

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Objektno oblikovanje

PoGO Raids

SEMINAR

Helena Novak, Borna Relić

Zagreb, siječanj, 2021.

SADRŽAJ

Uvod	3
Design story	4
Zahtjevi	5
Opis baze podataka	10
Opis Windows Forms aplikacije	11
Opis iOS mobilne aplikacije	15
Opis Backend dio sustava	19

1. Uvod - Opis problema

Cilj seminarskog zadatka je primjenom postupaka objektnog oblikovanja izraditi aplikaciju za spajanje igrača koji žele sudjelovati u borbama, tzv. raidovima u popularnoj mobilnoj igrici PokemonGo. To je open world igrica u kojoj igrači (korisnici) hodajući svijetom skupljaju pokemone iz različitih pokemon regija. Raidovi u igrici su vrsta mini turnira gdje se više igrača bori protiv legendarnog pokemona i na kraju, ukoliko ga pobijede imaju šansu uhvatiti ga i dodati svojoj kolekciji pokemona. S obzirom da za sudjelovanje raidovima igrači moraju fizički biti blizu jedni drugih, a takav način zbog pandemije više nije moguć, došlo je do promjene u igrici te su se pojavili tzv. remote raidovi. To je vrsta turnira u kojem se ljudi ne moraju fizički nalaziti nego se međusobno mogu pozivati, a za to su osmislili veliki broj chat grupa te grupa na Facebooku gdje se dogovaraju oko raidova, ali to je sve skupa vrlo nestrukturirano i mala je riječ da vlada kaos. Ovdje dolazi ideja za **PoGo Raids**, odnosno aplikaciju preko koje bi korisnici mogli predložiti raid na kojem bi htjeli sudjelovati te se priključiti već postojećima.

2. Design story

Navedena aplikacija bit će izgrađena kao funkcionalan i pregledan informacijski sustav koji će igračima pružiti uslugu pristupanja i kreiranja tzv. raida, odnosno borbe protiv legendarnog pokemona.

Početna stranica nudi prikaz sličice aplikacije ispod koje se nalaze mjesta za unos korisničkih podataka za prijavu u sustav. Osim navedenoga, nalaze se i gumb za registraciju u sustav.

Aplikacija ima sljedeće vrste korisnika:

1. Registrirani korisnik
2. Neregistrirani korisnik

Prijava u aplikaciju za postojeće korisnike vrši se upisivanjem vlastitog korisničkog imena (ili vlastite e-mail adrese s kojom je račun povezan) i lozinke. Pritiskom na registraciju, korisniku se otvara stranica na kojoj korisnik unosi podatke potrebne za registraciju (korisničko ime, lozinka, ponovljena lozinka, e-mail adresa, ime i prezime, level unutar igrice, tim kojem pripada, korisničko ime u igrici i korisnički kod).

Prilikom uspješne registracije korisnik ima mogućnosti pregledavanja aktivnih oglasa za raidove, pristupanje raidu, odustajanje od raida, pregledavanje detalja aktivnih raidovi, pregledavanje detalja drugih igrača, uređivanje vlastitog korisničkog računa.

Registrirani korisnik može uređivati svoj profil, to uključuje promjenu korisničkog imena i promjene podataka vezanih uz lik u PokemonGo igrici. Također, korisnik može i izbrisati svoj korisnički račun ili se odjaviti iz sustava.

Na početnoj stranici registrirani korisnik vidi listu svih trenutno aktivnih oglasa za raidove. Prilikom odabira raida, moguće je gledati njegove detalje, što uključuje sliku pokemona i listu igrača koji sudjeluju i u obliku “ *korisničko ime - level u igrici* ” te opciju priključivanja tom raidu. Pristupanje pojedinom raidu i njegov završetak rezultiraju ažuriranjem korisnikove statistike. Uz to, omogućeno je korisniku odustajanje od raida kojem je pristupio.

S početne stranice omogućeno je i stvaranje, odnosno predlaganje novog raida. Prilikom stvaranja raida korisnik mora upisati ime pokemona u raidu, minimalni level za pristup i vrijeme kretanja u raid, nakon čega se ostali igrači mogu prijaviti za sudjelovanje. Predloženi raid također može biti i obrisani.

Ovaj projekt bi potencijalno mogao biti koristan u stvarnom životu s obzirom na popularnost igrice PokemonGo i količinu ljudi koji aktivno putem ovakvih mini turnira žele osvajati nove pokemone.

3. Zahtjevi

Skup mogućnosti unutar aplikacije

Neregistrirani korisnik može:

- Registrirati se u sustav

Registrirani korisnik može:

- Prijaviti se u sustav
- Odjaviti se sa svog korisničkog računa
- Izbrisati svoj korisnički račun
- Uređivati svoj korisnički račun
- Pregledavati trenutno aktivne oglase za raid
- Pregledavati detalje pojedinog raida
- Pristupiti željenom raidu
- Odustati od prije pristupljenog raida
- Predložiti novi raid
- Obrisati raid (koji je sam kreirao)
- Prikaz statistike najaktivnijih igrača

Razrada funkcionalnosti preko obrazaca uporabe - use cases

UC1 – Registracija korisnika

- **Glavni sudionik:** Neregistrirani korisnik
- **Cilj:** Registracija korisnika u sustav
- **Sudionici:** Baza podataka
- **Preduvjet:**
 1. Neregistrirani korisnik odabire opciju „Registriraj se“
- **Opis osnovnog tijeka:**
 1. Neregistrirani korisnik odabire opciju „Registriraj se“
 2. Neregistrirani korisnik unosi korisničko ime, lozinku, ponovljenu lozinku, adresu e-maila, ime i prezime, level unutar igrice, tim kojem pripada, korisničko ime u igrici i korisnički kod
 3. Podaci se spremaju u bazu podataka
 4. Korisnika se obavijesti o uspješnoj registraciji
- **Opis mogućih odstupanja:**
 - 2a. Adresa e-maila već postoji u bazi podataka
 1. Sustav javlja grešku
 - 2b. Unesena lozinka je kraća od 8 znakova
 1. Sustav javlja grešku
 - 2c. Korisnik nije popunio sva obavezna polja
 1. Sustav javlja grešku
 - 2d. Lozinke se ne poklapaju
 1. Sustav javlja grešku
 - 2e. Korisničko ime već postoji
 1. Sustav javlja grešku

UC2 – Prijava u sustav

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Prijava korisnika u sustav
- **Sudionici:** Baza podataka
- **Preduvjet:**
 1. Korisnik je registriran u sustav
- **Opis osnovnog tijeka:**
 1. Korisnik odabire opciju prijave
 2. Korisnik unosi korisničko ime i lozinku
 3. Korisnik je prijavljen u sustav i preusmjeren na početnu stranicu
- **Opis mogućih odstupanja:**
 - 2a. Uneseno krivo korisničko ime
 1. Sustav javlja grešku
 - 2b. Korisnik nije popunio polje za lozinku i/ili korisničko ime
 1. Sustav javlja grešku
 - 2c. Korisnik je unio pogrešnu lozinku
 2. Sustav javlja grešku

