Dokumentacija za aplikaciju "Secret Santa"

1. Uvod

Aplikacija "Secret Santa" je dizajnirana za potrebe anonimnog darivanja među uposlenicima. Cilj aplikacije je omogućiti slučajno generisanje parova uposlenika koji će međusobno razmjenjivati poklone, uz poštivanje određenih pravila. Dodatno, aplikacija ima funkcionalnost "wishlist" koja omogućuje korisnicima da dodaju i upravljaju svojim listama želja.

2. Arhitektura aplikacije

Aplikacija je podijeljena na dva glavna dijela:

- Backend: Razvijen u .NET Core-u, koristi Entity Framework za upravljanje bazom podataka i Identity za autentikaciju i autorizaciju.
- Frontend: Razvijen u React-u, koristi Bootstrap za stilizaciju i Axios za komunikaciju sa backendom.

3. Backend

3.1. Program.cs

Fajl Program.cs je glavni fajl u .NET Core aplikaciji koji konfigurira i pokreće aplikaciju. Evo detaljnog objašnjenja svakog dijela:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using secret_santa.Context;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddControllers();

// Configure the database context.
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"))
);

// Configure CORS.
builder.Services.AddCors(options =>
{
```

```
    options.AddPolicy("AllowReactApp", builder =>
    {
        builder.WithOrigins("http://localhost:3000")
               .AllowAnyHeader()
               .AllowAnyMethod()
               .AllowCredentials(); // Allow credentials (cookies)
    });
});

// Configure Identity.
builder.Services.AddIdentity<IdentityUser, IdentityRole>(options =>
{
    options.SignIn.RequireConfirmedEmail = false;
    options.SignIn.RequireConfirmedAccount = false;
})
.AddEntityFrameworkStores<ApplicationDbContext>();

// Configure application cookies.
builder.Services.ConfigureApplicationCookie(options =>
{
    options.Cookie.SameSite = SameSiteMode.None; // Allow cross-site cookies
    options.Cookie.SecurePolicy = CookieSecurePolicy.Always; // Ensure cookies are sent over HTTPS
});

var app = builder.Build();
using var noviScope = app.Services.CreateScope();
var dbContext = noviScope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
dbContext.Database.Migrate();
// Create roles and admin user on application startup.
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
    var userManager = services.GetRequiredService<UserManager<IdentityUser>>();

    await CreateRolesAsync(roleManager);
    await CreateAdminUserAsync(userManager);
}

app.UseSwagger();
app.UseSwaggerUI();


app.UseHttpsRedirection();
app.UseRouting();

app.UseCors("AllowReactApp");

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();

// Helper method to create roles.
async Task CreateRolesAsync(RoleManager<IdentityRole> roleManager)
{
    string[] roleNames = { "ADMIN", "USER" };
    foreach (var roleName in roleNames)
    {
        var roleExist = await roleManager.RoleExistsAsync(roleName);
        if (!roleExist)
        {
            await roleManager.CreateAsync(new IdentityRole(roleName));
        }
    }
}

// Helper method to create the admin user.
```

```
async Task CreateAdminUserAsync(UserManager<IdentityUser> userManager)
{
    const string adminEmail = "admin@gmail.com";
    const string adminPassword = "Admin1!";

    var adminUser = await userManager.FindByEmailAsync(adminEmail);
    if (adminUser == null)
    {
        adminUser = new IdentityUser
        {
            UserName = adminEmail,
            Email = adminEmail,
            EmailConfirmed = true
        };

        var createUserResult = await userManager.CreateAsync(adminUser, adminPassword);
        if (createUserResult.Succeeded)
        {
            await userManager.AddToRoleAsync(adminUser, "ADMIN");
            Console.WriteLine("Admin user created successfully.");
        }
        else
        {
            Console.WriteLine("Failed to create admin user.");
        }
    }
    else
    {
        Console.WriteLine("Admin user already exists.");
    }
}
```

- Dodavanje servisa: builder.Services.AddEndpointsApiExplorer(), builder.Services.AddSwaggerGen() , i builder.Services.AddControllers() dodaju potrebne servise za API, Swagger dokumentaciju i kontrolere.
- Konfiguracija baze podataka: builder.Services.AddDbContext<ApplicationDbContext> konfigurira Entity Framework Core za korištenje SQL Server baze podataka.
- CORS konfiguracija: builder.Services.AddCors omogućuje komunikaciju između frontenda (React aplikacije na http://localhost:3000) i backenda.
- Identity konfiguracija: builder.Services.AddIdentity konfigurira ASP.NET Core Identity za upravljanje korisnicima i ulogama.
- Konfiguracija kolačića: builder.Services.ConfigureApplicationCookie konfigurira kolačiće za autentikaciju.
- Kreiranje uloga i admin korisnika: CreateRolesAsync i CreateAdminUserAsync metode kreiraju uloge i admin korisnika prilikom pokretanja aplikacije.

- Middleware: app.UseSwagger(), app.UseSwaggerUI(), app.UseHttpsRedirection(), app.UseRouting(), app.UseCors("AllowReactApp"), app.UseAuthentication(), i app.UseAuthorization() konfiguriraju middleware za Swagger, HTTPS redirekciju, rutiranje, CORS, autentikaciju i autorizaciju.
- Mapiranje kontrolera: app.MapControllers() mapira sve kontrolere u aplikaciji.

3.2. ApplicationDbContext.cs

Fajl ApplicationDbContext.cs definiše kontekst baze podataka za Entity Framework Core. On je povezan
sa Program.cs kroz builder.Services.AddDbContext<ApplicationDbContext>.

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using secret_santa.Models;

namespace secret_santa.Context
{
    public class ApplicationDbContext : IdentityDbContext<IdentityUser>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }

        public DbSet<Pair> Pairs { get; set; }
        public DbSet<Wishlist> Wishlists { get; set; }
    }
}
```

Objašnjenje:

- IdentityDbContext: Nasljeđuje IdentityDbContext<IdentityUser> za upravljanje korisnicima i ulogama.
- DbSet: DbSet<Pair> i DbSet<Wishlist> definišu tabele u bazi podataka za parove i želje.

