Arhitektura

1. Kontekst i cilj softverskog projekta

1.1. Kontekst sistema

Moderno praćenje sportskih događaja prevazišlo je pasivno gledanje utakmica i sve više se oslanja na socijalnu interakciju i "second-screen" iskustvo. Iako postoje brojne platforme za sportsko klađenje, na tržištu nedostaju rešenja koja su fokusirana isključivo na socijalni aspekt takmičenja unutar zatvorenih krugova ljudi (prijatelji, kolege), bez rizika monetarnog kockanja. Postojeći načini organizacije ovakvih takmičenja su neefikasni, skloni greškama i ne pružaju ažuriranje stanja u realnom vremenu.

Softverski sistem se razvija kao distribuirana web platforma koja omogućava korisnicima formiranje disjunktnih, privatnih grupa ("liga"). Unutar ovih izolovanih celina, sistem mora da obezbedi mehanizme za predviđanje ishoda, automatsko bodovanje i rangiranje.

1.2. Cilj softverskog projekta

Osnovni cilj projekta je projektovanje i implementacija robusne, skalabilne softverske arhitekture koja podržava kolaborativno predviđanje sportskih rezultata sa fokusom na korisničko iskustvo u realnom vremenu .

Specifični ciljevi sistema obuhvataju:

1. Upravljanje disjunktnim grupama: Implementacija logike za kreiranje izolovanih takmičarskih okruženja gde podaci i interakcije jedne grupe ne utiču na druge, čime se osigurava privatnost i relevantnost podataka.
2. Sistem predikcije i bodovanja: Razvoj fleksibilnog mehanizma za unos prognoza pre početka mečeva i automatsku evaluaciju uspešnosti nakon završetka ili promene rezultata, koristeći asinhronu obradu podataka.
3. Real-time komunikacija: Implementacija dvosmerne komunikacije (WebSocket) koja omogućava da svi članovi grupe vide tuđe prognoze (nakon zaključavanja unosa) i promene na rang listi onog trenutka kada se dese, bez potrebe za osvežavanjem stranice (Zero-refresh policy).
4. Vizualizacija podataka: Prikaz statistike uspešnosti kroz interaktivne grafičke interfejse, omogućavajući korisnicima uvid u istoriju pogađanja i komparativnu analizu sa drugim članovima grupe.

Tehnički cilj je demonstracija arhitekture koja razdvaja transakcionu logiku (upis prognoza) od procesiranja događaja (kalkulacija bodova i notifikacije), korišćenjem modernih arhitekturnih obrazaca i tehnologija (NestJS, Angular, RabbitMQ).

2. Arhitekturno specifični zahtevi, atribut i kvalitet, tehnička i poslovna ograničenja

Sistem mora podržati sledeće ključne funkcionalnosti koje direktno utiču na dizajn baze podataka i komunikacione protokole:

2.1. Glavni funkcionalni zahtevi

1. Autentifikacija i upravljanje nalogom:
   - Registracija: Sistem mora omogućiti kreiranje novog korisničkog naloga unosom validnih podataka (email, korisničko ime, lozinka). Lozinke se ne smeju čuvati u otvorenom tekstu, već moraju biti kriptografski heširane (npr. bcrypt/argon2).
   - Logovanje: Bezbedna prijava korisnika koja rezultira izdavanjem pristupnog tokena (JWT - JSON Web Token). Ovaj token se koristi za autorizaciju svih narednih zahteva ka serveru, čime se osigurava stateless priroda REST API-ja.
2. Upravljanje disjunktnim grupama (Tenancy & Isolation):
   - Korisnici mogu kreirati privatne grupe ("Lige") ili im se pridružiti putem jedinstvenog pristupnog koda.
   - Podaci o prognozama, četovima i rang listama moraju biti logički izolovani – aktivnost u jednoj grupi ne sme biti vidljiva u drugoj.
3. Upravljanje predikcijama sa vremenskim ograničenjem:
   - Korisnik može uneti ili izmeniti prognozu rezultata utakmice.
   - Sistem mora automatski "zaključati" mogućnost unosa prognoze u tačno definisanom trenutku (početak utakmice). Nakon zaključavanja, prognoze postaju vidljive ostalim članovima grupe radi transparentnosti.
4. Automatizovana evaluacija i bodovanje **:**
   - Sistem mora biti u stanju da primi informaciju o završenoj utakmici (ili promeni rezultata) i da na osnovu toga automatski ažurira bodovno stanje svih korisnika koji su tipovali taj meč.
   - Ovaj proces treba da se izvršava asinhrono kako ne bi blokirao korisnički interfejs.
5. Sinhronizacija stanja u realnom vremenu **:**
   - Svaka promena relevantna za grupu (novi rezultat utakmice, promena pozicije na tabeli, nova poruka u grupi) mora biti momentalno prosleđena svim aktivnim klijentima te grupe, bez potrebe da korisnik ručno osvežava stranicu.

