Univerzitet u Beogradu

Fakultet organizacionih nauka

Katedra za elektronsko poslovanje

Web aplikacija za Fitness

|  |  |  |  |
| --- | --- | --- | --- |
| Rbr | Ime | Prezime | Broj indeksa |
| 1. | Luka | Stajković | 2021/0129 |
| 2. | Draga | Raković | 2021/0288 |
| **Mentor** | | Aleksandra Trpkov | |
| **GitHub link** | | https://github.com/elab-development/internet-tehnologije-2024-projekat-fitnesswebapp\_20210129\_20210288 | |

Sadržaj

[1. UVOD 7](#_Toc214747198)

[2. METODE I TEHNOLOGIJE ZA RAZVOJ VEB APLIKACIJE 9](#_Toc214747199)

[**2.1.** Uprošćena Larmanova metoda 11](#_Toc214747200)

[**2.2.** Tehnologije za razvoj *frontend* dela aplikacije 13](#_Toc214747201)

[2.2.1. HTML 14](#_Toc214747202)

[2.2.2. CSS 15](#_Toc214747203)

[2.2.3. JavaScript 17](#_Toc214747204)

[2.2.4. React 19](#_Toc214747205)

[**2.3.** Tehnologije za razvoj *backend* dela aplikacije 21](#_Toc214747206)

[2.3.1. PHP 22](#_Toc214747207)

[2.3.2. Laravel 24](#_Toc214747208)

[2.3.3. MySQL 26](#_Toc214747209)

[2.3.4. phpMyAdmin 28](#_Toc214747210)

[2.3.5. REST API 30](#_Toc214747211)

[2.4. JavniWeb Servisi 32](#_Toc214747212)

[2.4.1. OpenWeather API 33](#_Toc214747213)

[3. IMPLEMENTACIJA VEB APLIKACIJE 35](#_Toc214747214)

[3.1. Faza prikupljanja korisničkih zahteva 36](#_Toc214747215)

[3.1.1. Verbalni opis 37](#_Toc214747216)

[3.1.2. Slučajevi korišćenja 39](#_Toc214747217)

[SK1: Prijava korisnika 42](#_Toc214747218)

[SK2: Registracija korisnika 43](#_Toc214747219)

[SK3: Prikaz javne početne stranice 44](#_Toc214747220)

[SK4: Pregled svih treninga korisnika (bez čuvanja podataka) 45](#_Toc214747221)

[SK5: Pregled vremenske prognoze grada (javni servis) 46](#_Toc214747222)

[SK6: Odjava člana 47](#_Toc214747223)

[SK7: Pregled svojih treninga 48](#_Toc214747224)

[SK8: Kreiranje treninga 49](#_Toc214747225)

[SK9: Izmena postojećih treninga 51](#_Toc214747226)

[SK10: Brisanje treninga 53](#_Toc214747227)

[SK11: Kreiranje nove vežbe 54](#_Toc214747228)

[SK12: Izmena vežbe 56](#_Toc214747229)

[SK13: Brisanje vežbe 58](#_Toc214747230)

[SK14: Pregled vežbi sa filtriranjem i sortiranje 60](#_Toc214747231)

[SK15: Pregled ciljeva 62](#_Toc214747232)

[SK16: Kreiranje novog cilja (za člana) 63](#_Toc214747233)

[SK17: Izmena ciljeva 65](#_Toc214747234)

[SK18: Brisanje ciljeva 66](#_Toc214747235)

[SK19: Pregled svih korisnika 67](#_Toc214747236)

[SK20: Brisanje korisnika 68](#_Toc214747237)

[SK21: Eksportovanje podataka o korisnicima (TXT, CSV) 69](#_Toc214747238)

[3.2. Faza analize 70](#_Toc214747239)

[3.2.1. Sistemski dijagrami sekvence 71](#_Toc214747240)

[DS1: Prijava korisnika 72](#_Toc214747241)

[DS2: Registracija korisnika 75](#_Toc214747242)

[DS3: Prikaz javne početne stranice 77](#_Toc214747243)

[DS4: Pregled treninga kao gost 78](#_Toc214747244)

[DS5: Pregled vremenske prognoze grada (gost) 81](#_Toc214747245)

[DS6: Odjava člana 84](#_Toc214747246)

[DS7: Pregled svojih treninga 87](#_Toc214747247)

[DS8: Kreiranje treninga 91](#_Toc214747248)

[DS9: Izmena postojećeg treninga 97](#_Toc214747249)

[DS10: Brisanje treninga 104](#_Toc214747250)

[DS11: Kreiranje nove vežbe 109](#_Toc214747251)

[DS12: Izmena vežbe 115](#_Toc214747252)

[DS13: Brisanje vežbe 122](#_Toc214747253)

[DS14: Pregled vežbi (filtriranje i sortiranje) 128](#_Toc214747254)

[DS15: Pregled ciljeva 135](#_Toc214747255)

[DS16: Kreiranje novog cilja 139](#_Toc214747256)

[DS17: Izmena cilja 144](#_Toc214747257)

[DS18: Brisanje cilja 149](#_Toc214747258)

[DS19: Pregled svih korisnika (admin) 155](#_Toc214747259)

[DS20: Eksport korisnika (TXT/CSV) 160](#_Toc214747260)

[DS21: Brisanje korisnika (admin) 162](#_Toc214747261)

[3.2.2. Struktura softverskog sistema 166](#_Toc214747262)

[**3.3.** Faza projektovanja 169](#_Toc214747263)

[3.3.1. Projektovanje korisničkog interfejsa (ekstranskih formi) 170](#_Toc214747264)

[Prijava 170](#_Toc214747265)

[Registracija 171](#_Toc214747266)

[Ulazak kao gost 172](#_Toc214747267)

[Početna stranica - Modal 173](#_Toc214747268)

[Početna stranica 174](#_Toc214747269)

[Treninzi 175](#_Toc214747270)

[Vežbe 176](#_Toc214747271)

[Ciljevi 177](#_Toc214747272)

[Korisnici – ADMIN dashboard 178](#_Toc214747273)

[3.3.2. Projektovanje aplikacione logike 179](#_Toc214747274)

[Laravel kontroleri (Controllers) 181](#_Toc214747275)

[Laravel autentifikacija (Authentication) 200](#_Toc214747276)

[Laravel resursi (Resources) 204](#_Toc214747277)

[Laravel Rest API Routes 210](#_Toc214747278)

[Laravel Database Seeder 251](#_Toc214747279)

[React Hooks 257](#_Toc214747280)

[React komponente (Components) 260](#_Toc214747281)

[React Rutiranje (Browser Router) 262](#_Toc214747282)

[Laravel i React 268](#_Toc214747283)

[3.3.3. Projektovanje strukture softverskog sistema 270](#_Toc214747284)

[Modeli 271](#_Toc214747285)

[Migracije 272](#_Toc214747286)

[3.3.4. Projektovanje skladišta podataka 273](#_Toc214747287)

[**3.4.** Faza implementacije 273](#_Toc214747288)

[4. ZAKLJUČAK 273](#_Toc214747289)

[5. LITERATURA 273](#_Toc214747290)

# UVOD

Dokumentacija za Rebel Fitness aplikaciju predstavlja sveobuhvatan tehnički i konceptualni pregled razvoja jedne savremene full-stack veb platforme namenjene praćenju treninga, planiranju fizičkih aktivnosti i administraciji korisnika. U uvodnom delu prikazujemo osnovnu ideju projekta, korišćene tehnologije, arhitektonske principe i biznis model aplikacije, dok kasnija poglavlja detaljno obrađuju implementaciju backend i frontend sloja, interakcije između komponenti i ključne funkcionalnosti.

Sa metodološkog stanovišta, razvoj aplikacije oslanja se na iterativni pristup implementaciji funkcionalnosti — počev od definisanja zahteva i korisničkih scenarija, preko projektovanja API strukture i baze podataka, pa sve do izgradnje interfejsa i završnih integracionih testova. Posebna pažnja posvećena je jasnoj podeli odgovornosti između serverske i klijentske strane, kao i doslednom poštovanju REST arhitekture, što je omogućilo jednostavno proširenje sistema i njegovu modularnost.

Na serverskoj strani primenjen je **Laravel** framework, koji obezbeđuje strukturiran MVC pristup, rad sa migracijama, validacijama i Eloquent ORM-om. Autentifikacija je realizovana putem **Sanctum** tokena, a sve funkcionalnosti — uključujući rad sa nalozima, treninzima, vežbama, ciljevima i administracijom korisnika — izložene su kroz jasan skup API ruta. Te rute dokazano funkcionišu kroz pripremljenu Postman kolekciju (login, registracija, CRUD operacije, admin sekcija, vežbe, treninzi i ciljevi), čime je osigurano testiranje i konzistentno ponašanje aplikacije tokom razvoja.

Frontend deo projekta izgrađen je korišćenjem **React** biblioteke i **Vite** alatke, što donosi brz razvojni ciklus i modularnu organizaciju komponenti. Navigacija se odvija putem React Router-a, dok je autentifikacija centralizovana kroz AuthContext i prilagođene zaštitne rute (ProtectedRoute, RoleRoute). UI je prilagođen modernom tamnom dizajnu sa narandžastim akcentima, fokusirajući se na preglednost, jednostavnost i responzivnost. Pored osnovnih komponenti, aplikacija uključuje i posebne funkcionalne celine — kao što su forme, modalni prozori, paginacija, filtriranje, sortiranje i export podataka (CSV/TXT).

Poseban segment aplikacije predstavlja integracija javnog servisa za vremensku prognozu preko backend proxy rute /weather/{city}, čime se daje primer komunikacije sa eksterne API-je i njihove inkorporacije u interfejs. Ova funkcionalnost kasnije je dodatno unapređena izdvajanjem u custom React hook, što doprinosi modularnosti i ponovnoj upotrebljivosti logike.

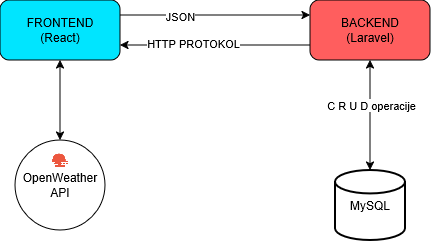
Biznis model Rebel Fitness aplikacije predstavlja kombinaciju korisnički vođenog pristupa i administrativne kontrole. Sistem omogućava rad gostima (read-only režim), registrovanim članovima (upravljanje treninzima, vežbama i ciljevima) i administratorima (upravljački panel za korisnike i sadržaj). Ovakav višeslojni pristup omogućava jednostavno širenje aplikacije u smeru premium pretplata, naprednih statistika, trenerskih naloga ili integracija sa eksternim uređajima.

U nastavku dokumentacije, poglavlja 2–7 razrađuju pomenute funkcionalnosti:

* **Analizu zahteva**, sa detaljnim opisom korisničkih uloga i scenarija korišćenja.
* **Projektovanje serverskog dela**, uključujući API rute, modele, migracije, validacije i Sanctum mehanizam.
* **Projektovanje klijentskog dela**, kroz strukturu React komponenti, hookova, konteksta i vizuelnih modula.
* **Prikaz ekrana aplikacije**, uz objašnjenje dizajnerskih odluka i funkcionalne logike svakog pogleda.
* **Tehničke smernice za pokretanje aplikacije**, uključujući instalaciju dependencija, migracije, seedovanje i razvojne komande.
* **Zaključak**, u kome se sumiraju rezultati rada i definišu smernice za dalji razvoj sistema.

Ovakav pristup omogućava ne samo razumevanje načina na koji je Rebel Fitness tehnički implementiran, već i širu sliku o tome kako se savremene veb aplikacije dizajniraju, organizuju i održavaju u praksi.

# METODE I TEHNOLOGIJE ZA RAZVOJ VEB APLIKACIJE

Fullstack web aplikacija Fitness App zasnovana je na modernoj arhitekturi klijent-server, gde React + Vite čine bogat i responzivan frontend, dok Laravel obezbeđuje robusni REST API i logiku poslovnih pravila na backendu.

Slika 1 - Fullstack Web Aplikacija

Komunikacija između frontend i backend sloja u *Rebel Fitness* aplikaciji odvija se u potpunosti kroz **RESTful JSON API**, uz jasno definisane HTTP metode koje odražavaju namenu svake operacije.  
Backend vraća standardizovane JSON odgovore, dok frontend putem Axios klijenta šalje zahteve i obrađuje rezultate. U sklopu aplikacije koriste se sledeći tipovi zahteva:

* **GET** — dohvat lista i pojedinačnih resursa (korisnici, treninzi, vežbe, ciljevi, vremenska prognoza)
* **POST** — kreiranje novih zapisa (registracija, login, workouts, exercises, goals)
* **PUT / PATCH** — izmena postojećih entiteta (uređivanje vežbi, ciljeva, korisnika, workouta)
* **DELETE** — brisanje resursa (uklanjanje vežbe, cilja ili korisnika iz admin panela)

Na strani klijenta, aplikacija koristi **React Router** za navigaciju i **Axios** za sve HTTP operacije. Tokom autentikacije backend preko Laravel Sanctum mehanizma generiše **token** koji se čuva u localStorage i automatski prosleđuje svim narednim zahtevima.  
Axios instanca automatski ubacuje zaglavlje: Authorization: Bearer <token>  
to omogućava backend middleware-u auth:sanctum da proveri identitet korisnika i njegova prava pristupa (admin, member, guest).

Frontend je organizovan kroz **modularne React komponente**, a poslovna logika autentifikacije izdvojena je u *AuthContext* i custom hook useAuth(). Time se omogućava centralizovano upravljanje tokenom, korisničkim podacima i metodama kao što su login, register, loginGuest i logout. UI se oslanja na ručno stilizovane komponente (Button, Card, Modal, SelectInput, TextInput), uz tamni vizuelni stil i orange akcent boju, što daje prepoznatljiv identitet aplikacije.

Backend deo sistema izgrađen je u **Laravel 10** okviru, koristeći Eloquent ORM za rad sa bazom. Sistemski modeli i njihovi odnosi uključuju:

* **User**, sa atributima role i fitness\_level
* **Workout**, povezan sa korisnikom (user\_id)
* **Exercise**, vezan za workout i korisnika
* **Goal**, povezan sa korisnikom i statusom (pending/completed)

Svi ovi resursi imaju pripadajuće kontrolere (UserController, WorkoutController, ExerciseController, GoalController) i kompletan set CRUD operacija. Validator i middleware sloj obezbeđuju sigurnost i doslednost podataka, dok se JSON resursi i Laravel kolekcije koriste za formatiranje izlaza.

Podaci se čuvaju u **MySQL bazi**, a struktura tabele dosledno prati definiciju modela:  
users, workouts, exercises, goals, password\_resets, personal\_access\_tokens i druge pomoćne tabele koje Laravel automatski generiše.  
Migracije i seederi omogućavaju lako podizanje sistema prilikom lokalnog razvoja i testiranja.

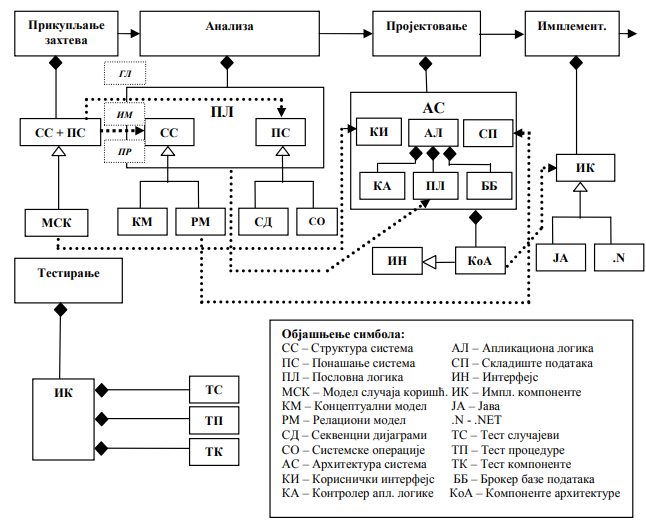
Kompletna interakcija frontend–backend sloja testirana je kroz Postman kolekciju (koju si dostavio), pokrivajući login, registraciju, guest ulaz, manipulaciju workoutima, pristup admin resursima i rad sa vežbama i ciljevima. Time je potvrđena stabilnost API-ja i usklađenost React aplikacije sa backend logikom.

Arhitektura je organizovana tako da frontend i backend funkcionišu kao odvojene celina, što omogućava nezavisan razvoj i lako buduće skaliranje. React može biti distribuiran kao statički build preko Vercel/Netlify servisa, dok Laravel backend može biti hostovan na VPS, Laravel Forge ili drugoj PHP platformi. Ovakav pristup donosi fleksibilnost i jednostavnost održavanja.

U celini, *Rebel Fitness* predstavlja savremeni primer full-stack aplikacije izgrađene prema dobrim praksama: sigurni token-based login, čisti REST API, responzivan UI, jasno razdvojen frontend/backend i mogućnost lakog proširenja (npr. dodavanje statistika, trenerskih naloga, premium modula ili integracije sa spoljnim API servisima). Ovakva arhitektura čini aplikaciju održivom, preglednom i spremnom za dalji razvoj.

## Uprošćena Larmanova metoda

**Uprošćena Larmanova metodologija** razvoja softvera pruža jasan putokaz kroz ceo životni ciklus razvoja jednog sistema, od prvobitne ideje do konačnog testiranja. U svojoj srži, ova metodologija se zasniva na ideji iterativnog i inkrementalnog razvoja, gde se svaka faza gradi na temeljima prethodne, ali se istovremeno stalno vraća u prethodne korake radi eventualnih ispravki ili dopuna. Time se obezbeđuje fleksibilnost i prilagodljivost promenama zahteva, što je naročito važno u današnjem dinamičnom okruženju (Vlajić, 2020).



Slika 2 - Uprošćena Larmanova metoda

**Prva faza, prikupljanje zahteva (engl. requirements elicitation)**, fokusira se na razumevanje poslovnih potreba i očekivanja krajnjih korisnika. Tijekom ovog koraka, tim za razvoj intenzivno razgovara sa stakeholderima kako bi identifikovao glavne funkcionalnosti koje sistem mora da podrži, ali i nefunkcionalne karakteristike kao što su performanse, bezbednost i interoperabilnost. Rezultat ove faze je model slučajeva korišćenja (use-case model), koji opisuje akcije koje korisnici mogu da izvrše u sistemu, ko su ti korisnici (aktori) i kako se odvijaju glavni i alternativni scenariji. Ovaj model služi kao komunikacioni most između tehničkog tima i poslovnih korisnika, obezbeđujući da su zahtevi precizno i sveobuhvatno zabeleženi (Vlajić, 2020).

U **fazi analize**, konceptualni model se prevodi u viši nivo tehničke apstrakcije. To znači da se na osnovu use-case scenarija definišu detaljni sekvencijalni dijagrami koji prikazuju tokove poruka između objekata sistema i pozive sistemskih operacija. Paralelno s tim, formira se konceptualni (domen) model koji otkriva klasu-po-klasu strukturu podataka i njihove međusobne veze – na primer, odnose između entiteta „Korisnik“, „Zahtev za odmor“ i „Odeljenje“. Ovaj korak oslikava logičku arhitekturu sistema, pripremajući tlo za dalji dizajn, a istovremeno otkriva eventualne praznine ili nejasnoće u definisanim zahtevima (Vlajić, 2020).

**Projektovanje** u Larmanovoj metodologiji podrazumeva razgradnju logičke arhitekture na tri senzibilna sloja: korisnički interfejs, poslovnu logiku i skladište podataka. U ovoj fazi definišu se konkretne komponente – ekranski obrasci, kontroleri korisničkog interfejsa, klijentski i serverski sloj poslovne logike, kao i sloj za pristup bazi podataka. Poseban akcenat je na pravilnoj separaciji odgovornosti: UI sloj rukuje prikazom i korisničkim interakcijama, aplikacioni sloj orkestrira operacije i sinhronizuje domen objekte, dok je DB sloj zadužen za perzistenciju i transakcijsku doslednost. Na taj način se omogućava da se promena u jednom sloju minimalno odrazi na ostale (Vlajić, 2020).

Kada je arhitektura definisana, prelazi se u **fazu implementacije** u kojoj se kreiraju stvarne klase, moduli i biblioteke prema šablonima iz projektne dokumentacije. Kod se piše na odabranom jeziku (npr. Java, PHP, C#) prateći principe objektno-orijentisanog dizajna i koristeći mehanizme upravljanja zavisnostima, testiranja jedinica i alate za kontinuiranu integraciju. Za svaku sistemsku operaciju razvijaju se jedinični testovi, a modul za pristup podacima se implementira tako da apstraktno preuzme sve promene u strukturi baze i izvršava ih kroz generičke procedure (Vlajić, 2020).

Na kraju, sve komponente se integrišu i sistem prihvatanja se podvrgava **sveobuhvatnom testiranju**: od funkcionalnog, preko integracionog, do sistemskog i prihvatnog testiranja. Cilj je proveriti da li aplikacija ispunjava definisane zahteve i da li je pouzdana u realnim uslovima rada. Greške ili neusaglašenosti se vraćaju u ranije faze, gde se zahtevi, analiza ili dizajn revidiraju, dopunjavaju i koriguju. Ova faza nije neophodna u okviru naseg projekta.

## Tehnologije za razvoj *frontend* dela aplikacije

U ovom poglavlju predstavljamo osnovne tehnologije i alate koji čine **frontend sloj** naše aplikacije, odnosno korisnički interfejs kojim se direktno interaguje. Svaka od ovih tehnologija ima svoju ulogu, ali zajedno omogućavaju gradnju responzivnih, dinamičnih i vizuelno konzistentnih aplikacija.

**HTML (HyperText Markup Language)** predstavlja temelj svakog web interfejsa. On definiše semantičku strukturu stranica – naslove, paragrafе, tabele, forme i sve druge elemente koje korisnik vidi i sa kojima komunicira. Korišćenjem HTML5 standarda dobijamo mogućnost za napredne semantičke oznake (<header>, <nav>, <article>, <section>, <footer>), ugradnju multimedijalnih sadržaja (<video>, <audio>), kao i optimizaciju pristupačnosti i SEO (npr. pomoću atributa aria- i semantičkih elemenata).

**CSS (Cascading Style Sheets)** je jezik koji definiše izgled i stilizaciju HTML elemenata. U našem projektu koristimo moderni CSS3, uključujući fleks-boks (flexbox), mrežni raspored (grid), tranzicije i animacije, varijable za boje i postavke, kao i responzivne merače (media queries). Zahvaljujući CSS-u, možemo prilagoditi dizajn različitim uređajima i veličinama ekrana, kreirati konzistentne teme i primeniti vizuelne efekte poput senki, zaobljenja ivica i gradijenata.

**JavaScript** je programski jezik koji omogućava dinamičku manipulaciju sadržajem stranica i komunikaciju sa backendom putem AJAX/Fetch poziva. U React okruženju koristimo JS (ili TypeScript) za upravljanje stanjem aplikacije, obradu korisničkih događaja (klikova, unosa u forme) i asinhrono učitavanje podataka. JavaScript runtime u browseru izvršava logiku komponenti, menja DOM strukturu i brine o interaktivnosti – na primer, validaciji polja u formularima, otvaranju modala ili paginaciji.

**React** je biblioteka za izgradnju korisničkih interfejsa zasnovana na komponentnom pristupu. Svaka komponenta enkapsulira deo UI-a i njegovu logiku, što omogućava ponovnu upotrebu koda, jasnu separaciju odgovornosti i lakše održavanje. React koristi virtuelni DOM, što ubrzava renderovanje promena i čini aplikaciju responzivnijom. U našem stacku, React Router upravlja navigacijom, a Context/Redux ili React Query služe za deljenje podataka između komponenti i keširanje API odgovora.

Kombinacija ove četiri tehnologije čini naš frontend moćnim: HTML i CSS brinu o strukturi i izgledu, JavaScript o interaktivnosti, React o arhitekturi i upravljanju stanjem. Zajedno omogućavaju razvoj modernih web aplikacija robusnih performansi, visokog nivoa korisničkog iskustva i lake proširivosti.

### HTML

**HTML (HyperText Markup Language)** je standardizovani jezik za opisivanje strukture i sadržaja veb stranica. On definiše kako će se tekst, slike, linkovi, tabele, forme i drugi elementi rasporediti i prikazati u pregledaču. Svaki HTML dokument je niz „oznaka” (tagova) koje se nalaze unutar uglastih zagrada, na primer <h1>, <p> ili <img>. Tagovi obično dolaze u paru — otvoreni tag <p> započinje paragraf, a zatvoreni </p> ga zatvara — mada postoje i samozatvarajući tagovi, kao što je <br /> za novi red ili <img /> za ugradnju slike (W3Schools, 2025).

Od svog nastanka početkom 1990-ih, HTML se znatno razvio; danas je u upotrebi HTML5, koji uvodi mnoge semantičke elemente i napredne mogućnosti. Semantički tagovi poput <header>, <nav>, <section>, <article> i <footer> ne služe samo za formatiranje, već programerima i pretraživačima daju informaciju o ulozi svakog dela stranice. Ovo značajno poboljšava pristupačnost (a11y) i optimizaciju za pretraživače (SEO), jer alati za čitanje ekrana i web crawler-i mogu bolje razumeti strukturu sadržaja (W3Schools, 2025).

Osim strukture, HTML omogućava ugradnju multimedijalnih sadržaja: <video> i <audio> tagovi od HTML5 standarda direktno integriraju reprodukciju videa i zvuka, bez potrebe za dodatnim plug-inovima. Forme (<form>, <input>, <select>, <textarea>) pružaju mehanizme za unos podataka od strane korisnika, dok atributi kao što su required, pattern ili placeholder olakšavaju osnovnu validaciju i korisničko iskustvo (W3Schools, 2025).

Slika 3 - HTML logo

Važan aspekt HTML-a je i mogućnost proširenja putem atributa i ARIA specifikacija za pristupačnost. ARIA (Accessible Rich Internet Applications) atributi poput role, aria-label i aria-describedby dodatno pomažu slepim i slabovidim korisnicima — omogućavajući im da asistivni softver precizno interpretira elemente stranice. Na taj način pravilno strukturiran HTML ne samo da kreira lep i funkcionalan interfejs, već ga čini i dostupnijim svima (W3Schools, 2025).

U modernom razvoju veb aplikacija, HTML se često kombinuje sa CSS-om za stilizaciju i JavaScript-om za interaktivnost. Ipak, bez jasne i semantičke HTML osnove, ni najsofisticiraniji stilovi ili skripte ne mogu ispravno funkcionisati. Zato se pri izgradnji bilo kakvog veb projekta prvo posvećujemo čistom, validnom i pristupačnom HTML-u, jer on predstavlja temelj na kojem će sve ostalo biti izgrađeno.

* + 1. CSS

**CSS (Cascading Style Sheets)** je jezik za opisivanje izgleda i formatiranja HTML dokumenata, razvijen kako bi se razdvojila struktura (HTML) od prezentacije (stilizacije). Dok HTML definiše elemente stranice — naslove, paragrafe, tabele, linkove i sl., CSS omogućava kontrolu boja, fontova, rasporeda, razmaka i vizuelnih efekata. Formatom „cascading” (kaskadnog) stilizovanja, CSS primenjuje pravila u slojevima: globalni stilovi, stilovi specifični za određene sekcije i inline stilovi unutar samog HTML-a kombinuju se prema pravilima prioriteta kako bi se postigla željena konačna prezentacija (W3Schools, 2025).

Osnovni gradivni blok CSS-a su selektori i deklaracije. Selektor određuje na koje HTML elemente će se primeniti sadržano pravilo, na primer p, .kartica ili #glavni-naslov, dok unutar vitičastih zagrada pišu deklaracije u obliku svojstvo: vrednost; — npr. color: #333; font-size: 16px;. Postoji ogroman broj svojstava (boje, margine, padding, širine, visine, pozicioniranje i dr.), a mnogi su podržani kroz napredne mehanizme poput CSS promenljivih (custom properties) koje omogućavaju definisanje vrednosti koje se mogu promeniti u toku razvoja i lakše održavati (W3Schools, 2025).

Jedan od ključnih koncepta u CSS-u je „box model” — svaki HTML element percipira se kao kutija (box) koja se sastoji od sadržaja (content), unutrašnjeg razmaka (padding), ivice (border) i spoljašnjeg razmaka (margin). Razumevanje box modela je ključno za precizno pozicioniranje elemenata i kontrolu njihovog međusobnog razmaka. Danas se, pored klasičnih tehnika float/clear, koriste moderne metode za raspoređivanje elemenata poput Flexboxa i CSS Grid-a — prvi olakšava izradu jednorednih ili višerednih rasporeda sa prilagodljivim odnose veličina, dok drugi pruža dvodimenzionalni sistem mreže za kompleksnije layout-e (W3Schools, 2025).

Slika 4 - CSS Logo

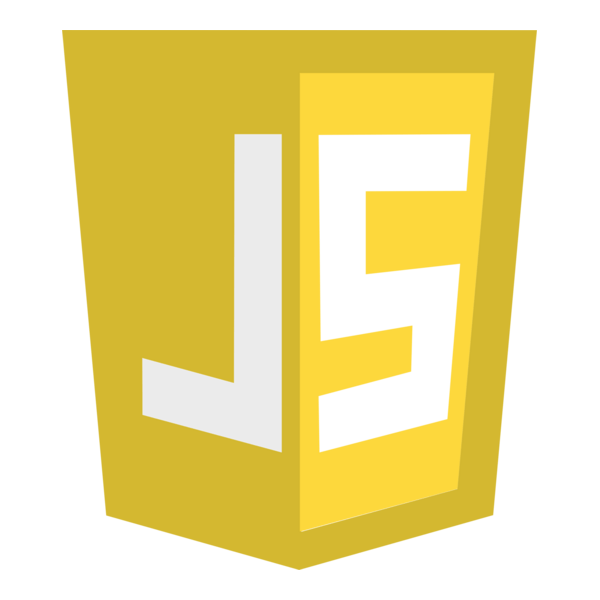
Responsive dizajn, odnosno prilagođavanje prikaza na različitim veličinama ekrana, postiže se CSS medijskim upitima (media queries). Developer može definisati pravila koja važe samo kada je širina ekrana ispod ili iznad određene granice — npr. za mobilne telefone, tablete ili desktop računare. Na taj način jedna te ista stranica može imati vertikalno štampani prikaz na malim ekranima i širok više-kolumni raspored na velikim ekranima, bez potrebe za zasebnim HTML verzijama (W3Schools, 2025).

Savremene tehnike uključuju i korišćenje CSS preprocessora (Sass, Less) i sistema za komponente (CSS Modules, Styled Components u React okruženju), koji dodaju mogućnost promenljivih, ugnježdenih pravila, mešavina (mixins) i drugih programerskih konstrukcija. Time se smanjuje dupliranje koda i poboljšava održivost velikih stilskih datoteka. Osim toga, alati za automatsko prefixovanje (Autoprefixer) pomažu u osiguranju kompatibilnosti sa različitim pregledačima, dok moderne biblioteke i framework-ovi (npr. Chakra UI, Tailwind CSS) donose unapred definisane stilove i UI komponente, ubrzavajući razvoj interfejsa (W3Schools, 2025).

Ukratko, CSS je neizostavan deo svakog veb projekta: on pretvara osnovni HTML kostur stranice u prijatno i konzistentno korisničko iskustvo. Dobro organizovan, modularan i responzivan CSS omogućava brzu nadogradnju, laku prilagodbu brend smernicama i visok nivo pristupačnosti, čineći konačni proizvod estetski privlačnim i funkcionalnim na svim uređajima.

* + 1. JavaScript

**JavaScript** je višenamenski, visoko dinamički jezik koji se izvodi u okviru veb pregledača (browser) ili na serverskoj strani putem okvira kao što je Node.js. Nastao je 1995. godine kao “skriptni” jezik za interakciju unutar HTML stranica, ali danas je ključni stub modernog web razvoja, gde pokreće kompletnu logiku interfejsa, upravljanje stanjima aplikacije i komunikaciju sa backend servisima. JavaScript podržava funkcionalni, imperativni i objektno-orijentisani stil programiranja, s tim da su funkcije građevni blok jezika i često se koriste kao “prvi građani” (first-class functions) za kreiranje callbackova, closure-a i višeg nivoa apstrakcije (MDN, 2025).

Iako je u početku JavaScript bio tumačeni („interpreted”) jezik, današnji JS engine-i (V8 u Chrome-u, SpiderMonkey u Firefox-u, JavaScriptCore u Safari-ju) koriste kombinaciju kompilacije u letu (JIT – Just-In-Time compilation) i optimizovanih izvršnih mašina. Kada se JS fajl učita, engine ga prvo lonksuje i parcira u apstraktno sintaksičko stablo (AST), zatim vrši osnovnu obradu i prevodi hot-path funkcije u mašinski kod, optimizujući ih na osnovu učestalosti njihovog poziva. Time se postiže brzina slična klasično kompajliranim jezicima, ali i fleksibilnost dinamičkog tipiziranja te mogućnost da se kod menja ili učitava tokom rada aplikacije (MDN, 2025).

Slika 5 - JS Logo

Na nivou izvršenja, JavaScript je jednodeljni (single-threaded), što znači da se svi operativni zadaci—parsiranja, izvršenja koda, događaja iz DOM-a i mrežnih zahteva—obrađuju u jednom glavnom “thread-u”. Kako bi se izbeglo blokiranje korisničkog interfejsa pri dugotrajnim operacijama (npr. mrežni poziv, čitanje fajla, tajmer), JS koristi asinhroni model zasnovan na event loop-u. Kada napišemo asinhroni zadatak (putem fetch(), setTimeout(), Promise-a ili async/await sintakse), taj zadatak se delegira veb API-ju ili Node.js sistemu koji ga izvršava nezavisno, a rezultat se vraća kroz callback koji se stavlja u red “task-ova”. Glavna petlja potom izvlači te callbackove iz reda i izvršava ih kada JavaScript stack postane slobodan (MDN, 2025).

Suprotno asinhronom, sinhrono izvršavanje znači da se svaki red izraza i funkcija obrađuje jednom za drugim i da se dalji kod ne pokreće dok se tekuci izraz ne završi. Ovo ponašanje može dovesti do “zamrzavanja” interfejsa ako se unutar glavnog steka vrše zahtevne ili blokirajuće operacije—zato je razumevanje granica sinhronog koda bitno za optimizaciju interaktivnosti (MDN, 2025).

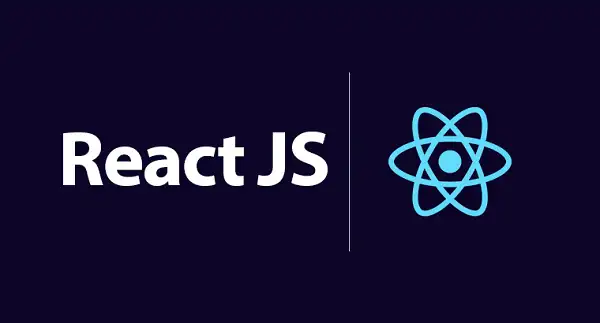
**Model asinhronog izvršenja u JS-u** komplikuje rad sa redosledom operacija, ali nudi i **standardizovane alate za upravljanje složenim tokovima** (MDN, 2025):

* **Callback funkcije:** osnovna tehnika, ali može dovesti do “callback hel” struktura.
* **Promise objekti:** omogućavaju da asinhroni zadaci vrate objekt na koji se možemo “pretplatiti” metodama .then(), .catch() i .finally(), čime se dobija linearniji tok koda.
* **async/await:** šećer sintakse nad Promise-ima — funkcija označena async vraća Promise, a await pauzira izvršenje sve dok se zadati Promise ne reši, zadržavajući čitljivost i strukturiranost kôda sličnu sinhronom pisanju.

Osim ovih, JavaScript donosi i napredne paradigme poput generatora (funkcija koje mogu pauzirati i nastaviti izvršenje više puta), event emitera i web workers (u browser-u) ili worker threads (u Node.js-u), koji omogućavaju paralelnu obradu van glavnog steka radi teških računskih zadataka. Dodatno, zbog svoje event-driven prirode, JavaScript je idealan za real-time aplikacije (čet servisi, live notifikacije) gde se stalno osluškuje i obrađuje veliki broj događaja (MDN, 2025).

Ukratko, JavaScript kombinuje brzinu i optimizacije modernih JIT kompajlera sa asinhronim, event-driven modelom izvršenja. Razumevanje razlike između sinhronog i asinhronog koda, mehanizama event loop-a i naprednih konstrukata kao što su Promise-i i async/await omogućava pisanje efikasnih, responzivnih aplikacija koje se ne blokiraju i lako skaliraju u suočavanju sa zahtevnim mrežnim i I/O operacijama.

* + 1. React

**React** je nastao 2013. godine unutar **Facebook** tima kao odgovor na izazove skalabilnog upravljanja kompleksnim korisničkim interfejsima; prepoznat je kao biblioteka koja uvodi deklarativni pristup izgradnji komponenti kako bi se olakšalo održavanje i nadogradnja velikih aplikacija. Umesto da ručno manipulišemo HTML strukturom i ručno pratimo promene u DOM-u, u React-u opisujemo krajnji izgled korisničkog interfejsa za svako stanje aplikacije, a biblioteka se brine o optimalnom ažuriranju prikaza (Meta, 2025).

Slika 6 - ReactJs Logo

**JSX** je sintaksni šećer nad JavaScript-om koji nam omogućava pisanje HTML-sličnog markupa direktno unutar JavaScript fajlova. JSX komponente predstavljaju malu, inkapsuliranu jedinicu UI-a koja prima ulazne parametre (props), održava sopstveno stanje (state) i vraća opis virtuelnog DOM-a. Iako izgleda kao kombinacija HTML-a i JavaScript-a, JSX se na kraju prevodi u pozive JavaScript funkcija, što omogućava bogatu i dinamičnu kontrolu nad sadržajem i ponašanjem komponenata (Meta, 2025).

**Koncept *reusable* komponenti** je u srcu React filozofije: umesto pisanja ponavljajućih HTML šablona i logike, kreiramo generičke komponente koje prihvataju parametre za prilagođavanje stila, teksta, događaja ili ponašanja. Na taj način možemo jednom razvijenu komponente koristiti na više mesta unutar aplikacije; to donosi konzistentnost dizajna i drastično smanjuje količinu ponovljenog koda. Svaka komponenta je samostalna — promena unutar nje ne utiče na ostale delove aplikacije ukoliko su jasno odvojeni odgovornosti (Meta, 2025).