UC3 – Odjava iz sustava

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Odjava korisnika iz sustava
- **Sudionici:** -
- **Preduvjet:**
 1. Korisnik je prijavljen u sustav
- **Opis osnovnog tijeka:**
 1. Korisnik odabire opciju odjave
 2. Korisnik je odjavljen iz sustava i preusmjeren na početnu stranicu

UC4 – Brisanje korisničkog računa

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Obrisati korisnički račun
- **Sudionici:** Baza podataka
- **Preduvjet:**
 1. Korisnik je prijavljen u sustav
- **Opis osnovnog tijeka:**
 1. Korisnik odabire opciju „Obriši korisnički račun“
 2. Korisniku se otvara poruka „Jeste li sigurni?“
 3. Korisnik odabire opciju „Da“
 4. Korisnički račun se briše iz baze podataka

UC5 – Uređivanje korisničkog računa

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Uređivanje korisničkog računa
- **Sudionici:** Baza podataka
- **Preduvjet:**
 1. Korisnik je registriran u sustav
 2. Korisnik je odabrao „Profil“
 3. Korisnik je odabrao „Uredi profil“
- **Opis osnovnog tijeka:**
 1. Korisnik odabire opciju uređivanja profila
 2. Korisniku se prikazu mogućnosti uređivanja korisničkog imena, lozinke i slike
 3. Korisnik bira željenu opciju koju želi mijenjati/unosi novo korisničko ime i/ili lozinku i/ili podatke vezane uz igricu PokemonGo
 4. Novo uneseni podaci se ažuriraju u bazi podataka
- **Opis mogućih odstupanja:**
 - 2a. Uneseno novo korisničko ime već postoji
 2. Sustav javlja grešku
 - 2b. Unesena lozinka je kraća od 8 znakova
 2. Sustav javlja grešku

UC6 – Pregled trenutno aktivnih oglasa za raid

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Pregledavanje trenutno aktivnih oglasa za raid
- **Sudionici:** Baza podataka
- **Preduvjet:**
 1. Korisnik je prijavljen u sustav
- **Opis osnovnog tijeka:**
 1. Korisniku se na početnoj stranici prikaže popis trenutno aktivnih raidova

UC7 – Pregled pojedinog raida

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Pregledavanje detalja pojedinog raida
- **Sudionici:** Baza podataka
- **Preduvjet:**
 1. Korisnik je registriran u sustav
- **Opis osnovnog tijeka:**
 1. Korisnik odabire raid na početnoj stranici za koji ga zanimaju detalji
 2. Korisniku se za odabrani raid prikaže slika pokemona te popis igrača koji sudjeluju u borbi te vrijeme početka raida

UC8 – Pristup Željenom raidu

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Pristupanje Željenoj borbi (raidu)
- **Sudionici:** Baza podataka
- **Preduvjet:**
 1. Korisnik je registriran u sustav
- **Opis osnovnog tijeka:**
 1. Korisnik odabire Željeni raid
 2. Korisnik odabire opciju pristupanja raidu
 3. Korisniku se po završetku raida ažurira statistika i preusmjerava se na početnu stranicu
- **Opis mogućih odstupanja:**
 - 2a. Raid je već završio – nemogućnost pristupa raidu
 1. Preusmjeravanje na početnu stranicu
 - 2a. Raid je već popunjen – nemogućnost pristupa raidu
 1. Sustav javlja grešku

UC9 – Odustajanje od raida

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Odustajanje od raida
- **Sudionici:** -
- **Preduvjet:**
 1. Korisnik je registriran u sustav
 2. Korisnik sudjeluje u aktivnom raidu
- **Opis osnovnog tijeka:**
 1. Korisnik odabire opciju odustajanja
 2. Korisniku se otvara poruka: „Jeste li sigurni da želite odustati?“
 3. Korisnik odabire opciju „Da“
 4. Korisnika se izbacuje iz raida i preusmjerava na početnu stranicu

UC10 – Stvaranje novog raida

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Stvaranje novog raida
- **Sudionici:** Baza podataka
- **Preduvjet:**
 1. Korisnik je registriran u sustav
- **Opis osnovnog tijeka:**
 1. Korisnik odabire opciju „Kreiraj novi raid“
 2. Korisnik unosi podatke o raidu(...)
 3. Novi raid se zapisuje u bazu podataka
- **Opis mogućih odstupanja:**
 - 2a. Korisnik nije popunio sva obavezna polja s podacima o raidu
 1. Sustav javlja grešku

