

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

**PROGRAMSKA POTPORA KOMUNIKACIJSKIM
SUSTAVIMA**

Aplikacija za grupno upravljanje zadacima

Nikola Botić,
Zagreb 2025.

Table of Contents

| | |
|--|-----------|
| Uvod..... | 3 |
| 1. Korištene tehnologije | 4 |
| 1.1. React – korisničko sučelje | 4 |
| 1.2. Spring Boot - poslužitelj | 4 |
| 1.2.1. REST API | 5 |
| 1.2.2. JWT autentikacija | 6 |
| 1.2.3. WebSocket komunikacija..... | 6 |
| 1.2.4. Hibernate (ORM sloj) | 7 |
| 1.3. PostgreSQL..... | 7 |
| 1.4. WebSocket | 8 |
| 2. Korištenje aplikacije | 13 |
| 3. Github Repo..... | 18 |

Uvod

U sklopu projektnog zadatka iz kolegija *Programsko inženjerstvo komunikacijskih sustava* razvijena je moderna web aplikacija namijenjena timskom upravljanju zadacima. Glavni cilj aplikacije je omogućiti timovima bolju organizaciju posla kroz jasno definirane zadatke, raspodjelu odgovornosti i mogućnost zajedničkog rada u stvarnom vremenu, bez obzira na fizičku udaljenost članova tima.

Aplikacija omogućava korisnicima kreiranje korisničkog računa putem jednostavnog procesa registracije, nakon čega mogu stvarati vlastite projekte. Svaki projekt sadrži radnu ploču (eng. *workspace*), koju korisnik može dodatno strukturirati u sekcije poput „TODO“, „IN PROGRESS“ i „DONE“. Ovakva podjela omogućava bolju preglednost zadataka i njihovo kategoriziranje prema statusu, fazi izrade ili tematskim cjelinama.

U svaku sekciju moguće je unositi konkretne zadatke, pri čemu svaki zadatak sadrži naslov, opis, status, informaciju o korisniku koji ga je kreirao, kao i onome tko ga je posljednji ažurirao. Također, zadacima se mogu pridružiti određeni članovi tima, što omogućava jasnu dodjelu odgovornosti i bolju koordinaciju među članovima.

Ključna funkcionalnost aplikacije je podrška za suradnju u stvarnom vremenu. Korisnici mogu druge korisnike pozvati da se pridruže određenom projektu putem pozivnica koje se šalju unutar same aplikacije. Nakon prihvatanja pozivnice, svi članovi projekta imaju jednak pristup zajedničkoj radnoj ploči i mogu simultano uređivati zadatke. Sve promjene – dodavanje, ažuriranje ili brisanje zadataka – automatski se prikazuju svim korisnicima, zahvaljujući integraciji WebSocket protokola koji omogućuje dvosmjernu komunikaciju u stvarnom vremenu bez potrebe za ponovnim učitavanjem stranice.

Tehnološki, aplikacija se temelji na suvremenom razvojnom stacku. Korisničko sučelje izrađeno je korištenjem React biblioteke s podrškom za TypeScript, što omogućuje bržu i sigurniju izradu interaktivnih komponenti. Serverska strana implementirana je pomoću Spring Boot okvira, koji pruža stabilnu i skalabilnu osnovu za backend logiku. Komunikacija između klijenta i poslužitelja odvija se putem RESTful API-ja i WebSocket protokola, dok se svi podaci o korisnicima, projektima i zadacima pohranjuju u relacijsku bazu podataka PostgreSQL.

Ovaj izvještaj nastavlja s detaljnim opisom implementiranih funkcionalnosti, arhitekture sustava, korištenih tehnologija te korisničkog iskustva kroz praktične primjere.

1. Korištene tehnologije

1.1. React – korisničko sučelje

Za razvoj korisničkog sučelja korištena je biblioteka React, koja omogućuje izradu modularnih, interaktivnih i responzivnih web aplikacija. React je odabran zbog svoje fleksibilnosti, bogatog ekosustava i aktivne zajednice, što značajno olakšava razvoj složenijih sučelja s velikim brojem međusobno povezanih komponenti.

Jedna od prednosti Reacta je njegovo komponentno arhitektonsko načelo, koje omogućava ponovno korištenje koda i veću preglednost strukture aplikacije. Komponente su jasno odvojene logičke cjeline koje upravljaju vlastitim stanjem i ponašanjem, što olakšava održavanje i proširivanje aplikacije.

U sklopu projekta korišten je i Ant Design (antd) — popularna React UI biblioteka koja pruža unaprijed definirane komponente dizajnirane prema suvremenim dizajnerskim principima. Korištenjem antd-a ubrzan je razvoj sučelja te je postignut dosljedan i profesionalan izgled aplikacije, uz minimalan trud u dizajnu i stiliziranju. Komponente poput formi, tablica, modalnih prozora i notifikacija iz antd biblioteke omogućile su brzu i intuitivnu implementaciju funkcionalnosti.

Za potrebe komunikacije u stvarnom vremenu, React aplikacija koristi WebSocket konekciju prema backendu. WebSocket omogućuje dvosmjernu komunikaciju između klijenta i poslužitelja, što je ključno za prikaz promjena u zadacima i projektima u stvarnom vremenu. React komponente osluškuju WebSocket događaje te u skladu s njima ažuriraju prikaz sučelja bez potrebe za ponovnim učitavanjem stranice, što znatno poboljšava korisničko iskustvo.

