Projekt BD2

System pośredniczący w udostępnianiu informacji turystycznej

Autorzy:
Katarzyna Majgier
Adam Hudziak
Dawid Musialik
Paweł Salicki
Adrian Skutela

Prowadząca:
dr inż. Bożena Małysiak-Mrozek

Rok akademicki:
2019/2020

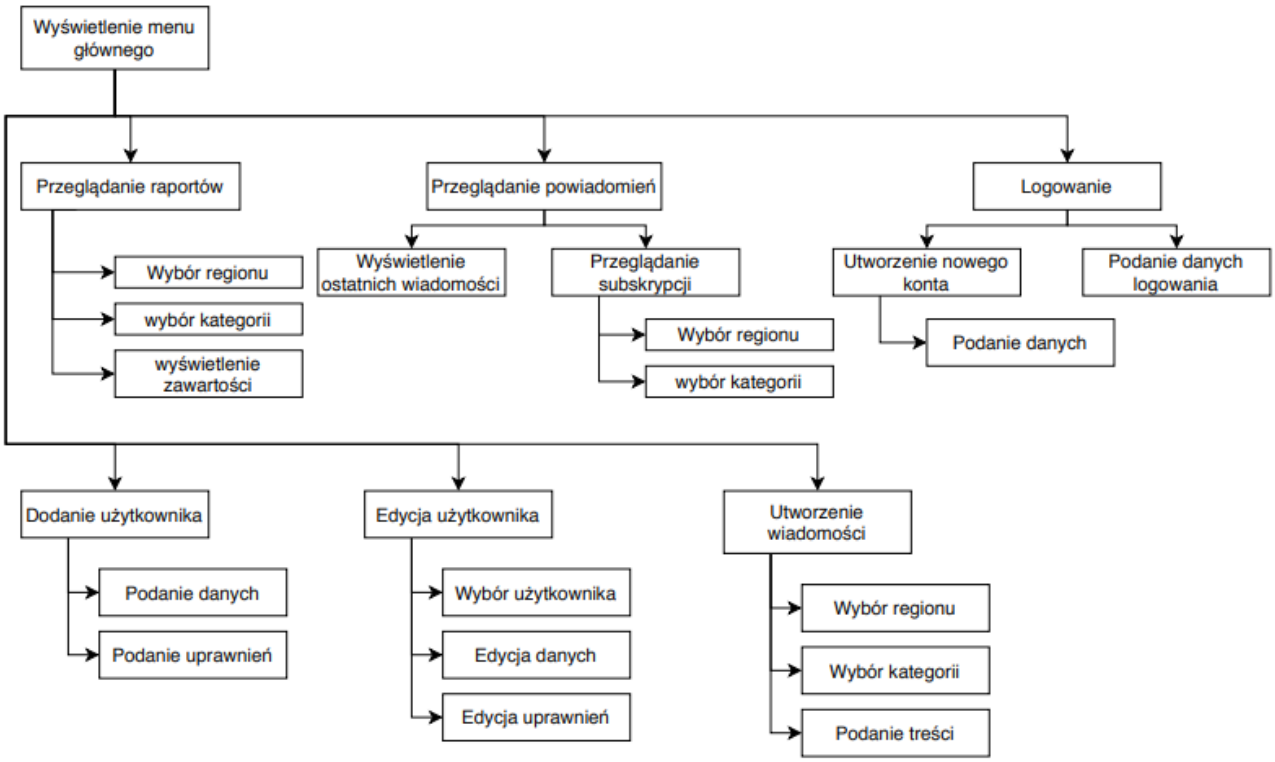
1. Opis projektu

Celem projektu było stworzenie systemu umożliwiającego udostępnianie, przeglądanie i modyfikowanie informacji o szlakach turystycznych, schroniskach, imprezach i innego rodzaju atrakcjach, jak np. muzea, restauracje itp. Jak również dodawanie i przeglądanie wiadomości dotyczących tych obiektów.

