**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ORGANIZACIJE I INFORMATIKE**
**VARAŽDIN**

**Matej Šanko**

# APLIKACIJA ZA UPRAVLJANJE KORISNIČKIM RAČUNIMA

**PROJEKT**

**Varaždin, 2026.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Matej Šanko**

**Matični broj: 52940/22–R**

**Studij: Baze podataka i baze znanja**

# APLIKACIJA ZA UPRAVLJANJE KORISNIČKIM RAČUNIMA

**PROJEKT**

**Mentor:**

Dr. sc. socio Tomislav Peharda

**Varaždin, siječanj 2026.**

*Matej Šanko*

**Izjava o izvornosti**

Izjavljujem da je ovaj projekt izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI Radovi*

---

**Sažetak**

Rad prikazuje razvoj aplikacije za upravljanje korisnicim temeljen na objektno-relacijskom modelu baze podataka. Aplikacija omogućuje razdvajanje privilegija kroz mehanizme nasljeđivanja, automatizaciju logiranja pristupa putem triggera, te pohranjivanje metapodataka koristeći JSONB format. Aplikacija je realizirana sa Node.js okruženjem, za realiziranje baze podataka korisito se PostgreSQL unutar aplikaije pg Admin.

**Ključne riječi**: PostgreSQL, objektno-relacijski model, nasljeđivanje tablica, trigger, Node.js, JSONB

# Sadržaj

# 1. Opis aplikacijske domene

Koncept I domena aplikacije obuhvaća sustav za centralizirano upravljanje korisničkim računima unutar organizacije. Glavni cilj aplikacije je omogućiti jasnu razliku između običnih korisnika i administratora tj. osoba sa povišenim ovlastima. Uz automatsko praćenje prijava u sustav, kao primjer praćenja sigurnosnih događaja.

**Koncepti i relacije**

- **Korisnik** – osnovni entitet unutar baze podataka koji sadrži opće podatke (`id`, `ime_prezime`, `email`, `prebivalište`, `dodatni_info`, `datum_registracije`).
- **Administrator** – posebni entitet koji ima sve atribute korisnika (nasljeđuje ih od korisnika), ali sadrži i dodatne metapodatke (npr. razina ovlasti) koji služe za kontrolu sustava. U bazi podataka to je realizirano kroz objektno-relacijsko nasljeđivanje, čime se izbjegava dupliciranje podataka.
- **Sustav logova** – entitet koji zabilježava povijest pristupa aplikaciji. Svaki zapis je u odnosu s korisnikom $N:1$, što omogućuje detaljni pregled aktivnosti.
- **Fleksibilni metapodaci** – koristeći `JSONB` format, sustav omogućuje pohranu nespecifičnih postavki profila (npr. preferencija sučelja i sl.), što domenu čini prilagodljivom budućim promjenama.

Kao motivaciju za odabir tehnologije PostgreSQL odlučio sam se iz razloga što mislim kako je ta vrsta baza podataka najčešća u upotrebi te sam s njom najviše upoznat. Osim toga, sam PostgreSQL nudi podršku za Inherits tj. nasljeđivanje baze podataka, što nije slučaj kod svih SQL sustava za upravljanje bazama podataka. A to je bila odlična opcija za nasljeđivanje u mojoj bazi podataka na relaciji Korisnik -> Administrator. Također PostgreSQL-ov sustav triggera i funkcija mi je omogućio da se kritička logika tj. prijava u sustav, preseli u samu bazu, čime se osigurava integritet aplikacije neovisno o kodu aplikacije. Također integracija sa web sučeljem sa bazom podataka je bila intuitivna.

# 2. Teorijski uvod

Za implementaciju aplikacije sam koristio objektno-relacijski pristup bazama podataka sa PostgreSQL-om, te sam koristio koncepte aktivnih baza podataka (triggeri) i polustrukturiranih podataka (JSONB).