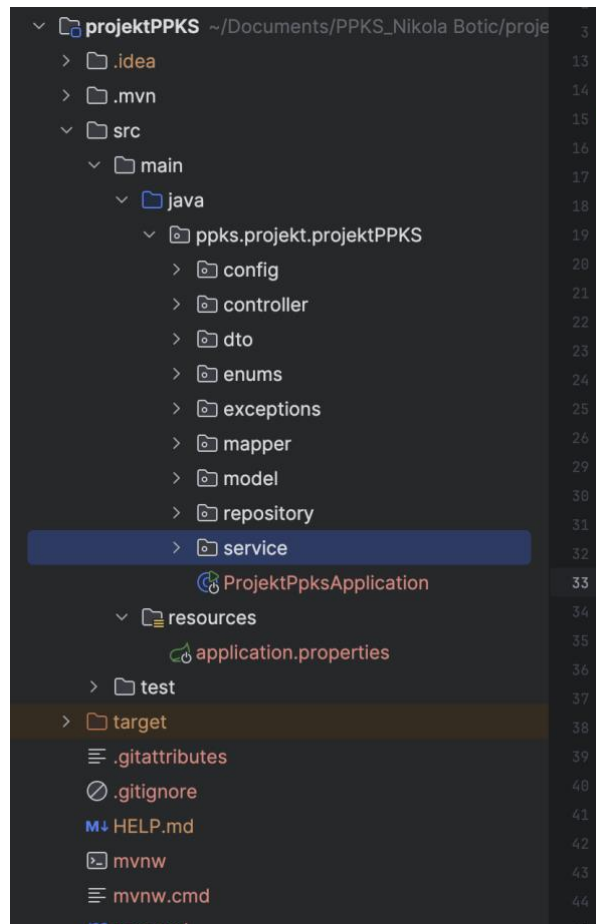
Kombinacijom Reacta, antd biblioteke i WebSocketa postignuto je moderno, responzivno i dinamično korisničko sučelje koje učinkovito podržava višekorisnički rad i kolaboraciju.

1.2. Spring Boot - poslužitelj

Za razvoj serverske strane aplikacije korišten je Spring Boot, okvir temeljen na Javi koji omogućava brzo i jednostavno postavljanje web aplikacija s minimalnim konfiguracijama. Njegova modularna arhitektura i bogata integracija s ostalim Spring tehnologijama čine ga idealnim izborom za izgradnju skalabilnih i održivih sustava.

Aplikacija je razvijena prema principima MVC (Model-View-Controller) arhitekture, koja omogućuje jasan razdvajanje slojeva sustava:

- Model predstavlja poslovne entitete i logiku,
- Controller obrađuje HTTP zahtjeve i usmjerava ih prema odgovarajućim servisima,
- View sloj se u ovom slučaju odnosi na REST API odgovore, jer se koristi frontend u Reactu.



Slika 1 - MVC u mojem projektu

1.2.1. REST API

Aplikacija koristi RESTful API za komunikaciju između frontend i backend dijela sustava. REST API koristi standardne HTTP metode za izvođenje CRUD operacija nad entitetima poput korisnika, projekata, zadataka i sekcija. JSON je korišten kao format za razmjenu podataka, što omogućava jednostavnu integraciju i jasnoću u komunikaciji između klijenta i servera.

```

@RestController
@RequestMapping("/projects")
@RequiredArgsConstructor
public class ProjectController extends ApplicationController {

    private final ProjectService projectService;
    private final ProjectWebSocketController projectWebSocketController;

    @GetMapping("/")
    public ResponseEntity<List<ProjectDTOResponse>> getAllProjects() {
        return ResponseEntity.ok(projectService.getAllProjects());
    }

    @GetMapping("/{id}")
    public ResponseEntity<ProjectDTOResponse> getProjectById(@PathVariable Long id) {
        return projectService.getProjectById(id).Optional<ProjectDTOResponse>
            .map(ResponseEntity::ok).Optional<ResponseEntity<...>>
            .orElse(ResponseEntity.notFound().build());
    }

    @GetMapping("/{id}/tasks")
    public ResponseEntity<List<TaskDTOResponse>> getTasksForProject(@PathVariable Long id) {
        return ResponseEntity.ok(projectService.getTasksByProjectId(id));
    }

    @PostMapping("/")
    public ResponseEntity<ProjectDTOResponse> createProject(@RequestBody ProjectDTO project) {
        ProjectDTOResponse response = projectService.createProject(project, currentUser);
        projectWebSocketController.sendProjectUpdate(response);
        return ResponseEntity.ok(response);
    }

    @PutMapping("/{id}")
    public ResponseEntity<ProjectDTOResponse> updateProject(@PathVariable Long id, @RequestBody ProjectDTO project) {
        ProjectDTOResponse response = projectService.updateProject(id, project);
        projectWebSocketController.sendProjectUpdate(response);
        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteProject(@PathVariable Long id) {
        projectService.deleteProject(id);
        projectWebSocketController.sendProjectUpdate(projectDto: null); // ili pošalji ID obrisano projekta
        return ResponseEntity.noContent().build();
    }
}

```

Slika 2 - Primjer Projekt Controllera

1.2.2. JWT autentikacija

Za autentikaciju korisnika implementiran je JWT (JSON Web Token) sustav. Nakon prijave, korisniku se generira token koji se koristi za pristup zaštićenim API resursima. JWT omogućava stateless autentikaciju jer server ne pohranjuje sesije, već se autentikacija vrši dekodiranjem i provjerom valjanosti tokena na svakom zahtjevu. Ovo pojednostavljuje sigurnosni model i povećava skalabilnost aplikacije.