**React Hooks** su uvedeni 2019. godine kako bi se funkcionalnim komponentama omogućilo upravljanje stanjem i korišćenje životnih ciklusa bez potrebe za pisanjem klasa. Najosnovniji Hooks, kao što su useState i useEffect, omogućavaju nam da u komponenti definišemo promenljive koje pokreću ponovno renderovanje i da reagujemo na promene vrednosti ili spoljnih resursa. Koristeći Hooks, logiku možemo grupisati po temi — npr. logiku forme, autentikacije ili mrežnih poziva — što rezultira čišćim i lako testiranim kodom (Meta, 2025).

***Custom Hooks*** nadograđuju ovu ideju tako što omogućavaju kreiranje sopstvenih funkcija koje u sebi kombinuju standardne Hooks i poslovnu logiku. Umesto da u svakoj komponenti dupliramo kod za učitavanje podataka ili validaciju polja, definišemo jedinstveni Custom Hook koji vraća kompletan set vrednosti i funkcija za taj konkretan problem. Komponente potom samo „upućuju” na taj hook i brišu iz sebe svu poslovnu logiku, ostavljajući samo UI deo (Meta, 2025).

**React DOM** je biblioteka koja implementira vezu između React-ovog virtuelnog DOM-a i stvarnog browser DOM-a. Virtuelni DOM je laka JavaScript struktura u memoriji koja čuva predstavu UI-a, a svakom promenom stanja React kreira novu verziju i računa minimalan broj izmena potrebnih u stvarnom DOM-u. Ovo difovanje (diffing) i pakovanje izmena u batch operacije čini renderovanje izrazito brzim u poređenju sa direktnim, sinhronim manipulacijama DOM-om (Meta, 2025).

Među **prednostima React-a** izdvajaju se visoka performansa zahvaljujući virtuelnom DOM-u, razvijen ekosistem alata i biblioteka te jednostavno komponovanje i deljenje koda. Za razliku od klasičnih MVC rešenja, React donosi jasniju separaciju odgovornosti i lakše testiranje. Mane mogu biti složenija kriva učenja za početnike (posebno zbog Hooks pravila) i potreba za dodatnim bibliotekama pri upravljanju složenim stanjem ili rutiranjem (Meta, 2025). Ipak, zbog svoje modularnosti i bogatog community-ja, React ostaje jedan od najpopularnijih izbora za razvoj modernih web aplikacija.

## Tehnologije za razvoj *backend* dela aplikacije

U okviru **backend sloja** moderne web aplikacije, ključnu ulogu igraju tehnologije koje omogućavaju pouzdanu, skalabilnu i bezbednu poslovnu logiku, rukovanje podacima i komunikaciju sa klijentom putem standardizovanih interfejsa. Tri osnovna stuba na kojima počiva razvoj server–side dela HR App-a su: jezik PHP, MVC framework Laravel i RESTful API pristup. Svaka od ovih komponenti donosi svoje prednosti i olakšava implementaciju glavnih funkcija sistema — od autentikacije, preko upravljanja korisnicima i podacima o odmorima, do generisanja izveštaja i statistika.

**PHP** je jedan od najraširenijih jezika za razvoj dinamičkih web aplikacija. Njegova prvenstvena prednost leži u velikoj zajednici, bogatom ekosistemu biblioteka i dugogodišnjoj stabilnosti na produkcionim serverima. PHP omogućava brzo prototipiranje i lako integrisanje sa popularnim bazama podataka poput MySQL-a ili PostgreSQL-a. U HR App-u, PHP obrađuje sve zahteve koje klijent šalje, proverava privilegije korisnika, validira ulazne podatke i poziva odgovarajuće modele za učitavanje ili izmenu zapisa u bazi.

**Laravel** je PHP framework koji se gradi po MVC (Model-View-Controller) paradigmi, što donosi strogu organizaciju koda i jasnu separaciju odgovornosti. Controller-i primaju HTTP zahteve, pozivaju Service ili Model sloj za obradu poslovne logike, a zatim vraćaju JSON odgovore (putem Resource klasa) ili usmeravaju na odgovarajuće biblioteke za eksport u PDF. Laravel dolazi sa mnogim ugrađenim alatima — od ORM sistema Eloquent za rad sa relacionim bazama, preko sistema migracija za verzionisanje šeme podataka, do sigurnosnih mehanizama kao što su CSRF zaštita i enkripcija sesijskih kolačića.

**REST API** predstavlja stil arhitekture putem kojeg frontend komunicira sa backendom na dosledan i stateless način. Svaki resurs (npr. korisnik, zahtev za odmor, performance review) izlaže se kroz URL endpoint-e, a standardne HTTP metode (GET, POST, PUT/PATCH, DELETE) omogućavaju CRUD operacije nad tim resursima. U HR App-u, RESTful pristup garantuje da React aplikacija može da učitava liste zahteva, šalje nove, ažurira postojeće ili briše one koje su završene, bez potrebe za ponovnim učitavanjem stranice, dok Laravel sanitizuje i validira svaki zahtev pre izvršenja bilo kakve izmene u bazi.

Kombinacija PHP-a, Laravela i REST API-ja stvara čvrst i dobro strukturisan backend sloj. Ovako dizajniran sistem omogućava lako skaliranje funkcionalnosti, jednostavno proširenje novih modula i održavanje kroz jasno definisane slojeve, dok istovremeno zadovoljava visoke zahteve za performansama, bezbednošću i zajedničkom razvoju unutar timova.

### PHP

**PHP (engl. “PHP: Hypertext Preprocessor”)** je skriptni jezik orijentisan na serversku stranu web aplikacija, koji je nastao još 1994. godine kao jednostavan skup CGI skripti za prikazivanje posećenosti lične stranice njegovog autora Rasmusa Lerdoffa. Ubrzo je preimenovan u PHP/FI, da bi tokom devedesetih evoluirao u robusniji sistem sa podrškom za bazu podataka, rad sa oblikovanim tekstom i osnovne programske konstrukcije. Njegov razvoj je vođen idejom da se u jednom jeziku spoji jednostavnost za izradu dinamičkih HTML stranica sa moćnim serverskim mogućnostima, što je ubrzo rezultiralo velikim prihvatanjem i rastom zajednice (PHP, 2025).

Slika 7 - php Logo

Arhitektura PHP-a zasnovana je na interpretatoru (Zend Engine) koji parsira i izvršava PHP fajlove svaki put kada server primi HTTP zahtev. Svaka PHP skripta se na serveru „prevodi“ u interni opkod (opcode), a zatim izvršava – nakon čega se generiše konačni HTML koji se vraća klijentu. Zahvaljujući Opcache keširanju, često korišćeni opkodi mogu biti sačuvani u memoriji, što značajno ubrzava izvođenje i smanjuje opterećenje servera. Upravo ta fleksibilnost u kombinaciji sa optimizacijama tipa JIT (Just-In-Time) kompilacije u novijim verzijama čini PHP vrlo efikasnim za dinamične web aplikacije (PHP, 2025).

Jedna od najvećih prednosti PHP-a je njegova besprekidna i snažna integracija sa relacionim bazama podataka – prvenstveno MySQL/MariaDB, ali i PostgreSQL, SQLite i brojnim drugim kroz ekstenzije i PDO (PHP Data Objects) sloj. PDO pruža apstrakciju nad različitim driver-ima, pripremljene izjave i transakcije, čime se drastično smanjuje rizik od SQL injekcija i omogućava pisanje čitljivog, održivog koda. Time programeri mogu da grade skalabilne i sigurnosno otporne aplikacije, upravljajući podacima na konsistentan, strukturiran način (PHP, 2025).

PHP svojim modularnim dizajnom podržava ekstenzije napisane u C-u ili drugim nativnim jezicima, pa se lako “vezuje” za biblioteke za obradu slika (GD, Imagick), rad sa JSON-om, XML-om, LDAP-om, Redisom, Memcachedom i mnogim drugim servisima. Zahvaljujući takvom ekosistemu, razvoj naprednih funkcionalnosti poput slanja mejlova, rada sa grafikom, enkripcije ili komunikacije putem web servisa postaje rutina, bez potrebe za pisanjem niskonivo kodova (PHP, 2025).

Kada govorimo o proizvodnji aplikacija, PHP je naročito popularan u MVC (Model-View-Controller) okruženjima; Laravel, Symfony, CodeIgniter i CakePHP su samo neki od framework-a koji donose robustne alate za rutiranje, ORM (Eloquent, Doctrine), validaciju, middleware i automatizovano testiranje. Ti framework-ovi unose disciplinu u pisanje koda, uvođenje servisa i zavisnosti preko Dependency Injection kontenjera, što ubrzava razvoj, poboljšava testabilnost i olakšava održavanje većih projekata (PHP, 2025).

U kontekstu skalabilnosti i deploymenta, PHP aplikacije se mogu pokretati na deljenom hostingu, VPS-u ili kontejnerima (Docker, Kubernetes). Platforme poput Laravel Vapor ili PHP FPM na Nginx/Apache serverima omogućavaju horizontalno skaliranje i dinamičko balansiranje opterećenja. U kombinaciji sa CI/CD alatima (Jenkins, GitHub Actions, GitLab CI), PHP projekti se lako integrišu u pipeline-e za automatsko testiranje, validaciju koda i deploy na produkcione okruženja, čime se znatno smanjuje manuelni rad i rizik od grešaka prilikom puštanja novih verzija (PHP, 2025).

Bezbednost je sveobuhvatna tema u PHP svetu – i iako su ranije verzije patele od propusta zbog neadekvatnih validacija ili zastarelih ekstenzija, danas PHP dolazi sa ugrađenim funkcijama za enkripciju, hash lozinki (password\_hash), filter\_input API-jem za saniranje ulaznih podataka i Strict Types mehanizmom za jaču tipizaciju. Redovna ažuriranja jezika i ekstenzija, kao i veliku pažnju zajednice, dodatno osiguravaju da PHP ostane relevantan i bezbedan izbor za server-side razvoj (PHP, 2025).

Na kraju, najveća vrednost koju PHP donosi leži u njegovoj ogromnoj zajednici, dokumentaciji bogatoj primerima, tutorialima i gotovim paketima na Packagist repozitorijumu. Bilo da se radi o malom blogu ili kompleksnom enterprise rešenju, PHP pruža sve gradivne blokove – od osnova HTTP sesija i kolačića, preko sofisticiranih biblioteka, do modernih devops praksi – čineći ga jednim od najomiljenijih i najčešće korišćenih jezika za kreiranje dinamičkih web aplikacija.

### Laravel

**Laravel** je **PHP framework** otvorenog koda koji je prvi put predstavljen 2011. godine od strane Taylor Otwella, sa ciljem da unapredi i pojednostavi proces izgradnje web aplikacija u PHP-u. Od svog nastanka, Laravel je brzo stekao popularnost zahvaljujući čistoj sintaksi, modularnoj arhitekturi i snažnoj zajednici koja kontinuirano doprinosi ekosistemu paketima, tutorijalima i alatima. Izgrađen na MVC (Model–View–Controller) principima, Laravel odvaja poslovnu logiku od prezentacione logike, što omogućava timovima da paralelno rade na frontend i backend delovima aplikacije uz minimum konflikata i maksimalnu čitljivost koda (Laravel, 2025).

Slika 8 - Laravel Logo

Za razvoj REST API-ja, Laravel nudi niz ugrađenih rešenja koja ubrzavaju i olakšavaju rad. Prvo, sistem rutiranja je izrazito fleksibilan – preko Route::apiResource definisanje standardnih CRUD ruta postaje trivialno, dok middleware sloj omogućava jednostavnu implementaciju autentikacije, autorizacije i zaštite od Common vulnerabilities. Drugo, Laravel-ovi JSON Resources (Resursi) transformišu Eloquent modele u JSON odgovore sa preciznom kontrolom nad formatom izlaza, što garantuje konzistentnost API-ja i olakšava verzionisanje. Pored toga, paketi kao što su Sanctum i Passport pružaju robustan sistem za izdavanje i verifikaciju API tokena ili OAuth2 autorizacije, uz gotove middleware i helper-e za zaštitu ruta (Laravel, 2025).

Eloquent ORM je jedan od najsnažnijih aduta Laravela – omogućava deklarativno i intuitivno definisanje veza između modela (jedan-prema-jednom, jedan-prema-više, više-prema-više), bez potrebe za pisanjem složenih SQL upita. Zahvaljujući “query builder” sloju, programeri mogu lako sastavljati dinamične upite, koristiti transakcije ili pripremljene izjave (prepared statements), što doprinosi sigurnosti i performansama. Kada je u pitanju skalabilnost, Laravel podržava rad sa queue workers (Redis, Beanstalkd, SQS), events i jobs mehanizmima za odvajanje dugotrajnih zadataka poput slanja mejlova ili obrade velikih fajlova u pozadinskim procesima (Laravel, 2025).

Arhitektura zasnovana na servis provajderima (Service Providers) i IoC (Inversion of Control) kontejneru omogućava visoku prilagodljivost i testabilnost aplikacije. Svaka komponenta, bilo da je to mailer, cache, ili custom servis, registruje se kroz provider i može se lako zameniti ili proširiti. Artisan CLI donosi stotine korisnih komandi za generisanje koda (scaffolding), migracije baze, pokretanje testova ili čak kreiranje sopstvenih komandi, što značajno ubrzava razvojni ciklus i automatizuje rutinske zadatke (Laravel, 2025).

U novijim verzijama Laravel je fokus stavio na unapređenje DX (Developer Experience) za API-centric aplikacije: Livewire za sinhrono dvosmerno vezivanje podataka, Jetstream za gotove pakete autentikacije i timskog rada, te Fortify za prilagodljivu sigurnosnu logiku. S druge strane, Laravel Nova predstavlja premium admin panel generator, a Horizon nadzor sistem za Redis queue-e. Ovi dodaci čine da se složeni enterprise sistemi mogu izgraditi i održavati brzo i pouzdano, bez žrtvovanja kvaliteta koda ili performansi (Laravel, 2025).

Sve u svemu, Laravel je kroz deset godina razvoja prerastao u jedan od najkompletnijih PHP framework-ova na tržištu. Pruža balans između jednostavnosti za početnike i dubine za iskusne programere, sa akcentom na elegantan kod, sigurnost, testabilnost i moćne alate za rad sa REST API-jima. Zahvaljujući kontinualnim ažuriranjima i bogatom ekosistemu, Laravel ostaje vodeća platforma za razvoj modernih web aplikacija na serverskoj strani.

* + 1. MySQL

**MySQL** je jedan od najpopularnijih sistema za upravljanje relacionim bazama podataka (RDBMS) otvorenog koda, razvijen početkom 1990-ih od strane švedske kompanije MySQL AB, a danas je u vlasništvu Oracle Corporation. Ono što ga je učinilo privlačnim za široku primenu jeste lakoća instalacije, stabilnost i izuzetno velika zajednica korisnika i programera koja kontinuirano doprinosi poboljšanjima i ekstenzijama. Zbog svoje arhitekture klijent-server, MySQL omogućava da se baza podataka pokreće na jednoj (ili više) serverskoj mašini, dok se aplikacije-klijenti (web serveri, desktop programi ili API slojevi) povezuju sa udaljenog računara, što ga čini veoma pogodnim za moderne distribuirane i cloud okoline (MySQL, 2025).

Slika 9 - MySQL Logo

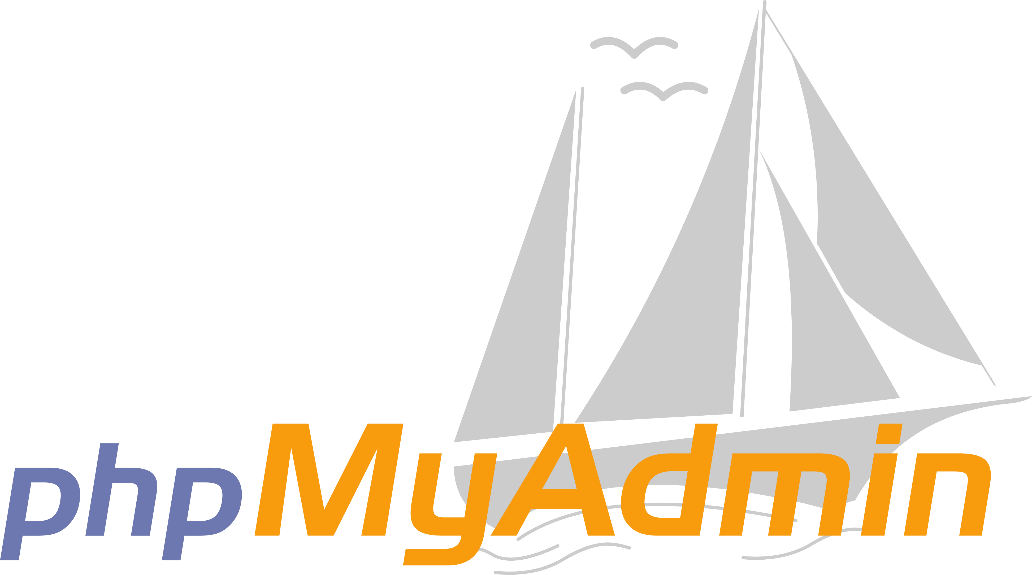
Relacijska priroda MySQL-a znači da se podaci skladište u tabelama međusobno povezanih „ključeva” i „stranih ključeva“, što omogućava doslednost i integritet podataka. Konkretno, podrška za ACID (Atomicity, Consistency, Isolation, Durability) transakcije putem InnoDB skladišnog motora garantuje da su svi podaci uvek u validnom stanju čak i u uslovima konkurentnog pristupa ili u slučaju pada sistema. Transakcije se zatvaraju COMMIT ili ROLLBACK naredbama i prate se visoki nivoi izolacije (npr. REPEATABLE-READ), što je ključno za poslovne aplikacije koje obrađuju plaćanja, porudžbine ili bilo kakve druge kritične podatke (MySQL, 2025).

MySQL takođe odlikuje veliki broj optimizacija za performanse. Automatsko keširanje upita (query cache), indeksi po više kolona, particionisanje tabela i prilagođeni slojevi za optimizaciju izvršenja omogućavaju brzo pretraživanje i filtriranje ogromnih količina podataka. Pored toga, replikacija u master-slave ili master-master konfiguraciji pruža skalabilnost čitanja i visoku dostupnost – u slučaju kvara jednog servera, drugi repliku preuzima ulogu primarnog (failover). Horizontalno skaliranje postiže se pomoću shardovanja (deljenja) podataka po više fizičkih instanci baza (MySQL, 2025).

Upravljanje i administracija MySQL-om je olakšana brojnim alatima: klijent-server konzola (mysql CLI), grafički alati poput MySQL Workbench, te open-source rešenja kao što su phpMyAdmin ili Adminer. Ovi alati omogućavaju vizuelno pregledanje struktura tabela, izradu ER dijagrama, pisanje i testiranje upita, kao i monitoring performansi (slow query log, performance schema). Zahvaljujući podršci za standardni jezik SQL, razvojni timovi mogu lako prebacivati postojeće upite između MySQL-a i drugih relaционих baza bez velikih izmena (MySQL, 2025).

Na kraju, MySQL ima veoma bogat ekosistem dodatnih skladišnih motora: osim InnoDB (podrazumevanog), tu su i MyISAM (za jednostavnije, nekritične scenarije), MEMORY (za privremene radne podatke u memoriji), CSV (za razmenu u CSV formatu), te engine-i za ŠIFROVANJE, GEO prostorne upite ili punotекстualno pretraživanje. Zahvaljujući tome, MySQL se prilagođava specifičnim zahtevima projekta, bilo da je u pitanju high-throughput online igra, analitika u realnom vremenu ili jednostavna CMS aplikacija. Ova kombinacija stabilnosti, performansi, fleksibilnosti i ogromne zajednice čini MySQL jednim od najčešće korišćenih rešenja za backend skladištenje podataka.

* + 1. phpMyAdmin

**phpMyAdmin** je popularni web-bazirani alat otvorenog koda napisan u PHP-u, namenjen za upravljanje MySQL i MariaDB bazama podataka preko preglednika. Razvijen još 1998. godine od strane Tobiasu Ratschillera i potom preuzet i nastavljen od strane zajednice, phpMyAdmin omogućava administratorima i programerima da intuitivno kreiraju, čitaju, ažuriraju i brišu (CRUD) podatke i strukture tabela bez potrebe za ručnim pisanjem SQL naredbi u komandnoj liniji. Interfejs se sastoji od navigacionog stabla levo za pregled baza i tabela, te centralnog panela u kojem se prikazuju detalji i omogućavaju sve tipične operacije kao što su dizajn šeme, eksport/import i izvođenje upita (phpMyAdmin, 2025).

Slika 10 - phpMyAdmin Logo

Jedna od ključnih prednosti phpMyAdmin-a je njegova sposobnost uvoza i izvoza podataka u raznim formatima: SQL dump, CSV, XML, JSON, Excel, YAML i dr. Korisnik može da izvozi kompletne baze ili odabrane tabele, a pri tome ima finu kontrolu nad stilovima izveštaja — kompresija fajlova, uključivanje DROP TABLE naredbi, ili podešavanje kodinga znakova za internacionalizaciju. Uvozni čarobnjaci dozvoljavaju brzo unošenje velikih dataset-ova, sa opcijama automatskog preskakanja duplikata i mappinga kolona, čime se znatno skraćuje vreme migracije podataka (phpMyAdmin, 2025).

phpMyAdmin takođe pruža vizuelne alate za dizajn baze: mogućnost crtanja Entity-Relationship dijagrama direktno iz postojeće šeme, zatim pregleda relacija između tabela i kreiranje indeksa ili stranih ključeva uz nekoliko klikova. Administratori mogu pratiti aktivne procese i sesije, pregledavati i ubijati dugotrajne upite, kao i pratiti statistiku performansi baze, kao što su veličine tabela i broj redova. Ovi alati pomažu da se problemi sa performansama identifikuju brzo, bez dodatnih eksternih monitora (phpMyAdmin, 2025).

Sa sigurnosnog aspekta, phpMyAdmin podržava višestruke metode autentikacije: tradicionalno čuvanje korisničkih podataka u config.inc.php, autentikaciju preko HTTP Basic ili cookie-based mehanizme, kao i LDAP integraciju. Preporučuje se postavljanje phpMyAdmin-a iza HTTPS konekcije i strogo ograničavanje pristupa (npr. .htaccess ili IP bijeljenje), jer alat omogućava kompletan pristup bazi. Takođe, njegove redovne nadogradnje rešavaju otkrivene ranjivosti, zbog čega je važno održavati instalacije ažurnima (phpMyAdmin, 2025).

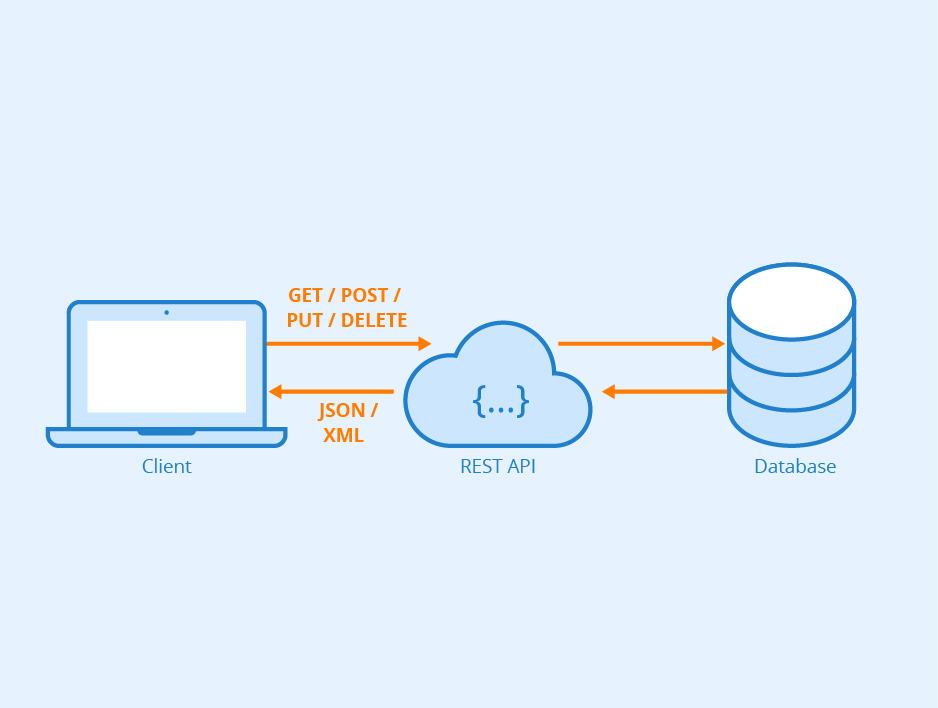
Iako je phpMyAdmin prvenstveno zamišljen za MySQL/MariaDB, zajednica je razvila forkove i dodatke koji omogućavaju rad sa drugim sistemima (PostgreSQL, SQLite) ili naprednim funkcionalnostima (profiler, query builder). Međutim, njegova jednostavnost i široka podrška čine ga i danas standardom u web-hosting okruženjima i među PHP programerima. Za vrlo velike produkcione baze ili enterprise scenarije mogu se preferirati specijalizovana rešenja ili CLI alati, ali za većinu administrativnih i razvojnih zadataka, phpMyAdmin ostaje nezamenjiv, brzo implementiran i lako prilagodljiv alat.

* + 1. REST API

**HTTP protokol** predstavlja temelj komunikacije na svetskoj mreži. Radi se o tekstualno zasnovanom protokolu na sloju aplikacije koji definiše kako klijent–server model razmenjuje poruke. Klijent (najčešće veb pregledač ili neka aplikacija) inicira zahtev (request) slanjem zaglavlja (headers) i, po potrebi, tela poruke (body) na određeni URL. Server prima zahtev, procesuira ga i vraća odgovor (response) koji sadrži statusni kod (npr. 200 OK, 404 Not Found, 500 Internal Server Error), sopstvena zaglavlja (kao što su Content-Type ili Cache-Control) i eventualno telo poruke. HTTP je po dizajnu **stateless** – svaki zahtev se tretira nezavisno, bez pamćenja prethodnih interakcija, što olakšava skaliranje i balansiranje opterećenja jer serveri ne drže sesije između zahteva sami po sebi. Verzije HTTP/1.1 i HTTP/2 donose poboljšanja u performansama (persistentne konekcije, multiplexing, header kompresija), dok HTTP/3 uvodi QUIC transport baziran na UDP-u za bržu i pouzdaniju isporuku paketa (Red Hat, 2020).

Slika 11 - HTTP Internet Protokol

**REST (Representational State Transfer)** je arhitektonski stil definisan od Roy Fieldinga 2000. godine, koji predlaže kako dizajnirati distribuirane hipermedijalne sisteme poput web servisa. Ključna ideja REST-a je izgradnja **resursno orijentisanih** interfejsa: sve što API izlaže tretira se kao resurs identifikovan jedinstvenim URI-jem (npr. /users/123 za korisnika sa ID-em 123). Svaki resurs ima svoje reprezentacije – najčešće JSON ili XML – i kroz HTTP metode (GET, POST, PUT, PATCH, DELETE) klijenti vrše operacije nad tim resursima. GET zahtevi preuzimaju reprezentaciju, POST kreira novi resurs, PUT ili PATCH ažurira postojeći resurs, a DELETE briše resurs. REST zahteva da server bude stateless, da koristi standardne HTTP mehanizme za mehanizme za keširanje (Cache-Control) i da benefituje od hijerarhije URI-ja radi lakše navigacije i razumljivosti (Red Hat, 2020).



Slika 12 - REST Api način funkcionisanja

**Principi REST API-ja** dodatno nalažu upotrebu uniformnog interfejsa, što znači da se sva komunikacija odvija na isti, predvidljiv način, bez skrivanja sopstvenih protokola ili specifičnih RPC poziva unutar HTTP-a. Resursi se mogu međusobno povezivati putem linkova (HATEOAS – Hypermedia As The Engine Of Application State), pružajući klijentima mogućnost da dinamički otkrivaju dostupne akcije kroz odgovore servera. Time se postiže visok stepen fleksibilnosti i samoopisnosti API-ja, jer klijent ne mora da zna unapred ceo skup ruta i operacija – dovoljan mu je početni „ulazni“ URI i hijerarhijski linkovi koje API vraća (Red Hat, 2020).

**Prednosti REST API-ja** ogledaju se u njegovoj jednostavnosti, interoperabilnosti i otpornosti na promene: budući da se oslanja na standardne HTTP metode i formate podataka, REST servisi se lako integrišu sa raznim klijentima—web, mobilnim ili mikrouslugama na drugim jezicima. Stateless priroda REST-a omogućava horizontalno skaliranje bez dodatnih slojeva za pamćenje stanja, a podrška za keširanje HTTP odgovora smanjuje opterećenje servera. Međutim, REST nije uvek najefikasniji za visoko polirane dvosmerne aplikacije u realnom vremenu ili kompleksne transakcije koje zahtevaju više koraka u jednoj sesiji; u takvim slučajevima često se koriste WebSocket-i, gRPC ili GraphQL kao dopuna ili zamena (Red Hat, 2020).

U savremenom razvoju web-aplikacija, REST API-ji preko HTTP-a predstavljaju zlatni standard za sinhronu komunikaciju između frontend i backend sloja. Jasna razgraničenja odgovornosti, predvidljiva struktura poruka i široka podrška u alatima (curl, Postman, Swagger/OpenAPI) čine REST idealnim izborom za većinu CRUD-orijentisanih sistema i mikroservisne arhitekture. Korišćenjem HTTPS protokola, REST API-ji obezbeđuju enkripciju i verifikaciju servera, što doprinosi sigurnosti prilikom razmene osetljivih podataka između klijenta i servera.

* 1. JavniWeb Servisi

**Javni web servisi**, često nazvani public API-ji (Application Programming Interfaces), predstavljaju skup protokola i definisanih metoda kojima se omogućava interakcija sa softverskim komponentama ili podacima nekog sistema sa spoljnog sveta. Umesto da krajnji korisnik direktno pristupa bazi podataka ili internim funkcijama aplikacije, on putem HTTP zahteva (najčešće RESTful GET, POST, PUT, DELETE) komunicira sa unapred definisanim URL endpoint-ima. Svaki endpoint vraća strukturirane odgovore — najčešće u JSON formatu — u skladu sa ugovorenom specifikacijom, čime se postiže jasna separacija između “onoga što” klijent zahteva i “onoga kako” server zapravo radi posao na pozadini.

Jedna od ključnih odlika javnih API-ja jeste da su dostupni široj grupi korisnika ili čak celokupnoj internet zajednici, uz minimalne (ili nikakve) barijere za primenu. Organizacije ih objavljuju kako bi omogućile ekosistem aplikacija trećih strana da se integrišu sa njihovim podacima i funkcionalnostima — na primer, geolokacioni servisi (Google Maps API), platni procesori (Stripe, PayPal), društvene mreže (Twitter, Facebook Graph API) ili vremenske prognoze (OpenWeatherMap). Time vlasnici servisa štede resurse na razvijanju frontend-a i dobivaju mogućnost da eksterne razvijače motivišu da grade nove aplikacije oko njihove platforme.

Uprkos širokoj dostupnosti, javni API-ji podrazumevaju strogu kontrolu pristupa i sigurnosne mere. Autentikacija se često vrši putem API ključeva ili OAuth 2.0 tokova, a primena rate limiting-a i „quota“ sistema sprečava zloupotrebu i preopterećenje servera. HTTPS osigurava enkripciju podataka u tranzitu, dok mehanizmi kao što su CORS (Cross-Origin Resource Sharing) definišu odakle klijenti smeju da šalju zahteve. Dobro dizajnirani public API takođe pruža detaljnu dokumentaciju (često generisanu Swagger/OpenAPI alatima) sa primerima zahteva, odgovora i objašnjenjima potencijalnih grešaka.

Prednost public API-ja ogleda se u brzom razvoju i skalabilnosti ekosistema: treće strane mogu da brzo integrišu funkcionalnosti, grade jedinstvene korisničke tokove i dopunjuju glavnu platformu novim uslugama. Istovremeno, kvalitet API-ja direktno utiče na reputaciju i korisničko iskustvo — nedosledni formati, slaba dokumentacija ili učestale izmene verzija mogu da izazovu frustraciju i odliv developera. Zbog toga se u praksi često koriste verzionisanje (npr. /v1/, /v2/ u URL-u) i striktna politika retrokompatibilnosti.

Na kraju, public API-ji su stub modernih distribuiranih sistema i servisa u oblaku. Omogućavaju mikroservisnu arhitekturu, gde svaka komponenta izlaže sopstveni API, ali i globalnu interoperabilnost — razvijači mogu kombinovati više API-ja iz različitih domena, kreirajući moćne mashup aplikacije koje povezuju podatke o vremenu, kartografiji, e-trgovini ili socijalnim mrežama. Kroz transparentne i dobro definisane web servise, internet postaje istinski modularna platforma, dostupna svima ko ume da pošalje i obradi HTTP zahtev.

* + 1. OpenWeather API

**OpenWeatherMap** je jedan od najpopularnijih javnih servisa za meteorološke podatke, namenjen programerima koji žele da u svoje aplikacije integrišu informacije o temperaturi, vlažnosti vazduha, vetru i aktuelnim vremenskim prilikama. Reč je o sveobuhvatnoj platformi koja nudi i besplatan API sloj, dovoljno bogat za potrebe prototipova, studentskih projekata i manjih produkcionih sistema. U kontekstu Rebel Fitness aplikacije, OpenWeather API se koristi kako bi korisniku omogućio brzu i intuitivnu proveru vremenskih uslova u odabranom gradu (OpenWeatherMap, 2025).

Arhitektura OpenWeather servisa zasnovana je na REST principima: svaki meteorološki podatak predstavlja resurs dostupan putem standardizovanih HTTP metoda. Najčešće se koristi jednostavan **GET** zahtev ka ruti:

[**https://api.openweathermap.org/data/2.5/weather?q={city}&appid={API\_KEY}**](https://api.openweathermap.org/data/2.5/weather?q=%7bcity%7d&appid=%7bAPI_KEY%7d)

****

Slika 13 - OpenWeather API Logo

Rezultat je JSON objekat koji sadrži sve ključne meteorološke parametre — trenutnu temperaturu, opis vremenskog stanja (npr. “clear sky”, “light rain”), procentualnu vlažnost vazduha, jačinu i pravac vetra, kao i geo-koordinate grada. Aplikacija *Rebel Fitness* koristi backend proxy rutu (/api/weather/{city}), koja preuzima podatke sa OpenWeather servisa i vraća ih u uprošćenom formatu prilagođenom potrebama interfejsa (OpenWeatherMap, 2025).

Podaci koje OpenWeather pruža izrazito su strukturirani i dosledni: temperature su dostupne u više jedinica (Kelvin, Celsius, Fahrenheit), vremenski opis dolazi uz jedinstveni ID ikone, a objekti poput main, wind i weather omogućavaju jasan, hijerarhijski pregled informacija. U vizuelnom kontekstu, frontend aplikacija koristi samo deo ovih podataka — temperaturu, opis vremena, vlažnost i brzinu vetra — što korisniku omogućava jednostavan i čist prikaz u okviru *Weather Widget* komponente. Zahvaljujući univerzalnom JSON formatu, integracija sa React komponentama je direktna i bez potrebe za dodatnom konverzijom strukture.

Jedna od ključnih prednosti OpenWeather API-ja jeste njegova otvorenost i dostupnost. Osnovni nivo je besplatan, uz veoma velik broj poziva mesečno, što ga čini idealnim za studentske i edukativne projekte. Potreban je samo API ključ, lako generisan nakon registracije, bez složenih autorizacionih mehanizama poput OAuth-a. Stabilnost, ažurnost podataka i široka community podrška dodatno ga pozicioniraju kao jedan od najčešće korišćenih servisa za integraciju vremenskih podataka.

Zahvaljujući ovim karakteristikama, OpenWeather API predstavlja pouzdan, jednostavan i funkcionalan izvor meteoroloških informacija unutar *Rebel Fitness* aplikacije. Omogućava korisnicima da brzo provere vremenske uslove pre planiranja fizičkih aktivnosti, dok je za razvojni tim njegova integracija minimalno opterećujuća i visoko skalabilna.

# IMPLEMENTACIJA VEB APLIKACIJE

U ovom poglavlju sledi detaljan **prikaz celokupnog procesa izrade veb aplikacije**, od inicijalnog prikupljanja zahteva do **konačne implementacije funkcionalnosti**. Kroz četiri međusobno povezane faze – prikupljanje korisničkih zahteva, analiza, projektovanje i implementacija – prate se sve ključne aktivnosti i odluke koje vode od ideje do gotovog rešenja. Prvo definišemo potrebe i očekivanja korisnika, zatim ih sistematski analiziramo kako bismo izbegli nesporazume i tehnička ograničenja. U fazi projektovanja razrađujemo strukturu interfejsa, arhitekturu i tokove podataka, koristeći kombinaciju Laravel back-enda i React front-enda. Na kraju, prikazuju se konkretni razvojni koraci, alati i najbolje prakse, čime se omogućava transparentan uvid u celokupni tok implementacije i olakšava njihova primena u sličnim projektima (Vlajić, 2020).

## Faza prikupljanja korisničkih zahteva

**Faza prikupljanja korisničkih zahteva** predstavlja polaznu tačku svakog uspešnog softverskog projekta, jer njome obezbeđujemo jasan okvir za sve naredne aktivnosti. U ovoj fazi, kroz saradnju sa budućim korisnicima, poslovnim analitičarima i drugim zainteresovanim stranama, identifikujemo i dokumentujemo njihove potrebe – kako funkcionalne (šta sistem treba da radi), tako i nefunkcionalne (performanse, bezbednost, dostupnost). Temeljito prikupljanje zahteva omogućava da se razume poslovni kontekst, prioriteti i ograničenja, čime se značajno smanjuje rizik od prepravki i nedoumica tokom razvoja (Vlajić, 2020).

Ključni ishodi ove faze su verbalni opis i definicija slučajeva korišćenja. **Verbalni opis** služi za precizno prenošenje ciljeva i opisa funkcionalnosti u prirodnom jeziku, dopunjen dijagramima ili ilustracijama po potrebi. On pomaže svim članovima tima – i onima bez dubinskog tehničkog znanja – da razumeju osnovne tokove podataka i poslovne procese koje sistem treba da podrži. Time se stvara zajednički vokabular i sprečavaju nesporazumi između korisnika, analitičara i razvojnih inženjera (Vlajić, 2020).

**Slučajevi korišćenja (use cases)** pak donose formalniju strukturu u opis interakcija korisnika sa sistemom. Svaki slučaj definiše aktore, preduslove, osnovni tok događaja i moguće alternativne scenarije, pa omogućava jasnu eksplicitaciju kako sistem odgovara na različite korisničke akcije. Ovaj model podstiče raniju identifikaciju graničnih stanja, izuzetaka i potrebnih provera, što doprinosi kvalitetnijoj proceni obima posla i planiranju arhitekture. Zajedno, verbalni opis i slučajevi korišćenja čine čvrstu osnovu za prelazak u fazu projektovanja sistema i implementacije (Vlajić, 2020).

