POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

SPEC. GRY I TECHNOLOGIE INTERNETOWE

Dokumentacja

Systemu zarządzania treścią

LABORATORIUM

DANIEL ZWIERZCHOWSKI, 141338
daniel.zwierzchowski@student.put.poznan.pl
ERYK ŁAWNICZAK, 158959
eryk.lawniczak@student.put.poznan.pl

PROWADZĄCY:
DR INŻ. MICHAŁ APOLINARSKI
michal.apolinarski@put.poznan.pl

Spis treści

Streszczenie

Celem projektu jest stworzenie systemu zarządzania treścią dla pizzerii Pani Basi. Koncepcja zakłada stworzenie dwóch podsystemów

- **Strony klienckiej** - pozwalającej wszystkim zainteresowanym uzytkownikom wyświetlić ofertę restauracji jak i zapoznać sie z firmą i jej zespołem.
- **Panelu administracyjnego** - umożliwiającego edycję zawartości strony w łatwy, dla przeciętnego uzytkownika, sposób.
- **Aplikacji obsługującej system** - zarządzającegj zapytaniami, przepływem informacji w systemie i komunikacją z bazą danych.

Strona pizzerii oferuje pełen dostęp do menu restauracji oraz do informacji, które nasz klient postanowi udostępnić. System został stworzony z myślą o realizacji konkretnego szablonu wybranego przez klienta. Projekt strony klienckiej jest dostępny pod adresem https://demo.templatemonster.com/demo/51689.html, natomiast projekt panelu administracyjnego znajduje się pod adresem https://www.creative-tim.com/product/soft-ui-dashboard?tracking=first-time

1. Wymagania(z podziałem na aktorów)

1.1. Funkcjonalne

Użytkownik niezalogowany

1. Wyświetlanie zamieszczonej przez administratora treści
   (a) Strony głównej i jej elementów
   (b) Galerii
   (c) Strony z menu restauracji - z możliwością wyświetlenia konretnych kategorii
   (d) Strony z kontaktem
2. Możliwość logowania do panelu administratora
3. Możliwość zresetowania zapomnianego hasła

Adminstrator

1. Dostęp do panelu administracyjnegu umożliwiającego zarządzanie zawartością
2. Możliwość zmiany hasła do konta
3. Dodawanie i edycja produktów wraz z kategoriami
4. Zarządzanie odnośnikami do mediów społecznościowych - dodawanie dowolnej ilości par ikon i linków
5. Zarządzanie menu strony klienckiej - edycja ilości odnośników menu wraz z obsługą zagłębienia elementów (do 2 poziomów)
6. Zarządzanie plikami graficznymi - możliwość dodawania i usuwania plików graficznych
   (a) automatyczne tworzenie przeskalowanych wersji obrazów, z mniejszą rozdzielczością
7. Zarządzanie galeriami - możliwość dodania wielu galerii ze zdjęciem głównym i dowolną liczbą zdjęć w galerii, tytułem oraz opisem
8. Zarządzanie sliderem na stronie głównej - slider składa się z obrazu, tytułu i krótkiego tekstu
9. Zarządzanie banerami między sekcjami - możliwość dodania wielu banerów składających się z obrazka, tekstu i opcjonalnego odnośnika
10. Możliwość zaznaczenia do 4 produktów jako polecane
11. Zarządzanie opiniami
12. Zarządzanie stopką
13. Zarządzanie członkami zespołu
14. Zarządzanie meta description

1.2. Niefunkcjonalne

1. Strona powinna być responsywna
2. Kompatybilność z przeglądarkami Chrome i Opera
3. Hasło administratora, przechowywane w bazie danych zabezpieczone z użyciem funkcji hashującej

2. Architektura systemu, narzędzia i środowisko

2.1. Narzędzia

1. Github Desktop / Fork
2. Visual Studio Code
3. Visual Studio
4. pgAdmin 4
5. sql developer data modeler

2.2. Kontrola wersji

Kontrola wersji jest realizowana przy pomocy narzędzia GitHub. Kod źródłowy można znaleźć pod adresem https://github.com/e-lawniczak/Cms

2.3. Frontend

Część wizualna projektu jest zaimplementowana przy użyciu następujących technologii

- RazorPage
- HTML5
- SCSS, CSS
- JS
- React

Uzycie tych technologii pozwala na prostą implementację szablonu. RazorPage pozwalają na łatwe zarządzanie strukturą projektu oraz zapewniają wbudowany rosty w obsłudze routing. Poszczególne podstrony zostały wykonane przy użyciu frameworka React.

2.4. Backend

Strona backendowa projektu została zrealizowana za pomocą frameworku **ASP.Net Core Web API** w wersji 6.0. Wybrany został, ponieważ jest wydajny i wieloplatformowowy. Dodatkowo jest duża dostępność narzędzi programistycznych i bogate wsparcie społeczności. Ostatnim powodem dlaczego wybrany został ten framework jest znany twócom język programowania C#.

2.5. ORM

Jako framework do mapowania encji (ORM), wybrano **FluentNHibernate** ze względu na jego wyjątkową zdolność do intuicyjnego mapowania obiektów. Jego kod jest kompilowany, co umożliwia wczesne wykrywanie potencjalnych błędów i ich skorygowanie. Ponadto, jego czytelność i zrozumiałość dla innych programistów czynią go doskonałym wyborem.

2.6. Baza danych

Do przechowywania wszystkich danych projektu wybrano jedną z najbardziej renomowanych o otwartym kodzie źródłowym baz danych relacyjnych - **Postgres**.