1.2.3. WebSocket komunikacija

Za omogućavanje ažuriranja podataka u stvarnom vremenu implementiran je WebSocket, koji pruža dvosmjernu komunikaciju između klijenta i servera. Na taj

način, sve promjene koje korisnik izvrši na zadacima – poput dodavanja, uređivanja ili brisanja – odmah se reflektiraju kod svih povezanih korisnika. Spring Boot nudi nativnu podršku za WebSocket, što je omogućilo učinkovitu i pouzdanu integraciju s postojećom infrastrukturom.

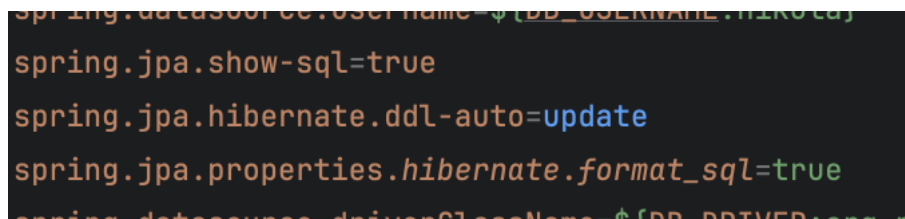
1.2.4. Hibernate (ORM sloj)

Za pristup i upravljanje podacima u bazi korišten je Hibernate, objektno-relacijski mapper (ORM) koji omogućuje povezivanje Java objekata s relacijskim tablicama. Hibernate automatizira velik dio SQL operacija i omogućava rad s bazom podataka na razini objekata, čime se značajno smanjuje količina ručno pisanog SQL koda.

Korištenje Hibernatea omogućilo je:

- automatsko mapiranje entiteta na tablice baze podataka,
- definiranje odnosa između entiteta (npr. one-to-many, many-to-one),
- upravljanje transakcijama i perzistencijom podataka putem JPA (Java Persistence API) standarda.

Zahvaljujući Hibernateu, poslovna logika ostaje čista i fokusirana, dok se pristup podacima odvija na deklarativan i tip-siguran način.



```
spring.datasource.username=${DB_USERNAME:hiberto}
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.datasource.driverClassName=${DB_DRIVER:org.p
```

Slika 3 - Definicija Hibernate-a

1.3. PostgreSQL

Kao sustav za upravljanje relacijskom bazom podataka korišten je PostgreSQL, moćan open-source RDBMS poznat po svojoj pouzdanosti, skalabilnosti i podršci za napredne SQL značajke. PostgreSQL je odabran zbog odlične integracije s Spring Boot i Hibernate alatima, kao i zbog robusne podrške za transakcije, integriteta podataka i kompleksnih odnosa među tablicama.

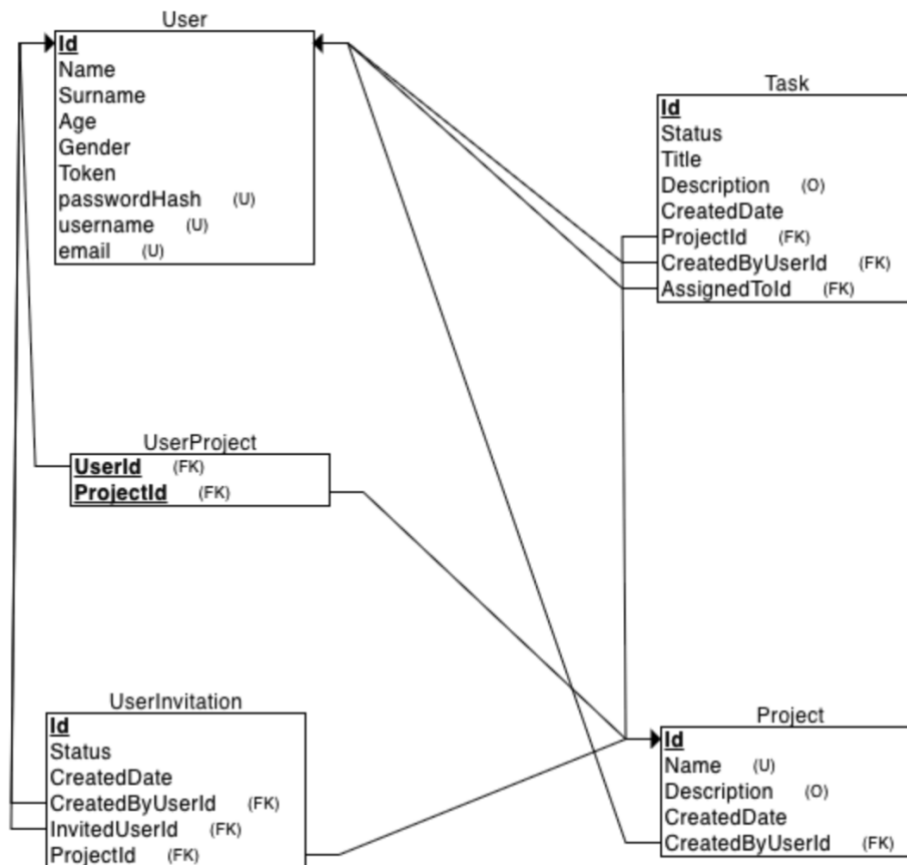
U aplikaciji, PostgreSQL služi kao centralna točka za pohranu svih podataka vezanih uz korisnike, projekte, zadatke, pozivnice i promjene statusa. Svaki entitet u aplikaciji (npr. korisnik, zadatak, sekcija) mapiran je na odgovarajuću tablicu u bazi putem Hibernatea.