### Verbalni opis

Rebel Fitness App predstavlja moderno dizajniranu full-stack veb aplikaciju razvijenu kao centralna platforma za upravljanje treninzima, vežbama, ciljevima i korisnicima u fitnes okruženju. Projekat kombinuje snagu Laravel back-enda i fleksibilnost React front-enda, dok se kompletna arhitektura oslanja na jasno definisan REST API i sigurnu autentifikaciju putem tokena. Backend deo, izgrađen u Laravelu, obezbeđuje robustan sloj poslovne logike, validacije i pristup bazama podataka, dok frontend u Reactu, organizovan kroz modulsku strukturu komponenti i konteksta, obezbeđuje intuitivan, responzivan i vizuelno koherentan korisnički interfejs. Komunikacija između dva sloja obavlja se JSON formatom preko HTTP zahteva, a Vite omogućava izuzetno brzu razvojnu konfiguraciju i optimizovane build procese.

Autentifikacija korisnika izvedena je pomoću Laravel Sanctuma, koji generiše personalizovane API tokene nakon prijave, registracije ili guest pristupa. Na klijentskoj strani React aplikacija koristi centralizovani AuthContext kako bi čuvala token i podatke o korisniku unutar localStorage-a. Axios klijent je dodatno konfigurisan da automatski ubacuje Bearer token u Authorization zaglavlje svakog narednog poziva, čime se obezbeđuje da sve sačuvane rute i zaštićeni resursi budu dostupni isključivo autentifikovanim korisnicima. Za kontrolu pristupa implementirane su dve ključne zaštitne komponente: ProtectedRoute, koja ograničava pristup samo korisnicima sa validnim tokenom, i RoleRoute, koja precizno filtrira stranice po roli — guest, member ili administrator.

**Neulogovani posetioci** imaju pristup inicijalnom ekranu za prijavu, registraciju i ulaz kao gost. Forma za prijavu je minimalna i zahteva samo email i lozinku, dok sekcija za registraciju obuhvata ime, email, lozinku, izbor uloge (member ili administrator) i izbor fitness nivoa (beginner, intermediate, expert). Guest login je specifična pogodnost aplikacije — omogućava korisnicima da testiraju funkcionalnosti bez kreiranja naloga, uz ograničenja koja važe za ovu tip-u rolu. Sve tri funkcionalnosti povezane su sa backend rutama /login, /register i /guest/login.

**Korisnik** nakon prijave ulazi na početnu stranicu (Home), čiji izgled i funkcionalnosti zavise od uloge. Na vrhu se nalazi modal koji se pojavljuje prilikom prvog pristupa i služi kao kratko uputstvo o tome šta aplikacija omogućava. Ostatak stranice ističe jednostavan i pregledan prikaz: listu aktivnih linkova u navigaciji, prikaz vremenske prognoze putem integrisanog OpenWeather API-ja (kroz backend rutu /api/weather/{city}) i dinamičke sekcije sa osnovnim podacima o korisniku. React komponenta WeatherWidget prikazuje trenutnu temperaturu, vremenske uslove, vlažnost i brzinu vetra za izabrani grad, dok se njegovo dohvaćanje obavlja asinhrono kroz custom hook useWeather, posebno dizajniran za ovaj projekat.

**Članovi** (member) imaju pristup kompletnoj fitnes funkcionalnosti: treninzima, vežbama i ličnim ciljevima. U sekciji Workouts prikazana je CRUD logika nad treninzima — njihovo kreiranje, izmena i brisanje. Svaki trening može sadržati više vežbi, koje se zasebno uređuju u sekciji Exercises. Vežbe imaju polja poput naziva, tipa (strength/cardio/flexibility) i opisa, dok postojeći filteri omogućavaju pretragu i kategorizaciju. Sekcija Goals pruža korisniku mogućnost kreiranja fitnes ciljeva, praćenja napretka i njihovog ažuriranja kroz jednostavnu i čistu tabelu. Sve tri celine direktno komuniciraju sa Laravel kontrolerima čije rute su prikazane u Postman kolekciji — /workouts, /exercises, /goals.

**Administrator** ima proširena ovlašćenja, uključujući pristup Admin Dashboard sekciji u kojoj može da upravlja svim registrovanim korisnicima. Stranica “Admin – Users” prikazuje tabelarni pregled svih korisnika osim drugih administratora. Admin ima mogućnost pretrage, filtriranja po ulozi i brisanja korisnika. Ove operacije obavljaju se pozivima ka rutama /admin/users i /admin/users/{id}, uz automatsko osvežavanje stanja kako bi se rezultati odmah videli. Administratorska uloga ima najširi opseg privilegija jer omogućava održavanje celog ekosistema aplikacije.

Bezbednosni model aplikacije strogo razdvaja rolnu logiku:

* **Guest** ima ograničena prava i ne može kreirati, uređivati niti brisati podatke.
* **Member** ima pristup treninzima, vežbama i ciljevima, uz mogućnost izmene samo sopstvenih podataka.
* **Administrator** upravlja kompletno sistemskim korisnicima i najkritičnijim podacima aplikacije.

Frontend je vizuelno stilizovan ručno pisanim CSS-om bez upotrebe UI biblioteka, što omogućava potpunu kontrolu nad izgledom. Interfejs je jednostavan, moderan i responzivan — od navigacije, preko formi, do tabličnih prikaza. Vite dodatno obezbeđuje izuzetno brzu izradu, hot-reload i optimizovan deployment.

Rebel Fitness App tako predstavlja spoj savremenih tehnologija, jasne arhitekture, modularnog razvoja i sigurnog pristupnog modela, nudeći intuitivno iskustvo za korisnike i efikasne alate za administratore. Aplikacija služi kao primer stabilno organizovanog i održivog full-stack rešenja, idealnog za dalja proširenja kao što su statistike, napredna analitika, društveni elementi ili mobilnonativna integracija.

### Slučajevi korišćenja

**Use-case (slučaj korišćenja)** predstavlja jednu funkciju ili uslugu koju sistem pruža korisniku (akteru). Akteri mogu biti ljudi ili drugi sistemi koji iniciraju neki tok rada u aplikaciji, npr. adinistrator ili član. Svaki use-case opisuje šta akter očekuje da sistem uradi i koje korake pri tom preduzima. Za svaki use-case definišemo (Vlajić, 2020):

* **Ime** (kratko, razumljivo, npr. “Kreiranje zahteva za odmor”),
* **Aktere** koji ga koriste (npr. član),
* **Preduslove** (stanje sistema pre pokretanja, npr. korisnik je ulogovan),
* **Osnovni tok** koraka (npr. popunjavanje datuma, potvrda zahteva, čuvanje u bazi),
* **Izuzetke** ili alternativne tokove (npr. neispravan format datuma ili nedozvoljeno vreme).

**Use-case model** služi da na jasan i strukturisan način zabeleži sve funkcionalne zahteve pre početka razvoja. Dobro osmišljeni use-case-ovi pomažu timu da (Vlajić, 2020):

* precizno razume potrebe korisnika,
* ispravno postavi informacije za dizajn interfejsa,
* planira testove za proveru ispravnosti svake funkcije.

**APUSO**, **ANSO**, **APSO**, **IA** i **SO** su vrste akcija koje definišu interakciju između aktera (korisnika) i softverskog sistema u okviru scenarija slučajeva korišćenja (Vlajić, 2020):

* **APUSO (Akcija pripreme ulaza sistemske operacije)** podrazumeva unos vrednosti ili parametara u polja pre pokretanja sistemske operacije (npr. prodavac unosi kriterijum pretrage).
* **APSO (Akcija poziva sistemske operacije)** označava trenutak kada akter zaista aktivira sistemsku operaciju, kao što je klik na dugme “Zapamti” ili “Obradi”.
* **SO (Sistemska operacija)** predstavlja atomsku logiku unutar sistema koja izvršava samu obradu – na primer, pretragu ili čuvanje podataka – bez neposredne akcije korisnika u toku obrade.
* **IA (Izlazni argumenti)** su rezultati ili poruke koje se nakon izvršene operacije vraćaju akteru, poput “Sistem je pronašao račun po zadatoj vrednosti” ili “Sistem je zapamtio račun”.
* **ANSO (Akcija ne-sistemske operacije)** obuhvata sve lokalne provere i validacije na strani klijenta koje akter vrši pre nego što pozove sistemsku operaciju (npr. korisnik proverava da li je ispravno uneo podatke).

Kroz use-case-ove prepoznajemo sve moguće tokove interakcije, smanjujemo rizik od zaboravljenih funkcija i pravovremeno otkrivamo neslaganja u zahtevima, što je ključni korak u uspešnom razvoju svakog softverskog sistema. Evo kompletnog pregleda svih slučajeva korišćenja (SK) podeljenih po korisničkim ulogama:

1. **Neulogovani korisnik:**

* **SK1:** Prijava korisnika
* **SK2:** Registracija korisnika
* **SK3:** Prikaz javne početne stranice

2. **Gost (Guest)**

* **SK4:** Pregled svih treninga korisnika (bez čuvanja podataka)
* **SK5:** Pregled vremenske prognoze grada (javni servis)

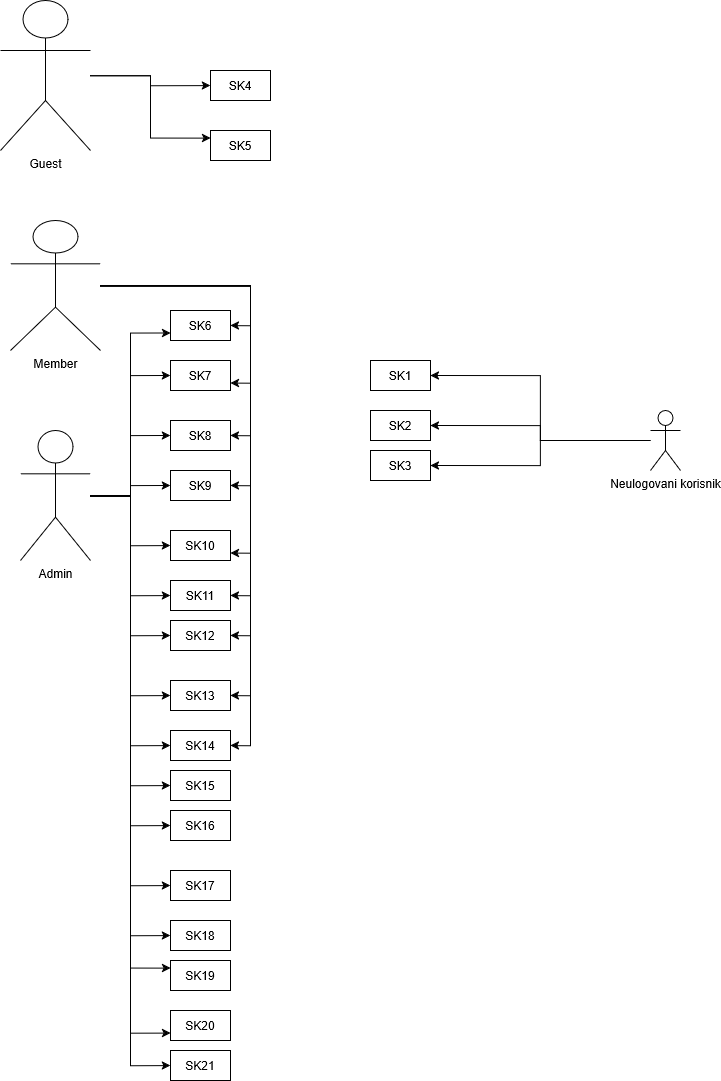
3. **Ulogovani korisnik – član (Member)**

* **SK6:** Odjava člana
* **SK7:** Pregled svojih treninga
* **SK8:** Kreiranje treninga
* **SK9:** Izmena postojećih treninga
* **SK10:** Brisanje treninga
* **SK11:** Kreiranje nove vežbe
* **SK12:** Izmena vežbe
* **SK13:** Brisanje vežbe
* **SK14:** Pregled vežbi sa filtriranjem i sortiranje

4. **Administrator (Admin)**

Pored svih mogućnosti koje ima običan član, administrator ima i dodatne administratorske slučajeve korišćenja:

* **SK15:** Pregled ciljeva
* **SK16:** Kreiranje novog cilja (za člana)
* **SK17:** Izmena ciljeva
* **SK18:** Brisanje ciljeva
* **SK19:** Pregled svih korisnika
* **SK20:** Brisanje korisnika
* **SK21:** Eksportovanje podataka o korisnicima (TXT, CSV)

****

### SK1: Prijava korisnika

* **Aktori SK:** Neulogovani korisnik
* **Učesnici SK:** Neulogovani korisnik i sistem
* **Preduslov:** Korisnik ima registrovan nalog i validan email/lozinka.
* **Osnovni scenario SK:**
  1. Korisnik otvara stranicu za prijavu **(APUSO).**
  2. Unosi email i lozinku u formu **(APUSO).**
  3. Klikne na dugme „Prijavi se“ **(APSO).**
  4. Sistem proverava kredencijale i generiše access\_token **(SO).**
  5. Sistem vraća token, klijent ga čuva u sessionStorage i preusmerava korisnika na početnu stranicu odgovarajuće uloge **(IA).**
* **Alternativni scenariji:**  
  **4.1** Ako su kredencijali neispravni, API vraća HTTP 401, klijent prikazuje “Invalid credentials” **(IA).**  
  **4.2** Ako server ne odgovori (timeout), klijent prikazuje “Network error, please try again” **(IA).**

### SK2: Registracija korisnika

* **Aktori SK:** Neulogovani korisnik
* **Učesnici SK:** Neulogovani korisnik i sistem
* **Preduslov:** Korisnik nema postojeći nalog.
* **Osnovni scenario SK:**
  1. Korisnik otvara stranicu „Login“ **(APUSO).**
  2. Popunjava sve obavezne podatke:
     + Ime i prezime (text)
     + Email adresa (email)
     + Lozinka (password)
     + Uloga (dropdown: member , admin)
     + Fitness nivo (dropdown: beginner, intermediate, expert)
  3. Klikne na dugme „Registruj se“ **(APSO).**
  4. Sistem validira unos i kreira novi nalog u bazi **(SO).**
  5. Sistem vraća poruku o uspehu i preusmerava korisnika na stranicu za prijavu **(IA).**
* **Alternativni scenariji:**  
  **4.1** Ako neki unos nije validan (npr. email već postoji, lozinke se ne poklapaju…) API vraća HTTP 400 sa detaljima grešaka **(IA).**

### SK3: Prikaz javne početne stranice

* **Aktori SK:** Neulogovani korisnik / posetilac
* **Učesnici SK:** Neulogovani korisnik i sistem
* **Preduslov:** Korisnik nije prijavljen i otvara osnovni URL aplikacije.
* **Osnovni scenario SK:**
  1. Korisnik u pregledaču otvara početni URL aplikacije ili klikne na link ka sajtu (**APUSO**).
  2. Sistem učitava React početnu stranicu i inicijalne podatke (logo, navigaciju, tekst dobrodošlice) (**SO**).
  3. Klijent prikazuje javni „hero“ deo sa opisom Rebel Fitness aplikacije, glavnim menijem i CTA dugmićima „Login“, „Registruj se“ i „Nastavi kao gost“ (**IA**).
  4. Sistem, ukoliko je uključeno održavanje, prikazuje modal „PAŽNJA! Zbog radova na održavanju sajta…“ koji korisnik potvrđuje klikom na dugme „OK“ ili zatvaranjem modala (**IA/ANSO**).

### SK4: Pregled svih treninga korisnika (bez čuvanja podataka)

* **Aktori SK:** Gost (ulogovani korisnik sa ulogom *guest*)
* **Učesnici SK:** Gost i sistem
* **Preduslov:** Korisnik je uspešno ulogovan preko brzog ulaza kao gost (validan *access\_token* i uloga *guest*).
* **Osnovni scenario SK:**

1. Gost klikne na stavku menija „Workouts“ / „Moji treninzi“ na početnoj stranici ( **ANSO** ).
2. Klijent šalje zahtev GET /api/workouts sa *Bearer* tokenom gosta u zaglavlju ( **APSO** ).
3. Sistem preko trenutno ulogovanog korisnika ( Auth::user() ) pronalazi sve treninge vezane za tog korisnika i priprema paginiranu listu rezultata ( **SO** ).
4. Sistem vraća JSON listu treninga (ID, naziv, opis, trajanje, kalorije, status…) a klijent renderuje grid kartica sa osnovnim informacijama o svakom treningu, bez mogućnosti uređivanja ili čuvanja promena za guest nalog ( **IA** ).

* **Alternativni scenariji:**

2.1 Ako API vrati HTTP 401 (istekao ili neispravan token), klijent briše lokalne podatke o gostu i preusmerava korisnika na javnu početnu stranicu sa porukom o potrebi ponovnog ulaska kao gost ( **IA** ).

3.1 Ako sistem vrati praznu listu treninga, klijent umesto grida prikazuje informativnu poruku da za gost nalog trenutno nema dostupnih treninga za prikaz ( **IA** ).

### SK5: Pregled vremenske prognoze grada (javni servis)

* **Aktori SK:** Gost
* **Učesnici SK:** Gost i sistem
* **Preduslov:** Gost je uspešno ušao u aplikaciju (Guest login) i nalazi se na stranici sa vidžetom za vremensku prognozu.
* **Osnovni scenario SK:**

1. Gost otvara sekciju **„Vremenska prognoza“** na stranici ( **ANSO** ).
2. U polje **„Grad“** unosi naziv željenog grada (npr. *Belgrade*, *Novi Sad*) ( **APUSO** ).
3. Gost klikne na dugme **„Prikaži vreme“** ( **APSO** ).
4. Klijent šalje zahtev **GET /api/weather/{city}** gde je *{city}* URL-enkodovan naziv grada ( **APSO** ).
5. Backend kontroler **WeatherController** poziva eksterni servis **OpenWeather API**, obrađuje odgovor i mapira ga na polja: *city, temperature, weather, humidity, wind\_speed* ( **SO** ).
6. Sistem vraća JSON odgovor sa podacima o vremenu za traženi grad ( **SO** ).
7. Klijent prikazuje karticu sa rezultatima: naziv grada, trenutna temperatura, opis vremenskih uslova, vlažnost vazduha i brzina vetra ( **IA** ).

* **Alternativni scenariji:**

5.1 Ako je polje za grad prazno, klijentska logika vraća grešku *„Grad nije prosleđen.“* i ne šalje zahtev ka API-ju ( **IA** ).

5.2 Ako OpenWeather API vrati grešku (nepostojeći grad, problem sa mrežom ili CORS), backend prosleđuje grešku, a klijent prikazuje poruku:  
**„Ne mogu da dohvatim prognozu. Proveri /api/weather/{city} i CORS.“** ( **IA** ).

### SK6: Odjava člana

* **Aktori SK:** Ulogovani korisnik – član (Member)
* **Učesnici SK:** Član i sistem
* **Preduslov:** Korisnik je uspešno prijavljen i ima validan token u localStorage.
* **Osnovni scenario SK:**

1. Član klikne na stavku menija **„Logout“** ( **ANSO** ).
2. Klijent šalje zahtev **POST /logout** ka serveru uz Bearer token ( **APSO** ).
3. Backend (AuthController) poništava korisnikov token i vraća potvrdu o uspešnom odjavljivanju ( **SO** ).
4. Klijent uklanja *token* i *user* iz localStorage i automatski preusmerava korisnika na stranicu **/login** ( **IA** ).

* **Alternativni scenariji:**

3.1 Ako server ne može da poništi token (istekao, nevažeći ili već obrisan), API vraća grešku — ali klijent svejedno briše lokalne podatke i prikazuje poruku  
**„Odjava delimično uspešna. Prijavite se ponovo.“** ( **IA** ).

3.2 Ako postoji mrežna greška (timeout), klijent obavlja lokalnu odjavu i prikazuje  
**„Network error — token je obrisan lokalno.“** ( **IA** ).

### SK7: Pregled svojih treninga

* **Aktori SK:** Ulogovani korisnik – član (Member)
* **Učesnici SK:** Član i sistem
* **Preduslov:** Korisnik je uspešno prijavljen i ima važeći Bearer token. Postoje ili ne postoje sačuvani treninzi u bazi.
* **Osnovni scenario SK:**

1. Član klikne na stavku menija **„Workouts“** ( **ANSO** ).
2. Klijent šalje zahtev **GET /workouts** backendu uz odgovarajući Authorization header ( **APSO** ).
3. Backend (WorkoutController@index) pronalazi sve treninge koji pripadaju **ulogovanom korisniku** i vraća listu u JSON formatu ( **SO** ).
4. Klijent renderuje grid/listu treninga, pri čemu se za svaki prikazuje:
   * naziv treninga
   * cilj (goal) ako postoji
   * broj vežbi
   * datum kreiranja
   * dugmad **Edit** i **Delete** ( **IA** ).

**• Alternativni scenariji:**

3.1 Ako korisnik nema nijedan trening, API vraća prazan niz, a klijent prikazuje poruku:  
**„You currently have no workouts.“** ( **IA** ).

3.2 Ako API vrati **HTTP 401** (istekao token), klijent briše token iz localStorage i preusmerava korisnika na **/login** ( **IA** ).

3.3 Ako dođe do greške na serveru (HTTP 500), klijent prikazuje poruku  
**„Failed to load workouts. Please try again later.“** ( **IA** ).

### SK8: Kreiranje treninga

* **Učesnici SK:** Član i sistem
* **Preduslov:** Korisnik je prijavljen kao *member* ili *admin* i ima važeći token; korisnik se nalazi na stranici sa listom treninga ili koristi link **„+ Novi trening“** / putanju **/workouts/new**.
* **Osnovni scenario SK:**

1. Korisnik klikne na dugme/link **„+ Novi trening“** u okviru sekcije *Workouts* ( **ANSO** ).
2. Klijent otvara stranicu **„Novi trening“** sa formom za unos podataka:
   * *Naziv* (obavezno polje, tekst),
   * *Opis* (opciono polje, tekst),
   * *Trajanje (min)* (opciono, broj ≥ 0),
   * *Kalorije (kcal)* (opciono, broj ≥ 0),
   * *Status* (dropdown: *pending*, *started*, *completed*) ( **IA** ).
3. Korisnik popunjava polja forme i unosi naziv treninga ( **APUSO** ).
4. Korisnik klikne na dugme **„Sačuvaj“** ( **APSO** ).
5. Klijent šalje zahtev **POST /users/workouts** backendu sa JSON telom koje sadrži:  
   { name, description, duration, calories\_burned, status } i Authorization header ( **APSO** ).
6. Backend (WorkoutController@store) validira podatke, kreira novi trening vezan za ulogovanog korisnika i čuva ga u bazi ( **SO** ).
7. Sistem vraća JSON odgovor sa novokreiranim treningom i statusom uspeha ( **SO** ).
8. Klijent na osnovu odgovora preusmerava korisnika nazad na listu treninga **/workouts**, gde se novi trening prikazuje u listi ( **IA** ).

* **Alternativni scenariji:**

4.1 Ako korisnik pokuša da pošalje formu bez unetog naziva, frontend validacija vraća poruku  
**„Naziv je obavezan.“** i sprečava slanje zahteva ka serveru ( **IA** ).

4.2 Ako korisnik unese negativnu vrednost za trajanje ili kalorije (npr. -10), klijent prikazuje odgovarajuću poruku o grešci:  
**„Trajanje mora biti nenegativan broj.“** ili **„Kalorije moraju biti nenegativan broj.“** i ne šalje zahtev ( **IA** ).

6.1 Ako backend vrati **HTTP 422** zbog nevalidnih podataka (npr. predugačak naziv ili neispravan format), klijent prikazuje poruku  
**„Kreiranje nije uspelo. Proveri privilegije i backend.“** ( **IA** ).

6.2 Ako backend vrati **HTTP 401** ili **403** (nema privilegije ili istekao token), klijent može obrisati lokalno sačuvani token i preusmeriti korisnika na **/login** ( **IA** ).

6.3 Ako dođe do interne greške servera (npr. **HTTP 500**), klijent prikazuje generičku poruku o grešci i ostavlja korisnika na formi za unos kako bi mogao da pokuša ponovo ( **IA** ).

### SK9: Izmena postojećih treninga

* **Aktori SK:** Ulogovani korisnik – član (**Member**)
* **Učesnici SK:** Član i sistem
* **Preduslov:** 
  + Korisnik je prijavljen kao *member* ili *admin* i ima važeći token.
  + Trening koji se menja postoji u bazi i pripada tom korisniku (ili je admin).
  + Korisnik se nalazi na listi treninga (*Workouts*), npr. na stranici **/workouts** ili **/users/workouts**.
* **Osnovni scenario SK:**

1. Korisnik na listi treninga klikne na dugme/link **„Uredi“** pored željenog treninga ( **ANSO** ).
2. Klijent preusmerava korisnika na rutu **/workouts/{id}/edit** i šalje zahtev **GET /workouts/{id}** da učita detalje treninga ( **APSO** ).
3. Sistem (backend – WorkoutController@show) pronalazi traženi trening i vraća JSON sa poljima  
   name, description, duration, calories\_burned, status ( **SO** ).
4. Klijent popunjava formu za izmenu treninga postojećim vrednostima:
   * *Naziv* (obavezno),
   * *Opis* (opciono),
   * *Trajanje (min)* (opciono, broj ≥ 0),
   * *Kalorije (kcal)* (opciono, broj ≥ 0),
   * *Status* (dropdown: *pending*, *started*, *completed*) ( **IA** ).
5. Korisnik menja željena polja u formi (npr. naziv, trajanje, status) ( **APUSO** ).
6. Korisnik klikne na dugme **„Sačuvaj izmene“** ( **APSO** ).
7. Klijent šalje zahtev **PUT /users/workouts/{id}** sa JSON telom:  
   { name, description, duration, calories\_burned, status } i **Authorization: Bearer {token}** ( **APSO** ).
8. Backend (WorkoutController@update) validira podatke, proverava da li korisnik ima pravo izmene, ažurira zapis u bazi i vraća osvežen resurs ( **SO** ).
9. Klijent na osnovu uspešnog odgovora preusmerava korisnika nazad na stranicu sa listom treninga **/workouts**, gde je izmenjeni trening prikazan sa novim vrednostima ( **IA** ).

* **Alternativni scenariji:**

3.1 Ako trening ne postoji ili nije dostupan korisniku (npr. traži tuđi ID), backend vraća **HTTP 404** ili **403**, a klijent prikazuje poruku  
**„Ne mogu da učitam ovaj trening.“** i može ponuditi povratak na listu treninga ( **IA** ).

5.1 Ako korisnik obriše naziv treninga i pokuša da sačuva, klijent javlja grešku validacije  
**„Naziv je obavezan.“** i ne šalje zahtev ka serveru dok polje nije ispravljeno ( **IA** ).

5.2 Ako korisnik unese negativno trajanje ili kalorije (npr. -5), klijent prikazuje poruke:  
**„Trajanje mora biti nenegativan broj.“** ili **„Kalorije moraju biti nenegativan broj.“** i sprečava slanje forme ( **IA** ).

8.1 Ako backend pri ažuriranju vrati **HTTP 422** (nevalidni podaci) ili **HTTP 500** (interna greška), klijent prikazuje poruku  
**„Ažuriranje nije uspelo. Pokušaj ponovo.“** i ostavlja korisniku mogućnost da ispravi podatke ili proba ponovo slanje ( **IA** ).

8.2 Ako backend vrati **HTTP 401** ili **403** (istekao token ili nema privilegije), klijent može obrisati lokalno sačuvan token, obavestiti korisnika da nema pristup i preusmeriti ga na stranicu **/login** ( **IA** ).

### SK10: Brisanje treninga

* **Aktori SK:** Ulogovani korisnik – član (**Member**) ili administrator (**Admin**)
* **Učesnici SK:** Član / administrator i sistem
* **Preduslov:**
  + Korisnik je prijavljen (*member* ili *admin*) i ima važeći token.
  + Trening koji se briše postoji u bazi i pripada tom korisniku (ili je admin).
  + Korisnik se nalazi na listi treninga (stranica **/workouts**).
* **Osnovni scenario SK:**

1. Korisnik na listi treninga pronalazi trening koji želi da obriše i klikne na dugme **„Obriši“** pored tog treninga ( **ANSO** ).
2. Sistem (frontend) otvara modalni prozor za potvrdu sa tekstom tipa:  
   **„Da li sigurno želiš da obrišeš ovaj trening?“** i prikazuje dugmad **„Ne“** i **„Da, obriši“** ( **IA** ).
3. Korisnik klikne na dugme **„Da, obriši“** u okviru modala ( **APSO** ).
4. Klijent šalje zahtev **DELETE /users/workouts/{id}** sa odgovarajućim id treninga i zaglavljem  
   **Authorization: Bearer {token}** ka backendu (WorkoutController@destroy) ( **APSO** ).
5. Backend proverava da li korisnik ima pravo da obriše taj trening (vlasnik ili admin), briše zapis iz baze i vraća odgovor o uspehu ( **SO** ).
6. Klijent, po uspešnom odgovoru servera, uklanja obrisani trening iz lokalne liste items i ažurira prikaz kartica na stranici **/workouts** ( **IA** ).
7. Modal se zatvara, a korisnik na ekranu više ne vidi obrisani trening u listi ( **IA** ).

* **Alternativni scenariji:**

2.1 Ako korisnik u modalu klikne na dugme **„Ne“**, modal se zatvara, ne šalje se nikakav zahtev ka serveru i lista treninga ostaje neizmenjena ( **IA** ).

5.1 Ako backend vrati **HTTP 403** (korisnik nema pravo da obriše taj trening), klijent prikazuje poruku tipa  
**„Nemate dozvolu za ovu akciju.“**, a trening ostaje u listi ( **IA** ).

5.2 Ako backend vrati **HTTP 404** (trening ne postoji) ili **HTTP 500** (interna greška), klijent obaveštava korisnika porukom tipa  
**„Brisanje nije uspelo. Pokušaj ponovo.“** i ne uklanja trening iz liste ( **IA** ).

5.3 Ako server vrati **HTTP 401** (istekao ili nevažeći token), klijent može obrisati lokalni token, prikazati poruku  
**„Sesija je istekla, prijavite se ponovo.“** i preusmeriti korisnika na stranicu **/login** ( **IA** ).

### SK11: Kreiranje nove vežbe

* **Aktori SK:** Ulogovani korisnik – član (**Member**) ili administrator (**Admin**)
* **Učesnici SK:** Član / administrator i sistem
* **Preduslov:**
  + Korisnik je prijavljen (*member* ili *admin*) i ima važeći token.
  + Korisnik ima barem jedan postojeći trening (**workout**) u sistemu ili zna workout\_id kome želi da doda vežbu.
  + Korisnik se nalazi na stranici za rad sa vežbama (**/exercises**) ili direktno na ruti **/exercises/new**.
* **Osnovni scenario SK:**

1. Korisnik na stranici **„Exercises“** klikne na dugme **„Nova vežba“** ili direktno otvara rutu **/exercises/new** ( **ANSO** ).
2. Sistem (frontend) prikazuje formu za kreiranje vežbe sa poljima:
   * *Naziv* (obavezno tekstualno polje),
   * *Opis* (opciono tekstualno polje),
   * *Ponavljanja/Vreme* (obavezno numeričko polje),
   * *Tip* (padajuća lista: **cardio**, **strength**, **flexibility**),
   * *Workout* (padajuća lista dostupnih treninga i opciono ručni unos workout\_id) ( **IA** ).
3. Korisnik popunjava polja forme: unosi naziv, opis (po želji), broj ponavljanja/vremena, bira tip vežbe i bira željeni trening iz liste (ili ručno unosi workout\_id) ( **APUSO** ).
4. Klijent obavlja osnovnu validaciju na strani browsera – proverava da **naziv nije prazan**, da je *Ponavljanja/Vreme* validan broj i da je odabran trening ( **ANSO** ).
5. Korisnik klikne na dugme **„Kreiraj“** u dnu forme ( **APSO** ).
6. Klijent šalje zahtev **POST /exercises** sa JSON telom u formatu npr.:  
   { name, description, reps\_or\_time, type, workout\_id } i zaglavljem  
   **Authorization: Bearer {token}** ka backendu (ExerciseController@store) ( **APSO** ).
7. Backend validira podatke, kreira novi zapis vežbe u bazi i povezuje ga sa izabranim treningom, zatim vraća odgovor o uspehu ( **SO** ).
8. Klijent po uspešnom odgovoru preusmerava korisnika nazad na stranicu **/exercises**, automatski osvežava listu vežbi i nova vežba se pojavljuje u tabeli ( **IA** ).
9. Korisnik na listi vežbi vidi novu vežbu sa unetim nazivom, tipom, brojem ponavljanja/vremena i pripadajućim treningom ( **IA** ).

* **Alternativni scenariji:**

4.1 Ako korisnik pokuša da pošalje formu sa praznim *Nazivom*, nevalidnim brojem u polju *Ponavljanja/Vreme* ili bez odabranog treninga, frontend prikazuje poruku tipa  
**„Naziv je obavezan.“, „Ponavljanja/Vreme mora biti broj.“ ili „Morate odabrati workout.“** i ne šalje zahtev backendu ( **IA** ).

7.1 Ako backend vrati **HTTP 422** (npr. nevalidni podaci, nepostojeći workout\_id), sistem vraća poruke o greškama, a klijent ih prikazuje ispod odgovarajućih polja u formi ( **IA** ).

7.2 Ako backend vrati **HTTP 401** (istekao ili nevažeći token), klijent može obrisati lokalni token, prikazati poruku  
**„Sesija je istekla, prijavite se ponovo.“** i preusmeriti korisnika na stranicu **/login** ( **IA** ).

7.3 Ako backend vrati **HTTP 500** (interna greška servera), klijent prikazuje obaveštenje tipa  
**„Kreiranje vežbe nije uspelo. Pokušaj ponovo.“** i ostavlja korisnika na formi sa sačuvanim unosom ( **IA** ).

### SK12: Izmena vežbe

* **Aktori SK:** Ulogovani korisnik – član (**Member**) ili administrator (**Admin**)
* **Učesnici SK:** Korisnik i sistem
* **Preduslov:**
  + Korisnik je prijavljen (validan token).
  + Vežba koju korisnik želi da izmeni postoji u bazi.
  + Korisnik ima pravo izmene (član može menjati samo *svoje* vežbe – one koje pripadaju njegovim treninzima; admin može menjati sve).
  + Korisnik se nalazi na stranici **/exercises** ili direktno na ruti **/exercises/edit/{id}**.
* **Osnovni scenario SK:**

1. **Korisnik klikne** na dugme **„Edit“** pored postojeće vežbe u tabeli vežbi ( **ANSO** ).
2. Klijent otvara formu za izmenu, u kojoj su **polja već popunjena** trenutnim vrednostima vežbe:
   * Naziv,
   * Opis,
   * Ponavljanja/Vreme,
   * Tip (cardio/strength/flexibility),
   * Workout kojem vežba pripada ( **IA** ).
3. Korisnik menja željena polja – npr. ispravlja naziv, povećava broj ponavljanja, menja tip ili prebacuje vežbu na drugi trening ( **APUSO** ).
4. Forma obavlja klijentsku validaciju – proverava da naziv nije prazan, da reps/time sadrži broj i da je izabran važeći trening ( **ANSO** ).
5. Korisnik klikne na dugme **„Sačuvaj izmene“** ( **APSO** ).
6. Klijent šalje zahtev **PUT /exercises/{id}** sa JSON podacima:  
   { name, description, reps\_or\_time, type, workout\_id } i Bearer token zaglavljem ka backendu (ExerciseController@update) ( **APSO** ).
7. Backend validira podatke, proverava da li korisnik ima pravo na izmenu, zatim ažurira vežbu u bazi i vraća izmenjeni resource ( **SO** ).
8. Klijent po uspešnom odgovoru osvežava listu vežbi, prikazuje ažurirane vrednosti i obaveštenje tipa  
   **„Vežba uspešno izmenjena.“** ( **IA** ).
9. Korisnik ostaje na stranici **/exercises**, gde vidi novu verziju iste vežbe ( **IA** ).

* **Alternativni scenariji:**

**4.1** Ako korisnik ostavi *Naziv* praznim ili unese nevalidan broj u *Ponavljanja/Vreme*, forma prikazuje poruke:

* „Naziv je obavezan.“
* „Ponavljanja/Vreme mora biti broj.“  
  i sprečava slanje zahteva ( **IA** ).

**7.1** Ako backend vrati **HTTP 422 (Validation error)**, klijent prikazuje server-side poruke grešaka ispod polja ( **IA** ).

**7.2** Ako backend vrati **HTTP 403 (Forbidden)** jer korisnik pokušava da izmeni vežbu koja ne pripada njegovom treningu, klijent prikazuje poruku:  
**„Nemate dozvolu da izmenite ovu vežbu.“** ( **IA** ).

**7.3** Ako backend vrati **HTTP 401 (Unauthorized)** zbog isteka tokena, klijent briše token i redirektuje korisnika na **/login** ( **IA** ).

**7.4** Ako backend vrati **HTTP 500 (Server error)**, prikazuje se poruka:  
**„Izmena nije uspela. Pokušajte ponovo.“** ( **IA** ).

### SK13: Brisanje vežbe

* **Aktori SK:** Ulogovani korisnik – član (**Member**) ili administrator (**Admin**)
* **Učesnici SK:** Korisnik i sistem
* **Preduslov:**
  + Korisnik je prijavljen (validan token postoji u localStorage).
  + Vežba koju korisnik želi da obriše postoji u bazi.
  + Korisnik ima pravo brisanja (član može brisati samo vežbe iz svojih treninga; admin može brisati bilo koju vežbu).
  + Korisnik se nalazi na stranici **„Exercises“** (**/exercises**) i vidi listu vežbi.
* **Osnovni scenario SK:**

1. **Korisnik klikne** na dugme **„Obriši“** pored vežbe koju želi da ukloni sa liste ( **ANSO** ).
2. Klijent otvara modal za potvrdu brisanja sa tekstom tipa:  
   **„Da li sigurno želiš da obrišeš ovu vežbu?“** i dugmadima **„Ne“** i **„Da, obriši“** ( **IA** ).
3. Korisnik potvrđuje akciju klikom na dugme **„Da, obriši“** u modalu ( **APSO** ).
4. Klijent šalje zahtev **DELETE /exercises/{id}** backendu, uz Bearer token u zaglavlju, koristeći deleteExercise(id) funkciju ( **APSO** ).
5. Backend proverava prava pristupa (da li korisnik sme da obriše tu vežbu) i, u slučaju da je sve ispravno, briše vežbu iz baze ( **SO** ).
6. Sistem vraća odgovor o uspehu brisanja (npr. status 200 i poruku) ( **SO** ).
7. Klijent zatvara modal, uklanja obrisanu vežbu iz prikazane liste i prikazuje poruku tipa:  
   **„Vežba je uspešno obrisana.“** ( **IA** ).

