

Program obrazovanja za stjecanje mikrokvalifikacije web programiranje

GRUPA WP3

# Dokumentacija završne provjere

Ribolovni Dnevnik

https://github.com/scheissman/RibolovniDnevnik

https://scheissman-001-site1.ftempurl.com/

# **Zdenko Barić**

Osijek, svibanj 2024.

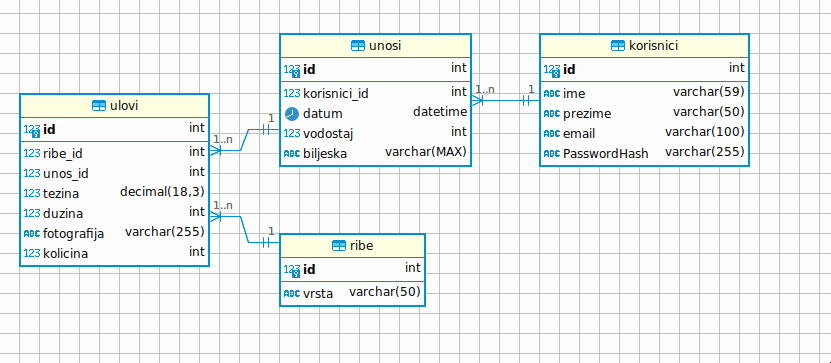
Sadržaj

[ERA dijagram (Vježba 1) 3](#_Toc148971343)

[Backend (Vježba 2) 4](#_Toc148971344)

[Frontend (Vježba 3 5](#_Toc148971345)

# **ERA dijagram (Vježba 1)**



ERA dijagram koristeći Crow's Foot shemu prikazuje četiri tablice: Korisnici, Unosi, Ulovi i Ribe.

\*\*Tipovi podataka:\*\*

- \*\*VARCHAR\*\*: Koristi se za tekstualne unose (stringovi), npr. VARCHAR(50) za unos do 50 znakova ili VARCHAR(MAX) za tekst bez ograničenja.

- \*\*INT\*\*: Za cijele brojeve.

- \*\*DECIMAL\*\*: Za decimalne brojeve, npr. DECIMAL(18,3) omogućuje unos do 18 brojeva ukupno, uključujući tri decimalna mjesta.

- \*\*DATETIME\*\*: Za unos datuma i vremena.

\*\*Podaci u tablicama:\*\*

- \*\*Korisnici\*\*: Ime, prezime, email, PasswordHash.

- \*\*Unosi\*\*: Korisnici\_id (vanjski ključ koji upućuje na tablicu Korisnici), datum, vodostaj, bilješka.

- \*\*Ulovi\*\*: Ribe\_id (vanjski ključ tablice Ribe), unos\_id (vanjski ključ tablice Unosi), težina, duljina, fotografija, količina.

- \*\*Ribe\*\*: Vrsta.