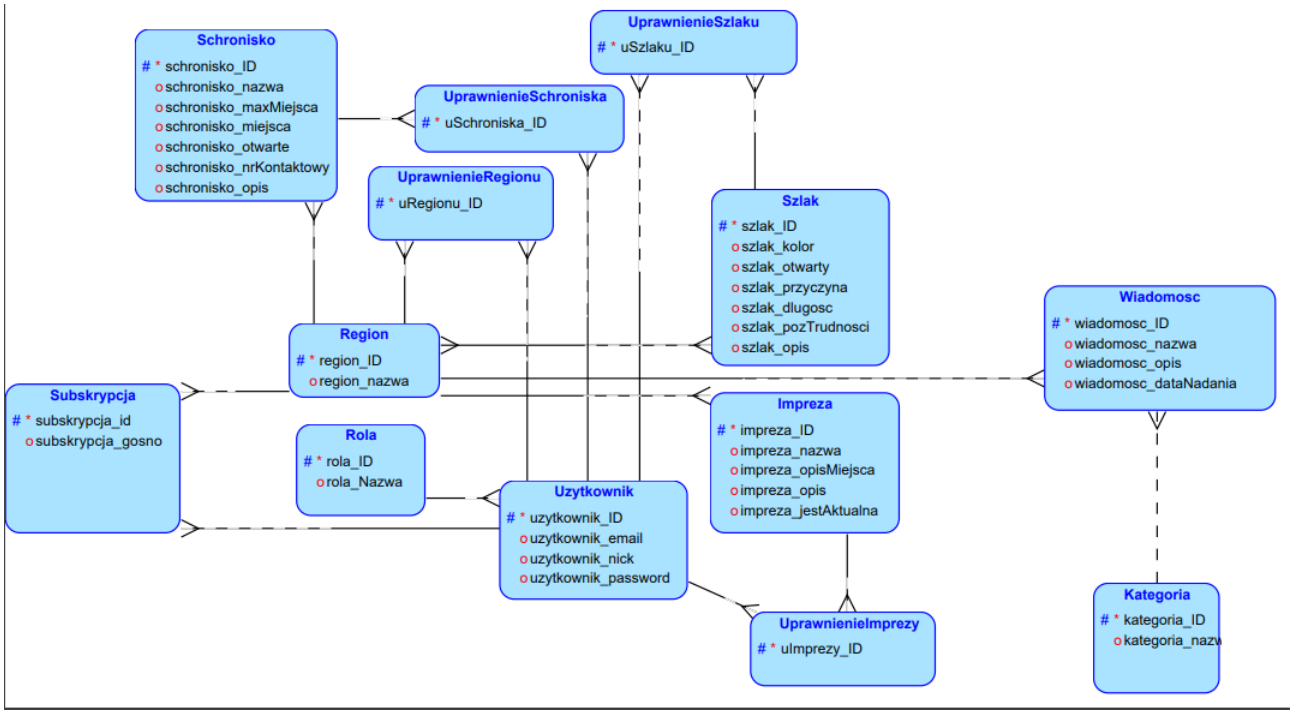
2. Założenia

1. Wykonanie systemu w postaci aplikacji webowej
2. Wykorzystanie języka C#, technologii ASP.NET oraz SQL Server
3. Podział użytkowników na
    - administratorów
    - moderatorów
    - użytkowników standardowych
    - użytkowników niezalogowanych
4. Użytkownik niezalogowany ma możliwość przeglądania dostępnych w systemie atrakcji, filtrowania ich pod kątem regionu i typu, wygenerowania raportu .pdf z tymi atrakcjami, jak również przeglądanie powiadomień dotyczących tych atrakcji. Ma dostęp do strony pomocy korzystania z systemu. Użytkownik niezalogowany ma także możliwość utworzyć konto w systemie i zalogować się.
5. Użytkownik zalogowany ma dostęp do wszystkich funkcjonalności użytkownika niezalogowanego oraz dodatkowo ma możliwość subskrypcji powiadomień z określonych regionów, co umożliwia szybkie znalezienie interesujących wiadomości.
6. Moderator ma dostęp do wszystkich funkcjonalności użytkownika zalogowanego oraz dodatkowo może dodawać powiadomienia dotyczące atrakcji znajdujących się w regionach, do których ma nadane uprawnienia. Może także przeglądać listę użytkowników, i nadawać im uprawnienia do regionów, do których sam ma uprawnienia
7. Moderator ma dostęp do wszystkich funkcjonalności moderatora oraz dodatkowo może dodawać wiadomości i uprawnienia do wszystkich regionów, posiada bezpośredni wgląd do bazy danych, oraz możliwość edycji tych danych. Może nadawać użytkownikom role moderatorów i administratorów.

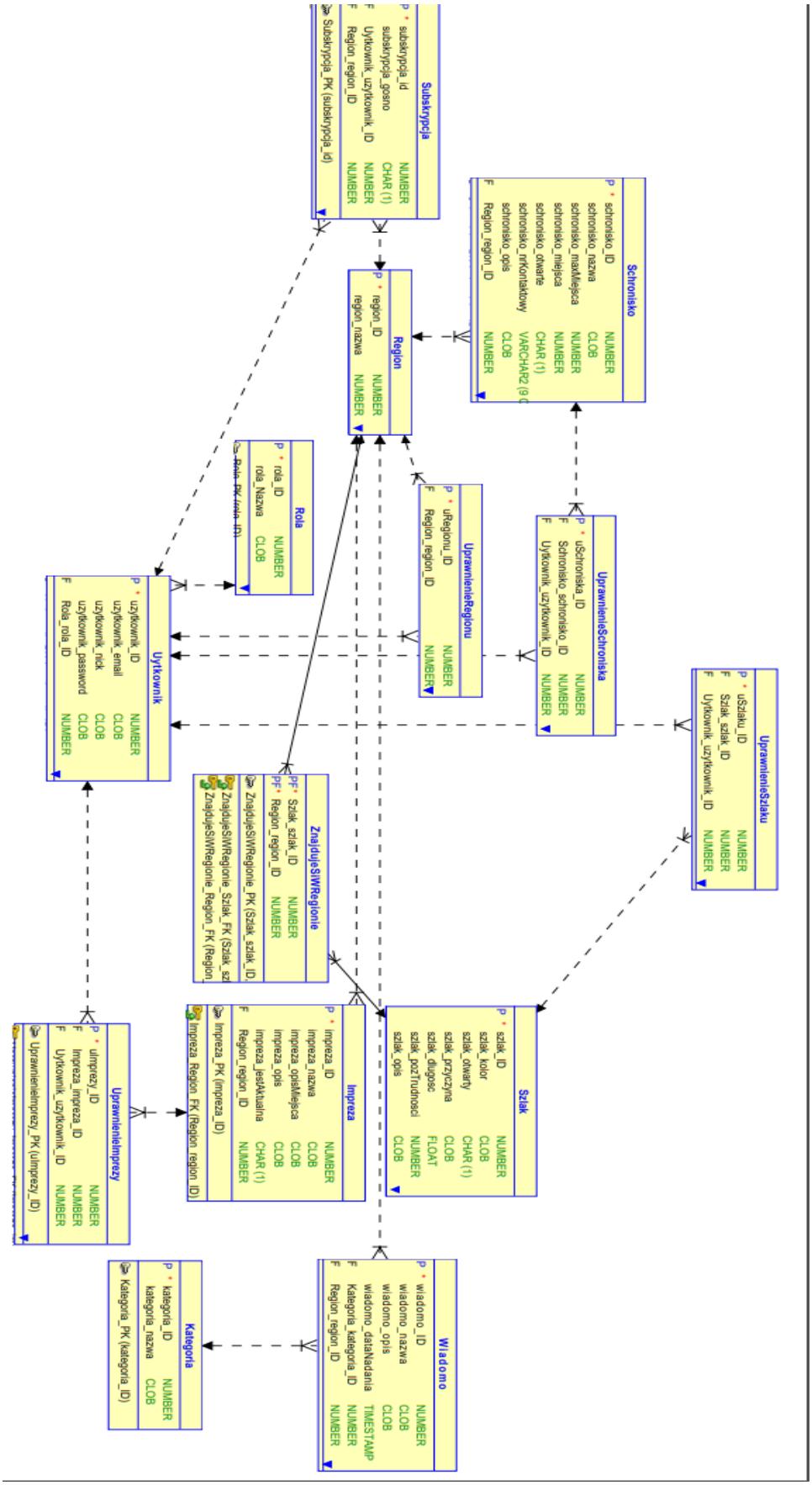
3. Diagram hierarchii funkcji



4. Model logiczny bazy danych



5. Model relacyjny bazy danych



6. Specyfikacja wewnętrzna

Program napisaliśmy w oparciu o architekturę MVC z dodatkową warstwą View Modeli i repozytoriów.