* **Alternativni scenariji:**

**2.1** Korisnik klikne na dugme **„Ne“** u modalu: modal se zatvara, ne šalje se nikakav zahtev ka serveru i lista vežbi ostaje nepromenjena ( **IA** ).

**5.1** Ako backend vrati **HTTP 403 (Forbidden)** jer korisnik nema pravo da obriše tu vežbu (npr. vežba pripada tuđem treningu), klijent prikazuje poruku:  
**„Nemate dozvolu da obrišete ovu vežbu.“** i lista ostaje nepromenjena ( **IA** ).

**5.2** Ako backend vrati **HTTP 401 (Unauthorized)** zbog nevalidnog ili isteknutog tokena, klijent briše lokalni token i preusmerava korisnika na stranicu **/login** ( **IA** ).

**5.3** Ako backend vrati **HTTP 404 (Not Found)** jer vežba više ne postoji u bazi, klijent uklanja vežbu iz liste (ako je još prikazana) i prikazuje poruku:  
**„Vežba nije pronađena.“** ( **IA** ).

**5.4** Ako backend vrati **HTTP 500 (Server error)** ili dođe do mrežne greške, klijent prikazuje generičku poruku:  
**„Brisanje nije uspelo. Pokušaj ponovo kasnije.“** i ne menja listu vežbi ( **IA** ).

### SK14: Pregled vežbi sa filtriranjem i sortiranje

* **Aktori SK:** Ulogovani korisnik – član (**Member**) ili administrator (**Admin**)
* **Učesnici SK:** Korisnik i sistem
* **Preduslov:**
  + Korisnik je prijavljen (validan token postoji u localStorage).
  + Korisnik ima ulogu *member* ili *admin* (guest nema pristup stranici **Exercises**).
  + U bazi postoje registrovane vežbe (ili će se u alternativnom scenariju prikazati poruka da nema podataka).
  + Korisnik ima pristup stranici **„Exercises“** (/exercises).
* **Osnovni scenario SK:**

1. **Korisnik klikne** na stavku menija **„Exercises“** u navigaciji aplikacije ( **ANSO** ).
2. Klijent (React aplikacija) šalje inicijalni zahtev ka backendu:  
   **GET /exercises?page=1&per\_page=10&sort=name&dir=asc** (podrazumevani parametri) koristeći funkciju fetchExercises(params) ( **APSO** ).
3. Backend obrađuje zahtev, primenjuje zadate parametre paginacije i sortiranja, čita podatke iz baze i vraća rezultat u obliku paginiranog JSON odgovora (lista vežbi + meta podaci o strani, ukupnom broju itd.) ( **SO** ).
4. Klijent prikazuje tabelu sa vežbama: kolone **ID**, **Naziv**, **Opis**, **Ponavljanja/Vreme**, **Tip**, **Workout**, kao i eventualne akcije (Uredi/Obriši) za ulogovanog člana ili admina ( **IA** ).
5. Korisnik po potrebi unosi tekst u polje **„Pretraga“** (npr. deo naziva ili opisa vežbe) ( **APUSO** ).
6. Klijent ponovno poziva backend:  
   **GET /exercises?search={query}&page=1&per\_page={trenutni}&sort={sort}&dir={dir}&type={typeFilter}&mine={0/1}** i osvežava listu vežbi na osnovu unetog pojma ( **APSO** ).
7. Korisnik bira tip vežbe iz padajuće liste **„Tip“** (npr. cardio, strength, flexibility) ( **APUSO** ).
8. Klijent prosleđuje novi zahtev backendu sa izabranim tipom kao filterom (**type=cardio/strength/flexibility**) i prikazuje samo vežbe tog tipa ( **APSO**, **SO**, **IA** ).
9. Korisnik opcionalno uključuje čekboks **„Samo moji“** da bi video samo vežbe koje su vezane za njegove treninge ( **APUSO** ).
10. Klijent ponovo šalje zahtev **GET /exercises?mine=1&...**; backend filtrira vežbe po ulogovanom korisniku i vraća rezultat ( **APSO**, **SO** ).
11. Klijent prikazuje novu listu – samo vežbe povezane sa treninzima tog korisnika, uz primenjene ostale filtere (tip, pretraga) ( **IA** ).
12. Korisnik menja kriterijum sortiranja preko padajućeg menija **„Sortiraj po“** (npr. name, type, reps\_or\_time, id) i po potrebi klikne na dugme za promenu smera sortiranja (**↑ / ↓**) ( **APUSO** ).
13. Klijent šalje novi zahtev sa parametrima **sort={polje}&dir={asc/desc}**, backend vraća sortiranu listu vežbi, a klijent osvežava tabelu ( **APSO**, **SO**, **IA** ).
14. Korisnik prilagođava broj zapisa po strani preko **„Po strani“** (npr. 5, 10, 20) i koristi dugmad **„Prethodna“** / **„Sledeća“** za navigaciju između strana ( **APUSO** ).
15. Klijent na svaku promenu stranice šalje odgovarajući **GET /exercises?page={n}&per\_page={pageSize}&…** zahtev, a sistem vraća traženu stranu rezultata; klijent prikazuje aktuelnu stranu i pokazuje indikator **„Strana X / Y“** ( **APSO**, **SO**, **IA** ).

* **Alternativni scenariji:**

**3.1** Ako backend vrati **HTTP 401 (Unauthorized)** zbog nevalidnog ili isteknutog tokena, klijent briše token i podatke o korisniku iz localStorage i preusmerava korisnika na stranicu **/login** ( **IA** ).

**3.2** Ako backend vrati **HTTP 500** ili dođe do mrežne greške, klijent prikazuje poruku tipa:  
**„Ne mogu da učitam vežbe. Pokušaj ponovo.“** i ostavlja prethodno stanje ekrana ( **IA** ).

**4.1** Ako sistem ne pronađe nijednu vežbu (bilo inicijalno, bilo nakon filtriranja), klijent prikazuje poruku u okviru kartice:  
**„Nema vežbi za prikaz.“** ( **IA** ).

**11.1** Ako je čekboks **„Samo moji“** uključen, ali korisnik nema nijednu vežbu vezanu za svoje treninge, lista ostaje prazna uz poruku **„Nema vežbi za prikaz.“** ( **IA** ).

**14.1** Ako korisnik postavi veliki broj po strani i poslednja strana nakon brisanja nekih vežbi više ne postoji, klijent automatski vraća korisnika na prethodnu validnu stranu i osvežava podatke ( **IA** ).

### SK15: Pregled ciljeva

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je uspešno prijavljen u sistem i ima dodeljenu ulogu admin.
* **Osnovni scenario SK:**
  1. Administrator u gornjoj navigaciji klikne na stavku menija **„Goals“** (ANSO).
  2. Klijent (React aplikacija) šalje zahtev **GET /api/goals** ka Laravel backendu, uz **Authorization: Bearer {token}** zaglavlje (APSO).
  3. Sistem (Laravel GoalController@index) čita sve zapise iz tabele goals preko Eloquent modela Goal i vraća JSON listu ciljeva (SO).
  4. Klijent renderuje stranicu **„Admin · Goals“**: u tabeli prikazuje po redu kolone **ID**, naziv cilja, skraćeni opis, korisnika (ime ili ID), rok (datum) i status cilja, uz akcije **„Uredi“** i **„Obriši“** za svaki red (IA).
* **Alternativni scenariji:**  
  2.1. Ako backend vrati **HTTP 401** ili **HTTP 403** (nevalidan token ili nedovoljno privilegija), klijent prikazuje poruku:  
  „**Ne mogu da učitam ciljeve. Proveri backend ili privilegije.**“ i onemogućava rad sa tabelom (IA).  
  3.1. Ako u bazi ne postoji nijedan cilj (prazan odgovor), klijent umesto tabele prikazuje prazno stanje sa tekstom:  
  „**Trenutno nema ciljeva.**“ unutar kartice (IA).  
  2.2. Ako dođe do mrežne greške ili neočekivane greške servera (npr. **HTTP 500**), klijent prikazuje istu poruku o grešci i omogućava administratoru da klikom na dugme **„Pokušaj ponovo“** ponovo pošalje zahtev za učitavanje ciljeva (IA).

### SK16: Kreiranje novog cilja (za člana)

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je uspešno prijavljen u sistem i ima ulogu admin. Stranica **„Admin · Goals“** je učitana ili administrator kroz meni otvara sekciju za ciljeve.
* **Osnovni scenario SK:**

1. Administrator na stranici **„Admin · Goals“** klikne na dugme **„+ Novi cilj“** (ANSO).
2. Klijent otvara formu **„Novi cilj“** (komponenta *AdminGoalForm*) sa praznim poljima za **naziv**, **opis**, **rok (datum)**, izbor **korisnika** iz padajuće liste i početni **status** cilja (IA).
3. Sistem (frontend) automatski šalje zahtev **GET /api/admin/users** kako bi popunio dropdown sa listom korisnika (SO), a klijent prikazuje korisnike u padajućoj listi (IA).
4. Administrator popunjava polja forme:
   * naziv cilja (obavezno),
   * opcioni opis,
   * rok cilja (datum),
   * bira korisnika kome se cilj dodeljuje (obavezno),
   * status cilja (npr. *pending* ili *completed*) (APUSO).
5. Administrator klikne na dugme **„Kreiraj“** kako bi sačuvao novi cilj (APSO).
6. Klijent šalje zahtev **POST /api/goals** sa JSON telom koje sadrži naslov, opis, ID korisnika i datum (polja title, description, user\_id, kao i datum u formatu target\_date / due\_date / deadline), zajedno sa **Authorization: Bearer {token}** zaglavljem (APSO).
7. Sistem (Laravel GoalController@store) validira podatke, kreira novi zapis u tabeli goals i vraća JSON reprezentaciju novog cilja sa generisanim ID-jem i ostalim poljima (SO).
8. Klijent po uspešnom odgovoru zatvara formu i preusmerava administratora nazad na stranicu **„Admin · Goals“**, gde se lista ciljeva ponovo učitava i prikazuje uključujući novokreirani cilj (IA).

* **Alternativni scenariji:**   
  4.1. Ako administrator pokuša da pošalje formu bez obaveznog naziva ili bez odabranog korisnika, frontend detektuje grešku i prikazuje tekstualnu poruku **„Naziv je obavezan.“** ili **„Morate odabrati korisnika.“** ispod odgovarajućih polja, bez slanja zahteva ka serveru (IA).  
  7.1. Ako backend pri validaciji vrati **HTTP 422** (npr. nevalidan datum ili nekonzistentni podaci), klijent hvata grešku i prikazuje poruku **„Čuvanje nije uspelo. Proveri polja i privilegije.“** u okviru forme, ostavljajući već unete vrednosti netaknutim (IA).  
  6.1. Ako token nije validan ili administrator nema pravo pristupa, backend vraća **HTTP 401** ili **HTTP 403**, klijent prikazuje poruku o grešci i ne preusmerava sa forme, omogućavajući pokušaj ponovnog slanja nakon ispravke stanja (IA).  
  6.2. Ako dođe do mrežne greške (timeout, nedostupan server), klijent prikazuje poruku **„Čuvanje nije uspelo. Proveri polja i privilegije.“** i ostavlja formu otvorenom kako administrator ne bi izgubio unete podatke (IA).

### SK17: Izmena ciljeva

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je uspešno prijavljen i nalazi se na stranici **„Admin · Goals“**, gde su ciljevi već učitani iz baze. Postoji najmanje jedan cilj koji se može izmeniti.
* **Osnovni scenario SK:**

1. Administrator klikne na dugme **„Izmeni“** pored nekog cilja u listi (ANSO).
2. Klijent otvara formu **„Edit Goal“** (komponenta *AdminGoalForm*) sa prethodno popunjenim poljima — naziv, opis, status, rok i korisnik kojem je cilj dodeljen (IA).
3. Klijent šalje zahtev **GET /api/admin/users** kako bi popunio dropdown za izbor korisnika (SO); lista korisnika se prikazuje (IA).
4. Administrator menja jedno ili više polja cilja:
   * naziv cilja,
   * opis,
   * rok (datum),
   * status cilja,
   * korisnika kome je cilj dodeljen (APUSO).
5. Administrator klikne dugme **„Sačuvaj izmene“** (APSO).
6. Klijent šalje zahtev **PUT /api/goals/{id}** sa izmenjenim podacima i Bearer tokenom u zaglavlju (APSO).
7. Sistem (GoalController@update) validira podatke, ažurira postojeći zapis u tabeli goals i vraća JSON sa novim vrednostima cilja (SO).
8. Klijent zatvara formu i osvežava listu ciljeva, prikazujući ažurirani cilj uz poruku o uspehu (IA).

* **Alternativni scenariji:**  
  4.1. Ako obavezna polja (npr. naziv, korisnik) ostanu prazna, frontend prikazuje validacione greške u okviru forme, bez slanja zahteva backendu (IA).  
  7.1. Ako backend vrati **HTTP 422** (nevalidan datum ili drugi nekorektni podaci), klijent prikazuje serverske greške i ostavlja formu otvorenom sa popunjenim vrednostima (IA).  
  6.1. Ako administrator nema pravo pristupa (npr. token istekao, nevalidan), backend vraća **HTTP 401/403**, a klijent prikazuje poruku i ne zatvara formu (IA).  
  6.2. Ako dođe do prekida konekcije, frontend prikazuje upozorenje **„Izmena nije uspela. Pokušajte ponovo.“** bez brisanja unetih podataka (IA).

### SK18: Brisanje ciljeva

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je uspešno prijavljen i nalazi se na stranici **„Admin · Goals“**, gde je prikazana lista ciljeva iz baze. Postoji cilj koji je moguće obrisati.
* **Osnovni scenario SK:**

1. Administrator klikne na dugme **„Obriši“** pored cilja koji želi da ukloni (ANSO).
2. Klijent prikazuje potvrdu (modal) sa pitanjem: „Da li ste sigurni da želite da obrišete ovaj cilj?“ (IA).
3. Administrator potvrđuje brisanje klikom na **„Yes, delete“** (APSO).
4. Klijent šalje zahtev **DELETE /api/goals/{id}** sa validnim Bearer tokenom (APSO).
5. Sistem (GoalController@destroy) proverava postojanje cilja, briše odgovarajući zapis iz tabele goals i vraća JSON poruku o uspehu (SO).
6. Klijent uklanja cilj iz liste na ekranu i prikazuje poruku **„Cilj uspešno obrisan.“** (IA).

* **Alternativni scenariji:**   
  3.1. Ako administrator klikne **„Cancel“**, modal se zatvara i ništa se ne menja na ekranu (IA).  
  4.1. Ako backend vrati **HTTP 404** (cilj ne postoji), klijent prikazuje upozorenje **„Cilj nije pronađen.“** (IA).  
  4.2. Ako backend vrati **HTTP 403** (nema permisije), klijent prikazuje **„Nemate dozvolu za ovu akciju.“** (IA).  
  4.3. Ako se dogodi greška na serveru (HTTP 500), prikazuje se poruka **„Brisanje neuspešno. Pokušajte kasnije.“** (IA).  
  5.1. Ako dođe do problema u mreži, klijent prikazuje **„Network error“** i ne uklanja cilj sa liste (IA).

### 

### SK19: Pregled svih korisnika

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je uspešno prijavljen i nalazi se u Admin panelu. Postoje korisnici u bazi koje je moguće prikazati.
* **Osnovni scenario SK:**

1. Administrator klikne stavku menija **„Users“** (ANSO).
2. Klijent šalje zahtev **GET /api/users** sa validnim Bearer tokenom (APSO).
3. Sistem (AdminController@index) dohvaća listu svih korisnika iz baze i vraća je u JSON formatu (SO).
4. Klijent prikazuje tabelu sa kolonama: **ID, Ime, Email, Uloga, Nivo fitnesa, Datum kreiranja, Akcije** (IA).
5. Administrator upisuje tekst u polje **„Search by name…“** da filtrira korisnike po imenu i prezimenu (APUSO).
6. Administrator iz padajuće liste bira filtriranje po ulozi (npr. *member* / *admin*) — tabela se odmah ažurira (APUSO).
7. Administrator menja stranicu paginacije (Prev/Next/1,2,…) i tabela prikazuje po 5 korisnika po strani (APUSO).

* **Alternativni scenariji:**  
  3.1 Ako u bazi nema korisnika, tabela prikazuje poruku **„No users found.“** (IA).  
  2.1 Ako server vrati **HTTP 403**, klijent prikazuje **„Nemate dozvolu.“** i vraća na /login (IA).  
  2.2 Ako server vrati **HTTP 500**, prikazuje se poruka **„Greška na serveru. Lista korisnika nije dostupna.“** (IA).  
  4.1 Ako API odgovori previše sporo, prikazuje se loading indikator dok se podaci ne učitaju (IA).

### SK20: Brisanje korisnika

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je prijavljen i nalazi se na stranici *„Admin · Users“* sa učitanom listom korisnika.
* **Osnovni scenario SK:**

1. Administrator klikne na dugme **„Eksportuj CSV“** (APSO).
2. Klijentska aplikacija prolazi kroz trenutno učitanu listu korisnika (items) i generiše CSV sadržaj sa kolonama *ID, Ime, Email, Uloga* (SO).
3. Sistem kreira privremeni fajl (*Blob*) i automatski pokreće preuzimanje fajla **users\_export.csv** u browseru (IA).
4. Administrator zatim klikne na dugme **„Eksportuj TXT“** (APSO).
5. Klijentska aplikacija formira tekstualni izveštaj, gde je svaki korisnik prikazan u formatu  
   ID: {id} | Ime: {name} | Email: {email} | Uloga: {role} (SO).
6. Sistem kreira *Blob* i pokreće preuzimanje fajla **users\_export.txt** (IA).

* **Alternativni scenariji:**  
  1.1 Ako je lista korisnika prazna u trenutku klika na **„Eksportuj CSV“** ili **„Eksportuj TXT“**, sistem ne generiše fajl, već prikazuje poruku **„Nema podataka za eksport.“** (IA).

### SK21: Eksportovanje podataka o korisnicima (TXT, CSV)

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:**
  + Administrator je prijavljen.
  + Lista korisnika je učitana na stranici **Admin · Users**.
* **Osnovni scenario SK:**

1. **Administrator klikne na dugme „Delete“** pored željenog korisnika u tabeli (APSO).
2. Klijent šalje zahtev **DELETE /api/admin/users/{id}** backendu (APSO).
3. Backend pronalazi korisnika po prosleđenom ID-ju i briše ga iz baze (SO).
4. Backend vraća JSON odgovor o uspešnom brisanju (SO).
5. Klijent uklanja korisnika iz prikazane tabele i prikazuje toast poruku **„Korisnik obrisan.“** (IA).

* **Alternativni scenariji:**

1.1 Ako API vrati **HTTP 403 (Forbidden)**, klijent prikazuje poruku  
**„Nemate dozvolu za ovu akciju.“** (IA).

1.2 Ako API vrati **HTTP 404**, klijent prikazuje poruku  
**„Korisnik nije pronađen.“** (IA).

1.3 Ako API vrati **HTTP 500**, prikazuje se poruka  
**„Greška pri brisanju korisnika.“** (IA).

## Faza analize

**Faza analize** predstavlja ključni korak u razvoju softverskog sistema, u okviru kojeg se detaljno definišu zahtevi i arhitektura pre nego što se pristupi implementaciji. U ovoj fazi prvo se izrađuju sistemski dijagrami sekvenci kako bi se modelirale međusobne interakcije između korisnika (aktora) i sistema, prikazujući tok poruka i kontrolu toka događaja. Paralelno s tim, definiše se i struktura softverskog sistema – organizacija modula, slojeva i međuzavisnosti komponenti, čime se postavlja čvrst temelj za efikasno kodiranje, testiranje i buduće održavanje. Time se obezbeđuje jasna vizija kako će se funkcionalnosti realizovati i kako će različiti delovi sistema korespondirati jedni sa drugima, minimizujući rizik od nesporazuma i tehničkih dugova u narednim fazama projekta (Vlajić, 2020).

### Sistemski dijagrami sekvence

**Sistemski dijagram sekvenci (SDS)** je specifična varijanta UML sekvencnog dijagrama koja naglašava redosled interakcija između jednog ili više spoljašnjih aktera i granice sistema. Dok klasični sekvencni dijagrami vizuelno prikazuju komunikaciju među objektima unutar sistema, SDS-ovi siptuju samo događaje (engl. events) koje akter inicira i odgovore koje sistem vraća, prikazujući ih kroz linije života (lifelines) aktera i centralnog “sistemskog” lifelina. Na ovaj način, SDS pruža jasan prikaz toka poruka u vremenu, gde svaki tok počinje pozivom aktera (APUSO ili ANSO) i završava se sistemskom reakcijom (SO) ili prikazom rezultata na strani klijenta (IA) (Vlajić, 2020).

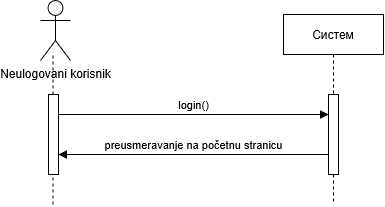
U prvom pasusu SDS-a definiše se opseg: na levi strani dijagrama se pojavljuju aktori (npr. korisnik ili eksterni sistem), a u centru jedno jedino “sistemsko” pojavljivanje, koje predstavlja interni mehanizam obrade poruka i izvršenja poslovne logike. Svaka poruka kojom akter pokreće operaciju označava se strelicom prema sistemu (APUSO ili ANSO), dok strelice natrag predstavljaju sistemske odgovore (SO) ili asinhrone notifikacije klijentu (IA). Ovo pojednostavljuje složene scenarije do nivoa “ulaz-izlaz” toka, eliminišući detalje njihove interne implementacije (Vlajić, 2020).

Drugi pasus SDS-a naglašava primenu u fazi analize i dizajna: SDS-ovi se obično kreiraju za svaki glav­nu poslovnu transakciju ili slučaj upotrebe (use case), dokumentujući osnovni tok i alternativne scenarije (npr. validacija podataka, greške pri autentifikaciji ili timeout). Time postaju temelj za izradu detaljnog modela procesa, omogućavajući razvojnim timovima da se usaglase oko tačnih ulaza, očekivanih izlaza i ponašanja sistema u raznim uslovima, pre nego što krenu u implementaciju ili pisanje koda (Vlajić, 2020).

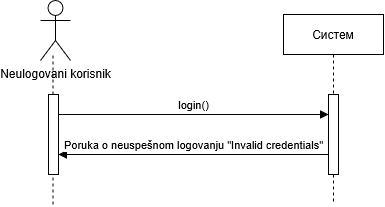
Treći pasus ističe prednosti i nedostatke SDS-a: prednost SDS-a leži u jasnom vremenskom prikazu toka poruka, olakšavajući razumevanje sekvencijalnosti i internih prelaza stanja. Međutim, ukoliko sistem prima veliki broj paralelnih poruka ili složene asinhrone događaje, SDS-ovi mogu postati pretrpani i zauzimati previše horizontalnog prostora, što otežava čitljivost. U ovakvim slučajevima, dizajneri mogu preći na dijagrame saradnje (collaboration diagrams), koji su fleksibilniji u prikazu čvorišta i veza, ali manje intuitivni za vremensku analizu.

### DS1: Prijava korisnika

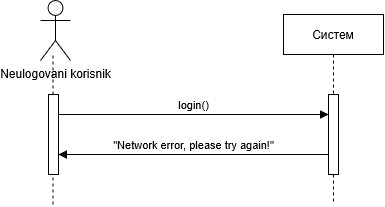
* **Aktori SK:** Neulogovani korisnik
* **Učesnici SK:** Neulogovani korisnik i sistem
* **Preduslov:** Korisnik ima registrovan nalog i validan email/lozinka.
* **Osnovni scenario SK:**
  1. Korisnik otvara stranicu za prijavu **(APUSO).**
  2. Unosi email i lozinku u formu **(APUSO).**
  3. Klikne na dugme „Prijavi se“ **(APSO).**
  4. Sistem proverava kredencijale i generiše access\_token **(SO).**
  5. Sistem vraća token, klijent ga čuva u sessionStorage i preusmerava korisnika na početnu stranicu odgovarajuće uloge **(IA).**



* **Alternativni scenariji:**  
  **4.1** Ako su kredencijali neispravni, API vraća HTTP 401, klijent prikazuje “Invalid credentials” **(IA).**

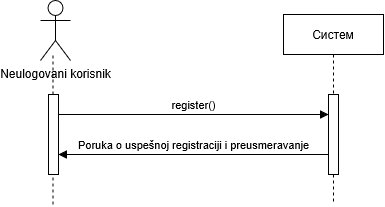


**4.2** Ako server ne odgovori (timeout), klijent prikazuje “Network error, please try again” **(IA).**

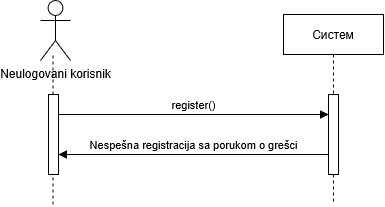
****

### DS2: Registracija korisnika

* **Aktori SK:** Neulogovani korisnik
* **Učesnici SK:** Neulogovani korisnik i sistem
* **Preduslov:** Korisnik nema postojeći nalog.
* **Osnovni scenario SK:**
  1. Korisnik otvara stranicu „Login“ **(APUSO).**
  2. Popunjava sve obavezne podatke:
     + Ime i prezime (text)
     + Email adresa (email)
     + Lozinka (password)
     + Uloga (dropdown: member , admin)
     + Fitness nivo (dropdown: beginner, intermediate, expert)
  3. Klikne na dugme „Registruj se“ **(APSO).**
  4. Sistem validira unos i kreira novi nalog u bazi **(SO).**
  5. Sistem vraća poruku o uspehu i preusmerava korisnika na stranicu za prijavu **(IA).**

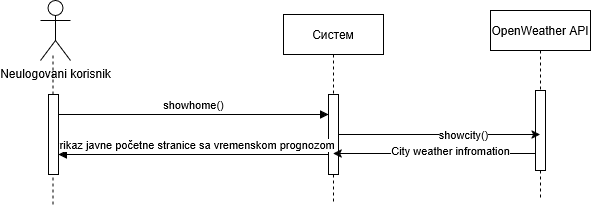
****

* **Alternativni scenariji:**  
  **4.1** Ako neki unos nije validan (npr. email već postoji, lozinke se ne poklapaju…), API vraća HTTP 400 sa detaljima grešaka **(IA).**



### DS3: Prikaz javne početne stranice

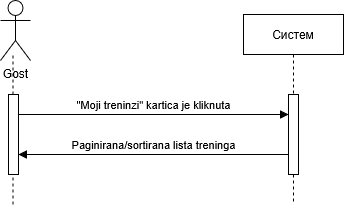
* **Aktori SK:** Neulogovani korisnik / posetilac
* **Učesnici SK:** Neulogovani korisnik i sistem
* **Preduslov:** Korisnik nije prijavljen i otvara osnovni URL aplikacije.
* **Osnovni scenario SK:**
  1. Korisnik u pregledaču otvara početni URL aplikacije ili klikne na link ka sajtu (**APUSO**).
  2. Sistem učitava React početnu stranicu i inicijalne podatke (logo, navigaciju, tekst dobrodošlice) (**SO**).
  3. Klijent prikazuje javni „hero“ deo sa opisom Rebel Fitness aplikacije, glavnim menijem i CTA dugmićima „Login“, „Registruj se“ i „Nastavi kao gost“ (**IA**).
  4. Sistem, ukoliko je uključeno održavanje, prikazuje modal „PAŽNJA! Zbog radova na održavanju sajta…“ koji korisnik potvrđuje klikom na dugme „OK“ ili zatvaranjem modala (**IA/ANSO**).



### DS4: Pregled treninga kao gost

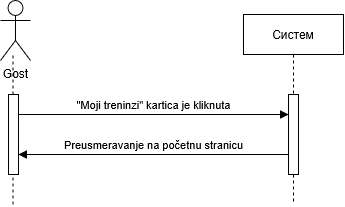
* **Aktori SK:** Gost (ulogovani korisnik sa ulogom *guest*)
* **Učesnici SK:** Gost i sistem
* **Preduslov:** Korisnik je uspešno ulogovan preko brzog ulaza kao gost (validan *access\_token* i uloga *guest*).
* **Osnovni scenario SK:**

1. Gost klikne na stavku menija „Workouts“ / „Moji treninzi“ na početnoj stranici ( **ANSO** ).
2. Klijent šalje zahtev GET /api/workouts sa *Bearer* tokenom gosta u zaglavlju ( **APSO** ).
3. Sistem preko trenutno ulogovanog korisnika ( Auth::user() ) pronalazi sve treninge vezane za tog korisnika i priprema paginiranu listu rezultata ( **SO** ).
4. Sistem vraća JSON listu treninga (ID, naziv, opis, trajanje, kalorije, status…) a klijent renderuje grid kartica sa osnovnim informacijama o svakom treningu, bez mogućnosti uređivanja ili čuvanja promena za guest nalog ( **IA** ).



* **Alternativni scenariji:**

2.1 Ako API vrati HTTP 401 (istekao ili neispravan token), klijent briše lokalne podatke o gostu i preusmerava korisnika na javnu početnu stranicu sa porukom o potrebi ponovnog ulaska kao gost ( **IA** ).



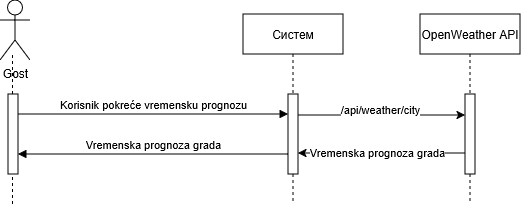
3.1 Ako sistem vrati praznu listu treninga, klijent umesto grida prikazuje informativnu poruku da za gost nalog trenutno nema dostupnih treninga za prikaz ( **IA** ).



### DS5: Pregled vremenske prognoze grada (gost)

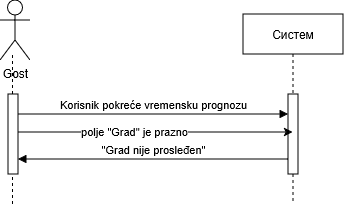
* **Aktori SK:** Gost
* **Učesnici SK:** Gost i sistem
* **Preduslov:** Gost je uspešno ušao u aplikaciju (Guest login) i nalazi se na stranici sa vidžetom za vremensku prognozu.
* **Osnovni scenario SK:**

1. Gost otvara sekciju **„Vremenska prognoza“** na stranici ( **ANSO** ).
2. U polje **„Grad“** unosi naziv željenog grada (npr. *Belgrade*, *Novi Sad*) ( **APUSO** ).
3. Gost klikne na dugme **„Prikaži vreme“** ( **APSO** ).
4. Klijent šalje zahtev **GET /api/weather/{city}** gde je *{city}* URL-enkodovan naziv grada ( **APSO** ).
5. Backend kontroler **WeatherController** poziva eksterni servis **OpenWeather API**, obrađuje odgovor i mapira ga na polja: *city, temperature, weather, humidity, wind\_speed* ( **SO** ).
6. Sistem vraća JSON odgovor sa podacima o vremenu za traženi grad ( **SO** ).
7. Klijent prikazuje karticu sa rezultatima: naziv grada, trenutna temperatura, opis vremenskih uslova, vlažnost vazduha i brzina vetra ( **IA** ).

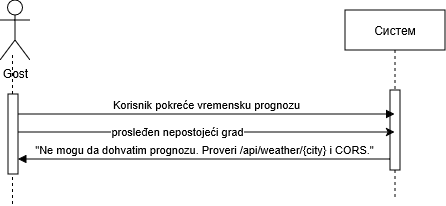


* **Alternativni scenariji:**

5.1 Ako je polje za grad prazno, klijentska logika vraća grešku *„Grad nije prosleđen.“* i ne šalje zahtev ka API-ju ( **IA** ).



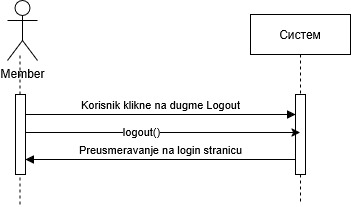
5.2 Ako OpenWeather API vrati grešku (nepostojeći grad, problem sa mrežom ili CORS), backend prosleđuje grešku, a klijent prikazuje poruku:  
**„Ne mogu da dohvatim prognozu. Proveri /api/weather/{city} i CORS.“** ( **IA** ).



### DS6: Odjava člana

* **Aktori SK:** Ulogovani korisnik – član (Member)
* **Učesnici SK:** Član i sistem
* **Preduslov:** Korisnik je uspešno prijavljen i ima validan token u localStorage.
* **Osnovni scenario SK:**

1. Član klikne na stavku menija **„Logout“** ( **ANSO** ).
2. Klijent šalje zahtev **POST /logout** ka serveru uz Bearer token ( **APSO** ).
3. Backend (AuthController) poništava korisnikov token i vraća potvrdu o uspešnom odjavljivanju ( **SO** ).
4. Klijent uklanja *token* i *user* iz localStorage i automatski preusmerava korisnika na stranicu **/login** ( **IA** ).

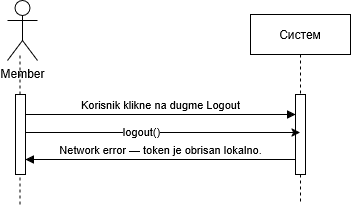
****

* **Alternativni scenariji:**

3.1 Ako server ne može da poništi token (istekao, nevažeći ili već obrisan), API vraća grešku — ali klijent svejedno briše lokalne podatke i prikazuje poruku  
**„“** ( **IA** ).

****

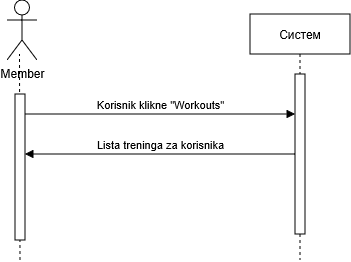
3.2 Ako postoji mrežna greška (timeout), klijent obavlja lokalnu odjavu i prikazuje  
**„Network error — token je obrisan lokalno.“** ( **IA** ).

****

### DS7: Pregled svojih treninga

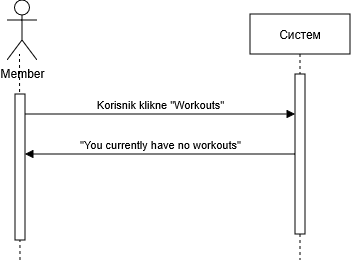
* **Aktori SK:** Ulogovani korisnik – član (Member)
* **Učesnici SK:** Član i sistem
* **Preduslov:** Korisnik je uspešno prijavljen i ima važeći Bearer token. Postoje ili ne postoje sačuvani treninzi u bazi.
* **Osnovni scenario SK:**

1. Član klikne na stavku menija **„Workouts“** ( **ANSO** ).
2. Klijent šalje zahtev **GET /workouts** backendu uz odgovarajući Authorization header ( **APSO** ).
3. Backend (WorkoutController@index) pronalazi sve treninge koji pripadaju **ulogovanom korisniku** i vraća listu u JSON formatu ( **SO** ).
4. Klijent renderuje grid/listu treninga, pri čemu se za svaki prikazuje:
   * naziv treninga
   * cilj (goal) ako postoji
   * broj vežbi
   * datum kreiranja
   * dugmad **Edit** i **Delete** ( **IA** ).

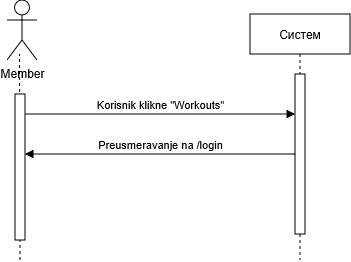
****

**• Alternativni scenariji:**

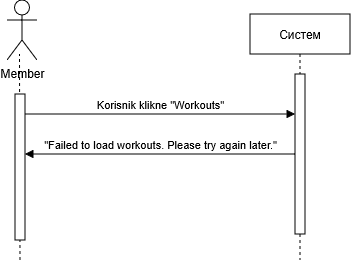
3.1 Ako korisnik nema nijedan trening, API vraća prazan niz, a klijent prikazuje poruku:  
**„You currently have no workouts.“** ( **IA** ).



3.2 Ako API vrati **HTTP 401** (istekao token), klijent briše token iz localStorage i preusmerava korisnika na **/login** ( **IA** ).



3.3 Ako dođe do greške na serveru (HTTP 500), klijent prikazuje poruku  
**„“** ( **IA** ).

****

### DS8: Kreiranje treninga

* **Učesnici SK:** Član i sistem
* **Preduslov:** Korisnik je prijavljen kao *member* ili *admin* i ima važeći token; korisnik se nalazi na stranici sa listom treninga ili koristi link **„+ Novi trening“** / putanju **/workouts/new**.
* **Osnovni scenario SK:**

1. Korisnik klikne na dugme/link **„+ Novi trening“** u okviru sekcije *Workouts* ( **ANSO** ).
2. Klijent otvara stranicu **„Novi trening“** sa formom za unos podataka:
   * *Naziv* (obavezno polje, tekst),
   * *Opis* (opciono polje, tekst),
   * *Trajanje (min)* (opciono, broj ≥ 0),
   * *Kalorije (kcal)* (opciono, broj ≥ 0),
   * *Status* (dropdown: *pending*, *started*, *completed*) ( **IA** ).
3. Korisnik popunjava polja forme i unosi naziv treninga ( **APUSO** ).
4. Korisnik klikne na dugme **„Sačuvaj“** ( **APSO** ).
5. Klijent šalje zahtev **POST /users/workouts** backendu sa JSON telom koje sadrži:  
   { name, description, duration, calories\_burned, status } i Authorization header ( **APSO** ).
6. Backend (WorkoutController@store) validira podatke, kreira novi trening vezan za ulogovanog korisnika i čuva ga u bazi ( **SO** ).
7. Sistem vraća JSON odgovor sa novokreiranim treningom i statusom uspeha ( **SO** ).
8. Klijent na osnovu odgovora preusmerava korisnika nazad na listu treninga **/workouts**, gde se novi trening prikazuje u listi ( **IA** ).

* **Alternativni scenariji:**

4.1 Ako korisnik pokuša da pošalje formu bez unetog naziva, frontend validacija vraća poruku  
**„Naziv je obavezan.“** i sprečava slanje zahteva ka serveru ( **IA** ).

4.2 Ako korisnik unese negativnu vrednost za trajanje ili kalorije (npr. -10), klijent prikazuje odgovarajuću poruku o grešci:  
**„Trajanje mora biti nenegativan broj.“** ili **„Kalorije moraju biti nenegativan broj.“** i ne šalje zahtev ( **IA** ).