\*\*Odnos tablica:\*\*

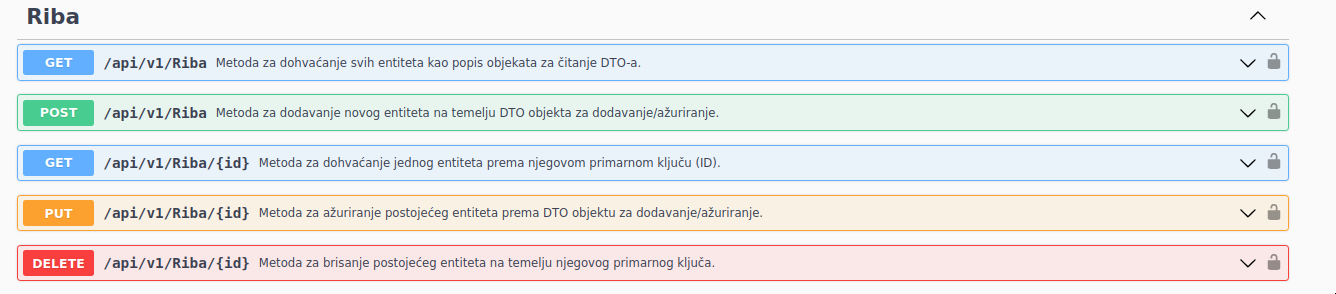
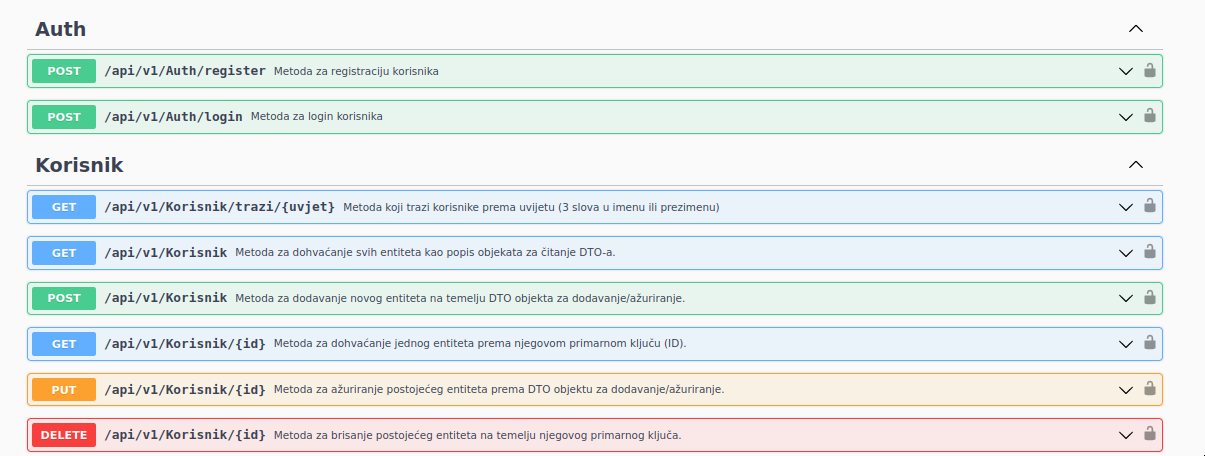
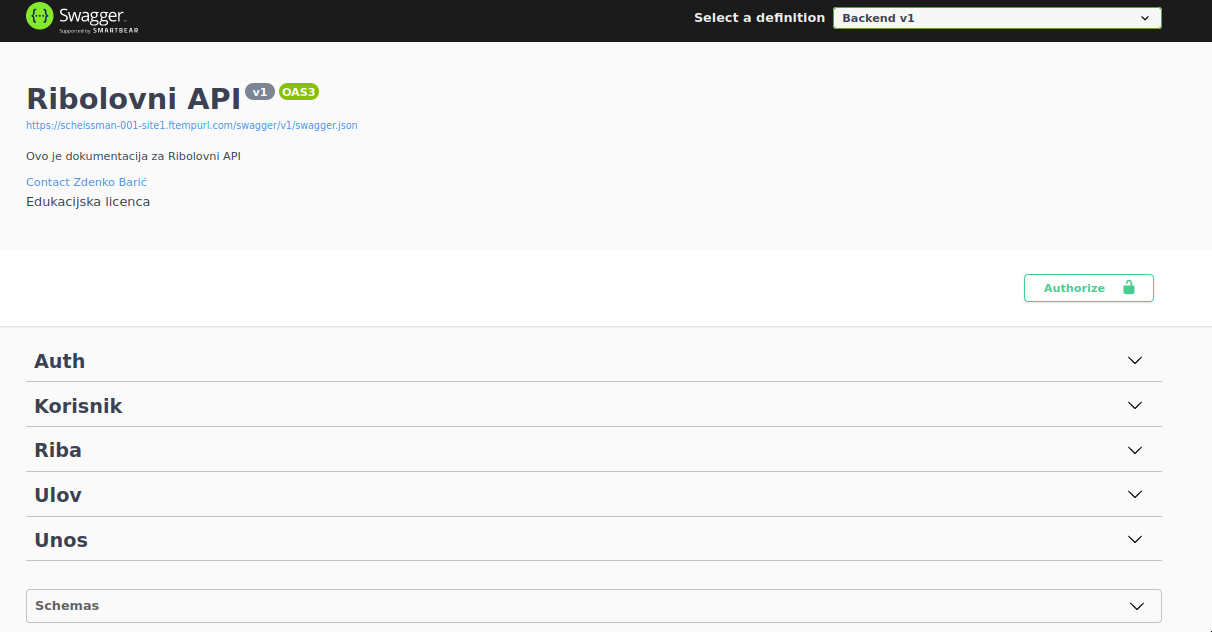
- \*\*Korisnici i Unosi\*\*: Omjer 1 prema više. korisnik\_id je vanjski ključ u tablici Unosi.

- \*\*Unosi i Ulovi\*\*: Omjer 1 prema više. unos\_id je vanjski ključ u tablici Ulovi.