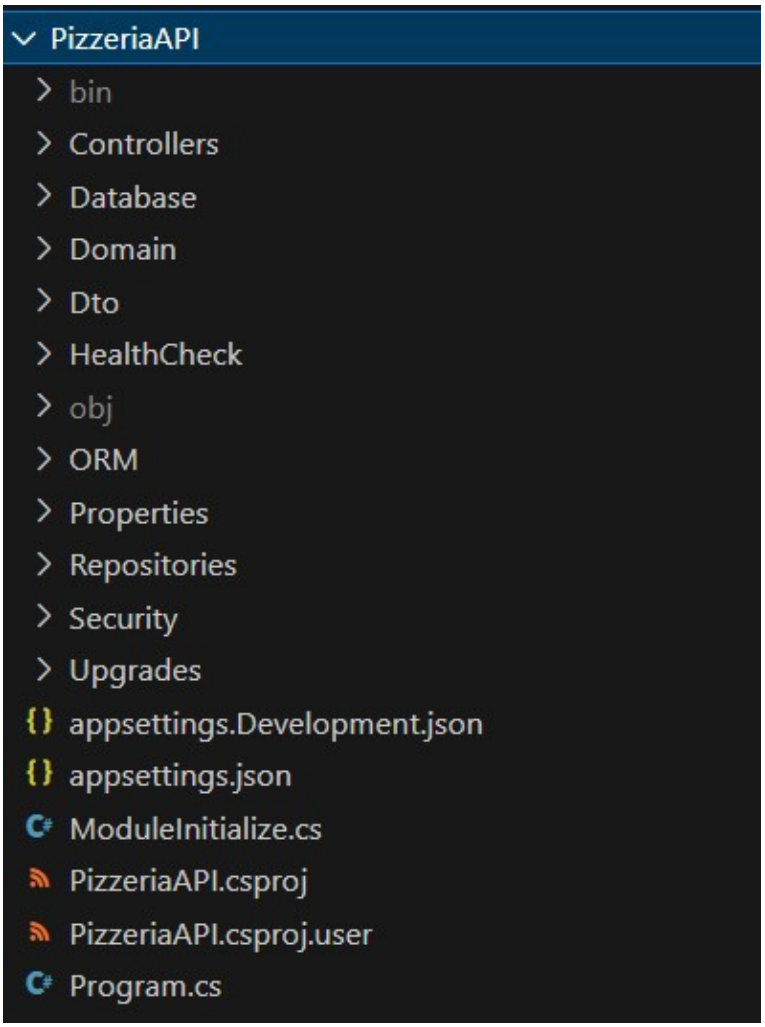
2.7. Struktura folderów

Rozwiązanie aplikacji składa się z 3 osobnych projektów. Każdy z nich odpowiada za osobną część systemu.

- PizzeriaAPI - metody api i połączenia z bazą danch
- PizzeriaFront - strona i logika klienta API dla zwykłych użytkowników
- PizzeriaFrontAdmin - strona i logika klienta API dla administratora strony

PizzeriaAPI Struktura projektów jest bardzo podobna. Zostały one rozdzielone z powodu wykorzystania szablonów, aby zapewnić łatwiejsze utrzymanie, oraz bardziej przejrzystą strukturę i kod.

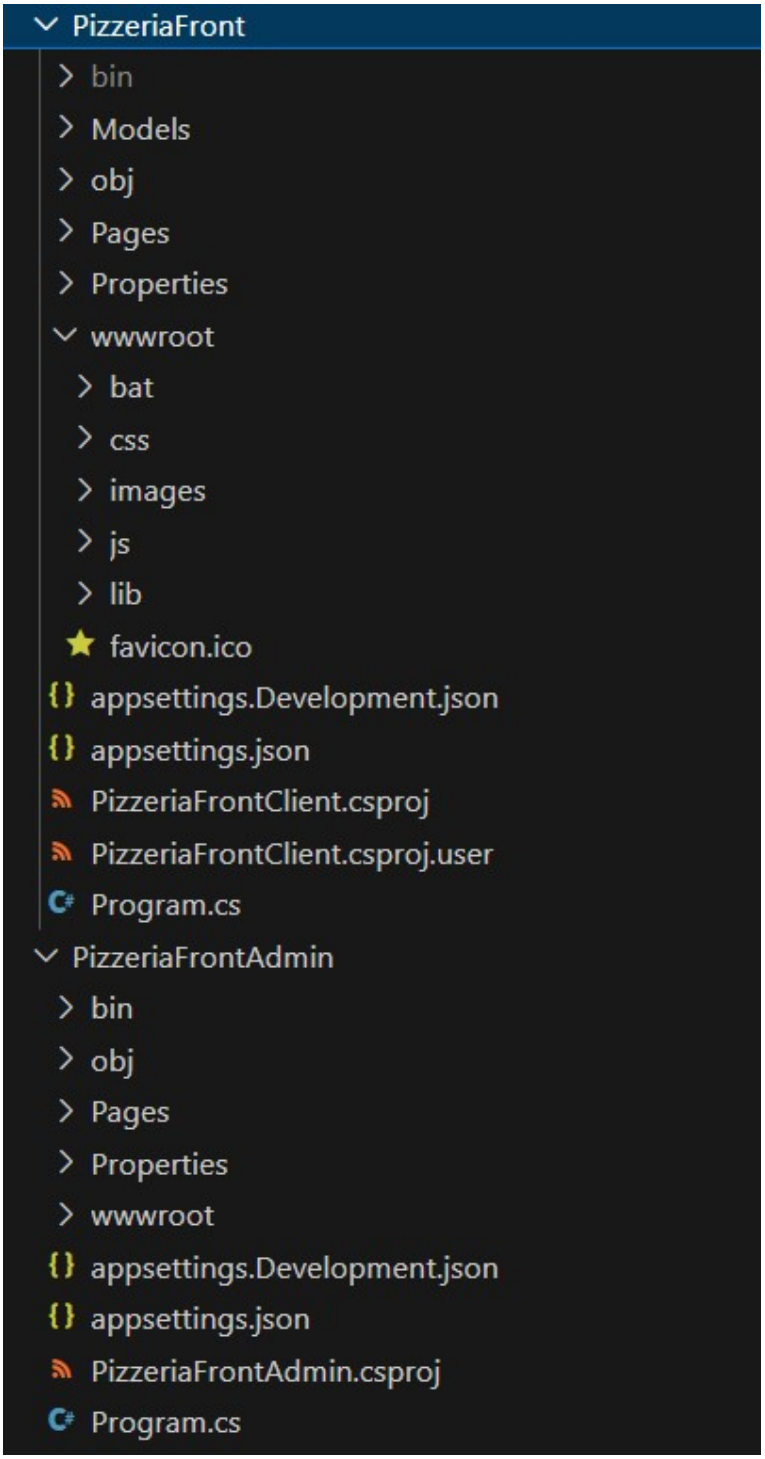
- Controllers - zawiera kontrolery, które odpowiadają za wystawianie endpointów z którymi komunikuje się frontend
- Database - zawarte są wszystkie encje występujące w bazie danych, wraz z ich mapowaniem
- Domain - występują dodatkowe klasy, któe nie występują w bazie danych
- Dto - obiekty, które są przekazywane pomiędzy backendem a frontendem zamiast obiektów bazodanowych
- HealthCheck - klasy odpowiadające za sprawdzanie stanu aplikacji
- ORM - zawarte są tutaj obiekty obsługujące połączenie z bazą danych
- Repositories - repozytoria odpowiadające za wykonywanie operacji bazodanowych. Tworzenie, edycja, pobieranie encji
- Security - zawarte są obiekty odpowiadające za uwierzytelnianie i autoryzacje
- Upgrades - System upgradów, pozwalających na tworzenie, modyfikację struktury, oraz modyfikację rekordów bazy danych
- ModuleInitialize - klasa rozszerzająca plik startowy, następuje tutaj inicjalizacja wszystkich repozytoriów, controllerów, upgradów oraz uwierzytelnianai i autoryzacji
- Program.cs - plik startowy aplikacji, w którym znajdują się polecenia konfiguracyjne



Rysunek 1: Struktura folderów dla projektu PizzeriaAPI

PizzeriaFront i PizzeriaFrontAdmin Struktura projektów jest bardzo podobna. Zostały one rozdzielone z powodu wykorzystania szablonów, aby zapewnić łatwiejsze utrzymanie, oraz bardziej przejrzystą strukturę i kod.