3.3. Kontroleri

Postoje tri glavna kontrolera:

- AccountController: Upravlja registracijom, prijavom i odjavom korisnika.

  Ključne metode:

  - Register: Registrije novog korisnika i dodaje ga u ulogu "USER".
  - Login: Prijavljuje korisnika i vraća njegovu ulogu.
  - Logout: Odjavljuje korisnika.
  - CheckSession: Provjerava trenutnu sesiju korisnika.

- PairingController: Upravlja generisanjem i dohvatanjem parova.

  Ključne metode:

  - PairUsers: Generiše parove uposlenika nasumičnim odabirom.
  - GetPairs: Vraća listu generisanih parova.
  - Shuffle: Pomoćna metoda za nasumično miješanje liste korisnika.

- WishlistController: Upravlja željama uposlenika.

  Ključne metode:

  - GetWishlist: Vraća listu želja za određenog korisnika.
  - AddToWishlist: Dodaje novu stavku u listu želja.
  - DeleteWishlistItem: Briše stavku iz liste želja.

---

4. Frontend

4.1. Komponente

Frontend se sastoji od nekoliko ključnih komponenti:

- Navigation: Navigacijska traka koja prikazuje korisničke informacije i opciju za odjavu.
- WishlistCard: Komponenta za prikaz i upravljanje željama.
- AdminPage: Stranica za administratore gdje mogu generisati i pregledati parove.

- UserPage: Stranica za obične korisnike gdje mogu vidjeti kome treba kupiti poklon i upravljati svojim željama.
- Login i Register: Stranice za prijavu i registraciju korisnika.

4.2. Autentikacija i autorizacija

Frontend koristi kontekst za upravljanje stanjem korisnika. Korisnički podaci se čuvaju u AuthContext-u, a pri svakom zahtjevu se provjerava sesija korisnika.

```
export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(() => {
    //restore the user from local storage if it exists
    const storedUser = localStorage.getItem('user');
    return storedUser ? JSON.parse(storedUser) : null;
  });
  const [loading, setLoading] = useState(true); //loading state

  useEffect(() => {
    const fetchUser = async () => {
      try {
        const response = await api.get('/Account/check-session');
        const userData = response.data;
        setUser(userData);
        localStorage.setItem('user', JSON.stringify(userData));
      } catch (error) {
        console.error('Failed to fetch user session:', error);
        setUser(null);
        localStorage.removeItem('user');
      } finally {
        setLoading(false);
      }
    };

    fetchUser();
  }, []);

  const login = (userData) => {
    setUser(userData);
    localStorage.setItem('user', JSON.stringify(userData));
  };

  const logout = async () => {
    try {
      await api.post('/Account/logout');
    } catch (error) {
      console.error('Failed to logout:', error);
    }
    setUser(null);
    localStorage.removeItem('user');
  };

  return (
    <AuthContext.Provider value={{ user, login, logout, loading }}>
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => useContext(AuthContext);
```

4.3. Komunikacija sa backendom

Za komunikaciju sa backendom koristi se Axios biblioteka. Svi zahtjevi se šalju na backend API, a odgovori se obrađuju na frontendu.

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'https://localhost:44394/api',
  headers: {
    'Content-Type': 'application/json',
  },
  withCredentials: true, // Include cookies in requests
});

export default api;
```

5. Dodatne informacije

- Sigurnosne mjere: Aplikacija koristi HTTPS za sigurnu komunikaciju između frontenda i backenda. Također, koristi se Identity za autentikaciju i autorizaciju, što osigurava da samo autorizovani korisnici mogu pristupiti određenim funkcionalnostima.
- Skalabilnost: Aplikacija je dizajnirana tako da može lako biti proširena dodatnim funkcionalnostima, kao što su notifikacije, dodatne uloge korisnika, itd.
- Testiranje: Aplikacija je testirana kroz različite scenarije kako bi se osiguralo da sve funkcionalnosti rade kako je predviđeno.

6. Upute za pokretanje aplikacije

6.1. Kloniranje repozitorija:

```
git clone https://github.com/asmer085/secret_santa.git
cd secret_santa
```

6.2. Pokretanje Docker kontejnera:

```
docker-compose up --build
```

6.3. Pristup aplikaciji:

Otvorite web preglednik i idite na http://localhost:3000.

6.4. Rješavanje problema sa HTTPS:

Ako frontend ne može pristupiti backend endpointima,
otvorite https://localhost:44394/swagger/index.html

u pregledniku i odobrite pristup. Ovo je potrebno zbog self-signed certifikata koji se koristi za HTTPS. Nakon odobrenja, možda ćete morati ponovo pokrenuti kontejnere koristeći docker-compose up --build.

6.5. Admin korisnik:

Admin korisnik se automatski kreira kada pri pokretanju backenda. Korisnički podaci za admina su:

- Username: admin@gmail.com
- Password: Admin1!

---

7. Zaključak

Aplikacija "Secret Santa" je jednostavna, ali efikasna aplikacija koja omogućuje uposlenicima da anonimno razmjenjuju poklone. Kombinacija .NET Core backenda i React frontenda omogućuje brz razvoj i jednostavno održavanje. Aplikacija je skalabilna i može se proširiti dodatnim funkcionalnostima u budućnosti.