
        }

        public PagedRespone<ReservationGetDTO> Execute(ReservationSearch request)
        {
            var query = _context.Reservations.AsQueryable();



            if(request.NameUser != null)
            {
                var name = request.NameUser.ToLower();
                query = query.Where(r => r.User.Name.ToLower().Contains(name));
            }

            var totalCount = query.Count();

            query = query.Skip((request.PageNumber - 1) * request.PerPage)
                .Take(request.PerPage);

            var pageCount = (int)Math.Ceiling((double)totalCount / request.PerPage);

            var response = new PagedRespone<ReservationGetDTO>
            {

                CurrentPage = request.PageNumber,
                TotalCount = totalCount,
                PageCount = pageCount,
                Data = query.Include(u => u.User)
                .Select(r => new ReservationGetDTO
                {
                    Id = r.Id,
                    ReservationDate = r.ReservationDate,
                    Guest = r.Guest,
                    Time = r.Time,
                    UserName = r.User.Name + ' ' + r.User.Surname

                })

            };
            return response;
        }
    }
}
```

RolleController

Pretraga Uloga je omogucena preko:

- RoleName
- Id

- IsDelete
- Keyword

```
    public class RuleSearch: BaseSearch
    {
        public string RoleName { get; set; }
    }

public abstract class BaseSearch
    {
        public int? Id { get; set; }
        public string Keyword { get; set; }
        public bool? IsDeleted { get; set; }
    }
```

Controler

```
public class RoleController : ControllerBase
    {
        private IAddRolleCommand _addRoleCommand;
        private IGetRolesCommand _getRolesCommand;
        private IGetRoleCommand _getRoleCommand;
        private IEditRoleCommand _editRoleCommand;
        private IDeleteRoleCommand _deleteRoleCommand;

        public RoleController(IAddRolleCommand addRoleCommand, IGetRolesCommand
getRolesCommand, IGetRoleCommand getRoleCommand, IEditRoleCommand editRoleCommand,
IDeleteRoleCommand deleteRoleCommand)
        {
            _addRoleCommand = addRoleCommand;
            _getRolesCommand = getRolesCommand;
            _getRoleCommand = getRoleCommand;
            _editRoleCommand = editRoleCommand;
            _deleteRoleCommand = deleteRoleCommand;
        }

        // GET: api/Role
        [HttpGet]
        public ActionResult<IEnumerable<RoleDTO>> GetRole([FromQuery] RuleSearch request)
        {
            try
            {
                var search = _getRolesCommand.Execute(request);
                return Ok(search);
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // GET: api/Role/5
```

```
[HttpGet("{id}")]
public ActionResult<IEnumerable<RoleDTO>> GetRole(int id)
{
    try
    {
        var search = _getRoleCommand.Execute(id);
        return Ok(search);
    }catch(NotFoundException)
    {
        return NotFound();
    }catch(Exception)
    {
        return StatusCode(500, "Server error, try later");
    }
}

// POST: api/Role
[HttpPost]
public ActionResult PostRole([FromBody] RoleDTO value)
{

    try
    {
        _addRoleCommand.Execute(value);
        return StatusCode(201,"Create now Role");
    }
    catch (AlredyExistException)
    {
        return StatusCode(422, "This role alredy exist.");
    }
    catch (DataCanNotBeNull)
    {
        return StatusCode(422, "Role name can not be null");
    }
    catch (Exception)
    {
        return StatusCode(500, "Server error, try later");
    }
}

// PUT: api/Role/5
[HttpPut("{id}")]
public ActionResult PutRole(int id, [FromBody] RoleDTO value)
{
    try
    {
        _editRoleCommand.Execute(value);
        return NoContent();
    }catch(AlredyExistException)
    {
        return StatusCode(422, "Role name or isDelete alredy set.");
    }catch(NotFoundException)
    {
        return NotFound();
    }catch(Exception)
    {
        return StatusCode(500, "Server error, try later");
```

```
            }
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public ActionResult DeleteRole(int id)
        {
            try
            {
                _deleteRoleCommand.Execute(id);
                return NoContent();
            }catch(NotFoundException)
            {
                return NotFound();
            }catch(AlredyExistException)
            {
                return StatusCode(422, "Role is alredy deleted");
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }
    }
}
```

RoleDTO