**Objektno-orjentacijski pristup**

Ovaj pristup predstavlja nadogradnju klasičnog relacijskog modela uvođenjem objektno-orijentacijskih koncepata. On omogućuje definirane korisničkih podataka, funkcija, te onoga što je potrebno za ovaj projekt, a to je nasljeđivanje tablica. Njegova prednost bi bila ta da smanjuje redundantnost podataka i omogućuje prirodnije strukturiranje objekata iz aplikacijskog koda u bazu.

Konkretno u projektu nasljeđivanje omogućuje da tablica Administrator automatski posjeduje sve stupce tablice Korisnik. Kao nedostatak ili mana bi bilo to što je sintaksa nešto drugačija od običnog relacijskog sustava poput MySQL-a, pa je potrebno pripaziti na te razlike u sintaksi. [1]

**Aktivne baze podataka**

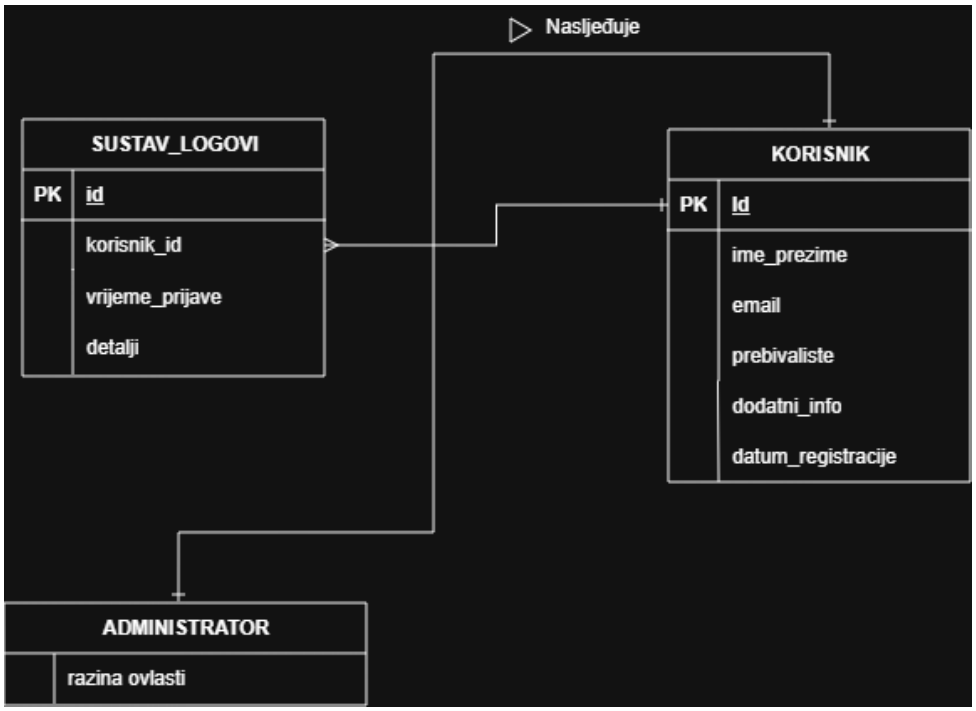
Predstavlja sposobnost sustava da automatski reagira na određene događaje unutar baze podataka bez vanjske intervencije aplikacije. Temelji se na mehanizmu ECA (engl. Event-Condition-Action) što je u projektu implementirano kroz triggere, koji na događaj prijave korisnika u aplikaciju automatski izvršavaju funkciju zapisa u log. Ovime se osigurava integritet podataka i poslovna logika na razini baze podataka, što garantira da će se log zapisati čak i ako aplikacijski sloj zakaže. Općenito prevelik broj triggera može usporiti bazu podataka, ali to u ovom projektu nije slučaj. [2] [3]

**Polustrukturirane baze podataka**

Ovaj pristup omogućuje pohranu podataka koji nemaju fiksnu i strogu shemu. U projektu to je napravljeno u PostgreSQL-u koji podržava JSONB format koji omogućuje pohranu dokumenta unutar relacijske tablice. U projektu, JSONB stupac dodatni_info omogućuje dodavanje novih atributa korisniku bez potrebe za mijenjanjem strukture cijele baze podataka. [4]