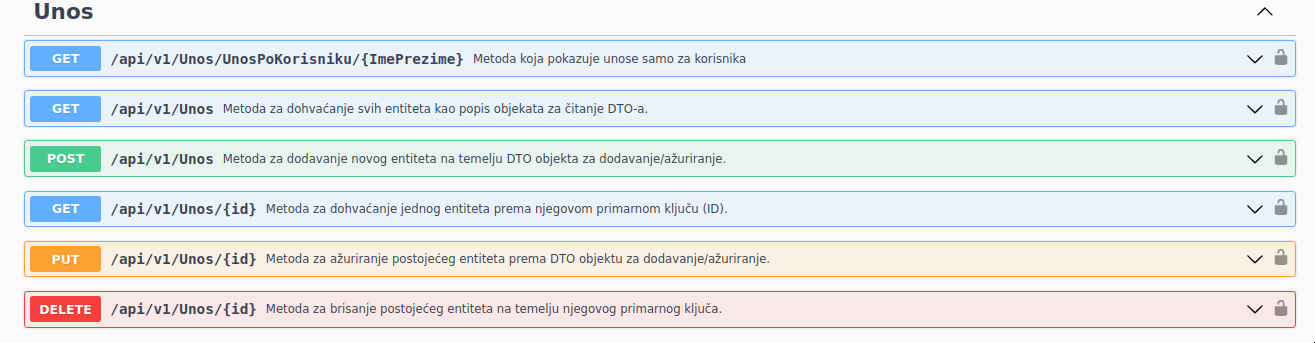
- \*\*Ribe i Ulovi\*\*: Omjer 1 prema više. ribe\_id je vanjski ključ u tablici Ulovi.

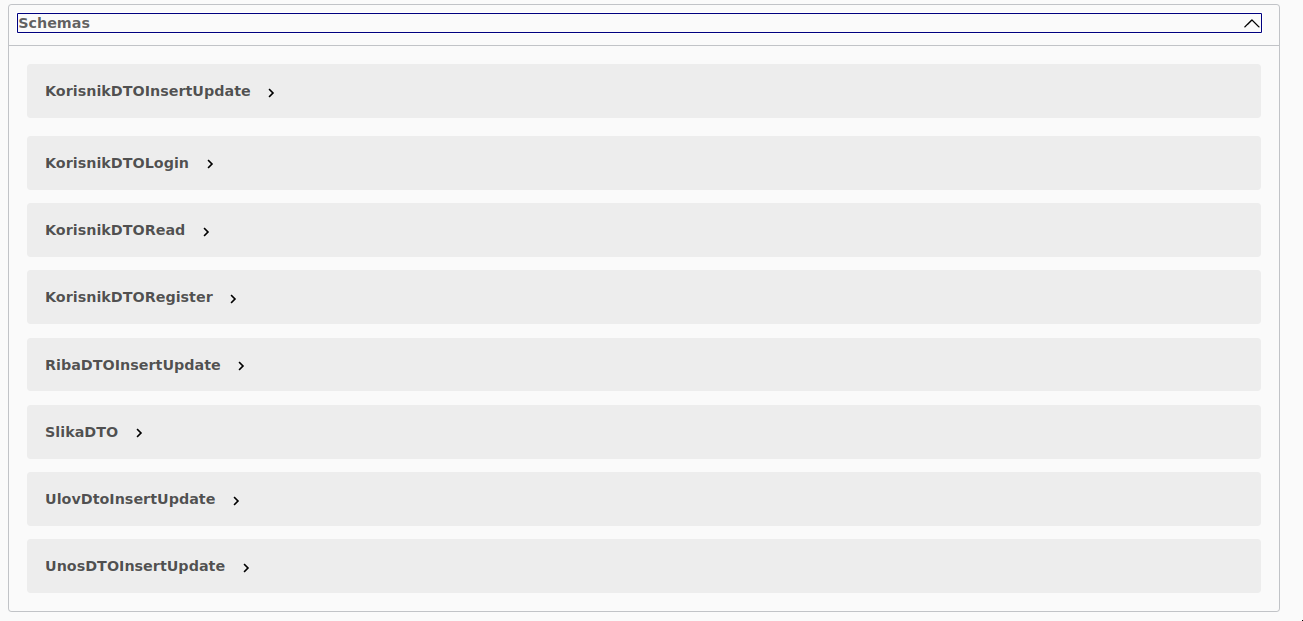
Tablica \*\*Ribe\*\* je POCO (Plain Old Class Object), služi samo za pohranu podataka, ne nasljeđuje posebne klase, i koristi standardne paradigme objektno-orijentiranog programiranja.

Backend (Vježba 2)



# 





Backend aplikacija se sastoji od pet kontrolera, koji su svi skriveni iza DTO biznis logike prema dobrim praksama programiranja u C# i .NET-u.

### `AuthController`

Ovaj kontroler upravlja registracijom i prijavom korisnika.

1. \*\*`Register` (metoda `POST /api/v1/auth/register`)\*\*:

- Omogućava registraciju novog korisnika.

