AKADEMIJA TEHNIČKO-UMETNIČKIH STRUKOVNIH STUDIJA BEOGRAD
ODSEK VISOKA ŠKOLA ZA INFORMACIONE I KOMUNIKACIONE TEHNOLOGIJE

ZAVRŠNI RAD IZ PREDMETA:

Web Programiranje ASP

Web aplikacija za upravljanje digitalnom zajednicom kompanije upotrebom .NET Core framework-a i React.js sa TypeScript-om

Mentor:
Mr Milanko Kragović, dipl. Inž.

Kandidat:
Miloš Tanasković 148/15

Beograd, 2022.

AKADEMIJA TEHNIČKO-UMETNIČKIH STRUKOVNIH STUDIJA BEOGRAD
ODSEK VISOKA ŠKOLA ZA INFORMACIONE I KOMUNIKACIONE TEHNOLOGIJE

Internet tehnologije, Web programiranje

Predmet: Web Programiranje ASP

Tema: Web aplikacija za upravljanje digitalnom zajednicom kompanije upotrebom .NET Core framework-a i React.js sa TypeScript-om

Mentor:
Mr Milanko Kragović, dipl. Inž.

Kandidat:
Miloš Tanasković 148/15

Beograd, 2022.

Sadržaj

Beograd, 2022.

Beograd, 2022.

1. Uvod:

Web aplikacija za upravljanje digitalnom zajednicom kompanije ima za cilj da pomogne zaposlenima, korisnicima i pratiocima kompanije u njihovom zbližavanju, boljem upoznavanju i međusobnoj interakciji organizovanjem interesatnih događaja u digitalnoj formi.
Platforma je organizovana u vidu društvene mreže tako da svaki registrovani korisnik ima priliku da prati druge korisnike u formi “onlajn prijatelja”, kao i da menja, komentariše objave ili dogadjaje u kome ta osoba ucestvuje.
Web aplikacija rađena je u .NET Core framework-a i React.js sa TypeScript-om. Aplikacija je zamišljena tako da je celokupna logika izmeštena na API(ASP.NET Core) tako da može biti korišćena od strane bilo koje druge aplikacije u svrhu upravljanja digitalnom zajednicom kompanije ili kao neki vid društvene mreže.
Osim toga, za konkretnu web aplikaciju moguce je napraviti aplikacije za razlicite uređaje, kao što su mobilni telefoni i desktop računari.
Ideja ove aplikacije da bude modularna i da bude dostupna na što više razlicitih tehnologija.

2. Funkcionalnost Aplikacije

Funkcionalnosti koje ova aplikacija pruža su:

- Autentikacija Korisnika (Login, Logout, Register).
- Izlistavanje Događaja(Activities).
- CRUD operacije za Događaj(Activity).
- CRUD operacije za Korisnika.
- Videti više informacija o trenutnom događaju.
- Mogucnost pridruživanja tom događaju.
- Mogucnost otkazivanja prisutnosti izabranom događaju.
- Mogucnost pracenja koliko korsnika ide na taj događaj i koji od navedenih clanova prate tebe ili ti pratiš njih.
- Mogucnost ostavljanja komentara i simultanog dopisivanja sa ulogovanim clanovima.
- Kreiranje novog događaja kao i mogucnost izmene tog događaja
- Filtriranje po svim događajima, koje je korisnik kreirao, kao i one na koje ulogovani korisnik ide.

- Filtriranje po datumu.
- Implementirano beskonacno pomeranje(infinite scrolling).
- Mogucnost isecanja i podesavanja profilne slike po želji korisnika.
- Mogucnost odabira profilne slike iz kolekcije slika.
- Mogucnost pracenja(Following and Unfollowing) drugog korisnika.

Neautorizovani korisnik mora da se registruje kako bi mogao da pristupi aplikaciji. Svaki ulogovan korisnik ima mogucnost da vidi sve izlistane događaje na pocetnoj stranici aplikacije (/activities). Na pocetnoj stranici u navigacionom delu korisnik ima mogucnost da izabere dali želi da kreira novi događaj, vidi svoj profil, ili se vrati na pecetnu stranicu sa svim dostupnim događajima.

Na pocetnoj stranici (/activities) korisnik može videti osnove informacije o nekom događaju kao sto su:
1) Ko je kreirao događaj.
2) Na koji događaj on ide.
3) Kog datuma je prestjeci događaj, ili kada je bio.
4) Koliko korisnika ide na specificni događaj
5) Ko od navedenih korisnika prati ulogovanog korisnika
6) Osnovne informacije o svim korisnicima koji su se prijavili za taj događaj
7) I mogucnost pregleda specificnog događaja i uvid u više informacija o izabranom događaju. (/activities/:id)
8) Na levoj strani se nalazi sekcija za brzu pretragu dostupnih događaja kao sto je (Svi događaji, Događaji na koje korisnik ide, Događaji koje je korisnik kreirao, i po datumu kada se održavaju)
9) Beskonacno pomeranje(infinite scrolling) ili paginacija ispod poslednjeg događaja na trenutnoj strani.

Kada se izabere događaj odlazi se na stranicu (/activities/:id) na kojoj se može procitati sve o tom događaju, a specificne funkcionalnsti na ovoj stranici su:
1) Mogucnost izmene kreiranog događaja.
2) Mogucnost (Cancel Activity) kao i (Re-activate Activty)
3) Prikljucivanje događaju.
4) Izlazak iz događaja
5) Mogucnost ostavljanja komentara kao i simultanog dopisivanje sa drugim korisnikom.
6) Pracenje koliko korisnika je yainteresovano za taj događaj.

Svaki ulogovan korisnik ima mogucnost da kreira nov događaj i na stranici (/createActivity) se nalazi forma u kojoj korisnik ima mogucnost da unese podatke o prestojecem događaju kao sto su:
1) Detalji o događaju.
2) Detalji o lokaciji.

Profilna stranica (/profiles/imeKorisnika) omogucava korisniku da:

1) Da ima uvid koliko korisnika prati taj prifil i koliko korisnika taj profil prati.
2) Mogucnost dodavanja i izmene licnih podataka korisnika.
3) Dodavanje, izmena profilne slike
4) Uvid u sve događaje ya koje je taj profil vezan.

3. Radno okruženje

Tehnologije:

- ASP.NET Core - Esecijalna struktura koja pomaže u razvoju C# aplikacija.
- JavaScript
- TypeScript
- HTML5
- CSS3
- React.js - Esecijalna struktura koja pomaže u razvoju JavaScript aplikacija.
- Semantic UI React
- PostgreSQL - Sistem za upravljanje relacionim bazama podataka.

Koristeći ASP.NET Core „framework“ uspešno je napravljen „Web API“ koji nam omogućava da logički obradimo podatke, čuvamo potrebne podatke u bazi podataka i da te iste podatke učitavamo ili modifikujemo na način koji odgovara našoj web aplikaciji.

Koristeći JavaScript klijenski jezik i njegov „framework“ React.js uspešno je napravljena klijenska „web browser“ aplikacija koja nam omogucava vizuelizaciju obrađenih podataka na našem „Web API-ju“. Ova aplikacija služi za slanje podataka na API i ya njihov prikaz prilikom odgovora. Osim JavaScript, React u ovoj aplikaciji se koristi još TypeScript kako bi nam omogucio strogo tiptiziran JavaScript, HTML5, CSS3 i Semantic UI React kako bi nam „browser“ omogucio graficki interpretaciju podataka na odgovarajuci naćin.

Korišćen je PostgreSQL relaciona baza podataka koja nam omogucava da postavljamo određena ogranicenja i na taj način je upravljanje podacima bezbednije i organiyovanije.

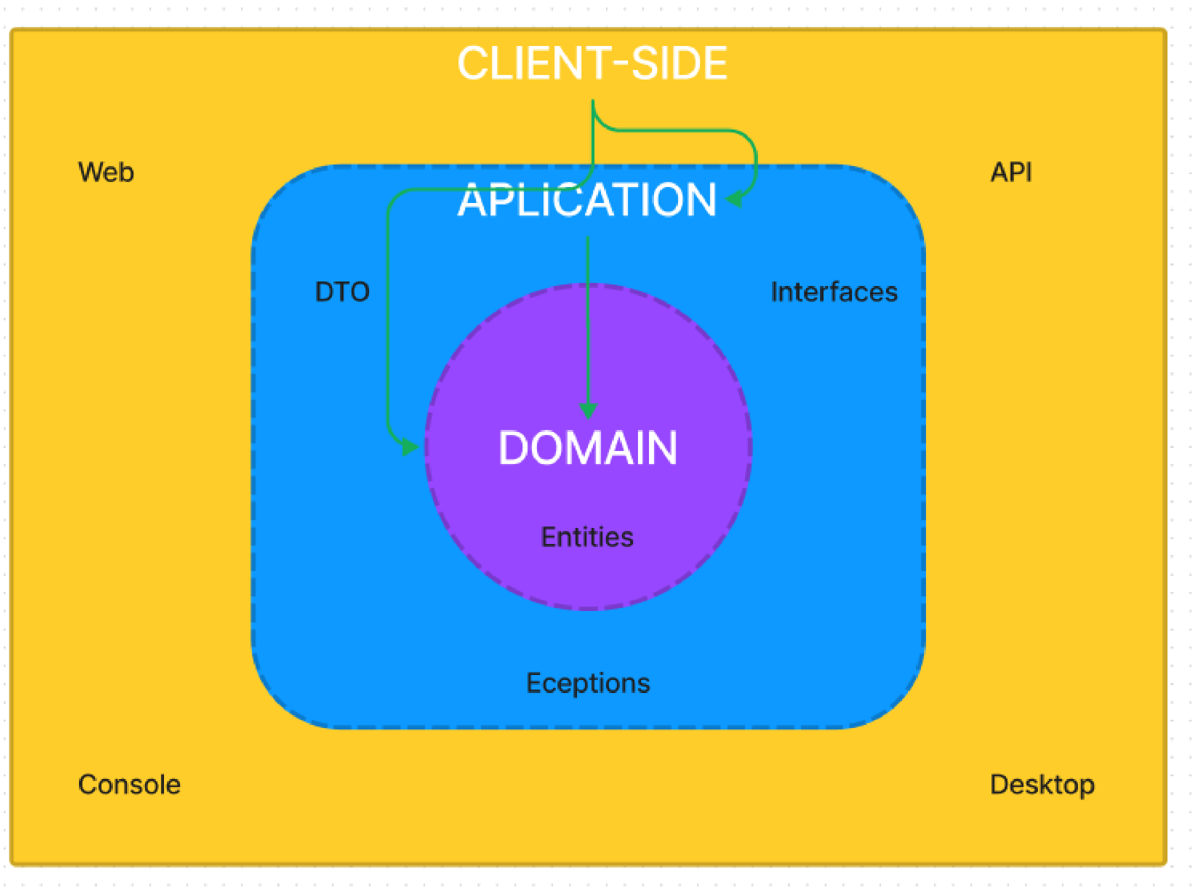
Projekat se sastoje od nekoliko fizicki odvojenih aplikacija koje međusobno komuniciraju preko HTTP protokola.

4. Arhitektura Aplikacije

Ideja ovog projekta je da se napravi arhitektura koja ce omoguciti modularnost, odnosto kada se napravi promena da se ta promena ne odnosi na kreiranje u potponosti novog „software“, već da možemo bez ikakvih problema menjati naš „software“.
Trenutno u projektu imamo šest aplikacija, odnosno šest projekata u „solution“.

Dobro dizajnirana arhitektura „software“ će nam omoguciti:

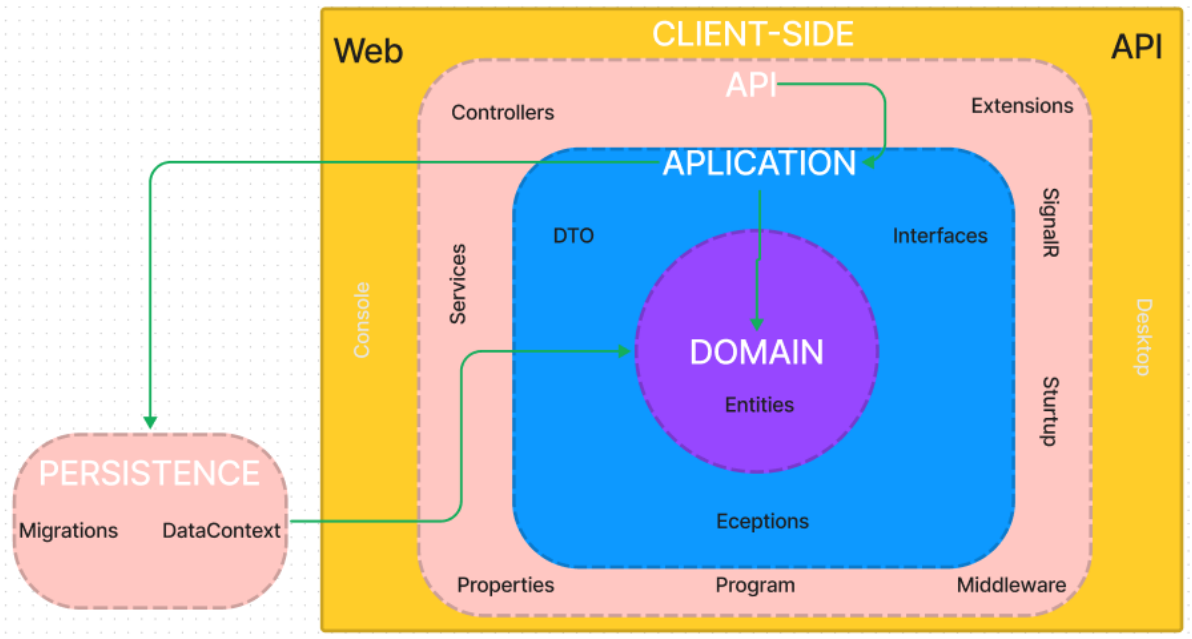
- Modularno razdvajanje vecih celina.
- Neponavljanje istog koda na više mesta u rzlicitim projektima
- Slojevito razdvajanje „software“ u manje, funkcionalnije deleove.
- Mogucnost promene razlicitih provajdera baza podataka
- Dodavanje novog sloja u „software“ veoma lako.
- Mogucnost lakseg pravljenja izmena.



Slika 4.1. Prikaz dobro ogranizovane software-ske arhitekture

- DOMAIN - Najvažniji deo aplikacije koji služi za cuvanje „Entities“
- Entities - Domenski objekti
- APPLICATION - Aplikacija gde cemo smetiti poslovno logika našeg „software“.

- Interfaces - daju nam mogucnost kreiranje apstraktnih klasa.
- Exception - mesto gde cemo hvatati izuzetke.
- Dto - Objekti za prenos.
- CLEINT-SIDE - Web, Mobile, Desktop, Console, Api…



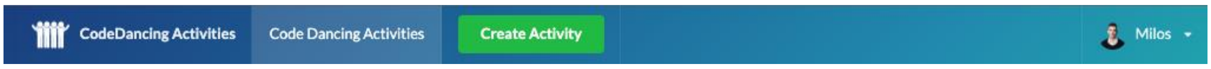
Slika 4.2. Prikaz dizajnirane arhitekture aplikacije „codedancingactivities"

5. Stranice

Web aplikacija kao socijalna društvena mreza omogućava korisnicima da na interaktivan naćin mogu da postignu svoje ciljeve koristeći razlicite web stranice koje ova aplikacija nudi.

5.1 NavBar Komponenta

Ova kompoenta služi da korisnik ima mogucnost bržek poyicioniranja i pronalaženja željene stranice. Komponenta se pojavljuje kao deo „Header" sekcije na svim stranicama aplikacije.

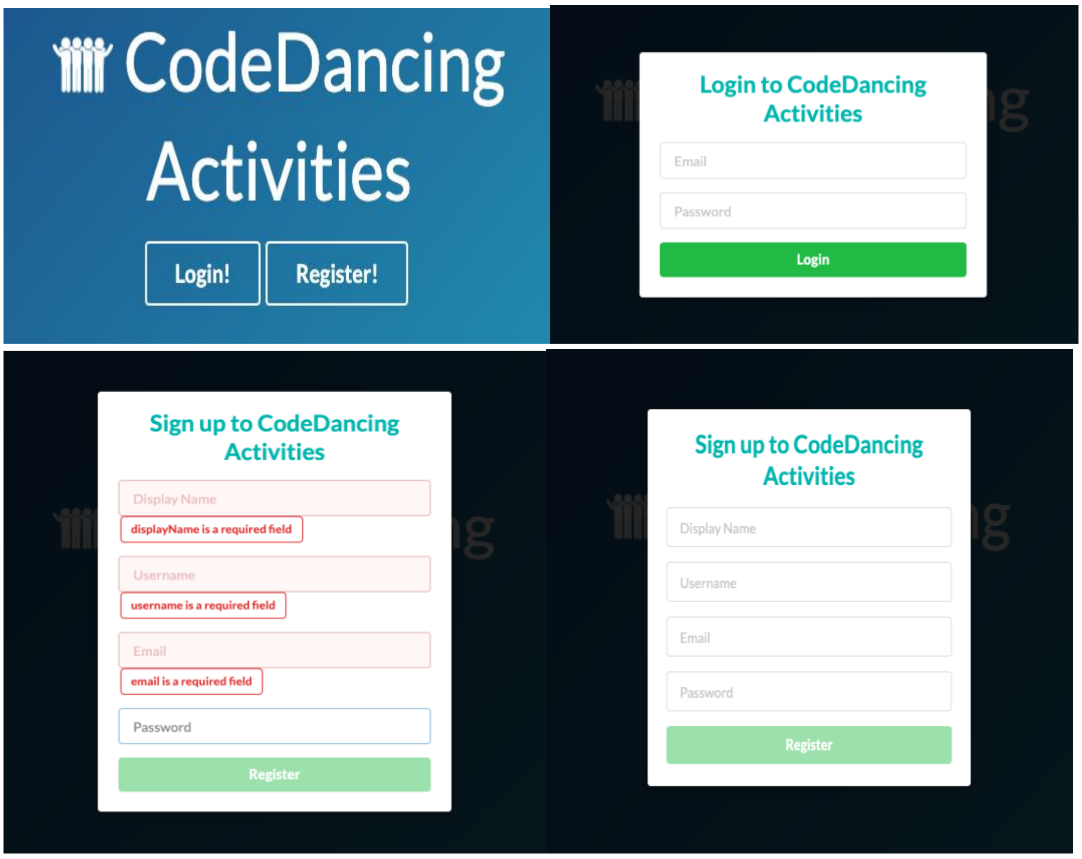


Slika 5.1. Prikaz NavBar sekcije

5.2 Prijava i Registracija

Ova stranica služi da korisnik unese svoje kredencijale i da nakon autentifikacije korisnik može da koristi aplikaciju.

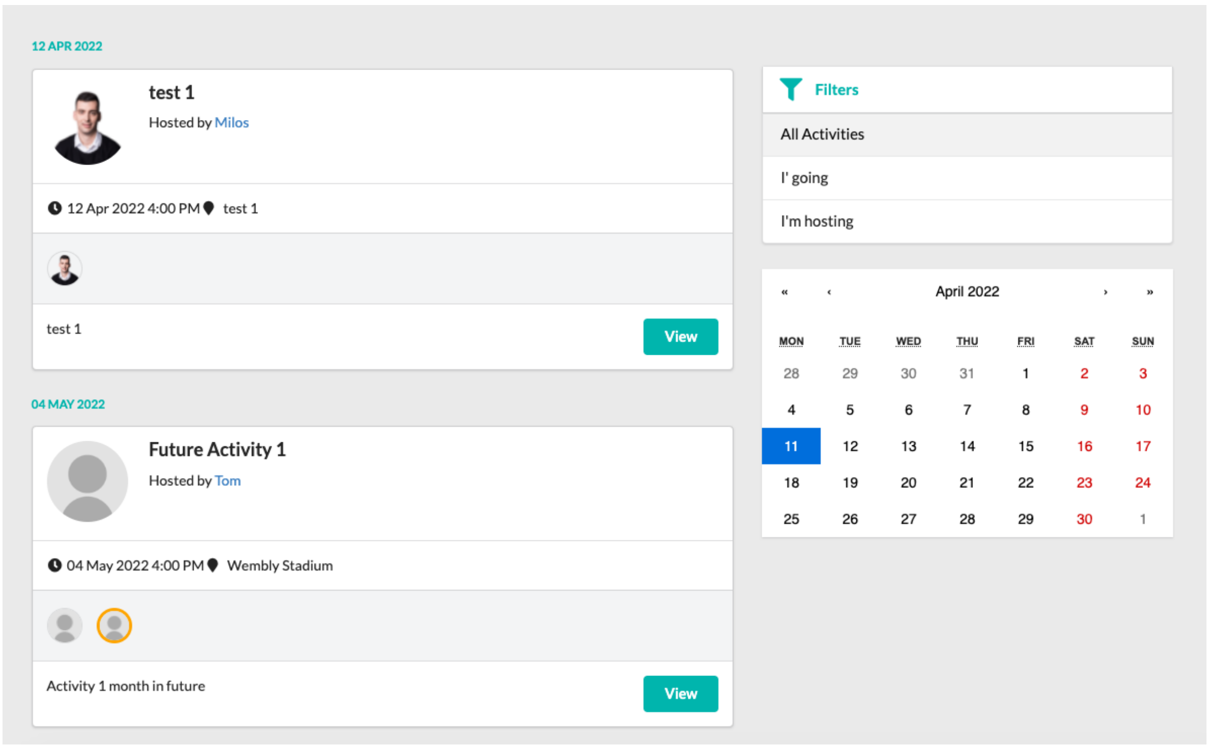
Ukoliko nije registrovan, potrebno je prvo da se registruje da bi nakon toga mogao da se autentifikuje.


CodeDancing Activities
Login!
Register!
Login to CodeDancing Activities
Email
Password
Login
Sign up to CodeDancing Activities
Display Name
displayName is a required field
Username
username is a required field
Email
email is a required field
Password
Register
Sign up to CodeDancing Activities
Display Name
Username
Email
Password
Register

Slika 5.2. Prikaz Login i Register forme

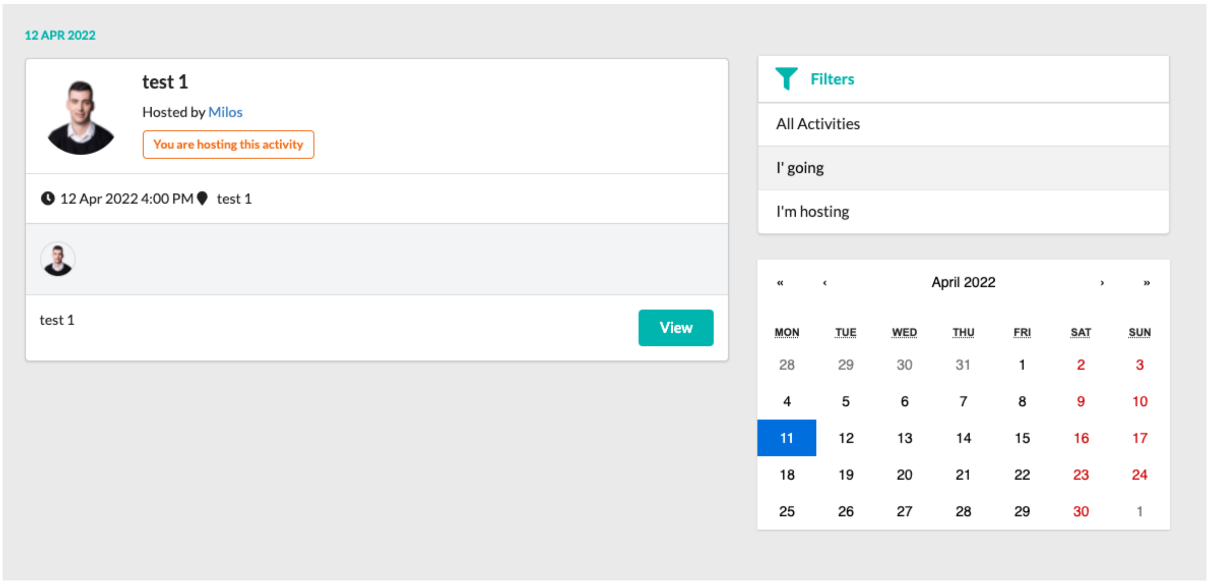
5.3 Izlistavanje događaja

Ova stranica omogućava korisniku da ima uvid u svaki događaj koji je kreiran od strane njega ili nekog drugog korisnika, sortira kao i da vidi pojedinacan događaj



MON	TUE	WED	THU	FRI	SAT	SUN
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1

5.3.1. Prikaz stranice sa svim događajima

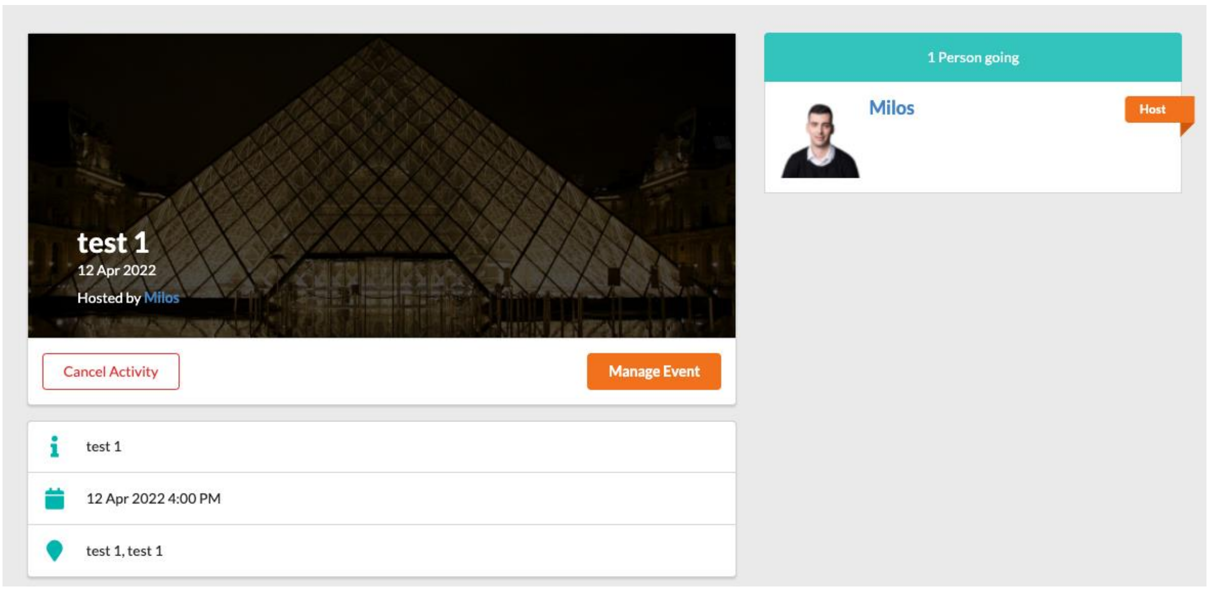


MON	TUE	WED	THU	FRI	SAT	SUN
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1

5.3.2. Prikaz stranice sa sortiranim podacima

5.4 Prikaz pojedinacnog događaja

Ova stranica omogućava korisniku da ima uvid u specifican događaj koji je izabrao. Stranica korisniku omogucava bolji uvid u izabrani događaj sa mogucnosti izmene, pridruzivanje, poništavanjem, reaktivaciji kao i simultanom dopisivanju sa drugim korisnikom iste aplikacije. U svakom trenutku korisnik može pratiti ko je zainteresovan za događaj.