2.2. Atributi kvaliteta

1. Performanse i nizak stepen kašnjenja:
   - S obzirom na prirodu praćenja sporta uživo, kašnjenje između promene rezultata na serveru i prikaza na klijentu treba da bude minimalno (ciljano < 500ms).
   - Arhitekturna odluka: Korišćenje WebSockets **(**Socket**.**io**)** umesto standardnog HTTP polling-a.

2. Skalabilnost i Elastičnost:
   - Sistem mora biti projektovan tako da podnese nagle skokove u saobraćaju koji se dešavaju neposredno pre početka popularnih utakmica ili neposredno nakon gola.
   - Arhitekturna odluka: Izdvajanje teških operacija (kalkulacija bodova za hiljade korisnika) u pozadinske procese korišćenjem Message Brokera , čime se rasterećuje glavni API server.

3. Modularnost i Održivost:
    - Arhitektura mora omogućiti nezavisan razvoj različitih delova sistema (npr. modul za autentifikaciju odvojen od modula za bodovanje).
    - Arhitekturna odluka: Korišćenje NestJS okvira koji forsira modularnu strukturu i Dependency Injection, kao i TypeScript**-a** za strogu tipizaciju koda.
4. Integritet podataka:
    - Sistem mora garantovati da su bodovi tačno izračunati i trajno sačuvani. Nije dozvoljeno da dva korisnika vide različito stanje tabele nakon završene obrade.
    - Arhitekturna odluka: Korišćenje relacione baze PostgreSQL sa ACID transakcijama.

2.3. Ograničenja

1. Tehnička ograničenja:
    - Backend**:** Mora se koristiti Node.js okruženje, specifično NestJS okvir.
    - Frontend**:** Obavezna upotreba Angular okvira.
    - Baza podataka**:** Relaciona baza podataka (PostgreSQL) uz korišćenje TypeORM-a za ORM sloj.
    - Komunikacija**:** Obavezna implementacija WebSocket protokola za real-time funkcionalnosti i Message Queue protokola (AMQP) za internu komunikaciju.
2. Poslovna ograničenja:
    - Rok isporuke**:** Faza arhitekture mora biti završena do 21.1.2026, a finalna verzija sa komunikacijom do 11.2.2026.
    - Regulatorna**:** Aplikacija je isključivo zabavnog karaktera i ne sme uključivati procesiranje pravog novca (zabrana implementacije kockanja/transakcija).

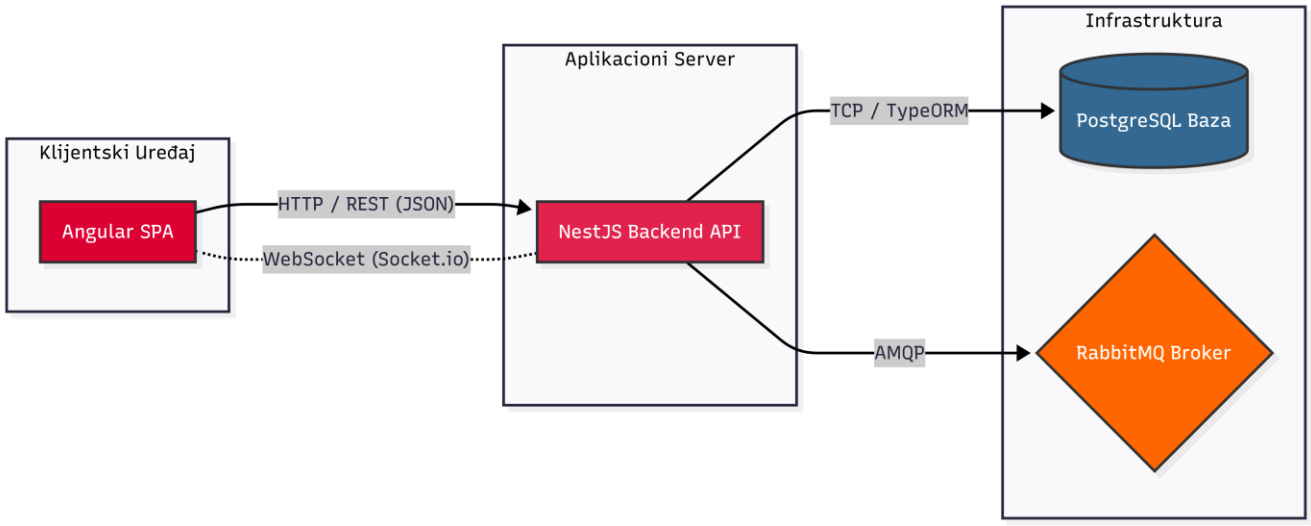
3. Arhitekturni dizajn softverskog sistema