```
public class RoleDTO
    {
        public int Id { get; set; }
        [Required(ErrorMessage = "Name of drink is required")]
        [MinLength(3, ErrorMessage = "Name of drink can not be short than 3")]
        [MaxLength(15, ErrorMessage = "Name of drink can not be loger than 20")]
        public string RoleName { get; set; }
        public bool IsDelete { get; set; }
    }
```

EFAddRoleCommand

```
ublic class EFAddRolleCommand : EFBaseCommand, IAddRolleCommand
    {
        public EFAddRolleCommand(DBContext context) : base(context)
        {
        }

        public void Execute(RoleDTO request)
        {
            if(_context.Roles.Any(r => r.NameRole == request.RoleName))
            {
                throw new AlredyExistException();
            }
```

```
            if(request.RoleName == null)
            {
                throw new DataCanNotBeNull();
            }

            _context.Add(new Role
            {
                NameRole = request.RoleName
            });

            _context.SaveChanges();
        }
    }
}
```

EFDeleteRolle

```
    public class EFDeleteRoleCommand : EFBaseCommand, IDeleteRoleCommand
    {
        public EFDeleteRoleCommand(DBContext context) : base(context)
        {
        }

        public void Execute(int id)
        {
            var role = _context.Roles.Find(id);

            if(role == null)
            {
                throw new NotFoundException();
            }

            if(role.IsDeleted == true)
            {
                throw new AlredyExistException();
            }

            role.IsDeleted = true;

            _context.SaveChanges();
        }
    }
}
```

EFEditRole

```
public class EFEditRoleCommand : EFBaseCommand, IEditRoleCommand
    {
        public EFEditRoleCommand(DBContext context) : base(context)
        {
        }
```

```
        public void Execute(RoleDTO request)
        {
            var searchRole = _context.Roles.Find(request.Id);

            if(searchRole == null)
            {
                throw new NotFoundException();
            }

            if(_context.Roles.Any(r => r.NameRole == request.RoleName))
            {
                throw new AlredyExistException();
            }

            if(searchRole.NameRole == request.RoleName)
            {
                throw new AlredyExistException();
            }

            if(searchRole.IsDeleted == request.IsDelete)
            {
                throw new AlredyExistException();
            }

            searchRole.NameRole = request.RoleName;
            searchRole.IsDeleted = request.IsDelete;

            _context.SaveChanges();
        }
    }
}
```

EFGetRoleCommand

```
public class EFGetRoleCommand : EFBaseCommand, IGetRoleCommand
    {
        public EFGetRoleCommand(DBContext context) : base(context)
        {
        }

        public IEnumerable<RoleDTO> Execute(int request)
        {
            var query = _context.Roles.AsQueryable();

            if(_context.Roles.Any(r => r.Id == request))
            {
                query = query.Where(r => r.Id == request);
            }else
            {
                throw new NotFoundException();
            }

            return query.Select(r => new RoleDTO {
                Id = r.Id,
```

```
                RoleName = r.NameRole,
                IsDelete = r.IsDeleted
            });
        }
    }
}
```

EFGetRollesCommand

```
public class EFGetRolesCommand : EFBaseCommand, IGetRolesCommand
    {
        public EFGetRolesCommand(DBContext context) : base(context)
        {
        }

        public IEnumerable<RoleDTO> Execute(RuleSearch request)
        {
            var query = _context.Roles.AsQueryable();

            if(request.RoleName != null)
            {
                var name = request.RoleName.ToLower();
                query = query.Where(r => r.NameRole.ToLower()
                .Contains(name) && r.IsDeleted == false);
            }

            if (request.Keyword != null)
            {
                var name = request.Keyword.ToLower();
                query = query.Where(r => r.NameRole.ToLower()
                .Contains(name) && r.IsDeleted == false);
            }

            if(request.IsDeleted.HasValue)
            {
                query = query.Where(r => r.IsDeleted == request.IsDeleted);
            }

            if(request.Id.HasValue)
            {
                query = query.Where(r => r.Id == request.Id && r.IsDeleted == false);
            }

            return query
                .Select(r => new RoleDTO {
                    Id = r.Id,
                    RoleName = r.NameRole,
                    IsDelete = r.IsDeleted
            });
        }
    }
}
```

UserCotroller

Useri mogu da se pretraze po Name, Surname, IsDeleted