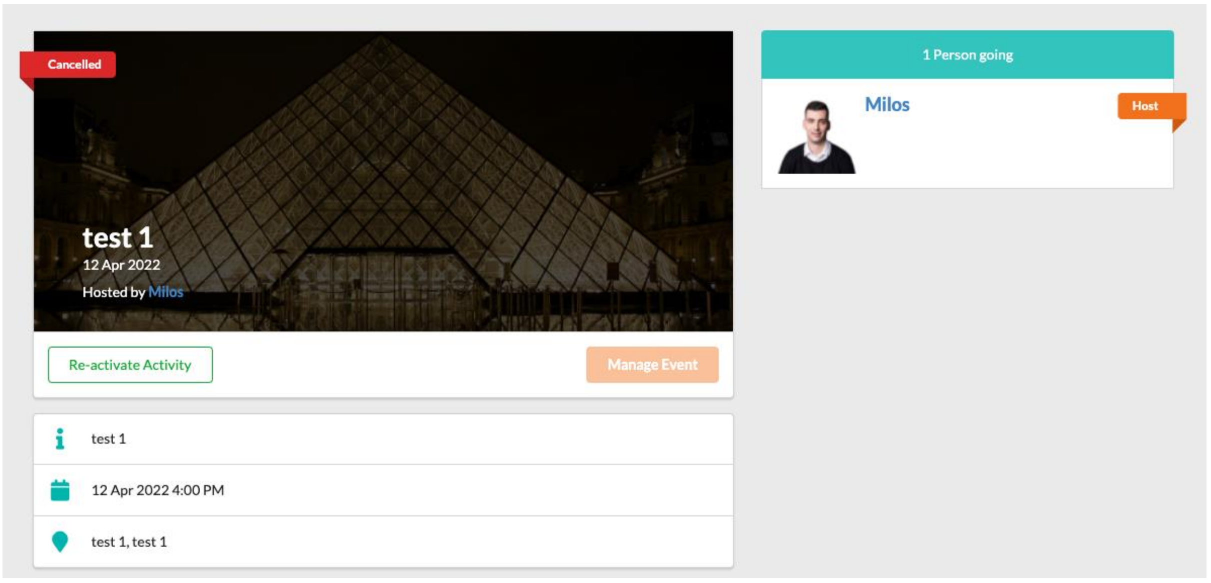


test 1

12 Apr 2022 4:00 PM

test 1, test 1

Slika 5.4.1. Prikaz pojedinacnog događaja sa mogucnosti izmene

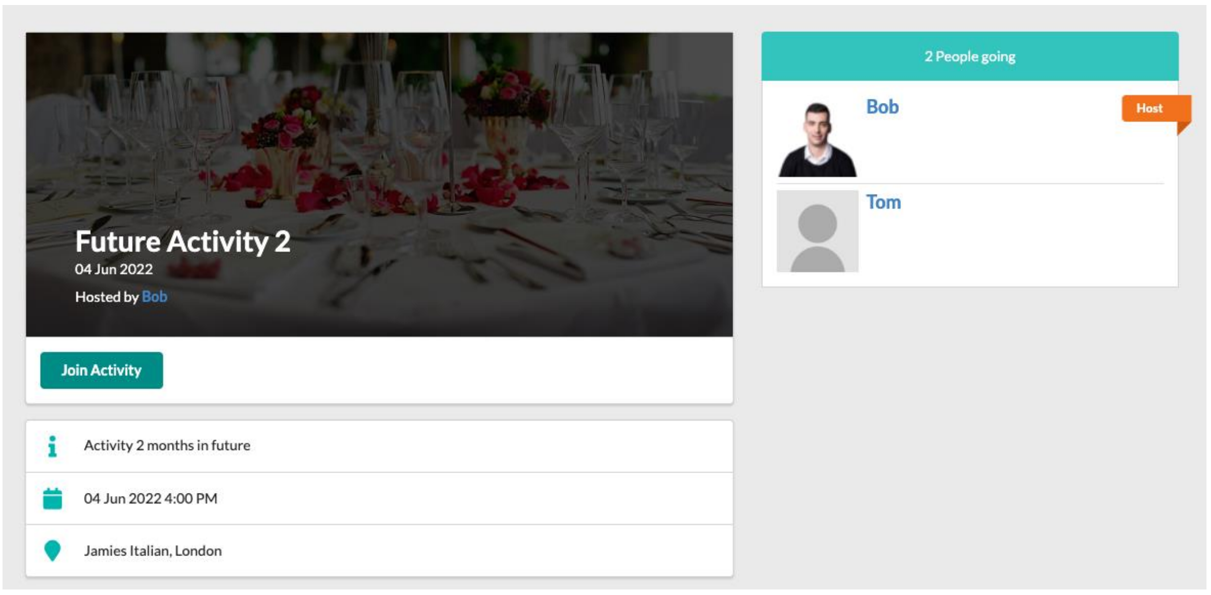


test 1

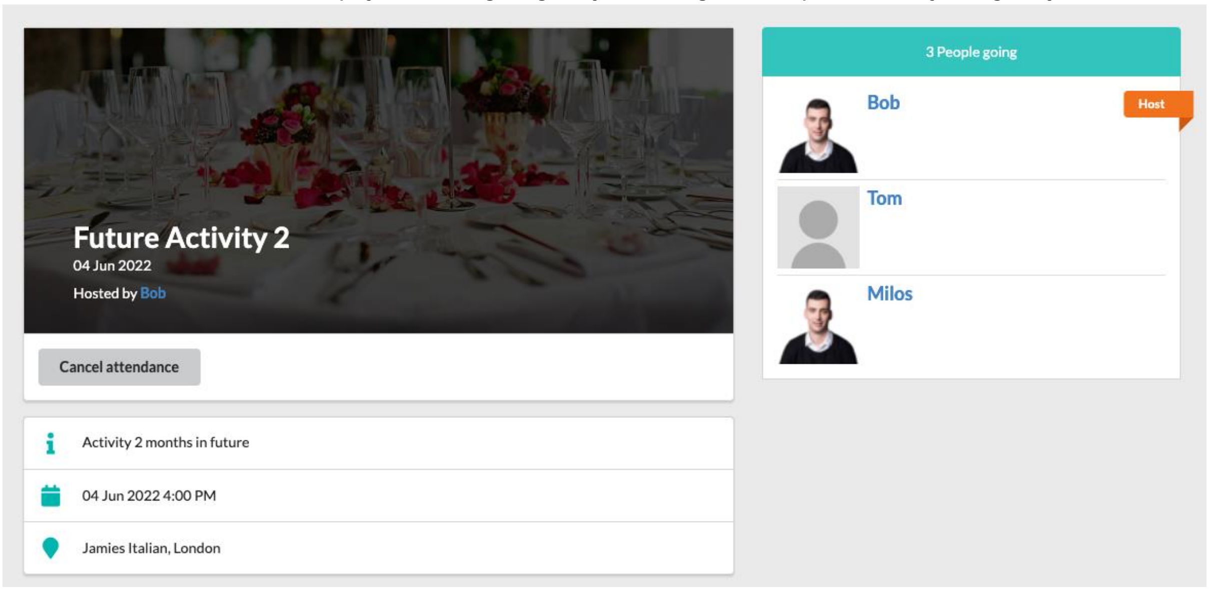
12 Apr 2022 4:00 PM

test 1, test 1

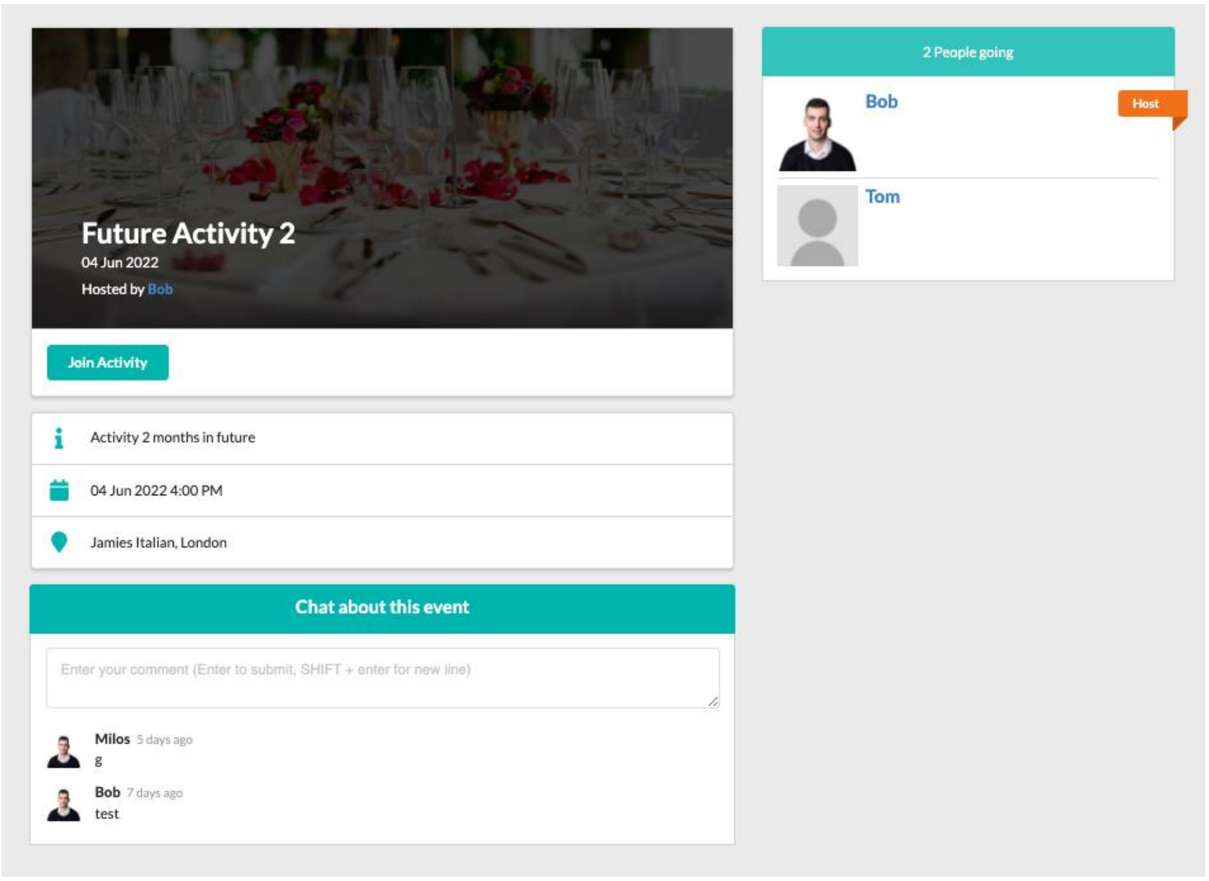
Slika 5.4.2. Prikaz pojedinacnog događaja sa obnovom događaja



Slika 5.4.3. Prikaz pojedinacnog događaja sa mogucnosti pridruzivanja događaju



Slika 5.4.4. Prikaz pojedinacnog događaja sa mogucnosti otkazivanja događaja


Future Activity 2
04 Jun 2022
Hosted by Bob

Join Activity

Activity 2 months in future
04 Jun 2022 4:00 PM
Jamies Italian, London

Chat about this event

Enter your comment (Enter to submit, SHIFT + enter for new line)

Milos 5 days ago
g

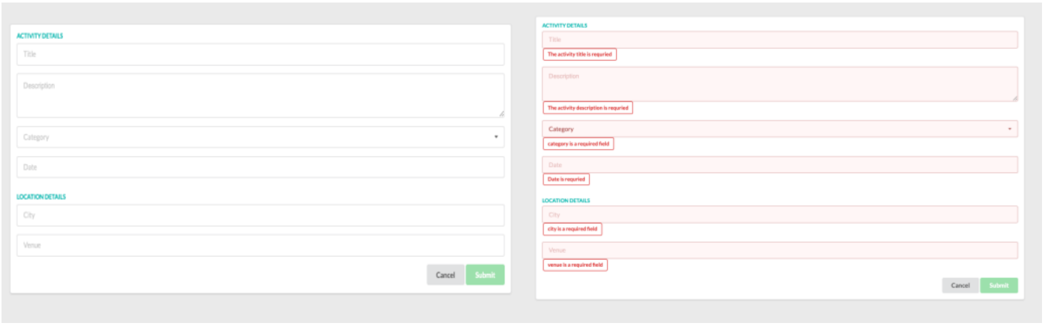
Bob 7 days ago
test

2 People going
Bob — Host
Tom

Slika 5.4.5. Prikaz pojedinacnog događaja sa mogucnosti razmene poruka

5.5 Kreiranje novog događaja

Ova stranica omogucava korisniku da kreira događaj.

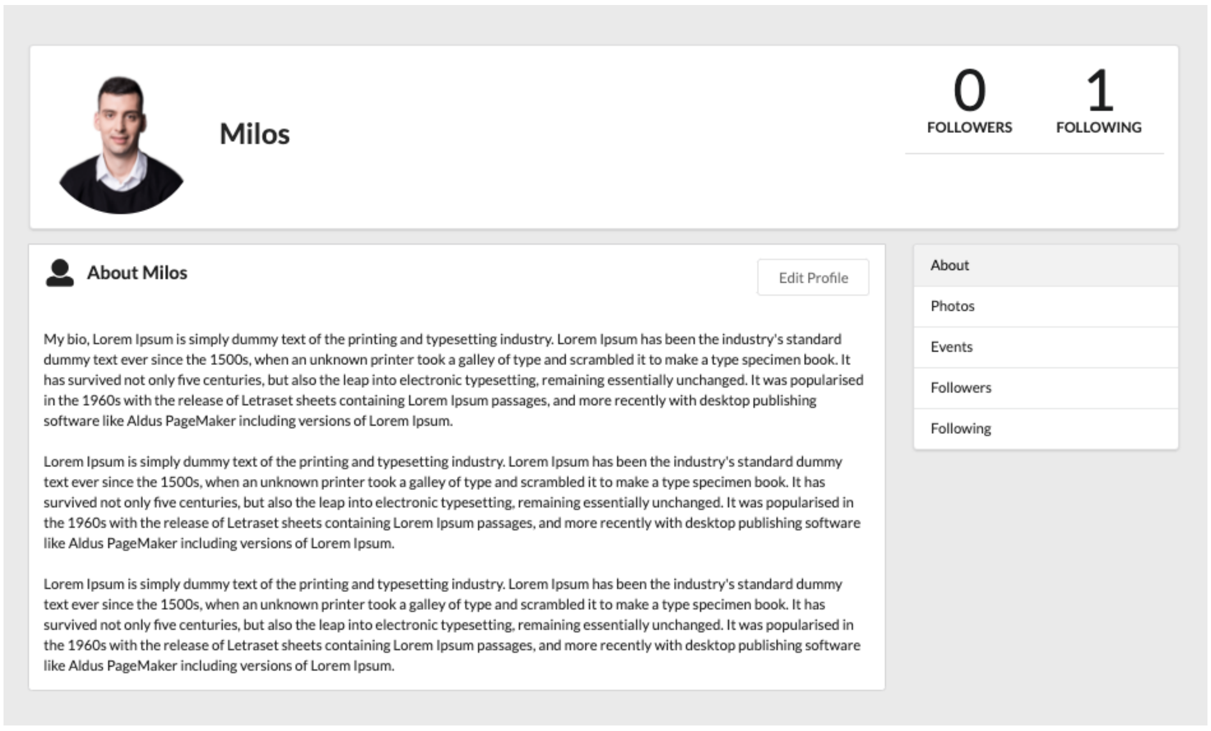

ACTIVITY DETAILS
Title
Description
Category
Date
LOCATION DETAILS
City
Venue
Cancel Submit

ACTIVITY DETAILS
Title
The activity title is requried
Description
The activity description is requried
Category
category is a required field
Date
Date is requried
LOCATION DETAILS
City
city is a required field
Venue
venue is a required field
Cancel Submit

Slika 5.5.1. Prikaz stranice za kreiranje novog događaja

5.6 Uvid u korisnicki profil

Ova stranica omogucava korisniku da ima uvid u svoj profil. Mogucnost dodavanja, izmene svoje biografije, slike za profilnu sliku. Prikaz korisnikovih događaja i pratilaca.

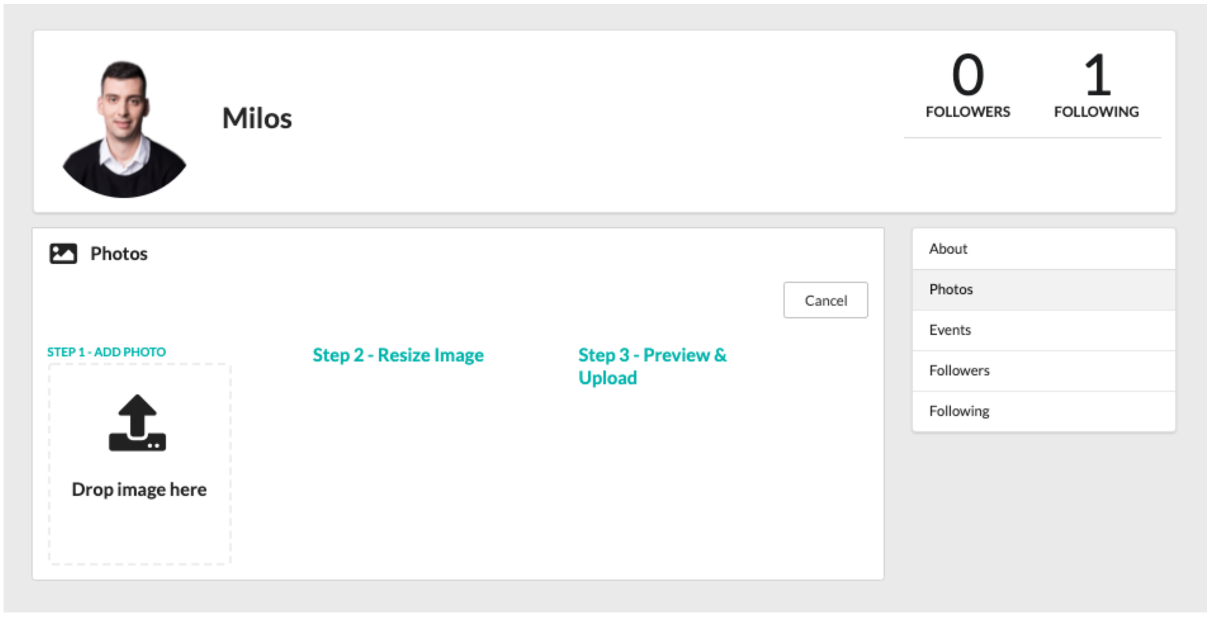


My bio, Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

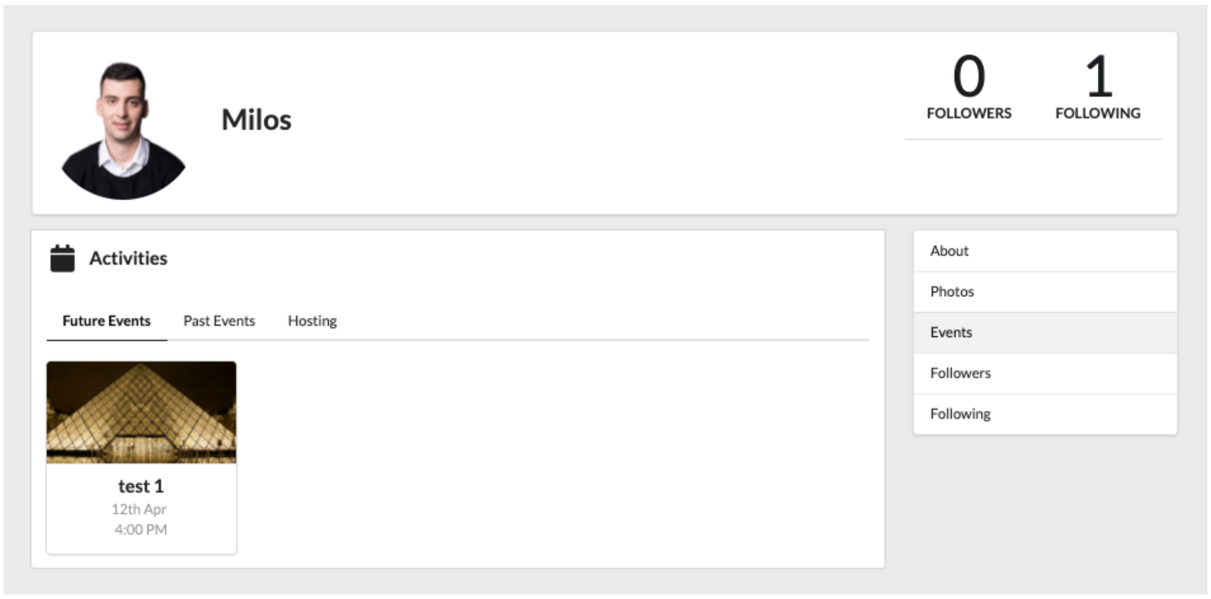
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Slika 5.6.1. Prikaz korisničkog prifila

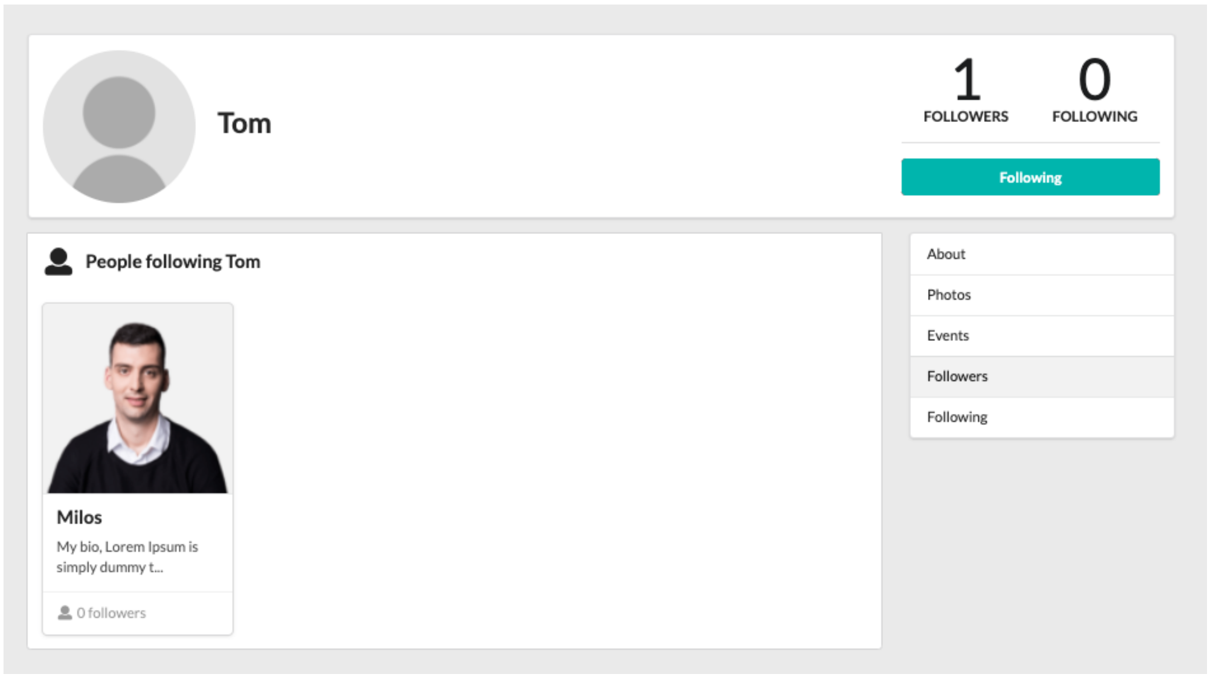


Slika 5.6.2. Prikaz korisničkog prifila sa mogucnosti dodavanje profilne slike

Beograd, 2022.



Slika 5.6.3. Prikaz korisničkog prifila sa mogucnosti uvida u događaje



Slika 5.6.4. Prikaz korisničkog prifila sa mogucnosti uvida u pratioce

6. Back-End Kodovi

Kodovi koji su korišćeni za ovu aplikaciju se izvršavaju na serveru. Ovi kodovi kao strukturni deo aplikacije koriste modele i kontrolere a ideja je da se logički celine aplikacije razdele u mikroservise kao što će biti prikazano u nastavku.