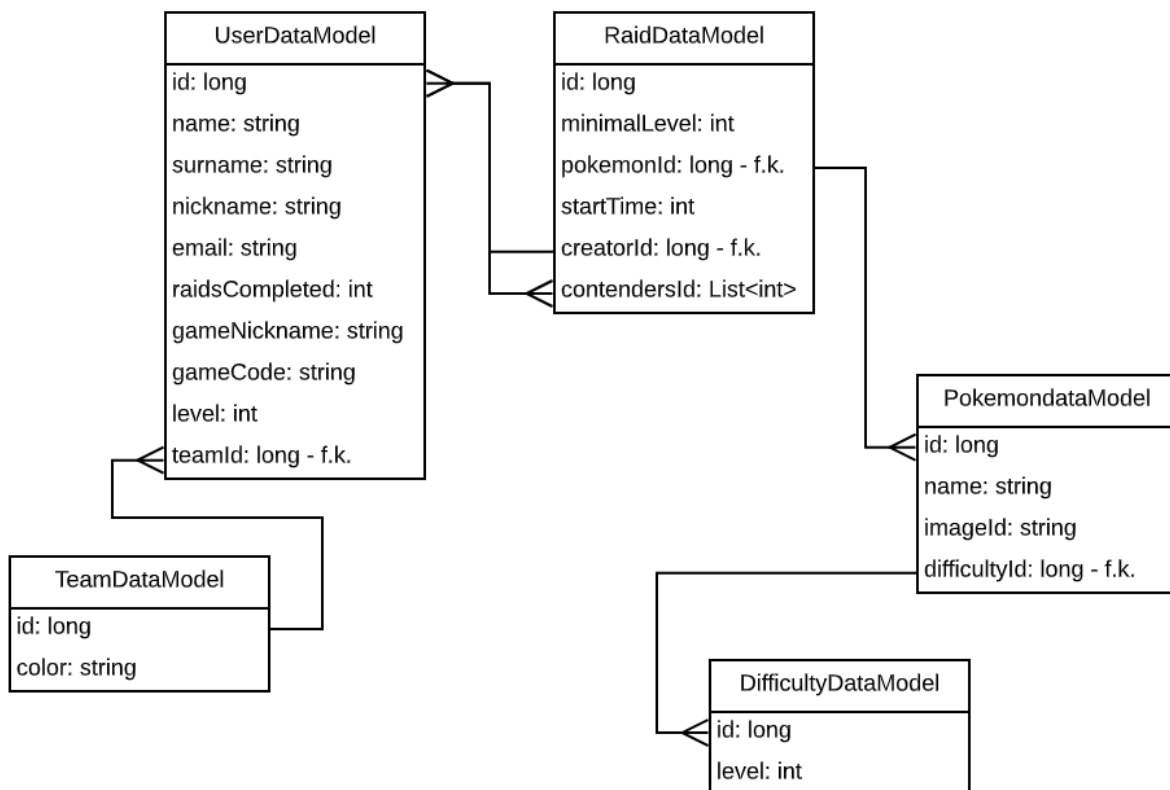
UC11 – Brisanje stvorenog raida

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Brisanje kreiranog raida
- **Sudionici:** Baza podataka
- **Preduvjet:**
 1. Korisnik je registriran u sustav
 2. Korisnik je sam kreirao raid koji želi obrisati
- **Opis osnovnog tijeka:**
 1. Korisnik odabire opciju brisanja stvorenog raida
 2. Korisniku se otvara poruka: „Jeste li sigurni da želite obrisati ovaj raid?“
 3. Korisnik odabire opciju „Da“
 4. Odabrani raid se briše iz baze podataka i korisnik je preusmjeren na početnu stranicu

UC12 – Prikaz najaktivnijih igrača

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Pregled najaktivnijih igrača
- **Sudionici:** Baza podataka
- **Preduvjet:**
 1. Korisnik je prijavljen u sustav
- **Opis osnovnog tijeka:**
 1. Korisnik odabire opciju prikaza najaktivnijih igrača
 2. Korisnik se prikazuje lista najaktivnijih igrača

3. Opis baze podataka



Slika 3.1 Dijagram razreda modela domene

Baza podataka implementirana je SQLite bazom podataka i sadrži 5 tablica kao što je prikazano na slici 3.1.

Tablica `UserDataModel` sadrži sve informacije o korisniku. Svakom registriranom korisniku dodijeljen je jedinstveni id koji ga razlikuje od ostalih korisnika. Prilikom registracije korisnik unosi ime i prezime, username koji želi koristiti u aplikaciji, email koji će mu biti sredstvo autentifikacije, password, nadimak koji koristi u igrici *Pokemon Go*, friendship code koji je kasnije prezentiran drugim igračima kako bi se mogli međusobno povezati, level unutar igrice *Pokemon Go* i tim kojem pripada. Moguće je izabrati jedan od 3 osnovna tima: plavi, crveni i žuti koji su pohranjeni u tablici `TeamDataModel`.

Tablica `RaidDataModel` sadrži sve kreirane raidove. Svaki registrirani korisnik kroz aplikaciju može kreirati raid tako da unese koji je minimalni level za pristup raidu te odabere pokemona protiv kojega će se boriti. Svi trenutno dostupni pokemoni nalaze se u tablici `PokemonDataModel`, dok se sve težine pokemona nalaze u tablici `DifficultyDataModel`. Svaki Raid sadržava jedan model korisnika koji je kreator raida, i više korisnika koji se mogu priključiti Raidu. Svaki korisnik može biti član jednog tima, svaki pokemon može imati samo jednu težinu i svaki raid može imati samo jednog pokemona.

4. Opis izgrađene desktop aplikacije

Prezentacijski sloj (Presentational Layer) je napravljen koristeći Mode-View-Presenter pattern čime je prezentacijski sloj (View) pasivan i ne sadrži nikakvu poslovnu logiku. Prezentacijski sloj sadrži forme koje korisniku prikazuju podatke i od njega primaju akcije.

Forme prezentacijskog sloja sve korisnikove akcije prosljeđuju kontroleru koji manipulira podacima i vraća nove forme za određene akcije.

Komunikacija između APIja i formi odvija se koristeći HttpClient, odnosno baznu klasu za slanje i primanje HTTP zahtjeva od izvora koji je specificiran URLjem. HttpClient obuhvaća adresu usluge koja je definirana u API. Kao odgovor na Http zahtjev dobiva se odgovor, tj. HttpResponseMessage nad kojim se može odgovor dobiti kao string.