6.1 Ako backend vrati **HTTP 422** zbog nevalidnih podataka (npr. predugačak naziv ili neispravan format), klijent prikazuje poruku  
**„Kreiranje nije uspelo. Proveri privilegije i backend.“** ( **IA** ).

6.2 Ako backend vrati **HTTP 401** ili **403** (nema privilegije ili istekao token), klijent može obrisati lokalno sačuvani token i preusmeriti korisnika na **/login** ( **IA** ).

6.3 Ako dođe do interne greške servera (npr. **HTTP 500**), klijent prikazuje generičku poruku o grešci i ostavlja korisnika na formi za unos kako bi mogao da pokuša ponovo ( **IA** ).

### DS9: Izmena postojećeg treninga

* **Aktori SK:** Ulogovani korisnik – član (**Member**)
* **Učesnici SK:** Član i sistem
* **Preduslov:** 
  + Korisnik je prijavljen kao *member* ili *admin* i ima važeći token.
  + Trening koji se menja postoji u bazi i pripada tom korisniku (ili je admin).
  + Korisnik se nalazi na listi treninga (*Workouts*), npr. na stranici **/workouts** ili **/users/workouts**.
* **Osnovni scenario SK:**

1. Korisnik na listi treninga klikne na dugme/link **„Uredi“** pored željenog treninga ( **ANSO** ).
2. Klijent preusmerava korisnika na rutu **/workouts/{id}/edit** i šalje zahtev **GET /workouts/{id}** da učita detalje treninga ( **APSO** ).
3. Sistem (backend – WorkoutController@show) pronalazi traženi trening i vraća JSON sa poljima  
   name, description, duration, calories\_burned, status ( **SO** ).
4. Klijent popunjava formu za izmenu treninga postojećim vrednostima:
   * *Naziv* (obavezno),
   * *Opis* (opciono),
   * *Trajanje (min)* (opciono, broj ≥ 0),
   * *Kalorije (kcal)* (opciono, broj ≥ 0),
   * *Status* (dropdown: *pending*, *started*, *completed*) ( **IA** ).
5. Korisnik menja željena polja u formi (npr. naziv, trajanje, status) ( **APUSO** ).
6. Korisnik klikne na dugme **„Sačuvaj izmene“** ( **APSO** ).
7. Klijent šalje zahtev **PUT /users/workouts/{id}** sa JSON telom:  
   { name, description, duration, calories\_burned, status } i **Authorization: Bearer {token}** ( **APSO** ).
8. Backend (WorkoutController@update) validira podatke, proverava da li korisnik ima pravo izmene, ažurira zapis u bazi i vraća osvežen resurs ( **SO** ).
9. Klijent na osnovu uspešnog odgovora preusmerava korisnika nazad na stranicu sa listom treninga **/workouts**, gde je izmenjeni trening prikazan sa novim vrednostima ( **IA** ).

* **Alternativni scenariji:**

3.1 Ako trening ne postoji ili nije dostupan korisniku (npr. traži tuđi ID), backend vraća **HTTP 404** ili **403**, a klijent prikazuje poruku  
**„Ne mogu da učitam ovaj trening.“** i može ponuditi povratak na listu treninga ( **IA** ).

5.1 Ako korisnik obriše naziv treninga i pokuša da sačuva, klijent javlja grešku validacije  
**„Naziv je obavezan.“** i ne šalje zahtev ka serveru dok polje nije ispravljeno ( **IA** ).

5.2 Ako korisnik unese negativno trajanje ili kalorije (npr. -5), klijent prikazuje poruke:  
**„Trajanje mora biti nenegativan broj.“** ili **„Kalorije moraju biti nenegativan broj.“** i sprečava slanje forme ( **IA** ).

8.1 Ako backend pri ažuriranju vrati **HTTP 422** (nevalidni podaci) ili **HTTP 500** (interna greška), klijent prikazuje poruku  
**„Ažuriranje nije uspelo. Pokušaj ponovo.“** i ostavlja korisniku mogućnost da ispravi podatke ili proba ponovo slanje ( **IA** ).

8.2 Ako backend vrati **HTTP 401** ili **403** (istekao token ili nema privilegije), klijent može obrisati lokalno sačuvan token, obavestiti korisnika da nema pristup i preusmeriti ga na stranicu **/login** ( **IA** ).

### DS10: Brisanje treninga

* **Aktori SK:** Ulogovani korisnik – član (**Member**) ili administrator (**Admin**)
* **Učesnici SK:** Član / administrator i sistem
* **Preduslov:**
  + Korisnik je prijavljen (*member* ili *admin*) i ima važeći token.
  + Trening koji se briše postoji u bazi i pripada tom korisniku (ili je admin).
  + Korisnik se nalazi na listi treninga (stranica **/workouts**).
* **Osnovni scenario SK:**

1. Korisnik na listi treninga pronalazi trening koji želi da obriše i klikne na dugme **„Obriši“** pored tog treninga ( **ANSO** ).
2. Sistem (frontend) otvara modalni prozor za potvrdu sa tekstom tipa:  
   **„Da li sigurno želiš da obrišeš ovaj trening?“** i prikazuje dugmad **„Ne“** i **„Da, obriši“** ( **IA** ).
3. Korisnik klikne na dugme **„Da, obriši“** u okviru modala ( **APSO** ).
4. Klijent šalje zahtev **DELETE /users/workouts/{id}** sa odgovarajućim id treninga i zaglavljem  
   **Authorization: Bearer {token}** ka backendu (WorkoutController@destroy) ( **APSO** ).
5. Backend proverava da li korisnik ima pravo da obriše taj trening (vlasnik ili admin), briše zapis iz baze i vraća odgovor o uspehu ( **SO** ).
6. Klijent, po uspešnom odgovoru servera, uklanja obrisani trening iz lokalne liste items i ažurira prikaz kartica na stranici **/workouts** ( **IA** ).
7. Modal se zatvara, a korisnik na ekranu više ne vidi obrisani trening u listi ( **IA** ).

* **Alternativni scenariji:**

2.1 Ako korisnik u modalu klikne na dugme **„Ne“**, modal se zatvara, ne šalje se nikakav zahtev ka serveru i lista treninga ostaje neizmenjena ( **IA** ).

5.1 Ako backend vrati **HTTP 403** (korisnik nema pravo da obriše taj trening), klijent prikazuje poruku tipa  
**„Nemate dozvolu za ovu akciju.“**, a trening ostaje u listi ( **IA** ).

5.2 Ako backend vrati **HTTP 404** (trening ne postoji) ili **HTTP 500** (interna greška), klijent obaveštava korisnika porukom tipa  
**„Brisanje nije uspelo. Pokušaj ponovo.“** i ne uklanja trening iz liste ( **IA** ).

5.3 Ako server vrati **HTTP 401** (istekao ili nevažeći token), klijent može obrisati lokalni token, prikazati poruku  
**„Sesija je istekla, prijavite se ponovo.“** i preusmeriti korisnika na stranicu **/login** ( **IA** ).

### DS11: Kreiranje nove vežbe

* **Aktori SK:** Ulogovani korisnik – član (**Member**) ili administrator (**Admin**)
* **Učesnici SK:** Član / administrator i sistem
* **Preduslov:**
  + Korisnik je prijavljen (*member* ili *admin*) i ima važeći token.
  + Korisnik ima barem jedan postojeći trening (**workout**) u sistemu ili zna workout\_id kome želi da doda vežbu.
  + Korisnik se nalazi na stranici za rad sa vežbama (**/exercises**) ili direktno na ruti **/exercises/new**.
* **Osnovni scenario SK:**

1. Korisnik na stranici **„Exercises“** klikne na dugme **„Nova vežba“** ili direktno otvara rutu **/exercises/new** ( **ANSO** ).
2. Sistem (frontend) prikazuje formu za kreiranje vežbe sa poljima:
   * *Naziv* (obavezno tekstualno polje),
   * *Opis* (opciono tekstualno polje),
   * *Ponavljanja/Vreme* (obavezno numeričko polje),
   * *Tip* (padajuća lista: **cardio**, **strength**, **flexibility**),
   * *Workout* (padajuća lista dostupnih treninga i opciono ručni unos workout\_id) ( **IA** ).
3. Korisnik popunjava polja forme: unosi naziv, opis (po želji), broj ponavljanja/vremena, bira tip vežbe i bira željeni trening iz liste (ili ručno unosi workout\_id) ( **APUSO** ).
4. Klijent obavlja osnovnu validaciju na strani browsera – proverava da **naziv nije prazan**, da je *Ponavljanja/Vreme* validan broj i da je odabran trening ( **ANSO** ).
5. Korisnik klikne na dugme **„Kreiraj“** u dnu forme ( **APSO** ).
6. Klijent šalje zahtev **POST /exercises** sa JSON telom u formatu npr.:  
   { name, description, reps\_or\_time, type, workout\_id } i zaglavljem  
   **Authorization: Bearer {token}** ka backendu (ExerciseController@store) ( **APSO** ).
7. Backend validira podatke, kreira novi zapis vežbe u bazi i povezuje ga sa izabranim treningom, zatim vraća odgovor o uspehu ( **SO** ).
8. Klijent po uspešnom odgovoru preusmerava korisnika nazad na stranicu **/exercises**, automatski osvežava listu vežbi i nova vežba se pojavljuje u tabeli ( **IA** ).
9. Korisnik na listi vežbi vidi novu vežbu sa unetim nazivom, tipom, brojem ponavljanja/vremena i pripadajućim treningom ( **IA** ).

* **Alternativni scenariji:**

4.1 Ako korisnik pokuša da pošalje formu sa praznim *Nazivom*, nevalidnim brojem u polju *Ponavljanja/Vreme* ili bez odabranog treninga, frontend prikazuje poruku tipa  
**„Naziv je obavezan.“, „Ponavljanja/Vreme mora biti broj.“ ili „Morate odabrati workout.“** i ne šalje zahtev backendu ( **IA** ).

7.1 Ako backend vrati **HTTP 422** (npr. nevalidni podaci, nepostojeći workout\_id), sistem vraća poruke o greškama, a klijent ih prikazuje ispod odgovarajućih polja u formi ( **IA** ).

7.2 Ako backend vrati **HTTP 401** (istekao ili nevažeći token), klijent može obrisati lokalni token, prikazati poruku  
**„Sesija je istekla, prijavite se ponovo.“** i preusmeriti korisnika na stranicu **/login** ( **IA** ).

7.3 Ako backend vrati **HTTP 500** (interna greška servera), klijent prikazuje obaveštenje tipa  
**„Kreiranje vežbe nije uspelo. Pokušaj ponovo.“** i ostavlja korisnika na formi sa sačuvanim unosom ( **IA** ).

### DS12: Izmena vežbe

* **Aktori SK:** Ulogovani korisnik – član (**Member**) ili administrator (**Admin**)
* **Učesnici SK:** Korisnik i sistem
* **Preduslov:**
  + Korisnik je prijavljen (validan token).
  + Vežba koju korisnik želi da izmeni postoji u bazi.
  + Korisnik ima pravo izmene (član može menjati samo *svoje* vežbe – one koje pripadaju njegovim treninzima; admin može menjati sve).
  + Korisnik se nalazi na stranici **/exercises** ili direktno na ruti **/exercises/edit/{id}**.
* **Osnovni scenario SK:**

1. **Korisnik klikne** na dugme **„Edit“** pored postojeće vežbe u tabeli vežbi ( **ANSO** ).
2. Klijent otvara formu za izmenu, u kojoj su **polja već popunjena** trenutnim vrednostima vežbe:
   * Naziv,
   * Opis,
   * Ponavljanja/Vreme,
   * Tip (cardio/strength/flexibility),
   * Workout kojem vežba pripada ( **IA** ).
3. Korisnik menja željena polja – npr. ispravlja naziv, povećava broj ponavljanja, menja tip ili prebacuje vežbu na drugi trening ( **APUSO** ).
4. Forma obavlja klijentsku validaciju – proverava da naziv nije prazan, da reps/time sadrži broj i da je izabran važeći trening ( **ANSO** ).
5. Korisnik klikne na dugme **„Sačuvaj izmene“** ( **APSO** ).
6. Klijent šalje zahtev **PUT /exercises/{id}** sa JSON podacima:  
   { name, description, reps\_or\_time, type, workout\_id } i Bearer token zaglavljem ka backendu (ExerciseController@update) ( **APSO** ).
7. Backend validira podatke, proverava da li korisnik ima pravo na izmenu, zatim ažurira vežbu u bazi i vraća izmenjeni resource ( **SO** ).
8. Klijent po uspešnom odgovoru osvežava listu vežbi, prikazuje ažurirane vrednosti i obaveštenje tipa  
   **„Vežba uspešno izmenjena.“** ( **IA** ).
9. Korisnik ostaje na stranici **/exercises**, gde vidi novu verziju iste vežbe ( **IA** ).

* **Alternativni scenariji:**

**4.1** Ako korisnik ostavi *Naziv* praznim ili unese nevalidan broj u *Ponavljanja/Vreme*, forma prikazuje poruke:

* „Naziv je obavezan.“
* „Ponavljanja/Vreme mora biti broj.“  
  i sprečava slanje zahteva ( **IA** ).

**7.1** Ako backend vrati **HTTP 422 (Validation error)**, klijent prikazuje server-side poruke grešaka ispod polja ( **IA** ).

**7.2** Ako backend vrati **HTTP 403 (Forbidden)** jer korisnik pokušava da izmeni vežbu koja ne pripada njegovom treningu, klijent prikazuje poruku:  
**„Nemate dozvolu da izmenite ovu vežbu.“** ( **IA** ).

**7.3** Ako backend vrati **HTTP 401 (Unauthorized)** zbog isteka tokena, klijent briše token i redirektuje korisnika na **/login** ( **IA** ).

**7.4** Ako backend vrati **HTTP 500 (Server error)**, prikazuje se poruka:  
**„Izmena nije uspela. Pokušajte ponovo.“** ( **IA** ).

### DS13: Brisanje vežbe

* **Aktori SK:** Ulogovani korisnik – član (**Member**) ili administrator (**Admin**)
* **Učesnici SK:** Korisnik i sistem
* **Preduslov:**
  + Korisnik je prijavljen (validan token postoji u localStorage).
  + Vežba koju korisnik želi da obriše postoji u bazi.
  + Korisnik ima pravo brisanja (član može brisati samo vežbe iz svojih treninga; admin može brisati bilo koju vežbu).
  + Korisnik se nalazi na stranici **„Exercises“** (**/exercises**) i vidi listu vežbi.
* **Osnovni scenario SK:**

1. **Korisnik klikne** na dugme **„Obriši“** pored vežbe koju želi da ukloni sa liste ( **ANSO** ).
2. Klijent otvara modal za potvrdu brisanja sa tekstom tipa:  
   **„Da li sigurno želiš da obrišeš ovu vežbu?“** i dugmadima **„Ne“** i **„Da, obriši“** ( **IA** ).
3. Korisnik potvrđuje akciju klikom na dugme **„Da, obriši“** u modalu ( **APSO** ).
4. Klijent šalje zahtev **DELETE /exercises/{id}** backendu, uz Bearer token u zaglavlju, koristeći deleteExercise(id) funkciju ( **APSO** ).
5. Backend proverava prava pristupa (da li korisnik sme da obriše tu vežbu) i, u slučaju da je sve ispravno, briše vežbu iz baze ( **SO** ).
6. Sistem vraća odgovor o uspehu brisanja (npr. status 200 i poruku) ( **SO** ).
7. Klijent zatvara modal, uklanja obrisanu vežbu iz prikazane liste i prikazuje poruku tipa:  
   **„Vežba je uspešno obrisana.“** ( **IA** ).

* **Alternativni scenariji:**

**2.1** Korisnik klikne na dugme **„Ne“** u modalu: modal se zatvara, ne šalje se nikakav zahtev ka serveru i lista vežbi ostaje nepromenjena ( **IA** ).

**5.1** Ako backend vrati **HTTP 403 (Forbidden)** jer korisnik nema pravo da obriše tu vežbu (npr. vežba pripada tuđem treningu), klijent prikazuje poruku:  
**„Nemate dozvolu da obrišete ovu vežbu.“** i lista ostaje nepromenjena ( **IA** ).

**5.2** Ako backend vrati **HTTP 401 (Unauthorized)** zbog nevalidnog ili isteknutog tokena, klijent briše lokalni token i preusmerava korisnika na stranicu **/login** ( **IA** ).

**5.3** Ako backend vrati **HTTP 404 (Not Found)** jer vežba više ne postoji u bazi, klijent uklanja vežbu iz liste (ako je još prikazana) i prikazuje poruku:  
**„Vežba nije pronađena.“** ( **IA** ).

**5.4** Ako backend vrati **HTTP 500 (Server error)** ili dođe do mrežne greške, klijent prikazuje generičku poruku:  
**„Brisanje nije uspelo. Pokušaj ponovo kasnije.“** i ne menja listu vežbi ( **IA** ).

### DS14: Pregled vežbi (filtriranje i sortiranje)

* **Aktori SK:** Ulogovani korisnik – član (**Member**) ili administrator (**Admin**)
* **Učesnici SK:** Korisnik i sistem
* **Preduslov:**
  + Korisnik je prijavljen (validan token postoji u localStorage).
  + Korisnik ima ulogu *member* ili *admin* (guest nema pristup stranici **Exercises**).
  + U bazi postoje registrovane vežbe (ili će se u alternativnom scenariju prikazati poruka da nema podataka).
  + Korisnik ima pristup stranici **„Exercises“** (/exercises).
* **Osnovni scenario SK:**

1. **Korisnik klikne** na stavku menija **„Exercises“** u navigaciji aplikacije ( **ANSO** ).
2. Klijent (React aplikacija) šalje inicijalni zahtev ka backendu:  
   **GET /exercises?page=1&per\_page=10&sort=name&dir=asc** (podrazumevani parametri) koristeći funkciju fetchExercises(params) ( **APSO** ).
3. Backend obrađuje zahtev, primenjuje zadate parametre paginacije i sortiranja, čita podatke iz baze i vraća rezultat u obliku paginiranog JSON odgovora (lista vežbi + meta podaci o strani, ukupnom broju itd.) ( **SO** ).
4. Klijent prikazuje tabelu sa vežbama: kolone **ID**, **Naziv**, **Opis**, **Ponavljanja/Vreme**, **Tip**, **Workout**, kao i eventualne akcije (Uredi/Obriši) za ulogovanog člana ili admina ( **IA** ).
5. Korisnik po potrebi unosi tekst u polje **„Pretraga“** (npr. deo naziva ili opisa vežbe) ( **APUSO** ).
6. Klijent ponovno poziva backend:  
   **GET /exercises?search={query}&page=1&per\_page={trenutni}&sort={sort}&dir={dir}&type={typeFilter}&mine={0/1}** i osvežava listu vežbi na osnovu unetog pojma ( **APSO** ).
7. Korisnik bira tip vežbe iz padajuće liste **„Tip“** (npr. cardio, strength, flexibility) ( **APUSO** ).
8. Klijent prosleđuje novi zahtev backendu sa izabranim tipom kao filterom (**type=cardio/strength/flexibility**) i prikazuje samo vežbe tog tipa ( **APSO**, **SO**, **IA** ).
9. Korisnik opcionalno uključuje čekboks **„Samo moji“** da bi video samo vežbe koje su vezane za njegove treninge ( **APUSO** ).
10. Klijent ponovo šalje zahtev **GET /exercises?mine=1&...**; backend filtrira vežbe po ulogovanom korisniku i vraća rezultat ( **APSO**, **SO** ).
11. Klijent prikazuje novu listu – samo vežbe povezane sa treninzima tog korisnika, uz primenjene ostale filtere (tip, pretraga) ( **IA** ).
12. Korisnik menja kriterijum sortiranja preko padajućeg menija **„Sortiraj po“** (npr. name, type, reps\_or\_time, id) i po potrebi klikne na dugme za promenu smera sortiranja (**↑ / ↓**) ( **APUSO** ).
13. Klijent šalje novi zahtev sa parametrima **sort={polje}&dir={asc/desc}**, backend vraća sortiranu listu vežbi, a klijent osvežava tabelu ( **APSO**, **SO**, **IA** ).
14. Korisnik prilagođava broj zapisa po strani preko **„Po strani“** (npr. 5, 10, 20) i koristi dugmad **„Prethodna“** / **„Sledeća“** za navigaciju između strana ( **APUSO** ).
15. Klijent na svaku promenu stranice šalje odgovarajući **GET /exercises?page={n}&per\_page={pageSize}&…** zahtev, a sistem vraća traženu stranu rezultata; klijent prikazuje aktuelnu stranu i pokazuje indikator **„Strana X / Y“** ( **APSO**, **SO**, **IA** ).

* **Alternativni scenariji:**

**3.1** Ako backend vrati **HTTP 401 (Unauthorized)** zbog nevalidnog ili isteknutog tokena, klijent briše token i podatke o korisniku iz localStorage i preusmerava korisnika na stranicu **/login** ( **IA** ).

**3.2** Ako backend vrati **HTTP 500** ili dođe do mrežne greške, klijent prikazuje poruku tipa:  
**„Ne mogu da učitam vežbe. Pokušaj ponovo.“** i ostavlja prethodno stanje ekrana ( **IA** ).

**4.1** Ako sistem ne pronađe nijednu vežbu (bilo inicijalno, bilo nakon filtriranja), klijent prikazuje poruku u okviru kartice:  
**„Nema vežbi za prikaz.“** ( **IA** ).

**11.1** Ako je čekboks **„Samo moji“** uključen, ali korisnik nema nijednu vežbu vezanu za svoje treninge, lista ostaje prazna uz poruku **„Nema vežbi za prikaz.“** ( **IA** ).

**14.1** Ako korisnik postavi veliki broj po strani i poslednja strana nakon brisanja nekih vežbi više ne postoji, klijent automatski vraća korisnika na prethodnu validnu stranu i osvežava podatke ( **IA** ).

### DS15: Pregled ciljeva

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je uspešno prijavljen u sistem i ima dodeljenu ulogu admin.
* **Osnovni scenario SK:**
  1. Administrator u gornjoj navigaciji klikne na stavku menija **„Goals“** (ANSO).
  2. Klijent (React aplikacija) šalje zahtev **GET /api/goals** ka Laravel backendu, uz **Authorization: Bearer {token}** zaglavlje (APSO).
  3. Sistem (Laravel GoalController@index) čita sve zapise iz tabele goals preko Eloquent modela Goal i vraća JSON listu ciljeva (SO).
  4. Klijent renderuje stranicu **„Admin · Goals“**: u tabeli prikazuje po redu kolone **ID**, naziv cilja, skraćeni opis, korisnika (ime ili ID), rok (datum) i status cilja, uz akcije **„Uredi“** i **„Obriši“** za svaki red (IA).
* **Alternativni scenariji:**  
  2.1. Ako backend vrati **HTTP 401** ili **HTTP 403** (nevalidan token ili nedovoljno privilegija), klijent prikazuje poruku:  
  „**Ne mogu da učitam ciljeve. Proveri backend ili privilegije.**“ i onemogućava rad sa tabelom (IA).
* 3.1. Ako u bazi ne postoji nijedan cilj (prazan odgovor), klijent umesto tabele prikazuje prazno stanje sa tekstom:  
  „**Trenutno nema ciljeva.**“ unutar kartice (IA).
* 2.2. Ako dođe do mrežne greške ili neočekivane greške servera (npr. **HTTP 500**), klijent prikazuje istu poruku o grešci i omogućava administratoru da klikom na dugme **„Pokušaj ponovo“** ponovo pošalje zahtev za učitavanje ciljeva (IA).

### DS16: Kreiranje novog cilja

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je uspešno prijavljen u sistem i ima ulogu admin. Stranica **„Admin · Goals“** je učitana ili administrator kroz meni otvara sekciju za ciljeve.
* **Osnovni scenario SK:**

1. Administrator na stranici **„Admin · Goals“** klikne na dugme **„+ Novi cilj“** (ANSO).
2. Klijent otvara formu **„Novi cilj“** (komponenta *AdminGoalForm*) sa praznim poljima za **naziv**, **opis**, **rok (datum)**, izbor **korisnika** iz padajuće liste i početni **status** cilja (IA).
3. Sistem (frontend) automatski šalje zahtev **GET /api/admin/users** kako bi popunio dropdown sa listom korisnika (SO), a klijent prikazuje korisnike u padajućoj listi (IA).
4. Administrator popunjava polja forme:
   * naziv cilja (obavezno),
   * opcioni opis,
   * rok cilja (datum),
   * bira korisnika kome se cilj dodeljuje (obavezno),
   * status cilja (npr. *pending* ili *completed*) (APUSO).
5. Administrator klikne na dugme **„Kreiraj“** kako bi sačuvao novi cilj (APSO).
6. Klijent šalje zahtev **POST /api/goals** sa JSON telom koje sadrži naslov, opis, ID korisnika i datum (polja title, description, user\_id, kao i datum u formatu target\_date / due\_date / deadline), zajedno sa **Authorization: Bearer {token}** zaglavljem (APSO).
7. Sistem (Laravel GoalController@store) validira podatke, kreira novi zapis u tabeli goals i vraća JSON reprezentaciju novog cilja sa generisanim ID-jem i ostalim poljima (SO).
8. Klijent po uspešnom odgovoru zatvara formu i preusmerava administratora nazad na stranicu **„Admin · Goals“**, gde se lista ciljeva ponovo učitava i prikazuje uključujući novokreirani cilj (IA).

* **Alternativni scenariji:**   
  4.1. Ako administrator pokuša da pošalje formu bez obaveznog naziva ili bez odabranog korisnika, frontend detektuje grešku i prikazuje tekstualnu poruku **„Naziv je obavezan.“** ili **„Morate odabrati korisnika.“** ispod odgovarajućih polja, bez slanja zahteva ka serveru (IA).
* 7.1. Ako backend pri validaciji vrati **HTTP 422** (npr. nevalidan datum ili nekonzistentni podaci), klijent hvata grešku i prikazuje poruku **„Čuvanje nije uspelo. Proveri polja i privilegije.“** u okviru forme, ostavljajući već unete vrednosti netaknutim (IA).
* 6.1. Ako token nije validan ili administrator nema pravo pristupa, backend vraća **HTTP 401** ili **HTTP 403**, klijent prikazuje poruku o grešci i ne preusmerava sa forme, omogućavajući pokušaj ponovnog slanja nakon ispravke stanja (IA).
* 6.2. Ako dođe do mrežne greške (timeout, nedostupan server), klijent prikazuje poruku **„Čuvanje nije uspelo. Proveri polja i privilegije.“** i ostavlja formu otvorenom kako administrator ne bi izgubio unete podatke (IA).

### DS17: Izmena cilja

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je uspešno prijavljen i nalazi se na stranici **„Admin · Goals“**, gde su ciljevi već učitani iz baze. Postoji najmanje jedan cilj koji se može izmeniti.
* **Osnovni scenario SK:**

1. Administrator klikne na dugme **„Izmeni“** pored nekog cilja u listi (ANSO).
2. Klijent otvara formu **„Edit Goal“** (komponenta *AdminGoalForm*) sa prethodno popunjenim poljima — naziv, opis, status, rok i korisnik kojem je cilj dodeljen (IA).
3. Klijent šalje zahtev **GET /api/admin/users** kako bi popunio dropdown za izbor korisnika (SO); lista korisnika se prikazuje (IA).
4. Administrator menja jedno ili više polja cilja:
   * naziv cilja,
   * opis,
   * rok (datum),
   * status cilja,
   * korisnika kome je cilj dodeljen (APUSO).
5. Administrator klikne dugme **„Sačuvaj izmene“** (APSO).
6. Klijent šalje zahtev **PUT /api/goals/{id}** sa izmenjenim podacima i Bearer tokenom u zaglavlju (APSO).
7. Sistem (GoalController@update) validira podatke, ažurira postojeći zapis u tabeli goals i vraća JSON sa novim vrednostima cilja (SO).
8. Klijent zatvara formu i osvežava listu ciljeva, prikazujući ažurirani cilj uz poruku o uspehu (IA).

* **Alternativni scenariji:**  
  4.1. Ako obavezna polja (npr. naziv, korisnik) ostanu prazna, frontend prikazuje validacione greške u okviru forme, bez slanja zahteva backendu (IA).
* 7.1. Ako backend vrati **HTTP 422** (nevalidan datum ili drugi nekorektni podaci), klijent prikazuje serverske greške i ostavlja formu otvorenom sa popunjenim vrednostima (IA).
* 6.1. Ako administrator nema pravo pristupa (npr. token istekao, nevalidan), backend vraća **HTTP 401/403**, a klijent prikazuje poruku i ne zatvara formu (IA).
* 6.2. Ako dođe do prekida konekcije, frontend prikazuje upozorenje **„Izmena nije uspela. Pokušajte ponovo.“** bez brisanja unetih podataka (IA).

### DS18: Brisanje cilja

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je uspešno prijavljen i nalazi se na stranici **„Admin · Goals“**, gde je prikazana lista ciljeva iz baze. Postoji cilj koji je moguće obrisati.
* **Osnovni scenario SK:**

1. Administrator klikne na dugme **„Obriši“** pored cilja koji želi da ukloni (ANSO).
2. Klijent prikazuje potvrdu (modal) sa pitanjem: „Da li ste sigurni da želite da obrišete ovaj cilj?“ (IA).
3. Administrator potvrđuje brisanje klikom na **„Yes, delete“** (APSO).
4. Klijent šalje zahtev **DELETE /api/goals/{id}** sa validnim Bearer tokenom (APSO).
5. Sistem (GoalController@destroy) proverava postojanje cilja, briše odgovarajući zapis iz tabele goals i vraća JSON poruku o uspehu (SO).
6. Klijent uklanja cilj iz liste na ekranu i prikazuje poruku **„Cilj uspešno obrisan.“** (IA).

* **Alternativni scenariji:**   
  3.1. Ako administrator klikne **„Cancel“**, modal se zatvara i ništa se ne menja na ekranu (IA).
* 4.1. Ako backend vrati **HTTP 404** (cilj ne postoji), klijent prikazuje upozorenje **„Cilj nije pronađen.“** (IA).
* 4.2. Ako backend vrati **HTTP 403** (nema permisije), klijent prikazuje **„Nemate dozvolu za ovu akciju.“** (IA).
* 4.3. Ako se dogodi greška na serveru (HTTP 500), prikazuje se poruka **„Brisanje neuspešno. Pokušajte kasnije.“** (IA).
* 5.1. Ako dođe do problema u mreži, klijent prikazuje **„Network error“** i ne uklanja cilj sa liste (IA).

### DS19: Pregled svih korisnika (admin)

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je uspešno prijavljen i nalazi se u Admin panelu. Postoje korisnici u bazi koje je moguće prikazati.
* **Osnovni scenario SK:**

1. Administrator klikne stavku menija **„Users“** (ANSO).
2. Klijent šalje zahtev **GET /api/users** sa validnim Bearer tokenom (APSO).
3. Sistem (AdminController@index) dohvaća listu svih korisnika iz baze i vraća je u JSON formatu (SO).
4. Klijent prikazuje tabelu sa kolonama: **ID, Ime, Email, Uloga, Nivo fitnesa, Datum kreiranja, Akcije** (IA).
5. Administrator upisuje tekst u polje **„Search by name…“** da filtrira korisnike po imenu i prezimenu (APUSO).
6. Administrator iz padajuće liste bira filtriranje po ulozi (npr. *member* / *admin*) — tabela se odmah ažurira (APUSO).
7. Administrator menja stranicu paginacije (Prev/Next/1,2,…) i tabela prikazuje po 5 korisnika po strani (APUSO).

* **Alternativni scenariji:**  
  3.1 Ako u bazi nema korisnika, tabela prikazuje poruku **„No users found.“** (IA).
* 2.1 Ako server vrati **HTTP 403**, klijent prikazuje **„Nemate dozvolu.“** i vraća na /login (IA).
* 2.2 Ako server vrati **HTTP 500**, prikazuje se poruka **„Greška na serveru. Lista korisnika nije dostupna.“** (IA).
* 4.1 Ako API odgovori previše sporo, prikazuje se loading indikator dok se podaci ne učitaju (IA).

### DS20: Eksport korisnika (TXT/CSV)

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:** Administrator je prijavljen i nalazi se na stranici *„Admin · Users“* sa učitanom listom korisnika.
* **Osnovni scenario SK:**

1. Administrator klikne na dugme **„Eksportuj CSV“** (APSO).
2. Klijentska aplikacija prolazi kroz trenutno učitanu listu korisnika (items) i generiše CSV sadržaj sa kolonama *ID, Ime, Email, Uloga* (SO).
3. Sistem kreira privremeni fajl (*Blob*) i automatski pokreće preuzimanje fajla **users\_export.csv** u browseru (IA).
4. Administrator zatim klikne na dugme **„Eksportuj TXT“** (APSO).
5. Klijentska aplikacija formira tekstualni izveštaj, gde je svaki korisnik prikazan u formatu  
   ID: {id} | Ime: {name} | Email: {email} | Uloga: {role} (SO).
6. Sistem kreira *Blob* i pokreće preuzimanje fajla **users\_export.txt** (IA).

* **Alternativni scenariji:**  
  1.1 Ako je lista korisnika prazna u trenutku klika na **„Eksportuj CSV“** ili **„Eksportuj TXT“**, sistem ne generiše fajl, već prikazuje poruku **„Nema podataka za eksport.“** (IA).

### DS21: Brisanje korisnika (admin)

* **Aktori SK:** Administrator
* **Učesnici SK:** Administrator i sistem
* **Preduslov:**
  + Administrator je prijavljen.
  + Lista korisnika je učitana na stranici **Admin · Users**.
* **Osnovni scenario SK:**

1. **Administrator klikne na dugme „Delete“** pored željenog korisnika u tabeli (APSO).
2. Klijent šalje zahtev **DELETE /api/admin/users/{id}** backendu (APSO).
3. Backend pronalazi korisnika po prosleđenom ID-ju i briše ga iz baze (SO).
4. Backend vraća JSON odgovor o uspešnom brisanju (SO).
5. Klijent uklanja korisnika iz prikazane tabele i prikazuje toast poruku **„Korisnik obrisan.“** (IA).

* **Alternativni scenariji:**

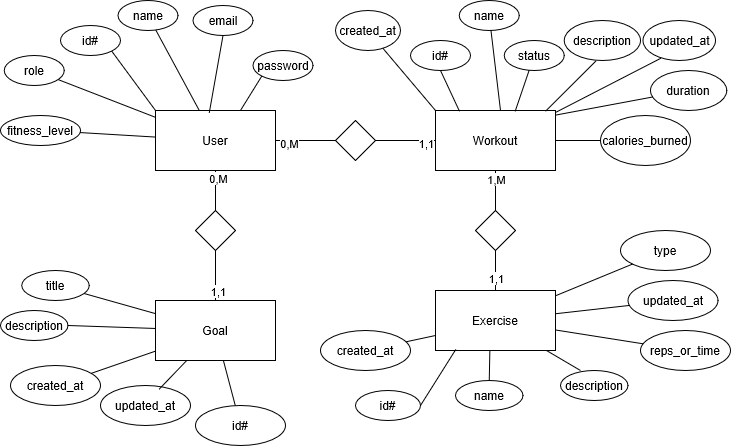
1.1 Ako API vrati **HTTP 403 (Forbidden)**, klijent prikazuje poruku  
**„Nemate dozvolu za ovu akciju.“** (IA).

1.2 Ako API vrati **HTTP 404**, klijent prikazuje poruku  
**„Korisnik nije pronađen.“** (IA).

1.3 Ako API vrati **HTTP 500**, prikazuje se poruka  
**„Greška pri brisanju korisnika.“** (IA).

### Struktura softverskog sistema

**PMOV (Proces modelovanja objekti‐veza)** predstavlja metodološki okvir i skup pravila za izradu konceptualnog modela baze podataka zasnovanog na ER (Entity–Relationship) pristupu, odnosno MOV-u (Model objekti-veza). Dok sam MOV definiše kako prikazati entitete, njihove atribute i međusobne veze grafičkim dijagramom, PMOV se bavi čitavim procesom njegova nastanka – od prikupljanja zahteva i identifikacije ključnih objekata u domenu preko definisanja tipova veza pa sve do iterativne validacije i usklađivanja modela sa krajnjim korisnicima i razvojnim timom. U okviru PMOV-a, prvobitni rad na crtežu entiteta i relacija proširuje se na organizovano vođenje sesija sa stručnjacima iz domena (workshop-ovi), evidentiranje pravila o kardinalnostima i opcionalnostima veza, kao i kontrolu verzija modela, što osigurava doslednost i potpunost konceptualne specifikacije bazirane na stvarnim poslovnim potrebama (Vlajić, 2020)



Na priloženom dijagramu ima 4 modela:

**1. User**

* **Atributi:**  
  id, name, email, password, role, fitness\_level,  
  email\_verified\_at, remember\_token, profile\_picture
* **Veze:**
  + *(implicitno)* ima mnogo treninga → hasMany(Workout::class)
* **Objašnjenje:**  
  Model korisnika čuva osnovne informacije, ulogu i nivo fizičke spremnosti korisnika.  
  Autentikacija se vrši pomoću Laravel Sanctum tokena.

**2. Workout**

* **Atributi:**  
  id, user\_id, name, description, duration, calories\_burned, status,  
  created\_at, updated\_at
* **Veze:**
  + Pripada korisniku → belongsTo(User::class)
  + Ima više vežbi → hasMany(Exercise::class)
* **Objašnjenje:**  
  Trening predstavlja sesiju koju korisnik kreira. Svaki trening je povezan sa korisnikom koji ga je kreirao i može sadržati više vežbi.

**3. Exercise**

* **Atributi:**  
  id, workout\_id, name, description, reps\_or\_time, type,  
  created\_at, updated\_at
* **Veze:**
  + Pripada treningu → belongsTo(Workout::class)
* **Objašnjenje:**  
  Svaka vežba je povezana sa tačno jednim treningom.  
  Polje reps\_or\_time označava broj ponavljanja ili trajanje, dok type definiše tip vežbe (npr. snaga, kardio itd.).

**4. Goal**

* **Atributi:**  
  id, user\_id, title, description, created\_at, updated\_at
* **Veze:**
  + *P*ripada korisniku → belongsTo(User::class)
* **Objašnjenje:**  
  Model ciljeva omogućava korisnicima da postave svoje fitnes ciljeve.  
  Svaki cilj je povezan sa korisnikom koji ga je kreirao.