6.1. API Sloj

6.1.1. Kontroleri

Prilikom izrade aplikacije zamišljeno da kontroleri imaju ulogu usmeravanja podataka i validnost poslatih podataka sa klijentskog dela, preusmeravanje u određeni servis kao i vraćanje odgovara o određenom formatu.(JSON)

6.1.1.1. AccountController

```
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using API.DTOs;
using API.Services;
using Domain;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace API.Controllers
{
   [AllowAnonymous]
   [ApiController]
   [Route("api/[controller]")]
   public class AccountController : ControllerBase
   {
       private readonly UserManager<AppUser> _userManager;
       private readonly SignInManager<AppUser> _signInManager;
       private readonly TokenService _tokenService;
       public AccountController(UserManager<AppUser> userManager,
SignInManager<AppUser> signInManager, TokenService tokenService)
       {
           _tokenService = tokenService;
           _signInManager = signInManager;
           _userManager = userManager;
       }

       [HttpPost("login")]
       public async Task<ActionResult<UserDto>> Login(LoginDto loginDto)
       {
           var user = await _userManager.Users.Include(p => p.Photos)
               .FirstOrDefaultAsync(x => x.Email == loginDto.Email);
```

```
            if (user == null) return Unauthorized();

            var result = await _signInManager.CheckPasswordSignInAsync(user,
loginDto.Password, false);

            if (result.Succeeded)
            {
                return CreateUserObject(user);
            }

            return Unauthorized();
        }

        [HttpPost("register")]
        public async Task<ActionResult<UserDto>> Register(RegisterDto
registerDto)
        {
            if (await _userManager.Users.AnyAsync(x => x.Email ==
registerDto.Email))
            {
                ModelState.AddModelError("email", "Email taken");
                return ValidationProblem();
            }

            if (await _userManager.Users.AnyAsync(x => x.UserName ==
registerDto.Username))
            {
                ModelState.AddModelError("username", "Username taken");
                return ValidationProblem();
            }

            var user = new AppUser{
                DisplayName = registerDto.DisplayName,
                Email = registerDto.Email,
                UserName = registerDto.Email
            };

            var result = await _userManager.CreateAsync(user,
registerDto.Password);

            if (result.Succeeded)
            {
                return CreateUserObject(user);
            }

            return BadRequest("Problem registering user");
        }

        [Authorize]
        [HttpGet]
        public async Task<ActionResult<UserDto>> GetCurrentUser()
        {
            var user = await _userManager.Users.Include(p => p.Photos)
                .FirstOrDefaultAsync(x => x.Email ==
User.FindFirstValue(ClaimTypes.Email));

            return CreateUserObject(user);
        }

        private UserDto CreateUserObject(AppUser user)
        {
            return new UserDto
                {
                    DisplayName = user.DisplayName,
                    Image = user?.Photos?.FirstOrDefault(x => x.IsMain)?.Url,
                    Token = _tokenService.CreateToken(user),
                    Username = user.UserName
                };
        }
    }
}
```

6.1.1.2. ActivitiesController

```
using System;
using System.Threading.Tasks;
using Application.Activities;
using Application.Core;
using Domain;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    [AllowAnonymous]
    public class ActivitiesController : BaseApiController
    {
        [HttpGet] // /activities
        public async Task<IActionResult>
GetActivities([FromQuery]ActivityParams param)
        {
            return HandlePageResult(await Mediator.Send(new List.Query{Params
= param}));
        }

        [HttpGet("{id}")] // /activities/id
        public async Task<IActionResult> GetActivity(Guid id)
        {
            return HandleResult(await Mediator.Send(new Details.Query{Id =
id}));
        }

        [HttpPost] // /activities
        public async Task<IActionResult> CreateActivty(Activity activity)
        {
            return HandleResult(await Mediator.Send(new
Create.Command{Activity = activity}));
        }

        [Authorize(Policy = "IsActivityHost")]
        [HttpPut("{id}")] // /activities/id
        public async Task<IActionResult> EditActivity(Guid id, Activity
activity)
        {
            activity.Id = id;

            return HandleResult(await Mediator.Send(new Edit.Command{Activity
= activity}));
        }

        [Authorize(Policy = "IsActivityHost")]
        [HttpDelete("{id}")] // /activities/id
        public async Task<IActionResult> DeleteActivity(Guid id)
        {
            return HandleResult(await Mediator.Send(new Delete.Command{Id =
id}));
        }

        [HttpPost("{id}/attend")]
        public async Task<IActionResult> Attend(Guid id)
        {
            return HandleResult(await Mediator.Send(new
UpdateAttendance.Command{Id = id}));
        }
    }
}
```

6.1.1.3. BaseApiController

```
using API.Extensions;
using Application.Core;
using MediatR;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.DependencyInjection;

namespace API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class BaseApiController : ControllerBase
    {
        private IMediator _mediator;

        protected IMediator Mediator => _mediator ??=
HttpContext.RequestServices.GetService<IMediator>();

        protected ActionResult HandleResult<T>(Result<T> result)
        {
            if(result == null) return NotFound();

            if(result.IsSuccess && result.Value != null) return
Ok(result.Value);

            if(result.IsSuccess && result.Value == null) return NotFound();

            return BadRequest(result.Error);
        }

        protected ActionResult HandlePageResult<T>(Result<PageList<T>>
result)
        {
            if(result == null) return NotFound();

            if(result.IsSuccess && result.Value != null)
            {
                Response.AddPaginationHeader(result.Value.CurrentPage, 
result.Value.PageSize, result.Value.TotalCount, result.Value.TotalPages);
                return Ok(result.Value);
            }

            if(result.IsSuccess && result.Value == null) return NotFound();

            return BadRequest(result.Error);
        }
    }
}
```

6.1.1.4. BuggyController

```
using System;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
   public class BuggyController : BaseApiController
   {
        [HttpGet("not-found")]
        public ActionResult GetNotFound()
        {
            return NotFound();
        }

        [HttpGet("bad-request")]
        public ActionResult GetBadRequest()
        {
            return BadRequest("This is a bad request");
        }

        [HttpGet("server-error")]
        public ActionResult GetServerError()
        {
            throw new Exception("This is a server error");
        }

        [HttpGet("unauthorised")]
        public ActionResult GetUnauthorised()
        {
            return Unauthorized();
        }
    }
}
```

6.1.1.5. FallbackController

```
using System.IO;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    [AllowAnonymous]
    public class FallbackController : Controller
    {
        public IActionResult Index()
        {
            return PhysicalFile(Path.Combine(Directory.GetCurrentDirectory(),
"wwwroot", "index.html"), "text/HTML");
        }
    }
}
```

6.1.1.6. FollowController

```
using System.Threading.Tasks;
using Application.Followers;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    public class FollowController : BaseApiController
    {
        [HttpPost("{username}")]
        public async Task<IActionResult> Follow(string username)
        {
            return HandleResult(await Mediator.Send(new
FollowToggle.Command{TargerUsername = username}));
        }

        [HttpGet("{username}")]
        public async Task<IActionResult> GetFollowings(string username,
string predicate)
        {
            return HandleResult(await Mediator.Send(new List.Query{Username =
username, Predicate = predicate}));
        }
    }
}
```

6.1.1.7. PhotosController

```
using System.Threading.Tasks;
using Application.Photos;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    public class PhotosController : BaseApiController
    {
        [HttpPost]
        public async Task<IActionResult> Add([FromForm] Add.Command command)
        {
            return HandleResult(await Mediator.Send(command));
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> Delete(string id)
        {
            return HandleResult(await Mediator.Send(new Delete.Command{Id =
id}));
        }

        [HttpPost("{id}/setMain")]
        public async Task<IActionResult> SetMain(string id)
        {
            return HandleResult(await Mediator.Send(new SetMain.Command{Id =
id}));
        }

    }
}
```

6.1.1.8. ProfilesController

```
using System.Threading.Tasks;
using Application.Profiles;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
   public class ProfilesController : BaseApiController
   {
       [HttpGet("{username}")]
       public async Task<IActionResult> GetProfile(string username)
       {
           return HandleResult(await Mediator.Send(new
Details.Query{Username = username}));
       }

       [HttpPut]
       public async Task<IActionResult> Edit(Edit.Command command)
       {
           return HandleResult(await Mediator.Send(command));
       }

       [HttpGet("{username}/activities")]
       public async Task<IActionResult> GetUserActivities(string username,
string predicate)
       {
           return HandleResult(await Mediator.Send(new
ListActivities.Query{Username = username, Predicate = predicate}));
       }
   }
}
```

6.1.2. DTOs

6.1.2.1. LoginDto

```
namespace API.DTOs
{
   public class LoginDto
   {
       public string Email { get; set; }
```

```
        public string Password { get; set; }
    }
}
```

6.1.2.2. RegisterDto

```
using System.ComponentModel.DataAnnotations;

namespace API.DTOs
{
    public class RegisterDto
    {
        [Required]
        public string DisplayName { get; set; }

        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [RegularExpression("(?=.*\\d)(?=.*[a-z])(?=.*[A-Z]).{4,8}$",
ErrorMessage = "Password must be complex")]
        public string Password { get; set; }

        [Required]
        public string Username { get; set; }
    }
}
```

6.1.2.3. UserDto

```
namespace API.DTOs
{
    public class UserDto
    {
        public string DisplayName { get; set; }
        public string Token { get; set; }
        public string Username { get; set; }
        public string Image { get; set; }
    }
}
```

6.1.3. Extensions

6.1.3.1. ApplicationServicesExtensions.cs

```
using System;
using Application.Activities;
using Application.Core;
using Application.Interfaces;
using AutoMapper;
using Infrastructure.Photos;
using Infrastructure.Security;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.OpenApi.Models;
using Persistence;

namespace API.Extensions
{
   public static class ApplicationServicesExtensions
   {
       public static IServiceCollection AddApplicationServices(this
IServiceCollection services, IConfiguration config)
       {
           services.AddSwaggerGen(c =>
           {
               c.SwaggerDoc("v1", new OpenApiInfo { Title = "API", Version =
"v1" });
           });
           // handle DbContext service

           // services.AddDbContext<DataContext>(opt =>
           // {
           //
opt.UseNpgsql(config.GetConnectionString("DefaultConnection"));
           // });
           services.AddDbContext<DataContext>(options =>
           {
               var env =
Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT");

               string connStr;
```

Beograd, 2022.

```
                // Depending on if in development or production, use either
Heroku-provided
                // connection string, or development connection string from
env var.
                if (env == "Development")
                {
                    // Use connection string from file.
                    connStr =
config.GetConnectionString("DefaultConnection");
                }
                else
                {
                    // Use connection string provided at runtime by Heroku.
                    var connUrl =
Environment.GetEnvironmentVariable("DATABASE_URL");

                    // Parse connection URL to connection string for Npgsql
                    connUrl = connUrl.Replace("postgres://", string.Empty);
                    var pgUserPass = connUrl.Split("@")[0];
                    var pgHostPortDb = connUrl.Split("@")[1];
                    var pgHostPort = pgHostPortDb.Split("/")[0];
                    var pgDb = pgHostPortDb.Split("/")[1];
                    var pgUser = pgUserPass.Split(":")[0];
                    var pgPass = pgUserPass.Split(":")[1];
                    var pgHost = pgHostPort.Split(":")[0];
                    var pgPort = pgHostPort.Split(":")[1];

                    connStr = $"Server={pgHost};Port={pgPort};User
Id={pgUser};Password={pgPass};Database={pgDb}; SSL Mode=Require; Trust
Server Certificate=true";
                }

                // Whether the connection string came from the local
development configuration file
                // or from the environment variable from Heroku, use it to
set up your DbContext.
                options.UseNpgsql(connStr);
            });

            // handle Cors service
            services.AddCors(opt =>
            {
                opt.AddPolicy("CorsPolicy", policy =>
                {
                    policy
                        .AllowAnyMethod()
                        .AllowAnyHeader()
                        .AllowCredentials()
                        .WithOrigins("http://localhost:3000");
                });
            });
            // handle Mediator Service
            services.AddMediatR(typeof(List.Handler).Assembly);
            // handle AutoMapper Service
            services.AddAutoMapper(typeof(MappingProfiles).Assembly);
            //
            services.AddScoped<IUserAccessor, UserAccessor>();
            services.AddScoped<IPhotoAccessor, PhotoAccessor>();
            // handle CloudinarySettings

services.Configure<CloudinarySettings>(config.GetSection("Cloudinary"));
            // handle SignalR
            services.AddSignalR();

            return services;
        }
    }
}
```

6.1.3.2. HttpExtensions.cs

```
using System.Text.Json;
using Microsoft.AspNetCore.Http;

namespace API.Extensions
{
   public static class HttpExtensions
   {
       public static void AddPaginationHeader(this HttpResponse response,
int currentPage, int itemsPerPage, int totalItems, int totalPages)
       {
           var paginationHeader = new
           {
               currentPage,
               itemsPerPage,
               totalItems,
               totalPages,
           };
```

```
            response.Headers.Add("Pagination",
JsonSerializer.Serialize(paginationHeader));
            response.Headers.Add("Access-Control-Expose-Headers",
"Pagination");
        }
    }
}
```

6.1.3.3. IdentityServiceExtensions.cs

```
using System.Text;
using System.Threading.Tasks;
using API.Services;
using Domain;
using Infrastructure.Security;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.IdentityModel.Tokens;
using Persistence;

namespace API.Extensions
{
    public static class IdentityServiceExtensions
    {
        public static IServiceCollection AddIdentityServices(this
IServiceCollection service, IConfiguration config)
        {
            service.AddIdentityCore<AppUser>(opt =>
            {
                opt.Password.RequireNonAlphanumeric = false;
            })
            .AddEntityFrameworkStores<DataContext>()
            .AddSignInManager<SignInManager<AppUser>>();

            var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(config["TokenKey"]));

            service.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                .AddJwtBearer(opt =>
                {
                    opt.TokenValidationParameters = new
TokenValidationParameters
                    {
                        ValidateIssuerSigningKey = true,
                        IssuerSigningKey = key,
                        ValidateIssuer = false,
                        ValidateAudience = false,
                    };
                    opt.Events = new JwtBearerEvents
                    {
                        OnMessageReceived = context =>
                        {
                            var accessToken =
context.Request.Query["access_token"];
                            var path = context.HttpContext.Request.Path;
                            if(!string.IsNullOrEmpty(accessToken) &&
(path.StartsWithSegments("/chat")))
                            {
                                context.Token = accessToken;
                            }
                            return Task.CompletedTask;
                        }
                    };
                });

            service.AddAuthorization(opt =>
            {
                opt.AddPolicy("IsActivityHost", policy =>
                {
                    policy.Requirements.Add(new IsHostRequirement());
                });
            });
            service.AddTransient<IAuthorizationHandler,
IsHostRequirementHandler>();
            service.AddScoped<TokenService>();

            return service;
        }
    }
}
```

6.1.4. Middleware

6.1.4.1. ExceptionMiddleware.cs

```
using System;
using System.Net;
using System.Text.Json;
using System.Threading.Tasks;
using Application.Core;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace API.Middleware
{
    public class ExceptionMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger<ExceptionMiddleware> _logger;
        private readonly IHostEnvironment _env;
        public ExceptionMiddleware(RequestDelegate next,
ILogger<ExceptionMiddleware> logger, IHostEnvironment env)
        {
            _env = env;
            _logger = logger;
            _next = next;

        }

        public async Task InvokeAsync(HttpContext context)
        {
            try
            {
                await _next(context);
            }
            catch (Exception ex)
            {

                _logger.LogError(ex, ex.Message);
                context.Response.ContentType = "application/json";
                context.Response.StatusCode = (int)
HttpStatusCode.InternalServerError;

                var response = _env.IsDevelopment()
```

```
                    ? new AppException(context.Response.StatusCode,
ex.Message, ex.StackTrace?.ToString())
                    : new AppException(context.Response.StatusCode, "Server
Error");

                var options = new JsonSerializerOptions{PropertyNamingPolicy
= JsonNamingPolicy.CamelCase};

                var json = JsonSerializer.Serialize(response, options);

                await context.Response.WriteAsync(json);
            }
        }
    }
}
```

6.1.5. Services

6.1.5.1. TokenService.cs

```
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using Domain;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;

namespace API.Services
{
   public class TokenService
   {
       private readonly IConfiguration _config;
       public TokenService(IConfiguration config)
       {
           _config = config;

       }
       public string CreateToken(AppUser user)
       {
           var claims = new List<Claim>
```

Beograd, 2022.

```
            {
                new Claim(ClaimTypes.Name, user.UserName),
                new Claim(ClaimTypes.NameIdentifier, user.Id),
                new Claim(ClaimTypes.Email, user.Email),
            };

            var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["TokenKey"]));
            var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha512Signature);

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(claims),
                Expires = DateTime.Now.AddDays(300),
                SigningCredentials = creds
            };

            var tokenHandler = new JwtSecurityTokenHandler();

            var token = tokenHandler.CreateToken(tokenDescriptor);

            return tokenHandler.WriteToken(token);
        }
    }
}
```

6.1.6. SignalR

6.1.6.1. ChatHub.cs

```
using System;
using System.Threading.Tasks;
using Application.Comments;
using MediatR;
using Microsoft.AspNetCore.SignalR;

namespace API.SignalR
{
   public class ChatHub : Hub
   {
       private readonly IMediator __mediator;
```

```
        public ChatHub(IMediator _mediator)
        {
            __mediator = _mediator;

        }

        public async Task SendComment(Create.Command command)
        {
            var comment = await __mediator.Send(command);

            await Clients.Group(command.ActivityId.ToString())
                .SendAsync("ReceiveComment", comment.Value);

        }

        public override async Task OnConnectedAsync()
        {
            var httpContext = Context.GetHttpContext();
            var activityId = httpContext.Request.Query["activityId"];
            await Groups.AddToGroupAsync(Context.ConnectionId, activityId);
            var result = await __mediator.Send(new List.Query{ActivityId =
Guid.Parse(activityId)});
            await Clients.Caller.SendAsync("LoadComments", result.Value);
        }
    }
}
```

6.1.7. Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Domain;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Persistence;
```

```
namespace API
{
   public class Program
   {
       public static async Task Main(string[] args)
       {
           var host = CreateHostBuilder(args).Build();

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

           var services = scope.ServiceProvider;

           try
           {
               var context = services.GetRequiredService<DataContext>();
               var userManager =
services.GetRequiredService<UserManager<AppUser>>();
               await context.Database.MigrateAsync();
               await Seed.SeedData(context, userManager);
           }
           catch (Exception ex)
           {
               var logger = services.GetRequiredService<ILogger<Program>>();
               logger.LogError(ex, "An error occured during migration");

           }

           await host.RunAsync();
       }

       public static IHostBuilder CreateHostBuilder(string[] args) =>
           Host.CreateDefaultBuilder(args)
               .ConfigureWebHostDefaults(webBuilder =>
               {
                   webBuilder.UseStartup<Startup>();
               });
   }
}
```

6.1.8. Starup.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using API.Extensions;
using API.Middleware;
using API.SignalR;
using Application.Activities;
using Application.Core;
using AutoMapper;
using FluentValidation.AspNetCore;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Authorization;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.OpenApi.Models;
using Persistence;

// The Clean Archit. -> Controllers, Gateways, Presenters | Interface
Adapter
namespace API
{
   public class Startup
   {
       private readonly IConfiguration _config;
       public Startup(IConfiguration config)
       {
           _config = config;
       }

       // This method gets called by the runtime. Use this method to add
services to the container.
       public void ConfigureServices(IServiceCollection services)
       {

           services.AddControllers(opt =>
           {
               var policy = new
AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();
               opt.Filters.Add(new AuthorizeFilter(policy));
           })
           .AddFluentValidation(config =>
           {
               config.RegisterValidatorsFromAssemblyContaining<Create>();
           });

           services.AddApplicationServices(_config);
           services.AddIdentityServices(_config);

       }

       // This method gets called by the runtime. Use this method to
configure the HTTP request pipeline.
       public void Configure(IApplicationBuilder app, IWebHostEnvironment
env)
       {
           app.UseMiddleware<ExceptionMiddleware>();

           if (env.IsDevelopment())
           {
               app.UseSwagger();
               app.UseSwaggerUI(c =>
c.SwaggerEndpoint("/swagger/v1/swagger.json", "API v1"));
           }

           // app.UseHttpsRedirection();

           app.UseRouting();

           app.UseDefaultFiles();
           app.UseStaticFiles();

           app.UseCors("CorsPolicy");

           app.UseAuthentication();

           app.UseAuthorization();

           app.UseEndpoints(endpoints =>
           {
               endpoints.MapControllers();
               endpoints.MapHub<ChatHub>("/chat");
               endpoints.MapFallbackToController("Index", "Fallback");
           });
       }
   }
}
```

6.2. Application Sloj

6.2.1. Activities

6.2.1.1. ActivityDto

```
using System;
using System.Collections.Generic;
using Application.Profiles;

namespace Application.Activities
{
   public class ActivityDto
   {
       public Guid Id { get; set; }
       public string Title { get; set; }
       public DateTime Date { get; set; }
       public string Description { get; set; }
       public string Category { get; set; }
       public string City { get; set; }
       public string Venue { get; set; }
       public string HostUsername { get; set; }
       public bool isCancelled { get; set; }
       public ICollection<AttendeeDto> Attendees { get; set; }
   }
}
```

6.2.1.2. ActivitzParams

```
using System;
using Application.Core;
```

```
namespace Application.Activities
{
    public class ActivityParams : PagingParams
    {
        public bool IsGoing { get; set; }
        public bool IsHost { get; set; }
        public DateTime StartDate { get; set; } = DateTime.UtcNow;
    }
}
```

6.2.1.3. ActivityValidator

```
using Domain;
using FluentValidation;

namespace Application.Activities
{
    public class ActivityValidator : AbstractValidator<Activity>
    {
        public ActivityValidator()
        {
            RuleFor(x => x.Title).NotEmpty();
            RuleFor(x => x.Description).NotEmpty();
            RuleFor(x => x.Date).NotEmpty();
            RuleFor(x => x.Category).NotEmpty();
            RuleFor(x => x.City).NotEmpty();
            RuleFor(x => x.Venue).NotEmpty();
        }
    }
}
```

6.2.1.4. AttendeeDto

```
namespace Application.Activities
{
    public class AttendeeDto
    {
        public string Username { get; set; }
        public string DisplayName { get; set; }
        public string Bio { get; set; }
        public string Image { get; set; }
        public bool Following { get; set; }
        public int FollowersCount { get; set; }
        public int FollowingCount { get; set; }

    }
}
```

6.2.1.5. Create

```
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using Domain;
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Activities
{
    public class Create
    {
        public class Command : IRequest<Result<Unit>>
        {
            public Activity Activity { get; set; }
        }

        public class CommandValidator : AbstractValidator<Command>
        {
            public CommandValidator()
            {
                RuleFor(x => x.Activity).SetValidator(new
ActivityValidator());
            }
        }

        public class Handler : IRequestHandler<Command, Result<Unit>>
        {
            private readonly DataContext _context;
            private readonly IUserAccessor _userAccessor;
            public Handler(DataContext context, IUserAccessor
userAccessor)
            {
                _userAccessor = userAccessor;
                _context = context;
            }
            public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
            {
                var user = await _context.Users.FirstOrDefaultAsync(x
=> x.UserName == _userAccessor.GetUsername());

                var attendee = new ActivityAttendee
                {
                    AppUser = user,
                    Activity = request.Activity,
                    IsHost = true
                };
                request.Activity.Attendees.Add(attendee);

                _context.Activities.Add(request.Activity);

                var result = await _context.SaveChangesAsync() > 0;

                if (!result) return Result<Unit>.Failure("Failed to
create activity");

                return Result<Unit>.Success(Unit.Value);
            }
        }
    }
}
```

6.2.1.6. Delete

```
using System;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using MediatR;
using Persistence;

namespace Application.Activities
{
    public class Delete
    {
        public class Command : IRequest<Result<Unit>>
        {
            public Guid Id { get; set; }
        }

        public class Handler : IRequestHandler<Command, Result<Unit>>
        {
            private readonly DataContext _context;
            public Handler(DataContext context)
            {
                _context = context;

            }
            public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
            {
                var activity = await
_context.Activities.FindAsync(request.Id);

                //if(activity == null) return null;

                _context.Remove(activity);

                var result = await _context.SaveChangesAsync() > 0;

                if(!result) return Result<Unit>.Failure("Failed to
delete the activity");

                return Result<Unit>.Success(Unit.Value);
            }
        }
    }
}
```

6.2.1.7. Details

```
using System;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using Domain;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Activities
{
   public class Details
   {
       public class Query : IRequest<Result<ActivityDto>>
       {
           public Guid Id { get; set; }
       }

       public class Handler : IRequestHandler<Query,
Result<ActivityDto>>
       {
           private readonly DataContext _context;
           private readonly IMapper _mapper;
           private readonly IUserAccessor _userAccessor;
           public Handler(DataContext context, IMapper mapper,
IUserAccessor userAccessor)
           {
               _userAccessor = userAccessor;
               _mapper = mapper;
               _context = context;
           }
           public async Task<Result<ActivityDto>> Handle(Query
request, CancellationToken cancellationToken)
           {

               var activity = await _context.Activities

.ProjectTo<ActivityDto>(_mapper.ConfigurationProvider, new
{currentUsername = _userAccessor.GetUsername()})
                   .FirstOrDefaultAsync(x => x.Id == request.Id);

               return Result<ActivityDto>.Success(activity);
           }
       }
   }
}
```

6.2.1.8. Edit

```
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using AutoMapper;
using Domain;
using FluentValidation;
using MediatR;
using Persistence;

namespace Application.Activities
{
    public class Edit
    {
        public class Command : IRequest<Result<Unit>>
        {
            public Activity Activity { get; set; }
        }

        public class CommandValidator : AbstractValidator<Command>
        {
            public CommandValidator()
            {
                RuleFor(x => x.Activity).SetValidator(new
ActivityValidator());
            }
        }

        public class Handler : IRequestHandler<Command, Result<Unit>>
        {
            private readonly DataContext _context;
            private readonly IMapper _mapper;
            public Handler(DataContext context, IMapper mapper)
            {
                _mapper = mapper;
                _context = context;
            }
            public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
            {
                var activity = await
_context.Activities.FindAsync(request.Activity.Id);

                if(activity == null) return null;

                _mapper.Map(request.Activity, activity); // map
from(req.Activity{client} to _context{DbSet activity})
```

```
                var result = await _context.SaveChangesAsync() > 0;

                if(!result) return Result<Unit>.Failure("Failed to
update activity");

                return Result<Unit>.Success(Unit.Value);
            }
        }
    }
}
```

6.2.1.9. List

```
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using Domain;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Activities
// The clean Archit. -> Application(Bussines) logic, Use Cases(read,
update, create, delete)
{
    public class List
    {
        public class Query : IRequest<Result<PageList<ActivityDto>>>
        {
            public ActivityParams Params { get; set; }
        }

        public class Handler : IRequestHandler<Query,
Result<PageList<ActivityDto>>>
        {
            private readonly DataContext _context;
            private readonly IMapper _mapper;
            private readonly IUserAccessor _userAccessor;
            public Handler(DataContext context, IMapper mapper,
IUserAccessor userAccessor)
            {
                _userAccessor = userAccessor;
                _mapper = mapper;
                _context = context;

            }
            public async Task<Result<PageList<ActivityDto>>>
Handle(Query request, CancellationToken cancellationToken)
            {
                var query =  _context.Activities
                    .Where(d => d.Date >= request.Params.StartDate)
                    .OrderBy(d => d.Date)

.ProjectTo<ActivityDto>(_mapper.ConfigurationProvider, new
{currentUsername = _userAccessor.GetUsername()})
                    .AsQueryable();

                if(request.Params.IsGoing && !request.Params.IsHost)
              {
                    query =  query.Where(x => x.Attendees.Any(a =>
a.Username == _userAccessor.GetUsername()));
                }

                if(request.Params.IsHost && !request.Params.IsGoing)
                {
                    query = query.Where(x => x.HostUsername ==
_userAccessor.GetUsername());
                }

                return Result<PageList<ActivityDto>>.Success(
                    await PageList<ActivityDto>.CreateAsync(query,
request.Params.PageNumber, request.Params.PageSize)
                );
            }
        }
    }
}
```

6.2.1.10. UpdateAttendeence

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using Domain;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Activities
{
    public class UpdateAttendance
    {
        public class Command : IRequest<Result<Unit>>
        {
            public Guid Id { get; set; }
        }

        public class Handler : IRequestHandler<Command, Result<Unit>>
        {
            private readonly DataContext _context;
            private readonly IUserAccessor _userAccessor;
            public Handler(DataContext context, IUserAccessor
userAccessor)
            {
                _userAccessor = userAccessor;
                _context = context;
            }
            public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
            {
                var activity = await _context.Activities
                    .Include(a => a.Attendees)
                    .ThenInclude(u => u.AppUser)
                    .SingleOrDefaultAsync(x => x.Id == request.Id);

                if(activity == null) return null;

                var user = await _context.Users.FirstOrDefaultAsync(x
=>
                    x.UserName == _userAccessor.GetUsername());

                    if(user == null) return null;
```

Beograd, 2022.