Primjer Get zahtjeva, za dohvaćanje usera, odnosno korisnika i njegovih podataka. Prvo se kreira objekt HttpClient i definira se URI usluge za dohvaćanje korisnika, zatim se odgovor dobiva kao objekt HttpResponseMessage koji sadrži status i podatke. Zatim se dobiveni podaci, tj. sadržaj odgovora (Content) serijalizira kao asinkrona operacija. Nakon toga se na formi ispisuju podaci o korisniku:

```
private void btnShowAboutClick(object sender, EventArgs e)
{
    HttpClient client = new HttpClient();

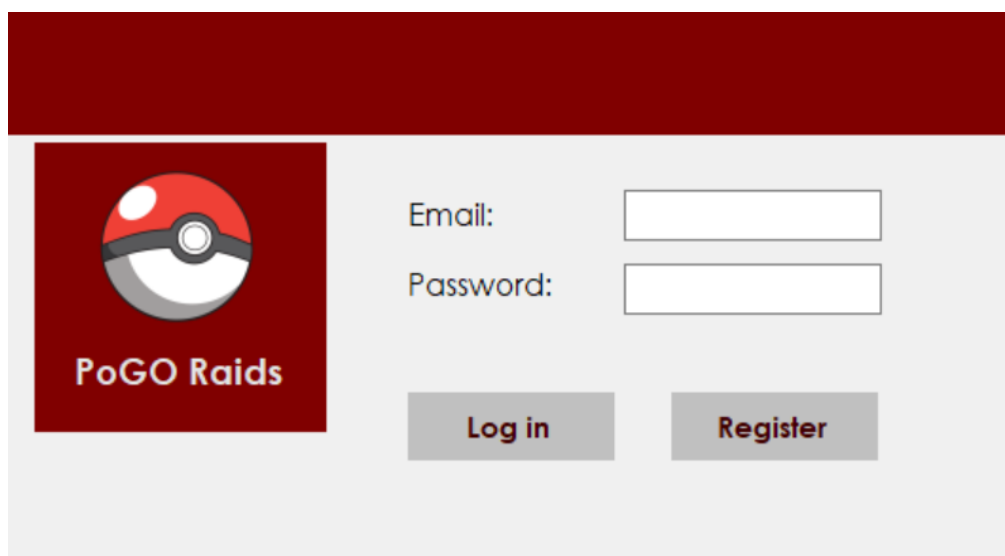
    client.BaseAddress = new Uri("https://localhost:44390/");

    HttpResponseMessage response = client.GetAsync("User/active").Result;

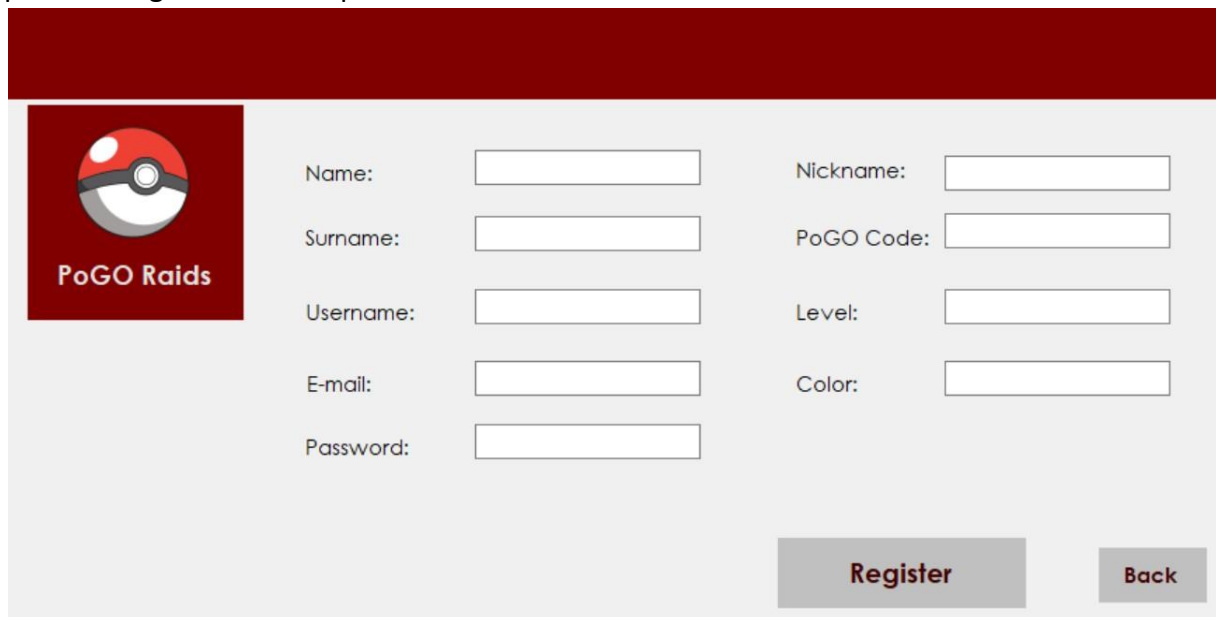
    var user = response.Content.ReadAsAsync<IEnumerable<UserDOM>>().Result;

    userInfo.DataSource = user;
}
```


Početni zaslon prikazuje mogućnosti prijave u sustav ili registracije i kreiranja novog računa:



Pritiskom na gumb Register korisniku se otvara nova forma u kojoj upisuje podatke za registraciju u sustav, a također može i odustati od registracije pritiskom na gumb Back prilikom čega se vraća na početni zaslon:

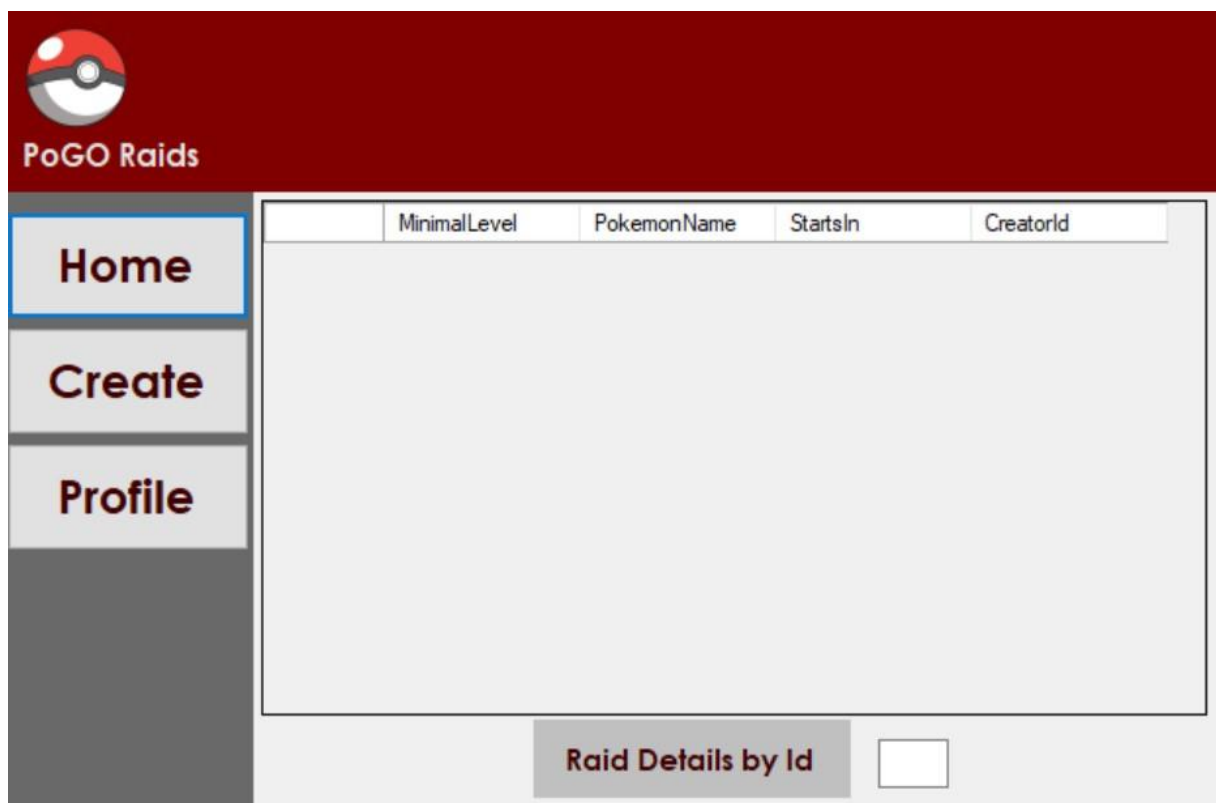


The registration form for PoGO Raids features a dark red header with a Poké Ball icon and the text "PoGO Raids". The form itself is light gray and contains several input fields for user information. At the bottom right, there are two buttons: "Register" and "Back".