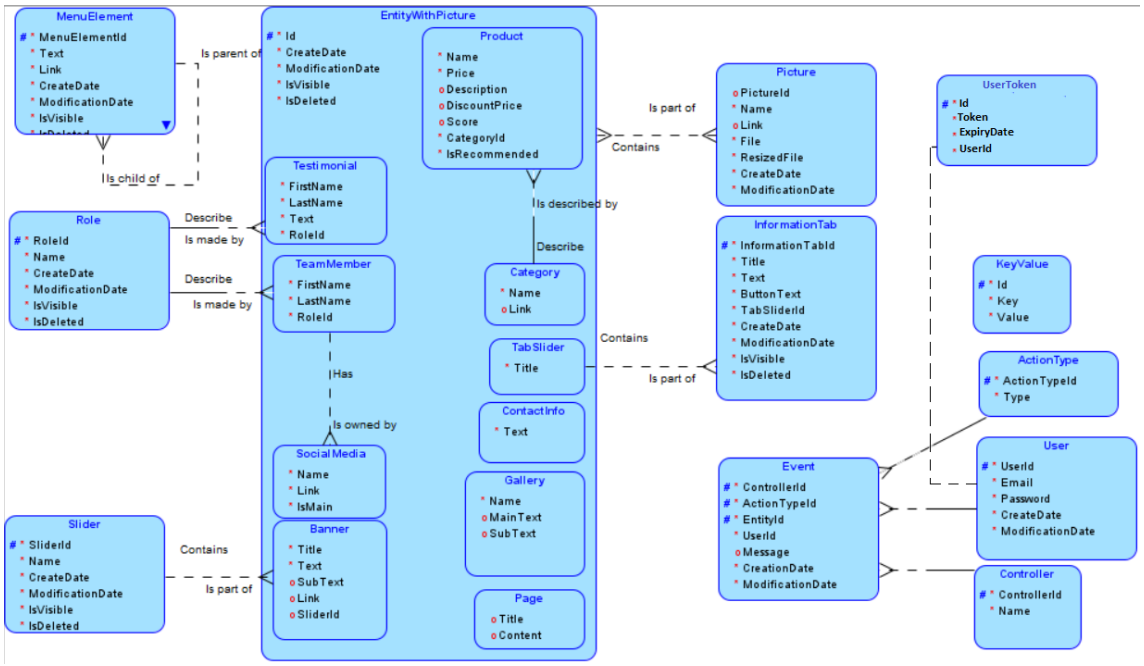
- Models - zawiera modele danych używane w projekcie
- Pages - zawiera podstrukturę folderów, które tworzą strukturę witryny, nazwy są istotne, ponieważ na ich podstawie tworzone są url'e stron.
- wwwroot - zawiera statyczne pliki, któe są wyświetlane w przeglądarce takie jak skrypty, style, czy obrazy
- Program.cs - plik startowy aplikacji, w któym znajdują się polecenia konfiguracyjne