```
    public class UserSearch
    {

        public string Name { get; set; }
        public string SurName { get; set; }
        public bool? IsDeleted { get; set; }

        public int PerPage { get; set; } = 4;
        public int PageNumber { get; set; } = 1;
    }
```

Controller

```
public class UserController : ControllerBase
    {
        private IAddUserCommand _addUserCommand;
        private IDeleteUserCommand _deleteUserCommand;
        private IGetUserCommand _getUserCommand;
        private IEditUserCommand _editUserCommand;
        private IGetUsersCommand _getUsersCommand;
        private IEmailSender _sender;

        public UserController(IAddUserCommand addUserCommand, IDeleteUserCommand
deleteUserCommand, IGetUserCommand getUserCommand, IEditUserCommand editUserCommand,
IGetUsersCommand getUsersCommand, IEmailSender sender)
        {
            _addUserCommand = addUserCommand;
            _deleteUserCommand = deleteUserCommand;
            _getUserCommand = getUserCommand;
            _editUserCommand = editUserCommand;
            _getUsersCommand = getUsersCommand;
            _sender = sender;
        }




        // GET: api/User
        [HttpGet]
        public ActionResult<IEnumerable<UserOnlySearchDTO>> Get([FromQuery] UserSearch
request)
        {
            try
            {
                var users = _getUsersCommand.Execute(request);
                return Ok(users);
            }catch(Exception)
```

```
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // GET: api/User/5
        [HttpGet("{id}")]
        public ActionResult<IEnumerable<UserOnlySearchDTO>> Get(int id)
        {
            try
            {
                var user = _getUserCommand.Execute(id);
                return Ok(user);
            }catch(NotFoundException)
            {
                return NotFound();
            }
            catch (Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // POST: api/User UserDTO request
        [HttpPost]
        public ActionResult Post([FromBody] UserDTO request)
        {
            try
            {
                _addUserCommand.Execute(request);
                _sender.Subject = "Uspesno ste se registrovali";
                _sender.ToEmail = request.Email;
                _sender.Body = "Dobrodosli na Api restorana Pulse sada imate mogucnost da
upisete Vas utisak ako ste nas posetili, ako niste sta cekate trk na rezervaciju vase
dorucka,rucka ili romanticke vecere. Vidimo se!";
                _sender.Send();

                return StatusCode(201, "User is succefuly create.");
            }
            catch(NotFoundException)
            {
                return StatusCode(404, "Role id Not Found");
            }catch(AlredyExistException)
            {
                return StatusCode(422, "Email alredy exist");
            }
            catch (Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }


        // PUT: api/User/5
        [HttpPut("{id}")]
        public ActionResult Put(int id, [FromBody] UserSearchDTO value)
        {
```

```
            try
            {
                _editUserCommand.Execute(value);
                return NoContent();
            }catch(AlredyExistException)
            {
                return StatusCode(422, "Email or Name or Surname or Rolle or Password
alredy exist");
            }
            catch (NotFoundException)
            {
                return NotFound();
            }
            catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public ActionResult Delete(int id)
        {
            try
            {
                _deleteUserCommand.Execute(id);
                return StatusCode(204,"User is deleted");
            }catch(AlredyExistException)
            {
                return Conflict("User is alredy deleted");
            }catch(NotFoundException)
            {
                return NotFound();
            }catch(Exception)
            {
                return StatusCode(500, "Server error, try later");
            }
        }
    }
}
```

UserAuthDTO

```
public class UserAuthDTO
    {
        [Required(ErrorMessage = "Email is required!")]
        [EmailAddress(ErrorMessage = "Email nije u dobrom formatu")]
        public string Email { get; set; }
        [MinLength(6, ErrorMessage = "Too less charachters for password, 6 is min!")]
        [Required(ErrorMessage = "Password is required!")]
        public string Password { get; set; }
    }
```

UserDTO

```
    public class UserDTO
    {
        public int Id { get; set; }
        [MaxLength(30, ErrorMessage = "Too many charachters for  name, 30 is max!")]
        [MinLength(3, ErrorMessage = "Too less charachters for  name, 3 is min!")]
        [Required(ErrorMessage ="Name is required!")]
        public string Name { get; set; }
        [MaxLength(30, ErrorMessage = "Too many charachters for Surname, 30 is max!")]
        [MinLength(3, ErrorMessage = "Too less charachters for Surname, 3 is min!")]
        [Required(ErrorMessage = "Surname is required!")]
        public string Surname { get; set; }
        [Required(ErrorMessage = "Email is required!")]
        [EmailAddress(ErrorMessage ="Email nije u dobrom formatu")]
        public string Email { get; set; }