# 3. Model baze podataka

Model baze podataka (ERA) dizajniran je korištenjem objektno-relacijskog pristupa. Središnja točka sustava je hijerarhija korisničkih računa realizirana kroz nasljeđivanje tablica. Također na modelu nalazi i entitet Sustav_logovi koji služi za pregled prijava u aplikaciju. Prva relacija je relacija nasljeđivanja između Korisnika i Administratora, između kojih postoji odnos specijalizacije. Administrator nasljeđuje strukturu i podatke Korisnika, što miče potrebu za redundantnim stupcima. Druga relacija je relacija između Korisnika i Sustav_logovi. Između ova dva entiteta postoji relacija 1:N jer jedan korisnik može imati više zapisa bazi logova, dok svaki zapis tj. log pripada točno jednom korisniku.




Slika 1: ERA model baze podataka

# 4. Implementacija

Implementacija baze je izvedena u sustavu PostgreSQL pomoću pgAdmin aplikacije, dok je aplikacijski sloj razvijen u Node.js okruženju.

Implementacija objektno-relacijskog dijela odnosi se na nasljeđivanje koje je postignuto naredbom INHERITS. Kreiran je i složeni tip podatka lokacija_objekt za strukturiranu pohranu adrese prebivališta.

```
CREATE TYPE lokacija_objekt AS (
    ulica TEXT,
    grad TEXT
);
```

Isječak koda 1: Kreiranje složenog tipa podataka lokacija_objekt

Što se tiče aktivnih pravila, kreirana je funkcija log_prijava() i pripadajući joj trigger koji se aktivira prilikom svake uspješne prijave u sustav, automatski popunjajući tablicu sustav logovi. Primjer triggera:

```
CREATE TRIGGER trg_nakon_unosa_korisnik
AFTER INSERT ON Korisnik
FOR EACH ROW
EXECUTE FUNCTION funkcija_log_korisnik();
```

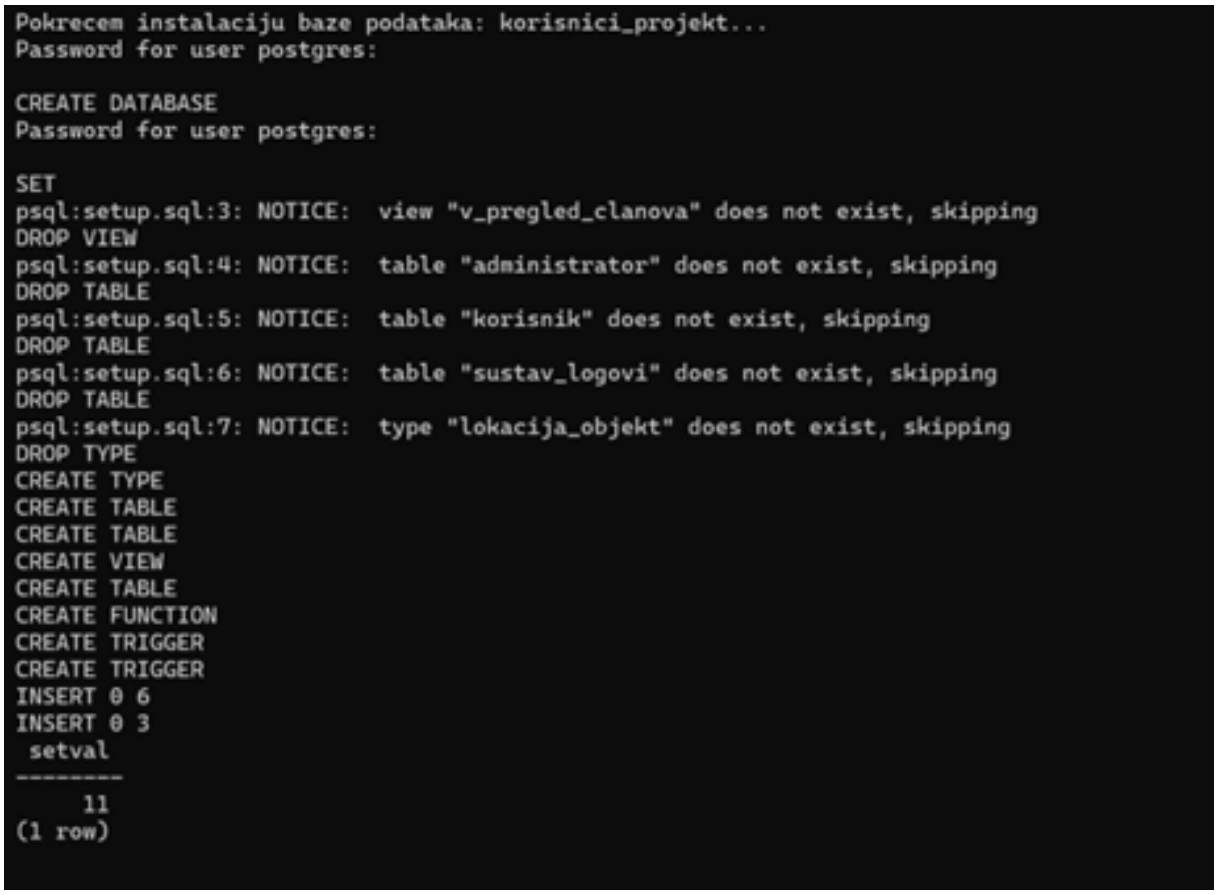
Isječak koda 2: Implementacija triggera za automatizirano logiranje