Glavne prednosti primene PMOV-a ogledaјu se u sledećem (Vlajić, 2020):

* **Jasnoća i zajednički jezik** - Budući da PMOV podrazumeva sistematsko uključenje svih zainteresovanih strana (analitičara, krajnjih korisnika, projektnih menadžera), dobija se jedinstven, strogo definisan konceptualni model. Diagrami objekti-veza postaju ne samo tehnička dokumentacija već i sredstvo komunikacije između poslovnih i IT timova, čime se drastično smanjuju nesporazumi prilikom daljeg razvoja sistema.
* **Rana detekcija nedoslednosti i rupa u zahtevima** - Strukturalni i semantički aspekti modela (npr. cardinality constraints, identifikacija slabih entiteta) razmatraju se još u fazi dizajna, što omogućava otkrivanje nedorečenih zahteva ili konflikata zahteva pre pisanja i testiranja koda. Time se znatno smanjuju troškovi kasnijih izmena na implementacionom nivou.
* **Apstrakcija od konkretne tehnologije** - PMOV naglašava konceptualnu nezavisnost – model je validan bez obzira na to da li će baza podataka biti realizovana kao relacijska, NoSQL ili graf-baza. Ovaj sloj apstrakcije olakšava migracije između tehnologija i izbor najboljeg rešenja za performanse ili skalabilnost, bez potrebe za povratkom na početak analize.
* **Osnova za automatsko generisanje** - Kada je konceptualni model kompletno definisan prema pravilima PMOV-a, on može poslužiti kao ulaz za alate koji automatski generišu deo DDL skripti, dokumentaciju ili čak skeletne klase u kodu (ORM entiteta). To ubrzava implementaciju schema i održavanje konzistentnosti između modela i produkcijske baze.
* **Podrška iterativnom razvoju i agilnim metodama** - PMOV je kompatibilan s iterativnim razvojem: u svakoj iteraciji model se može dopuniti ili izmeniti na osnovu novih zahteva, uz praćenje promena i verifikaciju sa timom. To omogućava agilno prilagođavanje promenama u poslovnom okruženju, a da se pritom ne ugrozi integritet ni konzistentnost celokupnog modela.

Primena PMOV-a u praksi dovodi do robusnijeg, lakše održivog i poslovno usklađenog konceptualnog dizajna baza podataka, koji služi kao pouzdana temeljna vrednost tokom čitavog životnog ciklusa softverskog sistema.

## Faza projektovanja

Tok **projektovanja softverskog sistema** započinje detaljnom analizom i definisanjem korisničkog interfejsa kroz ekranske forme koje predstavljaju direktnu tačku interakcije sa krajnjim korisnikom. U okviru ove faze modeliraju se svi ključni ekrani: od forme za prijavu i registraciju, preko početnih strana za zaposlene, HR radnike i administratore, pa do kompleksnijih prikaza kao što su liste zahteva za odsustvo, ocene zaposlenih ili globalna mapa korporativnih lokacija. Svaka forma se osmišljava s jasno definisanim poljima, dugmadima i tokovima navigacije, pri čemu se vodi računa o intuitivnosti, doslednosti u boji i rasporedu elemenata, kao i o odgovarajućim porukama za greške i potvrde akcija (Vlajić, 2020).

Nakon dizajniranja vidljivog sloja, prelazi se na modelovanje aplikacione logike na serverskoj strani. U Laravel delu, definisani su kontroleri za svaki domen funkcionalnosti (korisnici, zahtevi za odsustvo, ocene i administrativni metrički prikazi), dok je autentifikacija implementirana pomoću Sanctum paketa. Datoteke za resurse zadužene su za transformaciju modela u JSON strukture prilagođene front-endu, a API rute detaljno mapiraju sve CRUD operacije i specifične akcije kao što su menjanje statusa zahteva ili izvoz izveštaja u PDF. Dodatno, database seeder-i omogućavaju inicijalno popunjavanje sistema validnim podacima, što ubrzava razvoj, testiranje i demonstraciju aplikacije (Vlajić, 2020).

Na klijentskoj strani logiku preuzima skup React Hooks i komponenti. Osnovni React Hooks (useState, useEffect) koriste se za upravljanje lokalnim stanjem i asinhronim pozivima, dok custom Hooks kao useRandomFakeUsers i useRandomQuote enkapsuliraju specifične API pozive i rukovanje rezultatima. Komponente su organizovane hijerarhijski po ulogama, a zajednički delovi interfejsa smešteni su u odvojene module poput SideMenu, Breadcrumbs, ProtectedRoute i ProtectedLayout. Rutiranje unutar jednog-page aplikacije vrši se pomoću BrowserRouter-a koji omogućava navigaciju bez ponovnog učitavanja stranice i proveru autentikacije pre renderovanja pojedinačnih ruta (Vlajić, 2020).

Pritom, struktura celokupnog projekta je takva da je back-end (Laravel) jasan segment koji obuhvata foldere za kontrolere, modele, resurse, migracije i seeder-e, dok je front-end (React) organizovan u direktorijume za komponente, hooks i assets. Modeli i migracije definišu relacioni model podataka i šemu baze, a unutar faze projektovanja skladišta pažnja se posvećuje optimizaciji strukture tabela, postavljanju ključeva i indeksiranju. Na taj način, ceo sistem u fazi projektovanja dobija koherentnu arhitekturu u kojoj su korisnički interfejs, poslovna logika, organizacija fajlova i baza podataka usklađeni radi lakšeg razvoja, održavanja i budućih proširenja.

### Projektovanje korisničkog interfejsa (ekstranskih formi)

### Prijava

Stranica za prijavu omogućava postojećim korisnicima da pristupe aplikaciji unosom email adrese i lozinke. Forma je jasno podeljena na dva polja – Email i Lozinka – uz vizuelno naglašeno dugme **Login**. Prilikom unosa podataka korisniku se prikazuje povratna informacija o greškama (npr. neispravna lozinka ili nepostojeći nalog). Dugme **Reset** briše trenutne vrednosti iz polja i omogućava brz ponovni unos.  
Dizajn koristi tamnu temu aplikacije, sa karakterističnim narandžastim akcijskim dugmadima čime se dosledno zadržava vizuelni identitet celog sistema.

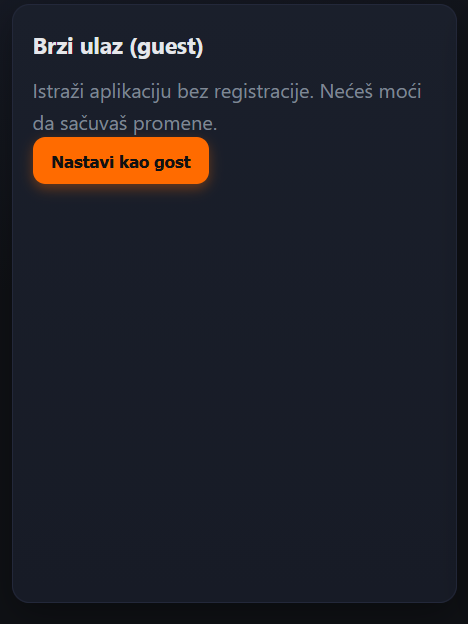
### Registracija

Registraciona forma omogućava novim korisnicima da kreiraju nalog. Polja koja se popunjavaju uključuju:

* Ime i prezime
* Email
* Lozinka
* Uloga (admin ili member)
* Fitness nivo (beginner, intermediate, expert)

Nakon popunjavanja podataka korisnik klikom na **Registruj se** kreira nalog i automatski se uloguje u aplikaciju. Tu je i dugme **Reset** koje vraća formu u početno stanje.  
Stranica je pregledno dizajnirana, a validacija onemogućava registraciju bez popunjenih obaveznih polja.

### Ulazak kao gost



Aplikacija nudi mogućnost brzog ulaska bez registracije kroz opciju Nastavi kao gost. Gost korisnik može pregledati sadržaj (workouts, exercises), ali ne može trajno čuvati izmene.

Ova sekcija jasno obaveštava korisnika o ograničenjima guest profila. Kartica je vizuelno u skladu sa ostatkom sistema i služi kao najbrži način da se demonstrira funkcionalnost.

### Početna stranica - Modal

Nakon prijave korisniku se može prikazati modalni prozor sa informacijom o održavanju sistema.  
Modal sadrži:

* istaknut naslov **PAŽNJA!**
* informativnu poruku o trenutnim radovima i mogućim ograničenjima
* dugme **OK** za zatvaranje modalnog prozora

Ovaj modal je centralno pozicioniran, ima tamnu temu i koristi narandžasto dugme u skladu sa vizuelnim identitetom aplikacije.

### Početna stranica

Početna stranica predstavlja uvod u aplikaciju „Rebel Fitness“. Na levoj strani nalazi se kratko objašnjenje o svrsi aplikacije i dve primarne akcije:

* **Moji treninzi** (za brzi ulazak u lične workout-e)
* **Dodaj workout** (za kreiranje novog treninga)

Desna strana prikazuje veliki prikaz logotipa brenda.  
Ispod uvodnog dela nalazi se sekcija „Šta aplikacija radi“, gde su u karticama objašnjene ključne funkcionalnosti: profili i uloge, planiranje treninga, fitnes dnevnik i drugo.  
Stranica služi kao centralna tačka navigacije kroz aplikaciju.

### Treninzi

Stranica „Workouts“ prikazuje sve treninge u obliku kartica. Svaka kartica sadrži:

* naziv treninga
* kratak opis
* trajanje u minutima
* potrošene kalorije
* status (pending/completed)
* dugme **Detalji** koje vodi na pojedinačni prikaz treninga

Na vrhu stranice nalazi se dugme **+ Novi trening** za kreiranje novog workout-a, kao i dugme **Osveži** koje ponovo učitava podatke sa servera.  
Dizajn je pregledan, moderan i koristi responsivne kartice koje se prilagođavaju širini ekrana.

### Vežbe

Ova sekcija prikazuje tabelarni prikaz svih vežbi uz napredne opcije filtriranja i sortiranja:

* „Samo moji“ – prikazuje vežbe kreirane od strane trenutnog korisnika
* Filter po tipu (svi tipovi, cardio, strength, flexibility…)
* Pretraga po nazivu i opisu
* Sortiranje po više kriterijuma (npr. naziv)
* Odabir broja prikaza po stranici

Tabela sadrži kolone ID, Naziv, Opis, Ponavljanja/Vreme, Tip, Workout i Akcije. Akcije uključuju:

* **Uredi** – otvara formu za izmenu vežbe
* **Obriši** – uklanja vežbu

Interfejs je optimizovan za administraciju i pregled većeg broja elemenata.

### Ciljevi

Stranica za administraciju ciljeva omogućava pregled, uređivanje i brisanje korisničkih fitness ciljeva.  
Tabela sadrži:

* ID
* Naziv cilja
* Opis
* Korisnik (ID korisnika)
* Rok
* Status (pending/completed)
* Akcije

Desno se nalaze dugmad:

* **Novi cilj** – otvara formu za kreiranje novog cilja
* **Osveži** – ponovo učitava podatke

Akcije u tabeli sadrže:

* **Uredi** – za izmenu cilja
* **Obriši** – za uklanjanje cilja

Stranica je deo admin funkcionalnosti i omogućava detaljnu kontrolu fitness ciljeva članova.

### Korisnici – ADMIN dashboard

Ova sekcija je dostupna samo administratorima i prikazuje sve registrovane korisnike u tabelarnom prikazu.  
Tabela uključuje:

* ID
* Ime
* Email
* Uloga (admin/member)
* Akcije

Dostupne akcije:

* **Detalji** – otvara modal sa kompletnim informacijama o korisniku
* **Obriši** – trajno uklanja korisnika iz sistema

Na vrhu stranice se nalaze dodatne administrativne funkcije:

* **Osveži** – osvežava tabelu
* **Eksportuj CSV** – izvoz podataka u CSV format
* **Eksportuj TXT** – izvoz podataka u TXT format

Ovo je centralno mesto za upravljanje korisnicima.

### Projektovanje aplikacione logike

U **fazi projektovanja aplikacione logike** uspostavlja se način na koji će klijentski zahtevi biti obrađivani i kako će se podaci kretati kroz sistem sve do sloja za skladištenje. U Laravel delu aplikacije, **kontroleri** igraju ulogu koordinatora: oni primaju HTTP zahteve, pozivaju odgovarajuće servise ili modele i vraćaju rezultat klijentu u vidu JSON odgovora ili preusmeravanja na drugi resurs. Kontroleri su organizovani po domenima („UserController“, „LeaveRequestController“ itd.), što olakšava održavanje i testiranje, jer se za svaki entitet ili poslovni slučaj korisćenja definiše poseban set metoda za osnovne CRUD operacije, kao i specifične akcije poput odobravanja zahteva ili generisanja PDF izveštaja.

Autentifikacija korisnika realizovana je preko Laravel Sanctum paketa: ruta za registraciju („register“) vrši validaciju podataka, kreira korisnika u bazi i odmah generiše pristupni token, dok ruta za prijavu („login“) proverava kredencijale i obnavlja ili izdaje novi token. Zaštićene rute grupisane su middleware-om „auth:sanctum“, što onemogućava neautorizovan pristup osetljivim podacima poput liste zahteva za odmor ili izveštaja o performansama. Logout akcija briše trenutni token i time efektivno prekida sesiju, ostavljajući čitav postupak bezbednim i jednostavnim za klijenta i server.

Kako bi se model podataka jednostavno transformisao u JSON strukturu pogodnu za front-end, koriste se **Laravel resursi** (Resources). Svaki model poput LeaveRequest ili PerformanceReview ima pripadajući resurs koji eksponira onoliko atributa koliko je potrebno — na primer, prikazuje relacije korisnika ili odeljenja, ali sakriva osetljive informacije poput hashovane lozinke. Time se zadržava jasna granica između interne strukture baze i spoljnog API-ja, a klijent dobija uvek uniformne podatke bez dodatnog ručnog mapiranja.

Na nivou REST API-ja, svi endpoint-i su definisani u **api.php**, gde svaka ruta eksplicitno mapira HTTP metodu (GET, POST, PUT, PATCH, DELETE) na određenu akciju kontrolera. Postavljanjem resursa kroz Route::apiResource i ručno dodavanjem specijalnih ruta, dobija se kompletna i dosledna struktura za CRUD operacije, a middleware čuva konzistentnost i bezbednost. Ovako definisan API omogućava front-endu da jednostavno „razgovara“ sa serverom putem standardnih HTTP zahteva i REST paradigme.

U React delu aplikacije, ključni gradivni elementi su **Hooks** i **komponente**. Hooks kao što su useState i useEffect omogućavaju upravljanje stanjem i efektima unutar funkcionalnih komponenti, dok se prilagođeni (custom) Hooks, poput useWeather, koriste za enkapsulaciju specifične logike ponovnog korišćenja—na primer, pozivanje eksternih API-ja, rukovanje greškama ili automatsko osvežavanje podataka na određenim intervalima. Time se React kod čini čistijim i modularnijim, jer se logika odvaja od vizuelne prezentacije.

**React komponente** su organizovane hijerarhijski prema korisničkim ulogama i zajedničkim delovima interfejsa (SideMenu, ProtectedRoute, Breadcrumbs). Centralno mesto drži App.js, unutar kojeg se, korisćenjem **BrowserRouter-a**, definišu rute do pojedinačnih stranica—svaka ruta odgovara nekoj React komponenti. BrowserRouter omogućava da se navigacija odvija bez osvežavanja cele stranice, čime se stvara osećaj „single-page“ aplikacije. Zaštićene rute proveravaju pristupni token i ulogu korisnika pre renderovanja željene komponente, obezbeđujući da samo ovlašćeni korisnici mogu da dođu do odgovarajućih delova aplikacije.

Konačno, **integracija Laravel back-enda i React front-enda** odvija se preko REST API-ja: React aplikacija, pokrenuta komandom npm start, komunicira sa Laravel serverom (npr. na http://localhost:8000/api) slanjem HTTP zahteva kroz fetch ili axios. Laravel obrađuje zahteve, učitava ili čuva podatke u relacione tabele i vraća JSON, dok React instantno ažurira prikaz na osnovu pristiglih podataka. Ovako implementiran MVC obrazac postaje raslojen: React komponente predstavljaju View, personalizovani Hooks i servisi služe kao Controller u front-endu, a Laravel modeli i migracije čine Model i skladište podataka, što rezultira jasnom podelom odgovornosti i lakšim održavanjem čitavog sistema.

### Laravel kontroleri (Controllers)

**Laravel kontroleri** predstavljaju jezgro upravljanja HTTP zahteva i odgovora unutar MVC (Model–View–Controller) arhitekture Laravel framework-a. Svaki kontroler je PHP klasa koja grupiše srodne akcije (metode) koje obrađuju određene rute aplikacije. Kada korisnik pošalje zahtev (npr. otvaranje stranice, slanje formulara), Laravel rutiranje određuje koji kontroler i koju njegovu metodu treba pozvati. Kontroleri organizuju poslovnu logiku, učitavaju podatke iz modela, obrađuju validaciju i na kraju vraćaju odgovarajući view ili JSON odgovor (Laravel, 2025).

Osnovni kontroler u Laravel-u, često nazvan jednostavno Controller.php, služi kao roditeljska klasa svim drugim kontrolerima. U njega umeštamo zajedničke osobine ili trait-ove (npr. middlewares) koje će naslediti specifični kontroleri. Time se izbegava dupliranje koda i olakšava održavanje zajedničkih funkcionalnosti, poput autorizacije ili globalnih validacija, na jednom mestu (Laravel, 2025).

**AuthController**

Obezbeđuje registraciju, prijavu i odjavu korisnika: register validira i kreira korisnika sa role i fitness\_level, izdaje Sanctum token; login proverava akreditive i vraća novi token; logout briše sve aktivne tokene prijavljenog korisnika.

<?php

namespace *App\Http\Controllers*;

use *Illuminate\Http\Request*;

use *App\Models\User*;

use *Illuminate\Support\Facades\Hash*;

use *Illuminate\Support\Facades\Validator*;

use *Illuminate\Support\Str*;

class AuthController extends *Controller*

{

    public function register(*Request* $request)

    {

        // Validacija unosa

        $validator = *Validator*::make($request->all(), [

            'name'          => 'required|string|max:255',

            'email'         => 'required|string|email|unique:users',

            'password'      => 'required|string|min:8',

            'role'          => 'required|in:admin,member,guest',

            'fitness\_level' => 'required|in:beginner,intermediate,expert',

        ]);

        if ($validator->fails()) {

            return response()->json(['errors' => $validator->errors()], 400);

        }

        // Kreiranje korisnika u bazi podataka

        $user = *User*::create([

            'name'              => $request->name,

            'email'             => $request->email,

            'password'          => *Hash*::make($request->password),

            'role'              => $request->role,

            'fitness\_level'     => $request->fitness\_level,

            'email\_verified\_at' => now(),

            'remember\_token'    => *Str*::random(60),

        ]);

        // Provera da li je metoda createToken dostupna

        if (!method\_exists($user, 'createToken')) {

            return response()->json(['error' => 'Sanctum nije ispravno konfigurisan.'], 500);

        }

        // Kreiraj API token

        $token = $user->createToken('auth\_token')->plainTextToken;

        return response()->json([

            'message' => 'User registered successfully',

            'token'   => $token,

            'user'    => $user,

        ], 201);

    }

    public function login(*Request* $request)

    {

        $request->validate([

            'email'    => 'required|string|email',

            'password' => 'required|string',

        ]);

        $user = *User*::where('email', $request->email)->first();

        if (!$user || !*Hash*::check($request->password, $user->password)) {

            return response()->json(['message' => 'Invalid credentials'], 401);

        }

        $token = $user->createToken('auth\_token')->plainTextToken;

        return response()->json([

            'message' => 'Login successful',

            'token'   => $token,

            'user'    => $user,

        ], 200);

    }

    public function logout()

    {

*\Auth*::user()->tokens()->delete();

        return response()->json(['message' => 'Logged out'], 200);

    }

    // Prijava kao gost korisnik

    public function loginAsGuest(*Request* $request)

    {

        // Predefinisani podaci za gosta

        $name    = $request->input('name', 'Guest ' . *Str*::upper(*Str*::random(4)));

        $fitness = $request->input('fitness\_level', 'beginner');

        $email   = 'guest\_' . *Str*::uuid() . '@example.test';

        $user = *User*::create([

            'name'              => $name,

            'email'             => $email,

            'password'          => *Hash*::make(*Str*::random(32)),

            'role'              => 'guest',

            'fitness\_level'     => $fitness,

            'email\_verified\_at' => now(),

            'remember\_token'    => *Str*::random(60),

        ]);

        $token = $user->createToken('guest\_token')->plainTextToken;

        return response()->json([

            'message' => 'Guest account created',

            'token'   => $token,

            'user'    => [

                'id'             => $user->id,

                'name'           => $user->name,

                'role'           => $user->role,

                'fitness\_level'  => $user->fitness\_level,

                'email\_verified' => (*bool*) $user->email\_verified\_at,

            ],

        ], 201);

    }

}

**WorkoutController**

Upravlja treninzima: index i show vraćaju listu/single trening, store kreira trening za ulogovanog korisnika, update menja podatke (dozvoljeno vlasniku ili adminu), destroy briše trening, startWorkout postavlja status na „started“, getUserWorkouts vraća treninge prijavljenog korisnika, deleteUserWorkouts briše sve njegove treninge.

<?php

namespace *App\Http\Controllers*;

use *Illuminate\Http\Request*;

use *Illuminate\Support\Facades\Auth*;

use *App\Models\Workout*;

use *Illuminate\Database\Eloquent\ModelNotFoundException*;

use *App\Http\Resources\WorkoutResource*;

class WorkoutController extends *Controller*

{

    //  Dohvatanje svih treninga

    public function index()

    {

        $user = *Auth*::user();

        if (!$user) {

            return response()->json(['message' => 'Unauthorized'], 401);

        }

        // Pravila pristupa

        if ($user->role === 'admin') {

            $workouts = *Workout*::latest()->get(); // Admin vidi sve

        } elseif ($user->role === 'member') {

            $workouts = *Workout*::where('user\_id', $user->id)->latest()->get(); // Member vidi samo svoje

        } else { // Guest

            $workouts = *Workout*::latest()->get(); // Guest vidi sve (read-only)

        }

        return response()->json([

            'status' => 'success',

            'data' => $workouts

        ], 200);

    }

    //  Dohvatanje treninga po ID-u

    public function show($id)

    {

        try {

            $workout = *Workout*::findOrFail($id);

            $user = *Auth*::user();

            if (!$user) {

                return response()->json(['message' => 'Unauthorized'], 401);

            }

            // Pravila pristupa

            if ($user->role === 'admin') {

                // Admin vidi sve

            } elseif ($user->role === 'member' && $workout->user\_id !== $user->id) {

                return response()->json(['message' => 'Forbidden'], 403);

            } elseif ($user->role === 'guest') {

                // Gost vidi sve (read-only)

            }

            return response()->json([

                'status' => 'success',

                'data' => $workout

            ], 200);

        } catch (*ModelNotFoundException* $e) {

            return response()->json([

                'status' => 'error',

                'message' => 'Workout not found'

            ], 404);

        }

    }

    //  Kreiranje novog treninga (member/admin)

    public function store(*Request* $request)

    {

        $validated = $request->validate([

            'name' => 'required|string|max:255',

            'description' => 'nullable|string',

            'duration' => 'required|integer|min:1',

            'calories\_burned' => 'required|integer|min:1',

        ]);

        $validated['user\_id'] = *Auth*::id();

        $workout = *Workout*::create($validated);

        return response()->json([

            'status' => 'success',

            'data' => $workout

        ], 201);

    }

    //  Ažuriranje treninga (vlasnik ili admin)

    public function update(*Request* $request, $id)

    {

        $user = *Auth*::user();

        // 404 ako ne postoji trening

        $workout = *Workout*::find($id);

        if (!$workout) {

            return response()->json(['message' => 'Workout not found'], 404);

        }

        // Dozvola: vlasnik ili admin

        if ($user->role !== 'admin' && $workout->user\_id !== $user->id) {

            return response()->json(['message' => 'Forbidden'], 403);

        }

        // Delimično ažuriranje

        $validated = $request->validate([

            'name'            => 'sometimes|required|string|max:255',

            'description'     => 'sometimes|nullable|string',

            'duration'        => 'sometimes|integer|min:1',

            'calories\_burned' => 'sometimes|integer|min:1',

            'status'          => 'sometimes|string|in:pending,started,completed',

        ]);

        $workout->update($validated);

        return response()->json([

            'status'  => 'success',

            'message' => 'Workout successfully modified',

            'id'      => $workout->id,

        ], 200);

    }

    //  Brisanje treninga po ID-u

    public function destroy($id)

    {

        try {

            $workout = *Workout*::findOrFail($id);

            $user = *Auth*::user();

            if ($user->role !== 'admin' && $workout->user\_id !== $user->id) {

                return response()->json(['message' => 'Forbidden'], 403);

            }

            $workout->delete();

            return response()->json([

                'status' => 'success',

                'message' => 'Workout deleted'

            ], 200);

        } catch (*ModelNotFoundException* $e) {

            return response()->json([

                'status' => 'error',

                'message' => 'Workout not found'

            ], 404);

        }

    }

    //  Pokretanje treninga

    public function startWorkout($id)

    {

        $workout = *Workout*::find($id);

        if (!$workout) {

            return response()->json(['message' => 'Workout not found'], 404);

        }

        $workout->status = 'started';

        $workout->save();

        return response()->json([

            'message' => 'Workout started',

            'data' => $workout

        ], 200);

    }

    //  Dohvatanje treninga određenog korisnika

    public function getUserWorkouts()

    {

        $userId = *Auth*::id();

        if (!$userId) {

            return response()->json(['message' => 'Unauthorized'], 401);

        }

        $workouts = *Workout*::where('user\_id', $userId)->get();

        if ($workouts->isEmpty()) {

            return response()->json(['message' => 'No workouts found for this user'], 404);

        }

        return response()->json([

            'message' => 'Workouts retrieved successfully',

            'data' => $workouts

        ], 200);

    }

    // Brisanje svih treninga korisnika

    public function deleteUserWorkouts()

    {

        $userId = *Auth*::id();

        if (!$userId) {

            return response()->json(['message' => 'Unauthorized'], 401);

        }

        $deletedCount = *Workout*::where('user\_id', $userId)->delete();

        if ($deletedCount == 0) {

            return response()->json(['message' => 'No workouts found for this user'], 404);

        }

        return response()->json(['message' => 'All workouts for the user have been deleted'], 200);

    }

}

**ExerciseController**

CRUD nad vežbama: index vraća sve vežbe, store kreira vežbu uz obavezni workout\_id (mora postojati u workouts), show prikazuje vežbu, update ažurira prosleđena polja, destroy briše vežbu.

<?php

namespace *App\Http\Controllers*;

use *Illuminate\Http\Request*;

use *Illuminate\Support\Facades\Auth*;

use *App\Models\Exercise*;

class ExerciseController extends *Controller*

{

    // Vraća vežbe sa paginacijom, filtriranjem i sortiranjem

    public function index(*Request* $request)

    {

        $query = *Exercise*::with(['workout:id,user\_id,name']);

        // Filtriranje po tipu (type=cardio/strength/flexibility)

        if ($request->filled('type')) {

            $query->where('type', $request->type);

        }

        // Pretraga po nazivu/opisu

        if ($request->filled('search')) {

            $text = mb\_strtolower($request->search, 'UTF-8');

            $query->where(function ($q) use ($text) {

                $q->whereRaw('LOWER(name) LIKE ?', ["%{$text}%"])

                  ->orWhereRaw('LOWER(description) LIKE ?', ["%{$text}%"]);

            });

        }

        // Samo moje vežbe (mine=1) – filtrira po vlasniku workout-a

        if ($request->boolean('mine')) {

            $userId = *Auth*::id();

            if ($userId) {

                $query->whereHas('workout', function ($q) use ($userId) {

                    $q->where('user\_id', $userId);

                });

            }

        }

        // Sortiranje: sort=name|type|reps\_or\_time|id, dir=asc|desc

        $sort = $request->get('sort', 'name');

        $dir  = $request->get('dir', 'asc');

        $allowedSorts = ['id', 'name', 'type', 'reps\_or\_time'];

        if (!in\_array($sort, $allowedSorts, true)) {

            $sort = 'name';

        }

        $query->orderBy($sort, $dir === 'desc' ? 'desc' : 'asc');

        // Paginacija

        $perPage = (*int*) $request->get('per\_page', 10);

        if ($perPage <= 0) {

            $perPage = 10;

        }

        $paginator = $query->paginate($perPage)->appends($request->query());

        return response()->json($paginator, 200);

    }

    // Kreira novu vežbu

    public function store(*Request* $request)

    {

        $validatedData = $request->validate([

            'name'         => 'required|string|max:255',

            'description'  => 'nullable|string',

            'reps\_or\_time' => 'required|integer',

            'type'         => 'required|string|max:255',

            'workout\_id'   => 'required|exists:workouts,id',

        ]);

        $exercise = *Exercise*::create($validatedData);

        $exercise->load(['workout:id,user\_id,name']);

        return response()->json([

            'message'  => 'Exercise successfully created.',

            'exercise' => $exercise,

        ], 201);

    }

    // Vraća jednu vežbu

    public function show($id)

    {

        $exercise = *Exercise*::with(['workout:id,user\_id,name'])->find($id);

        if (!$exercise) {

            return response()->json(['message' => 'Exercise not found.'], 404);

        }

        return response()->json($exercise, 200);

    }

    // Ažurira postojeću vežbu

    public function update(*Request* $request, $id)

    {

        $exercise = *Exercise*::find($id);

        if (!$exercise) {

            return response()->json(['message' => 'Exercise not found.'], 404);

        }

        $validatedData = $request->validate([

            'name'         => 'sometimes|required|string|max:255',

            'description'  => 'nullable|string',

            'reps\_or\_time' => 'sometimes|required|integer',

            'type'         => 'sometimes|required|string|max:255',

            'workout\_id'   => 'sometimes|required|exists:workouts,id',

        ]);

        $exercise->update($validatedData);

        $exercise->load(['workout:id,user\_id,name']);

        return response()->json([

            'message'  => 'Exercise successfully updated.',

            'exercise' => $exercise,

        ], 200);

    }

    // Briše vežbu

    public function destroy($id)

    {

        $exercise = *Exercise*::find($id);

        if (!$exercise) {

            return response()->json(['message' => 'Exercise not found.'], 404);

        }

        $exercise->delete();

        return response()->json([

            'message' => 'Exercise successfully deleted.',

        ], 200);

    }

}

**GoalController**

Rad sa ciljevima: store kreira cilj za datog korisnika (user\_id, title, description, target\_date), index vraća sve ciljeve, show prikazuje cilj, update menja polja (title, description, opcioni deadline, status), destroy briše cilj.

<?php

namespace *App\Http\Controllers*;

use *Illuminate\Http\Request*;

use *App\Models\Goal*;

class GoalController extends *Controller*

{

    // Kreira novi cilj

    public function store(*Request* $request)

    {

        $validated = $request->validate([

            'user\_id'     => 'required|exists:users,id',

            'title'       => 'required|string|max:255',

            'description' => 'nullable|string',

            'target\_date' => 'required|date',

            'status'      => 'required|string',

        ]);

        $goal = *Goal*::create($validated);

        return response()->json(['status' => 'success', 'data' => $goal], 201);

    }

    // Vraća sve ciljeve

    public function index()

    {

        return response()->json(*Goal*::all(), 200);

    }

    // Vraća jedan cilj po ID-ju

    public function show($id)

    {

        $goal = *Goal*::find($id);

        if (!$goal) {

            return response()->json(['message' => 'Goal not found'], 404);

        }

        return response()->json($goal);

    }

    // Ažurira cilj po ID-ju

    public function update(*Request* $request, $id)

    {

        $goal = *Goal*::find($id);

        if (!$goal) {

            return response()->json(['message' => 'Goal not found'], 404);

        }

        $validated = $request->validate([

            'title'       => 'sometimes|required|string|max:255',

            'description' => 'nullable|string',

            'target\_date' => 'nullable|date',

            'status'      => 'nullable|string',

            'user\_id'     => 'sometimes|integer|exists:users,id',

        ]);

        $goal->fill($validated);

        $goal->save();

        return response()->json($goal);

    }

    // Briše cilj po ID-ju

    public function destroy($id)

    {

        $goal = *Goal*::find($id);

        if (!$goal) {

            return response()->json(['message' => 'Goal not found'], 404);

        }

        $goal->delete();

        return response()->json(['message' => 'Goal deleted'], 200);

    }

}

**AdminController**

Administrativne akcije: dashboard vraća JSON poruku dobrodošlice, listUsers vraća sve korisnike, deleteUser briše korisnika po ID-ju (404 ako ne postoji).

<?php

namespace *App\Http\Controllers*;

use *Illuminate\Http\Request*;

use *App\Models\User*;

class AdminController extends *Controller*

{

    /\*\*

     \* Prikazuje admin dashboard.

     \*

     \* @return *\Illuminate\Http\JsonResponse*

     \*/

    public function dashboard()

    {

        // Vraća JSON odgovor sa porukom dobrodošlice

        return response()->json([

            'message' => 'Dobrodošli na admin dashboard!',

        ]);

    }

   // Dohvata i vraća listu svih korisnika u JSON formatu

    public function listUsers()

    {

        $users = *User*::all();

        return response()->json($users);

    }

    // Briše korisnika po ID-ju

    public function deleteUser($id)

    {

        $user = *User*::find($id);

        // Proverava da li korisnik postoji

        if (!$user) {

            return response()->json(['message' => 'Korisnik nije pronađen'], 404);

        }

        // Briše korisnika

        $user->delete();

        // Vraća poruku o uspešnom brisanju

        return response()->json(['message' => 'Korisnik uspešno obrisan']);

    }

}

**WeatherController**

Integracija sa javnim servisom: getWeather($city) poziva OpenWeatherMap preko Guzzle klijenta i vraća obrađene podatke (grad, temperatura, opis vremena, vlažnost, brzina vetra) u JSON formatu.

<?php

namespace *App\Http\Controllers*;

use *Illuminate\Http\Request*;

use *GuzzleHttp\Client*;

class WeatherController extends *Controller*

{

    // Dohvata vremenske uslove za dati grad preko OpenWeatherMap API-ja

    public function getWeather($city)

    {

        // Dohvatanje API ključa iz konfiguracionih fajlova

        $apiKey = config('services.openweathermap.key');

        // Kreiranje HTTP klijenta pomoću Guzzle biblioteke

        $client = new *Client*([

            'verify' => false, // Isključivanje verifikacije SSL sertifikata

        ]);

        // Slanje GET zahteva ka OpenWeatherMap API-ju sa prosleđenim gradom i API ključem

        $response = $client->get("https://api.openweathermap.org/data/2.5/weather?q={$city}&appid={$apiKey}&units=metric");

        $data = json\_decode($response->getBody(), true);

        // Vraćanje JSON odgovora sa bitnim informacijama o vremenskim uslovima

        return response()->json([

            'city' => $data['name'], // Naziv grada

            'temperature' => $data['main']['temp'] . '°C', // Temperatura u stepenima Celzijusa

            'weather' => $data['weather'][0]['description'], // Opis vremenskih uslova

            'humidity' => $data['main']['humidity'] . '%', // Vlažnost vazduha u procentima

            'wind\_speed' => $data['wind']['speed'] . ' m/s', // Brzina vetra u metrima u sekundi

        ]);

    }

}

### Laravel autentifikacija (Authentication)

**AuthController** predstavlja centralni deo backend arhitekture zadužen za procese registracije, prijave, odjave i gost pristupa u aplikaciji. Kontroler nasleđuje osnovnu Laravel Controller klasu i koristi ključne servise iz Laravel frameworka: Request za obradu ulaznih podataka, Hash za bezbedno heširanje lozinki, Validator za validaciju unosa, kao i Str za generisanje tokena i UUID vrednosti. Autentikacija i autorizacija realizovane su putem Laravel Sanctum paketa, koji omogućava kreiranje i upravljanje API tokenima, a koji frontend koristi u svakom zaštićenom zahtevu.

Metoda “register” vrši validaciju svih podataka potrebnih za registraciju korisnika. Validiraju se ime, email adresa, lozinka, korisnička uloga (admin, member ili guest) i nivo fizičke spreme (beginner, intermediate, expert). U slučaju greške vraća se odgovor sa listom validacionih poruka i HTTP statusom 400. Nakon uspešne validacije, u bazi se kreira novi korisnik sa heširanom lozinkom, automatski postavljenim vremenom verifikacije email adrese i generisanim “remember\_token” stringom. Nakon kreiranja korisnika, generiše se API token pomoću metode createToken, koji frontend koristi za dalju komunikaciju prema zaštićenim API rutama. Odgovor sadrži status 201, poruku o uspehu, podatke o korisniku i generisani token.

Metoda “login” omogućava prijavu postojećeg korisnika. Najpre se proverava da li su prosleđeni email i lozinka validni, a zatim se u bazi pronalazi korisnik sa odgovarajućim email-om. Ako kombinacija email/lozinka nije ispravna, metoda vraća poruku “Invalid credentials” sa statusom 401. Ako su podaci tačni, kreira se novi Sanctum API token i vraća se JSON odgovor sa porukom o uspehu, korisničkim podacima i tokenom koji frontend čuva u sessionStorage-u.

Metoda “logout” briše sve aktivne Sanctum tokene trenutno prijavljenog korisnika. Na ovaj način korisnik gubi pristup svim zaštićenim rutama, a frontend ga nakon toga preusmerava na stranicu za prijavu. Ovaj pristup predstavlja stateless logout mehanizam, u skladu sa REST arhitekturom, jer server ne čuva klasičnu sesiju već identitet korisnika određuje kroz validnost tokena.

Metoda “loginAsGuest” omogućava kreiranje i prijavu privremenog, gostujućeg korisnika bez potrebe za registracijom. Sistem automatski generiše jedinstveno ime gosta, jedinstvenu email adresu baziranu na UUID identifikatoru i nasumičnu lozinku. Gost se registruje sa ulogom “guest” i inicijalnim fitness\_level parametrom (podrazumevano “beginner”, osim ako je drugačije prosleđeno u zahtevu). Nakon kreiranja naloga generiše se poseban token (“guest\_token”), a odgovor sadrži minimalan skup podataka o korisniku koji su dovoljni za prikaz i rad gost režima. Ovaj funkcionalni mehanizam omogućava pristup aplikaciji bez klasične registracije, što je korisno za demonstraciju funkcionalnosti i brzi ulazak u sistem.

Autentikaciona logika implementirana u AuthController-u jasno razdvaja tok registracije, prijave, odjave i gost pristupa, obezbeđuje sigurnu obradu korisničkih podataka i omogućava jednostavnu integraciju sa React frontendom koji se oslanja na Bearer token mehanizam za pristup backend resursima.