Klasy modeli reprezentują dane przechowywane w bazie danych.

Widoki, stworzone z wykorzystaniem C# (Razor), HTML, CSS i Javascript, odpowiadają za interfejs użytkownika.

Klasy View Modeli reprezentują dane, wyświetlane użytkownikowi w danym widoku. Niektóre widoki z nich nie korzystają, gdyż odpowiadają 1 do 1 modelowi.

Klasy kontrolerów odpowiadają za logikę aplikacji, oraz serwowanie widoków. Odczytują dane z modelu, przetwarzają je, mapują na View Model jeśli jest potrzeba i przesyłają do widoków, jak również w drugą stronę – przetwarzają dane z widoków i zapisują w bazie danych.

Klasy repozytoriów pośredniczą w komunikacji z bazą danych.

6.1 Opis klas

```
public class AppDbContext : IdentityDbContext<AppUser>
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
        {
        }

        public DbSet<Category> Categories { get; set; }
        public DbSet<Party> Partys { get; set; }
        public DbSet<Message> Messages { get; set; }
        public DbSet<PermissionParty> PermissionPartys { get; set; }
        public DbSet<PermissionRegion> PermissionRegions { get; set; }
        public DbSet<PermissionShelter> PermissionShelters { get; set; }
        public DbSet<PermissionTrail> PermissionTrails { get; set; }
        public DbSet<Region> Regions { get; set; }
        public DbSet<Shelter> Shelters { get; set; }
        public DbSet<Subscription> Subscriptions { get; set; }
        public DbSet<RegionLocation> RegionLocations { get; set; }
        public DbSet<Trail> Trails { get; set; }
        public DbSet<Attraction> Attractions { get; set; }
        public override DbSet<AppUser> Users { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
    }
```

Klasa reprezentująca bazę danych. Służy do wykonywania zapytań LINQ.

```
public class Attraction
    {
        [Key]
        [Display(Name = "Klucz atrakcje")]
        public int IdAttraction { get; set; } // K

        [Display(Name = "Typ atrakcji")]
        public string AttractionType { get; set; }

        [Display(Name = "Nazwa atrakcji")]
        public string Name { get; set; }

        [Display(Name = "Opis atrakcji")]
        public string Description { get; set; }

        [Display(Name = "Klucz obcy region")]
```

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

        [Display(Name = "Region")]
        public virtual Region Region { get; set; }
    }
```

Klasa reprezentująca atrakcję turystyczną. Posiada pola przechowujące typ atrakcji, jej nazwę, opis oraz region w którym się znajduje.

```
public class Party
    {
        [Key]
        [Display(Name = "Klucz impreza")]
        public int IdParty { get; set; } // K

        [Display(Name = "Nazwa imprezy")]
        public string Name { get; set; }

        [Display(Name = "Lokalizacja")]
        public string PlaceDescription { get; set; }

        [Display(Name = "Opis")]
        public string Description { get; set; }

        [Display(Name = "Aktualna")]
        public bool UpToDate { get; set; }

        [Display(Name = "Klucz obcy region")]
        public int IdRegion { get; set; }

        [Display(Name = "Region")]
        public virtual Region Region { get; set; }

        public virtual ICollection<PermissionParty> PermissionParty { get; set; }
    }
```

Klasa reprezentująca imprezę. Przechowuje nazwę, region, opis, lokalizację, jak również listę użytkowników, którzy mają uprawnienia do modyfikacji tej imprezy. Lista uprawnień nie jest jednak wykorzystywana, gdyż zdecydowaliśmy się zaimplementować uprawnienia tylko dla regionów.

```
public class Shelter
    {
        [Key]
        [Display(Name = "Klucz schronisko")]
        public int IdShelter { get; set; }

        [Display(Name = "Nazwa schroniska")]
        public string Name { get; set; }

        [Display(Name = "Max il. miejsc")]
        public int MaxPlaces { get; set; }

        [Display(Name = "Miejsca")]
        public int Places { get; set; }

        [Display(Name = "Otwarte")]
        public bool IsOpen { get; set; }

        [Display(Name = "Numer telefonu")]
        public string PhoneNumber { get; set; }

        [Display(Name = "Opis")]
```

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

        [Display(Name = "Klucz obcy region")]
        public int IdRegion { get; set; }

        [Display(Name = "Region")]
        public virtual Region Region { get; set; }

        public virtual ICollection<PermissionShelter> PermissionShelters { get; set; }
    }
```

Klasa reprezentująca schronisko. Przechowuje nazwę liczbę miejsc, informację o otwarciu, numer telefonu, opis, oraz region, w którym się znajduje.

```
    public class Category
    {
        [Key]
        [Display(Name = "Klucz kategoria")]
        public int IdCategory { get; set; } // K

        [Display(Name = "Nazwa kategorii")]
        public string Name { get; set; }

        public virtual ICollection<Message> Messages { get; set; }
    }
```

Klasa reprezentująca kategorię wiadomości. Posiada nazwę kategorii, oraz referencje na wszystkie wiadomości, które do tej kategorii należą.

```
public class Message
    {
        [Key]
        [Display(Name = "Klucz wiadomość")]
        public int IdMessage { get; set; } // K

        [Display(Name = "Nazwa")]
        public string Name { get; set; }

        [Display(Name = "Treść")]
        public string Description { get; set; }

        [Display(Name = "Data zapostowania")]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime PostingDate1 { get; set; }

        [Display(Name = "Klucz obcy kategoria")]
        public int? IdCategory { get; set; }

        [Display(Name = "Kategoria")]
        public virtual Category Category { get; set; }

        [Display(Name = "Klucz obcy region")]
        public int? IdRegion { get; set; }

        [Display(Name = "Region")]
        public virtual Region Region { get; set; }
    }
```

Klasa reprezentująca powiadomienie przesyłane w naszym systemie. Przechowuje tytuł wiadomości, jej treść, datę nadania, kategorię do której należy, oraz region, którego dotyczy.

```
   public class Region
    {
        [Key]
        [Display(Name = "Klucz region")]
        public int IdRegion { get; set; }
```

```
        [Display(Name = "Nazwa regionu")]
        public string Name { get; set; }

        public virtual ICollection<Message> Message { get; set; }
        public virtual ICollection<Party> Party { get; set; }
        public virtual ICollection<RegionLocation> RegionLocation { get; set; }
        public virtual ICollection<Shelter> Shelter { get; set; }
        public virtual ICollection<PermissionRegion> PermissionRegion {get; set;}
        public virtual ICollection<Subscription> Subscription { get; set; }
        public virtual ICollection<Attraction> Attraction { get; set; }
    }
```