        [MinLength(6, ErrorMessage = "Too less charachters for password, 6 is min!")]
        [Required(ErrorMessage = "Password is required!")]
        public string Password { get; set; }


        public int RoleId { get; set; }

        public bool IsDelete { get; set; }
    }
}
```

UserOnlySearchDTO

```
public class UserOnlySearchDTO
    {
        public int Id { get; set; }
        [MaxLength(30, ErrorMessage = "Too many charachters for  name, 30 is max!")]
        [MinLength(3, ErrorMessage = "Too less charachters for  name, 3 is min!")]
        public string Name { get; set; }
        [MaxLength(30, ErrorMessage = "Too many charachters for Surname, 30 is max!")]
        [MinLength(3, ErrorMessage = "Too less charachters for Surname, 3 is min!")]
        public string Surname { get; set; }
        public string Email { get; set; }

        public string RoleName { get; set; }
        public bool IsDelete { get; set; }
    }
```

EFCommands

EFAddUser

```
public class EFAddUserCommand : EFBaseCommand, IAddUserCommand
    {
        public EFAddUserCommand(DBContext context) : base(context)
        {
        }

        public void Execute(UserDTO request)
        {
            if(_context.Users.Any(u => u.Email == request.Email))
            {
                throw new AlredyExistException();
            }

            if (_context.Roles.Any(r => r.Id == request.RoleId))
            {

            }else
            {
                throw new NotFoundException();
            }

            _context.Users.Add(new User {
                Name = request.Name,
                Surname = request.Surname,
                Email = request.Email,
                Password = request.Password,
                RoleId = request.RoleId
            });

            _context.SaveChanges();
        }
    }
}
```

EFAuthUserCommand

```
public class EFAuthUserCommand : EFBaseCommand, IAuthUserCommand
    {
        public EFAuthUserCommand(DBContext context) : base(context)
        {
        }

        public LoggedUser Execute(UserAuthDTO request)
        {
            var user = _context.Users
                 .Include(u => u.Role)
                 .Where(u => u.Email == request.Email)
                 .Where(u => u.Password == request.Password)
                 .FirstOrDefault();
```

```
            if(user == null)
            {
                throw new NotFoundException();
            }
            return new LoggedUser
            {
                Id = user.Id,
                 Name = user.Name,
                  Surname = user.Surname,
                   Email = user.Email,
                    RoleName = user.Role.NameRole
            };
        }
    }
}
```

EFDeleteUser

```
public class EFDeleteUserCommand : EFBaseCommand, IDeleteUserCommand
    {
        public EFDeleteUserCommand(DBContext context) : base(context)
        {
        }

        public void Execute(int id)
        {
            var user = _context.Users.Find(id);

            if(user == null)
            {
                throw new NotFoundException();
            }

            if(user.IsDeleted == true)
            {
                throw new AlredyExistException();
            }

            user.IsDeleted = true;

            _context.SaveChanges();
        }
    }
}
```

EFEditUser

```
public class EFEditUserCommand : EFBaseCommand, IEditUserCommand
    {
        public EFEditUserCommand(DBContext context) : base(context)
        {
        }
```

```
public void Execute(UserSearchDTO request)
{
    var user = _context.Users.Find(request.Id);

    if(user == null)
    {
        throw new NotFoundException();
    }

    if(request.Email != null)
    {
        if(!_context.Users.Any(u => u.Email == request.Email))
        {
            user.Email = request.Email;
        }
        else
        {
            throw new AlredyExistException();
        }
    }

    if (request.Name != null)
    {
        if (user.Name != request.Name)
        {
            user.Name = request.Name;
        }
        else
        {
            throw new AlredyExistException();
        }
    }

    if(request.Surname != null)
    {
        if(user.Surname != request.Surname)
        {
            user.Surname = request.Surname;
        }else
        {
            throw new AlredyExistException();
        }

    }

    if (request.Password != null)
    {
        if (user.Password != request.Password)
        {
            user.Password = request.Password;
        }
        else
        {
            throw new AlredyExistException();
        }
```