Rysunek 2: Struktura folderów dla projektów PizzeriaFron oraz PizzeriaFrontAdmin

3. Schemat bazy danych

3.1. Model związków encji



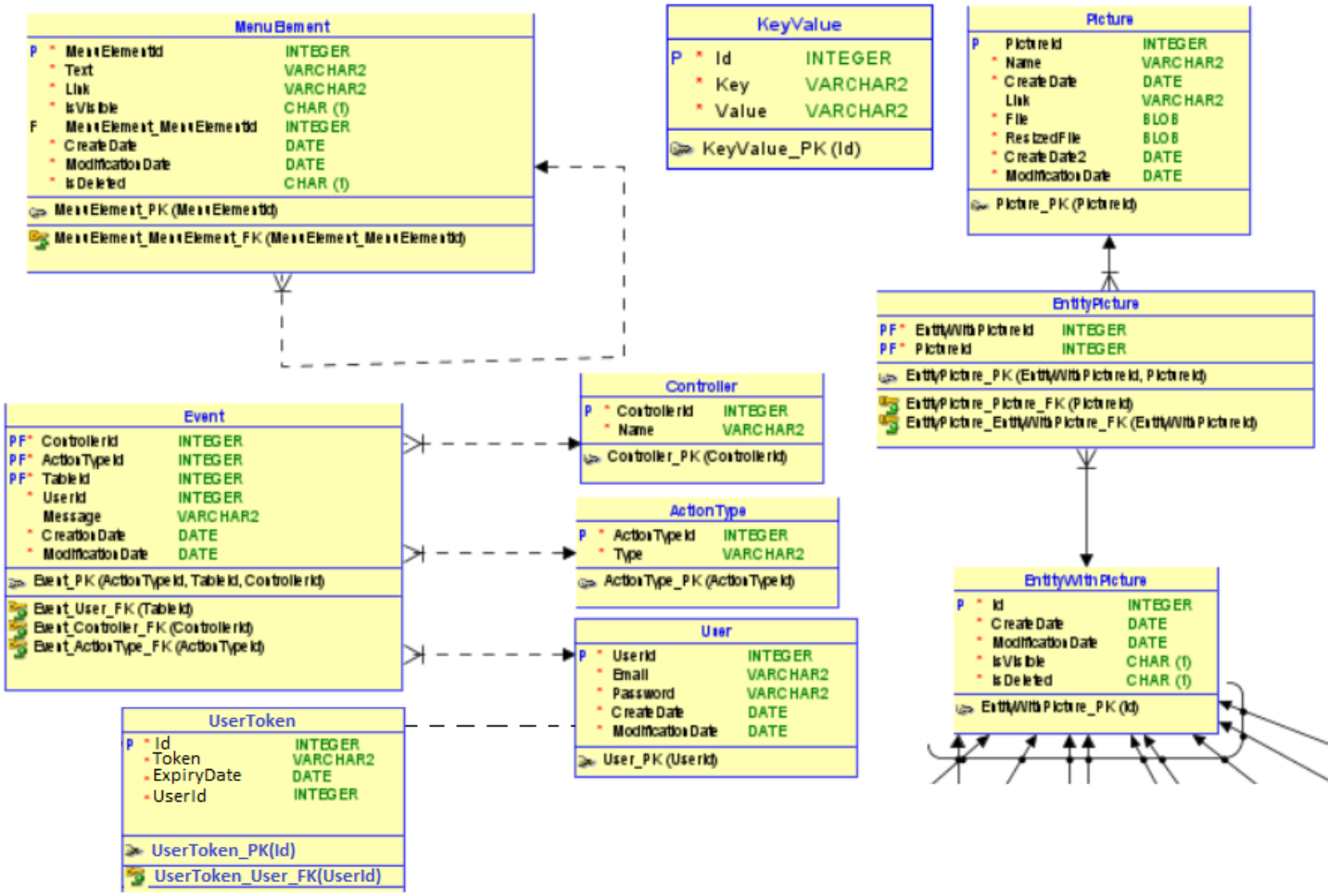
Rysunek 3: Model związków encji

Poszczególne encje widoczne na rysunku 1 mają następujące znaczenie:

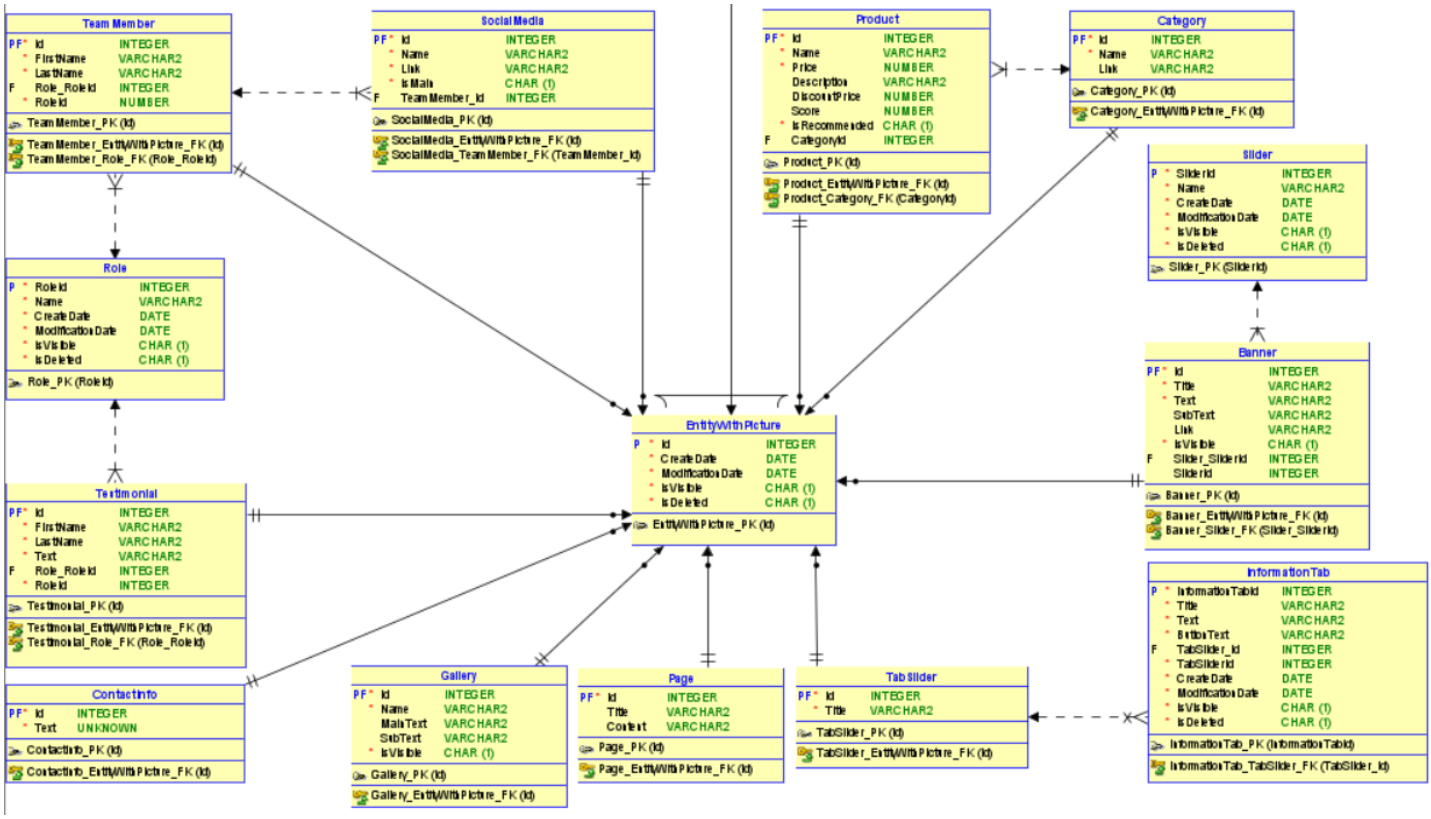
1. Action type - encja opisująca typy akcji w zdarzeniach systemowych.
2. Banner - encja opisująca obiekty z informacjami i promocjami pojawiające się między sekcjami, może także wchodzić w skład slideru.
3. Category - encja opisująca istniejące kategorie produktów.
4. Contact info - encja opisująca dane kontaktowe wyświetlane na stronie.
5. Controller - encja opisuąca instniejące kontrolery.
6. Entity with picture - Zbiorcza encja opisująca relację encji dziedziczących z encją Picture.
7. Event - encja opisująca zdarzenie w systemie. Identyfikowana za pomocą typu akcji, obiektu, którego dotyczy akcja, kontrolera z któego została ona wywołana.
8. Gallery - encja opisująca stworzone galerie, wyświetlane na stronie.
9. Information tab - encja opisująca zawartość jednej zakładki z tekstem, wchodzącej w skład tab slideru.
10. Menu element - encja opisująca elementy menu i zależności między nimi.
11. Picture - encja opisująca obrazy zamieszczone przez administratora jak i informacje o nich.
12. Product - encja opisująca informacje dotyczące dostępnych produktów takie jak cena, nazwa, opis, przecena, zdjęcie, kategoria.
13. Rich text element - encja opisująca stałe elementy strony składające się z samego tesktu, których tresć jest edytowalna.
14. Role - encja opisująca role, przypisywane członkom zespołu oraz osobom, które przekazały swoją opinię.
15. Slider - encja opisująca slidery pojawiające się na stronie. Slider to kilka bannerów występujących po sobie.
16. Social media - encja opisująca istniejące obiekty mediów społecznościowych, składająca się z linku i zdjęcia.