Klasa reprezentująca region. Przechowuje nazwę regionu, jak również referencję do wszystkich wiadomości, imprez szlaków itp. Które dotyczą danego regionu, jak również referencję na użytkowników, którzy do danego regionu są zasubskrybowani, lub mają uprawnienia do dodawania w nim wiadomości.

```
    public class RegionLocation
    {
        public int? IdTrail { get; set; }
        public Trail Trail { get; set; }

        public int? IdRegion { get; set; }
        public Region Region { get; set; }
    }
```

Klasa reprezentująca relację wiele do wielu szlaków z regionami – jeden szlak może przechodzić przez wiele regionów, a przez jeden region może przechodzić wiele szlaków.

```
   public class PermissionRegion
    {
        public int? IdRegion { get; set; }
        public Region Region { get; set; }

        public string IdUser { get; set; }
        public AppUser User { get; set; }
    }
```

Klasa reprezentująca uprawnienie nadane danemu użytkownikowi do dodawania powiadomień odnośnie atrakcji w danym regionie.

```
    public class Subscription
    {
        [Key]
        [Display(Name = "Klucz subskrypcja")]
        public int IdSubscription { get; set; }

        [Display(Name = "Zasubskrybowano")]
        public bool IsSubscribed { get; set; }

        [Display(Name = "Klucz obcy region")]
        public int IdRegion { get; set; }

        [Display(Name = "Region")]
        public virtual Region Region { get; set; }

        [Display(Name = "Klucz obcy użytkownik")]
        public string IdUser { get; set; }

        [Display(Name = "Użytkownik")]
        public virtual AppUser User { get; set; }
    }
```

Klasa reprezentująca subskrypcję danego użytkownika do powiadomień z danego regionu.

```
namespace InformacjeTurystyczne.Models
{
    public class AppUser : IdentityUser
```

```
    {
        public virtual ICollection<Subscription> Subscriptions { get; set; }
        public virtual ICollection<PermissionParty> PermissionPartys { get; set; }
        public virtual ICollection<PermissionRegion> PermissionRegions { get; set; }
        public virtual ICollection<PermissionShelter> PermissionShelters { get; set; }
        public virtual ICollection<PermissionTrail> PermissionTrails { get; set; }
    }
}
```

Klasa reprezentująca użytkownika aplikacji. Zawiera informacje o subskrypcjach i uprawnieniach użytkownika.

```
public class EmailSender : IEmailSender
    {
        private readonly EmailService _emailConfiguration;

        public EmailSender(EmailService emailConfiguration)
        {
            _emailConfiguration = emailConfiguration;
        }

        public void SendEmail(Message message)
        public async Task SendEmailAsync(Message message)

        private MimeMessage CreateEmailMessage(Message message)

        private void Send(MimeMessage mailMessage)

        private async Task SendAsync(MimeMessage mailMessage)
    }
```

Klasa służąca do tworzenia i wysyłania wiadomości email. Pozawala na wysłanie maila zarówno asynchronicznie jak i "blokująco".

```
public class Message
    {
        public List<MailboxAddress> To { get; set; }
        public string Subject { get; set; }
        public string Content { get; set; }

        public Message(IEnumerable<string> to, string subject, string content)
        {
            To = new List<MailboxAddress>();

            To.AddRange(to.Select(x => new MailboxAddress(x)));
            Subject = subject;
            Content = content;
        }
    }
```

Klasa reprezentująca tekstową wiadomość email. Zawiera adresy nadawców, temat i zawartość.

```
public class EmailService
    {
        public string From { get; set; }
        public string SmtpServer { get; set; }
        public int Port { get; set; }
        public string UserName { get; set; }
        public string Password { get; set; }
    }
```

Klasa reprezentująca usługę wysyłania emaili. Zawiera dane logowania do serwera SMTP.

```
public class AttractionRepository : IAttractionRepository
    {
        public readonly AppDbContext _appDbContext;

        public AttractionRepository(AppDbContext appDbContext)
        {
            _appDbContext = appDbContext;
        }

        public async Task<IEnumerable<Attraction>> GetAllAttraction()
        public async Task<Attraction> GetAttractionByID(int? attractionID)
        public async Task<Attraction> GetAttractionByIDWithoutInclude(int? attractionID)
        public async Task<Attraction> GetAttractionByIDWithoutIncludeAndAsNoTracking(int?
attractionID)
        public async Task AddAttractionAsync(Attraction attraction)
        public async Task DeleteAttractionAsync(Attraction attraction)
        public void EditAttraction(Attraction attraction)
        public async Task SaveChangesAsync()
        public IEnumerable<Attraction> GetAllAttractionAsNoTracking()
        public IEnumerable<Region> GetAllRegionAsNoTracking()
        public IEnumerable<Attraction> GetAllAttractionToUser()
    }
```

Klasy repozytoriów umożliwiają łatwy dostęp, modyfikację i zapis danych w bazie danych.
Metoda GetAllAttraction zwraca listę wszystkich atrakcji.
Metoda GetAttractionById zwraca atrakcję o określonym ID
Metoda GetAttractionByIDWithoutInclude zwraca atrakcję o określonym ID, ale bez wykonywania zapytań do powiązanych tabel.
Metoda GetAttractionByIDWithoutIncludeAndAsNoTracking zwraca atrakcję o określonym ID, ale bez wykonywania zapytań do powiązanych tabel. Ponadto zmiany wprowadzone w zwróconej atrakcji nie są śledzone, czyli nie zostaną zapisane w bazie danych.
Metoda AddAttractionAsync dodaje nową atrakcję do bazy danych.
Metoda DeleteAttractionAsync usuwa daną atrakcję z bazy danych.
Metoda EditAttraction modyfikuje daną atrakcję w bazie danych.
Metoda SaveChangesAsync zapisuje zmiany wprowadzone do śledzonych obiektów atrakcji w bazie danych.
Metoda GetAllAttractionAsNoTracking zwraca listę wszystkich atrakcji, ale zmiany wprowadzone w zwróconych atrakcjach nie są śledzone, więc nie zostaną zapisane w bazie danych.
Metoda GetAllAttractionToUser zwraca listę wszystkich atrakcji, ale blokuje do czasu zakończenia zapytania.