```
                    var hostUsername =
activity.Attendees.FirstOrDefault(x => x.IsHost)?.AppUser?.UserName;

                    var attendace = activity.Attendees.FirstOrDefault(x
=> x.AppUser.UserName == user.UserName);

                    if(attendace != null && hostUsername ==
user.UserName)
                        activity.isCancelled = !activity.isCancelled;

                    if(attendace != null && hostUsername !=
user.UserName)
                        activity.Attendees.Remove(attendace);

                    if(attendace == null)
                    {
                        attendace = new ActivityAttendee
                        {
                            AppUser = user,
                            Activity = activity,
                            IsHost = false
                        };

                        activity.Attendees.Add(attendace);
                    }

                    var result = await _context.SaveChangesAsync() > 0;

                    return result ? Result<Unit>.Success(Unit.Value) :
Result<Unit>.Failure("Problem updating attendances");
            }
        }
    }
}
```

6.2.2. Comments

6.2.2.1. CommentDto

```
using System;
```

Beograd, 2022.

```
namespace Application.Comments
{
    public class CommentDto
    {
        public int Id { get; set; }
        public DateTime CreatedAt { get; set; }
        public string Body { get; set; }
        public string Username { get; set; }
        public string DisplayName { get; set; }
        public string Image { get; set; }
    }
}
```

6.2.2.2. Create

```
using System;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using AutoMapper;
using Domain;
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Comments
{
    public class Create
    {
        public class Command : IRequest<Result<CommentDto>>
        {
            public string Body { get; set; }
            public Guid ActivityId { get; set; }
        }

        public class CommandValidator : AbstractValidator<Command>
        {
            public CommandValidator()
            {
                RuleFor(x => x.Body).NotEmpty();
            }
        }

        public class Handler : IRequestHandler<Command,
Result<CommentDto>>
        {
            private readonly DataContext _context;
            private readonly IMapper _mapper;
            private readonly IUserAccessor _userAccessor;
            public Handler(DataContext context, IMapper mapper,
IUserAccessor userAccessor)
            {
                _userAccessor = userAccessor;
                _mapper = mapper;
                _context = context;

            }
            public async Task<Result<CommentDto>> Handle(Command
request, CancellationToken cancellationToken)
            {
                var activity = await
_context.Activities.FindAsync(request.ActivityId);

                if(activity == null) return null;

                var user = await _context.Users
                    .Include(p => p.Photos)
                    .SingleOrDefaultAsync(x => x.UserName ==
_userAccessor.GetUsername());

                var comment = new Comment
                {
                    Author = user,
                    Activity = activity,
                    Body = request.Body
                };

                activity.Comments.Add(comment);

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

                if(success) return
Result<CommentDto>.Success(_mapper.Map<CommentDto>(comment));
```

```
                return Result<CommentDto>.Failure("Failed to add
comment");
            }
        }
    }
}
```

6.2.2.3. List

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Comments
{
    public class List
    {
        public class Query : IRequest<Result<List<CommentDto>>>
        {
            public Guid ActivityId { get; set; }
        }

        public class Handler : IRequestHandler<Query,
Result<List<CommentDto>>>
        {
            private readonly DataContext _context;
            private readonly IMapper _mapper;
            public Handler(DataContext context, IMapper mapper)
            {
                _mapper = mapper;
                _context = context;

            }
            public async Task<Result<List<CommentDto>>> Handle(Query
request, CancellationToken cancellationToken)
            {
                    var comments = await _context.Comments
                         .Where(x => x.Activity.Id == request.ActivityId)
                         .OrderByDescending(x => x.CreatedAt)

.ProjectTo<CommentDto>(_mapper.ConfigurationProvider)
                         .ToListAsync();

                    return Result<List<CommentDto>>.Success(comments);
               }
          }
     }
}
```

6.2.3. Core

6.2.3.1. AppEception

```
namespace Application.Core
{
   public class AppException
   {
        public AppException(int statusCode, string message, string
details = null)
        {
            this.StatusCode = statusCode;
            this.Message = message;
            this.Details = details;

        }
        public int StatusCode { get; set; }
        public string Message { get; set; }
        public string Details { get; set; }
   }
}
```

6.2.3.2. MappingProfiles

```
using System.Linq;
using Application.Activities;
using Application.Comments;
using Application.Profiles;
using AutoMapper;
using Domain;

namespace Application.Core
{
   public class MappingProfiles : AutoMapper.Profile
   {
       public MappingProfiles()
       {
           string currentUsername = null;
           CreateMap<Activity, Activity>();
           CreateMap<Activity, ActivityDto>()
               .ForMember(d => d.HostUsername, o => o.MapFrom(s =>
s.Attendees.FirstOrDefault(x => x.IsHost).AppUser.UserName));
           CreateMap<ActivityAttendee, AttendeeDto>()
               .ForMember(d => d.DisplayName, o => o.MapFrom(s =>
s.AppUser.DisplayName))
               .ForMember(d => d.Username, o => o.MapFrom(s =>
s.AppUser.UserName))
               .ForMember(d => d.Bio, o => o.MapFrom(s =>
s.AppUser.Bio))
               .ForMember(d => d.Image, o => o.MapFrom(s =>
s.AppUser.Photos.FirstOrDefault(x => x.IsMain).Url))
               .ForMember(d => d.FollowersCount, o => o.MapFrom(s =>
s.AppUser.Followers.Count))
               .ForMember(d => d.FollowingCount, o => o.MapFrom(s =>
s.AppUser.Followings.Count))
               .ForMember(d => d.Following, o => o.MapFrom(s =>
s.AppUser.Followers.Any(x => x.Observer.UserName ==
currentUsername)));
           CreateMap<AppUser, Profiles.Profile>()
               .ForMember(d => d.Image, o => o.MapFrom(s =>
s.Photos.FirstOrDefault(x => x.IsMain).Url))
               .ForMember(d => d.FollowersCount, o => o.MapFrom(s =>
s.Followers.Count))
               .ForMember(d => d.FollowingCount, o => o.MapFrom(s =>
s.Followings.Count))
               .ForMember(d => d.Following, o => o.MapFrom(s =>
s.Followers.Any(x => x.Observer.UserName == currentUsername)));
           CreateMap<Comment, CommentDto>()
               .ForMember(d => d.DisplayName, o => o.MapFrom(s =>
s.Author.DisplayName))
               .ForMember(d => d.Username, o => o.MapFrom(s =>
s.Author.UserName))
```

```
                .ForMember(d => d.Image, o => o.MapFrom(s =>
s.Author.Photos.FirstOrDefault(x => x.IsMain).Url));
            CreateMap<ActivityAttendee, UserActivityDto>()
                .ForMember(d => d.Id, o => o.MapFrom(s =>
s.Activity.Id))
                .ForMember(d => d.Date, o => o.MapFrom(s =>
s.Activity.Date))
                .ForMember(d => d.Title, o => o.MapFrom(s =>
s.Activity.Title))
                .ForMember(d => d.Category, o => o.MapFrom(s =>
s.Activity.Category))
                .ForMember(d => d.HostUsername, o => o.MapFrom(s =>
s.Activity.Attendees.FirstOrDefault(x => x.IsHost).AppUser.UserName));

        }
    }
}
```

6.2.3.3. PageList

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace Application.Core
{
    public class PageList<T> : List<T>
    {
        public PageList(IEnumerable<T> items, int count, int
pageNumber, int pageSize)
        {
            CurrentPage = pageNumber;
            TotalPages = (int)Math.Ceiling(count / (double)pageSize);
            PageSize = pageSize;
            TotalCount = count;
            AddRange(items);
        }
        public int CurrentPage { get; set; }
        public int TotalPages { get; set; }
        public int PageSize { get; set; }
        public int TotalCount { get; set; }
        public static async Task<PageList<T>> CreateAsync(IQueryable<T>
source, int pageNumber, int pageSize)
        {
            var count = await source.CountAsync();
            var items = await source.Skip((pageNumber - 1) *
pageSize).Take(pageSize).ToListAsync();
            return new PageList<T>(items, count, pageNumber, pageSize);
        }
    }
}
```

6.2.3.4. PagingParams

```
namespace Application.Core
{
    public class PagingParams
    {
        private const int MaxPageSize = 50;
        public int PageNumber { get; set; } = 1;

        private int _pageSize = 10;
        public int PageSize
        {
            get => _pageSize;
            set => _pageSize = (value > MaxPageSize) ? MaxPageSize :
value;
        }

    }
}
```

6.2.3.5. Result

```
namespace Application.Core
{
    public class Result<T>
    {
        public bool IsSuccess { get; set; }
        public T Value { get; set; }
        public string Error { get; set; }
        public static Result<T> Success(T value) => new Result<T> {
IsSuccess = true, Value = value };
```

```
        public static Result<T> Failure(string error) => new Result<T>
{ IsSuccess = false, Error = error };
    }
}
```

6.2.4. Followers

6.2.4.1. FollowTogle

```
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using Domain;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Followers
{
   public class FollowToggle
   {
       public class Command : IRequest<Result<Unit>>
       {
           public string TargerUsername { get; set; }
       }

       public class Handler : IRequestHandler<Command, Result<Unit>>
       {
           private readonly DataContext _context;
           private readonly IUserAccessor _userAccessor;
           public Handler(DataContext context, IUserAccessor
userAccessor)
           {
               _userAccessor = userAccessor;
               _context = context;

           }
           public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
           {
               var observer = await
_context.Users.FirstOrDefaultAsync(x => x.UserName ==
_userAccessor.GetUsername());

               var target = await _context.Users.FirstOrDefaultAsync(x
=> x.UserName == request.TargerUsername);

               if(target == null) return null;

               var following = await
_context.UserFollowings.FindAsync(observer.Id, target.Id);

               if(following == null) {

                   following = new UserFollowing
                   {
                       Observer = observer,
                       Target = target,
                   };

                   _context.UserFollowings.Add(following);
               } else {
                   _context.UserFollowings.Remove(following);
               }

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

               if(success) return Result<Unit>.Success(Unit.Value);

               return Result<Unit>.Failure("Failed to update
following");
           }
       }
   }
}
```

6.2.4.2. List

```
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Followers
{
   public class List
   {
       public class Query : IRequest<Result<List<Profiles.Profile>>>
       {
           public string Predicate { get; set; }
           public string Username { get; set; }
       }

       public class Handler : IRequestHandler<Query,
Result<List<Profiles.Profile>>>
       {
           private readonly DataContext _context;
           private readonly IMapper _mapper;
           private readonly IUserAccessor _userAccessor;
           public Handler(DataContext context, IMapper mapper,
IUserAccessor userAccessor)
           {
               _userAccessor = userAccessor;
               _mapper = mapper;
               _context = context;

           }
           public async Task<Result<List<Profiles.Profile>>>
Handle(Query request, CancellationToken cancellationToken)
           {
               var profiles = new List<Profiles.Profile>();

               switch (request.Predicate)
               {
                   case "followers":
                       profiles = await
_context.UserFollowings.Where(x => x.Target.UserName ==
request.Username)
                           .Select(u => u.Observer)
.ProjectTo<Profiles.Profile>(_mapper.ConfigurationProvider, new
{currentUsername = _userAccessor.GetUsername()})
                           .ToListAsync();
                       break;
                   case "following":
                       profiles = await
_context.UserFollowings.Where(x => x.Observer.UserName ==
request.Username)
                           .Select(u => u.Target)

.ProjectTo<Profiles.Profile>(_mapper.ConfigurationProvider, new
{currentUsername = _userAccessor.GetUsername()})
                           .ToListAsync();
                       break;

               }

               return
Result<List<Profiles.Profile>>.Success(profiles);

           }
       }
   }
}
```

6.2.5. Interfaces

6.2.5.1. IPhotoAccessor

```
using System.Threading.Tasks;
using Application.Photos;
using Microsoft.AspNetCore.Http;

namespace Application.Interfaces
{
   public interface IPhotoAccessor
   {
       Task<PhotoUploadResult> AddPhoto(IFormFile file);
       Task<string> DeletePhoto(string publicId);
   }
}
```

6.2.5.2. IUserAccessor

```
namespace Application.Interfaces
{
   public interface IUserAccessor
   {
        string GetUsername();
   }
}
```

6.2.6. Photos

6.2.6.1. Add

```
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using Domain;
using MediatR;
using Microsoft.AspNetCore.Http;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Photos
{
   public class Add
   {
       public class Command : IRequest<Result<Photo>>
       {
           public IFormFile File { get; set; }
       }

       public class Handler : IRequestHandler<Command, Result<Photo>>
       {
           private readonly DataContext _context;
           private readonly IPhotoAccessor _photoAccessor;
           private readonly IUserAccessor _userAccessor;
           public Handler(DataContext context, IPhotoAccessor
photoAccessor, IUserAccessor userAccessor)
           {
               _userAccessor = userAccessor;
               _photoAccessor = photoAccessor;
```

```
                _context = context;

            }
            public async Task<Result<Photo>> Handle(Command request,
CancellationToken cancellationToken)
            {
                var user = await _context.Users.Include(p => p.Photos)
                    .FirstOrDefaultAsync(x => x.UserName ==
_userAccessor.GetUsername());

                if(user == null) return null;

                var photoUploadResult = await
_photoAccessor.AddPhoto(request.File);

                var photo = new Photo
                {
                    Url = photoUploadResult.Url,
                    Id = photoUploadResult.PublicId
                };

                if(!user.Photos.Any(x => x.IsMain)) photo.IsMain =
true;

                user.Photos.Add(photo);

                var result = await _context.SaveChangesAsync() > 0;

                if(result) return Result<Photo>.Success(photo);

                return Result<Photo>.Failure("Problem adding photo");
            }
        }
    }
}
```

6.2.6.2. Delete

```
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Photos
{
   public class Delete
   {
       public class Command : IRequest<Result<Unit>>
       {
           public string Id { get; set; }
       }

       public class Handler : IRequestHandler<Command, Result<Unit>>
       {
           private readonly DataContext _context;
           private readonly IPhotoAccessor _photoAccessor;
           private readonly IUserAccessor _userAccessor;
           public Handler(DataContext context, IPhotoAccessor
photoAccessor, IUserAccessor userAccessor)
           {
               _userAccessor = userAccessor;
               _photoAccessor = photoAccessor;
               _context = context;

           }
           public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
           {
               var user = await _context.Users.Include(p => p.Photos)
                   .FirstOrDefaultAsync(x => x.UserName ==
_userAccessor.GetUsername());

               if(user == null) return null;

               var photo = user.Photos.FirstOrDefault(x => x.Id ==
request.Id);

               if(photo == null) return null;

               if(photo.IsMain) return Result<Unit>.Failure("You
cannot delete your main photo");

               var result = await
_photoAccessor.DeletePhoto(photo.Id);
```

Beograd, 2022.

```
                if(result == null) return Result<Unit>.Failure("Problem
deleting photo from Cloudinary");

                user.Photos.Remove(photo);

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

                if(success) return Result<Unit>.Success(Unit.Value);

                return Result<Unit>.Failure("Problem deleting photo
from API");
            }
        }
    }
}
```

6.2.6.3. PhotoUpladResult

```
namespace Application.Photos
{
    public class PhotoUploadResult
    {
        public string PublicId { get; set; }
        public string Url { get; set; }
    }
}
```

6.2.6.4. SetMain

```
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Photos
{
    public class SetMain
    {
```

Beograd, 2022.

```
        public class Command : IRequest<Result<Unit>>
        {
            public string Id { get; set; }
        }

        public class Handler : IRequestHandler<Command, Result<Unit>>
        {
            private readonly DataContext _context;
            private readonly IUserAccessor _userAccessor;
            public Handler(DataContext context, IUserAccessor
userAccessor)
            {
                _userAccessor = userAccessor;
                _context = context;

            }
            public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
            {
                var user = await _context.Users.Include(p => p.Photos)
                    .FirstOrDefaultAsync(x => x.UserName ==
_userAccessor.GetUsername());

                if(user == null) return null;

                var photo = user.Photos.FirstOrDefault(x => x.Id ==
request.Id);

                if(photo == null) return null;

                var currentMain = user.Photos.FirstOrDefault(x =>
x.IsMain);

                if(currentMain != null) currentMain.IsMain = false;

                photo.IsMain = true;

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

                if(success) return Result<Unit>.Success(Unit.Value);

                return Result<Unit>.Failure("Problem setting main
photo");
             }
        }
    }
}
```

6.2.7. Profiles

6.2.7.1. Details

```
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Profiles
{
   public class Details
   {
       public class Query : IRequest<Result<Profile>>
       {
           public string Username { get; set; }

       }

       public class Handler : IRequestHandler<Query, Result<Profile>>
       {
           private readonly DataContext _context;
           private readonly IMapper _mapper;
           private readonly IUserAccessor _userAccessor;
           public Handler(DataContext context, IMapper mapper,
IUserAccessor userAccessor)
           {
               _userAccessor = userAccessor;
               _mapper = mapper;
               _context = context;

           }
           public async Task<Result<Profile>> Handle(Query request,
CancellationToken cancellationToken)
```

```
            {
                var user = await _context.Users
                    .ProjectTo<Profile>(_mapper.ConfigurationProvider,
new {currentUsername = _userAccessor.GetUsername()})
                    .SingleOrDefaultAsync(x => x.Username ==
request.Username);

                if (user == null) return null;

                return Result<Profile>.Success(user);
            }
        }
    }
}
```

6.2.7.2. Edit

```
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using AutoMapper;
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Profiles {
        public class Edit
        {
            public class Command : IRequest<Result<Unit>>
            {
                public string DisplayName { get; set; }
                public string Bio { get; set; }
            }
            public class CommandValidator : AbstractValidator<Command>
            {
               public CommandValidator()
               {
                   RuleFor(x => x.DisplayName).NotEmpty(); }
               }
            public class Handler : IRequestHandler<Command,
Result<Unit>>
            {
                private readonly DataContext _context;
                private readonly IUserAccessor _userAccessor;
                public Handler(DataContext context, IUserAccessor
userAccessor)
                {
                    _userAccessor = userAccessor;
                    _context = context;
                }
                public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
                {
                    var user = await
_context.Users.FirstOrDefaultAsync(x => x.UserName ==
_userAccessor.GetUsername());

                    user.Bio = request.Bio ?? user.Bio;

                    user.DisplayName = request.DisplayName ??
user.DisplayName;

                    _context.Entry(user).State = EntityState.Modified;

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

                    if (success) return
Result<Unit>.Success(Unit.Value);

                    return Result<Unit>.Failure("Problem updating
profile");
                }
            }
} }
```

6.2.7.3. ListActivities

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Profiles
{
   public class ListActivities
   {
       public class Query : IRequest<Result<List<UserActivityDto>>>
       {
           public string Username { get; set; }
           public string Predicate { get; set; }
       }
       public class Handler : IRequestHandler<Query,
Result<List<UserActivityDto>>>
       {
           private readonly DataContext _context;
           private readonly IMapper _mapper;
           public Handler(DataContext context, IMapper mapper)
           {
               _mapper = mapper;
               _context = context;
           }
           public async Task<Result<List<UserActivityDto>>>
Handle(Query request, CancellationToken cancellationToken)
           {
               var query = _context.ActivityAttendee
                   .Where(u => u.AppUser.UserName == request.Username)
                   .OrderBy(a => a.Activity.Date)

.ProjectTo<UserActivityDto>(_mapper.ConfigurationProvider)
                   .AsQueryable();

               query = request.Predicate switch
               {
                   "past" => query.Where(a => a.Date <= DateTime.Now),
                   "hosting" => query.Where(a => a.HostUsername ==
request.Username),
                   _ => query.Where(a => a.Date >= DateTime.Now)
               };

               var activities = await query.ToListAsync();
```

```
                return
Result<List<UserActivityDto>>.Success(activities);

            }
        }
    }
}
```

6.2.7.4. Profile

```
using System.Collections.Generic;
using Domain;

namespace Application.Profiles
{
   public class Profile
   {
       public string Username { get; set; }
       public string DisplayName { get; set; }
       public string Bio { get; set; }
       public string Image { get; set; }
       public bool Following { get; set; }
       public int FollowersCount { get; set; }
       public int FollowingCount { get; set; }
       public ICollection<Photo> Photos { get; set; }
   }
}
```

6.2.7.5. UserActivityDto

```
using System;
using System.Text.Json.Serialization;

namespace Application.Profiles
{
   public class UserActivityDto
   {
       public Guid Id { get; set; }
       public string Title { get; set; }
       public string Category { get; set; }
       public DateTime Date { get; set; }

       [JsonIgnore]
```

```
        public string HostUsername { get; set; }
    }
}
```

6.3. Domain

6.3.1. Activity

```
using System;
using System.Collections.Generic;

// The Clean Archit. -> Entitites leyer
namespace Domain
{
    public class Activity
    {
        public Guid Id { get; set; }
        public string Title { get; set; }
        public DateTime Date { get; set; }
        public string Description { get; set; }
        public string Category { get; set; }
        public string City { get; set; }
        public string Venue { get; set; }
        public bool isCancelled { get; set; }
        public ICollection<ActivityAttendee> Attendees { get; set; } = new
List<ActivityAttendee>();
        public ICollection<Comment> Comments { get; set; } = new
List<Comment>();
    }
}
```

6.3.2. ActivityAttendee

```
using System;

namespace Domain
{
    public class ActivityAttendee
    {
        public string AppUserId { get; set; }
        public AppUser AppUser { get; set; }
        public Guid ActivityId { get; set; }
        public Activity Activity { get; set; }
        public bool IsHost { get; set; }
    }
}
```

6.3.3. AppUser

```
using System.Collections.Generic;
using Microsoft.AspNetCore.Identity;

namespace Domain
{
    public class AppUser : IdentityUser
    {
        public string DisplayName { get; set; }
        public string Bio { get; set; }
        public ICollection<ActivityAttendee> Activities { get; set; }
        public ICollection<Photo> Photos { get; set; }
        public ICollection<UserFollowing> Followings { get; set; }
        public ICollection<UserFollowing> Followers { get; set; }
    }
}
```

6.3.4. Comment

```
using System;

namespace Domain
{
    public class Comment
    {
        public int Id { get; set; }
        public string Body { get; set; }
        public AppUser Author { get; set; }
        public Activity Activity { get; set; }
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    }
}
```

6.3.5. Photo

```
namespace Domain
{
    public class Photo
    {
        public string Id { get; set; }
        public string Url { get; set; }
        public bool IsMain { get; set; }
    }
}
```

6.3.6. UserFollowing

```
namespace Domain
{
    public class UserFollowing
    {
        public string ObserverId { get; set; }
        public AppUser Observer { get; set; }
        public string TargetId { get; set; }
        public AppUser Target { get; set; }
    }
}
```

6.4. Infrastructure

6.4.1. Photos

6.4.1.1. CloudanrySetting

```
namespace Infrastructure.Photos
{
    public class CloudinarySettings
    {
        public string CloudName { get; set; }
        public string ApiKey { get; set; }
        public string ApiSecret { get; set; }
    }
}
```

6.4.1.2. PhotoAccessor

```
using System;
using System.Threading.Tasks;
using Application.Interfaces;
using Application.Photos;
using CloudinaryDotNet;
using CloudinaryDotNet.Actions;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Options;

namespace Infrastructure.Photos
{
   public class PhotoAccessor : IPhotoAccessor
   {
       private readonly Cloudinary _cloudinary;
       public PhotoAccessor(IOptions<CloudinarySettings> config)
       {
           var account = new Account(
               config.Value.CloudName,
               config.Value.ApiKey,
               config.Value.ApiSecret
           );
           _cloudinary = new Cloudinary(account);
       }
       public async Task<PhotoUploadResult> AddPhoto(IFormFile file)
       {
           if(file.Length > 0)
           {
               await using var stream = file.OpenReadStream();
               var uploadParams = new ImageUploadParams
               {
                   File = new FileDescription(file.FileName, stream),
                   Transformation = new
Transformation().Height(500).Width(500).Crop("fill")
               };

               var uploadResult = await
_cloudinary.UploadAsync(uploadParams);

               if(uploadResult.Error != null)
               {
                   throw new Exception(uploadResult.Error.Message);
               }
```

```
                return new PhotoUploadResult
                {
                    PublicId = uploadResult.PublicId,
                    Url = uploadResult.SecureUrl.ToString()
                };
            }

            return null;
        }

        public async Task<string> DeletePhoto(string publicId)
        {
            var deleteParms = new DeletionParams(publicId);
            var result = await _cloudinary.DestroyAsync(deleteParms);

            return result.Result == "ok" ? result.Result : null;
        }
    }
}
```

6.4.2. Security

6.4.2.1. IsHostRequriment

```
using System;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Infrastructure.Security
{
    public class IsHostRequirement : IAuthorizationRequirement
    {

    }

    public class IsHostRequirementHandler :
AuthorizationHandler<IsHostRequirement>
    {
       private readonly DataContext _dbContext;
       private readonly IHttpContextAccessor _httpContextAccessor;
       public IsHostRequirementHandler(DataContext dbContext,
IHttpContextAccessor httpContextAccessor)
       {
           _httpContextAccessor = httpContextAccessor;
           _dbContext = dbContext;

       }
       protected override Task
HandleRequirementAsync(AuthorizationHandlerContext context,
IsHostRequirement requirement)
       {
           var userId =
context.User.FindFirstValue(ClaimTypes.NameIdentifier);

           if(userId == null) return Task.CompletedTask;

           var activityId =
Guid.Parse(_httpContextAccessor.HttpContext?.Request.RouteValues.Singl
eOrDefault(x => x.Key == "id").Value?.ToString());

           var attendee = _dbContext.ActivityAttendee
               .AsNoTracking()
               .SingleOrDefaultAsync(x => x.AppUserId == userId &&
x.ActivityId == activityId)
               .Result;

           if(attendee == null) return Task.CompletedTask;

           if(attendee.IsHost) context.Succeed(requirement);

           return Task.CompletedTask;
       }
   }
}
```

6.4.2.2. UserAccessor

```
using System.Security.Claims;
using Application.Interfaces;
using Microsoft.AspNetCore.Http;

namespace Infrastructure.Security
```

```
{
   public class UserAccessor : IUserAccessor
   {
       private readonly IHttpContextAccessor _httpContextAccessor;
       public UserAccessor(IHttpContextAccessor httpContextAccessor)
       {
           _httpContextAccessor = httpContextAccessor;

       }
       public string GetUsername()
       {
           return
_httpContextAccessor.HttpContext.User.FindFirstValue(ClaimTypes.Name);
       }
   }
}
```