	Name:	<input type="text"/>	Nickname:	<input type="text"/>
	Surname:	<input type="text"/>	PoGO Code:	<input type="text"/>
	Username:	<input type="text"/>	Level:	<input type="text"/>
	E-mail:	<input type="text"/>	Color:	<input type="text"/>
	Password:	<input type="text"/>		

Register **Back**

Nakon uspješne registracije i nakon uspješne prijave u sustav korisniku se prikazuje sljedeća forma u sustavu koja je Home forma. Pritiskom na gumb Home korisniku se prikazuju svi trenutno aktivni raidovi u sustavu, a također ima mogućnost gledati detalje pojedinog raida ako upiše id onog koji ga zanima i pritisne gumb za prikaz detalja:

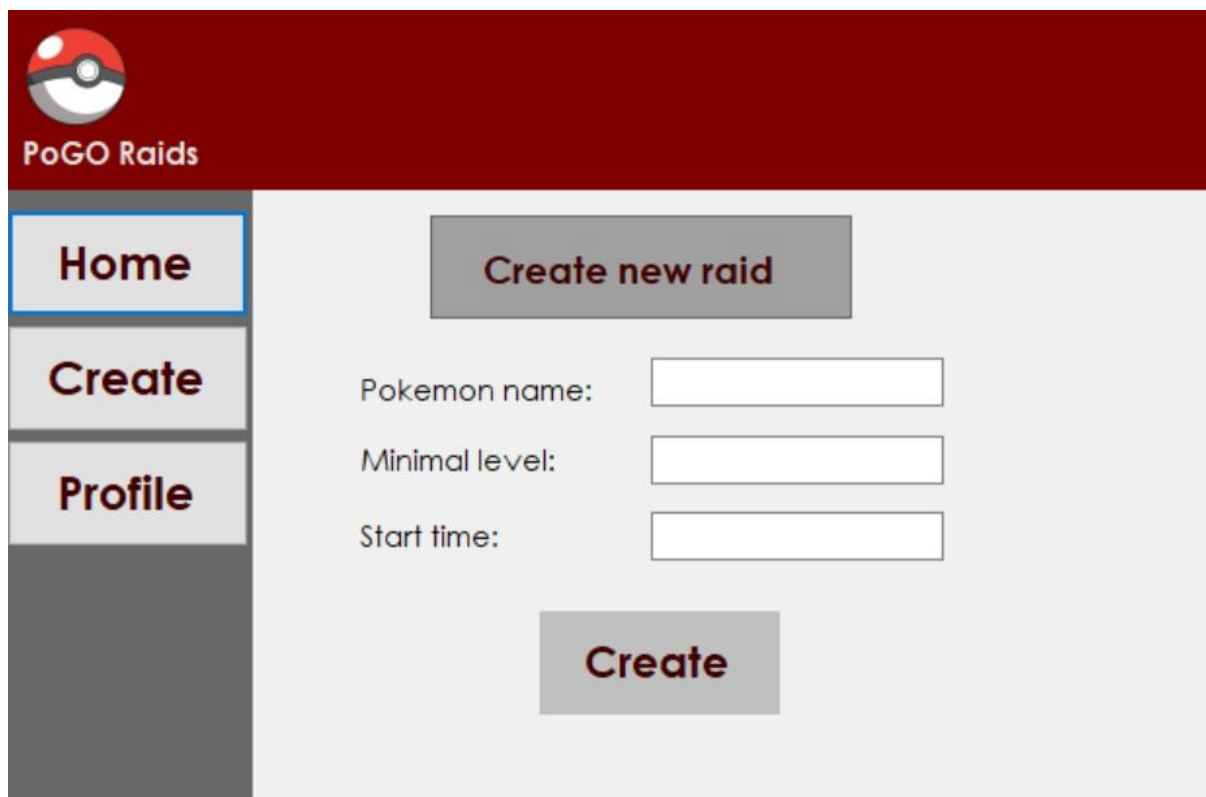


The Home screen of PoGO Raids has a dark red header with the Poké Ball icon and "PoGO Raids" text. On the left is a vertical sidebar with three buttons: "Home" (highlighted with a blue border), "Create", and "Profile". The main area contains a table with four columns: "MinimalLevel", "PokemonName", "StartsIn", and "CreatorId". Below the table is a button labeled "Raid Details by Id" followed by an empty input field.

MinimalLevel	PokemonName	StartsIn	CreatorId
--------------	-------------	----------	-----------

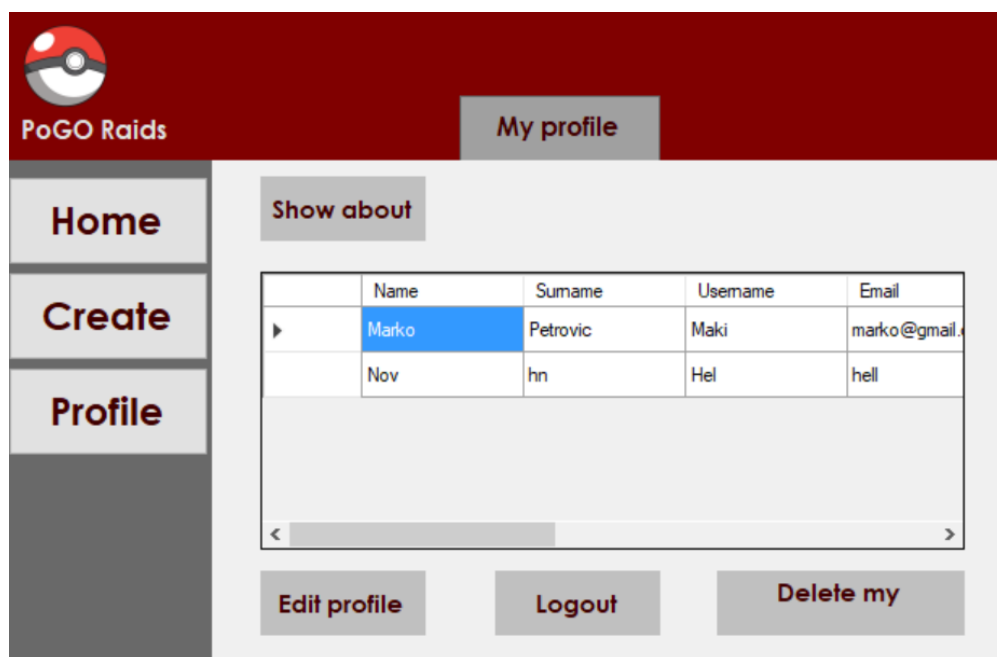
Raid Details by Id

Gumb Create vraća novu formu koja omogućuje kreiranje raida, korisnik treba upisati potrebne podatke o raidu prilikom čega se novi raid kreira, a korisnik se vraća na početnu stranicu Home:



The screenshot shows the 'PoGO Raids' application interface. On the left is a sidebar with three buttons: 'Home' (highlighted with a blue border), 'Create', and 'Profile'. The main content area has a red header with the PoGO logo and the text 'PoGO Raids'. Below the header, there is a 'Create new raid' button. Underneath this button are three input fields labeled 'Pokemon name:', 'Minimal level:', and 'Start time:'. At the bottom of the main area is a large 'Create' button.