Repozytoria dla pozostałych danych zaimplementowane są podobnie. Poniżej znajduje się lista pozostałych klas repozytoriów:

- CategoryRepository
- MessageRepository
- PartyRepository
- RegionLocationRepository
- RegionRepository
- ShelterRepository
- SubscriptionRepository
- TrailRepository

W projekcie znajdują się następujące ViewModele.

```
public class HomepageVM
    {
        private Random rnd = new Random();
        public int RandomInt(int max)
        {
            return rnd.Next(max);
        }
    }
```

Prosta klasa, która dostarcza widokowi losową liczbę pozwalającą wybrać filmik w tle na stronie głównej.

```
public class LoginVM
    {
            [Display(Name = "Nazwa użytkownika")]
            public string UserName { get; set; }

            [Display(Name = "Hasło")]
            [DataType(DataType.Password)]
            public string Password { get; set; }

            public string ReturnUrl { get; set; }

            public IList<AuthenticationScheme> ExternalLogins { get; set; }
    }
```

ViewModel reprezentujący użytkownika podczas procesu logowania. Zawiera jedynie potrzebne dane o użytkowniku, tj. nazwę i hasło. Ponadto zawiera URL na, który ma nastąpić przekierowanie po zalogowaniu się.

```
public class NotificationsVM
    {
        public IMessageRepository messages { get; set; }
        public List<Region> regions { get; set; }

        public AppUser user { get; set; }
    }
```

ViewModel grupujący dane wykorzystywane przez widok powiadomień. Zawiera listę wszystkich wiadomości, oraz listę wszystkich regionów, jak również aktualnie zalogowanego użytkownika.

```
public class PermissionPartyData
    {
        public int IdParty { get; set; }
        public string Name { get; set; }
        public bool Assigned { get; set; }
    }
```

ViewModel reprezentujący uprawnienie nadane użytkownikowi do danej imprezy. Aktualnie uprawnienia są zaimplementowane tylko dla regionów.

```
public class PermissionRegionData
    {
        public int IdRegion { get; set; }
        public string Name { get; set; }
        public bool Assigned { get; set; }
    }
```

ViewModel reprezentujący uprawnienie nadane użytkownikowi do danego regionu.

```
   public class PermissionShelterData
    {
        public int IdShelter { get; set; }
        public string Name { get; set; }
        public bool Assigned { get; set; }
    }
```

ViewModel reprezentujący uprawnienie nadane użytkownikowi do danego schroniska. Aktualnie uprawnienia są zaimplementowane tylko dla regionów.

```
 public class PermissionTrailData
    {
        public int IdTrail { get; set; }
        public string Name { get; set; }
        public bool Assigned { get; set; }
    }
```

ViewModel reprezentujący uprawnienie nadane użytkownikowi do danego szlaku. Aktualnie uprawnienia są zaimplementowane tylko dla regionów.

```
    public class RegisterVM
    {
        [Required]
        [Display(Name="Nazwa użytkownika")]
        public string UserName { get; set; }

        [Required]
        [RegularExpression("^[a-zA-Z0-9_\\.-]+@([a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,6}$",
ErrorMessage = "E-mail nie jest poprawny")]
        public string Email { get; set; }

        [Required]
        [Display(Name="Hasło")]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
```

ViewModel reprezentujący użytkownika podczas procesu rejestracji. Zawiera nazwę użytkownika, email i hasło.

```
    public class RoleUsersVM
    {
        public IdentityRole Role { get; set; }

        public IEnumerable<AppUser> Members { get; set; }

        public IEnumerable<AppUser> NonMembers { get; set; }
    }
```

ViewModel reprezentujący użytkowników danej roli. Zawiera informacje o roli, oraz listę użytkowników, którym ta rola została nadana, jak również listę użytkowników, którzy danej roli nie posiadają.

```
    public class RoleVM
    {
        public string Id { get; set; }

        public string RoleName { get; set; }

        public string[] AddId { get; set; }

        public string[] DeleteId { get; set; }
    }
```

ViewModel reprezentujący rolę, służy do przekazywania danych z widoku do kontrolera. Zawiera id użytkowników, którzy mają byś usunięci lub dodani do roli.

```
    public class TouristInformationVM
    {
        public IAttractionRepository attractions { get; set; }
        public IPartyRepository parties { get; set; }
        public IShelterRepository shelters { get; set; }
        public ITrailRepository trails { get; set; }
        public IRegionRepository regions { get; set; }
    }
```

ViewModel grupujący dane potrzebne do zaprezentowania użytkownikowi wszystkich obiektów w systemie.

```
public class UserIndexData
{
    public IEnumerable<AppUser> Users { get; set; }
    public IEnumerable<Region> Regions { get; set; }
    public IEnumerable<Shelter> Shelters { get; set; }
    public IEnumerable<Trail> Trails { get; set; }
    public IEnumerable<Party> Partys { get; set; }
}
```

ViewModel zawierający listy regionów, schronisk, szlaków i atrakcji, do których użytkownik ma uprawnienia. W systemie wykorzystywane są tylko uprawnienia regionów.

```
public class UsersVM
{
    public struct User
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Email { get; set; }
        public string Role { get; set; }
        public List<string> RegionsPermissions { get; set; }
        public List<string> ShelterPermissions { get; set; }

        public List<string> PartysPermissions { get; set; }
        public List<string> TrailsPermissions { get; set; }
    }
    public List<User> AllUsers { get; set; }
    public User CurrentUser { get; set; }
    public List<string> AllawedRoles { get; set; }
}
```

ViewModel przechowujący dane o użytkowniku. Dodatkowo zawiera także referencję na aktualnie zalogowanego użytkownika, jak również listę wszystkich użytkowników.

```
public class AccountController : Controller
    {
        private readonly SignInManager<AppUser> _signInManager;
        private readonly UserManager<AppUser> _userManager;
        private readonly ILogger<AccountController> _iLogger;
        private readonly IEmailSender _emailSender;


        public AccountController(SignInManager<AppUser> signInManager,
            UserManager<AppUser> userManager, ILogger<AccountController> iLogger,
            IEmailSender emailSender)

        [HttpGet]
        [AllowAnonymous]
        public async Task<IActionResult> Login(string returnUrl)

        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Login(LoginVM loginVM)

        [AllowAnonymous]
        public IActionResult Register()

        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Register(RegisterVM loginVM)
```

```
        [HttpGet]
        [AllowAnonymous]
        public async Task<IActionResult> ConfirmEmail(string token, string email)

        [AllowAnonymous]
        [HttpPost]
        public IActionResult ExternalLogin(string provider, string returnUrl)

        [AllowAnonymous]
        public async Task<IActionResult> ExternalLoginCallback(string returnUrl = null,
string remoteUrl = null)

        [HttpGet]
        [AllowAnonymous]
        public IActionResult SuccessRegistration()
    }
```