6.5. Persistence

6.5.1. DataContext

```
using Domain;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace Persistence
{
   public class DataContext : IdentityDbContext<AppUser>
   {
       public DataContext(DbContextOptions options) : base(options)
       {
       }
       public DbSet<Activity> Activities { get; set; }
       public DbSet<ActivityAttendee> ActivityAttendee { get; set; }
       public DbSet<Photo> Photos { get; set; }
       public DbSet<Comment> Comments {get; set; }
       public DbSet<UserFollowing> UserFollowings { get; set; }

       protected override void OnModelCreating(ModelBuilder builder)
       {
           base.OnModelCreating(builder);
```

```
            builder.Entity<ActivityAttendee>(x => x.HasKey(aa => new
{aa.AppUserId, aa.ActivityId}));

            builder.Entity<ActivityAttendee>()
                .HasOne(u => u.AppUser)
                .WithMany(a => a.Activities)
                .HasForeignKey(aa => aa.AppUserId);

            builder.Entity<ActivityAttendee>()
                .HasOne(u => u.Activity)
                .WithMany(a => a.Attendees)
                .HasForeignKey(aa => aa.ActivityId);

            builder.Entity<Comment>()
                .HasOne(a => a.Activity)
                .WithMany(c => c.Comments)
                .OnDelete(DeleteBehavior.Cascade);

            builder.Entity<UserFollowing>(b =>
            {
                b.HasKey(k => new {k.ObserverId, k.TargetId});

                b.HasOne(o => o.Observer)
                    .WithMany(f => f.Followings)
                    .HasForeignKey(o => o.ObserverId)
                    .OnDelete(DeleteBehavior.Cascade);

                b.HasOne(o => o.Target)
                    .WithMany(f => f.Followers)
                    .HasForeignKey(o => o.TargetId)
                    .OnDelete(DeleteBehavior.Cascade);
            });
        }
    }
}
```

6.5.2. Migrations

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

namespace Persistence.Migrations
{
   public partial class PGInitial : Migration
```

```
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Activities",
                columns: table => new
                {
                    Id = table.Column<Guid>(type: "uuid", nullable: false),
                    Title = table.Column<string>(type: "text", nullable:
true),
                    Date = table.Column<DateTime>(type: "timestamp without
time zone", nullable: false),
                    Description = table.Column<string>(type: "text",
nullable: true),
                    Category = table.Column<string>(type: "text", nullable:
true),
                    City = table.Column<string>(type: "text", nullable:
true),
                    Venue = table.Column<string>(type: "text", nullable:
true),
                    isCancelled = table.Column<bool>(type: "boolean",
nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Activities", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "AspNetRoles",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Name = table.Column<string>(type: "character
varying(256)", maxLength: 256, nullable: true),
                    NormalizedName = table.Column<string>(type: "character
varying(256)", maxLength: 256, nullable: true),
                    ConcurrencyStamp = table.Column<string>(type: "text",
nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetRoles", x => x.Id);
                });
            migrationBuilder.CreateTable(
                name: "AspNetUsers",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    DisplayName = table.Column<string>(type: "text",
nullable: true),
                    Bio = table.Column<string>(type: "text", nullable: true),
                    UserName = table.Column<string>(type: "character
varying(256)", maxLength: 256, nullable: true),
                    NormalizedUserName = table.Column<string>(type:
"character varying(256)", maxLength: 256, nullable: true),
                    Email = table.Column<string>(type: "character
varying(256)", maxLength: 256, nullable: true),
                    NormalizedEmail = table.Column<string>(type: "character
varying(256)", maxLength: 256, nullable: true),
                    EmailConfirmed = table.Column<bool>(type: "boolean",
nullable: false),
                    PasswordHash = table.Column<string>(type: "text",
nullable: true),
                    SecurityStamp = table.Column<string>(type: "text",
nullable: true),
                    ConcurrencyStamp = table.Column<string>(type: "text",
nullable: true),
                    PhoneNumber = table.Column<string>(type: "text",
nullable: true),
                    PhoneNumberConfirmed = table.Column<bool>(type:
"boolean", nullable: false),
                    TwoFactorEnabled = table.Column<bool>(type: "boolean",
nullable: false),
                    LockoutEnd = table.Column<DateTimeOffset>(type:
"timestamp with time zone", nullable: true),
                    LockoutEnabled = table.Column<bool>(type: "boolean",
nullable: false),
                    AccessFailedCount = table.Column<int>(type: "integer",
nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUsers", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "AspNetRoleClaims",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    RoleId = table.Column<string>(type: "text", nullable:
false),
                    ClaimType = table.Column<string>(type: "text", nullable:
true),
                    ClaimValue = table.Column<string>(type: "text", nullable:
true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetRoleClaims", x => x.Id);
                    table.ForeignKey(
                        name: "FK_AspNetRoleClaims_AspNetRoles_RoleId",
                        column: x => x.RoleId,
                        principalTable: "AspNetRoles",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "ActivityAttendee",
                columns: table => new
                {
                    AppUserId = table.Column<string>(type: "text", nullable:
false),
                    ActivityId = table.Column<Guid>(type: "uuid", nullable:
false),
                    IsHost = table.Column<bool>(type: "boolean", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_ActivityAttendee", x => new {
x.AppUserId, x.ActivityId });
                    table.ForeignKey(
                        name: "FK_ActivityAttendee_Activities_ActivityId",
                        column: x => x.ActivityId,
                        principalTable: "Activities",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_ActivityAttendee_AspNetUsers_AppUserId",
                        column: x => x.AppUserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUserClaims",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    UserId = table.Column<string>(type: "text", nullable:
false),
                    ClaimType = table.Column<string>(type: "text", nullable:
true),
                    ClaimValue = table.Column<string>(type: "text", nullable:
true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUserClaims", x => x.Id);
                    table.ForeignKey(
                        name: "FK_AspNetUserClaims_AspNetUsers_UserId",
                        column: x => x.UserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUserLogins",
                columns: table => new
                {
                    LoginProvider = table.Column<string>(type: "text",
nullable: false),
                    ProviderKey = table.Column<string>(type: "text",
nullable: false),
                    ProviderDisplayName = table.Column<string>(type: "text",
nullable: true),
                    UserId = table.Column<string>(type: "text", nullable: 
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUserLogins", x => new { 
x.LoginProvider, x.ProviderKey });
                    table.ForeignKey(
                        name: "FK_AspNetUserLogins_AspNetUsers_UserId",
                        column: x => x.UserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUserRoles",
                columns: table => new
                {
                    UserId = table.Column<string>(type: "text", nullable: 
false),
                    RoleId = table.Column<string>(type: "text", nullable: 
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUserRoles", x => new { 
x.UserId, x.RoleId });
                    table.ForeignKey(
                        name: "FK_AspNetUserRoles_AspNetRoles_RoleId",
                        column: x => x.RoleId,
                        principalTable: "AspNetRoles",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_AspNetUserRoles_AspNetUsers_UserId",
                        column: x => x.UserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUserTokens",
                columns: table => new
                {
                    UserId = table.Column<string>(type: "text", nullable:
false),
                    LoginProvider = table.Column<string>(type: "text",
nullable: false),
                    Name = table.Column<string>(type: "text", nullable:
false),
                    Value = table.Column<string>(type: "text", nullable:
true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUserTokens", x => new {
x.UserId, x.LoginProvider, x.Name });
                    table.ForeignKey(
                        name: "FK_AspNetUserTokens_AspNetUsers_UserId",
                        column: x => x.UserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "Comments",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Body = table.Column<string>(type: "text", nullable:
true),
                    AuthorId = table.Column<string>(type: "text", nullable:
true),
                    ActivityId = table.Column<Guid>(type: "uuid", nullable:
true),
                    CreatedAt = table.Column<DateTime>(type: "timestamp
without time zone", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Comments", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Comments_Activities_ActivityId",
                        column: x => x.ActivityId,
                        principalTable: "Activities",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_Comments_AspNetUsers_AuthorId",
                        column: x => x.AuthorId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Restrict);
                });

            migrationBuilder.CreateTable(
                name: "Photos",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Url = table.Column<string>(type: "text", nullable: true),
                    IsMain = table.Column<bool>(type: "boolean", nullable:
false),
                    AppUserId = table.Column<string>(type: "text", nullable:
true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Photos", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Photos_AspNetUsers_AppUserId",
                        column: x => x.AppUserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Restrict);
                });

            migrationBuilder.CreateTable(
                name: "UserFollowings",
                columns: table => new
                {
                    ObserverId = table.Column<string>(type: "text", nullable:
false),
                    TargetId = table.Column<string>(type: "text", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_UserFollowings", x => new {
x.ObserverId, x.TargetId });
                    table.ForeignKey(
                        name: "FK_UserFollowings_AspNetUsers_ObserverId",
                        column: x => x.ObserverId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_UserFollowings_AspNetUsers_TargetId",
                        column: x => x.TargetId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateIndex(
                name: "IX_ActivityAttendee_ActivityId",
                table: "ActivityAttendee",
                column: "ActivityId");

            migrationBuilder.CreateIndex(
                name: "IX_AspNetRoleClaims_RoleId",
                table: "AspNetRoleClaims",
                column: "RoleId");

            migrationBuilder.CreateIndex(
                name: "RoleNameIndex",
                table: "AspNetRoles",
                column: "NormalizedName",
                unique: true);

            migrationBuilder.CreateIndex(
                name: "IX_AspNetUserClaims_UserId",
                table: "AspNetUserClaims",
                column: "UserId");

            migrationBuilder.CreateIndex(
                name: "IX_AspNetUserLogins_UserId",
                table: "AspNetUserLogins",
                column: "UserId");

            migrationBuilder.CreateIndex(
                name: "IX_AspNetUserRoles_RoleId",
                table: "AspNetUserRoles",
                column: "RoleId");

            migrationBuilder.CreateIndex(
                name: "EmailIndex",
                table: "AspNetUsers",
                column: "NormalizedEmail");

            migrationBuilder.CreateIndex(
                name: "UserNameIndex",
                table: "AspNetUsers",
                column: "NormalizedUserName",
                unique: true);

            migrationBuilder.CreateIndex(
                name: "IX_Comments_ActivityId",
                table: "Comments",
                column: "ActivityId");

            migrationBuilder.CreateIndex(
                name: "IX_Comments_AuthorId",
                table: "Comments",
                column: "AuthorId");

            migrationBuilder.CreateIndex(
                name: "IX_Photos_AppUserId",
                table: "Photos",
                column: "AppUserId");

            migrationBuilder.CreateIndex(
                name: "IX_UserFollowings_TargetId",
                table: "UserFollowings",
                column: "TargetId");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "ActivityAttendee");

            migrationBuilder.DropTable(
                name: "AspNetRoleClaims");
            migrationBuilder.DropTable(
                name: "AspNetUserClaims");

            migrationBuilder.DropTable(
                name: "AspNetUserLogins");

            migrationBuilder.DropTable(
                name: "AspNetUserRoles");

            migrationBuilder.DropTable(
                name: "AspNetUserTokens");

            migrationBuilder.DropTable(
                name: "Comments");

            migrationBuilder.DropTable(
                name: "Photos");

            migrationBuilder.DropTable(
                name: "UserFollowings");

            migrationBuilder.DropTable(
                name: "AspNetRoles");

            migrationBuilder.DropTable(
                name: "Activities");

            migrationBuilder.DropTable(
                name: "AspNetUsers");
        }
    }
}
```

6.5.3. Seed.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Domain;
using Microsoft.AspNetCore.Identity;

namespace Persistence
{
```

```
public class Seed
{
    public static async Task SeedData(DataContext context,
        UserManager<AppUser> userManager)
    {
        if (!userManager.Users.Any() && !context.Activities.Any())
        {
            var users = new List<AppUser>
            {
                new AppUser
                {
                    DisplayName = "Bob",
                    UserName = "bob",
                    Email = "bob@test.com"
                },
                new AppUser
                {
                    DisplayName = "Jane",
                    UserName = "jane",
                    Email = "jane@test.com"
                },
                new AppUser
                {
                    DisplayName = "Tom",
                    UserName = "tom",
                    Email = "tom@test.com"
                },
            };

            foreach (var user in users)
            {
                await userManager.CreateAsync(user, "Pa$$w0rd");
            }

            var activities = new List<Activity>
            {
                new Activity
                {
                    Title = "Past Activity 1",
                    Date = DateTime.Now.AddMonths(-2),
                    Description = "Activity 2 months ago",
                    Category = "drinks",
                    City = "London",
                    Venue = "Pub",
                    Attendees = new List<ActivityAttendee>
                    {
                        new ActivityAttendee
                        {
                            AppUser = users[0],
                            IsHost = true
                        }
                    }
                },
                new Activity
                {
                    Title = "Past Activity 2",
                    Date = DateTime.Now.AddMonths(-1),
                    Description = "Activity 1 month ago",
                    Category = "culture",
                    City = "Paris",
                    Venue = "The Louvre",
                    Attendees = new List<ActivityAttendee>
                    {
                        new ActivityAttendee
                        {
                            AppUser = users[0],
                            IsHost = true
                        },
                        new ActivityAttendee
                        {
                            AppUser = users[1],
                            IsHost = false
                        },
                    }
                },
                new Activity
                {
                    Title = "Future Activity 1",
                    Date = DateTime.Now.AddMonths(1),
                    Description = "Activity 1 month in future",
                    Category = "music",
                    City = "London",
                    Venue = "Wembly Stadium",
                    Attendees = new List<ActivityAttendee>
                    {
                        new ActivityAttendee
                        {
                            AppUser = users[2],
                IsHost = true
            },
            new ActivityAttendee
            {
                AppUser = users[1],
                IsHost = false
            },
        }
},
new Activity
{
    Title = "Future Activity 2",
    Date = DateTime.Now.AddMonths(2),
    Description = "Activity 2 months in future",
    Category = "food",
    City = "London",
    Venue = "Jamies Italian",
    Attendees = new List<ActivityAttendee>
    {
        new ActivityAttendee
        {
            AppUser = users[0],
            IsHost = true
        },
        new ActivityAttendee
        {
            AppUser = users[2],
            IsHost = false
        },
    }
},
new Activity
{
    Title = "Future Activity 3",
    Date = DateTime.Now.AddMonths(3),
    Description = "Activity 3 months in future",
    Category = "drinks",
    City = "London",
    Venue = "Pub",
    Attendees = new List<ActivityAttendee>
    {
        new ActivityAttendee
        {
            AppUser = users[1],
            IsHost = true
        },
        new ActivityAttendee
        {
            AppUser = users[0],
            IsHost = false
        },
    }
},
new Activity
{
    Title = "Future Activity 4",
    Date = DateTime.Now.AddMonths(4),
    Description = "Activity 4 months in future",
    Category = "culture",
    City = "London",
    Venue = "British Museum",
    Attendees = new List<ActivityAttendee>
    {
        new ActivityAttendee
        {
            AppUser = users[1],
            IsHost = true
        }
    }
},
new Activity
{
    Title = "Future Activity 5",
    Date = DateTime.Now.AddMonths(5),
    Description = "Activity 5 months in future",
    Category = "drinks",
    City = "London",
    Venue = "Punch and Judy",
    Attendees = new List<ActivityAttendee>
    {
        new ActivityAttendee
        {
            AppUser = users[0],
            IsHost = true
        },
        new ActivityAttendee
        {
            AppUser = users[1],
            IsHost = false
        },
    }
},
new Activity
{
    Title = "Future Activity 6",
    Date = DateTime.Now.AddMonths(6),
    Description = "Activity 6 months in future",
    Category = "music",
    City = "London",
    Venue = "O2 Arena",
    Attendees = new List<ActivityAttendee>
    {
        new ActivityAttendee
        {
            AppUser = users[2],
            IsHost = true
        },
        new ActivityAttendee
        {
            AppUser = users[1],
            IsHost = false
        },
    }
},
new Activity
{
    Title = "Future Activity 7",
    Date = DateTime.Now.AddMonths(7),
    Description = "Activity 7 months in future",
    Category = "travel",
    City = "Berlin",
    Venue = "All",
    Attendees = new List<ActivityAttendee>
    {
        new ActivityAttendee
        {
            AppUser = users[0],
            IsHost = true
        },
        new ActivityAttendee
        {
            AppUser = users[2],
                                IsHost = false
                            },
                        }
                    },
                    new Activity
                    {
                        Title = "Future Activity 8",
                        Date = DateTime.Now.AddMonths(8),
                        Description = "Activity 8 months in future",
                        Category = "drinks",
                        City = "London",
                        Venue = "Pub",
                        Attendees = new List<ActivityAttendee>
                        {
                            new ActivityAttendee
                            {
                                AppUser = users[2],
                                IsHost = true
                            },
                            new ActivityAttendee
                            {
                                AppUser = users[1],
                                IsHost = false
                            },
                        }
                    }
                };

                await context.Activities.AddRangeAsync(activities);
                await context.SaveChangesAsync();
            }
        }
    }
}
```

7. Front-End Kodovi

Client-app project

7.1. Public

assets

7.2. src

7.2.1. app

7.2.1.1. api

7.2.1.1.1. agent.ts

```
import axios, { AxiosError, AxiosResponse } from 'axios';
import { toast } from 'react-toastify';
import {Activity, ActivityFormValues} from
'../models/activity';
import { PaginatedRusult } from '../models/pagination';
import { Photo, Profile, UserActivity } from
'../models/profile';
import { User, UserFormValues } from '../models/user';
import { store } from '../stores/store';

const sleep = (delay: number) => {
   return new Promise((resolve) => {
       setTimeout(resolve, delay);
   });
}

axios.defaults.baseURL = process.env.REACT_APP_API_URL;

axios.interceptors.request.use((config: any) => {
   const token = store.commonStore.token;
   if (token) config.headers.Authorization = `Bearer ${token}`
   return config;
})

axios.interceptors.response.use(async response => {
   if(process.env.NODE_ENV === 'development')  await
sleep(1000);

       const pagination = response.headers['pagination'];
       if(pagination) {
           response.data = new PaginatedRusult(response.data,
JSON.parse(pagination));
           return response as
AxiosResponse<PaginatedRusult<any>>
       }
       return response;
}, (error: AxiosError) => {
   const { data, status, config} = error.response!;
   //console.log(error.response);
   switch(status) {
       case 400:
           if(typeof data === 'string'){
               toast.error(data);
           }
           if(config.method === 'get' &&
data.errors.hasOwnProperty('id')) {
               //history.push('/not-found');
               toast.error('not found');
           }
           if(data.errors) {
               const modalStateErrors = [];
               for (const key in data.errors) {
                   if(data.errors[key]) {
                       modalStateErrors.push(data.errors[key])
                   }
               }
               throw modalStateErrors.flat();
           }
           break;
       case 401:
           toast.error('unauthorised');
           break;
       case 404:
           //history.push('/not-found');
           toast.error('not found');
           break;
       case 500:
           store.commonStore.setServerError(data);
           //history.push('/server-error');
           toast.error('server error');
           break;
   }
   return Promise.reject(error);
})

const responseBody = <T> (response: AxiosResponse<T>) =>
response.data;

const requests = {
   get: <T>  (url: string) =>
axios.get<T>(url).then(responseBody),
   post: <T> (url: string, body: {}) => axios.post<T>(url,
body).then(responseBody),
   put: <T> (url: string, body: {}) => axios.put<T>(url,
body).then(responseBody),
   del: <T> (url: string) =>
axios.delete<T>(url).then(responseBody),
};

const Activities = {
   list: (params: URLSearchParams) =>
axios.get<PaginatedRusult<Activity[]>>('/activities',
{params}).then(responseBody),
   details: (id: string) =>
requests.get<Activity>(`/activities/${id}`),
   create: (activity: ActivityFormValues) =>
requests.post<Activity>('/activities', activity),
   update: (activity: ActivityFormValues) =>
requests.put<void>(`/activities/${activity.id}`, activity),
   delete: (id: string) =>
requests.del<void>(`/activities/${id}`),
   attend: (id: string) =>
requests.post<void>(`/activities/${id}/attend`, {})
};

const Account = {
   current: () => requests.get<User>('/account'),
   login: (user: UserFormValues) =>
requests.post<User>('/account/login', user),
   register: (user: UserFormValues) =>
requests.post<User>('/account/register', user),
}

const Profiles = {
   get: (username: string) =>
requests.get<Profile>(`/profiles/${username}`),
   uploadPhoto: (file: Blob) => {
       let formData = new FormData();
       formData.append('File', file);
       return axios.post<Photo>('photos', formData, {
           headers: {'Content-type': 'multipart/form-data'}
       })
   },
   setMinPhoto: (id: string) =>
requests.post(`/photos/${id}/setMain`, {}),
   deletePhoto: (id: string) => requests.del(`/photos/${id}`),
   updateProfile: (profile: Partial<Profile>) =>
requests.put(`/profiles`, profile),
   updateFollowing: (username: string) =>
requests.post(`/follow/${username}`, {}),
   listFollowings: (username: string, predicate: string) =>
requests.get<Profile[]>(`/follow/${username}?predicate=${predic
ate}`),
   listActivities: (username: string, predicate: string) =>
requests.get<UserActivity[]>(`/profiles/${username}/activities?
predicate=${predicate}`)
}

const agent = {
   Activities,
   Account,
   Profiles,
}

export default agent;
```

7.2.1.2. common

7.2.1.2.1. form

7.2.1.2.1.1. MyDateInput.tsx

```
import { useField } from 'formik'
import React from 'react'
import { Form, Label } from 'semantic-ui-react'
import DatePicker, {ReactDatePickerProps} from 'react-
datepicker';

export default function MyDateInput(props:
Partial<ReactDatePickerProps>) {
   const [field, meta, helpers] = useField(props.name!)
   return (
       <Form.Field error={meta.touched && !!meta.error}>
           <DatePicker
               {...field}
               {...props}
```

```
                selected={(field.value && new
Date(field.value)) || null}
                onChange={value =>
helpers.setValue(value)}
            />

            {meta.touched && meta.error ? (
                <Label basic
color='red'>{meta.error}</Label>
            ): null }
        </Form.Field>
    )
}
```

7.2.1.2.1.2. MySelectInput

```
import { useField } from 'formik'
import React from 'react'
import { Form, Label, Select } from 'semantic-ui-react'

interface Props {
    placeholder: string;
    name: string
    options: any;
    label?: string
}
export default function MySelectInput(props: Props) {
    const [field, meta, helpers] = useField(props.name)
    return (
        <Form.Field error={meta.touched && !!meta.error}>
            <label>{props.label}</label>
            <Select
                clearable
                options={props.options}
                value={field.value || null}
                onChange={(event, data) =>
helpers.setValue(data.value)}
                onBlur={() => helpers.setTouched(true)}
                placeholder={props.placeholder}
            />
            {meta.touched && meta.error ? (
                <Label basic
color='red'>{meta.error}</Label>
```

```
            ): null }
        </Form.Field>
    )
}
```

7.2.1.2.1.3. MyTextArea

```
import { useField } from 'formik'
import React from 'react'
import { Form, Label } from 'semantic-ui-react'

interface Props {
   placeholder: string;
   name: string;
   rows: number;
   label?: string;
}
export default function MyTextArea(props: Props) {
   const [field, meta] = useField(props.name)
   return (
       <Form.Field error={meta.touched && !!meta.error}>
           <label>{props.label}</label>
           <textarea {...field} {...props} />
           {meta.touched && meta.error ? (
               <Label basic
color='red'>{meta.error}</Label>
           ): null }
       </Form.Field>
   )
}
```

7.2.1.2.1.4. MyTextInput

```
import { useField } from 'formik'
import React from 'react'
import { Form, Label } from 'semantic-ui-react'

interface Props {
   placeholder: string;
   name: string;
   type?: string;
   label?: string;
}
export default function MyTextInput(props: Props) {
   const [field, meta] = useField(props.name)
   return (
       <Form.Field error={meta.touched && !!meta.error}>
           <label>{props.label}</label>
           <input {...field} {...props} />
           {meta.touched && meta.error ? (
               <Label basic
color='red'>{meta.error}</Label>
           ): null }
       </Form.Field>
   )
}
```

7.2.1.2.2. imageUpload

7.2.1.2.2.1. PhotoUploadWidget.tsx

```
import React, { useState, useEffect } from 'react'
import { Button, Grid, Header } from 'semantic-ui-react'
import PhotoWidgetCropper from './PhotoWidgetCropper';
import PhotoWidgetDropzone from './PhotoWidgetDropzone'

interface Props {
   loading: boolean;
   uploadPhoto: (file: Blob) => void;
}

export default function PhotoUploadWidget({loading,
uploadPhoto}: Props) {
   const [files, setFiles] = useState<any>([]);
   const [cropper, setCropper] = useState<Cropper>();

   const onCrop = () => {
       if(cropper) {
           cropper.getCroppedCanvas().toBlob(blob =>
uploadPhoto(blob!));
       }
   }
   useEffect(() => {
       return () => {
           files.forEach((file: any) =>
URL.revokeObjectURL(file.preview));
       }
   }, [files])

   return (
       <Grid>
           <Grid.Column width={4}>
               <Header sub color='teal' content='Step 1
- Add Photo' />
               <PhotoWidgetDropzone setFiles={setFiles}
/>
           </Grid.Column>
           <Grid.Column width={1} />
           <Grid.Column width={4}>
               <Header color='teal' content='Step 2 -
Resize Image' />
               {files && files.length > 0 && (
                   <PhotoWidgetCropper
setCropper={setCropper} imagePreview={files[0].preview}
/>
               )}
           </Grid.Column>
           <Grid.Column width={1} />
           <Grid.Column width={4}>
               <Header color='teal' content='Step 3 -
Preview & Upload' />
               {files && files.length > 0 &&
                   <>
                       <div className='img-preview'
style={{minHeight: 200, overflow: 'hidden'}} />
                       <Button.Group>
                           <Button loading={loading}
onClick={onCrop} positive icon='check' />
                           <Button disabled={loading}
onClick={() => setFiles([])} icon='close' />
                       </Button.Group>
                   </>
               }
           </Grid.Column>
       </Grid>
   )
```

```
}
```

7.2.1.2.2.2. PhotoWidgetCropper

```
import React from 'react'
import { Cropper } from 'react-cropper';
import 'cropperjs/dist/cropper.css';

interface Props {
   imagePreview: string;
   setCropper: (cropper: Cropper) => void;
}

export default function
PhotoWidgetCropper({imagePreview, setCropper}: Props) {
   return (
       <Cropper
           src={imagePreview}
           style={{height: 200, width: '100%'}}
           initialAspectRatio={1}
           aspectRatio={1}
           preview='.img-preview'
           guides={false}
           viewMode={1}
           autoCropArea={1}
           background={false}
           onInitialized={cropper =>
setCropper(cropper)}
       />
   )
}
```