```
            }

            if(request.RoleId != 0)
            {
                if(user.RoleId != request.RoleId)
                {
                    if (_context.Roles.Any(r => r.Id == request.Id))
                    {
                        user.RoleId = request.RoleId;
                    }else
                    {
                        throw new NotFoundException();
                    }
                }else
                {
                    throw new AlredyExistException();
                }
            }



                if(user.IsDeleted != request.IsDelete)
                {
                    user.IsDeleted = request.IsDelete;
                }


            _context.SaveChanges();



        }
    }
}
```

EFGetUser

```
public class EFGetUserCommand : EFBaseCommand, IGetUserCommand
    {
        public EFGetUserCommand(DBContext context) : base(context)
        {
        }

        public IEnumerable<UserOnlySearchDTO> Execute(int id)
        {
            var query = _context.Users.AsQueryable();

            if (_context.Users.Any(u => u.Id == id))
            {
                query = query.Where(u => u.Id == id);
            }
            else
            {
```

```
                throw new NotFoundException();
            }

            return query
                .Include(r => r.Role)
                .Select(u => new UserOnlySearchDTO
                {
                    Id = u.Id,
                    Name = u.Name,
                    Surname = u.Surname,
                    Email = u.Email,
                    RoleName = u.Role.NameRole,
                    IsDelete = u.IsDeleted
                });
        }
    }
}
```

EFGetUsersCommand

```
public class EFGetUsersCommand : EFBaseCommand, IGetUsersCommand
    {
        public EFGetUsersCommand(DBContext context) : base(context)
        {
        }

        public PagedRespone<UserOnlySearchDTO> Execute(UserSearch request)
        {
            var query = _context.Users.AsQueryable();

            if (request.Name != null)
            {
                var name = request.Name.ToLower();
                query = query.Where(u => u.Name.ToLower()
                .Contains(name)
                && u.IsDeleted == false);
            }

            if (request.SurName != null)
            {
                var name = request.SurName.ToLower();
                query = query.Where(u => u.Surname.ToLower()
                .Contains(name)
                && u.IsDeleted == false);
            }

            var totalCount = query.Count();

            query = query.Skip((request.PageNumber - 1) * request.PerPage)
                .Take(request.PerPage);

            var pageCount = (int)Math.Ceiling((double)totalCount / request.PerPage);
```

```
            var response = new PagedRespone<UserOnlySearchDTO>
            {

                CurrentPage = request.PageNumber,
                TotalCount = totalCount,
                PageCount = pageCount,
                Data = query.Include(r => r.Role)
                .Select(u => new UserOnlySearchDTO
                {
                    Id = u.Id,
                    Name = u.Name,
                    Surname = u.Surname,
                    Email = u.Email,
                    RoleName = u.Role.NameRole,
                    IsDelete = u.IsDeleted
                })

            };
            return response;
        }
```

WEB APP

ImpressionController

Pretraga je moguca samo preko Content-a

```
public class ImpressionController : Controller
    {
        private IAddImpresssionCommand _addImpressionCommand;
        private IDeleteImpressionCommand _deleteImpressionCommand;
        private IGetImpressionCommand _getImpressionCommand;
        private IEditImpressionCommand _editImpressionCommand;
        private IGetImpressionsCommand _getImpressionsCommand;
        private IGetAllImpressionWebCommand _getAllImpressionCommand;
        private IGetImpressionWebCommand _getImpressionWebCommand;

        public ImpressionController(IAddImpresssionCommand addImpressionCommand,
IDeleteImpressionCommand deleteImpressionCommand, IGetImpressionCommand
getImpressionCommand, IEditImpressionCommand editImpressionCommand, IGetImpressionsCommand
getImpressionsCommand, IGetAllImpressionWebCommand getAllImpressionCommand,
IGetImpressionWebCommand getImpressionWebCommand)
        {
            _addImpressionCommand = addImpressionCommand;
            _deleteImpressionCommand = deleteImpressionCommand;
            _getImpressionCommand = getImpressionCommand;
            _editImpressionCommand = editImpressionCommand;
            _getImpressionsCommand = getImpressionsCommand;
            _getAllImpressionCommand = getAllImpressionCommand;
            _getImpressionWebCommand = getImpressionWebCommand;
        }
```