AccountController odpowiada za proces logowania i rejestracji użytkownika. Dostarcza odpowiednie widoki, uwierzytelnia użytkownika, przy rejestracji weryfikuje adres email.

```
public class AdminController : Controller
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly IPasswordHasher<AppUser> _passwordHasher;

        public AdminController(UserManager<AppUser> userManager, IPasswordHasher<AppUser>
passwordHasher)

        public IActionResult Index()

        [HttpGet]
        public async Task<IActionResult> UpdateUser(string id)

        [HttpPost]
        public async Task<IActionResult> UpdateUser(string id, string email, string
password, bool emailConfirmed)

        private void ErrorResult(IdentityResult result)

        public ViewResult Create() => View();

        [HttpPost]
        public async Task<IActionResult> DeleteUser(string id)

        [HttpPost]
        public async Task<IActionResult> Create(RegisterVM user)

        [HttpGet]
        [AllowAnonymous]
        public IActionResult AccessDenied()
    }
```

AdminController odpowiada za zarządzanie użytkownikami w panelu admina. Umożliwia usuwanie, dodawanie i edycję użytkowników.

```
public class AttractionController : Controller
    {
        private readonly IAttractionRepository _attractionRepository;

        public AttractionController(IAttractionRepository attractionRepository)

        public async Task<IActionResult> Index()
```

```
        public async Task<IActionResult> Details(int? id)

        public IActionResult Create()

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("IdAttraction, AttractionType, Name,
Description, IdRegion")] Attraction attraction)

        public async Task<IActionResult> Edit(int? id)

        [HttpPost, ActionName("Edit")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> EditPost(int? id)

        public void PopulateAttraction(object selectedRegion = null)

        public async Task<IActionResult> Delete(int? id)

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
    }
```

AttractionController odpowiada za zarządzanie atrakcjami w panelu admina. Umożliwia usuwanie, dodawanie i edycję atrakcji.

```
public class CategoryController : Controller
    {
        private readonly ICategoryRepository _categoryRepository;

        private readonly IMessageRepository _messageRepository;

        public CategoryController(ICategoryRepository categoryRepository, IMessageRepository
messageRepository)

        public async Task<IActionResult> Index()

        public async Task<IActionResult> Details(int? id)

        public IActionResult Create()

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("IdCategory,Name,PlaceDescription,Description,UpToDate,IdRegion")] Category
category)

        public async Task<IActionResult> Edit(int? id)

        [HttpPost, ActionName("Edit")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> EditPost(int? id)

        public async Task<IActionResult> Delete(int? id)

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
    }
```

CategoryController odpowiada za zarządzanie kategoriami wiadomości w panelu admina. Umożliwia usuwanie, dodawanie i edycję kategorii.

```
public c
```

```
lass ErrorController : Controller
    {
        [Route("Error/{statusCode}")]
        public IActionResult HttpStatusCodeHandler(int statusCode)
    }
```
AdminController odpowiada za wyświetlenie odpowiedniej strony błędu (404, etc.).

```
 public class HelpController : Controller
    {
        public IActionResult Index()
    }
```
Jedyną rolą klasy HelpController jest wyświetlenie strony pomocy.

```
public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)

        public IActionResult Index()

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
    }
```
Home Controller odpowiada za wyświetlenie strony głównej naszego systemu.

```
public class MailController : Controller
    {
        private readonly AppDbContext _appDbContext;
        private readonly IMessageRepository _messageRepository;

        public MailController(IMessageRepository messageRepository, AppDbContext
appDbContext)

        public async Task<IActionResult> Index()

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Index([Bind("IdMessage,Name,Description,PostingDate1,IdCategory,IdRegion")] Message message)

        public void PopulateMessage(AppUser user = null, object selectedRegion = null,
object selectedCategory = null)
    }
```
MailController odpowiada za dodawanie wiadomości przez moderatorów, z uwzględnieniem uprawnień poszczególnych użytkowników.

```
public class MessageController : Controller
    {
        private readonly IMessageRepository _messageRepository;

        public MessageController(IMessageRepository messageRepository)

        public async Task<IActionResult> Index()

        public async Task<IActionResult> Details(int? id)

        public IActionResult Create()
```

```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("IdMessage,Name,Description,PostingDate1,IdCategory,IdRegion")] Message
message)

        public async Task<IActionResult> Edit(int? id)

        [HttpPost, ActionName("Edit")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> EditPost(int? id)

        public void PopulateMessage(object selectedRegion = null, object selectedCategory =
null)

        public async Task<IActionResult> Delete(int? id)

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
    }
```

MessageController odpowiada za zarządzanie wiadomościami w panelu admina. Umożliwia usuwanie, dodawanie i edycję wiadomości.

```
public class NotificationsController : Controller
    {
        private readonly IMessageRepository _messageRepository;
        private readonly IRegionRepository _regionRepository;
        private readonly AppDbContext _appDbContext;

        public NotificationsController(IMessageRepository messageRepository,
IRegionRepository regionRepository, AppDbContext appDbContext)

        public async Task<IActionResult> Index()

        public class RegionData
        {
            public string RegionName { get; set; }
        }

        [HttpPost]
        public async Task<EmptyResult> AddSubscription([FromBody] RegionData data)

        [HttpPost]
        public async Task<EmptyResult> RemoveSubscription([FromBody] RegionData data)
    }
```

NotificationsController odpowiada za wyświetlenie listy wiadomości, jak również za dodawanie lub usuwanie subskrypcji, przez danego użytkownika. Możliwe jest filtrowanie wyświetlanej listy, jednak dzieje się ono po stronie klienta.

```
public class PartyController : Controller
    {
        private readonly IPartyRepository _partyRepository;

        public PartyController(IPartyRepository partyRepository)

        public async Task<IActionResult> Index()

        public async Task<IActionResult> Details(int? id)
```

```
        public IActionResult Create()

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("IdParty,Name,PlaceDescription,Description,UpToDate,IdRegion")] Party party)

        public async Task<IActionResult> Edit(int? id)

        [HttpPost, ActionName("Edit")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> EditPost(int? id)

        public void PopulateParty(object selectedRegion = null)

        public async Task<IActionResult> Delete(int? id)

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
    }
```