- Prima podatke o korisniku (ime, prezime, email i lozinku) putem objekta `KorisnikDTORegister`.

- Lozinka se hashira koristeći `BCrypt` prije pohrane u bazu podataka.

- Stvara novi objekt `Korisnik` s primljenim podacima, uključujući hashiranu lozinku, i dodaje ga u bazu podataka.

- Nakon spremanja promjena u bazu podataka, vraća objekt korisnika.

2. \*\*`Login` (metoda `POST /api/v1/auth/login`)\*\*:

- Omogućava prijavu korisnika.

- Prima podatke o korisniku (email i lozinku) putem objekta `KorisnikDTOLogin`.

- Pronalazi korisnika u bazi podataka prema email adresi.

- Provjerava ispravnost unesene lozinke koristeći `BCrypt`.

- Ako je korisnik pronađen i lozinka ispravna, stvara JWT token za korisnika.

- Vraća odgovor koji uključuje JWT token i ID korisnika.

### `KorisnikController`

Ovaj kontroler upravlja CRUD operacijama (stvaranje, čitanje, ažuriranje, brisanje) za entitet `Korisnik` koristeći DTO klase `KorisnikDTORead` i `KorisnikDTOInsertUpdate`.

1. \*\*`TraziKorisnik` (metoda `GET /api/v1/korisnik/trazi/{uvjet}`)\*\*:

- Omogućava pretraživanje korisnika prema uvjetu (`uvjet`).

- Ako je uvjet prazan ili ima manje od 3 znaka, metoda vraća `BadRequest`.

- Traži korisnike kojima se ime ili prezime podudaraju s uvjetom.

- Rezultat pretraživanja mapira entitet `Korisnik` u DTO formate i vraća rezultat.

2. \*\*`Get` (metoda `GET /api/v1/korisnik`)\*\*:

- Vraća sve korisnike iz baze podataka.

- Ako nije pronađen nijedan korisnik, vraća `BadRequest`.

- Rezultati su mapirani u DTO oblik i vraćeni kao JSON.

3. \*\*`GetBySifra` (metoda `GET /api/v1/korisnik/{id}`)\*\*:

- Vraća korisnika prema ID-u (`id`).

- Provjerava ispravnost modela i ID-a.

- Ako je korisnik pronađen, rezultat se mapira u DTO oblik i vraća kao JSON.

4. \*\*`Post` (metoda `POST /api/v1/korisnik`)\*\*:

- Dodaje novog korisnika.

- Prima korisnički DTO objekt (`KorisnikDTOInsertUpdate`).

- Kreira entitet `Korisnik` iz DTO objekta i dodaje ga u bazu podataka.

- Rezultat se mapira u DTO oblik i vraća kao JSON sa statusom `201 Created`.

5. \*\*`Put` (metoda `PUT /api/v1/korisnik/{id}`)\*\*:

- Ažurira postojeći korisnika prema ID-u (`id`).

- Prima ID korisnika i DTO objekt (`KorisnikDTOInsertUpdate`).

- Pronalazi korisnika u bazi podataka, ažurira ga s novim podacima i sprema promjene.

- Rezultat se mapira u DTO oblik i vraća kao JSON sa statusom `200 OK`.

6. \*\*`Delete` (metoda `DELETE /api/v1/korisnik/{id}`)\*\*:

- Briše korisnika prema ID-u (`id`).

- Provjerava može li se korisnik obrisati i uklanja ga iz baze podataka.

- Uspješnim brisanjem vraća status `200 OK` s porukom "Obrisano".

### `RibaController`

Ovaj kontroler upravlja CRUD operacijama za entitet `Riba` koristeći DTO klase `RibaDTORead` i `RibaDTOInsertUpdate`.

1. \*\*`Get` (metoda `GET /api/v1/riba`)\*\*:

- Vraća sve ribe iz baze podataka.

- Ako nije pronađena nijedna riba, vraća `BadRequest`.

- Rezultati su mapirani u DTO oblik i vraćeni kao JSON.

2. \*\*`GetBySifra` (metoda `GET /api/v1/riba/{id}`)\*\*:

- Vraća ribu prema ID-u (`id`).

- Provjerava ispravnost modela i ID-a.

- Ako je riba pronađena, rezultat se mapira u DTO oblik i vraća kao JSON.

3. \*\*`Post` (metoda `POST /api/v1/riba`)\*\*:

- Dodaje novu ribu.

- Prima ribin DTO objekt (`RibaDTOInsertUpdate`).

- Kreira entitet `Riba` iz DTO objekta i dodaje ga u bazu podataka.

- Rezultat se mapira u DTO oblik i vraća kao JSON sa statusom `201 Created`.