17. Tab slider - encja opisująca zbiór zakładek z informacjami.
18. Team member - encja opisująca członków zespołu wyświetlanych na stronie "About".
19. Testimonials - encja opisująca elementów opinii na temat fimry.
20. User - encja opisująca użytkownika.

3.2. Model relacyjny



Rysunek 4: Pierwsza część modelu relacyjnego

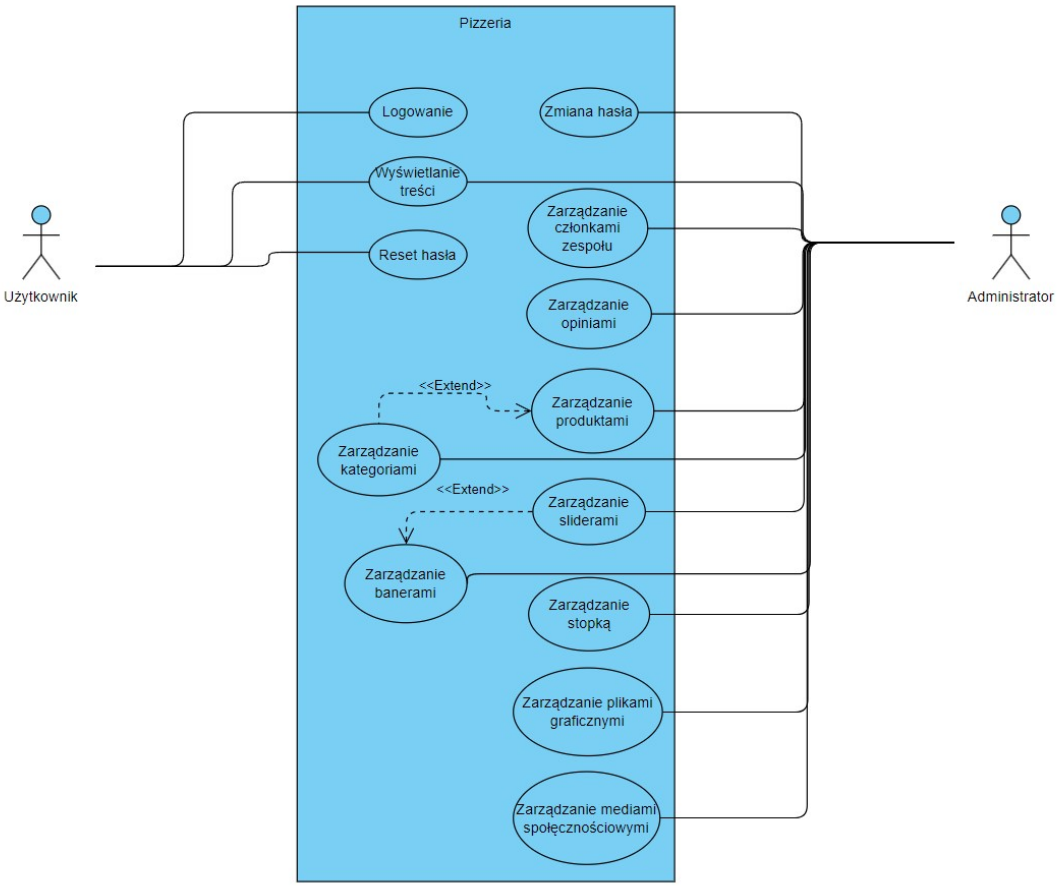


Rysunek 5: Druga część modelu relacyjnego

4. Diagramy UML

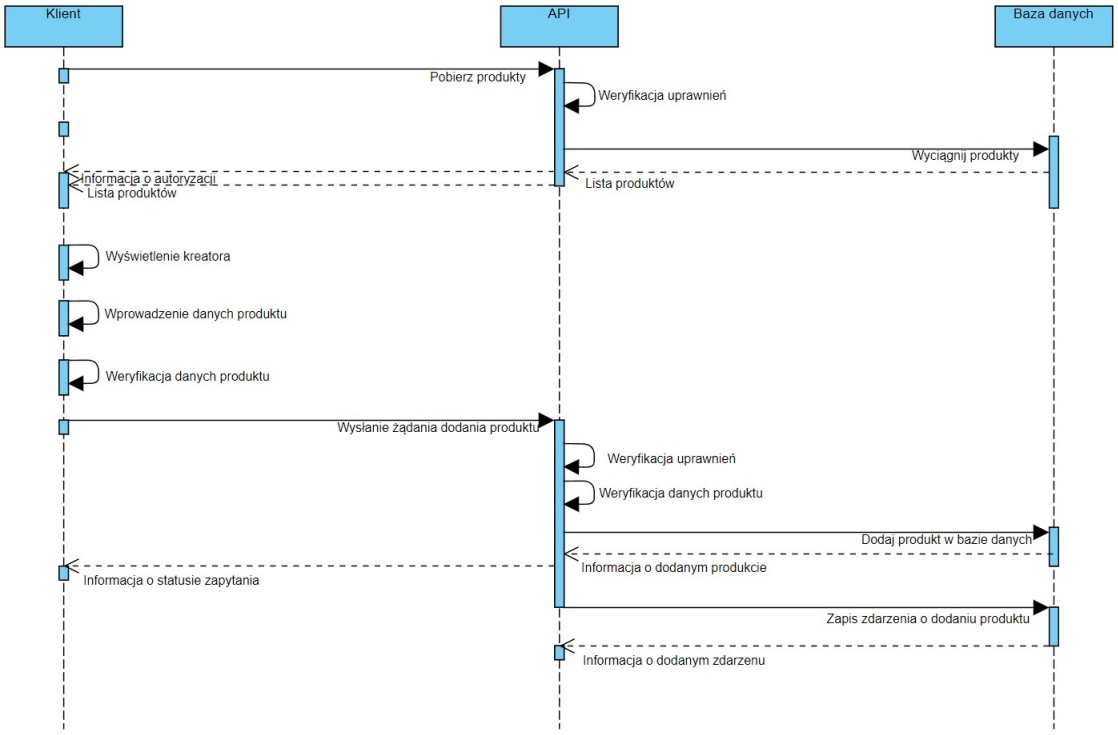
4.1. Diagram przypadków użycia

Diagram przedstawia przypadki użycia naszego systemu z podziałem na użytkownika i administratora. Użyte słowo kluczowe zarządzanie, oznacza dodawanie, usuwanie oraz edycję danego elementu systemu.