Dodatne prednosti korištenja PostgreSQL baze uključuju:

- podršku za ACID transakcije, čime se osigurava konzistentnost podataka,
- mogućnost definiranja vanjskih ključeva, indeksa i ograničenja radi očuvanja integriteta,

- napredne mogućnosti pretraživanja i filtriranja podataka pomoću SQL upita,
- dobru podršku za razvoj u okruženjima s više korisnika i većim opterećenjem.

Baza je inicijalno strukturirana koristeći JPA anotacije unutar entitetskih klasa, čime je omogućeno automatsko generiranje tablica i relacija na temelju definicija u kodu.



Slika 1 - Relacijski model baze podataka

1.4. WebSocket

WebSocket je komunikacijski protokol koji omogućuje dvosmjernu, trajnu i full-duplex vezu između klijenta i poslužitelja. Za razliku od klasičnog HTTP protokola, koji je temeljen na modelu zahtjev-odgovor i za svaku novu informaciju zahtijeva novi HTTP zahtjev, WebSocket omogućuje stalnu povezanost između dviju strana, čime se znatno smanjuje latencija i poboljšava korisničko iskustvo.

Ključne značajke WebSoketa:

- Dvosmjerna komunikacija: Server može inicirati poruke prema klijentu bez prethodnog zahtjeva, što nije moguće u klasičnom HTTP modelu.

- Niska latencija: Nakon uspostavljanja veze, komunikacija se odvija bez ponovnog otvaranja konekcije, što znatno ubrzava prijenos podataka.
- Učinkovitost: Manji overhead u odnosu na HTTP protokol – koristi binarni ili tekstualni frame format umjesto HTTP headera za svaki zahtjev.
- Stalna povezanost: Jednom uspostavljena veza ostaje otvorena sve dok jedna od strana ne zatraži prekid, omogućujući sinkronizaciju u stvarnom vremenu.

Primjena u aplikaciji

U aplikaciji za timsko upravljanje zadacima WebSocket je ključna komponenta koja omogućuje kolaboraciju u stvarnom vremenu. Kada jedan korisnik unese promjenu – npr. doda novi zadatak, promijeni status postojećeg, ili uređuje sadržaj – ta se promjena odmah prenosi preko WebSocket konekcije i prikazuje svim ostalim korisnicima koji su povezani na isti projekt.

To omogućuje:

- Trenutačno osvježavanje zadataka na radnoj ploči svih korisnika bez potrebe za ručnim reloadom stranice.
- Višekorisničku suradnju u stvarnom vremenu – svi sudionici vide iste informacije u sinkroniziranom obliku.
- Veću učinkovitost rada tima i bolju koordinaciju, jer su sve promjene odmah vidljive.

Tehnička implementacija

Na backend strani korišten je Spring Boot s podrškom za WebSocket putem Spring WebSocket modula. Implementacija se sastoji od:

- Konfiguracijske klase (WebSocketConfig) koja definira endpoint i omogućuje CORS ako je potreban.
- Kontrolera/servisa koji obrađuje poruke – npr. mapira ih na određene projekte i šalje korisnicima koji su pretplaćeni na taj kanal.
- Modela poruke – definiran je format poruke koji sadrži potrebne podatke (tip događaja, sadržaj zadatka, ID projekta itd.).

Na frontend strani (React), koristi se JavaScript WebSocket API (ili biblioteka poput socket.io ili stomp.js ako se koristi STOMP protokol preko SockJS) za:

- Inicijalizaciju konekcije s backendom.
- Osluškivanje i primanje poruka s poslužitelja.
- Emitiranje poruka prema poslužitelju kada korisnik izvrši neku akciju (npr. dodavanje zadatka).

Primjer toka komunikacije:

1. Klijent otvara WebSocket konekciju prema serveru (/ws endpoint).
2. Klijent se prijavljuje na određeni “kanal” (npr. projekt ID).

3. Kada korisnik doda novi zadatak, ta se akcija šalje serveru putem WebSocketeta.
4. Server prima poruku, obrađuje je i šalje natrag svim klijentima koji su prijavljeni na isti kanal.
5. Klijentska aplikacija ažurira prikaz na temelju primljene poruke.