Gumb profil nudi prikaz vlastitog profila gdje je moguće vidjeti podatke o korisniku. Osim toga podatke je moguće i uređivati (Edit profile), odjaviti se iz sustava (Logout) ili izbrisati vlastiti korisnički račun (Delete):




The screenshot shows the 'My profile' page in the 'PoGO Raids' application. The sidebar on the left has three buttons: 'Home', 'Create', and 'Profile' (highlighted with a blue border). The main content area has a red header with the PoGO logo and the text 'PoGO Raids'. Below the header, there is a 'My profile' button. Underneath this button is a 'Show about' button. Below 'Show about' is a table with the following data:

	Name	Surname	Username	Email
▶	Marko	Petrovic	Maki	marko@gmail.com
	Nov	hn	Hel	hell

Below the table is a horizontal scrollbar. At the bottom of the main area are three buttons: 'Edit profile', 'Logout', and 'Delete my'.

Prikaz forme za izmjenu podataka o korisničkom računu:



PoGO Raids

Home

Create

Profile

Edit my profile

Name:	<input type="text"/>	Surname:	<input type="text"/>
E-mail:	<input type="text"/>	Password:	<input type="text"/>
Level:	<input type="text"/>	PoGO Code:	<input type="text"/>
Nickname:	<input type="text"/>		

Save changes

Logout

Delete my

5. Opis izgrađene iOS mobilne aplikacije

iOS aplikacija izgrađena je korištenjem dependency injection-a i implementacijom *Clean* arhitekture. Zbog toga što iOS ne nudi *dependency injection* kao dio *base library*-a, za *dependency injection* se koristio *Resolver library*. *Resolver library* nudi brojne načine registracije u *dependency container*. Za domenski i poslovni sloj korišten je *application scope*, što označava da se objekt registrira kao *Singleton* i njegova instanca postoji dok je aplikacija živa. Objekti prezentacijskog sloja registrirani su sa *unique scope*-om, što znači da se svaki put kreira nova *instanca klase*. Registracija se odvija u *AppModule* klasi, dok se dohvaćanje objekta iz *dependency container*-a izvršava u *AppRouter* klasi prilikom instanciranja **ViewController* klase. Primjer registracije i dohvaćanja objekata može se vidjeti na slici 5.1.

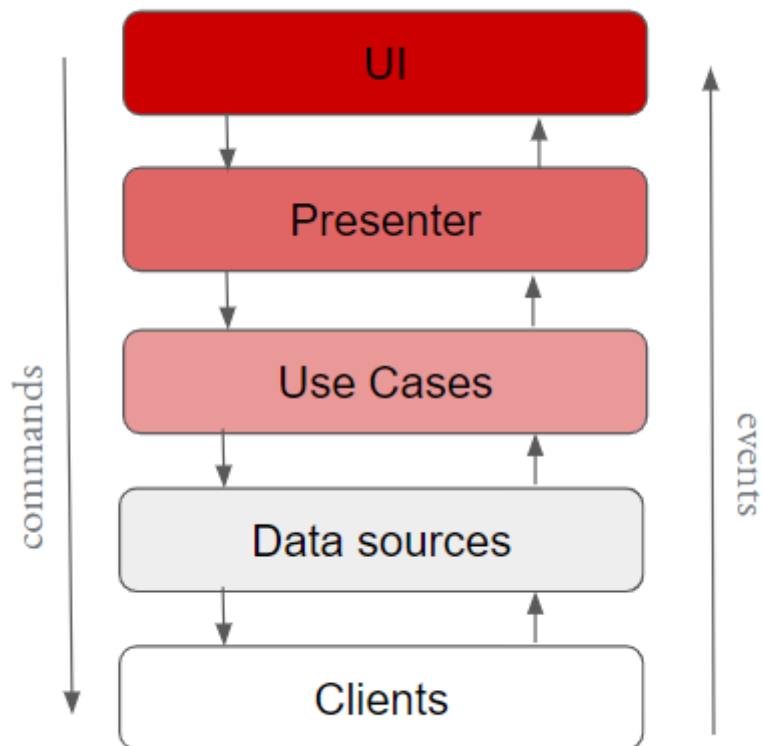
```
private func registerPresenters(in container: Resolver) {~
    ~~~container~
    ~~~~~register { LoginPresenter(router: container.resolve(), useCase: container.resolve()) }~
    ~~~~~scope(Resolver.unique)~
    ~~~container~
    ~~~~~register { RegisterPresenter(router: container.resolve(), userUseCase: container.resolve(), teamUseCase: container.resolve()) }~
    ~~~~~scope(Resolver.unique)~
    ~~~container~
    ~~~~~register { ProfileDetailsPresenter(router: container.resolve(), userUseCase: container.resolve()) }~
    ~~~~~scope(Resolver.unique)~
    ~~~container~
    ~~~~~register { CreateRaidPresenter(router: container.resolve(), useCase: container.resolve()) }~
    ~~~~~scope(Resolver.unique)~
    ~~~container~
    ~~~~~register { (_, arg) -> RaidsDetailsPresenter? in~
    ~~~~~guard let raidId = arg as? Int else { return nil }~
    ~~~~~return RaidsDetailsPresenter(router: container.resolve(), raidUseCase: container.resolve(), raidId: raidId)~
    ~~~~~}~
    ~~~~~scope(Resolver.unique)~
    ~~~container~
    ~~~~~register { RaidsPresenter(router: container.resolve(), raidUseCase: container.resolve()) }~
    ~~~~~scope(Resolver.unique)~
}~

func showRaidDetailsScreen(raidId: Int) {~
    ~~~let controller = container.resolve(RaidsDetailsViewController.self, args: raidId)~
    ~~~navigationController.pushViewController(controller, animated: true)~
}~
```

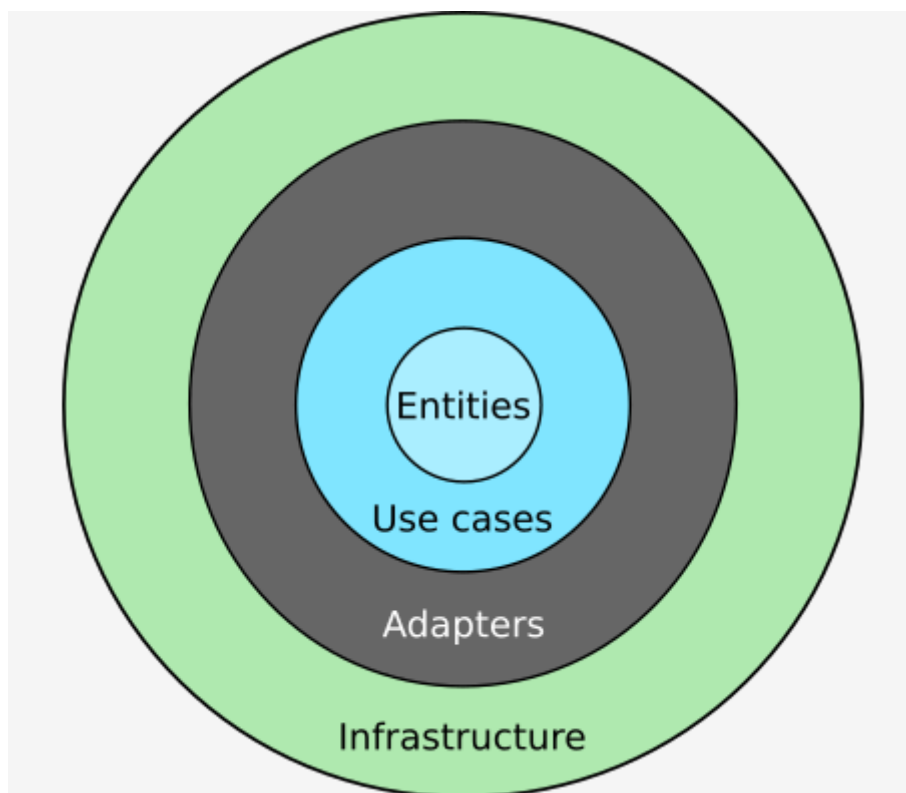
Slika 5.1.