4. \*\*`Put` (metoda `PUT /api/v1/riba/{id}`)\*\*:

- Ažurira postojeću ribu prema ID-u (`id`).

- Prima ID ribe i DTO objekt (`RibaDTOInsertUpdate`).

- Pronalazi ribu u bazi podataka, ažurira ju s novim podacima i sprema promjene.

- Rezultat se mapira u DTO oblik i vraća kao JSON sa statusom `200 OK`.

5. \*\*`Delete` (metoda `DELETE /api/v1/riba/{id}`)\*\*:

- Briše ribu prema ID-u (`id`).

- Provjerava može li se riba obrisati i uklanja je iz baze podataka.

- Uspješnim brisanjem vraća status `200 OK` s porukom "Obrisano".

### `UlovController`

Ovaj kontroler koristi sve metode za CRUD operacije i pruža dodatne metode specifične za entitet `Ulov`.

1. \*\*`Get`\*\*:

- Dohvaća sve instance entiteta `Ulov` iz baze podataka i vraća ih kao DTO objekte.

2. \*\*`GetBySifra`\*\*:

- Dohvaća pojedinačni entitet `Ulov` prema ID-u (`id`) iz baze podataka i vraća ga kao DTO objekt.

3. \*\*`Post`\*\*:

- Dodaje novi entitet `Ulov` u bazu podataka koristeći DTO podatke, kreira novi entitet i vraća ga kao DTO objekt.

4. \*\*`Put`\*\*:

- Ažurira postojeći entitet `Ulov` u bazi podataka prema ID-u (`id`) koristeći DTO podatke i vraća ažurirani DTO objekt.

5. \*\*`Delete`\*\*:

- Briše entitet `Ulov` iz baze podataka prema ID-u (`id`).

6. \*\*`GetUloviByUnosId`\*\*:

- Dohvaća ulove (ulov entitete) po ID-u unosa (`unosId`) i vraća ih kao DTO objekte.

- Omogućava dohvaćanje ulova prema unosa.

7. \*\*`DodajRibu`\*\*:

- Dodaje ribu u ulov prema ID-u ulova (`id`) i šifri ribe (`ribasifra`).

- Koristi ulov i ribu iz baze podataka za dodavanje ribe u ulov.

8. \*\*`PostaviSliku`\*\*:

- Postavlja sliku za ulov prema ID-u (`id`) koristeći Base64 enkodirani sadržaj slike.

- Pohranjuje sliku u lokalni direktorij.

9. \*\*`ObrisiRibu`\*\*:

- Uklanja ribu iz ulova prema ID-u ulova (`id`) i šifri ribe (`ribasifra`).

- Postavlja `Riba` polje ulova na `null` i ažurira ulov.

10. \*\*`UlovDodajSSlikom`\*\*:

- Dodaje novi ulov zajedno sa slikom.

- Koristi DTO podatke za kreiranje novog ulova, dodaje ga u bazu podataka i pohranjuje sliku u lokalni direktorij.

11. \*\*`UlovPromjeniSSlikom`\*\*:

- Ažurira postojeći ulov i njegovu sliku prema ID-u ulova (`ulovId`) koristeći DTO podatke.

- Ažurira ulov u bazi podataka i pohranjuje novu sliku u lokalni direktorij.

### `UnosController`

Ovaj kontroler koristi sve metode CRUD operacije za entitet `Unos`. Osim toga, `UnosController` pruža dodatne metode specifične za rad s entitetom `Unos`.

1. \*\*`Get`\*\*:

- Dohvaća sve instance entiteta `Unos` iz baze podataka i vraća ih kao DTO objekte.

2. \*\*`GetBySifra`\*\*:

- Dohvaća pojedinačni entitet `Unos` prema ID-u (`id`) iz baze podataka i vraća ga kao DTO objekt.

3. \*\*`Post`\*\*:

- Dodaje novi entitet `Unos` u bazu podataka koristeći DTO podatke, kreira novi entitet i vraća ga kao DTO objekt.

4. \*\*`Put`\*\*:

- Ažurira postojeći entitet `Unos` u bazi podataka prema ID-u (`id`) koristeći DTO podatke i vraća ažurirani DTO objekt.

5. \*\*`Delete`\*\*:

- Briše entitet `Unos` iz baze podataka prema ID-u (`id`).

6. \*\*`GetUnosiByImePrezime`\*\*:

- Dohvaća unose prema ID-u korisnika (`ImePrezime`) i vraća ih kao DTO objekte.

- Omogućava dohvaćanje unosa na temelju korisničkog ID-a.