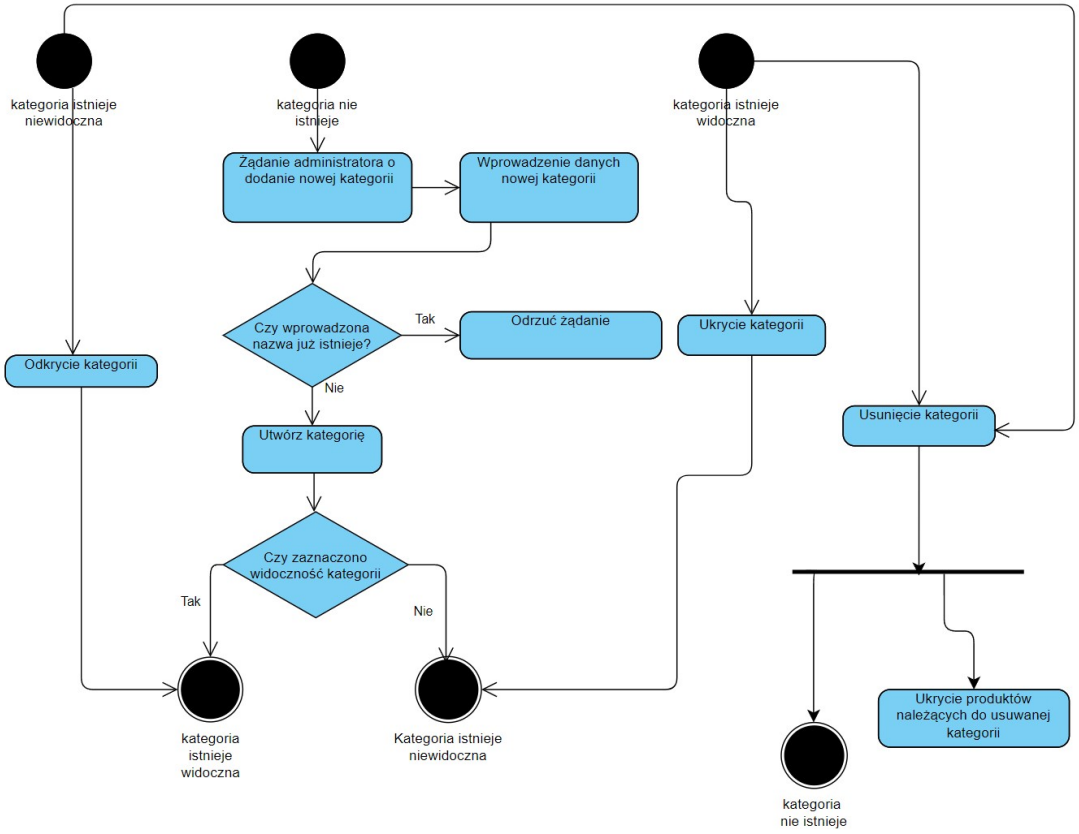
Rysunek 6: Diagram przypadków użycia

4.2. Diagram przebiegów



Rysunek 7: Diagram przebiegów dodawania produktu dla administratora (w przypadku wpisywania poprawnych danych)

4.3. Diagram stanów



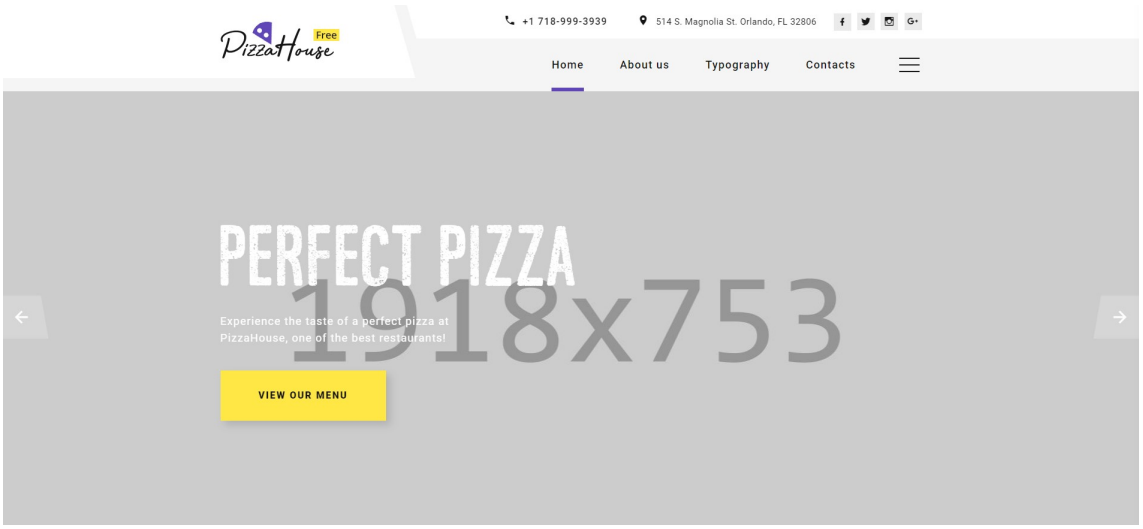
Rysunek 8: Diagram stanów

4.4. Diagram klas

Diagram klas z racji na swoje duże wymiary, został dołączony do dokumentacji jako załącznik. Znajduje się na dysku google pod adresem: https://drive.google.com/file/d/169avQ2oeeM2BjTi3Ce7R7cnYRLHJVmmf/view?usp=sharing