Prednosti za krajnjeg korisnika

Korištenje WebSocketeta korisnicima donosi iskustvo suvremene kolaborativne aplikacije, gdje svi članovi tima u stvarnom vremenu mogu vidjeti i reagirati na promjene. Nema kašnjenja, potrebe za ručnim osvježavanjem niti rizika od nedosljednih podataka. Ovo je posebno važno kod timskog rada na projektima, gdje su ažurne informacije ključne za donošenje odluka i praćenje napretka.

```

export const useWebSocket = () => useContext(webSocketContext);

export const WebSocketProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [isConnected, setIsConnected] = useState(false);
  const [lastProjectUpdate, setLastProjectUpdate] = useState<ProjectDTOResponse | null>(null);
  const [lastTaskUpdate, setLastTaskUpdate] = useState<TaskDTOResponse | null>(null);
  const [lastInvitationUpdate, setLastInvitationUpdate] = useState<UserInvitationResponseDTO | null>(null);
  const { global } = useContext(StateContext);

  useEffect(() => {
    if (!global.user?.token) return;

    const ws = new WebSocket(`ws://localhost:8080/api/ws?token=${global.user.token}`);
    const stompClient = new Client({
      websocketFactory: () => ws,
      connectHeaders: {
        Authorization: `Bearer ${global.user.token}`
      },
      debug: (str) => console.log('STOMP: ' + str),
      reconnectDelay: 5000,
      heartbeatIncoming: 4000,
      heartbeatOutgoing: 4000,
      onConnect: () => {
        console.log('Connected to WebSocket');
        setIsConnected(true);

        // Subscribe to project updates
        stompClient.subscribe('/topic/projects', (message) => {
          const projectUpdate = JSON.parse(message.body);
          setLastProjectUpdate(projectUpdate);
        });

        // Subscribe to task updates
        stompClient.subscribe('/topic/tasks', (message) => {
          const taskUpdate = JSON.parse(message.body);
          setLastTaskUpdate(taskUpdate);
        });

        // Subscribe to user invitation updates
        stompClient.subscribe(`/topic/invitations/${global.user?.id}`, (message) => {
          console.log('Received invitation update:', message.body);
          const invitationUpdate = JSON.parse(message.body);
          setLastInvitationUpdate(invitationUpdate);
        });
      },
      onDisconnect: () => {
        console.log('Disconnected from WebSocket');
        setIsConnected(false);
      },
      onStompError: (frame) => {
        console.error('STOMP error:', frame);
      }
    });
  }, [global.user?.token]);
};

```

Slika 4 - Primjer WebSocketa iz projekta u Reactu

```

// WebSocketConfig.java
package ppks.projekt.projektPPKS.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.*;
import ppks.projekt.projektPPKS.config.JwtHandshakeInterceptor;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    private final JwtHandshakeInterceptor jwtHandshakeInterceptor; 2 usages

    public WebSocketConfig(JwtHandshakeInterceptor jwtHandshakeInterceptor) {
        this.jwtHandshakeInterceptor = jwtHandshakeInterceptor;
    }

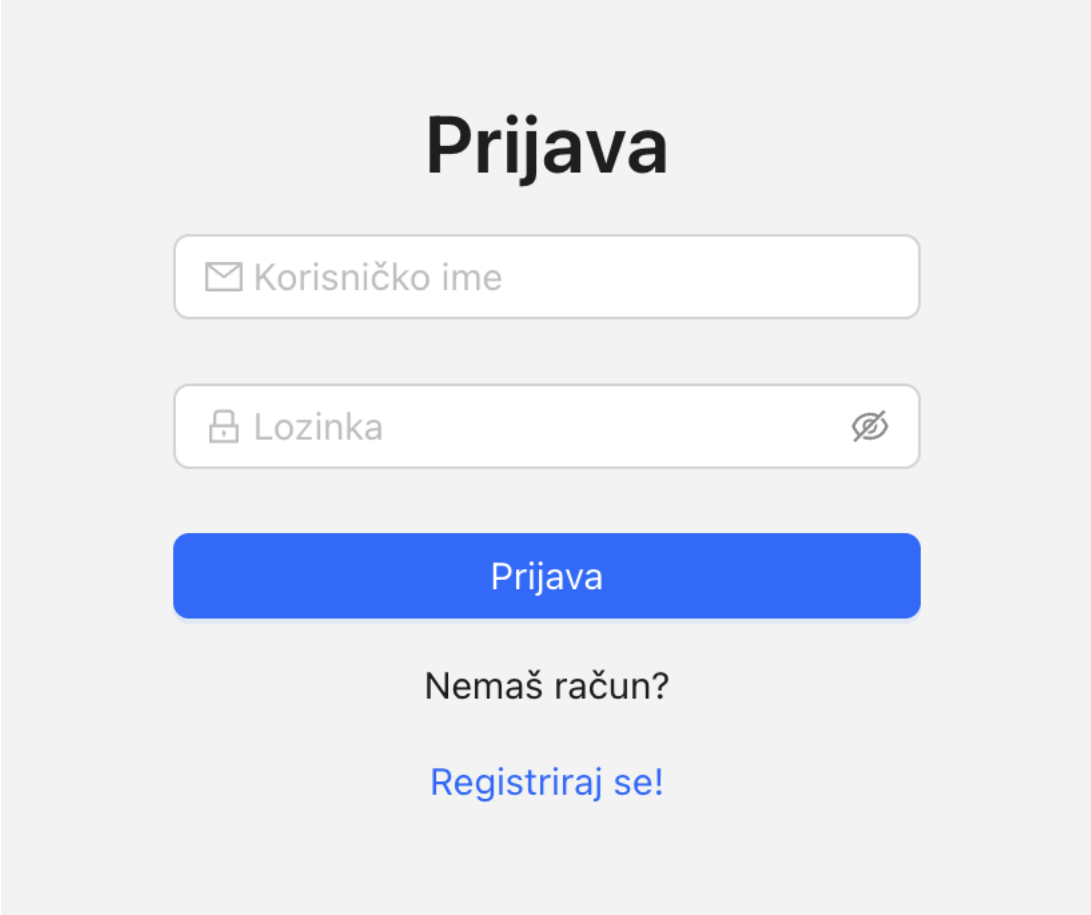
    @Override no usages
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry
            .addEndpoint(...paths: "/ws")
            .addInterceptors(jwtHandshakeInterceptor)
            .setAllowedOrigins("http://localhost:5173");
    }

    @Override no usages
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker(...destinationPrefixes: "/topic"); // broker za /topic/*
        config.setApplicationDestinationPrefixes("/app"); // prefix za @MessageMapping
    }
}