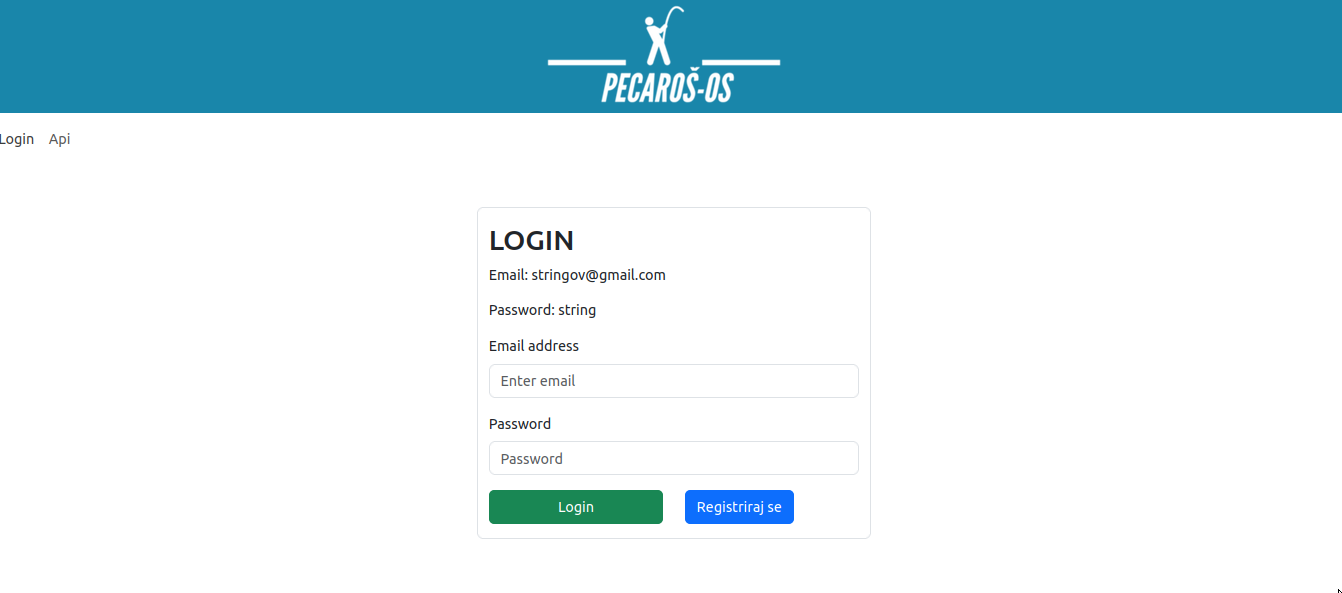
**Frontend (Vježba 3)**

Evo kako bi vaš tekst mogao izgledati nakon pregleda za sintaksne greške:

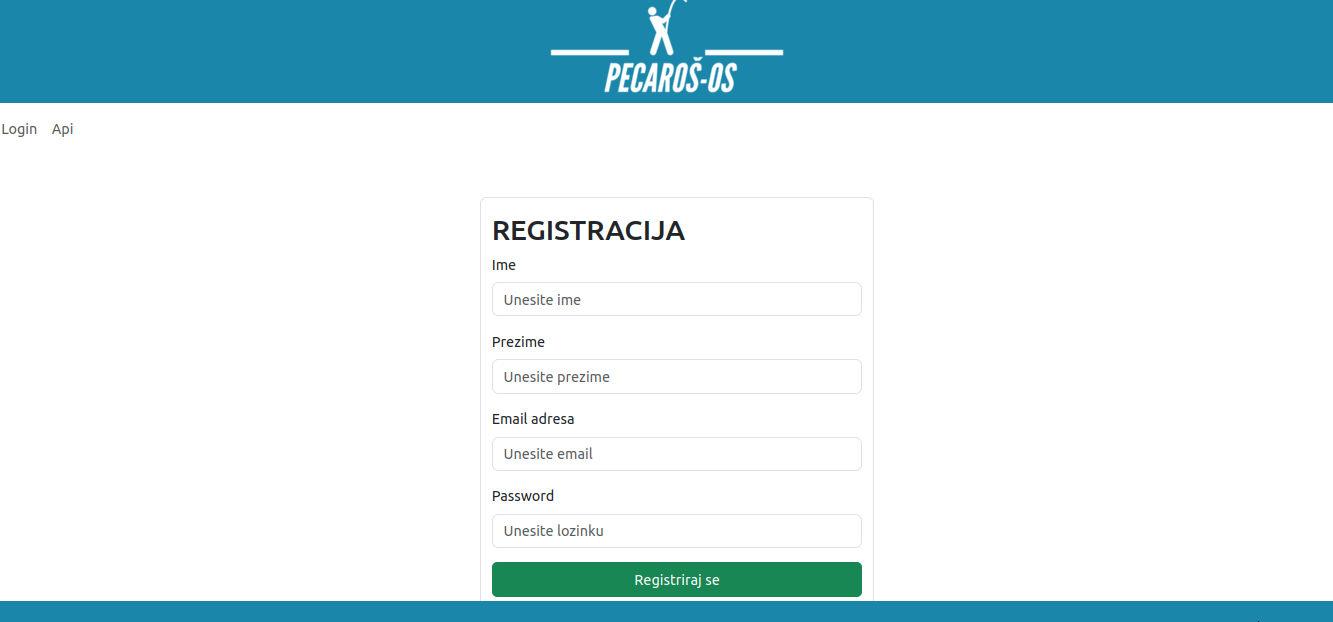
"Početni zaslon na kojem sam parsao i ubacio podatke s vremenske postaje te lunarnu prognozu koja predviđa ulov prema mjesečevim mijenama.