```
// GET: Impression prikaz svih
public ActionResult Index(ImpressionSearchWeb request)
{
    var lista = _getAllImpressionCommand.Execute(request);
    return View(lista);
}

// GET: Impression/Details/5
public ActionResult Details(int id)
{

    try
    {
        var dto = _getImpressionCommand.Execute(id);
        return View(dto);
    }
    catch (NotFoundException)
    {
        TempData["error"] = "Ne postoji utisak sa id koji ste uneli";
    }

    return RedirectToAction(nameof(Index));
}

// GET: Impression/Create prikaz forme
public ActionResult Create()
{
    return View();
}

// POST: Impression/Create unos jednog
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create(ImpressionDTO request)
{
    if(!ModelState.IsValid)
    {
        return View(request);
    }
    try
    {
        _addImpressionCommand.Execute(request);

        return RedirectToAction(nameof(Index));
    }
    catch(NotFoundException)
    {
        TempData["error"] = "Uneli ste id usera koji ne postoji";
    }catch(Exception)
    {
        TempData["error"] = "Serverska greska";
    }
    return View();
}

// GET: Impression/Edit/5 prikaz forme
```

```
public ActionResult Edit(int id)
{
    var obj = _getImpressionWebCommand.Execute(id);
    return View(obj);
}

// POST: Impression/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(ImpressionEditDTO request)
{
    if (!ModelState.IsValid)
    {
        return View(request);
    }
    try
    {

        _editImpressionCommand.Execute(request);

        return RedirectToAction(nameof(Index));
    }
    catch
    {
        return View();
    }
}

// GET: Category/Delete/5
public ActionResult Delete(int id)
{
    try
    {
        _deleteImpressionCommand.Execute(id);
        return RedirectToAction(nameof(Index));
    }
    catch (NotFoundException)
    {
        return NotFound();
    }
    catch (Exception)
    {
        return BadRequest("Something went wrong");
    }
}
```

GetAll

WebApp Home Privacy

Use this space to summarize your privacy and cookie use policy. Learn More. Accept

Index

Create New

Id	Content	UserId	NameSurname	
2	Ovo je nemanja napisao	2	Veca Ranisavljevic	Edit | Details | Delete
4	Radi njam njam	2	Veca Ranisavljevic	Edit | Details | Delete

© 2019 - WebApp - Privacy

Details

Details

ImpressionDTO

Id	2
Content	Ovo je nemanja napisao
UserId	2
NameSurname	Veca Ranisavljevic

Edit | Back to List

Create

Content

UserId

Create

Edit

Content

Ovo je nemanja napisao

Save

Meni Web

Pri dodavanju I izmeni obavezno je sliku uneti kao podatak. Ime hrane ne moze da bude isto pri dodavanju ili izmeni.Pretraga je moguca samo preko imena hrane

```
public class MeniSearchWeb
    {
        public string NameFood { get; set; }
    }
```

Controller

```
public class MeniController : Controller
    {
        private IAddMeniCommand _addMeniCommand;
        private IGetMeniCommand _getMeniCommand;
        private IGetMeniesCommand _getMeniesCommand;
        private IEditMeniCommand _editMeniCommand;
        private IDeleteMeniCommand _deleteMeniCommand;
        private IGetAllMeniesCommandWeb _getAllMeniesCommandWeb;
        private IGetMeniWebCommand _getOneMeniCommand;
        private IGetEditMeniCommand _getEditMeniCommand;
        private IGetWebMealsCommand _getWebMeals;

        public MeniController(IAddMeniCommand addMeniCommand, IGetMeniCommand
getMeniCommand, IGetMeniesCommand getMeniesCommand, IEditMeniCommand editMeniCommand,
IDeleteMeniCommand deleteMeniCommand, IGetAllMeniesCommandWeb getAllMeniesCommandWeb,
IGetMeniWebCommand getOneMeniCommand, IGetEditMeniCommand getEditMeniCommand,
IGetWebMealsCommand getWebMeals)
        {
```

```
        _addMeniCommand = addMeniCommand;
        _getMeniCommand = getMeniCommand;
        _getMeniesCommand = getMeniesCommand;
        _editMeniCommand = editMeniCommand;
        _deleteMeniCommand = deleteMeniCommand;
        _getAllMeniesCommandWeb = getAllMeniesCommandWeb;
        _getOneMeniCommand = getOneMeniCommand;
        _getEditMeniCommand = getEditMeniCommand;
        _getWebMeals = getWebMeals;
    }