<?php

namespace *App\Http\Controllers*;

use *Illuminate\Http\Request*;

use *App\Models\User*;

use *Illuminate\Support\Facades\Hash*;

use *Illuminate\Support\Facades\Validator*;

use *Illuminate\Support\Str*;

class AuthController extends *Controller*

{

    public function register(*Request* $request)

    {

        // Validacija unosa

        $validator = *Validator*::make($request->all(), [

            'name'          => 'required|string|max:255',

            'email'         => 'required|string|email|unique:users',

            'password'      => 'required|string|min:8',

            'role'          => 'required|in:admin,member,guest',

            'fitness\_level' => 'required|in:beginner,intermediate,expert',

        ]);

        if ($validator->fails()) {

            return response()->json(['errors' => $validator->errors()], 400);

        }

        // Kreiranje korisnika u bazi podataka

        $user = *User*::create([

            'name'              => $request->name,

            'email'             => $request->email,

            'password'          => *Hash*::make($request->password),

            'role'              => $request->role,

            'fitness\_level'     => $request->fitness\_level,

            'email\_verified\_at' => now(),

            'remember\_token'    => *Str*::random(60),

        ]);

        // Provera da li je metoda createToken dostupna

        if (!method\_exists($user, 'createToken')) {

            return response()->json(['error' => 'Sanctum nije ispravno konfigurisan.'], 500);

        }

        // Kreiraj API token

        $token = $user->createToken('auth\_token')->plainTextToken;

        return response()->json([

            'message' => 'User registered successfully',

            'token'   => $token,

            'user'    => $user,

        ], 201);

    }

    public function login(*Request* $request)

    {

        $request->validate([

            'email'    => 'required|string|email',

            'password' => 'required|string',

        ]);

        $user = *User*::where('email', $request->email)->first();

        if (!$user || !*Hash*::check($request->password, $user->password)) {

            return response()->json(['message' => 'Invalid credentials'], 401);

        }

        $token = $user->createToken('auth\_token')->plainTextToken;

        return response()->json([

            'message' => 'Login successful',

            'token'   => $token,

            'user'    => $user,

        ], 200);

    }

    public function logout()

    {

*\Auth*::user()->tokens()->delete();

        return response()->json(['message' => 'Logged out'], 200);

    }

    // Prijava kao gost korisnik

    public function loginAsGuest(*Request* $request)

    {

        // Predefinisani podaci za gosta

        $name    = $request->input('name', 'Guest ' . *Str*::upper(*Str*::random(4)));

        $fitness = $request->input('fitness\_level', 'beginner');

        $email   = 'guest\_' . *Str*::uuid() . '@example.test';

        $user = *User*::create([

            'name'              => $name,

            'email'             => $email,

            'password'          => *Hash*::make(*Str*::random(32)),

            'role'              => 'guest',

            'fitness\_level'     => $fitness,

            'email\_verified\_at' => now(),

            'remember\_token'    => *Str*::random(60),

        ]);

        $token = $user->createToken('guest\_token')->plainTextToken;

        return response()->json([

            'message' => 'Guest account created',

            'token'   => $token,

            'user'    => [

                'id'             => $user->id,

                'name'           => $user->name,

                'role'           => $user->role,

                'fitness\_level'  => $user->fitness\_level,

                'email\_verified' => (*bool*) $user->email\_verified\_at,

            ],

        ], 201);

    }

}

### Laravel resursi (Resources)

**Laravel API resursi (API Resources)** predstavljaju mehanizam unutar Laravel frameworka namenjen transformaciji Eloquent modela u strukturirani JSON format pogodan za REST API komunikaciju. Umesto da se klijentu direktno vraćaju modeli ili kompletni atributi iz baze podataka, resursi omogućavaju preciznu kontrolu nad time koja polja, u kom formatu i kojim redosledom dospevaju do frontenda. Na ovaj način API dobija stabilan i uniforman sloj za prezentaciju podataka, dok backend zadržava fleksibilnost u menjanju modela i interne logike bez narušavanja klijentskih aplikacija.

Svaki resurs nasleđuje JsonResource i definiše metodu toArray(), u kojoj se određuje konačna struktura objekta koji će se isporučiti klijentu. Kao i u većim proizvodnim sistemima, i u ovoj aplikaciji resurse koristimo kako bismo:  
(1) izbegli izlaganje osetljivih ili nepotrebnih podataka,  
(2) formatirali vrednosti (npr. datume) na konzistentan način,  
(3) obezbedili međusobnu usklađenost svih API odgovora.

U nastavku je pregled svih resursa korišćenih u projektu i njihove uloge.

**ExerciseResource**

Ovaj resurs predstavlja JSON prikaz pojedinačne vežbe povezane sa određenim treningom. Transformacija obuhvata osnovna polja modela exercise: id, name, description, reps\_or\_time i type. Polje "type" dodatno se formatira tako da počinje velikim slovom, čime rezultat postaje vizuelno uredniji na stranama korisničkog interfejsa. Resurs takođe vraća i workout\_id, što frontend-u omogućava da odredi kojoj celini vežba pripada bez potrebe za dodatnim HTTP pozivima. Ovaj resurs se koristi u svim prikazima i CRUD operacijama nad vežbama, obezbeđujući konzistentnu strukturu odgovora backend-a.

**GoalResource**

GoalResource definiše JSON format ciljeva koje korisnici kreiraju u okviru aplikacije. Pored osnovnih polja id, user\_id, title i description, resurs vraća i status cilja, kao i target\_date formatiran kao “Y-m-d”, kako bi frontend lako prikazivao datume u formama i pregledima. Takođe se izdvaja kreiranje i ažuriranje datuma u ISO formatu (created\_at i updated\_at), što omogućava standardizovano korišćenje ovih vrednosti u React komponentama. Na ovaj način ciljevi su jasno strukturirani, konzistentni i spremni za prikaz u administratorskom panelu, kao i u drugim delovima aplikacije.

**UserResource**

UserResource transformiše korisnika u JSON format koji se koristi pri prikazu liste korisnika ili u detaljima naloga. Resurs eksplicitno vraća id, ime, email, korisničku ulogu (role) i nivo fizičke spremnosti (fitness\_level). Datum kreiranja naloga formatiran je u obliku “d.m.Y H:i”, što je čitljiv stil za krajnjeg korisnika. Dodatno, resurs koristi whenLoaded mehanizam kako bi se opciono vratio broj treninga korisnika (workouts\_count) samo ukoliko je relacija workouts prethodno učitana u kontroleru. Time se sprečava višak SQL upita, a frontend dobija sve potrebne podatke u jednoj API ruti.

**WorkoutResource**

Ovaj resurs predstavlja centralnu strukturu podataka za treninge unutar aplikacije. Transformacija uključuje osnovne atribute: id, user\_id, naziv treninga (name), opis, trajanje (duration), procenjene kalorije (calories\_burned) i status. Datumi kreiranja i ažuriranja formatirani su u ISO standard (toISOString), što je kompatibilno sa većinom React date-handling biblioteka. Resurs se koristi prilikom prikaza liste treninga, pojedinačnog treninga, kao i nakon kreiranja, izmene ili brisanja treninga, obezbeđujući jedinstven i uredan JSON format kroz sve delove aplikacije.

**ExerciseResource**

<?php

namespace *App\Http\Resources*;

use *Illuminate\Http\Request*;

use *Illuminate\Http\Resources\Json\JsonResource*;

class ExerciseResource extends *JsonResource*

{

    /\*\*

     \* Transform the resource into an array.

     \*/

    public function toArray(*Request* $request): array

    {

        return [

            'id' => *$this*->id,

            'name' => *$this*->name,

            'description' => *$this*->description,

            'reps\_or\_time' => *$this*->reps\_or\_time,

            'type' => ucfirst(*$this*->type),

            'workout\_id' => *$this*->workout\_id,

        ];

    }

}

**GoalResource**

<?php

namespace *App\Http\Resources*;

use *Illuminate\Http\Request*;

use *Illuminate\Http\Resources\Json\JsonResource*;

class GoalResource extends *JsonResource*

{

    /\*\*

     \* Transform the resource into an array.

     \*/

    public function toArray(*Request* $request): array

    {

        return [

            'id'          => *$this*->id,

            'user\_id'     => *$this*->user\_id,

            'title'       => *$this*->title,

            'description' => *$this*->description,

            'status'      => *$this*->status,

            'target\_date' => *$this*->target\_date?->format('Y-m-d'),

            'created\_at'  => *$this*->created\_at?->toISOString(),

            'updated\_at'  => *$this*->updated\_at?->toISOString(),

        ];

    }

}

**UserResource**

<?php

namespace *App\Http\Resources*;

use *Illuminate\Http\Request*;

use *Illuminate\Http\Resources\Json\JsonResource*;

class UserResource extends *JsonResource*

{

    /\*\*

     \* Transform the resource into an array.

     \*/

    public function toArray(*Request* $request): array

    {

        return [

            'id' => *$this*->id,

            'name' => *$this*->name,

            'email' => *$this*->email,

            'role' => *$this*->role,

            'fitness\_level' => *$this*->fitness\_level,

            'created\_at' => *$this*->created\_at->format('d.m.Y H:i'),

            'workouts\_count' => *$this*->whenLoaded('workouts', fn() => *$this*->workouts->count()),

        ];

    }

}

**WorkoutResource**

<?php

namespace *App\Http\Resources*;

use *Illuminate\Http\Request*;

use *Illuminate\Http\Resources\Json\JsonResource*;

class WorkoutResource extends *JsonResource*

{

    /\*\*

     \* Transform the resource into an array.

     \*

     \* @return array<string, mixed>

     \*/

    public function toArray(*Request* $request): array

    {

        return [

            'id'              => *$this*->id,

            'user\_id'         => *$this*->user\_id,

            'name'            => *$this*->name,

            'description'     => *$this*->description,

            'duration'        => *$this*->duration,

            'calories\_burned' => *$this*->calories\_burned,

            'status'          => *$this*->status,

            'created\_at'      => optional(*$this*->created\_at)->toISOString(),

            'updated\_at'      => optional(*$this*->updated\_at)->toISOString(),

        ];

    }

}

### Laravel Rest API Routes

**REST API (Representational State Transfer)** predstavlja stil arhitekture web servisa koji definiše kako klijent i server komuniciraju koristeći standardne HTTP metode poput GET, POST, PUT, PATCH i DELETE. Svaki resurs u sistemu – bilo da je to korisnik, odeljenje ili zahtev za odsustvo – izlaže se putem jedinstvenog URL-a i može se dohvatiti, kreirati, izmeniti ili obrisati razmenom JSON podataka. Ključna karakteristika REST API-ja je stateless princip, što znači da server ne pamti prethodno stanje klijenta između zahteva, čime se pojednostavljuje skaliranje aplikacije i smanjuju komplikacije prilikom otklanjanja grešaka (Red Hat, 2020).

**Postman** je popularan alat namenjen razvoju, testiranju i dokumentovanju REST API-ja. Uz Postman možete lako konfigurisati zaglavlja zahteva, tela poruka i parametre upita, kao i koristiti promenljive okruženja za prebacivanje između razvojnih, testnih i produkcionih servera. Jedna od njegovih najvećih prednosti je mogućnost grupisanja zahteva u kolekcije koje se mogu deliti unutar tima, automatizovati u formi skripti i koristiti za regresiono testiranje API-ja. Dodatno, ugrađeni test editor omogućava pisanje jednostavnih JavaScript assercija na odgovor servera, što ubrzava proces pronalaženja i ispravljanja grešaka (Postman, 2025).

U fajlu **api.php**, rute za neautentifikovane korisnike omogućavaju pristup padu podataka potrebnih za inicijalni rad aplikacije. GET pozivi na /workouts i /exercises vraćaju liste svih treninga i vežbi, dok njihove verzije sa parametrima (/{id}) izdvajaju pojedinačne zapise. Ruta GET /users pruža osnovni pregled korisnika. Javne POST rute /register i /login služe za registraciju i prijavu korisnika, pri čemu se pri uspešnoj prijavi generiše pristupni token kojim se dalje autentifikuju svi zaštićeni zahtevi (Laravel, 2025).

<?php

use *Illuminate\Support\Facades\Route*;

use *App\Http\Controllers\AuthController*;

use *App\Http\Controllers\WorkoutController*;

use *App\Http\Controllers\GoalController*;

use *App\Http\Controllers\ExerciseController*;

use *App\Http\Controllers\AdminController*;

use *App\Http\Controllers\WeatherController*;

// --- PUBLIC (bez tokena)

*Route*::get('/weather/{city}', [*WeatherController*::class, 'getWeather']);

*Route*::post('/register', [*AuthController*::class, 'register']);

*Route*::post('/login', [*AuthController*::class, 'login']);

*Route*::post('/guest/login', [*AuthController*::class, 'loginAsGuest']);

// --- LOGOUT

*Route*::middleware('auth:sanctum')->post('/logout', [*AuthController*::class, 'logout']);

// --- WORKOUTS: dostupno SVIM ulogovanim korisnicima (guest, member, admin)

*Route*::middleware(['auth:sanctum'])->group(function () {

*Route*::get('/workouts', [*WorkoutController*::class, 'index']);

*Route*::get('/workouts/{id}', [*WorkoutController*::class, 'show']);

});

// --- MEMBER (i admin)

*Route*::middleware(['auth:sanctum', *\App\Http\Middleware\RoleMiddleware*::class . ':member'])

    ->group(function () {

        // Users workouts CRUD (sopstveni ili admin)

*Route*::post('/users/workouts', [*WorkoutController*::class, 'store']);

*Route*::put('/users/workouts/{id}', [*WorkoutController*::class, 'update']);

*Route*::delete('/users/workouts/{id}', [*WorkoutController*::class, 'destroy']);

*Route*::get('/users/workouts', [*WorkoutController*::class, 'getUserWorkouts']);

        // Exercises CRUD

*Route*::apiResource('exercises', *ExerciseController*::class);

    });

// --- ADMIN

*Route*::middleware(['auth:sanctum', *\App\Http\Middleware\RoleMiddleware*::class . ':admin'])

    ->group(function () {

*Route*::get('/admin/users', [*AdminController*::class, 'listUsers']);

*Route*::delete('/admin/users/{id}', [*AdminController*::class, 'deleteUser']);

        // Goals CRUD

*Route*::apiResource('goals', *GoalController*::class);

    });

// --- TEST

*Route*::get('/test', function () {

    return response()->json(['message' => 'API radi!']);

});

REST API aplikacije organizovan je tako da jasno odvaja javne rute, rute za ulogovane korisnike i administratorske rute, uz obaveznu upotrebu Laravel Sanctum autentikacije za zaštićene delove sistema. Na ovaj način obezbeđena je sigurnost, predvidljivost i konzistentnost svih interakcija između klijenta i servera.

Javne rute dostupne neulogovanim korisnicima obuhvataju registraciju, prijavu i poseban mehanizam brze prijave kao gost. Ruta */weather/{city}* takođe je javna i koristi se za dohvat vremenske prognoze iz eksternog servisa OpenWeatherMap, što omogućava prikaz vremenskih podataka bez potrebe za autentikacijom. Sve ove rute koriste HTTP POST ili GET metode i vraćaju JSON odgovore prilagođene interfejsu.

Rute koje zahtevaju autentikaciju zaštićene su middleware-om *auth:sanctum*. Nakon što se korisnik uspešno prijavi i dobije Bearer token, sve naredne operacije izvršavaju se u ovlašćenom kontekstu. U ovoj grupi nalaze se rute za pregled treninga (workouts), koje su dostupne svim ulogovanim korisnicima — uključujući gosta, člana i administratora.

Članovi (member) imaju pristup proširenom setu operacija. Uz osnovni pregled treninga, članovi mogu da kreiraju, menjaju i brišu sopstvene treninge pomoću ruta */users/workouts* i */users/workouts/{id}*, kao i da pristupe listi svojih treninga. Pored toga, članovima je omogućen i kompletan CRUD nad vežbama kroz *apiResource* definisan za */exercises*, što uključuje kreiranje, izmenu, brisanje i prikaz pojedinačnih vežbi. Ove rute služe za organizaciju sadržaja treninga i omogućavaju korisniku da precizno personalizuje strukturu svojih aktivnosti.

Administratorske rute predstavljaju najviši nivo pristupa u sistemu i takođe su zaštićene Sanctum autentikacijom, uz dodatni RoleMiddleware koji dozvoljava pristup samo korisnicima sa ulogom *admin*. Administrator ima mogućnost pregleda svih korisnika preko */admin/users* rute, kao i brisanja korisnika pomoću */admin/users/{id}*, pri čemu sistem obavlja odgovarajuće kaskadne promene. Administratorskom nivou takođe pripada kompletan CRUD nad ciljevima (goals), realizovan preko *apiResource* mehanizma za */goals*. Na ovaj način administrator može da kreira ciljeve za korisnike, ažurira ih ili uklanja, u skladu sa potrebama aplikacije.

Sve rute vraćaju JSON format i dosledno koriste standardne HTTP metode:  
• GET – za preuzimanje podataka,  
• POST – za kreiranje novih resursa,  
• PUT – za punu izmenu resursa,  
• PATCH – za delimične izmene,  
• DELETE – za brisanje.

Zahvaljujući jasnoj organizaciji ruta i striktnoj upotrebi Sanctum tokena, API ostaje pregledan, bezbedan i jednostavan za proširivanje, omogućavajući da frontend i backend funkcionišu kao potpuno odvojeni, ali perfektno usklađeni podsistemi.

|  |  |
| --- | --- |
| Opis funkcije | **Registracija člana** |
| HTTP metoda | POST |
| URL | /api/register |
| URL parametri | (nema) |
| HTTP body parametri | {      "name": "Lule",      "email": "lule@gmail.com",      "password": "12345678",      "role": "member",      "fitness\_level": "intermediate"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "message": "User registered successfully",      "token": "8|jXl8KlEZazwhTybT1Gi9xuWPczME3M7akpyXepiv8feb7ad9",      "user": {          "name": "Lule",          "email": "lule@gmail.com",          "role": "member",          "updated\_at": "2025-10-27T19:24:54.000000Z",          "created\_at": "2025-10-27T19:24:54.000000Z",          "id": 36      }  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Prijava korisnika** |
| HTTP metoda | POST |
| URL | /api/login |
| URL parametri | nema |
| HTTP body parametri | {      "email": "miki@gmail.com",      "password": "12345678"  } |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | {      "message": "Login successful",      "token": "9|wAelIoYkSuE8mZiPLljuyiYCgiAgLhmnEMljCTNL9a1cdbee",      "user": {          "id": 31,          "name": "Milan Martic",          "email": "miki@gmail.com",          "created\_at": "2025-06-25T21:54:50.000000Z",          "updated\_at": "2025-06-25T21:54:50.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": null      }  } |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Odjava korisnika** |
| HTTP metoda | POST |
| URL | /api/logout |
| URL parametri | (nema) |
| HTTP body parametri | nema |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "message": "Logged out"  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Prikaz podataka o korisnicima (ADMIN)** |
| HTTP metoda | GET |
| URL | /api/admin/users |
| URL parametri | nema |
| HTTP body parametri | (nema) |
| Format HTTP body parametara | (nema) |
| Izlazni parametri | [      {          "id": 1,          "name": "Mr. Filiberto Lind",          "email": "kenyatta.boyer@example.org",          "created\_at": "2025-06-25T21:53:13.000000Z",          "updated\_at": "2025-06-25T21:53:13.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:10.000000Z"      },      {          "id": 2,          "name": "Dr. Noemi Kreiger Sr.",          "email": "rosenbaum.emelie@example.net",          "created\_at": "2025-06-25T21:53:13.000000Z",          "updated\_at": "2025-06-25T21:53:13.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:10.000000Z"      },      {          "id": 3,          "name": "Joanne Blick",          "email": "melvina.stiedemann@example.org",          "created\_at": "2025-06-25T21:53:13.000000Z",          "updated\_at": "2025-06-25T21:53:13.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:10.000000Z"      },      {          "id": 4,          "name": "Elise Kub",          "email": "mozell.lehner@example.com",          "created\_at": "2025-06-25T21:53:13.000000Z",          "updated\_at": "2025-06-25T21:53:13.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:11.000000Z"      },      {          "id": 5,          "name": "Lon Pfeffer",          "email": "haley.davon@example.com",          "created\_at": "2025-06-25T21:53:13.000000Z",          "updated\_at": "2025-06-25T21:53:13.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:11.000000Z"      },      {          "id": 6,          "name": "Miss Abigail Sanford",          "email": "rose90@example.net",          "created\_at": "2025-06-25T21:53:13.000000Z",          "updated\_at": "2025-06-25T21:53:13.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:11.000000Z"      },      {          "id": 7,          "name": "Dr. Darrell Shanahan",          "email": "meaghan.vandervort@example.com",          "created\_at": "2025-06-25T21:53:13.000000Z",          "updated\_at": "2025-06-25T21:53:13.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:12.000000Z"      },      {          "id": 8,          "name": "Dr. Rhett Konopelski",          "email": "hermina.roberts@example.net",          "created\_at": "2025-06-25T21:53:13.000000Z",          "updated\_at": "2025-06-25T21:53:13.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:12.000000Z"      },      {          "id": 9,          "name": "Elisha Swift",          "email": "keven78@example.org",          "created\_at": "2025-06-25T21:53:13.000000Z",          "updated\_at": "2025-06-25T21:53:13.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:12.000000Z"      },      {          "id": 10,          "name": "Orion Deckow MD",          "email": "aileen25@example.com",          "created\_at": "2025-06-25T21:53:13.000000Z",          "updated\_at": "2025-06-25T21:53:13.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:13.000000Z"      },      {          "id": 11,          "name": "Dustin Schimmel",          "email": "annabell65@example.net",          "created\_at": "2025-06-25T21:53:14.000000Z",          "updated\_at": "2025-06-25T21:53:14.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:13.000000Z"      },      {          "id": 12,          "name": "Presley Berge IV",          "email": "macie.fadel@example.org",          "created\_at": "2025-06-25T21:53:14.000000Z",          "updated\_at": "2025-06-25T21:53:14.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:14.000000Z"      },      {          "id": 13,          "name": "Yesenia Parker",          "email": "eliane.mills@example.net",          "created\_at": "2025-06-25T21:53:14.000000Z",          "updated\_at": "2025-06-25T21:53:14.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:14.000000Z"      },      {          "id": 14,          "name": "Amos Larson",          "email": "destiney02@example.net",          "created\_at": "2025-06-25T21:53:15.000000Z",          "updated\_at": "2025-06-25T21:53:15.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:14.000000Z"      },      {          "id": 15,          "name": "Myriam Kling",          "email": "barton.mathew@example.org",          "created\_at": "2025-06-25T21:53:15.000000Z",          "updated\_at": "2025-06-25T21:53:15.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:15.000000Z"      },      {          "id": 16,          "name": "Seamus Stracke",          "email": "lindgren.davonte@example.com",          "created\_at": "2025-06-25T21:53:15.000000Z",          "updated\_at": "2025-06-25T21:53:15.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:15.000000Z"      },      {          "id": 17,          "name": "Ms. Gerda Ullrich",          "email": "ymaggio@example.com",          "created\_at": "2025-06-25T21:53:16.000000Z",          "updated\_at": "2025-06-25T21:53:16.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:15.000000Z"      },      {          "id": 18,          "name": "Elisabeth Waelchi III",          "email": "nicholaus22@example.org",          "created\_at": "2025-06-25T21:53:16.000000Z",          "updated\_at": "2025-06-25T21:53:16.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:16.000000Z"      },      {          "id": 19,          "name": "Brycen Osinski",          "email": "gibson.vito@example.org",          "created\_at": "2025-06-25T21:53:16.000000Z",          "updated\_at": "2025-06-25T21:53:16.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:16.000000Z"      },      {          "id": 20,          "name": "Prof. Bertha Mante V",          "email": "hudson.keira@example.com",          "created\_at": "2025-06-25T21:53:17.000000Z",          "updated\_at": "2025-06-25T21:53:17.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:16.000000Z"      },      {          "id": 21,          "name": "Olen Schaefer",          "email": "bkassulke@example.com",          "created\_at": "2025-06-25T21:53:17.000000Z",          "updated\_at": "2025-06-25T21:53:17.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:17.000000Z"      },      {          "id": 22,          "name": "Modesto Balistreri",          "email": "june.treutel@example.net",          "created\_at": "2025-06-25T21:53:18.000000Z",          "updated\_at": "2025-06-25T21:53:18.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:17.000000Z"      },      {          "id": 23,          "name": "Albertha Breitenberg",          "email": "jimmie.runolfsdottir@example.com",          "created\_at": "2025-06-25T21:53:18.000000Z",          "updated\_at": "2025-06-25T21:53:18.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:18.000000Z"      },      {          "id": 24,          "name": "Norene Borer",          "email": "gibson.maximillian@example.net",          "created\_at": "2025-06-25T21:53:18.000000Z",          "updated\_at": "2025-06-25T21:53:18.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:18.000000Z"      },      {          "id": 25,          "name": "Ari Fay III",          "email": "rondricka@example.org",          "created\_at": "2025-06-25T21:53:19.000000Z",          "updated\_at": "2025-06-25T21:53:19.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:18.000000Z"      },      {          "id": 26,          "name": "Prof. Kallie Gibson PhD",          "email": "oreilly.lera@example.com",          "created\_at": "2025-06-25T21:53:19.000000Z",          "updated\_at": "2025-06-25T21:53:19.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:19.000000Z"      },      {          "id": 27,          "name": "Domenica McKenzie",          "email": "cormier.dale@example.net",          "created\_at": "2025-06-25T21:53:19.000000Z",          "updated\_at": "2025-06-25T21:53:19.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:19.000000Z"      },      {          "id": 28,          "name": "Katelin Kemmer",          "email": "lesly.raynor@example.org",          "created\_at": "2025-06-25T21:53:19.000000Z",          "updated\_at": "2025-06-25T21:53:19.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:19.000000Z"      },      {          "id": 29,          "name": "Beverly Schmidt",          "email": "marco00@example.org",          "created\_at": "2025-06-25T21:53:20.000000Z",          "updated\_at": "2025-06-25T21:53:20.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:19.000000Z"      },      {          "id": 30,          "name": "Prof. Fae Goodwin Jr.",          "email": "qkris@example.net",          "created\_at": "2025-06-25T21:53:20.000000Z",          "updated\_at": "2025-06-25T21:53:20.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": "2025-06-25T21:53:20.000000Z"      },      {          "id": 31,          "name": "Milan Martic",          "email": "miki@gmail.com",          "created\_at": "2025-06-25T21:54:50.000000Z",          "updated\_at": "2025-06-25T21:54:50.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": null      },      {          "id": 32,          "name": "Milan Martic",          "email": "stojanstole@gmail.com",          "created\_at": "2025-10-27T10:14:10.000000Z",          "updated\_at": "2025-10-27T10:14:10.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": null      },      {          "id": 33,          "name": "Lanmi",          "email": "tukikuki@gmail.com",          "created\_at": "2025-10-27T10:15:43.000000Z",          "updated\_at": "2025-10-27T10:15:43.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": null      },      {          "id": 34,          "name": "Lulee",          "email": "lule@gmail.com",          "created\_at": "2025-10-27T10:21:48.000000Z",          "updated\_at": "2025-10-27T10:21:48.000000Z",          "role": "admin",          "fitness\_level": "beginner",          "email\_verified\_at": null      },      {          "id": 35,          "name": "Lulee",          "email": "lulqweqweqwee@gmail.com",          "created\_at": "2025-10-27T11:24:18.000000Z",          "updated\_at": "2025-10-27T11:24:18.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": null      },      {          "id": 36,          "name": "Lule",          "email": "lules@gmail.com",          "created\_at": "2025-10-27T19:24:54.000000Z",          "updated\_at": "2025-10-27T19:24:54.000000Z",          "role": "member",          "fitness\_level": "beginner",          "email\_verified\_at": null      },      {          "id": 37,          "name": "Djordje Kujundzic",          "email": "djole@gmail.com",          "created\_at": "2025-10-27T19:29:34.000000Z",          "updated\_at": "2025-10-27T19:29:34.000000Z",          "role": "admin",          "fitness\_level": "beginner",          "email\_verified\_at": null      }  ] |
| Format izlaznih parametara | application/json |