3.1. Korišćeni arhitekturni obrasci

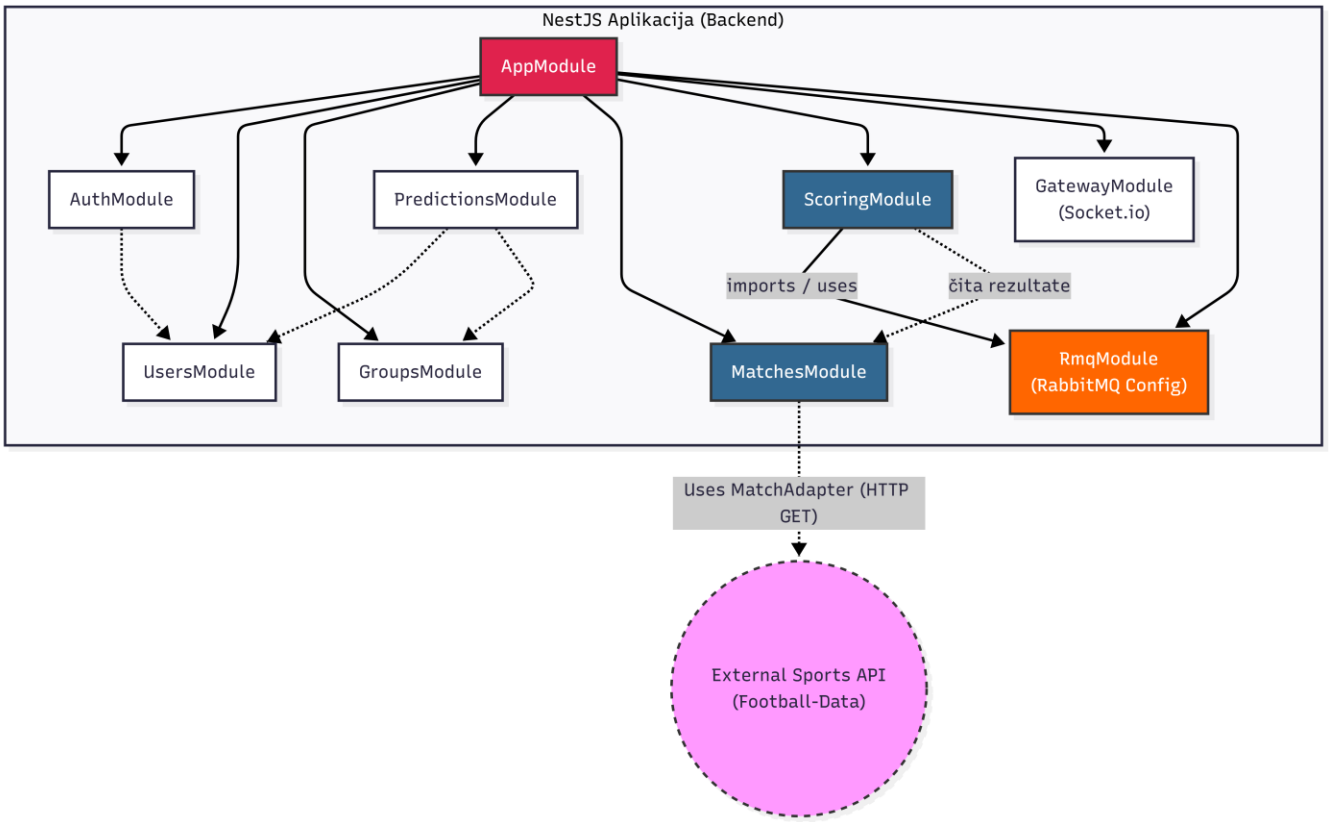
1. Klijent-Server (Client-Server Architecture): Ovo je osnovni arhitekturni stil sistema. Postoji jasna separacija između prezentacionog dela i poslovne logike:
    - Klijent **:** Angular aplikacija koja se izvršava u pretraživaču korisnika. Zadužena je za rendering, validaciju unosa i korisničko iskustvo.
    - Server**:** NestJS aplikacija koja se izvršava na serveru. Zadužena je za bezbednost, pristup bazi podataka i teške kalkulacije.
    - Komunikacija se odvija putem HTTP protokola (REST API) i WebSocket protokola.
2. Slojevita arhitektura (Layered Architecture): Backend sistem je striktno podeljen na slojeve radi lakšeg održavanja:
    - Controller Layer: Prihvata zahteve i validira ulazne podatke.
    - Service Layer: Sadrži poslovnu logiku .
    - Data Access Layer (Repository): Komunicira direktno sa bazom podataka.