[https://www.fishingreminder.com/](https://www.fishingreminder.com/)

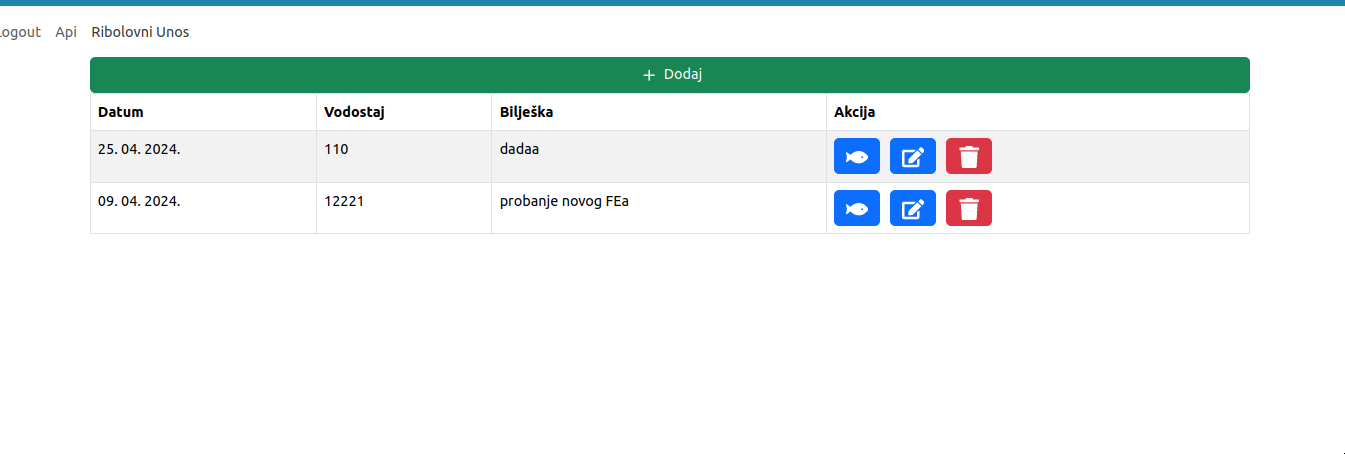
[https://forecast7.com/](https://forecast7.com/)"



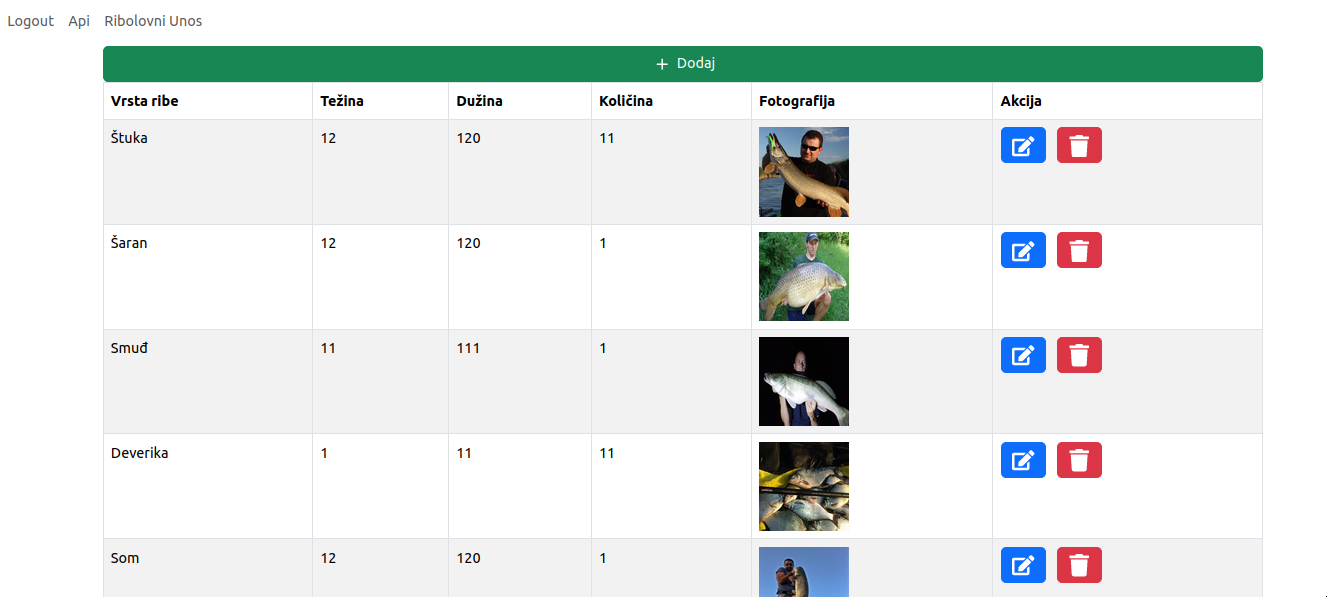
Korisnik se mora ulogirati da bi dobio funkcionalnost aplikacije tj pristup unosima i ulovima