PartyController odpowiada za zarządzanie imprezami w panelu admina. Umożliwia usuwanie, dodawanie i edycję imprez.

```
public class RegionController : Controller
    {
        private readonly IRegionRepository _regionRepository;

        public RegionController(IRegionRepository regionRepository)

        public async Task<IActionResult> Index(int? id, int? IdParty)

        public async Task<IActionResult> Details(int? id)

        public IActionResult Create()

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("IdRegion,Name")] Region region,
string[] selectedTrails)

        public async Task<IActionResult> Edit(int? id)

        [HttpPost, ActionName("Edit")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> EditPost(int? id, string[] selectedTrails)

        public async Task<IActionResult> Delete(int? id)

        private void PopulateRegion(Region regionToUpdate)

        private void UpdateRegion(string[] selectedTrails, Region regionToUpdate)

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
    }
```

RegionController odpowiada za zarządzanie regionami w panelu admina. Umożliwia usuwanie, dodawanie i edycję regionów.

```
public class RoleController : Controller
    {
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly UserManager<AppUser> _userManager;

        public RoleController(RoleManager<IdentityRole> roleManager, UserManager<AppUser>
userManager)

        public async Task<ViewResult> Index()

        [HttpGet]
        public IActionResult CreateRole()

        [HttpPost]
        public async Task<IActionResult> CreateRole(RoleVM roleVM)

        [HttpPost]
        public async Task<IActionResult> DeleteRole(RoleVM roleVM)

        public async Task<IActionResult> UpdateRole(string id)

        [HttpPost]
        public async Task<IActionResult> UpdateRole(RoleVM roleVM)
    }
```

RoleController odpowiada za zarządzanie rolami w panelu admina. Umożliwia usuwanie, dodawanie i edycję ról.

```
[System.Runtime.InteropServices.Guid("28B2BD28-2FAF-47D2-A4DD-AB28DADED1F0")]
    public class ShelterController : Controller
    {
        private readonly IShelterRepository _shelterRepository;

        public ShelterController(IShelterRepository shelterRepository)

        public async Task<IActionResult> Index()

        public async Task<IActionResult> Details(int? id)

        public IActionResult Create()

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("IdShelter, Name, MaxPlaces, Places,
IsOpen, PhoneNumber, Description, IdRegion")] Shelter shelter)

        public async Task<IActionResult> Edit(int? id)

        [HttpPost, ActionName("Edit")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> EditPost(int? id)

        public void PopulateShelter(object selectedRegion = null)

        public async Task<IActionResult> Delete(int? id)

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
    }
```

ShelterController odpowiada za zarządzanie schroniskami w panelu admina. Umożliwia usuwanie, dodawanie i edycję schronisk.

```
public class SubscriptionController : Controller
    {
        private readonly ISubscriptionRepository _subscriptionRepository;

        public SubscriptionController(ISubscriptionRepository subscriptionRepository)

        public async Task<IActionResult> Index()

        public async Task<IActionResult> Details(int? id)

        public IActionResult Create()

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("IdSubscription, IsSubscribed, IdUser,
IdRegion")] Subscription subscription)

        public async Task<IActionResult> Edit(int? id)

        [HttpPost, ActionName("Edit")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> EditPost(int? id)

        public void PopulateSubscription(object selectedRegion = null, object selectedUser =
null)

        public async Task<IActionResult> Delete(int? id)

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
    }
```

SubscriptionController odpowiada za zarządzanie subskrypcjami poszczególnych użytkowników w panelu admina. Umożliwia usuwanie, dodawanie i edycję subskrypcji użytkowników systemu.

```
public class TouristInformationController : Controller
    {
        private readonly IConverter _converter;


        private readonly IAttractionRepository _attractionRepository;
        private readonly IPartyRepository _partyRepository;
        private readonly IShelterRepository _shelterRepository;
        private readonly ITrailRepository _trailRepository;
        private readonly IRegionRepository _regionRepository;


        public TouristInformationController(
            IAttractionRepository attractionRepository,
            IPartyRepository partyRepository,
            IShelterRepository shelterRepository,
            ITrailRepository trailRepository,
            IRegionRepository regionRepository,
            IConverter converter
            )

        public IActionResult Index()

        [HttpGet]
        public IActionResult GeneratePDF()

        private string PrepareDocument()
    }
```

TouristInformationController odpowiada za wyświetlenie użytkownikowi listy wszystkich obiektów dodanych do naszego systemu, jak również za wygenerowanie raportu .pdf z tymi obiektami na życzenie użytkownika. Możliwe jest filtrowanie listy obiektów, jednak dzieje się ono po stronie klienta.

```
public class TrailController : Controller
    {
        private readonly ITrailRepository _trailRepository;

        public TrailController(ITrailRepository trailRepository)

        public async Task<IActionResult> Index()

        public async Task<IActionResult> Details(int? id)

        public IActionResult Create()

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("IdTrail, Name, Colour, Open,
Feedback, Length, Difficulty, Description")] Trail trail)

        public async Task<IActionResult> Edit(int? id)

        [HttpPost, ActionName("Edit")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> EditPost(int? id)

        public async Task<IActionResult> Delete(int? id)

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
    }
```

TrailController odpowiada za zarządzanie szlakami w panelu admina. Umożliwia usuwanie, dodawanie i edycję szlaków.

```
public class UserController : Controller
    {
        private readonly AppDbContext _appDbContext;
        private readonly IRegionRepository _regionRepository;

        public UserController(AppDbContext appDbContext, IRegionRepository regionRepository)

        public async Task<IActionResult> Index()

        public class RegionData

        [HttpPost]
        public async Task<EmptyResult> AddRegionPermission([FromBody] RegionData data)

        [HttpPost]
        public async Task<EmptyResult> RemoveRegionPermission([FromBody] RegionData data)
    }
```

UserController odpowiada za zarządzanie użytkownikami przez moderatorów. Umożliwia im dodawanie uprawnień poszczególnym użytkownikom. Nadane uprawnienia nie mogą przekraczać uprawnień posiadanych przez moderatora.

```
 public class UserPermissionController : Controller
    {
        private readonly AppDbContext _appDbContext;

        public UserPermissionController(AppDbContext appDbContext)

        public async Task<IActionResult> Index(string id, int? IdRegion, int? IdTrail, int?
IdShelter, int? IdParty)

        public async Task<IActionResult> Details(string? id)

        public IActionResult Create()

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(AppUser user, string[] selectedPartys,
string[] selectedRegions, string[] selectedShelters, string[] selectedTrails)

        public async Task<IActionResult> Edit(string id)

        private void PopulateUser(AppUser userToUpdate)

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(string id, string[] selectedPartys, string[]
selectedRegions, string[] selectedShelters, string[] selectedTrails)

    }
```

UserPermissionController odpowiada za zarządzanie uprawnieniami użytkowników w panelu admina. Umożliwia usuwanie i dodawanie uprawnień poszczególnym użytkownikom.