3. MVC (Model-View-Controller): Iako je ovo primarno backend arhitektura, ona se reflektuje i na organizaciju koda:
   - Model: TypeORM entiteti i DTO objekti.
   - View: JSON odgovor koji server šalje klijentu , dok pravi UI renderuje Angular.
   - Controller: NestJS kontroleri koji upravljaju tokom zahteva.
4. Publish-Subscribe (Pub/Sub)**:** Koristi se za asinhronu komunikaciju i real-time funkcionalnosti.
   - RabbitMQ*:* Server objavljuje događaj ("Utakmica završena"), a odgovarajući servisi su pretplaćeni na taj događaj kako bi izračunali bodove.
   - Socket.io: Server objavljuje promene klijentima koji "slušaju" kanal svoje grupe.
5. Repository Pattern & Dependency Injection**:** Pristup bazi je apstrahovan kroz repozitorijume, a sve zavisnosti između klasa se ubacuju kroz DI kontejner (Inversion of Control), čime se postiže modularnost i olakšava testiranje.

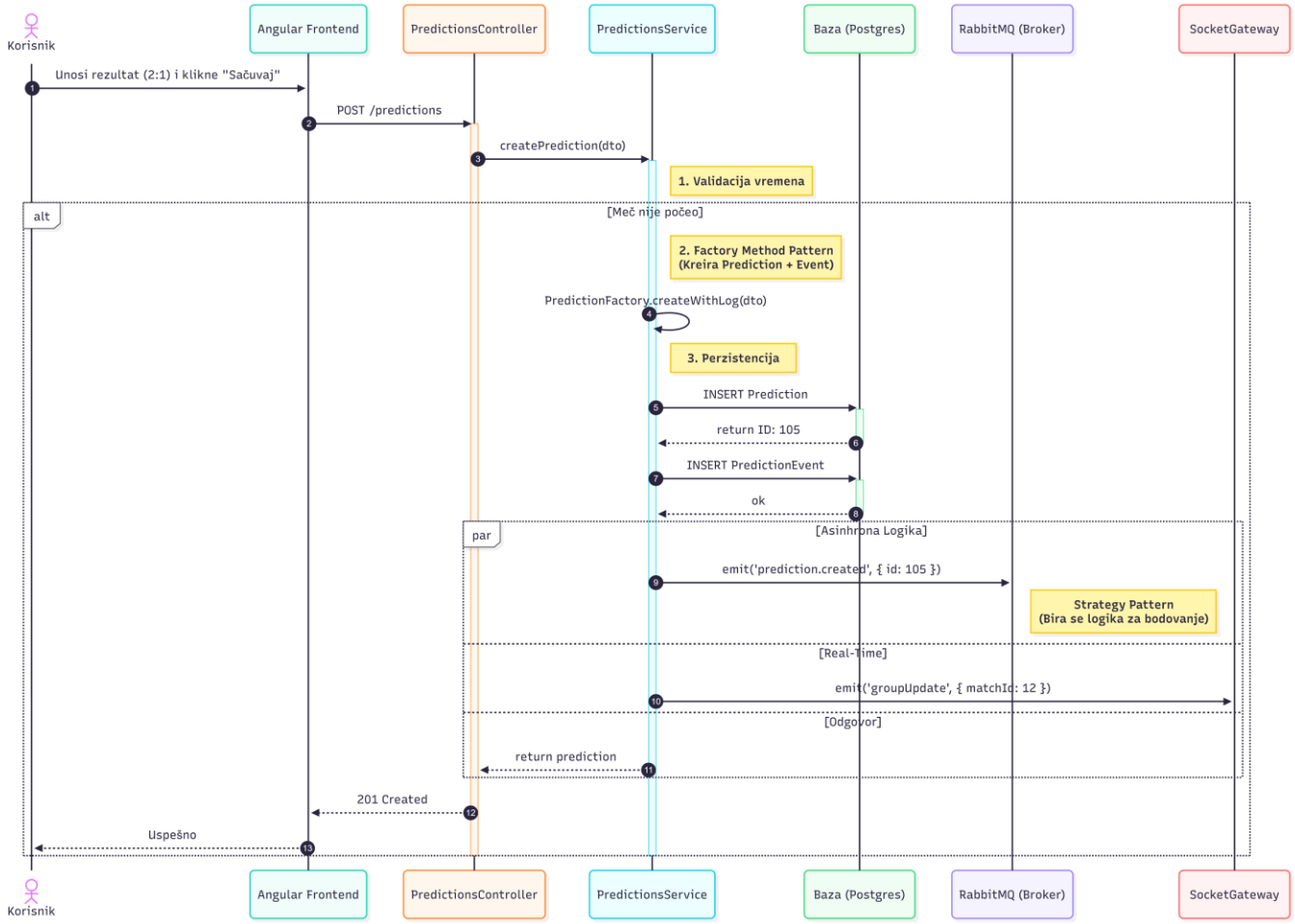
3.2. Generalna arhitektura (box-linde dijagram)