```

Slika 5 - Razred na poslužitelju za WebSocket

2. Korištenje aplikacije



The screenshot shows a login interface on a light gray background. At the top, the word "Prijava" is centered in a large, bold, black font. Below it are two input fields. The first field is labeled "Korisničko ime" with an envelope icon on the left. The second field is labeled "Lozinka" with a lock icon on the left and a toggle icon (an eye with a diagonal line) on the right. Below these fields is a solid blue button with the text "Prijava" in white. Under the button, the text "Nemaš račun?" is displayed, followed by a blue hyperlink "Registriraj se!".

Prijava

✉ Korisničko ime

🔒 Lozinka 

Prijava

Nemaš račun?

[Registriraj se!](#)

Slika 6 - Ekran prijave korisnika

Registracija

* Korisničko ime:

* Ime i prezime:

* Email:

* Lozinka:

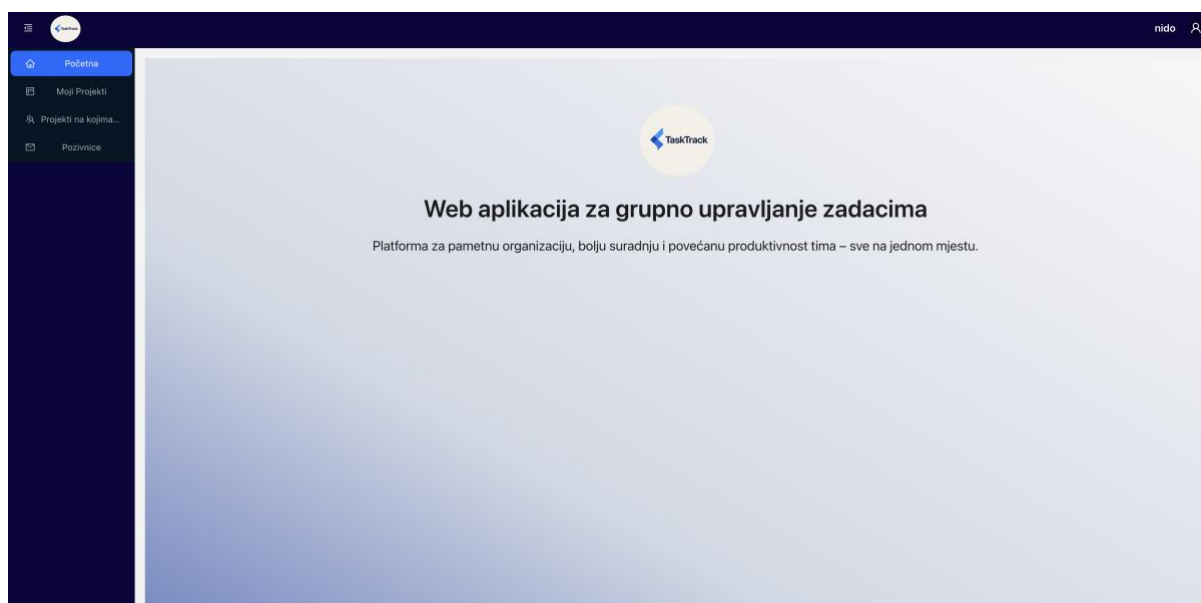
* Ponovite lozinku:

* Broj godina:

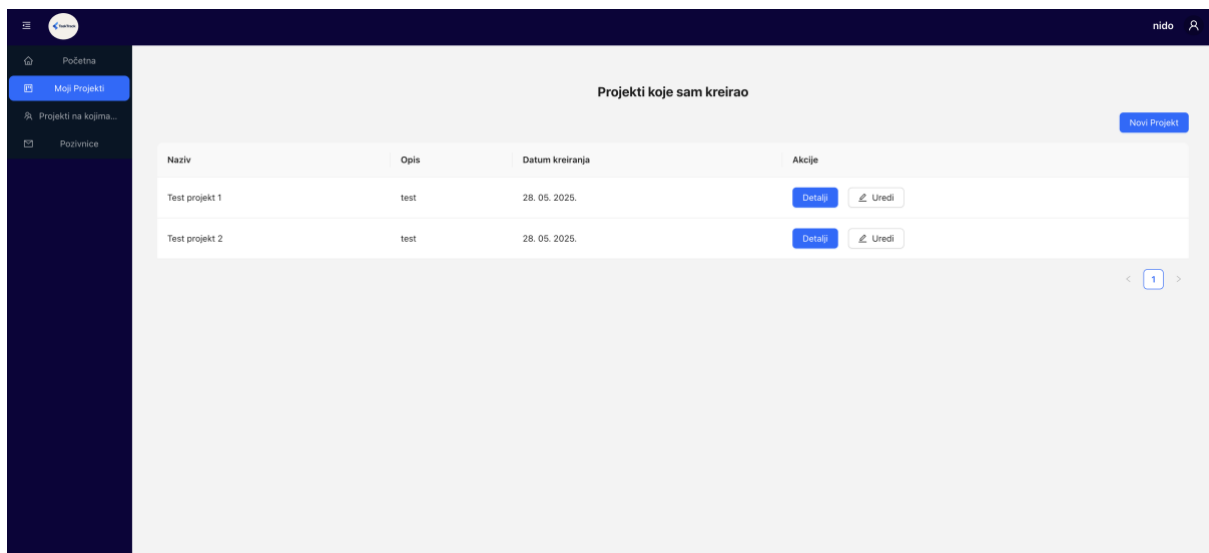
* Spol: ☐ Muški ☐ Ženski

[Registriraj se](#)