7.2.1.2.2.3. PhotoWidgetDropzone

```
import React, {useCallback} from 'react'
import {useDropzone} from 'react-dropzone'
import { Header, Icon } from 'semantic-ui-react';

interface Props {
 setFiles: (files: any) => void;
}
```

```
export default function PhotoWidgetDropzone({setFiles}:
Props) {
const dzStyles = {
 border: 'dashed 3px #eee',
 borderColor: '#eee',
 borderRadius: '5px',
 paddingTop: '30px',
 textAlign: 'center' as 'center',
 height: 200,
}

const dzActive = {
 borderColor: 'green',
}

 const onDrop = useCallback(acceptedFiles => {
   setFiles(acceptedFiles.map((file: any) =>
Object.assign(file, {
     preview: URL.createObjectURL(file)
   })))
 }, [setFiles]);

 const {getRootProps, getInputProps, isDragActive} =
useDropzone({onDrop})

 return (
   <div {...getRootProps()} style={isDragActive ?
{...dzStyles, ...dzActive} : dzStyles}>
     <input {...getInputProps()} />
     <Icon
       name='upload'
       size='huge'
     />
     <Header content='Drop image here' />
   </div>
 )
}
```

7.2.1.2.3. modals

7.2.1.2.3.1. ModalContainer

```
import { observer } from 'mobx-react-lite';
import React from 'react'
import { Modal } from 'semantic-ui-react';
import { useStore } from '../../stores/store'

export default observer(function ModalContainer() {
   const {modalStore} = useStore();
   return (
       <Modal open={modalStore.modal.open}
onClose={modalStore.closeModal} size='mini'>
           <Modal.Content>
               {modalStore.modal.body}
           </Modal.Content>
       </Modal>
   )
})
```

7.2.1.2.4. options

7.2.1.2.4.1. categorzOptions.ts

```
export const categoryOptions = [
   {text: 'Drinks', value: 'drinks'},
   {text: 'Culture', value: 'culture'},
   {text: 'Film', value: 'film'},
   {text: 'Food', value: 'food'},
   {text: 'Music', value: 'music'},
   {text: 'Travel', value: 'travel'}
];
```

7.2.1.3. layout

7.2.1.3.1. App.tsx

```
import React, {useEffect} from 'react';
import { Container} from 'semantic-ui-react';
import NavBar from './NavBar';
import ActivityDashboard from
'../../features/activities/dashboard/ActivityDashboard';
import { observer } from 'mobx-react-lite';
import { Route, Switch, useLocation } from 'react-router';
import HomePage from '../../features/home/HomePage';
import ActivityForm from
'../../features/activities/form/ActivityForm';
import ActivityDetails from
'../../features/activities/details/ActivityDetails';
import TestErrors from '../../features/errors/TestError';
import { ToastContainer } from 'react-toastify';
import NotFound from '../../features/errors/NotFound';
import ServerError from '../../features/errors/ServerError';
import { useStore } from '../stores/store';
import LoadingComponent from './LoadingComponent';
import ModalContainer from '../common/modals/ModalContainer';
import ProfilePage from '../../features/profiles/ProfilePage';
import PrivateRoute from './PrivateRoute';

function App(): JSX.Element {
 const location = useLocation();
 const { commonStore, userStore } = useStore();

 useEffect(() => {
  if(commonStore.token){
    userStore.getUser().finally(() =>
commonStore.setAppLoaded());
  } else {
    commonStore.setAppLoaded();
  }
 }, [commonStore, userStore])

 if (!commonStore.appLoaded) return <LoadingComponent
content='Loading app' />

 return (
   <>
     <ToastContainer position='bottom-right' hideProgressBar />
     <ModalContainer />
     <Route path='/' exact component={HomePage} />
     <Route
       path={'/(.+)'}
       render={() => (
         <>
           <NavBar />
           <Container style={{marginTop: '7em'}}>
             <Switch>
               <PrivateRoute path='/activities' exact
component={ActivityDashboard} />
               <PrivateRoute path='/activities/:id'
component={ActivityDetails} />
               <PrivateRoute key={location.key}
path={['/createActivity', '/manage/:id']}
component={ActivityForm} />
               <PrivateRoute path='/profiles/:username'
component={ProfilePage} />
               <PrivateRoute path='/errors'
component={TestErrors} />
               <Route component={NotFound} />
               <Route component={ServerError} />
             </Switch>
           </Container>
         </>
       )}
     />
   </>
 );
}

export default observer(App);
```

7.2.1.3.2. LoadingComponent

```
import React from 'react';
import { Dimmer, Loader } from 'semantic-ui-react';

interface Props {
   inverted?: boolean;
   content?: string;
}

export default function LoadingComponent({inverted = true,
content = 'Loading...'}: Props) {
   return (
       <Dimmer active={true} inverted={inverted}>
           <Loader content={content} />
       </Dimmer>
   )
}
```

7.2.1.3.3. NavBar

```
import { observer } from 'mobx-react-lite';
import React from 'react'
import { Link, NavLink } from 'react-router-dom';
import { Button, Container, Dropdown, Image, Menu } from
'semantic-ui-react'
import { useStore } from '../stores/store';

export default observer(function NavBar() {
   const {userStore: {user, logout}} = useStore();
   return (
       <Menu inverted fixed='top' >
           <Container>
               <Menu.Item as={NavLink} to='/' exact header>
                   <img src="/assets/logo.png" alt="codedancing
logo" style={{marginRight: 10}}/>
                   CodeDancing Activities
               </Menu.Item>
               <Menu.Item as={NavLink} to='/activities'
name='CodeDancing Activities' />
               {/* <Menu.Item as={NavLink} to='/errors'
name='Errors' /> */}
               <Menu.Item>
                   <Button as={NavLink} to='/createActivity'
positive content='Create Activity' />
               </Menu.Item>
               <Menu.Item position='right'>
                   <Image src={user?.image ||
'/assets/user.png'} avatar spaced='right'/>
                   <Dropdown position='top left'
text={user?.displayName}>
                       <Dropdown.Menu>
                           <Dropdown.Item as={Link}
to={`/profiles/${user?.username}`} text='My profile'
icon='user'/>
                           <Dropdown.Item onClick={logout}
text='Logout' icon='power' />
                       </Dropdown.Menu>
```

```
                  </Dropdown>
              </Menu.Item>
          </Container>
      </Menu>
   )
})
```

7.2.1.3.4. PrivateRoute

```
import React from 'react'
import { Redirect, Route, RouteComponentProps, RouteProps }
from 'react-router'
import { useStore } from '../stores/store'

interface Props extends RouteProps {
   component: React.ComponentType<RouteComponentProps<any>> |
React.ComponentType<any>;
}

export default function PrivateRoute({component: Component,
...rest}: Props) {
   const {userStore: {isLoggedIn}} = useStore();
   return (
       <Route
           {...rest}
           render={(props) => isLoggedIn ? <Component
{...props} /> : <Redirect to='/' /> }
       />
   )
}
```

7.2.1.3.5. ScrollToTop

```
import React, {useEffect} from 'react';
import { useLocation } from 'react-router-dom';

export default function ScrollToTop() {
   const { pathname } = useLocation();

   useEffect(() => {
       window.scrollTo(0, 0);
   }, [pathname])

   return null;
}
```

7.2.1.3.6. style.css

```
body {
 background-color: #EAEAEA !important;
}

.ui.inverted.top.fixed.menu {
 background-image: linear-gradient(135deg, rgb(24,42,115) 0%,
rgb(33, 138, 174) 69%, rgb(32, 167, 172) 89%) !important;
}

.react-calendar {
 width: 100%;
 border: none;
 box-shadow: 0 1px 2px 0 rgba(34, 36, 38, .15);
}

.react-datepicker-wrapper {
 width: 100%;
}

/*home page styles*/

.masthead {
 display: flex;
 align-items: center;
 background-image: linear-gradient(
         135deg,
         rgb(24, 42, 115) 0%,
         rgb(33, 138, 174) 69%,
         rgb(32, 167, 172) 89%
 ) !important;
 height: 100vh;
}

.masthead .ui.menu .ui.button,
.ui.menu a.ui.inverted.button {
 margin-left: 0.5em;
}

.masthead h1.ui.header {
 font-size: 4em;
 font-weight: normal;
}

.masthead h2 {
 font-size: 1.7em;
 font-weight: normal;
}

/*end home page styles*/
```

7.2.1.4. models

7.2.1.4.1. activity.ts

```
import { Profile } from "./profile";

export interface Activity {
   id: string;
   title: string;
   date: Date | null;
   description: string;
   category: string;
   city: string;
   venue: string;
   hostUsername: string;
   isCancelled: boolean;
   isGoing: boolean;
   isHost: boolean;
   host?: Profile;
   attendees?: Profile[];
}

export class Activity implements Activity {
   constructor(init?: ActivityFormValues){
       Object.assign(this, init);
   }
}
```

```
export class ActivityFormValues {
   id?: string = undefined;
   title?: string = '';
   category: string = '';
   description: string = '';
   date: Date | null = null;
   city: string = '';
   venue: string = '';

   constructor(activity?: ActivityFormValues) {
       if(activity) {
           this.id = activity.id;
           this.title = activity.title;
           this.category = activity.category;
           this.description = activity.description;
           this.date = activity.date;
           this.venue = activity.venue;
           this.city = activity.city;
       }
   }
}
```

7.2.1.4.2. comment.ts

```
export interface ChatComment {
   id: number;
   createdAt: Date;
   body: string;
   username: string;
   displayName: string;
   image: string;
}
```

7.2.1.4.3. pagination.ts

```
export interface Pagination {
   currentPage: number;
   itemsPerPage: number;
   totalItems: number;
   totalPages: number;
}

export class PaginatedRusult<T> {
   data: T;
   pagination: Pagination;

   constructor(data: T, pagination: Pagination) {
       this.data = data;
       this.pagination = pagination;
   }
}

export class PagingParams {
   pageNumber;
   pageSize;

   constructor(pageNumber = 1, pageSize = 2){
       this.pageNumber = pageNumber;
       this.pageSize = pageSize;
   }

}
```

7.2.1.4.4. profile.ts

```
import { User } from "./user";

export interface Profile {
   username: string;
   displayName: string;
   image?: string;
   bio?: string;
   followersCount: number;
   followingCount: number;
   following: boolean;
   photos?: Photo[];

}

export class Profile implements Profile {
   constructor(user: User) {
       this.username = user.username;
       this.displayName = user.displayName;
       this.image = user.image;
   }
}
```

```
export interface Photo {
   id: string;
   url: string;
   isMain: boolean
}

export interface UserActivity {
   id: string;
   title: string;
   category: string;
   date: Date;
}
```

7.2.1.4.5. serverError.ts

```
export interface ServerError {
   statusCode: number;
   message: string;
   details: string;
}
```

7.2.1.4.6. user.ts

```
export interface User {
   username: string;
   displayName: string;
   token: string;
   image?: string;
}

export interface UserFormValues {
   email: string;
   password: string;
   displayName?: string;
   username?: string;
}
```

7.2.1.5. stores

7.2.1.5.1. activitzStore.ts

```
import { action, makeAutoObservable, makeObservable,
observable, reaction, runInAction } from "mobx";
import agent from "../api/agent";
import { Activity, ActivityFormValues } from
"../models/activity";
import {format} from 'date-fns';
import { store } from "./store";
import { Profile } from "../models/profile";
import { Pagination, PagingParams } from
"../models/pagination";
import { resourceLimits } from "worker_threads";

export default class ActivityStore {
   activityRegistry = new Map<string, Activity>();
   selectedActivity: Activity | undefined = undefined;
   editMode = false;
   loading = false; // is submiting
   loadingInitial = false;
   pagination: Pagination | null = null;
   pagingParams = new PagingParams();
   predicate = new Map().set('all', true);

   constructor() {
      makeAutoObservable(this);

      reaction(
          () => this.predicate.keys(),
          () => {
              this.pagingParams = new PagingParams();
              this.activityRegistry.clear();
              this.loadActivities();
          }
      )
   }

   setPagingParams = (pagingParams: PagingParams) => {
       this.pagingParams = pagingParams;
   }

   // COMUTED PROPERTIES
```

```
    get activitiesByDate() {
        return
Array.from(this.activityRegistry.values()).sort((a,b) =>
a.date!.getTime() - b.date!.getTime());
    }

    setPredicate = (predicate: string, value: string | Date) =>
{
        const resetPredicate = () => {
            this.predicate.forEach((value, key) => {
                if(key !== 'startDate')
this.predicate.delete(key);
            })
        }
        switch (predicate) {
            case 'all':
                resetPredicate();
                this.predicate.set('all', true);
                break;
            case 'isGoing':
                resetPredicate();
                this.predicate.set('isGoing', true);
                break;
            case 'isHost':
                resetPredicate();
                this.predicate.set('isHost', true);
                break;
            case 'startDate':
                this.predicate.delete('startDate');
                this.predicate.set('startDate', value);
        }
    }

    get axiosParams() {
        const params = new URLSearchParams();
        params.append('pageNumber',
this.pagingParams.pageNumber.toString());
        params.append('pageSize',
this.pagingParams.pageSize.toString());
        this.predicate.forEach((value, key) => {
            if(key === 'startDate') {
                params.append(key, (value as
Date).toISOString())
            } else {
                params.append(key, value);
            }
        })
        return params;
    }

    // COMUTED GROUPED ACTIVITIES
    get groupedActivities() {
        return Object.entries(
            this.activitiesByDate.reduce((activities, activity)
=> {
                const date = format(activity.date!, 'dd MMM
yyyy');
                activities[date] = activities[date] ?
[...activities[date], activity] : [activity];
                return activities;
            }, {} as {[key: string]: Activity[]})
        )
    }

    // ACTIONS
    loadActivities = async () => {
        this.loadingInitial = true;
        try {
            const result = await
agent.Activities.list(this.axiosParams);

            result.data.forEach(activity => {
                this.setActivity(activity);
            });
            this.setPagination(result.pagination);
            this.setLoadingInitial(false);
        } catch (error) {
            console.log(error);
            this.setLoadingInitial(false);

        }
    }

    setPagination = (pagination: Pagination) => {
        this.pagination = pagination;
    }

    loadingActivity = async (id: string) => {
        let activity = this.getActivity(id);
        if(activity) {
            this.selectedActivity = activity;
            return activity;
        } else {
            this.loadingInitial = true;
            try {
                activity = await agent.Activities.details(id);
                this.setActivity(activity);
                this.selectedActivity = activity;
                this.setLoadingInitial(false);
                return activity
            } catch (error) {
                console.log(error);
                this.setLoadingInitial(false);
            }
        }
    }

    private setActivity = (activity: Activity) => {
        const user = store.userStore.user;
        if(user) {
            activity.isGoing = activity.attendees!.some(
                a => a.username === user.username
            )
            activity.isHost = activity.hostUsername ===
user.username;
            activity.host = activity.attendees?.find(x =>
x.username === activity.hostUsername);
        }
        activity.date = new Date(activity.date!);
        this.activityRegistry.set(activity.id, activity);
    }

    private getActivity = (id: string) => {
        return this.activityRegistry.get(id);
    }

    setLoadingInitial = (state: boolean) => {
        this.loadingInitial = state;
    }

    /* POST  */
    // handle create or edit activty
    createActivity = async (activity: ActivityFormValues) => {
        const user = store.userStore.user;
        const attendee = new Profile(user!);
        try {
            await agent.Activities.create(activity);
            const newActivity = new Activity(activity);
            newActivity.hostUsername = user!.username;
            newActivity.attendees = [attendee];
            this.setActivity(newActivity);
            runInAction(() => {
                this.selectedActivity = newActivity;
            })
        } catch (error) {
            console.log(error);

        }
    }
    /* UPDATE */
    updateActivity = async(activity: ActivityFormValues) => {
        try {
            await agent.Activities.update(activity);
            runInAction(() => {
                if(activity.id){
                    let updatedActivity =
{...this.getActivity(activity.id), ...activity}
                    this.activityRegistry.set(activity.id,
updatedActivity as Activity);
                    this.selectedActivity = updatedActivity as
Activity;
                }
            })
        } catch (error) {
            console.log(error);
        }
    }
    /* DELETE */
    // handle delete actvity from list
    deleteActivity = async(id: string) => {
        this.setLoading(true);
        try {
            await agent.Activities.delete(id);
            this.activityRegistry.delete(id);
            this.setLoading(false);
        } catch (error) {
            console.log(error);
            this.setLoading(false);
        }
    }
    setLoading = (state: boolean) => {
        this.loading = state;
    }

    updateAttendance = async () => {
        const user = store.userStore.user;
        this.loading = true;
        try {
            await
agent.Activities.attend(this.selectedActivity!.id)
            runInAction(() => {
                if(this.selectedActivity?.isGoing) {
                    this.selectedActivity.attendees =
this.selectedActivity.attendees?.filter(a => a.username !==
user?.username);
                    this.selectedActivity.isGoing = false;
                } else {
                    const attendee = new Profile(user!);

this.selectedActivity?.attendees?.push(attendee);
                    this.selectedActivity!.isGoing = true;
                }

this.activityRegistry.set(this.selectedActivity!.id,
this.selectedActivity!);
            })
        } catch (error) {
            console.log(error);
        } finally {
            runInAction(() => this.loading = false);
        }
    }

    cancelActivityToggle = async () => {
        this.loading = true;
        try {
            await
agent.Activities.attend(this.selectedActivity!.id);
            runInAction(() => {
                this.selectedActivity!.isCancelled =
 !this.selectedActivity?.isCancelled;

 this.activityRegistry.set(this.selectedActivity!.id,
 this.selectedActivity!);
            })
        } catch (error) {
            console.log(error);
        } finally {
            runInAction(() => this.loading = false);
        }
    }

    clearSelectedActivity = () => {
        this.selectedActivity = undefined;
    }

    updateAttendeeFollowing = (username: string) => {
        this.activityRegistry.forEach(activity => {
            activity.attendees?.forEach(attendee => {
                if(attendee.username === username){
                    attendee.following ?
 attendee.followersCount-- : attendee.followersCount++;
                    attendee.following = !attendee.following;
                }
            })
        })
    }

 }
```

7.2.1.5.2. commentStore.ts

```
import { HubConnection, HubConnectionBuilder, LogLevel } from
"@microsoft/signalr";
import { makeAutoObservable, runInAction } from "mobx";
import { ChatComment } from "../models/comment";
import { store } from "./store";

export default class CommentStore {
   comments: ChatComment[] = [];
   hubConnection: HubConnection | null = null;

   constructor() {
       makeAutoObservable(this);
   }

   createHubConnection = (activityId: string) => {
       if(store.activityStore.selectedActivity){
           this.hubConnection = new HubConnectionBuilder()
               .withUrl(process.env.REACT_APP_CHAT_URL +
'?activityId=' + activityId, {
                   accessTokenFactory: () =>
store.userStore.user?.token!
               })
               .withAutomaticReconnect()
               .configureLogging(LogLevel.Information)
               .build();

           this.hubConnection.start().catch(error =>
console.log('Error establishing the connection', error));

           this.hubConnection.on('LoadComments', (comments:
ChatComment[]) => {
               runInAction(() => {
                   comments.forEach(comment => {
                       comment.createdAt = new
Date(comment.createdAt + 'Z');
                   });
                   this.comments = comments
               });
           })

           this.hubConnection.on('ReceiveComment', (comment:
ChatComment) => {
               runInAction(() => {
                   comment.createdAt = new
Date(comment.createdAt);
                   this.comments.unshift(comment)
               });
           })
       }
   }

   stopHubConnection = () => {
       this.hubConnection?.stop().catch(error =>
console.log('Error stopping connection:', error));
   }
```

```
   clearComments = () => {
       this.comments = [];
       this.stopHubConnection();
   }

   addComment = async (values: any) => {
       values.activityId =
store.activityStore.selectedActivity?.id;
       try {
           await this.hubConnection?.invoke('SendComment',
values);
       } catch (error) {
           console.log(error)
       }
   }
}
```

7.2.1.5.3. commonStore.ts

```
import { makeAutoObservable, reaction } from "mobx";
import { ServerError } from "../models/serverError";

export default class CommonStore {
   // var
   error: ServerError | null = null;
   token: string | null = window.localStorage.getItem('jwt');
   appLoaded = false;

   // binding
   constructor() {
       makeAutoObservable(this);

       reaction(
           () => this.token,
           token => {
               if(token) {
                   window.localStorage.setItem('jwt', token)
               }else {
                   window.localStorage.removeItem('jwt')
               }
           }
       )
   }
```

```
    // action
    setServerError = (error: ServerError) => {
        this.error = error;
    }

    setToken = (token: string | null) => {
        // if(token) window.localStorage.setItem('jwt', token);
        this.token = token;
    }

    setAppLoaded = () => {
        this.appLoaded = true;
    }

}
```

7.2.1.5.4. modalStore.ts

```
import { makeAutoObservable } from "mobx"

interface Modal {
    open: boolean;
    body: JSX.Element | null;
}

export default class ModalStore {
    modal: Modal = {
        open: false,
        body: null,
    }

    constructor() {
        makeAutoObservable(this)
    }

    openModal = (content: JSX.Element) => {
        this.modal.open = true;
        this.modal.body = content;
    }

    closeModal = () => {
        this.modal.open = false;
        this.modal.body = null;
    }
}
```

7.2.1.5.5. profileStore.ts

```
import { makeAutoObservable, reaction, runInAction } from
"mobx";
import agent from "../api/agent";
import { Photo, Profile, UserActivity } from
"../models/profile";
import { store } from "./store";

export default class ProfileStore {
    profile: Profile | null = null;
    loadingProfile = false;
    uploading = false;
    loading = false;
    followings: Profile[] = [];
    loadingFollowings = false;
    activeTab = 0;
    userActivities: UserActivity[] = [];
    loadingActivities = false;

    constructor(){
        makeAutoObservable(this);

        reaction(
            () => this.activeTab,
            activeTab => {
                if(activeTab === 3 || activeTab === 4){
                    const predicate = activeTab === 3 ?
'followers' : 'following';
                    this.loadFollowings(predicate);
                } else {
                    this.followings = [];
                }
            }
        )
    }

    setActiveTab = (activeTab: any) => {
        this.activeTab = activeTab;
    }
    get isCurrentUser() {
        if (store.userStore.user && this.profile){
            return store.userStore.user.username ===
this.profile.username;
        }
        return false;
    }

    loadProfile = async (username: string) => {
        this.loadingProfile = true;
        try {
            const profile = await agent.Profiles.get(username);
            runInAction(() => {
                this.profile = profile;
                this.loadingProfile = false;
            })
        } catch (error) {
            console.log(error);
            runInAction(() => this.loadingProfile = false)
        }
    }

    uploadPhoto = async (file: Blob) => {
        this.uploading = true;
        try {
            const response = await
agent.Profiles.uploadPhoto(file);
            const photo = response.data;
            runInAction(() => {
                if(this.profile) {
                    this.profile.photos?.push(photo);
                    if(photo.isMain && store.userStore.user){
                        store.userStore.setImage(photo.url);
                        this.profile.image = photo.url;
                    }
                }
                this.uploading = false;
            })
        } catch (error) {
            console.log(error);
            runInAction(() => this.uploading = false);
        }
    }
```

```
   setMainPhoto = async (photo: Photo) => {
       this.loading = true;
       try {
           await agent.Profiles.setMinPhoto(photo.id);
           store.userStore.setImage(photo.url);
           runInAction(() => {
               if(this.profile && this.profile.photos){
                   this.profile.photos.find(p =>
p.isMain)!.isMain = false;
                   this.profile.photos.find(p => p.id ==
photo.id)!.isMain = true;
                   this.profile.image = photo.url;
                   this.loading = false;
               }
           })
       } catch (error) {
           runInAction(() => this.loading = false);
           console.log(error);
       }
   }

   updateProfile = async (profile: Partial<Profile>) => {
       this.loading = true;
       try {
           await agent.Profiles.updateProfile(profile);
               runInAction(() => {
                   if (profile.displayName &&
profile.displayName !== store.userStore.user?.displayName) {

store.userStore.setDisplayName(profile.displayName);
                   }
                   this.profile = {...this.profile, ...profile
as Profile};
                   this.loading = false;
               })
       } catch (error) {
           console.log(error);
           runInAction(() => this.loading = false);
       }
   }

   deletePhoto = async (photo: Photo) => {
       this.loading = true;
       try {
           await agent.Profiles.deletePhoto(photo.id);
           runInAction(() => {
               if(this.profile){
                   this.profile.photos =
this.profile.photos?.filter(p => p.id !== photo.id);
                   this.loading = false;
               }
           })
       } catch (error) {
           runInAction(() => this.loading = false);
           console.log(error);
       }
   }

   updateFollowing = async (username: string, following:
boolean) => {
       this.loading = true;
       try {
           await agent.Profiles.updateFollowing(username);

store.activityStore.updateAttendeeFollowing(username);
           runInAction(() => {
               if(this.profile && this.profile.username !==
store.userStore.user?.username && this.profile.username ===
username){
                   following ? this.profile.followersCount++ :
this.profile.followersCount--;
                   this.profile.following =
!this.profile.following;
               }
               if(this.profile && this.profile.username ===
store.userStore.user?.username){
                   following ? this.profile.followersCount++ :
this.profile.followersCount--;
               }
               this.followings.forEach(profile => {
                   if(profile.username === username){
                       profile.following ?
profile.followersCount-- : profile.followersCount++;
                       profile.following = !profile.following;
                   }
               })
               this.loading = false;
           })
       } catch (error) {
           console.log(error);
           runInAction(() => this.loading = false);
       }
   }

   loadFollowings = async (predicate: string) => {
       this.loadingFollowings = true;
       try {
           const followings = await
agent.Profiles.listFollowings(this.profile!.username,
predicate);
           runInAction(() => {
               this.followings = followings;
               this.loadingFollowings = false;
           })
       } catch (error) {
           console.log(error);
           runInAction(() => this.loadingFollowings = false);
       }
   }

   loadUserActivities = async (username: string, predicate?:
string) => {
       this.loadingActivities = true;
       try {
           const activities = await
agent.Profiles.listActivities(username, predicate!);
           runInAction(() => {
               this.userActivities = activities;
               this.loadingActivities = false;
           })
       } catch (error) {
           console.log(error);
           runInAction(() => {
               this.loadingActivities = false;
           })
       }
   }
}
```

7.2.1.5.6. store.ts

```
import { createContext, useContext } from "react";
import ActivityStore from "./activityStore";
import CommentStore from "./commentStore";
import CommonStore from "./commonStore";
import ModalStore from "./modalStore";
import ProfileStore from "./profileStore";
import UserStore from "./userStore";

interface Store {
   activityStore: ActivityStore;
   commonStore: CommonStore;
   userStore: UserStore;
   modalStore: ModalStore;
   profileStore: ProfileStore;
   commentStore: CommentStore;
}

export const store: Store = {
   activityStore: new ActivityStore(),
   commonStore: new CommonStore(),
   userStore: new UserStore(),
   modalStore: new ModalStore(),
   profileStore: new ProfileStore(),
   commentStore: new CommentStore(),
}

export const StoreContext = createContext(store);

export function useStore() {
   return useContext(StoreContext);
}
```