3.3. Strukturni dijagram



3.4. Dijagram sekvence



4. Izbor aplikacionih okvira i biblioteka

4.1. Serverska strana (Backend)

Za razvoj serverske logike i API servisa izabran je NestJS.

- Razlog izbora**:** NestJS je progresivni Node.js okvir koji nudi striktnu modularnu strukturu i ugrađenu podršku za TypeScript. Njegova prednost je Dependency Injection kontejner koji omogućava lako upravljanje zavisnostima i testiranje komponenti, kao i laka integracija sa mikroservisnim arhitekturama (RabbitMQ).
- Primenjeni arhitekturni obrazac: MVC (Model-View-Controller) NestJS prirodno implementira MVC obrazac na sledeći način:
    - Controller**:** Klasa dekorisana sa @Controller() koja prihvata HTTP zahteve, validira ulazne podatke (putem DTO objekata) i poziva odgovarajuće servise.
    - Model (Service/Entity)**:** Poslovna logika je izolovana u servisima (@Injectable()), dok strukturu podataka definišu TypeORM entiteti.
    - View (Pogled): S obzirom da se razvija REST API, "View" je u ovom slučaju serijalizovani JSON odgovor koji se šalje klijentu.

4.2. Klijentska strana (Frontend)

Za razvoj interfejsa aplikacije izabran je Angular.

- Razlog izbora: Angular je opširni okvir koji deli isti programski jezik (TypeScript) sa backend-om, što omogućava deljenje interfejsa podataka. Njegov sistem reaktivnog programiranja je odličan za rukovanje *real-time* podacima koji stižu preko WebSocketa.
- Primenjeni arhitekturni obrazac: MVVM (Model-View-ViewModel) Angular primenjuje varijaciju MVVM obrasca kroz svoju komponentnu arhitekturu:
    - View (Pogled): HTML šablon (Template) komponente koji definiše strukturu i izgled.
    - ViewModel**:** TypeScript klasa komponente. Ona sadrži stanja i metode koje reaguju na akcije korisnika i "vezuje" podatke za View .
    - Model: Servisi i RxJS Observable objekti koji dobavljaju podatke sa servera i upravljaju globalnim stanjem aplikacije.

4.3. Ključne biblioteke za perzistenciju i komunikaciju

Pored osnovnih okvira, sistem se oslanja na sledeće biblioteke koje definišu specifične arhitekturne slojeve:

1. TypeORM (Perzistencija):
    - Koristi se za komunikaciju sa PostgreSQL bazom podataka.
    - Implementira Repository Pattern i Data Mapper Pattern, čime se postiže apstrakcija pristupa podacima – poslovna logika ne zavisi od SQL upita, već od metoda repozitorijuma (npr. save, findOne).
2. Socket.io (Real-time komunikacija):
    - Biblioteka koja omogućava dvosmernu komunikaciju između klijenta i servera.
    - Implementira Observer / Pub-Sub obrazac, gde klijenti "slušaju" (subscribe) događaje u određenim "sobama" (grupama), a server "objavljuje" (publish) promene.
3. Amqplib (Asinhrona obrada):
    - Klijentska biblioteka za RabbitMQ. Omogućava implementaciju Event-Driven arhitekture slanjem poruka o događajima (npr. prediction.created) u redove za čekanje radi pozadinske obrade.