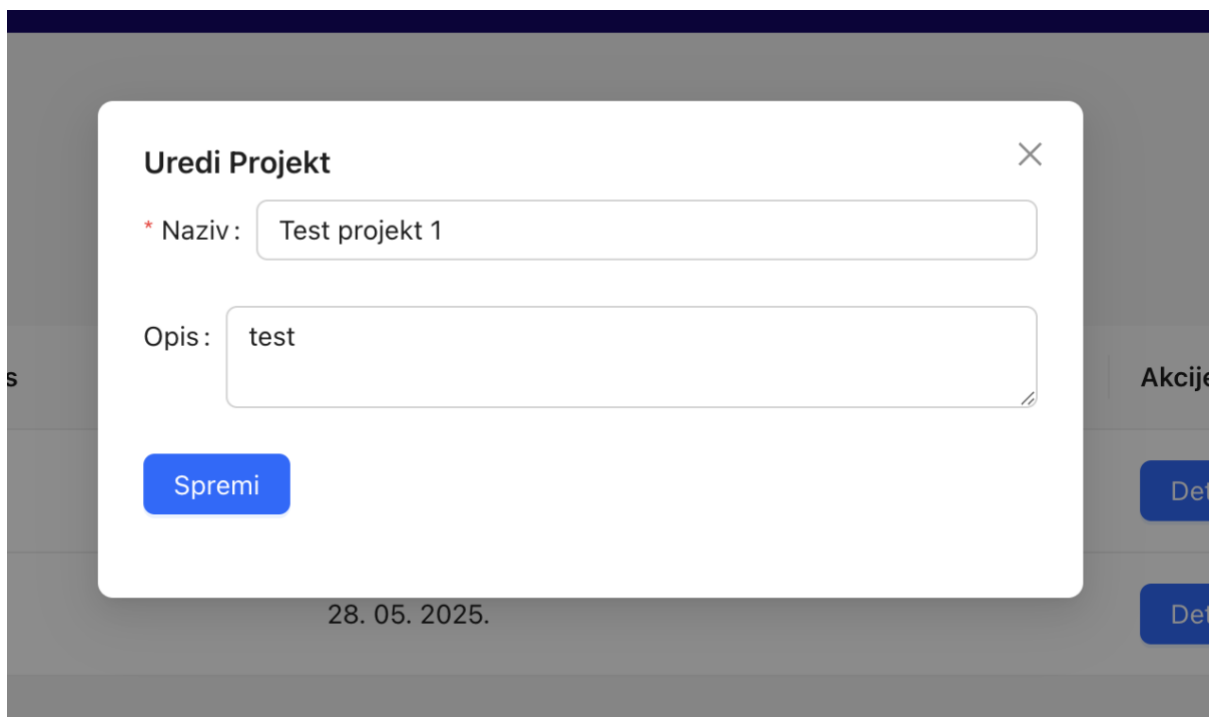
Slika 7 - Ekran registracije korisnika



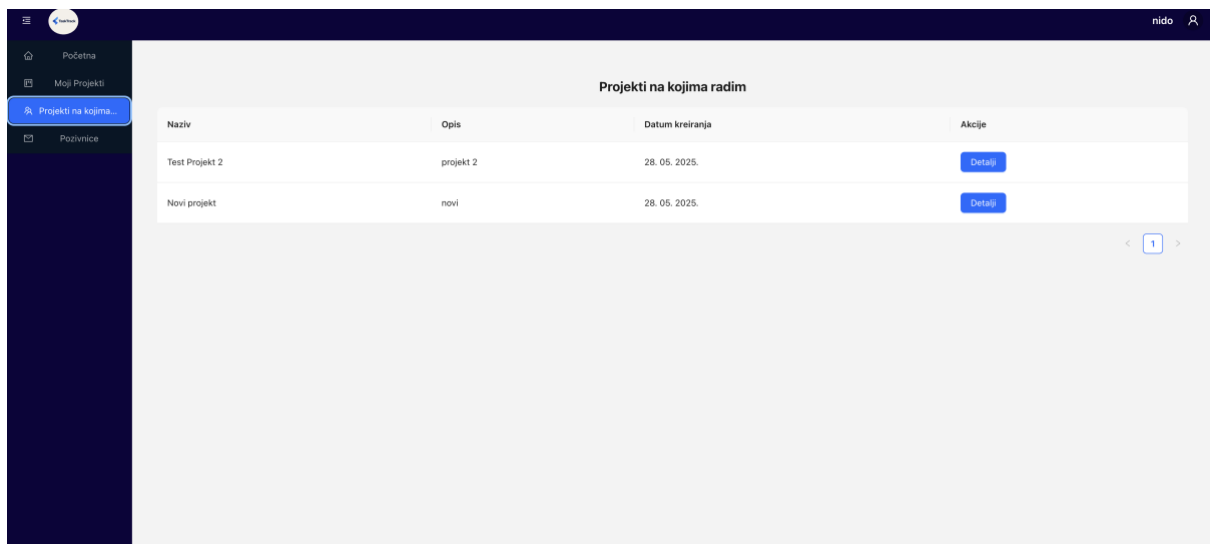
Slika 8 - Početni ekran nakon registracije/prijave