5. Projekty interfejsu graficznego

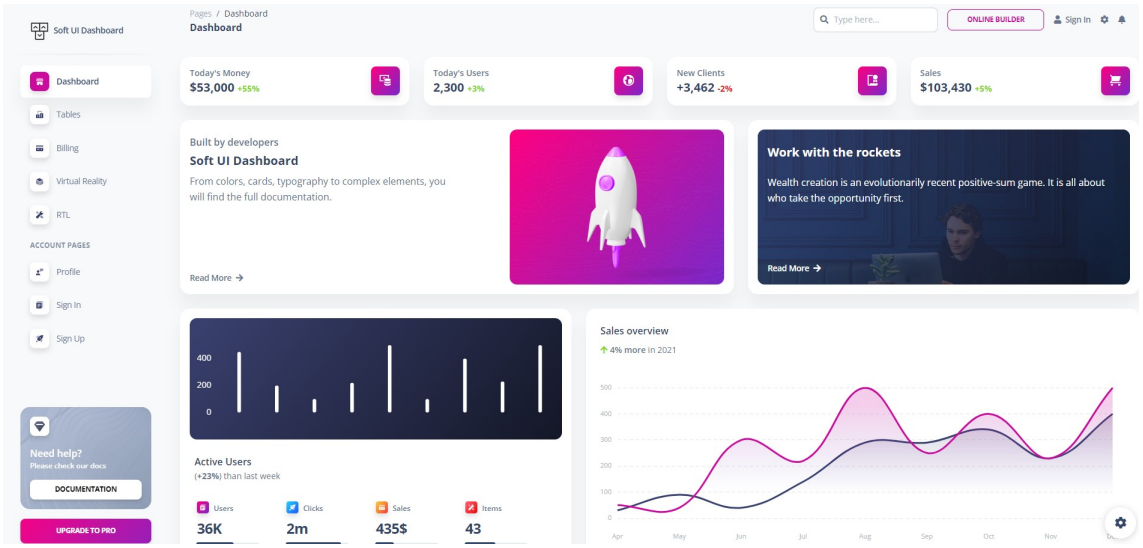
5.1. Widok publiczny



Rysunek 9: Interfejs graficzny widoku klienta

Projekt interfejsu graficznego dla zwykłego użytkownika strony wykorzystuje szablon *Free HTML5 Theme for Restaurant Website Website Template* pobrany ze strony https://demo.templatemonster.com/demo/51689.html

5.2. Panel administracyjny



Rysunek 10: Interfejs graficzny panelu administracyjnego

Projekt interfejsu graficznego widoku administracyjnego wykorzystuje szablon *Soft UI dashboard* pobrany ze strony https://www.creative-tim.com/product/soft-ui-dashboard?tracking=first-time

6. Najważniejsze metody i fragmenty kodu aplikacji

6.1. Logowanie

```
public async Task<AuthenticationResponse> LoginAsync(AuthenticationRequest request)
{
    var eduUser = await userManager.FindByEmailAsync(request?.Email ?? "");

    if (eduUser == null)
        throw new Exception("Not found user with given email");
    var hash = BCrypt.Net.BCrypt.HashPassword(request?.Password, securitySettings.Salt);
    if (hash != eduUser.Password)
        throw new UnauthorizedAccessException("Wrong email or password");
    JwtSecurityToken jwtSecurityToken = await GenerateToken(eduUser);

    AuthenticationResponse response = new AuthenticationResponse
    {
        Id = eduUser.Id,
        Token = new JwtSecurityTokenHandler().WriteToken(jwtSecurityToken),
        Email = eduUser.Email,
    };

    return response;
}
```

Listing 1: AuthenticationService.cs

6.2. Generowanie tokenu JWT

```
private async Task<JwtSecurityToken> GenerateToken(User user)
{
    var userClaims = await userManager.GetClaimsAsync(user);
    var roles = await userManager.GetRolesAsync(user);

    var roleClaims = new List<Claim>();

    for (int i = 0; i < roles.Count; i++)
    {
        roleClaims.Add(new Claim(ClaimTypes.Role, roles[i]));
    }

    var claims = new[]
    {
    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
    new Claim(JwtRegisteredClaimNames.Email, user?.Email??""),
    new Claim("uid", user?.Id.ToString()??""),
    new Claim(ClaimTypes.Role,"adminEdu")
    }
    .Union(userClaims)
    .Union(roleClaims);

    var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(securitySettings?.Key ??
    ""));
    var signingCredentials = new SigningCredentials(symmetricSecurityKey,
    SecurityAlgorithms.HmacSha256);

    var jwtSecurityToken = new JwtSecurityToken(
        issuer: securitySettings?.Issuer,
        audience: securitySettings?.Audience,
        claims: claims,
```

```
        expires: DateTime.UtcNow.AddMinutes(securitySettings.DurationInMinutes),
        signingCredentials: signingCredentials);
    return jwtSecurityToken;
}
```

Listing 2: AuthenticationService.cs

6.3. Konfiguracja uwierzytelniania

```
services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
    .AddJwtBearer(o =>
    {
        o.RequireHttpsMetadata = false;
        o.SaveToken = false;
        o.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ClockSkew = TimeSpan.Zero,
            ValidIssuer = configuration["SecuritySettings:Issuer"],
            ValidAudience = configuration["SecuritySettings:Audience"],
            IssuerSigningKey = new
    SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["SecuritySettings:Key"] ?? ""))
        };

        o.Events = new JwtBearerEvents()
        {
            OnAuthenticationFailed = context => { return Task.CompletedTask; },
            OnChallenge = context =>
            {
                context.HandleResponse();
                context.Response.StatusCode = 401;
                context.Response.ContentType = "application/json";
                var result = JsonConvert.SerializeObject("401 Not authorized");
                return context.Response.WriteAsync(result);
            },
            OnForbidden = context =>
            {
                context.Response.StatusCode = 403;
                context.Response.ContentType = "application/json";
                var result = JsonConvert.SerializeObject("403 Not authorized");
                return context.Response.WriteAsync(result);
            },
        };
    });
}
```

Listing 3: ModuleInitialize.cs

6.4. Endpoint do osbłsugi elementu

```
[HttpPost]
[Authorize]
[Route("/AddBanner")]
[SwaggerResponse(HttpStatusCode.OK, "Banner inserted successfully")]
public async Task<ActionResult> AddBanner([FromBody] AddBannerDto bannerDto)
{
    var banner = await GetBanner(bannerDto);
    await transactionCoordinator.InCommitScopeAsync(async session =>
    {
        await bannerRepository.InsertAsync(banner, session);
    });

    return Ok("Banner inserted successfully");
}
```

Listing 4: BannerController.cs

6.5. Obsługa transakcji zatwierdzającej

```
public async Task<T> InCommitScopeAsync<T>(Func<ISession, Task<T>> action)
{
    T? result;
    using (var session = nHibernateHelper.OpenSession())
    using (var transaction = session.BeginTransaction())
    {
        try
        {
            result = await action(session);
            await transaction.CommitAsync();
        }
        catch
        {
            await transaction?.RollbackAsync();
            throw;
        }
        finally
        {
            session?.Close();
        }

        return result;

    }
}
```