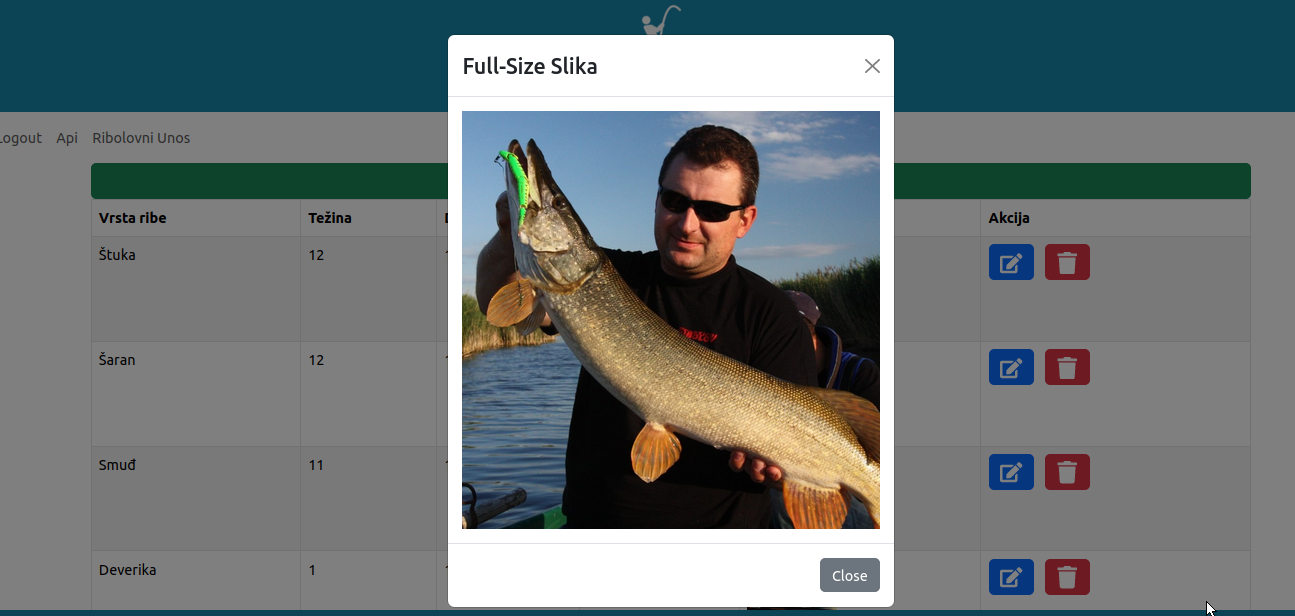


Ako korisnik ne postoji ima obrazac za registraciju

Nakon logiranja korisnik pristupa stranici unosi koja se sastoji od dodavanja novog unosa te Datuma, vodostaja i bilješke.

Na akcijama imamo simbol ribice gdje se dodaju ribolovni ulovi i mogućnost izmjene unosa i brisanje istog.

Na ribolovnim ulovima se unosi(+dodaj) vrsta ulovljene ribe, težina , dužina i količina te prikladna fotografija ulova



Na Ulovima je također moguće dodati , promijeniti ulov te isti izbrisati.

Na samom kraju imamo Pie chart koji pokazuje odnos ribolovnih vrsta prema broju ulova