Domenski sloj sastoji se od **Client* i **DataSource* klase. One su zadužene za spajanje na REST api i prosljeđivanje podataka prema poslovnom sloju. Za komunikaciju sa REST api-em korištena je *BaseApiClient* klasa koja je nadogradnja na *Alamofire library*. Poslovni sloj sastoji se od **UseCase* klase. U njemu se obrađuju podaci iz različitih **DataSource* objekata i prosljeđuju se prezentacijskom sloju. Prezentacijski sloj kreiran je koristeći *Model-View-Presenter pattern*-a, kojem je glavni fokus da *ViewController (view)* sadrži samo logiku za iscrtavanje na ekran. Ostala logika trebala bi biti smještena u **Presenter* klasi. To nam omogućava da prilikom promjene biznis logike nemamo gotovo nikakve promjene u **ViewController* klasi.

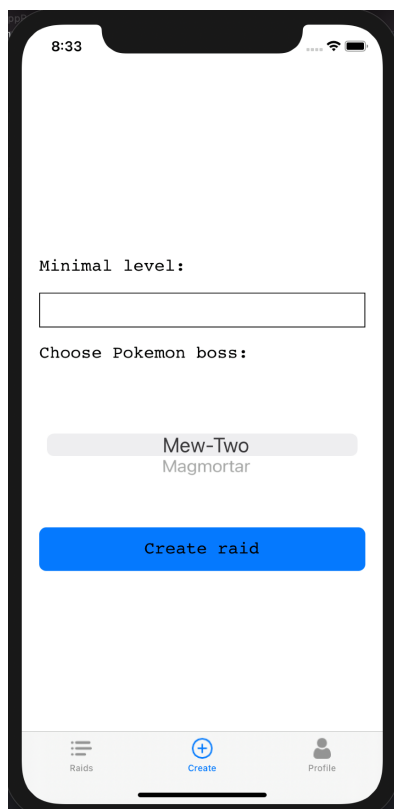
Slojevi međusobno smiju komunicirati samo izvana prema središtu, odnosno klase u prezentacijskom sloju smiju imati reference na klase iz poslovnog sloja, ali ne i suprotno. Uz to, komunikacija se smije događati samo između dva susjedna sloja. Prikaz arhitekture na slici 5.2. Svaka klasa u poslovnom i domenskom sloju sadrži svoj interface što omogućava jednostavno testiranje i *mock*-anje podataka.



Slika 5.2. iOS arhitektura aplikacije



Slika 5.3. Clean arhitektura



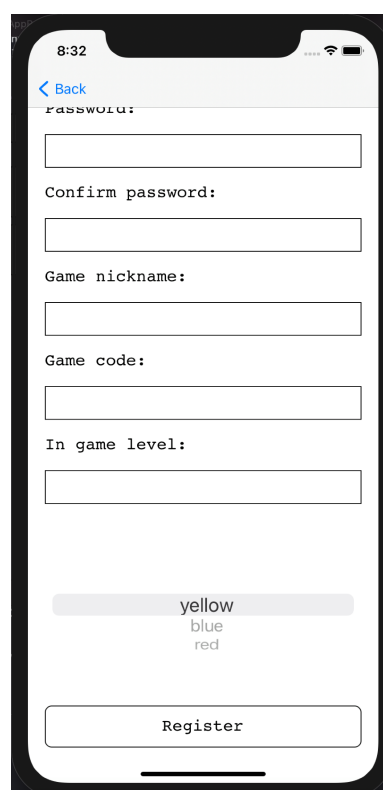
Slika 5.4. Kreiranje raida



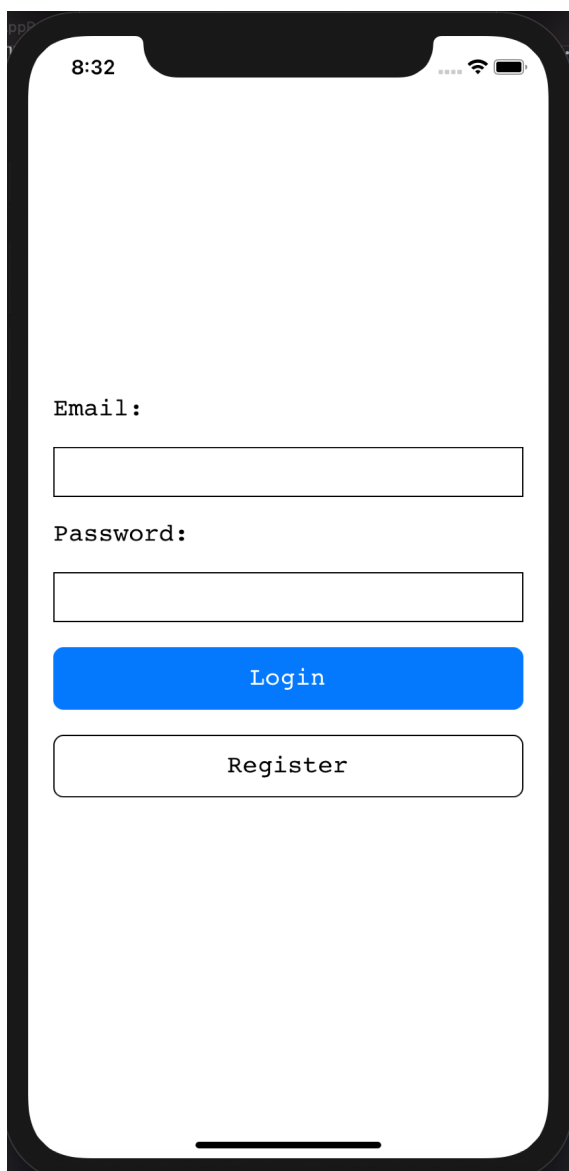
Slika 5.5. Pregled aktivnih raidova



Slika 5.6. Pregled profila



Slika 5.7. Registracija korisnika



Slika 5.8. Login ekran



Slika 5.9. Pregled detalja Raida

6. Opis izgrađenog backend dijela sustava

Struktura *backend* projekta podijeljena je u tri dijela: *Controllers*, *Services* i *Repositories*.