    // GET: Meni
    public ActionResult Index(MeniSearchWeb request)
    {
        var obj = _getAllMeniesCommandWeb.Execute(request);
        return View(obj);
    }

    // GET: Meni/Details/5
    public ActionResult Details(int id)
    {
        try
        {
            var obj = _getOneMeniCommand.Execute(id);
            return View(obj);
        }catch(NotFoundException)
        {
            return NotFound();
        }

    }

    // GET: Meni/Create
    public ActionResult Create()
    {

        return View();
    }

    // POST: Meni/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create([FromForm] HttpSlikaDTO p)
    {

        var ext = Path.GetExtension(p.Image.FileName); //daje ekstenziju .jpg

        if (!FileUpload.AllowExtensions.Contains(ext))
        {
            return UnprocessableEntity("Image extension is not allowed.");
        }


        try
        {
```

```
                var newFileName = Guid.NewGuid().ToString() + "_" +
p.Image.FileName;//unique za ime fajla

                var filePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot",
"uploads", newFileName);

                p.Image.CopyTo(new FileStream(filePath, FileMode.Create));


                var dto = new MeniAddDTO
                {
                    NameFood = p.NameFood,
                    Ingradiant = p.Ingradiant,
                    FileName = newFileName,
                    Price = p.Price,
                    MealId = p.MealId
                };

                _addMeniCommand.Execute(dto);
                return RedirectToAction(nameof(Index));
            }
            catch (NotFoundException)
            {

                TempData["error"] = "Meal of food not found";
            }
            catch (AlredyExistException)
            {
                TempData["error"] = "Name of food alredy exist";

            }
            catch (Exception)
            {

                return StatusCode(500, "Server error try later");
            }
            return RedirectToAction(nameof(Index));
        }

        // GET: Meni/Edit/5
        public ActionResult Edit(int id, ClassForNullObj request)
        {
            var obj = _getEditMeniCommand.Execute(id);

            return View(obj);
        }

        // POST: Meni/Edit/5
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit(int id,[FromForm] HttpEditMeni p)
        {
            var ext = Path.GetExtension(p.Image.FileName); //daje ekstenziju .jpg

            if (!FileUpload.AllowExtensions.Contains(ext))
            {
                return UnprocessableEntity("Image extension is not allowed.");
```

```
            }

            try
            {
                var newFileName = Guid.NewGuid().ToString() + "_" +
p.Image.FileName;//unique za ime fajla

                var filePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot",
"uploads", newFileName);

                p.Image.CopyTo(new FileStream(filePath, FileMode.Create));


                var dto = new MeniAddDTO
                {
                    Id = p.Id,
                    NameFood = p.NameFood,
                    Ingradiant = p.Ingradiant,
                    FileName = newFileName,
                    Price = p.Price,
                    MealId = p.MealId
                };

                _editMeniCommand.Execute(dto);
                return RedirectToAction(nameof(Index));
            }
            catch (NotFoundException)
            {
                return StatusCode(404, "Meal id not found or id of meni");
            }
            catch (AlredyExistException)
            {
                return StatusCode(422, "Name of food alredy exist");
            }
            catch (Exception)
            {
                return StatusCode(500, "Server error try later");
            }
        }

        // GET: Meni/Delete/5
        public ActionResult Delete(int id)
        {
            try
            {
                _deleteMeniCommand.Execute(id);
                return RedirectToAction(nameof(Index));
            }catch
            {
                TempData["error"] = "Ne postoji objekat koji trazite";
                return RedirectToAction(nameof(Index));
            }
        }


    }
}
```

Index

Create New

Id	NameFood	Ingradiant	Price	FileName	MealName	Alt	
4	NjamaNjam	jaja, sir, paprika	50		Rucak	NjamaNjam	Edit | Details | Delete

Edit

Edit

HttpEditMeni

NameFood

NjamaNjam

Ingradiant

jaja, sir, paprika

Price

50

Image

Преглед... Није одабрана датотека.

MealId

2

Save

Back to List

Details

Details

MeniGetDTO

Id	4
NameFood	NjamaNjam
Ingradiant	jaja, sir, paprika
Price	50
FileName	
MealName	Rucak
Alt	NjamaNjam

Edit | Back to List