Kreirano je i atribut dodatni_info koji je tipa JSONB koji omogućava spremanje metapodataka bez fiksne sheme. A što se tiće povezivanja za komunikaciju između Node.js-a i baze podataka, korišten je modul pg koji omogućava asinkrone upite i sigurnu obradu podataka. [5]

Također, prikaz svih logova u aplikaciji možemo vidjeti upisivanjem SQL upita koji je prikazan u isječku koda 3.

```
SELECT * FROM sustav_logovi;
```

Isječak koda 3: Upit za dohvaćanje svih zapisa iz tablice logova

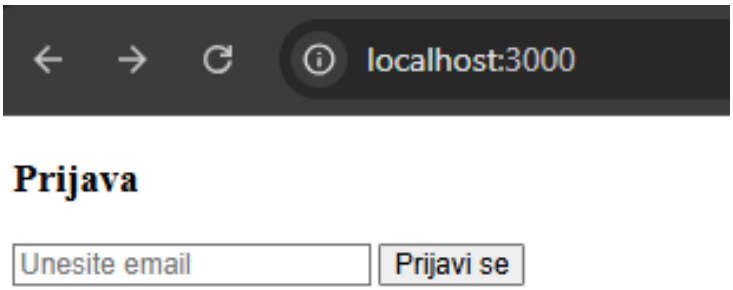



Slika 2: Uspješno izvršavanje batch skripte za automatiziranu instalaciju baze podataka i njezinih objekata. [6]