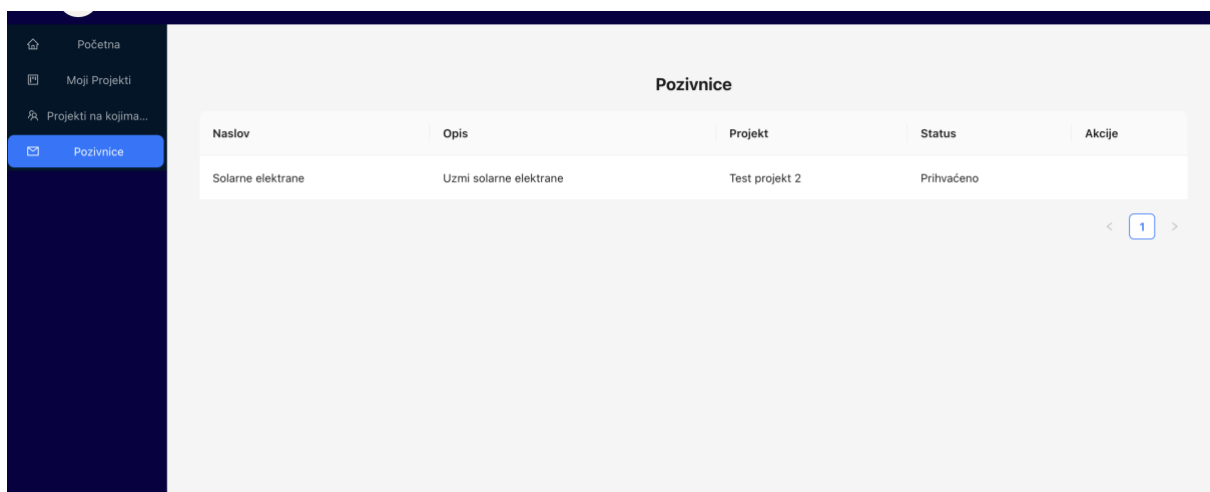
Slika 9 - Projekti koje je korisnik kreirao



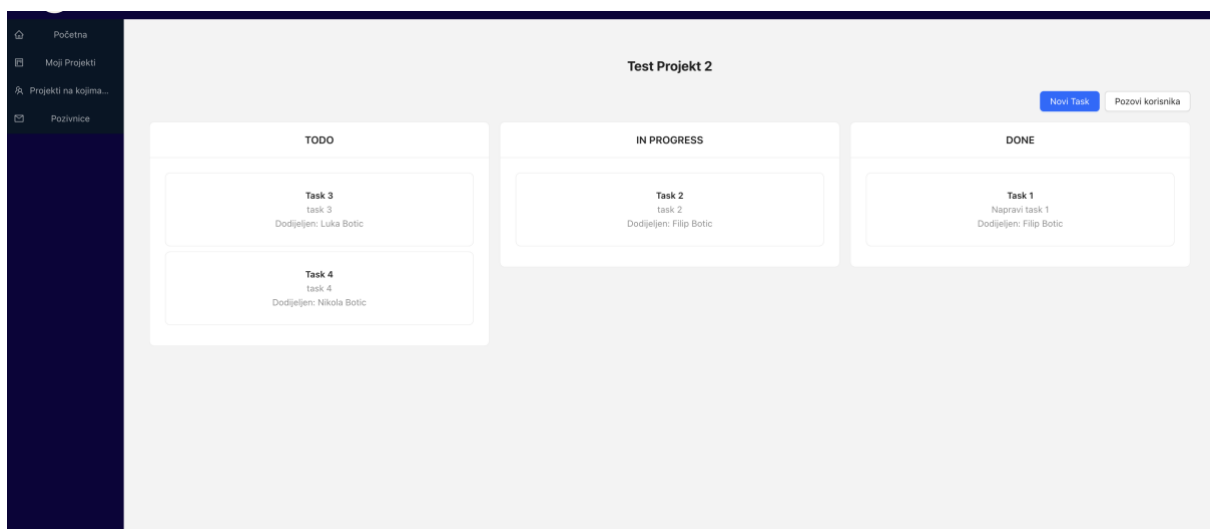
Slika 10 - Uređivanje podataka projekta



Slika 11 - Projekti u koje je korisnik pozvan i potvrdio je sudjelovanje



Slika 12 - Prikaz pozivnica u projekte



Slika 13 - Prikaz radne površine projekta s kreiranim taskovima

Novi Task

*

Naziv:

Opis:

Dodijeli korisniku:

Odaberi korisnika

Kreiraj

Slika 14 - Forma za kreiranje Taska

Pozovi korisnika u projekt

*

Naslov:

*

Opis:

*

Korisnik:

Odaberi korisnika

Pošalji pozivnicu

Slika 15 – Forma za poziv korisnika u projekt

Task 3

*

Naziv:

Task 3

Opis:

task 3

*

Status:

TODO

Dodijeli korisniku:

Luka Botic

Kreirao:

Nikola Botic

Datum kreiranja:

28. 05. 2025.

Slika 16 - Prikaz forme za uređivanje podataka Taska

3. Github Repo

Link: https://github.com/NBotic02/PPKS_Projekt.git