Domenski sloj se sastoji od repozitorija koji služe za komunikaciju s bazom podataka i u njima su smještene sve *Create*, *Read*, *Update* i *Delete* (CRUD) metode. Svaki domenski model ima svoj *repository*, npr. CRUD metode za *UserDataModel* su smještene u *UserRepository* klasi. Svaki *repository* implementira sučelje, što nam omogućava jednostavno i brzo testiranje. Primjer *repository* sučelja može se vidjeti na slici 6.1.

```
public interface IUserRepository
{
    3 references
    UserDataModel Get(long id);
    2 references
    IList<UserDataModel> GetAll();
    2 references
    UserDataModel Save(UserDataModel userModel);
    2 references
    UserDataModel Login(string password, string email);
    2 references
    void Delete(long id);
    2 references
    void Update(long id, string name, string surname, string username, string gameNickname, string email, string password, int level);
    2 references
    IList<UserDataModel> GetMostActive();
}
```

Slika 6.1. *IUserRepository*

Na domenski sloj veže se poslovni sloj, odnosno sloj u kojem se kombinira više repozitorija kako bi se pripremili podaci koji se šalju u REST api odgovoru. Primjer servisa koji koristi više repozitorija može se vidjeti na slici 6.2. U poslovnom sloju nalaze se servisi i njima pripadajuća sučelja.

```
2 references
public class UserService : IUserService
{
    private ITeamRepository TeamRepository;
    private IUserRepository UserRepository;

    0 references
    public UserService(ITeamRepository TeamRepository, IUserRepository UserRepository)
    {
        this.TeamRepository = TeamRepository;
        this.UserRepository = UserRepository;
    }

    2 references
    public UserModel Create(UserDOM model)
    {
        var team = TeamRepository.GetByColor(model.Color);
        var user = new UserDataModel { Name = model.Name, Email = model.Email, GameCode = model.GameCode, GameNickname = model.GameNickname, Level = model.Level,
        team.Members.Add(user);

        return new UserModel(UserRepository.Save(user));
    }
}
```

Slika 6.2. implementacija *UserService-a*

Prezentacijski sloj sastoji se od *controller-a* koji su *lightweight*, tj. ne sadrže nikakvu poslovnu logiku već služe samo za prezentiranje podataka. Ovakva arhitektura omogućava nam jednostavnije promjene u slučaju promjene dizajna sustava te *mocking* testiranje. Zbog toga što svaki *repository* i svaki servis implementiraju sučelje, jednostavno se mogu izgraditi *mocking* objekti koje bi koristili kod *mocking* testiranja.

Svi domain modeli smješteni su u *Domain* folder *PoGoRaids.Backend library-a*. Dok su njihovi O/R maperi smješteni u folderu *Mappings*.

Za konfiguraciju baze podataka korišten je *FluentNHibernate*. Inicijalizacija baze i vezivanje mappera napravljeno je u *DatabaseNHibernateHelper* klasi koja je u *dependency injection container*-u instancirana kao *Singleton*. *DatabaseNHibernateHelper* implementira *INHibernateHelper* sučelje, što nam omogućava jednostavnu promjenu konfiguracije baze u ovisnosti o okruženju kojem se nalazimo (*test, development, production*).

```
29 references
public class UserDataModel
{
    9 references
    public virtual long Id { get; protected set; }
    4 references
    public virtual string Name { get; set; }
    4 references
    public virtual string Surname { get; set; }
    4 references
    public virtual string Username { get; set; }
    4 references
    public virtual string Password { get; set; }
    5 references
    public virtual string Email { get; set; }
    7 references
    public virtual long RaidsCompleted { get; set; }
    4 references
    public virtual string GameNickname { get; set; }
    3 references
    public virtual string GameCode { get; set; }
    4 references
    public virtual int Level { get; set; }
    5 references
    public virtual TeamDataModel Team { get; set; }
    6 references
    public virtual IList<RaidDataModel> CreatedRaids { get; set; }
    8 references
    public virtual IList<RaidDataModel> ParticipatedRaids { get; set; }

    1 reference
    public UserDataModel()
    {
        CreatedRaids = new List<RaidDataModel>();
        ParticipatedRaids = new List<RaidDataModel>();
    }
}
```

Slika 6.3.Implementacija domenskog modela

```

class UserDataModelMap: ClassMap<UserDataModel>
{
    0 references
    public UserDataModelMap()
    {
        Id(x => x.Id);
        Map(x => x.Name);
        Map(x => x.Surname);
        Map(x => x.Password);
        Map(x => x.Email);
        Map(x => x.RaidsCompleted);
        Map(x => x.Username);
        Map(x => x.Level);
        Map(x => x.GameNickname);
        Map(x => x.GameCode);
        References(x => x.Team).Not.LazyLoad();
        HasMany(x => x.CreatedRaids).Not.LazyLoad().Inverse().Cascade.All();
        HasManyToMany(x => x.ParticipatedRaids).Not.LazyLoad().Inverse().Cascade.All().Table("UserRaid");
    }
}

```

Slika 6.4. Implementacija NHibernate mapera domenskog modela

API pozivi koji su korišteni u Windows forms i iOS aplikaciji:

- GET /user/login
 - koristi se za provjeru postojanja korisnika u bazi podataka
 - kao parametri poziva šalju se *password* i *email*
- POST /user/register
 - koristi se za registraciju novih korisnika
- GET /user/{id}
 - koristi se za dohvaćanje pojedinog korisnika
- PATCH /user/{id}
 - koristi se za izmjenu korisničkih podataka
- DELETE /user/{id}
 - koristi se za brisanje korisničkog računa
- GET /raid/all
 - dohvat svih aktivnih *raid*-ova
- POST /raid
 - kreiranje novog *raid*-a
- DELETE /raid/{raidId}/{userId}
 - koristi se za brisanje *raid*-a. Samo user koji je napravio *raid* ga može i obrisati
- GET /raid/{raidId}
 - koristi se za dohvaćanje pojedinog *raid*-a
- PATCH /raid/join/{raidId}/{userId}
 - koristi se za pristup korisnika *raid*-u
- PATCH /raid/leave/{raidId}/{userId}
 - koristi se kada korisnik želi odustati od *raid*-a
- GET /teams
 - dohvat svih postojećih timova
- GET /pokemons
 - dohvat svih aktivnih pokemona

Uz navedene pozive, moguće je napraviti sve CRUD metode, ali se one nisu implementirale u aplikacijama. Njih je moguće pozivati kroz *Postman* ili uz pomoć *terminal* skripti.