# 5. Primjer korištenja

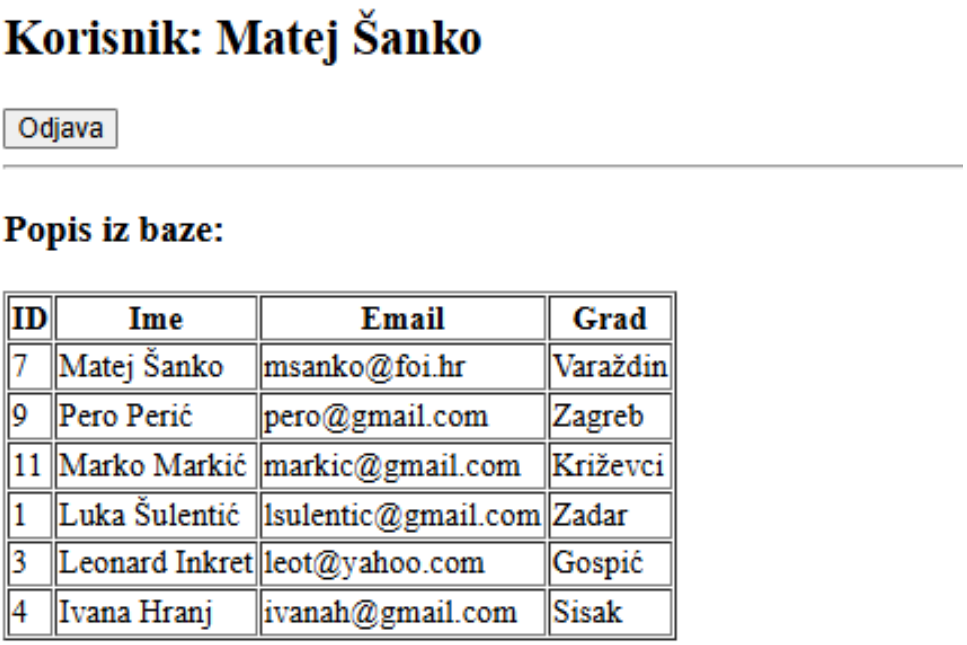
Aplikacija je vidljiva na linku `http://localhost:3000` nakon postavljana potrebnih predispozicija.

Na slici je prikazana početna stranica aplikacije s obrascem za prijavu tj. autentifikaciju. Korisnik unosi svoju email adresu. Prilikom klika na gumb prijavi se, Node.js dio aplikacije putem pg modula šalje asinkroni upit bazi. U pozadini se aktivira trigger koji u tablicu sustav_logovi zapisuje pokušaj pristupa.




Slika 3: Sučelje za prijavu

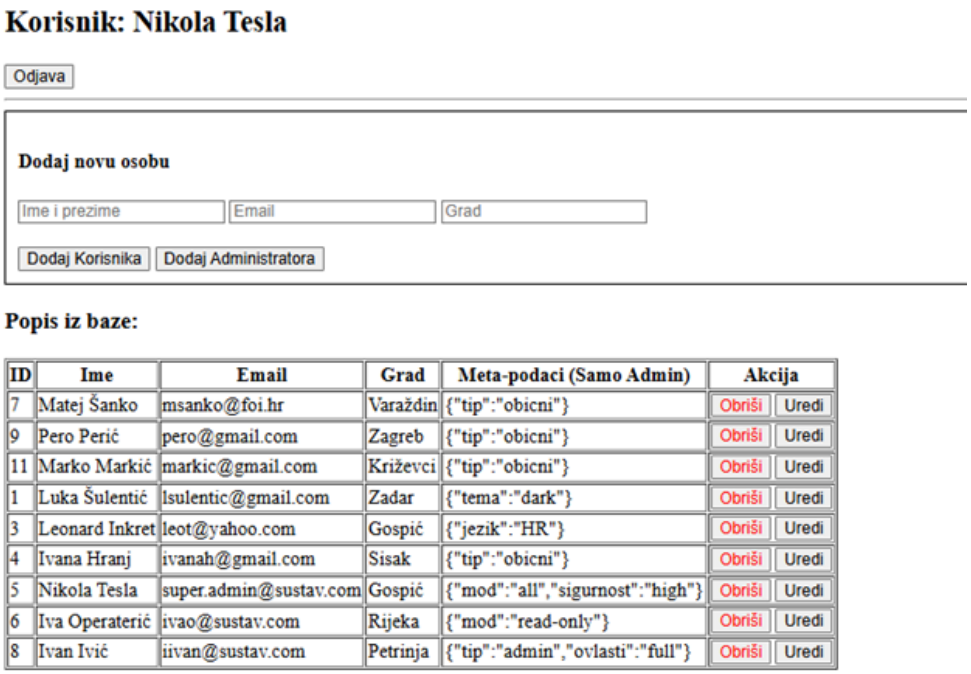
Nakon uspješne prijave običnog korisnika, sustav prikazuje osnovu tablicu s podacima. Podaci se dohvaćaju iz tablice korisnik.