7.2.1.5.7. userStore.ts

```
import { makeAutoObservable, runInAction } from "mobx";
import { history } from "../..";
import agent from "../api/agent";
import { User, UserFormValues } from "../models/user";
import { store } from "./store";

export default class UserStore {
   user: User | null = null;
```

```
constructor() {
    makeAutoObservable(this)
}

get isLoggedIn() {
    return !!this.user;
}

login = async (creds: UserFormValues) => {
    try {
        const user = await agent.Account.login(creds);
        store.commonStore.setToken(user.token);
        runInAction(() => this.user = user);
        history.push('/activities');
        store.modalStore.closeModal();
    } catch (error) {
        throw error;
    }
}

logout = () => {
    store.commonStore.setToken(null);
    window.localStorage.removeItem('jwt');
    this.user = null;
    history.push('/');
}

getUser = async () => {
    try {
        const user = await agent.Account.current();
        runInAction(() => this.user = user);
    } catch (error) {
        console.log(error);
    }
}

register = async (creds: UserFormValues) => {
    try {
        const user = await agent.Account.register(creds);
        store.commonStore.setToken(user.token);
        runInAction(() => this.user = user);
        history.push('/activities');
        store.modalStore.closeModal();
```

```
            } catch (error) {
                throw error;
            }
        }

        setImage = (image: string) => {
            if(this.user) this.user.image = image;
        }

        setDisplayName = (name: string) => {
            if (this.user) this.user.displayName = name;
        }
    }
```

7.2.2. feaures

7.2.2.1. activities

7.2.2.1.1. dashboard

7.2.2.1.1.1. ActivitiesListItemAttendee.tsx

```
import { observer } from 'mobx-react-lite'
import React from 'react'
import { Link } from 'react-router-dom'
import { Image, List, Popup } from 'semantic-ui-react'
import { Profile } from '../../../app/models/profile'
import ProfileCard from '../../profiles/ProfileCard'

interface Props {
   attendees: Profile[];
}

export default observer(function
ActivitiesListItemAttendee({attendees}: Props) {
   const styles = {
       borderColor: 'orange',
       borderWidth: 3
   }
   return (
       <List horizontal>
           {attendees.map(attendee => (
               <Popup
                   hoverable
                   key={attendee.username}
                   trigger={
                       <List.Item
key={attendee.username} as={Link}
to={`/profiles/${attendee.username}`}>
                           <Image
                               size='mini'
                               circular
                               src={attendee.image ||
'/assets/user.png'}
                               bordered
                               style={attendee.following
? styles : null}
                           />
                       </List.Item>
                   }
               >
                   <Popup.Content>
                       <ProfileCard profile={attendee}
/>
                   </Popup.Content>
               </Popup>

           ))}
       </List>
   )
})
```

7.2.2.1.1.2. ActivityDashboard

```
import { observer } from 'mobx-react-lite'
import React, {useState, useEffect } from 'react'
import InfiniteScroll from 'react-infinite-scroller'
import { Grid, Loader } from 'semantic-ui-react'
import { PagingParams } from
'../../../app/models/pagination'
import { useStore } from '../../../app/stores/store'
import ActivityFilters from './ActivityFilters'
import ActivityList from './ActivityList'
import ActivityListItemPlaceholder from
'./ActivityListItemPlaceholder'
```

```
export default observer(function ActivityDashboard() {
   const { activityStore } = useStore();
   const { loadActivities, activityRegistry,
setPagingParams, pagination } = activityStore;
   const [loadingNext, setLoadingNext] =
useState(false);

   const handleGetNext = () => {
       setLoadingNext(true);
       setPagingParams(new
PagingParams(pagination!.currentPage + 1))
       loadActivities().then(() =>
setLoadingNext(false));
   }

   useEffect(() => {
       if(activityRegistry.size === 0) loadActivities();
   }, [activityRegistry.size, loadActivities]);

   // if (activityStore.loadingInitial && !loadingNext)
return <LoadingComponents content='Loading
activities...' />

   return (
       <Grid>
           <Grid.Column width='10'>
               {activityStore.loadingInitial &&
!loadingNext ? (
                   <>
                       <ActivityListItemPlaceholder />
                       <ActivityListItemPlaceholder />
                   </>
               ) : (
                   <InfiniteScroll
                       pageStart={0}
                       loadMore={handleGetNext}
                       hasMore={!loadingNext &&
!!pagination && pagination.currentPage <
pagination.totalPages}
                       initialLoad={false}
                   >
                       <ActivityList />
                   </InfiniteScroll>
```

```
                )}

            </Grid.Column>
            <Grid.Column width='6'>
               <ActivityFilters />
            </Grid.Column>
            <Grid.Column width={10}>
                <Loader active={loadingNext} />
            </Grid.Column>
        </Grid>
   )
});
```

7.2.2.1.1.3. ActivityFilters

```
import { observer } from 'mobx-react-lite'
import React from 'react'
import Calendar from 'react-calendar'
import { Header, Menu } from 'semantic-ui-react'
import { useStore } from '../../../app/stores/store'

export default observer (function ActivityFilters() {
   const { activityStore: {predicate, setPredicate} } =
useStore();
   return (
       <>
           <Menu vertical size='large' style={{width:
'100%', marginTop: 25}}>
               <Header icon='filter' attached
color='teal' content='Filters' />
               <Menu.Item
                   content='All Activities'
                   active={predicate.has('all')}
                   onClick={() => setPredicate('all',
'true')}
               />
               <Menu.Item
                   content="I' going"
                   active={predicate.has('isGoing')}
                   onClick={() =>
setPredicate('isGoing', 'true')}
               />
               <Menu.Item
```

```
                   content="I'm hosting"
                   active={predicate.has('isHost')}
                   onClick={() => setPredicate('isHost',
'true')}
               />
           </Menu>
           <Header />
           <Calendar
               onChange={(date : Date) =>
setPredicate('startDate', date)}
               value={predicate.get('startDate') || new
Date()}

           />
       </>

   )
})
```

7.2.2.1.1.4. ActivityList

```
import { observer } from 'mobx-react-lite';
import { Fragment } from 'react';
import { Header } from 'semantic-ui-react'
import { useStore } from '../../../app/stores/store';
import ActivityListItem from './ActivityListItem';

export default observer(function ActivityList() {
   const { activityStore } = useStore();
   const { groupedActivities } = activityStore;

   return (
       <>
           {groupedActivities.map(([group, activities])
=> (
               <Fragment key={group}>
                   <Header sub color='teal'>
                       {group}
                   </Header>
                   {activities?.map((activity) => (
                       <ActivityListItem
key={activity.id} activity={activity} />
                   ))}
               </Fragment>
           ))}
       </>
   )
});
```

7.2.2.1.1.5. ActivityListItem

```
import React from 'react'
import { format } from 'date-fns';
import { Link } from 'react-router-dom'
import { Button, Icon, Item, Label, Segment } from
'semantic-ui-react'
import { Activity } from '../../../app/models/activity'
import ActivitiesListItemAttendee from
'./ActivitiesListItemAttendee';

interface Props {
   activity: Activity;
}

export default function ActivityListItem({activity}:
Props) {
   console.log(activity);
   return (
      <Segment.Group>
          <Segment>
              {activity.isCancelled && (
                  <Label
                       attached='top'
                       color='red'
                       content='Cancelled'
                       style={{textAlign: 'center'}}
                  />
              )}
              <Item.Group>
                   <Item>
                       <Item.Image style={{marginBottom:
4}} size='tiny' circular src={activity.host?.image ||
'/assets/user.png'} />
                       <Item.Content>
                           <Item.Header as={Link}
to={`/activities/${activity.id}`}>
```

```
                                {activity.title}
                            </Item.Header>
                            <Item.Description>Hosted by
<Link
to={`/profiles/${activity.hostUsername}`}>{activity.host
?.displayName}</Link></Item.Description>
                            {activity.isHost && (
                                <Item.Description>
                                    <Label basic
color='orange'>
                                        You are hosting
this activity
                                    </Label>
                                </Item.Description>
                            )}
                            {activity.isGoing &&
!activity.isHost && (
                                <Item.Description>
                                    <Label basic
color='green'>
                                        You are going to
this activity
                                    </Label>
                                </Item.Description>
                            )}
                        </Item.Content>
                    </Item>
               </Item.Group>
           </Segment>
           <Segment>
               <span>
                   <Icon name='clock'
/>{format(activity.date!, 'dd MMM yyyy h:mm aa')}
                   <Icon name='marker'/> {activity.venue}
               </span>
           </Segment>
           <Segment secondary>
                <ActivitiesListItemAttendee
attendees={activity.attendees!} />
           </Segment>
           <Segment clearing>
                <span>{activity.description}</span>
                <Button
                    as={Link}
```

Beograd, 2022.

```
                    to={`/activities/${activity.id}`}
                    color='teal'
                    floated='right'
                    content='View'
                />
            </Segment>
        </Segment.Group>
    )
}
```

7.2.2.1.1.6. ActivityListItemPlaceholder

```
import React from 'react';
import { Segment, Button, Placeholder } from 'semantic-
ui-react';

export default function ActivityListItemPlaceholder() {
    return (
        <>
            <Placeholder fluid style={{ marginTop: 25 }}>
                <Segment.Group>
                    <Segment style={{ minHeight: 110 }}>
                        <Placeholder>
                            <Placeholder.Header image>
                                <Placeholder.Line />
                                <Placeholder.Line />
                            </Placeholder.Header>
                            <Placeholder.Paragraph>
                                <Placeholder.Line />
                            </Placeholder.Paragraph>
                        </Placeholder>
                    </Segment>
                    <Segment>
                        <Placeholder>
                            <Placeholder.Line />
                            <Placeholder.Line />
                        </Placeholder>
                    </Segment>
                    <Segment secondary style={{
minHeight: 70 }} />
                    <Segment clearing>
                        <Button disabled color='blue'
floated='right' content='View' />
```

```
                    </Segment>
                </Segment.Group>
            </Placeholder>
        </>
   );
};
```

7.2.2.1.2. details

7.2.2.1.2.1. ActivityDetailedChat.tsx

```
import { Formik, Form, Field, FieldProps } from 'formik'
import { observer } from 'mobx-react-lite'
import React, {useEffect} from 'react'
import { Link } from 'react-router-dom'
import {Segment, Header, Comment, Button, Loader} from
'semantic-ui-react'
import { useStore } from '../../../app/stores/store'
import * as Yup from 'yup';
import { formatDistanceToNow } from 'date-fns'

interface Props {
   activityId: string;
}

export default observer(function
ActivityDetailedChat({activityId}: Props) {
   const {commentStore} = useStore();

   useEffect(() => {
      if(activityId){
          commentStore.createHubConnection(activityId);
      }
      return () => {
          commentStore.clearComments();
      }
   }, [commentStore, activityId]);
   return (
       <>
           <Segment
               textAlign='center'
               attached='top'
```

```
                inverted
                color='teal'
                style={{border: 'none'}}
            >
                <Header>Chat about this event</Header>
            </Segment>
            <Segment attached clearing>
                <Formik
                    onSubmit={(values, {resetForm}) =>

commentStore.addComment(values).then(() => resetForm())}
                    initialValues={{body: ''}}
                    validationSchema={Yup.object({
                        body: Yup.string().required()
                    })}
                >
                    {({isSubmitting, isValid,
handleSubmit}) => (
                        <Form className='ui form'>
                            <Field name='body'>
                                {(props: FieldProps) => (
                                    <div
style={{position: 'relative'}}>
                                        <Loader
active={isSubmitting} />
                                        <textarea

placeholder='Enter your comment (Enter to submit, SHIFT
+ enter for new line)'
                                            rows={2}

{...props.field}
                                            onKeyPress={e
=> {
                                                if(e.key
=== 'Enter' && e.shiftKey){

return;
                                                }
                                                if(e.key
=== 'Enter' && !e.shiftKey){

e.preventDefault();
isValid && handleSubmit();
                                                }
                                            }}
                                        />
                                    </div>
                                )}
                            </Field>
                        </Form>
                    )}
                </Formik>
                <Comment.Group>
                    {commentStore.comments.map(comment =>
(
                        <Comment key={comment.id}>
                            <Comment.Avatar
src={comment.image || '/assets/user.png'}/>
                            <Comment.Content>
                                <Comment.Author as={Link}
to={`/profiles/${comment.username}`}>
                                    {comment.displayName}
                                </Comment.Author>
                                <Comment.Metadata>

<div>{formatDistanceToNow(comment.createdAt)} ago</div>
                                </Comment.Metadata>
                                <Comment.Text
style={{whiteSpace: 'pre-
wrap'}}>{comment.body}</Comment.Text>
                            </Comment.Content>
                        </Comment>
                    ))}
                </Comment.Group>
            </Segment>
        </>

    )
})
```

7.2.2.1.2.2. ActivityDetailedHeader

```
import { observer } from 'mobx-react-lite';
import React from 'react';
import { format } from 'date-fns';
import { Link } from 'react-router-dom';
import {Button, Header, Item, Segment, Image, Label}
from 'semantic-ui-react'
import {Activity} from "../../../app/models/activity";
import { useStore } from '../../../app/stores/store';

const activityImageStyle = {
   filter: 'brightness(30%)'
};

const activityImageTextStyle = {
   position: 'absolute',
   bottom: '5%',
   left: '5%',
   width: '100%',
   height: 'auto',
   color: 'white'
};

interface Props {
   activity: Activity
}

export default observer (function
ActivityDetailedHeader({activity}: Props) {
   const {activityStore: {updateAttendance, loading,
cancelActivityToggle}} = useStore();
   return (
       <Segment.Group>
           <Segment basic attached='top'
style={{padding: '0'}}>
               {activity.isCancelled && (
                   <Label
                       style={{position: 'absolute',
zIndex: 1000, left: -14, top: 20}}
                       ribbon
                       color='red'
                       content='Cancelled'
                   />
               )}
               <Image
src={`/assets/categoryImages/${activity.category}.jpg`}
fluid style={activityImageStyle}/>
```

```
               <Segment style={activityImageTextStyle}
basic>
                   <Item.Group>
                       <Item>
                           <Item.Content>
                               <Header
                                   size='huge'

content={activity.title}
                                   style={{color:
'white'}}
                               />

<p>{format(activity.date!, 'dd MMM yyyy')}</p>
                               <p>
                                   Hosted by
<strong><Link
to={`/profiles/${activity.host?.username}`}>{activity.ho
st?.displayName}</Link></strong>
                               </p>
                           </Item.Content>
                       </Item>
                   </Item.Group>
               </Segment>
           </Segment>
           <Segment clearing attached='bottom'>
               {activity.isHost ? (
                   <>
                       <Button
                           color={activity.isCancelled ?
'green' : 'red'}
                           floated='left'
                           basic
                           content={activity.isCancelled
? 'Re-activate Activity' : 'Cancel Activity'}

onClick={cancelActivityToggle}
                           loading={loading}
                       />
                       <Button

disabled={activity.isCancelled}
                           as={Link}
                           to={`/manage/${activity.id}`}
                           color='orange'
                           floated='right'
                       >
                           Manage Event
                       </Button>
                   </>

               ) : activity.isGoing ? (
                   <Button loading={loading}
onClick={updateAttendance}>Cancel attendance</Button>
               ) : (
                   <Button
                       disabled={activity.isCancelled}
                       loading={loading}
                       onClick={updateAttendance}
                       color='teal'>
                           Join Activity
                   </Button>
               )}
           </Segment>
       </Segment.Group>
   )
})
```

7.2.2.1.2.3. ActivtyDeatiledInfo

```
import { observer } from 'mobx-react-lite';
import React from 'react'
import {format} from 'date-fns';
import {Segment, Grid, Icon} from 'semantic-ui-react'
import {Activity} from "../../../app/models/activity";

interface Props {
   activity: Activity
}

export default observer(function
ActivityDetailedInfo({activity}: Props) {
   return (
       <Segment.Group>
           <Segment attached='top'>
               <Grid>
```

```
                   <Grid.Column width={1}>
                       <Icon size='large' color='teal'
name='info'/>
                   </Grid.Column>
                   <Grid.Column width={15}>
                       <p>{activity.description}</p>
                   </Grid.Column>
               </Grid>
           </Segment>
           <Segment attached>
               <Grid verticalAlign='middle'>
                   <Grid.Column width={1}>
                       <Icon name='calendar'
size='large' color='teal'/>
                   </Grid.Column>
                   <Grid.Column width={15}>
           <span>
             {format(activity.date!, 'dd MMM yyyy h:mm
aa')}
           </span>
                   </Grid.Column>
               </Grid>
           </Segment>
           <Segment attached>
               <Grid verticalAlign='middle'>
                   <Grid.Column width={1}>
                       <Icon name='marker' size='large'
color='teal'/>
                   </Grid.Column>
                   <Grid.Column width={11}>
                       <span>{activity.venue},
{activity.city}</span>
                   </Grid.Column>
               </Grid>
           </Segment>
       </Segment.Group>
   )
})
```

7.2.2.1.2.4. ActivityDetailedSidebar

```
import React from 'react'
import { Segment, List, Label, Item, Image } from
'semantic-ui-react'
import { Link } from 'react-router-dom'
import { observer } from 'mobx-react-lite'
import { Activity } from '../../../app/models/activity'

interface Props {
   activity: Activity
}

export default observer(function ActivityDetailedSidebar
({activity: {attendees, host}}: Props) {
   if(!attendees) return null;
   return (
       <>
           <Segment
               textAlign='center'
               style={{ border: 'none' }}
               attached='top'
               secondary
               inverted
               color='teal'
           >
               {attendees.length} {attendees.length ===
1 ? 'Person': 'People'} going
           </Segment>
           <Segment attached>
               <List relaxed divided>
                   {attendees.map(attendee => (
                       <Item style={{ position:
'relative' }} key={attendee.username}>
                           {attendee.username ===
host?.username &&
                               <Label
                                   style={{ position:
'absolute' }}
                                   color='orange'
                                   ribbon='right'
                               >
                                   Host
                               </Label>
                           }
                           <Image size='tiny'
src={attendee.image || '/assets/user.png'} />
```

Beograd, 2022.

```
                                <Item.Content
verticalAlign='middle'>
                                    <Item.Header as='h3'>
                                        <Link
to={`/profiles/${attendee.username}`}>{attendee.displayN
ame}</Link>
                                    </Item.Header>
                                    {attendee.following && (
                                        <Item.Extra style={{
color: 'orange' }}>Following</Item.Extra>
                                    )}
                                </Item.Content>
                            </Item>
                        ))}
                    </List>
                </Segment>
            </>

    )
})
```

7.2.2.1.2.5. ActivityDetails

```
import { observer } from 'mobx-react-lite';
import React, { useEffect } from 'react'
import { useParams } from 'react-router';
import { Grid } from 'semantic-ui-react';
import LoadingComponent from
'../../../app/layout/LoadingComponent';
import { useStore } from '../../../app/stores/store';
import ActivityDetailedChat from
'./ActivityDetailedChat';
import ActivityDetailedHeader from
'./ActivityDetailedHeader';
import ActivityDetailedInfo from
'./ActivityDetailedInfo';
import ActivityDetailedSidebar from
'./ActivityDetailedSidebar';

export default observer(function ActivityDetails() {
   const { activityStore } = useStore();
   const { selectedActivity: activity, loadingActivity,
loadingInitial, clearSelectedActivity } = activityStore;
```

```
   const { id } = useParams<{id: string}>();

   useEffect(() => {
       if(id) loadingActivity(id);
       return () => clearSelectedActivity();
   }, [id, loadingActivity, clearSelectedActivity]);

   if(loadingInitial || !activity) return
<LoadingComponent/>;

   return (
       <Grid>
           <Grid.Column width={10}>
               <ActivityDetailedHeader
activity={activity} />
               <ActivityDetailedInfo activity={activity}
/>
               <ActivityDetailedChat
activityId={activity.id} />
           </Grid.Column>
           <Grid.Column width={6}>
               <ActivityDetailedSidebar
activity={activity} />
           </Grid.Column>
       </Grid>
   )
});
```

7.2.2.1.3. form

7.2.2.1.3.1. ActivityForm.tsx

```
import { observer } from 'mobx-react-lite'
import {useEffect, useState} from 'react'
import { useHistory, useParams } from 'react-router';
import { Button, Header, Segment } from 'semantic-ui-
react'
import LoadingComponent from
'../../../app/layout/LoadingComponent';
import { useStore } from '../../../app/stores/store'
import { v4 as uuid } from 'uuid';
import { Link } from 'react-router-dom';
import { Formik, Form } from 'formik';
import * as Yup from 'yup';
import MyTextInput from
'../../../app/common/form/MyTextInput';
import MyTextArea from
'../../../app/common/form/MyTextArea';
import MySelectInput from
'../../../app/common/form/MySelectInput';
import { categoryOptions } from
'../../../app/common/options/categoryOptions';
import MyDateInput from
'../../../app/common/form/MyDateInput';
import { ActivityFormValues } from
'../../../app/models/activity';

export default observer(function ActivityForm() {
   const history = useHistory();
   const {activityStore} = useStore();
   const {createActivity, updateActivity, loading,
loadingActivity, loadingInitial} = activityStore;
   const { id } = useParams<{id: string}>();

   const [activity, setActivity] =
useState<ActivityFormValues>(new ActivityFormValues);

   const validationSchema = Yup.object({
       title: Yup.string().required('The activity title
is requried'),
       description: Yup.string().required('The activity
description is requried'),
       category: Yup.string().required(),
       date: Yup.string().required('Date is
requried').nullable(),
       venue: Yup.string().required(),
       city: Yup.string().required(),

   })

   useEffect(() => {
       if(id) loadingActivity(id).then(activity =>
setActivity(new ActivityFormValues(activity)));
   }, [id, loadingActivity]);
   // handle Submit
   const handleFormSubmit = (activity:
ActivityFormValues) => {
       if(!activity.id) {
           let newActivity = {
               ...activity,
               id: uuid()
           }
           createActivity(newActivity).then(() =>
history.push(`/activities/${newActivity.id}`))
       }else {
           updateActivity(activity).then(() =>
history.push(`/activities/${activity.id}`));
       }
   }

   if(loadingInitial) return <LoadingComponent
content='Loading activity...' />

   return (
       <Segment clearing>
           <Header content='Activity Details' sub
color='teal' />
           {/** initialValues and onSubmit are required
*/}
           <Formik
               validationSchema={validationSchema}
               enableReinitialize
               initialValues={activity}
               // values => activity
               onSubmit={values =>
handleFormSubmit(values)}>
               {/** these 4. props are from formik */}
               {({handleSubmit, isValid, isSubmitting,
dirty}) => (
                   <Form className='ui form'
onSubmit={handleSubmit} autoComplete='off'>
                   <MyTextInput name='title'
placeholder='Title' />
                   <MyTextArea rows={3}
name='description' placeholder='Description' />
```

```
                    <MySelectInput
options={categoryOptions} name='category'
placeholder='Category' />
                    <MyDateInput name='date'
placeholderText='Date' showTimeSelect timeCaption='time'
dateFormat='MMMM d, yyyy h:mm aa' />
                    <Header content='Location Details'
sub color='teal' />
                    <MyTextInput name='city'
placeholder='City' />
                    <MyTextInput name='venue'
placeholder='Venue' />
                    <Button
                        disabled={isSubmitting || !dirty
|| !isValid}
                        loading={isSubmitting}
                        floated='right'
                        positive
                        type='submit'
                        content='Submit'
                    />
                    <Button as={Link} to='/activities'
floated='right' type='button' content='Cancel' />
                </Form>
                )}
            </Formik>

        </Segment>
    )
});
```

7.2.2.2. errors

7.2.2.2.1. NotFound.tsx

```
import React from 'react'
import { Link } from 'react-router-dom'
import { Button, Header, Icon, Segment } from 'semantic-ui-
react'

export default function NotFound() {
   return (
       <Segment placeholder>
           <Header icon>
               <Icon name='search' />
               Oops - we've looked everywhere and could not
find this
           </Header>
           <Segment.Inline>
               <Button as={Link} to='/activities' primary>
                   Return to activities page
               </Button>
           </Segment.Inline>
       </Segment>
   )
}
```

7.2.2.2.2. ServerError

```
import { observer } from 'mobx-react-lite';
import React from 'react'
import { Container, Header, Segment } from 'semantic-ui-react'
import { useStore } from '../../app/stores/store'


export default observer(function ServerError() {
   const { commonStore } = useStore();
   return (
       <Container>
           <Header as='h1' content='Server Error' />
           <Header sub as='h5' color='red'
content={commonStore.error?.message} />
           {
               commonStore.error?.details &&
                   <Segment>
                       <Header as='h4' content='Stack trace'
color='teal' />
                       <code style={{marginTop:
'10px'}}>{commonStore.error.details}</code>
                   </Segment>
           }
       </Container>
   )
})
```

7.2.2.2.3. TestError

```
import React, {useState} from 'react';
import {Button, Header, Segment} from "semantic-ui-react";
import axios from 'axios';
import ValidationErrors from './ValidationErrors';

export default function TestErrors() {
   const baseUrl = process.env.REACT_APP_API_URL;
   const [errors, setErrors] = useState(null);

   function handleNotFound() {
       axios.get(baseUrl + 'buggy/not-found').catch(err =>
console.log(err.response));
   }

   function handleBadRequest() {
       axios.get(baseUrl + 'buggy/bad-request').catch(err =>
console.log(err.response));
   }

   function handleServerError() {
       axios.get(baseUrl + 'buggy/server-error').catch(err =>
console.log(err.response));
   }

   function handleUnauthorised() {
       axios.get(baseUrl + 'buggy/unauthorised').catch(err =>
console.log(err.response));
   }

   function handleBadGuid() {
       axios.get(baseUrl + 'activities/notaguid').catch(err =>
console.log(err.response));
   }

   function handleValidationError() {
       axios.post(baseUrl + 'activities', {}).catch(err =>
setErrors(err));
   }

   return (
       <>
```

```
            <Header as='h1' content='Test Error component' />
            <Segment>
                <Button.Group widths='7'>
                    <Button onClick={handleNotFound}
content='Not Found' basic primary />
                    <Button onClick={handleBadRequest}
content='Bad Request' basic primary />
                    <Button onClick={handleValidationError}
content='Validation Error' basic primary />
                    <Button onClick={handleServerError}
content='Server Error' basic primary />
                    <Button onClick={handleUnauthorised}
content='Unauthorised' basic primary />
                    <Button onClick={handleBadGuid} content='Bad
Guid' basic primary />
                </Button.Group>
            </Segment>
            {errors && (
                <ValidationErrors errors={errors} />
            )}
        </>
    )
}
```

7.2.2.2.4. ValidationErrors

```
import React from 'react'
import { Message } from 'semantic-ui-react'

interface Props {
    errors: any;
}

export default function ValidationErrors({errors}: Props) {
    return (
        <Message error>
            {errors && (
                <Message.List>
                    {errors.map((err: any, i: any) => (
                        <Message.Item key={i}>
                            {err}
                        </Message.Item>
                    ))}
                </Message.List>
            )}
        </Message>
    )
}
```