Listing 5: TransactionCoordinator.cs

6.6. Model bazowy stron aplikacji frontendowej dla admina

```
public async Task<T> InCommitScopeAsync<T>(Func<ISession, Task<T>> action)
 public abstract class PizzeriaPageModel : PageModel
 {
     public IOptions<HashSettings> hashSettings { get; set; }
     public UserModel? User { get; set; }
     public string Title { get; set; } = string.Empty;

     public virtual void OnGet()
```

```
    {
        var user = Request.Cookies["user"];
        var token = Request.Cookies["token"];
        var id = Request.Cookies["id"];
        if (token != null && user != null && id != null)
        {
            User = new UserModel(user, token, Int32.Parse(id));
            OnUserLogged();
        }
        else
        {
            User = null;
            OnUserNotLogged();
        }
    }
    protected void SaveUser(string email, string token, int id)
    {
        User = new UserModel(email, token, id);

        Response.Cookies.Append("user", email);
        Response.Cookies.Append("token", token);
        Response.Cookies.Append("id", id.ToString());
    }
    protected void LogOut()
    {
        User = null;
        HttpContext.Session.Clear();
        Response.Cookies.Delete("user");
        Response.Cookies.Delete("token");
        Response.Cookies.Delete("id");
    }
    public void OnUserNotLogged()
    {

        var url = HttpContext.Request.Path.Value;
        var query = HttpContext.Request.Query;
        if (User == null && url != "/Login" && url != "/Top/Secret/Register" && url !=
    "/ResetPassword")
            Response.Redirect("/Login");



    }
    public void OnUserLogged()
    {
        var url = HttpContext.Request.Path.Value;

        if (User != null && (url == "/Login" || url == "/Top/Secret/Register" || url ==
    "/ResetPassword"))
            Response.Redirect("/");
    }
}
```

Listing 6: PizzeriaPageModel.cs

7. Analiza bezpieczeństwa

7.1. Bezpieczeństwo haseł uzytkowników

Hasła i loginy użytkownika, są przechowywane w bazie danych na serwerze centralnym, do którego dostęp jest chroniony osobnym hasłem administracyjnym. Dodatkowo hasła w bazie, przechowywane są w postaci wyniku z jednokierunkowej funkcji skrótu. Wykorzystany został algorytm BCrypt, użyty dwukrotnie - przy wysyłaniu żądania oraz przy zapisie do bazy. Dzięki temu nawet po wycieku danych, odczytanie ich jest utrudnione.

7.2. Bezpieczeństwo zapytań API

Zapytania umożliwiające komunikację z bazą danych odbywają się za pomocą API. Dostęp do niego jest ograniczony za pomocą mechanizmu CORS. Aplikacja jest podzielona na części front-end i bakck-end, a dostęp do bazy i zapytań API ma tylko nasza strona. Dodatkowo zaimplementowany jest mechanizm sesji wykorzystujący tokeny JWT. Większość wrażliwych zapytań do API można wykonać, tylko za pomocą specjalnie wygenerowanego tokenu, który jest przyznawany użytkownikowi po zalogowaniu.

8. Podsumowanie

8.1. Podział prac

Projekt został wykonany wspólnie przez: Eryk Ławniczak, Daniel Zwierzchowski

- Eryk Ławniczak - frontend, projekt bazy danych, dokumentacja projektu
- Daniel Zwierzchowski - backend, projekt bazy danych, dokumentacja projektu

8.2. Cele zrealizowane, cele niezrealizowane, napotkane problemy

Cele zrealizowane Podczas implementacji projektu, udało się zrealizować wszystkie cele postawione przez założone wymagania funkcjonalne i niefunkcjonalne.

Funkcjonalne

Użytkownik niezalogowany

1. Wyświetlanie zamieszczonej przez administratora treści
   (a) Strony głównej i jej elementów
   (b) Galerii
   (c) Strony z menu restauracji - z możliwością wyświetlenia konretnych kategorii
   (d) Strony z kontaktem
2. Możliwość logowania do panelu administratora
3. Możliwość zresetowania zapomnianego hasła

Adminstrator

1. Dostęp do panelu administracyjnegu umożliwiającego zarządzanie zawartością
2. Możliwość zmiany hasła do konta
3. Dodawanie i edycja produktów wraz z kategoriami
4. Zarządzanie odnośnikami do mediów społecznościowych - dodawanie dowolnej ilości par ikon i linków
5. Zarządzanie menu strony klienckiej - edycja ilości odnośników menu wraz z obsługą zagłębienia elementów (do 2 poziomów)
6. Zarządzanie plikami graficznymi - możliwość dodawania i usuwania plików graficznych
   (a) automatyczne tworzenie przeskalowanych wersji obrazów, z mniejszą rozdzielczością
7. Zarządzanie galeriami - możliwość dodania wielu galerii ze zdjęciem głównym i dowolną liczbą zdjęć w galerii, tytułem oraz opisem
8. Zarządzanie sliderem na stronie głównej - slider składa się z obrazu, tytułu i krótkiego tekstu
9. Zarządzanie banerami między sekcjami - możliwość dodania wielu banerów składających się z obrazka, tekstu i opcjonalnego odnośnika
10. Możliwość zaznaczenia do 4 produktów jako polecane
11. Zarządzanie opiniami
12. Zarządzanie stopką
13. Zarządzanie członkami zespołu
14. Zarządzanie meta description

Niefunkcjonalne

1. Strona powinna być responsywna
2. Kompatybilność z przeglądarkami Chrome i Opera
3. Hasło administratora, przechowywane w bazie danych zabezpieczone z użyciem funkcji hashującej

Napotkane problemy Podczas implementacji napotkaliśmy kilka problemów. Większość z nich dotyczyła faktycznego wdrożenia zaprojektowanego schematu bazy danych, któy obejmował dziedziczenie tabel, w celu bardziej przejżystego i intuicyjnego wprowadzenia zdjęć do projektu. Wiele problemów pojawiło się podczas wykonywania operacji edycji obiektów w bazie.

Problemem okazała się również skala projektu. Chcąc zaimplementować każdy element strony jako osobny byt w bazie danych, zwiększona została ilość kodu. Wydaje się, żę część elementó można było uprościć i zgrupować w jednej tabeli.

8.3. Perspektywa rozwoju

Dzięki dosć unwersalnej implementacji backendu i bazy danych projekt nadaje się do ponownego użycia. Popodmienianiu zawartości i grafik, szablon mógłby zostać użyty do innych stron, niekoniecznie związanych z pizzą czy restauracjami.

Gdyby znaleźć kogoś, kto chciały kontynuować ten projekt przy pewnym nakłądzie pracy, możnaby dostosować projekt do tworzenia uniwersalnych stron opartych o szablon.