7. Opis wybranych funkcjonalności

7.2 Przeglądanie powiadomień

1. Niezalogowany użytkownik wchodzi na stronę powiadomień (/Notifications)
2. Wykonywana jest metoda Index kontrolera NotificationsController
3. Tworzony jest nowy widok, do którego przekazywane są: messageRepository umożliwiające dostęp do wiadomości bazy danych, oraz lista regionów
4. W widoku (ale po stronie serwera) wykonywana jest metoda GetAllMessage() na rzecz obiektu messageRepository, która wykonuje odpowiednie zapytanie do bazy danych i zwraca listę wiadomości
5. Widok jest generowany i wysyłany do klienta
6. Po stronie klienta lista wiadomości jest następnie filtrowana, na podstawie wybranych przez użytkownika ustawień

7.3 Dodawanie wiadomości

1. Moderator wchodzi na stronę dodawania wiadomości
2. Weryfikowane są jego uprawnienia
3. Wyświetlany jest formularz pozwalający dodać wiadomość, tylko do regionów do których moderator ma uprawnienia

4. Po naciśnięciu przycisku "Create" strona jest odświeżana i wywoływana jest metoda Index klasy MailController z argumentami zawierającymi m.in. Tytuł i treść nowej wiadomości
5. Weryfikowane są uprawnienia, czy zalogowany użytkownik mógł tą wiadomość dodać
6. Ustawiana jest godzina nadania wiadomości
7. na rzecz messageRepository wywoływana jest metoda AddMessageAsync, która wykonuje odpowiednie zapytanie do bazy danych dodające do niej wiadomość.

8. Specyfikacja zewnętrzna

Opis użytkowania poszczególnych stron:

1. Informacje turystyczne
   Strona umożliwia przeglądanie wszystkich atrakcji, szlaków, wydarzeń czy schronisk, które są wpisane w naszym systemie. Umożliwia filtrowanie wyników według regionów oraz typu atrakcji, jak również wygenerowanie raportu .pdf ze wszystkimi atrakcjami. Aby przefiltrować wyniki należy kliknąć w "filtruj", a następnie zaznaczać lub odznaczać poszczególne opcje. Strona będzie aktualizować się automatycznie
2. Powiadomienia
   Strona umożliwia przeglądanie wszystkich powiadomień dotyczących atrakcji, szlaków, wydarzeń czy schronisk, które są wpisane w naszym systemie. Umożliwia filtrowanie wyników według regionów, jak również subskrypcje do danego regionu. Opcja subskrypcji dostępna jest tylko dla zalogowanych użytkowników. Aby edytować swoje subskrypcje należy kliknąć "edytuj subskrypcje" a następnie wybrać, które regiony chce się zasubskrybować. Aby wyświetlić tylko wiadomości z danego regionu należy kliknąć przycisk "pokaż subskrypcje"
3. Wiadomości
   Strona umożliwia dodawanie nowych wiadomości do systemu. Wymaga bycia zalogowanym jako administrator lub moderator oraz, w przypadku moderatora, wymaga uprawnień do danego regionu"
4. Użytkownicy
   Strona umożliwia nadawanie uprawnień użytkownikom. Wymaga bycia zalogowanym jako administrator lub moderator oraz, w przypadku moderatora, nie pozwala na nadanie uprawnień do regionów, do których moderator sam nie ma uprawnień"
5. Panel Admina
   Strona umożliwia bezpośredni wgląd oraz edycję bazy danych. Wymaga bycia zalogowanym jako administrator."
6. Zarządzanie rolami
   Strona umożliwia bezpośredni wgląd oraz edycję bazy danych. Wymaga bycia zalogowanym jako administrator."
7. Rejestracja
   W celu rejestracji w systemie, należy podać nazwę użytkownika, email, hasło a następnie potwierdzić adres, poprzez wejście w przesłany emailem link. Można teraz zalogować się do systemu

W systemie na potrzeby testów są utworzone następujące konta:

Konto moderatora - login: kaktus, hasło: kaktus1

Konto administratora – login konto, hasło: haslo1

9. Wnioski

Projekt okazał się dla nas wyzwaniem na wielu płaszczyznach. Nie tylko pierwszy raz tworzyliśmy program zintegrowany z bazą danych, lecz także pierwszy raz tworzyliśmy stronę internetową w oparciu o ASP.NET. Zastosowaliśmy w praktyce, nabyte w poprzednim semestrze umiejętności związane z projektowaniem baz danych, choć w trakcie tworzenia programu projekt bazy danych i tak uległ drobnym zmianom. Nabyliśmy także umiejętność tworzenia zapytań do bazy danych z poziomu języka C# korzystając z LINQ oraz Entity Framework, jak również modelowania w programie klas encji odpowiadających bazie danych.

Poznaliśmy także podstawy tworzenia aplikacji internetowych z wykorzystaniem ASP.NET. Jest to niezwykle rozbudowane narzędzie, które momentami nas nieco przerastało. Jego rozbudowanie ma jednak także zalety gdyż, gdy się już je nieco pozna, pozwala łatwo tworzyć np. mechanizm logowania i weryfikacji użytkowników, formularze do edycji modelu itp.

Tworzenie front endu okazało się bardziej czasochłonne niż myśleliśmy, co sprawiło, że niektóre ekrany (dla moderatora czy administratora) pozostawiają wiele do życzenia. Zdecydowaliśmy się na użycie czystego JS i CSS, ale z perspektywy czasu stwierdziliśmy, ze użycie np. Bootstrapa znacznie przyspieszyłoby tworzenie front endu.