| ID | Ime | Email | Grad |
|---|---|---|---|
| 7 | Matej Šanko | msanko@foi.hr | Varaždin |
| 9 | Pero Perić | pero@gmail.com | Zagreb |
| 11 | Marko Markić | markic@gmail.com | Križevci |
| 1 | Luka Šulentić | lsulentic@gmail.com | Zadar |
| 3 | Leonard Inkret | leot@yahoo.com | Gospić |
| 4 | Ivana Hranj | ivanah@gmail.com | Sisak |

Slika 4: Pogled običnog korisnika

Na slici se vidi pogled kada se prijavimo kao administrator, s kojim se vide dodatne informacije u odnosu na pogled korisnika. Te informacije su razina ovlasti (metapodaci) te mogućnost da se doda novi korisnik ili administrator te da se iz baze izbriše neki korisnik ili administrator te mogućnost izmjene njihovih podataka. Ovim prikazom se pokazuje polimorfizam i nasljeđivanje. Iako su podaci o administratoru fizički u tablici koja nasljeđuje korisnika, aplikacija ih dohvaća upitom.




Slika 5: Pogled administratora

# 6. Zaključak

Ovim projektom sam uspješno demonstrirao primjenu objektno-relacijskog modela baze podataka uz pomoć PostgreSQL-a i web okruženja sa Node.js-om. Korištenjem sustava PostgreSQL kao glavne korištene tehnologije postignut je zadovoljavajući stupanj automatizacije i fleksibilnosti koji klasični relacijski modeli teško ostvaruju bez kompleksnih rješenja.

Primijenjenim konceptom nasljeđivanja tablica uspio sam eliminirati redundantnost stupaca između korisnika i administratora te time omogućio polimorfne upite cijelim sustavom. Samom implementacijom triggera zabilježava se prijava u sustav, što je uspjelo poboljšati integritet podataka. A korištenje polustrukturiranog formata JSONB omogućilo je spremanje metapodataka korisnika bez potrebe za stalnim izmjenama sheme baze podataka, čime sam sustav učinio skalabilnim za buduće nadogradnje. Sami Node.js i PostgreSQL su se pokazali kao izvrsno rješenje za implementaciju aplikacijske domene koja zahtijeva strožu strukturu i fleksibilnost u radu sa podacima. Aplikacija je u potpunosti funkcionalna, a naravno uvijek postoji i prostor za potencijalne nadogradnje.

# 7. Poveznica na Overleaf projekt

https://www.overleaf.com/read/hnttfjwqtpyv#aa7f88

# Popis literature


[1] PostgreSQL Global Development Group. „PostgreSQL Documentation: Inheritance," pogledano 20. svibnja 2024. adresa: https://www.postgresql.org/docs/current/ddl-inherit.html.

[2] PostgreSQL Global Development Group. „PostgreSQL Documentation: Triggers," pogledano 20. svibnja 2024. adresa: https://www.postgresql.org/docs/current/triggers.html.

[3] GeeksforGeeks. „Active Databases in SQL," pogledano 20. svibnja 2024. adresa: https://www.geeksforgeeks.org/sql-active-databases/.

[4] PostgreSQL Global Development Group. „PostgreSQL Documentation: JSON Types," pogledano 20. svibnja 2024. adresa: https://www.postgresql.org/docs/current/datatype-json.html.

[5] Brianc. „node-postgres: PostgreSQL client for Node.js," pogledano 20. svibnja 2024. adresa: https://node-postgres.com/.

[6] PostgreSQL Global Development Group. „PostgreSQL Documentation: psql - interactive terminal," pogledano 20. svibnja 2024. adresa: https://www.postgresql.org/docs/current/app-psql.html.

# Popis slika

# Popis isječaka koda