7.2.2.3. home

7.2.2.3.1. HomePage.tsx

```
import { observer } from 'mobx-react-lite'
import React from 'react'
import { Link } from 'react-router-dom'
import { Button, Container, Header, Image, Segment } from
'semantic-ui-react'
import { useStore } from '../../app/stores/store'
import LoginForm from '../users/LoginForm'
import RegisterForm from '../users/RegisterForm'

export default observer(function HomePage() {
    const {userStore, modalStore} = useStore()
    return (
        <Segment inverted textAlign='center' vertical
className='masthead'>
            <Container text>
                <Header as='h1' inverted >
                    <Image size='massive' src='/assets/logo.png'
alt='logo' style={{marginBottom: 12}} />
                    CodeDancing Activities
                </Header>
                {userStore.isLoggedIn ? (
                    <>
                     <Header as='h2' inverted content='Welcome
to CodeDancing Activities' />
                     <Button as={Link} to='/activities'
size='huge' inverted>
                         Go to Activities!
                     </Button>
                     </>
                ): (
                    <>
                    <Button onClick={() =>
modalStore.openModal(<LoginForm />)} size='huge' inverted>
                        Login!
                    </Button>
                    <Button onClick={() =>
modalStore.openModal(<RegisterForm />)} size='huge' inverted>
                        Register!
                    </Button>
                    </>
                )}

            </Container>
        </Segment>
    )
})
```

7.2.2.4. profiles

7.2.2.4.1. FollowButton.tsx

```
import { observer } from 'mobx-react-lite';
import React, { SyntheticEvent } from 'react';
import { Button, Reveal } from 'semantic-ui-react';
import { Profile } from '../../app/models/profile';
import { useStore } from '../../app/stores/store';

interface Props {
   profile: Profile;
}

export default observer(function FollowButton({profile}: Props)
{
   const {profileStore, userStore} = useStore();
   const {updateFollowing, loading} = profileStore;

   if(userStore.user?.username === profile.username) return
null;

   const handleFollow = (event: SyntheticEvent, username:
string) => {
       event.preventDefault();
       profile.following ? updateFollowing(username, false) :
updateFollowing(username, true);
   }
   return (
       <Reveal animated='move' >
       <Reveal.Content visible style={{width: '100%'}}>
           <Button
               fluid
               color='teal'
               content={profile.following ? 'Following' : 'Not
following'}
           />
       </Reveal.Content>
       <Reveal.Content hidden style={{width: '100%'}}>
           <Button
               fluid
               basic
               color={profile.following ? 'red' : 'green'}
               content={profile.following ? 'Unfollow' :
'Follow'}
               loading={loading}
               onClick={(event) => handleFollow(event,
profile.username)}
           />
       </Reveal.Content>
   </Reveal>
   )
})
```

7.2.2.4.2. ProfileAbout

```
import React, {useState} from 'react';
import {useStore} from "../../app/stores/store";
import {Button, Grid, Header, Tab} from "semantic-ui-react";
import ProfileEditForm from "./ProfileEditForm";
import { observer } from 'mobx-react-lite';

export default observer(function ProfileAbout() {
   const {profileStore} = useStore();
   const {isCurrentUser, profile} = profileStore;
   const [editMode, setEditMode] = useState(false);

   return (
       <Tab.Pane>
           <Grid>
               <Grid.Column width='16'>
               <Header floated='left' icon='user'
content={`About ${profile?.displayName}`} />
                   {isCurrentUser && (
                       <Button
                           floated='right'
                           basic
                           content={editMode ? 'Cancel' : 'Edit
Profile'}
                           onClick={() =>
setEditMode(!editMode)}
                       />
                   )}
               </Grid.Column>
               <Grid.Column width='16'>
                   {editMode
                       ? <ProfileEditForm
setEditMode={setEditMode} />
                       : <span style={{whiteSpace: 'pre-
wrap'}}>{profile?.bio}</span>
                   }
               </Grid.Column>
           </Grid>
       </Tab.Pane>
   )
})
```

7.2.2.4.3. ProfileActivities

```
import React, { SyntheticEvent, useEffect } from 'react';
import { observer } from 'mobx-react-lite';
import { Tab, Grid, Header, Card, Image, TabProps }
from 'semantic-ui-react'; import { Link } from 'react-router-
dom';
import { UserActivity } from '../../app/models/profile';
import { format } from 'date-fns';
import { useStore } from "../../app/stores/store";

const panes = [
   { menuItem: 'Future Events', pane: { key: 'future' } },
   { menuItem: 'Past Events', pane: { key: 'past' } },
   { menuItem: 'Hosting', pane: { key: 'hosting' } }
```

```
];

export default observer(function ProfileActivities() {
   const { profileStore } = useStore();
   const {
       loadUserActivities,
       profile,
       loadingActivities,
       userActivities
   } = profileStore;

   useEffect(() => {
       loadUserActivities(profile!.username);
   }, [loadUserActivities, profile]);

   const handleTabChange = (e: SyntheticEvent, data: TabProps)
=> {
       loadUserActivities(profile!.username,
panes[data.activeIndex as number].pane.key);
   };
   return (
       <Tab.Pane loading={loadingActivities}>
           <Grid>
               <Grid.Column width={16}>
                   <Header
                       floated='left'
                       icon='calendar'
                       content={'Activities'}
                   />
               </Grid.Column>
               <Grid.Column width={16}>
                   <Tab
                       panes={panes}
                       menu={{ secondary: true, pointing: true
}}
                       onTabChange={(e, data) =>
handleTabChange(e, data)}
                   />
               <br />
               <Card.Group itemsPerRow={4}>
                   {userActivities.map((activity: UserActivity)
=> (
                       <Card
                           as={Link}
                           to={`/activities/${activity.id}`}
                           key={activity.id}
                       >
                           <Image

src={`/assets/categoryImages/${activity.category}.jpg`}
                               style={{ minHeight: 100,
objectFit:'cover' }}
                           />
                           <Card.Content>
                               <Card.Header textAlign='center'>
                                       {activity.title}
                               </Card.Header>
                               <Card.Meta textAlign='center'>
                                   <div>{format(new
Date(activity.date), 'do LLL')}</div>
                                   <div>{format(new
Date(activity.date), 'h:mm a')}</div>
                               </Card.Meta>
                           </Card.Content>
                       </Card> ))}
                   </Card.Group>
               </Grid.Column>
           </Grid>
       </Tab.Pane>
   );
});
```

7.2.2.4.4. ProfileCard

```
import { observer } from 'mobx-react-lite'
import React from 'react'
import { Link } from 'react-router-dom'
import { Card, Icon, Image } from 'semantic-ui-react'
import { Profile } from '../../app/models/profile'
import FollowButton from './FollowButton'

interface Props {
   profile: Profile;
}

export default observer(function ProfileCard({profile}: Props)
{
   const handleTruncate = (str: string | undefined) => {
       if (str) {
           return str.length > 40 ? str.substring(0, 37) +
'...' : str;
       }
   }
   return (
       <Card as={Link} to={`/profiles/${profile.username}`}>
           <Image src={profile.image || '/assets/user.png'} />
           <Card.Content>
               <Card.Header>{profile.displayName}</Card.Header>

<Card.Description>{handleTruncate(profile.bio)}</Card.Descripti
on>
           </Card.Content>
           <Card.Content extra>
               <Icon name='user' />
               {profile.followersCount} followers
           </Card.Content>
           <FollowButton profile={profile} />
       </Card>
   )
})
```

7.2.2.4.5. ProfileContent

```
import { observer } from 'mobx-react-lite'
import React from 'react'
import { Tab } from 'semantic-ui-react'
import { Profile } from '../../app/models/profile'
import { useStore } from '../../app/stores/store'
import ProfileAbout from './ProfileAbout'
import ProfileActivities from './ProfileActivities'
import ProfileFollowings from './ProfileFollowings'
import ProfilePhotos from './ProfilePhotos'

interface Props {
    profile: Profile;
}

export default observer(function ProfileContent({profile}:
Props) {
    const {profileStore} = useStore();
```

```
   const panes = [
       {menuItem: 'About', render: () => <ProfileAbout /> },
       {menuItem: 'Photos', render: () => <ProfilePhotos
profile={profile!} /> },
       {menuItem: 'Events', render: () => <ProfileActivities />
},
       {menuItem: 'Followers', render: () => <ProfileFollowings
/> },
       {menuItem: 'Following', render: () => <ProfileFollowings
/> },
   ]
   return (
       <Tab
           menu={{fluid: true, vertical: true}}
           menuPosition='right'
           panes={panes}
           onTabChange={(event, data) =>
profileStore.setActiveTab(data.activeIndex)}
       />
   )
});
```

7.2.2.4.6. ProfileEditForm

```
import { Form, Formik } from "formik";
import { observer } from "mobx-react-lite";
import { Button } from "semantic-ui-react";
import MyTextArea from "../../app/common/form/MyTextArea";
import MyTextInput from "../../app/common/form/MyTextInput";
import { useStore } from "../../app/stores/store";
import * as Yup from 'yup';

interface Props {
   setEditMode: (editMode: boolean) => void;
}

export default observer(function ProfileEditForm({setEditMode}:
Props) {
   const {profileStore: {profile, updateProfile}} = useStore();
   return (
       <Formik
           initialValues={{displayName: profile?.displayName,
bio: profile?.bio}}
```

```
           onSubmit={values => {
                   updateProfile(values).then(() => {
setEditMode(false);
               })
           }}
           validationSchema={Yup.object({
               displayName: Yup.string().required()
           })}
       >
           {({isSubmitting, isValid, dirty}) => (
               <Form className='ui form'>
                   <MyTextInput placeholder='Display Name'
name='displayName' />
                   <MyTextArea rows={3} placeholder='Add your
bio' name='bio' />
                   <Button
                       positive
                       type='submit'
                       loading={isSubmitting}
                       content='Update profile'
                       floated='right'
                       disabled={!isValid || !dirty}
                   />
               </Form>
           )}
       </Formik>
   )
})
```

7.2.2.4.7. ProfileFollowings

```
import { observer } from 'mobx-react-lite';
import React from 'react'
import { Card, Grid, Header, Tab } from 'semantic-ui-react';
import { useStore } from '../../app/stores/store'
import ProfileCard from './ProfileCard';

export default observer(function ProfileFollowings() {
   const {profileStore} = useStore();
   const {profile, followings, loadingFollowings, activeTab} =
profileStore;


   return (
       <Tab.Pane loading={loadingFollowings} >
           <Grid>
               <Grid.Column width={16}>
                   <Header
                       floated='left'
                       icon='user'
                       content={ activeTab === 3 ? `People
following ${profile?.displayName}` : `People
${profile?.displayName} is following`}
                   />
               </Grid.Column>
               <Grid.Column width={16}>
                   <Card.Group itemsPerRow={4}>
                       {followings.map(profile => (
                           <ProfileCard key={profile.username}
profile={profile} />
                       ))}
                   </Card.Group>
               </Grid.Column>
           </Grid>
       </Tab.Pane>
   )
})
```

7.2.2.4.8. ProfileHeader

```
import { observer } from 'mobx-react-lite'
import React from 'react'
import { Divider, Grid, Header, Item, Reveal, Segment,
Statistic } from 'semantic-ui-react'
import { Profile } from '../../app/models/profile'
import FollowButton from './FollowButton'

interface Props {
   profile: Profile;
}
export default observer(function ProfileHeader({profile}:
Props) {
   return (
       <Segment>
           <Grid>
               <Grid.Column width={12}>
                   <Item.Group>
```

```
                        <Item>
                            <Item.Image avatar size='small'
src={profile.image || '/assets/user.png'} />
                            <Item.Content
verticalAlign='middle'>
                                <Header as='h1'
content={profile.displayName} />
                            </Item.Content>
                        </Item>
                    </Item.Group>
                </Grid.Column>
                <Grid.Column width={4}>
                    <Statistic.Group widths={2}>
                        <Statistic label='Followers'
value={profile.followersCount} />
                        <Statistic label='Following'
value={profile.followingCount} />
                    </Statistic.Group>
                    <Divider />
                   <FollowButton profile={profile} />
                </Grid.Column>
            </Grid>
        </Segment>
    )
});
```

7.2.2.4.9. ProfilePage

```
import { observer } from 'mobx-react-lite'
import React, { useEffect } from 'react'
import { useParams } from 'react-router'
import { Grid } from 'semantic-ui-react'
import LoadingComponent from
'../../app/layout/LoadingComponent'
import { useStore } from '../../app/stores/store'
import ProfileContent from './ProfileContent'
import ProfileHeader from './ProfileHeader'

export default observer(function ProfilePage() {
    const {username} = useParams<{username: string}>();
    const {profileStore} = useStore();
    const {loadingProfile, loadProfile, profile, setActiveTab }
= profileStore;
```

```
   useEffect(() => {
       loadProfile(username);
       return () => {
           setActiveTab(0);
       }
   }, [loadProfile, username]);

   if(loadingProfile) return <LoadingComponent content='Loading
profile...' />

   return (
       <Grid>
           <Grid.Column width={16}>
               {profile &&
                   <>
                       <ProfileHeader profile={profile} />
                       <ProfileContent profile={profile} />
                   </>
               }
           </Grid.Column>
       </Grid>
   )
})
```

7.2.2.4.10. ProfilePhotos

```
import { observer } from 'mobx-react-lite'
import React, {SyntheticEvent, useState} from 'react'
import { Button, Card, Grid, Header, Image, Tab } from
'semantic-ui-react';
import PhotoUploadWidget from
'../../app/common/imageUpload/PhotoUploadWidget';
import { Photo, Profile } from '../../app/models/profile';
import { useStore } from '../../app/stores/store';

interface Props {
   profile: Profile;
}
export default observer(function ProfilePhotos({profile}:
Props) {
   const {profileStore: {isCurrentUser, uploadPhoto, uploading,
loading, setMainPhoto, deletePhoto}} = useStore();
```

```
   const [addPhotoMode, setAddPhotoMode] = useState(false);
   const [target, setTarget] = useState('');

   const handlePhotoUpload = (file: Blob) => {
       uploadPhoto(file).then(() => setAddPhotoMode(false));
   }

   const handleSetMainPhoto = (photo: Photo, event:
SyntheticEvent<HTMLButtonElement>) => {
       setTarget(event.currentTarget.name);
       setMainPhoto(photo);
   }

   const handleDeletePhoto = (photo: Photo, event:
SyntheticEvent<HTMLButtonElement>) => {
       setTarget(event.currentTarget.name);
       deletePhoto(photo);
   }
   return (
       <Tab.Pane>
           <Grid>
               <Grid.Column width={16}>
                   <Header icon='image' content='Photos' />
                   {isCurrentUser && (
                       <Button
                           floated='right'
                           basic
                           content={addPhotoMode ? 'Cancel' :
'Add Photo'}
                           onClick={() =>
setAddPhotoMode(!addPhotoMode)}
                       />
                   )}
               </Grid.Column>
               <Grid.Column width={16}>
                   {addPhotoMode ? (
                       <PhotoUploadWidget
uploadPhoto={handlePhotoUpload} loading={uploading}  />
                   ) : (
                       <Card.Group itemsPerRow={5}>
                           {profile.photos?.map(photo => (
                               <Card>
                                   <Image src={photo.url} />
                                   {isCurrentUser && (
                                       <Button.Group fluid
widths={2}>
                                           <Button
                                               basic
                                               color='green'
                                               content='Main'
                                               name={'main' +
photo.id}

disabled={photo.isMain}
                                               loading={target
=== 'main' + photo.id && loading}
                                               onClick={(event)
=> handleSetMainPhoto(photo, event)}
                                           />
                                           <Button
                                               basic
                                               color='red'
                                               icon='trash'
                                               loading={target
=== photo.id && loading}
                                               onClick={(event)
=> handleDeletePhoto(photo, event)}

disabled={photo.isMain}
                                               name={photo.id}
                                           />
                                       </Button.Group>
                                   )}
                               </Card>
                           ))}
                       </Card.Group>
                   )}
               </Grid.Column>
           </Grid>
       </Tab.Pane>
   )
});
```

7.2.2.5. users

7.2.2.5.1. LoginForm.tsx

```
import { ErrorMessage, Form, Formik } from 'formik'
import { values } from 'mobx'
import { observer } from 'mobx-react-lite'
import React from 'react'
import { Button, Header, Label } from 'semantic-ui-react'
import MyTextInput from '../../app/common/form/MyTextInput'
import { useStore } from '../../app/stores/store'

export default observer(function LoginForm() {
   const {userStore} = useStore();
   return (
       <Formik
        initialValues={{email: '', password: '', error: null}}
        onSubmit={(values, {setErrors}) =>
userStore.login(values).catch(error => setErrors({error:
'Invalid email or password'}))}
       >
           {({handleSubmit, isSubmitting, errors}) => (
               <Form className='ui form'
onSubmit={handleSubmit} autoComplete='off'>
                   <Header as='h2' content='Login to
CodeDancing Activities' color='teal' textAlign='center' />
                   <MyTextInput name='email'
placeholder='Email' />
                   <MyTextInput  name='password'
placeholder='Password' type='password' />
                   <ErrorMessage
                       name='error'
                       render={() => <Label
style={{marginBottom: 10}} basic color='red'
content={errors.error} />}
                   />
                   <Button loading={isSubmitting} positive
content='Login' type='submit' fluid />
               </Form>
           )}
       </Formik>
   )
})
```

7.2.2.5.2. RegisterForm.tsx

```
import { ErrorMessage, Form, Formik } from 'formik'
import { observer } from 'mobx-react-lite'
import React from 'react'
import { Button, Header } from 'semantic-ui-react'
import MyTextInput from '../../app/common/form/MyTextInput'
import { useStore } from '../../app/stores/store'
import * as Yup from 'yup';
import ValidationErrors from '../errors/ValidationErrors'

export default observer(function RegisterForm() {
   const {userStore} = useStore();
   return (
       <Formik
        initialValues={{displayName: '', username: '' , email:
'', password: '', error: null}}
        onSubmit={(values, {setErrors}) =>
userStore.register(values).catch(error => setErrors({error}))}
        validationSchema={Yup.object({
            displayName: Yup.string().required(),
            username: Yup.string().required(),
            email: Yup.string().required().email(),
            password: Yup.string().required(),

        })}
       >
           {({handleSubmit, isSubmitting, errors, isValid,
dirty}) => (
               <Form className='ui form error'
onSubmit={handleSubmit} autoComplete='off'>
                   <Header as='h2' content='Sign up to
CodeDancing Activities' color='teal' textAlign='center' />
                   <MyTextInput name='displayName'
placeholder='Display Name' />
                   <MyTextInput name='username'
placeholder='Username' />
                   <MyTextInput name='email'
placeholder='Email' />
                   <MyTextInput  name='password'
placeholder='Password' type='password' />
                   <ErrorMessage
                       name='error'
```

```
                        render={() => <ValidationErrors
errors={errors.error} />}
                    />
                    <Button
                        disabled={!isValid || !dirty ||
isSubmitting} loading={isSubmitting}
                        positive
                        content='Register'
                        type='submit'
                        fluid
                    />
                </Form>
            )}
        </Formik>
    )
})
```

7.2.3. intex.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
//import 'semantic-ui-css/semantic.min.css';
import 'react-calendar/dist/Calendar.css';
import 'react-toastify/dist/ReactToastify.min.css';
import 'react-datepicker/dist/react-datepicker.css';
import './app/layout/styles.css';
import App from './app/layout/App';
import reportWebVitals from './reportWebVitals';
import { store, StoreContext } from './app/stores/store';
import {BrowserRouter, Router } from 'react-router-dom';
import { createBrowserHistory } from 'history';
import ScrollToTop from './app/layout/ScrollToTop';

export const history = createBrowserHistory();

ReactDOM.render(
 <StoreContext.Provider value={store}>
   <Router history={history}>
     <React.StrictMode>
       <ScrollToTop />
       <App />
     </React.StrictMode>
   </Router>
 </StoreContext.Provider>,
 document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a
function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-
vitals
reportWebVitals();
```

7.2.4. package.json

```
{
 "name": "client-app",
 "version": "0.1.0",
 "private": true,
 "dependencies": {
   "@microsoft/signalr": "^6.0.3",
   "@testing-library/jest-dom": "^5.16.2",
   "@testing-library/react": "^12.1.2",
   "@testing-library/user-event": "^13.5.0",
   "@types/history": "^4.7.11",
   "@types/jest": "^27.4.0",
   "@types/node": "^16.11.22",
   "@types/react": "^17.0.39",
   "@types/react-calendar": "^3.5.0",
   "@types/react-dom": "^17.0.11",
   "@types/react-router-dom": "^5.3.2",
   "axios": "^0.25.0",
   "date-fns": "^2.28.0",
   "formik": "^2.2.9",
   "history": "^4.10.1",
   "mobx": "^6.3.13",
   "mobx-react-lite": "^3.2.3",
   "react": "^17.0.2",
   "react-calendar": "^3.7.0",
   "react-cropper": "^2.1.8",
   "react-datepicker": "^4.7.0",
   "react-dom": "^17.0.2",
   "react-dropzone": "^12.0.4",
   "react-infinite-scroller": "^1.2.5",
```

```json
    "react-router-dom": "^5.3.0",
    "react-scripts": "5.0.0",
    "react-toastify": "^8.2.0",
    "semantic-ui-css": "^2.4.1",
    "semantic-ui-react": "^2.1.1",
    "typescript": "^4.5.5",
    "uuid": "^8.3.2",
    "web-vitals": "^2.1.4",
    "yup": "^0.32.11"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "postbuild": "mv build ../API/wwwroot",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "@types/react-datepicker": "^4.3.4",
    "@types/react-infinite-scroller": "^1.2.3",
    "@types/uuid": "^8.3.4",
    "@types/yup": "^0.29.13"
  }
}
```

8. Dizajin Baze Podataka

Za kreiranje baze podataka u ovom projekto korišćen je „Code First“ pristup.

8.1. Activity.cs

```
using System;
using System.Collections.Generic;

// The Clean Archit. -> Entitites leyer
namespace Domain
{
   public class Activity
   {
       public Guid Id { get; set; }
       public string Title { get; set; }
       public DateTime Date { get; set; }
       public string Description { get; set; }
       public string Category { get; set; }
       public string City { get; set; }
       public string Venue { get; set; }
       public bool isCancelled { get; set; }
       public ICollection<ActivityAttendee> Attendees { get; set; } = new
List<ActivityAttendee>();
       public ICollection<Comment> Comments { get; set; } = new
List<Comment>();
   }
}
```

8.2. ActivityAttendee.cs

```
using System;

namespace Domain
{
   public class ActivityAttendee
   {
       public string AppUserId { get; set; }
       public AppUser AppUser { get; set; }
       public Guid ActivityId { get; set; }
       public Activity Activity { get; set; }
```

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

8.3. AppUser.cs

```
using System.Collections.Generic;
using Microsoft.AspNetCore.Identity;

namespace Domain
{
    public class AppUser : IdentityUser
    {
        public string DisplayName { get; set; }
        public string Bio { get; set; }
        public ICollection<ActivityAttendee> Activities { get; set; }
        public ICollection<Photo> Photos { get; set; }
        public ICollection<UserFollowing> Followings { get; set; }
        public ICollection<UserFollowing> Followers { get; set; }
    }
}
```

8.4. Comment.cs

```
using System;

namespace Domain
{
    public class Comment
    {
        public int Id { get; set; }
        public string Body { get; set; }
        public AppUser Author { get; set; }
        public Activity Activity { get; set; }
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    }
}
```

8.5. Photo.cs

```
namespace Domain
{
   public class Photo
   {
       public string Id { get; set; }
       public string Url { get; set; }
       public bool IsMain { get; set; }
   }
}
```

8.6. UserFollowing.cs

```
namespace Domain
{
    public class UserFollowing
    {
        public string ObserverId { get; set; }
        public AppUser Observer { get; set; }
        public string TargetId { get; set; }
        public AppUser Target { get; set; }
    }
}
```

9. Zaključak

Korišćenje razdvojenih celina za Front-End i Back-End aplikaciju se postiže veća jasnoća u programiranju i lakši razvoj aplikacija. Usklađivanje ove dve razdvojene celine uopšte nije teško pogotovo ako se koriste neki moderni alati za dokumentaciju, tako da više različitih ljudi može raditi na ovim odvojenim celinama neometano.
Raslojavanje aplikacije na više logičkih celina je izuzetno korisna stvar iz nekoliko razloga. Pre svega kod je značajno čitljiviji pa je samim tim lakše razumeti i održavati takav kod. Sa druge strane ukoliko postoji upotreba testova, onda se vrlo precizno mogu testirati različiti delovi aplikacije i greška se može lako uočiti. Korišćenjem "dependency injectiona" na vrlo jednostavan način možemo zameniti celine koje nas ne zadovoljavaju svojom performansom ili kvalitetom koda ili nekim drugim razlgom. Testovi bi nam u tom slučaju pomogli da ne izostavimo neke bitne funkcionalnosti prilikom te zamene.
Aplikacija koja ima razdvojene celine, ne samo na nivou koda nego i na arhitekturalnom nivou, jednostavnija je za održavanje, lakše podnosi izmene i promene i može se koristiti na više različitih načina. Konkretno u ovoj situaciji, Front-End je napravljen kao web stranica ali je isto tako mogao da bude i IOS ili Android aplikacija ili neka desktop aplikacija ili sve to zajedno. Poenta je da je celokupna

logika aplikacije centralizovana na jednom mestu i da može biti korišćena na različite načine.
Najbitniji deo ove aplikacije je napravljen kao izdvojena logička celina, stoga se može koristiti kao udaljeni saradnik za simulaciju određenih procesa.

10. Literatura

- **Carl-Hugo Marcotte**. “An Atypical ASP.NET Core 5 Design Patterns Guide: A SOLID adventure into architectural principles, design patterns, .NET 5, and C#”. Pocketbok 2020
- **Buschmann Frank, Meunier Regine, Rohnert Hans, Sommerlad Peter, Stal Michael**. “Pattern-Oriented Software Architecture, Volume 1, A System of Patterns”. Wiley (1996).
- **O’Reilly.** “Layered Architecture”. https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html
- **Modern ASP.NET documentation**, https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures
- **Modern ASP.NET documentation.** “CQRS paterns”**.** https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs
- **Code Maze**. “CQRS and MediatR in ASP.NET Core”. https://code-maze.com/cqrs-mediatr-in-aspnet-core/
- **Microsoft Documentation**. “ASP.NET Documentation”. https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0
- **React.JS.** “React.js Documentation”. https://reactjs.org/docs/getting-started.html
- **React Router.** “React Router Documentation” https://v5.reactrouter.com/web/guides/quick-start
- **TypeScript**. “TypeScript Documentation”. https://react-typescript-cheatsheet.netlify.app/docs/basic/setup
- **Semantic UI React.** “Semantic UI React Documentation”. https://react.semantic-ui.com/