|  |  |
| --- | --- |
| Opis funkcije | **Brisanje člana (ADMIN)** |
| HTTP metoda | DELETE |
| URL | /api/admin/users |
| URL parametri | {id} |
| HTTP body parametri | nema |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "message": "Korisnik uspešno obrisan"  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Dodavanje treninga** |
| HTTP metoda | POST |
| URL | /api/users/workouts |
| URL parametri | nema |
| HTTP body parametri | {      "name": "Probni trening",      "description": "Trening hala sportova - fudbal",      "duration": 230,      "calories\_burned": 620  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "status": "success",      "data": {          "name": "Probni trening",          "description": "Trening hala sportova - fudbal",          "duration": 230,          "calories\_burned": 620,          "user\_id": 31,          "updated\_at": "2025-10-28T15:42:36.000000Z",          "created\_at": "2025-10-28T15:42:36.000000Z",          "id": 60      }  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Prikaz svih treninga člana** |
| HTTP metoda | GET |
| URL | /api/users/workouts |
| URL parametri | (nema) |
| HTTP body parametri | nema |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "message": "Workouts retrieved successfully",      "data": [          {              "id": 57,              "user\_id": 31,              "name": "Probni trening",              "description": "Trening hala sportova - fudbal",              "duration": 230,              "calories\_burned": 620,              "status": "pending",              "created\_at": "2025-10-28T15:41:36.000000Z",              "updated\_at": "2025-10-28T15:41:36.000000Z"          },          {              "id": 58,              "user\_id": 31,              "name": "Probni trening",              "description": "Trening hala sportova - fudbal",              "duration": 230,              "calories\_burned": 620,              "status": "pending",              "created\_at": "2025-10-28T15:41:44.000000Z",              "updated\_at": "2025-10-28T15:41:44.000000Z"          },          {              "id": 59,              "user\_id": 31,              "name": "Probni trening",              "description": "Trening hala sportova - fudbal",              "duration": 230,              "calories\_burned": 620,              "status": "pending",              "created\_at": "2025-10-28T15:41:46.000000Z",              "updated\_at": "2025-10-28T15:41:46.000000Z"          },          {              "id": 60,              "user\_id": 31,              "name": "Probni trening",              "description": "Trening hala sportova - fudbal",              "duration": 230,              "calories\_burned": 620,              "status": "pending",              "created\_at": "2025-10-28T15:42:36.000000Z",              "updated\_at": "2025-10-28T15:42:36.000000Z"          }      ]  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Prijava (GOST)** |
| HTTP metoda | POST |
| URL | /api/guest/login |
| URL parametri |  |
| HTTP body parametri | {    "name": "Kiki Krile",    "fitness\_level": "intermediate"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "message": "Guest account created",      "token": "4|DSDF7G9ynhsa5GIbCZUE3mVSbXaiVo7I9u5U8Zib9ad55ad5",      "user": {          "id": 33,          "name": "Kiki Krile",          "role": "guest",          "fitness\_level": "intermediate"      }  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Javni servis – vremenska prognoza (SVI)** |
| HTTP metoda | GET |
| URL | /api/weather/{city} |
| URL parametri | nema |
| HTTP body parametri | nema |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "city": "Belgrade",      "temperature": "20.76°C",      "weather": "clear sky",      "humidity": "37%",      "wind\_speed": "2.06 m/s"  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Izmena odredjenog treninga** |
| HTTP metoda | PUT |
| URL | /api/users/workouts/{id} |
| URL parametri | / |
| HTTP body parametri | {      "name": "Večernje trčanje",      "description": "Trčanje kroz šumu od 7km",      "duration": 45,      "calories\_burned": 350,      "status": "completed"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "data": {          "id": 32,          "user\_id": 31,          "name": "Večernje trčanje",          "description": "Trčanje kroz šumu od 7km",          "duration": 45,          "calories\_burned": 350,          "status": "completed",          "created\_at": "2025-10-29T15:37:37.000000Z",          "updated\_at": "2025-10-29T15:49:20.000000Z"      }  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Brisanje odredjenog treninga** |
| HTTP metoda | DELETE |
| URL | /api/users/workouts/{id} |
| URL parametri | / |
| HTTP body parametri | / |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "status": "success",      "message": "Workout deleted"  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Prikaz svih vezbi** |
| HTTP metoda | GET |
| URL | /api/api/exercises |
| URL parametri | / |
| HTTP body parametri | / |
| Format HTTP body parametara | / |
| Izlazni parametri | [      {          "id": 1,          "name": "amet",          "description": "Eaque non tempora et a.",          "reps\_or\_time": 56,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 1      },      {          "id": 2,          "name": "eos",          "description": "Vel repellendus aut consequuntur aliquam.",          "reps\_or\_time": 50,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 1      },      {          "id": 3,          "name": "quasi",          "description": "Dignissimos ut nobis a odit.",          "reps\_or\_time": 57,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 1      },      {          "id": 4,          "name": "adipisci",          "description": "Est soluta quas ipsum ut.",          "reps\_or\_time": 13,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 1      },      {          "id": 9,          "name": "qui",          "description": "Neque ut accusantium eos similique ut sit.",          "reps\_or\_time": 21,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "cardio",          "workout\_id": 3      },      {          "id": 10,          "name": "nihil",          "description": "Non pariatur autem odio ipsa ut.",          "reps\_or\_time": 51,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "cardio",          "workout\_id": 3      },      {          "id": 11,          "name": "impedit",          "description": "Hic est consectetur qui sed.",          "reps\_or\_time": 55,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "cardio",          "workout\_id": 3      },      {          "id": 12,          "name": "quos",          "description": "Sint fugiat enim dolore qui.",          "reps\_or\_time": 17,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 4      },      {          "id": 13,          "name": "quas",          "description": "Nulla distinctio et voluptatibus eaque aliquam odio illum facilis.",          "reps\_or\_time": 43,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 4      },      {          "id": 14,          "name": "quas",          "description": "Aut beatae corporis aut.",          "reps\_or\_time": 30,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 4      },      {          "id": 15,          "name": "omnis",          "description": "Et praesentium dolorem sed quam eveniet officia.",          "reps\_or\_time": 39,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 4      },      {          "id": 16,          "name": "hic",          "description": "Dolor consequatur perferendis possimus ipsa.",          "reps\_or\_time": 33,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "strength",          "workout\_id": 5      },      {          "id": 17,          "name": "molestiae",          "description": "Culpa est odio minus voluptatem.",          "reps\_or\_time": 18,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "strength",          "workout\_id": 5      },      {          "id": 18,          "name": "harum",          "description": "Aliquam laudantium nihil reiciendis reiciendis nihil veniam.",          "reps\_or\_time": 21,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "strength",          "workout\_id": 5      },      {          "id": 19,          "name": "laboriosam",          "description": "Quasi minus maxime nihil odit in exercitationem.",          "reps\_or\_time": 17,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "strength",          "workout\_id": 5      },      {          "id": 20,          "name": "vel",          "description": "Exercitationem cumque quia non voluptatem laudantium.",          "reps\_or\_time": 45,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 6      },      {          "id": 21,          "name": "voluptatem",          "description": "Et incidunt eveniet explicabo ipsa laborum fuga laboriosam.",          "reps\_or\_time": 24,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 6      },      {          "id": 22,          "name": "maxime",          "description": "Incidunt hic sunt explicabo corporis.",          "reps\_or\_time": 30,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 6      },      {          "id": 23,          "name": "ab",          "description": "Aliquam impedit quia iste dolores et at qui laudantium.",          "reps\_or\_time": 54,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 6      },      {          "id": 24,          "name": "aliquid",          "description": "Autem neque aperiam aut quidem.",          "reps\_or\_time": 29,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 6      },      {          "id": 25,          "name": "omnis",          "description": "Veniam quaerat possimus animi necessitatibus.",          "reps\_or\_time": 23,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "cardio",          "workout\_id": 7      },      {          "id": 26,          "name": "aut",          "description": "Ipsa doloremque quidem quidem iusto voluptates.",          "reps\_or\_time": 28,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "cardio",          "workout\_id": 7      },      {          "id": 27,          "name": "magnam",          "description": "Est veritatis vitae porro illo culpa autem qui.",          "reps\_or\_time": 8,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "cardio",          "workout\_id": 7      },      {          "id": 28,          "name": "omnis",          "description": "Exercitationem ut dolore minus consequuntur aperiam doloribus.",          "reps\_or\_time": 27,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "cardio",          "workout\_id": 7      },      {          "id": 29,          "name": "nemo",          "description": "Quo tempore perspiciatis at est assumenda.",          "reps\_or\_time": 51,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 8      },      {          "id": 30,          "name": "qui",          "description": "Veniam quia eaque repellat maxime itaque a quo.",          "reps\_or\_time": 57,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 8      },      {          "id": 31,          "name": "non",          "description": "Est in facere beatae illo.",          "reps\_or\_time": 5,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 8      },      {          "id": 32,          "name": "eum",          "description": "Aut ut maxime vel velit.",          "reps\_or\_time": 42,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 8      },      {          "id": 33,          "name": "quidem",          "description": "Id quibusdam nulla et tempore beatae.",          "reps\_or\_time": 16,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 8      },      {          "id": 34,          "name": "autem",          "description": "Corporis et quos magni animi consequatur laudantium.",          "reps\_or\_time": 45,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 9      },      {          "id": 35,          "name": "nam",          "description": "Ipsa nemo qui quaerat aliquid eum.",          "reps\_or\_time": 19,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 9      },      {          "id": 36,          "name": "sunt",          "description": "Voluptatem voluptatibus minus ipsa recusandae eaque quidem beatae iusto.",          "reps\_or\_time": 24,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 9      },      {          "id": 37,          "name": "excepturi",          "description": "Quaerat velit dicta consequatur.",          "reps\_or\_time": 50,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 9      },      {          "id": 38,          "name": "ut",          "description": "Rem necessitatibus sapiente sit velit rerum quod ducimus.",          "reps\_or\_time": 33,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 10      },      {          "id": 39,          "name": "ut",          "description": "Eligendi harum pariatur beatae sunt eligendi nam rem.",          "reps\_or\_time": 16,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 10      },      {          "id": 40,          "name": "et",          "description": "Reiciendis deleniti eum et libero reprehenderit voluptatum molestias tenetur.",          "reps\_or\_time": 35,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 10      },      {          "id": 41,          "name": "sed",          "description": "Est voluptates aut sit aut tenetur facere.",          "reps\_or\_time": 35,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 10      },      {          "id": 42,          "name": "rerum",          "description": "Dolorem et dolorum architecto.",          "reps\_or\_time": 60,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 10      },      {          "id": 43,          "name": "quo",          "description": "In quo impedit aut autem quod quidem molestiae.",          "reps\_or\_time": 34,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 10      },      {          "id": 44,          "name": "veniam",          "description": "Sunt possimus laboriosam voluptatem velit reprehenderit sint hic.",          "reps\_or\_time": 40,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "strength",          "workout\_id": 11      },      {          "id": 45,          "name": "asperiores",          "description": "Incidunt sed quia eum sit velit in.",          "reps\_or\_time": 46,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "strength",          "workout\_id": 11      },      {          "id": 46,          "name": "omnis",          "description": "Repudiandae sint quo ut ea error aut omnis odio.",          "reps\_or\_time": 53,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "strength",          "workout\_id": 11      },      {          "id": 47,          "name": "porro",          "description": "Quis repudiandae natus dignissimos nam laudantium.",          "reps\_or\_time": 41,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "strength",          "workout\_id": 11      },      {          "id": 48,          "name": "labore",          "description": "Qui et ut ut eos accusamus optio.",          "reps\_or\_time": 57,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "strength",          "workout\_id": 11      },      {          "id": 49,          "name": "consequatur",          "description": "Laborum vel cupiditate provident ut aliquid consequatur.",          "reps\_or\_time": 27,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "strength",          "workout\_id": 11      },      {          "id": 50,          "name": "a",          "description": "Ullam et neque qui dicta.",          "reps\_or\_time": 45,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 12      },      {          "id": 51,          "name": "quas",          "description": "Quam aut suscipit sed qui ipsum.",          "reps\_or\_time": 39,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 12      },      {          "id": 52,          "name": "et",          "description": "Illo atque et sit aspernatur.",          "reps\_or\_time": 32,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "flexibility",          "workout\_id": 12      },      {          "id": 53,          "name": "sint",          "description": "Eos dolorum ut aut est explicabo voluptatem neque.",          "reps\_or\_time": 43,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "cardio",          "workout\_id": 13      },      {          "id": 54,          "name": "non",          "description": "Veniam nulla et labore sint dolorem non.",          "reps\_or\_time": 60,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "cardio",          "workout\_id": 13      },      {          "id": 55,          "name": "sed",          "description": "Dignissimos nobis atque minus consequatur quia.",          "reps\_or\_time": 37,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T11:16:45.000000Z",          "type": "cardio",          "workout\_id": 13      },      {          "id": 56,          "name": "quia",          "description": "Corrupti est illo enim corporis similique illo.",          "reps\_or\_time": 12,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "cardio",          "workout\_id": 13      },      {          "id": 57,          "name": "expedita",          "description": "Vero quaerat ducimus totam quas.",          "reps\_or\_time": 14,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "cardio",          "workout\_id": 13      },      {          "id": 58,          "name": "quam",          "description": "Rerum ratione omnis harum nostrum et aliquid ducimus harum.",          "reps\_or\_time": 29,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 14      },      {          "id": 59,          "name": "ut",          "description": "Nesciunt nesciunt omnis porro qui quaerat.",          "reps\_or\_time": 32,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 14      },      {          "id": 60,          "name": "doloremque",          "description": "Et corrupti praesentium sequi saepe rerum ut.",          "reps\_or\_time": 45,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 14      },      {          "id": 61,          "name": "ut",          "description": "Quae ex quidem occaecati enim.",          "reps\_or\_time": 18,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 15      },      {          "id": 62,          "name": "non",          "description": "Fugiat unde odio nesciunt accusamus possimus cupiditate neque.",          "reps\_or\_time": 39,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 15      },      {          "id": 63,          "name": "deserunt",          "description": "Omnis ducimus quidem qui qui.",          "reps\_or\_time": 15,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 15      },      {          "id": 64,          "name": "animi",          "description": "Unde qui dolorum nostrum.",          "reps\_or\_time": 32,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 15      },      {          "id": 65,          "name": "ea",          "description": "Maxime beatae est sed perferendis.",          "reps\_or\_time": 41,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 15      },      {          "id": 66,          "name": "velit",          "description": "Sequi adipisci vel nostrum fugiat ut.",          "reps\_or\_time": 50,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 15      },      {          "id": 67,          "name": "sit",          "description": "At quae beatae libero porro.",          "reps\_or\_time": 9,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "flexibility",          "workout\_id": 16      },      {          "id": 68,          "name": "qui",          "description": "Praesentium doloremque ipsa optio.",          "reps\_or\_time": 47,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "flexibility",          "workout\_id": 16      },      {          "id": 69,          "name": "nostrum",          "description": "Vero animi qui rerum mollitia dolor excepturi voluptate.",          "reps\_or\_time": 59,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "flexibility",          "workout\_id": 16      },      {          "id": 70,          "name": "nemo",          "description": "Voluptatum architecto qui qui id dolorem.",          "reps\_or\_time": 16,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 17      },      {          "id": 71,          "name": "dolorum",          "description": "Similique fuga est aut vitae.",          "reps\_or\_time": 42,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 17      },      {          "id": 72,          "name": "ut",          "description": "Et nesciunt sint quod dignissimos omnis esse voluptatem qui.",          "reps\_or\_time": 22,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 17      },      {          "id": 73,          "name": "voluptatem",          "description": "Doloribus et similique exercitationem.",          "reps\_or\_time": 26,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 17      },      {          "id": 74,          "name": "qui",          "description": "Ratione perspiciatis dolorum voluptatum asperiores blanditiis rerum consequuntur.",          "reps\_or\_time": 30,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 17      },      {          "id": 75,          "name": "esse",          "description": "Ea dolores aut id est mollitia.",          "reps\_or\_time": 8,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 17      },      {          "id": 76,          "name": "eveniet",          "description": "Voluptatem modi labore aliquid ipsa et aliquam et.",          "reps\_or\_time": 53,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "flexibility",          "workout\_id": 18      },      {          "id": 77,          "name": "atque",          "description": "Aut animi quis similique.",          "reps\_or\_time": 34,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "flexibility",          "workout\_id": 18      },      {          "id": 78,          "name": "odio",          "description": "Sit voluptas deserunt praesentium iste velit voluptas.",          "reps\_or\_time": 25,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "flexibility",          "workout\_id": 18      },      {          "id": 79,          "name": "iste",          "description": "Adipisci ut quo ipsa dolor beatae.",          "reps\_or\_time": 38,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "flexibility",          "workout\_id": 18      },      {          "id": 80,          "name": "sint",          "description": "Tempore ut pariatur beatae corrupti excepturi quia.",          "reps\_or\_time": 19,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "flexibility",          "workout\_id": 18      },      {          "id": 81,          "name": "nesciunt",          "description": "Quo error quo odit ullam occaecati nemo.",          "reps\_or\_time": 8,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 19      },      {          "id": 82,          "name": "ratione",          "description": "Libero quisquam qui nihil est aut.",          "reps\_or\_time": 39,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 19      },      {          "id": 83,          "name": "quis",          "description": "Recusandae aliquid assumenda magnam dicta.",          "reps\_or\_time": 35,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 19      },      {          "id": 84,          "name": "dicta",          "description": "Quos natus molestiae necessitatibus aut molestias qui illum numquam.",          "reps\_or\_time": 40,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 19      },      {          "id": 85,          "name": "nisi",          "description": "Hic aliquam aliquid aliquid dicta quo qui sunt non.",          "reps\_or\_time": 7,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "strength",          "workout\_id": 19      },      {          "id": 86,          "name": "mollitia",          "description": "Qui ducimus ducimus quidem qui ipsam enim.",          "reps\_or\_time": 46,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "cardio",          "workout\_id": 20      },      {          "id": 87,          "name": "repudiandae",          "description": "Omnis cum quos laudantium ratione.",          "reps\_or\_time": 22,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "cardio",          "workout\_id": 20      },      {          "id": 88,          "name": "vitae",          "description": "Veniam aliquam dolores nihil.",          "reps\_or\_time": 58,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "cardio",          "workout\_id": 20      },      {          "id": 89,          "name": "consequatur",          "description": "Voluptates enim vero ratione.",          "reps\_or\_time": 53,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "cardio",          "workout\_id": 20      },      {          "id": 90,          "name": "ut",          "description": "Explicabo voluptatem id excepturi voluptates earum distinctio distinctio.",          "reps\_or\_time": 49,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "cardio",          "workout\_id": 20      },      {          "id": 91,          "name": "ipsam",          "description": "Nesciunt aperiam est asperiores occaecati.",          "reps\_or\_time": 57,          "created\_at": "2025-10-29T11:16:46.000000Z",          "updated\_at": "2025-10-29T11:16:46.000000Z",          "type": "cardio",          "workout\_id": 20      }  ] |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Dodavanje nove vežbe** |
| HTTP metoda | POST |
| URL | /api/exercises |
| URL parametri | - |
| HTTP body parametri | {      "name": "Skokovi",      "description": "Istezanje",      "reps\_or\_time": 40,      "type": "snaga",      "workout\_id": 1  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "message": "Exercise successfully created.",      "exercise": {          "name": "Skokovi",          "description": "Istezanje",          "reps\_or\_time": 40,          "type": "snaga",          "workout\_id": 1,          "updated\_at": "2025-10-29T18:58:56.000000Z",          "created\_at": "2025-10-29T18:58:56.000000Z",          "id": 92      }  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Prikaz određene vežbe** |
| HTTP metoda | GET |
| URL | /api/exercises/{id} |
| URL parametri | / |
| HTTP body parametri | / |
| Format HTTP body parametara | / |
| Izlazni parametri | {      "id": 1,      "name": "amet",      "description": "Eaque non tempora et a.",      "reps\_or\_time": 56,      "created\_at": "2025-10-29T11:16:45.000000Z",      "updated\_at": "2025-10-29T11:16:45.000000Z",      "type": "flexibility",      "workout\_id": 1  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | Ažuriranje vežbe |
| HTTP metoda | PUT |
| URL | /api/exercises/{id} |
| URL parametri | / |
| HTTP body parametri | {    "name": "Ažurirani naziv vežbe",    "description": "Ovo je ažurirani opis vežbe",    "reps\_or\_time": 60,    "type": "kardio"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "message": "Exercise successfully updated.",      "exercise": {          "id": 1,          "name": "Ažurirani naziv vežbe",          "description": "Ovo je ažurirani opis vežbe",          "reps\_or\_time": 60,          "created\_at": "2025-10-29T11:16:45.000000Z",          "updated\_at": "2025-10-29T20:46:53.000000Z",          "type": "kardio",          "workout\_id": 1      }  } |
| Format izlaznih parametara | JSON |
| Opis funkcije | **Brisanje vežbe** |
| HTTP metoda | DELETE |
| URL | /api/exercises/{id} |
| URL parametri | / |
| HTTP body parametri | / |
| Format HTTP body parametara | / |
| Izlazni parametri | {      "message": "Exercise successfully deleted."  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Dodavanje novog cilja (ADMIN)** |
| HTTP metoda | POST |
| URL | /api/goals |
| URL parametri | / |
| HTTP body parametri | {      "user\_id": 2,      "title": "Smrsati 20kg",      "description": "Smanjenje telesne tezine do decembra",      "target\_date": "2026-04-15"  } |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "status": "success",      "data": {          "user\_id": 2,          "title": "Smrsati 20kg",          "description": "Smanjenje telesne tezine do decembra",          "updated\_at": "2025-10-29T20:53:22.000000Z",          "created\_at": "2025-10-29T20:53:22.000000Z",          "id": 21      }  } |
| Format izlaznih parametara | application/json |
| Opis funkcije | **Prikaži sve ciljeve** |
| HTTP metoda | GET |
| URL | /api/goals |
| URL parametri | / |
| HTTP body parametri | / |
| Format HTTP body parametara | JSON |
| Izlazni parametri | [      {          "id": 1,          "title": "Praesentium quos beatae deserunt sunt maxime est minus et.",          "description": "Doloremque excepturi exercitationem fugit sequi. Maiores illum dolores eaque accusamus fugiat. Blanditiis dignissimos facere aut in eum velit a.",          "user\_id": 11,          "target\_date": "2026-06-15",          "status": "pending",          "created\_at": "2025-10-29T11:16:50.000000Z",          "updated\_at": "2025-10-29T11:16:50.000000Z"      },      {          "id": 2,          "title": "A cumque dolorem itaque sit nam quis.",          "description": "Ut vel nam consequatur aut. Qui temporibus autem eos. Deleniti repudiandae assumenda architecto laborum beatae.",          "user\_id": 12,          "target\_date": "2025-11-01",          "status": "completed",          "created\_at": "2025-10-29T11:16:50.000000Z",          "updated\_at": "2025-10-29T11:16:50.000000Z"      },      {          "id": 3,          "title": "Rerum id et ut veritatis animi.",          "description": "Harum hic id quibusdam autem qui at. Aut esse quisquam quibusdam impedit eveniet voluptas dicta. Dolor autem recusandae natus ab consequatur dolores sunt atque. Aperiam iusto dolor expedita hic.",          "user\_id": 13,          "target\_date": "2026-01-05",          "status": "pending",          "created\_at": "2025-10-29T11:16:50.000000Z",          "updated\_at": "2025-10-29T11:16:50.000000Z"      },      {          "id": 4,          "title": "In consequatur dolorem asperiores ut qui dolor.",          "description": "Eum non voluptas facilis qui blanditiis nisi. Aliquam quae rerum voluptatibus eaque vel. Sequi corrupti voluptatum quia doloribus qui laboriosam possimus.",          "user\_id": 14,          "target\_date": "2026-05-02",          "status": "pending",          "created\_at": "2025-10-29T11:16:50.000000Z",          "updated\_at": "2025-10-29T11:16:50.000000Z"      },      {  ] |
| Format izlaznih parametara | Application/json |
| Opis funkcije | **Prikaži neki određeni cilj** |
| HTTP metoda | GET |
| URL | /api/goals/{id} |
| URL parametri | / |
| HTTP body parametri | / |
| Format HTTP body parametara | JSON |
| Izlazni parametri | {      "id": 4,      "title": "In consequatur dolorem asperiores ut qui dolor.",      "description": "Eum non voluptas facilis qui blanditiis nisi. Aliquam quae rerum voluptatibus eaque vel. Sequi corrupti voluptatum quia doloribus qui laboriosam possimus.",      "user\_id": 14,      "target\_date": "2026-05-02",      "status": "pending",      "created\_at": "2025-10-29T11:16:50.000000Z",      "updated\_at": "2025-10-29T11:16:50.000000Z"  } |
| Format izlaznih parametara | **Application/json** |

### Laravel Database Seeder

**Laravel seederi** su specijalne klase koje služe za inicijalno ili testno popunjavanje baze podataka podacima. One se nalaze u direktorijumu database/seeders i pokreću se komandom php artisan db:seed. Seederi omogućavaju da u razvojnom ili testnom okruženju imate konzistentne i predvidive skupove podataka, što ubrzava razvoj i olakšava pisanje automatskih testova. U praksi, seedere se često kombinuju s model fabrikama (factories), kako bi se generisali realistični, nasumični podaci za različite entitete u aplikaciji (Laravel, 2025).

U samom vrhu hijerarhije seedera je klasa **DatabaseSeeder**. Ona služi kao “glavni” seeder koji u metodi run() poziva sve pojedinačne seedere aplikacije. U našem slučaju, DatabaseSeeder registrovan je da pokrene ExerciseSeeder, WorkoutSeeder, UserSeeder, GoalSeeder (Laravel, 2025).

**DatabaseSeeder**

DatabaseSeeder predstavlja centralnu tačku za pokretanje svih ostalih seedera u aplikaciji. Tokom izvršavanja komande php artisan db:seed, Laravel automatski poziva ovu klasu, koja zatim redom pokreće UserSeeder, WorkoutSeeder, ExerciseSeeder i GoalSeeder. Na ovaj način obezbeđuje se da se baza popuni konzistentnim početnim podacima u tačno definisanom redosledu — prvo korisnici, zatim treninzi, pa vežbe povezane sa njima i na kraju korisnički ciljevi. Ovaj seeder služi kao orkestrator i omogućava jednostavno resetovanje baze tokom razvoja.

<?php

namespace *Database\Seeders*;

use *Illuminate\Database\Seeder*;

// Seeder za popunjavanje baze podataka početnim podacima

class DatabaseSeeder extends *Seeder*

{

    public function run(): void

    {

*$this*->call([

*UserSeeder*::class, // Poziva UserSeeder za kreiranje korisnika

*WorkoutSeeder*::class, // Poziva WorkoutSeeder za kreiranje treninga

*ExerciseSeeder*::class, // Poziva ExerciseSeeder za kreiranje vežbi

*GoalSeeder*::class, // Poziva GoalSeeder za kreiranje ciljeva korisnika

        ]);

    }

}

**UserSeeder**

<?php

namespace *Database\Seeders*;

use *Illuminate\Database\Seeder*;

use *App\Models\User*;

// Seeder za kreiranje korisnika sa različitim ulogama i nivoima fitnesa

class UserSeeder extends *Seeder*

{

    public function run()

    {

        // Kreira jednog admina sa poznatim podacima

*User*::factory()->create([

            'name' => 'Admin User',

            'email' => 'admin@fitnessapp.com',

            'role' => 'admin',

            'fitness\_level' => 'expert',

        ]);

        // Kreira 9 običnih članova sa random vrednostima

*User*::factory(9)->create([

            'role' => 'member',

            'fitness\_level' => fake()->randomElement(['beginner', 'intermediate', 'expert']),

        ]);

    }

}

**ExerciseSeeder**

<?php

namespace *Database\Seeders*;

use *Illuminate\Database\Seeder*;

use *App\Models\Exercise*;

use *App\Models\Workout*;

// Seeder za kreiranje vežbi i njihovo povezivanje sa treninzima

class ExerciseSeeder extends *Seeder*

{

    public function run()

    {

        $workouts = *Workout*::all();

        foreach ($workouts as $workout) {

*Exercise*::factory(rand(3, 6))->create([

                'workout\_id' => $workout->id,

                'type' => fake()->randomElement(['cardio', 'strength', 'flexibility']),

            ]);

        }

    }

}

**WorkoutSeeder**

<?php

namespace *Database\Seeders*;

use *Illuminate\Database\Seeder*;

use *App\Models\Workout*;

// Seeder za kreiranje treninga

class WorkoutSeeder extends *Seeder*

{

    public function run()

    {

*Workout*::factory(20)->create(); // Kreira 20 treninga

    }

}

**GoalSeeder**

<?php

namespace *Database\Seeders*;

use *Illuminate\Database\Seeder*;

use *App\Models\Goal*;

// Seeder za kreiranje ciljeva korisnika

class GoalSeeder extends *Seeder*

{

    public function run()

    {

*Goal*::factory(20)->create(); // Kreira 20 random ciljeva

    }

}

### React Hooks

React Hooks predstavljaju funkcije uvedene u React 16.8 verziji koje omogućavaju korišćenje stanja (state) i drugih React funkcionalnosti unutar funkcijskih komponenti, bez potrebe za klasnim komponentama. Najosnovniji i najčešće korišćeni hook je **useState**, koji omogućava čuvanje i ažuriranje stanja unutar komponente. Pored njega, **useEffect** se koristi za izvršavanje sporednih efekata (side effects), poput asinhronih HTTP poziva prema API-ju, rada sa tajmerima ili manipulacije DOM strukturama, i može se pokretati u zavisnosti od prosleđenih vrednosti u listi zavisnosti. Dodatni hookovi kao što su **useRef**, **useMemo**, **useCallback**, **useReducer** i **useContext** omogućavaju optimizaciju performansi, rad sa složenijim stanjima, referenciranje DOM čvorova i rad sa React kontekstom (Meta, 2025).

Custom hookovi predstavljaju korisnički definisane hookove koji počinju prefiksom **use** i omogućavaju izdvajanje i ponovnu upotrebu logike među različitim komponentama. Oni mogu sadržati kombinacije više standardnih React hookova, upita ka API-ju i unutrašnje poslovne logike, stvarajući apstrakciju koja olakšava održavanje koda i unapređuje njegovu čitljivost. Umesto ponavljanja iste logike u više komponenti, zajedničke funkcionalnosti se izdvajaju u custom hook koji se zatim može pozivati iz bilo koje komponente (Meta, 2025).

U okviru ove aplikacije implementiran je custom hook **useWeather**, koji centralizuje logiku za dobavljanje vremenske prognoze za odabrani grad korišćenjem javnog servisa OpenWeather. Hook koristi **useState** za čuvanje podataka o vremenu, indikator učitavanja i eventualnu poruku greške, dok se preko **useEffect** izvodi HTTP GET poziv ka backend ruti /weather/{city} svaki put kada korisnik unese ili promeni naziv grada. Backend obrađuje zahtev, prosleđuje ga OpenWeather API-ju i vraća JSON odgovor koji hook zatim formatira i prosleđuje komponentama u aplikaciji. Time se obezbeđuje da sve komponente koje prikazuju vremensku prognozu koriste jedinstvenu, ponovo upotrebljivu i čitljivu logiku, bez dupliranja koda. Custom hook tako značajno pojednostavljuje implementaciju i održavanje funkcionalnosti vremenske prognoze u aplikaciji.

// CUSTOM HOOK ZA VREMENSKU PROGNOZU

// src/hooks/useWeather.js

import { useState, useCallback } from "react";

import { getWeather } from "../api/weather";

/\*\*

 \* Custom hook za rad sa vremenskom prognozom.

 \* Brine o stanju: grad, rezultat, loading i error,

 \* i vraća fetchWeather funkciju koju komponente mogu da pozovu.

 \*

 \* @param *{string}* *initialCity* - početni grad

 \* @returns *{{*

 \* *city: string,*

 \* *setCity: (v: string) => void,*

 \* *weather: null | {*

 \* *city: string,*

 \* *temperature: string,*

 \* *weather: string,*

 \* *humidity: string,*

 \* *wind\_speed: string*

 \* *},*

 \* *loading: boolean,*

 \* *error: string,*

 \* *fetchWeather: (cityOverride?: string) => Promise<void>*

 \* *}}*

 \*/

export function useWeather(*initialCity* = "Belgrade") {

    const [city, setCity] = useState(*initialCity*);

    const [weather, setWeather] = useState(null);

    const [loading, setLoading] = useState(false);

    const [error, setError] = useState("");

    const fetchWeather = useCallback(

        async (*cityOverride*) => {

            const queryCity = (*cityOverride* ?? city ?? "").trim();

            if (!queryCity) {

                setError("Grad je obavezan.");

                setWeather(null);

                return;

            }

            setError("");

            setLoading(true);

            setWeather(null);

            try {

                const data = await getWeather(queryCity);

                setWeather(data);

            } catch (e) {

                setError("Ne mogu da dohvatim prognozu. Proveri /api/weather/{city} i CORS.");

            } finally {

                setLoading(false);

            }

        },

        [city]

    );

    return {

        city,

        setCity,

        weather,

        loading,

        error,

        fetchWeather,

    };

}

### React komponente (Components)

**React JSX komponente** predstavljaju osnovne građevinske blokove korisničkog interfejsa u okviru Fitness Web App aplikacije. Svaka komponenta je smeštena u poseban .jsx fajl i kombinuje JavaScript logiku sa HTML-sličnom JSX sintaksom, što omogućava deklarativno opisivanje izgleda i ponašanja interfejsa. Umesto da ručno manipuliše DOM stablom, programer definiše kako UI treba da izgleda za dato stanje podataka, a React se brine o efikasnom ažuriranju prikaza kada se stanje promeni.

U projektu su komponente organizovane modularno po direktorijumima kako bi se jasno razdvojili zajednički layout elementi, specifične stranice (pages) i pomoćna logika. Na vrhu hijerarhije nalazi se folder src/, unutar koga su smešteni direktorijumi components, pages, hooks, context i utils.

U okviru src/components/ui/ nalaze se UI komponente koje se koriste širom aplikacije:  
• **NavBar**.jsx – glavni navigacioni meni koji se prikazuje nakon prijave i omogućava kretanje kroz ključne sekcije aplikacije (početna stranica, javni treninzi, lični treninzi, vežbe i administratorski paneli). Ova komponenta se nalazi iznad Routes dela u App.jsx, pa je prisutna na svim stranama.  
• **ProtectedRoute**.jsx – pomoćna komponenta koja proverava da li je korisnik autentifikovan (na osnovu tokena i konteksta) i, ako nije, preusmerava ga na stranicu za prijavu. Na taj način obezbeđuje da zaštićene rute ne mogu da se otvore bez validne sesije.  
• **RoleRoute**.jsx – dodatni zaštitni sloj koji kontroliše pristup na osnovu korisničke uloge (guest, member, admin). Prima listu dozvoljenih uloga i renderuje decu-komponente samo ako trenutni korisnik ispunjava uslov.  
• **SetTitle**.jsx – mala pomoćna komponenta koja postavlja naslov stranice (document.title) u zavisnosti od prikazane rute, čime unapređuje korisničko iskustvo i jasnoću u browser tabu.  
• **WeatherWidget**.jsx – vidžet za prikaz vremenske prognoze iz OpenWeather servisa. Interno koristi naš custom hook useWeather i prikazuje grad, temperaturu, opis vremena, vlažnost i brzinu vetra.

Direktorijum src/pages/ sadrži “page” komponente, svaka u posebnom podfolderu u zavisnosti od funkcionalne celine:  
• pages/auth/Login.jsx – stranica za prijavu korisnika. Sadrži formu za unos email adrese i lozinke, poziva /api/login na backendu i, u slučaju uspeha, čuva token i osnovne podatke o korisniku putem AuthContext-a i pomoćnih funkcija iz utils/auth.js.  
• pages/home/Home.jsx – početna stranica nakon prijave, koja može da prikazuje kratak pregled aplikacije, pozdravnu poruku i elementi kao što je WeatherWidget (npr. “vreme za trening” po gradu).

Poddirektorijum pages/workouts/ grupiše sve stranice vezane za treninge:  
• **MyWorkouts**.jsx – pregled svih treninga prijavljenog korisnika (member ili admin), uz mogućnost editovanja i brisanja sopstvenih treninga. Komunicira sa zaštićenim rutama /api/users/workouts i koristi token za autentikaciju.  
• **PublicWorkouts**.jsx – lista treninga dostupnih svim ulogovanim korisnicima (guest, member, admin), preuzeta sa backend rute /api/workouts. Služi kao “galerija” primera treninga koja se možda koristi i pri gost pristupu.  
• **PublicWorkoutDetail**.jsx – detaljan prikaz pojedinačnog treninga (opis, trajanje, potrošene kalorije, status), dohvaćen sa rute /api/workouts/{id}.  
• **CreateWorkout**.jsx – forma za kreiranje novog treninga za ulogovanog člana ili administratora. Nakon validacije unosa šalje POST /api/users/workouts.  
• **EditWorkout**.jsx – forma za izmenu postojećeg treninga, koja učitava trenutne podatke, omogućava izmenu i šalje PUT /api/users/workouts/{id}.

U pages/exercises/ nalaze se stranice vezane za vežbe:  
• **Exercises**.jsx – prikaz liste svih vežbi (Exercise entiteta) koje su povezane sa treninzima korisnika. Integrisana je sa backend API resursima za vežbe (Route::apiResource('exercises', ...)), pa može da podržava paginaciju, filtriranje i sortiranje po tipu (cardio, strength, flexibility).  
• **ExerciseForm**.jsx – forma koja služi i za kreiranje i za izmenu vežbe, u zavisnosti od toga da li se stranici pristupa preko /exercises/new ili /exercises/:id/edit. Komponenta koristi isti layout, a po prisustvu parametra id odlučuje da li šalje POST ili PUT zahtev ka API-ju.

Administratorske stranice su grupisane u pages/admin/:  
• **AdminUsers**.jsx – prikaz svih korisnika (osim admina ili sa dodatnim filtrima, u zavisnosti od kontrolera). Ova stranica omogućava administratoru da prelista korisnike, vidi osnovne informacije i pozove brisanje naloga preko DELETE /api/admin/users/{id}.  
• **AdminUserDetails**.jsx – detaljan prikaz pojedinačnog korisnika odabranog sa liste, sa fokusom na ulogu, nivo fitnesa i osnovne statistike (npr. broj treninga).  
• **AdminGoals**.jsx – stranica za pregled ciljeva korisnika (Goal entitet), ograničena samo na admin ulogu. Učitava listu ciljeva preko /api/goals i omogućava pregled statusa, opisa i ciljanog datuma.  
• **AdminGoalForm**.jsx – koristi se i za dodavanje i za izmenu ciljeva. Kao i kod ExerciseForm, forma razlikuje način rada na osnovu postojanja parametra id (kreiranje ili ažuriranje cilja).

U folderu src/context/ nalazi se AuthContext.jsx, koji definiše React Context za autentikaciju. On čuva informacije o trenutno prijavljenom korisniku (token, uloga, ime) i izlaže funkcije za prijavu, odjavu i proveru uloge. ProtectedRoute i RoleRoute koriste ovaj kontekst da bi odlučili da li treba da prikažu decu-komponente ili da korisnika vrate na /login.

Pored toga, u src/hooks/ nalazi se JavaScript fajl useWeather.js, koji implementira custom hook useWeather. Ovaj hook obavlja poziv prema backend ruti /api/weather/{city}, parsira odgovor OpenWeather servisa i komponentama koje ga koriste (npr. WeatherWidget) vraća spreman objekat sa vrednostima temperature, opisa vremena i dodatnih meteo informacija.

Na kraju, direktorijum src/utils/ sadrži fajl auth.js, gde su obično smeštene pomoćne funkcije vezane za rad sa tokenom (čuvanje i čitanje iz localStorage/sessionStorage, dekodiranje, jednostavne provere), čime se izbegava dupliranje logike po komponentama.

Ovakva organizacija React komponenti jasno odvaja ulogu layout elemenata (NavBar, ProtectedRoute, RoleRoute) od konkretnih stranica (Home, Login, MyWorkouts, Exercises, AdminGoals…), dok se posebni folderi za hooks i context brinu o zajedničkoj logici i stanju. Rezultat je pregledan, modularan i lako proširiv front-end deo Fitness Web App aplikacije.

### React Rutiranje (Browser Router)

**React Router** biblioteka omogućava jednostavno i deklarativno upravljanje navigacijom unutar Single Page Application (SPA) arhitekture. Umesto da svaka promena URL-a dovede do kompletnog osvežavanja stranice sa servera, React Router presreće promenu putanje, ažurira stanje u memoriji i renderuje odgovarajuću React komponentu, što dovodi do bržeg i prijatnijeg korisničkog iskustva. (Meta, 2025).

U okviru Fitness Web App projekta koristi se paket react-router-dom, koji obezbeđuje ključne komponente BrowserRouter, Routes i Route. BrowserRouter tipično obavija korensku React komponentu (App) u ulaznoj tački aplikacije (npr. main.jsx), čime se omogućava da sve ugnježdene komponente imaju pristup informaciji o trenutnoj lokaciji (URL putanji) i istoriji navigacije. BrowserRouter se oslanja na HTML5 History API (pushState, replaceState, popstate događaji) kako bi sinhronizovao adresnu traku pregledača sa internim stanjem aplikacije bez slanja klasičnih HTTP GET zahteva za nove HTML stranice.

U samoj App.jsx komponenti definisana je struktura ruta pomoću kombinacije Routes i Route elemenata. Na samom početku se renderuje NavBar komponenta, koja predstavlja globalni navigacioni meni prisutan na svim stranama. Ispod nje se nalazi Routes blok, u kojem se definišu pojedinačne putanje i komponente koje treba prikazati za svaku od njih.

Osnovne javne rute su:  
• **path="/"** → komponenta Home – predstavlja početnu stranicu aplikacije nakon prijave korisnika, s osnovnim pregledom i mogućnošću prikaza vremenske prognoze.  
**• path="/login"** → komponenta Login – prikazuje formu za prijavu korisnika, poziva backend /api/login i, u slučaju uspeha, čuva podatke o sesiji u AuthContext-u i/ili lokalnoj memoriji.

Sve ostale rute u App.jsx su zaštićene preko kombinacije ProtectedRoute i RoleRoute komponenti. ProtectedRoute proverava da li je korisnik ulogovan (ima validan token i podatke u kontekstu), dok RoleRoute dodatno proverava da li korisnik ima odgovarajuću ulogu (“guest”, “member” ili “admin”).

Grupa ruta za javne treninge (dostupne svim ulogovanim korisnicima) izgleda ovako:  
• path="/workouts" – unutar ProtectedRoute i RoleRoute sa allowed=["guest", "member", "admin"]. Ova ruta prikazuje PublicWorkouts komponentu, koja učitava listu treninga preko GET /api/workouts.  
**• path="/workouts/:id"** – takođe za uloge guest, member, admin. Ova ruta prikazuje PublicWorkoutDetail komponentu, koja na osnovu parametra id dohvaća i prikazuje detalje pojedinačnog treninga.

Lični treninzi korisnika (MyWorkouts) ograničeni su na članove i administratore:  
**• path="/users/workouts"** → MyWorkouts komponenta – dostupna samo ako RoleRoute propusti uloge “member” ili “admin”. Prikazuje listu treninga povezanih sa trenutno prijavljenim korisnikom, koristeći backend rute /api/users/workouts.  
**• path="/workouts/new**" → CreateWorkout komponenta – omogućava kreiranje sopstvenog treninga (POST /api/users/workouts).  
• **path="/workouts/:id/edit"** → EditWorkout komponenta – omogućava izmenu postojećeg treninga (PUT /api/users/workouts/{id}).

Rute za vežbe (Exercises) takođe su dostupne samo member i admin ulogama:  
**• path="/exercises"** → Exercises komponenta – prikaz liste svih vežbi sa servera (GET /api/exercises), uz mogućnost filtriranja i pretrage.  
• **path="/exercises/new"** → ExerciseForm komponenta – režim kreiranja nove vežbe (POST /api/exercises).  
**• path="/exercises/:id/edit"** → ExerciseForm komponenta – režim izmene postojeće vežbe (PUT /api/exercises/{id}) na osnovu parametra id.

Administratorski deo sadržan je u grupi ruta gde RoleRoute dozvoljava isključivo ulogu “admin”:  
**• path="/admin/users"** → AdminUsers komponenta – prikazuje tabelu svih korisnika i omogućava administrativne operacije, poput brisanja korisnika preko DELETE /api/admin/users/{id}.  
• **path="/admin/users/:id"** → AdminUserDetail komponenta – detaljan prikaz jednog korisnika.  
• **path="/goals"** → AdminGoals komponenta – prikaz svih ciljeva korisnika (GET /api/goals).  
• **path="/goals/new"** i **path="/goals/:id/edit"** → AdminGoalForm komponenta – kreiranje i izmena ciljeva, povezano sa POST/PUT/DELETE operacijama nad Goal entitetom.

Na kraju definisana je i fallback ruta:  
• **path="\*"** – vraća jednostavnu poruku “Not found” ukoliko korisnik unese nepostojeću putanju u adresnu traku. Ovo obezbeđuje da aplikacija na uredan način obradi sve nepoznate URL-ove.

Kombinacijom BrowserRouter-a (na ulazu aplikacije), NavBar komponente (kao zajedničkog layout elementa) i sistema ruta sa ProtectedRoute i RoleRoute, Fitness Web App ostvaruje jasnu separaciju javnih i zaštićenih stranica, kao i preciznu kontrolu pristupa na osnovu uloga korisnika. Frontend rutiranje je usklađeno sa backend logikom (auth:sanctum middleware i RoleMiddleware), pa korisnik može da pristupi određenoj funkcionalnosti samo ako je za nju ovlašćen i na klijentskoj i na serverskoj strani. Ovo obezbeđuje dobro korisničko iskustvo, a istovremeno zadržava neophodan nivo sigurnosti i konzistentnosti u radu sistema.

import { Routes, Route } from "react-router-dom";

import NavBar from "./components/NavBar.jsx";

import Home from "./pages/Home.jsx";

import Login from "./pages/Login.jsx";

/\* PUBLIC WORKOUTS (ovo već imaš) \*/

import PublicWorkouts from "./pages/PublicWorkouts.jsx";

import PublicWorkoutDetail from "./pages/PublicWorkoutDetail.jsx";

/\* MY WORKOUTS (ovo već imaš) \*/

import MyWorkouts from "./pages/MyWorkouts.jsx";

import CreateWorkout from "./pages/CreateWorkout.jsx";

import EditWorkout from "./pages/EditWorkout.jsx";

/\* ZAŠTITA \*/

import ProtectedRoute from "./components/ProtectedRoute.jsx";

import RoleRoute from "./components/RoleRoute.jsx";

/\* ADMIN \*/

import AdminUsers from "./pages/AdminUsers.jsx";

import AdminUserDetail from "./pages/AdminUserDetails.jsx";

import AdminGoals from "./pages/AdminGoals.jsx";

import AdminGoalForm from "./pages/AdminGoalForm.jsx";

/\* NOVO: EXERCISES STRANICE \*/

import Exercises from "./pages/Exercises.jsx";

import ExerciseForm from "./pages/ExerciseForm.jsx";

export default function App() {

  return (

    <div *style*={{ minHeight: "100vh" }}>

      <*NavBar* />

      <*Routes*>

        {/\* Public \*/}

        <*Route* *path*="/" *element*={<*Home* />} />

        <*Route* *path*="/login" *element*={<*Login* />} />

        {/\* Workouts dostupno svima (guest/member/admin) \*/}

        <*Route*

*path*="/workouts"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["guest", "member", "admin"]}>

                <*PublicWorkouts* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        <*Route*

*path*="/workouts/:id"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["guest", "member", "admin"]}>

                <*PublicWorkoutDetail* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        {/\* My workouts (member/admin) \*/}

        <*Route*

*path*="/users/workouts"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["member", "admin"]}>

                <*MyWorkouts* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        <*Route*

*path*="/workouts/new"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["member", "admin"]}>

                <*CreateWorkout* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        <*Route*

*path*="/workouts/:id/edit"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["member", "admin"]}>

                <*EditWorkout* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        {/\* Exercises (member/admin) \*/}

        <*Route*

*path*="/exercises"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["member", "admin"]}>

                <*Exercises* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        <*Route*

*path*="/exercises/new"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["member", "admin"]}>

                <*ExerciseForm* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        <*Route*

*path*="/exercises/:id/edit"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["member", "admin"]}>

                <*ExerciseForm* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        {/\* Admin \*/}

        <*Route*

*path*="/admin/users"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["admin"]}>

                <*AdminUsers* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        <*Route*

*path*="/admin/users/:id"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["admin"]}>

                <*AdminUserDetail* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        <*Route*

*path*="/goals"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["admin"]}>

                <*AdminGoals* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        <*Route*

*path*="/goals/new"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["admin"]}>

                <*AdminGoalForm* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        <*Route*

*path*="/goals/:id/edit"

*element*={

            <*ProtectedRoute*>

              <*RoleRoute* *allowed*={["admin"]}>

                <*AdminGoalForm* />

              </*RoleRoute*>

            </*ProtectedRoute*>

          }

        />

        {/\* Ako pukne nešto \*/}

        <*Route* *path*="\*" *element*={<div *className*="container section">Not found</div>} />

      </*Routes*>

    </div>

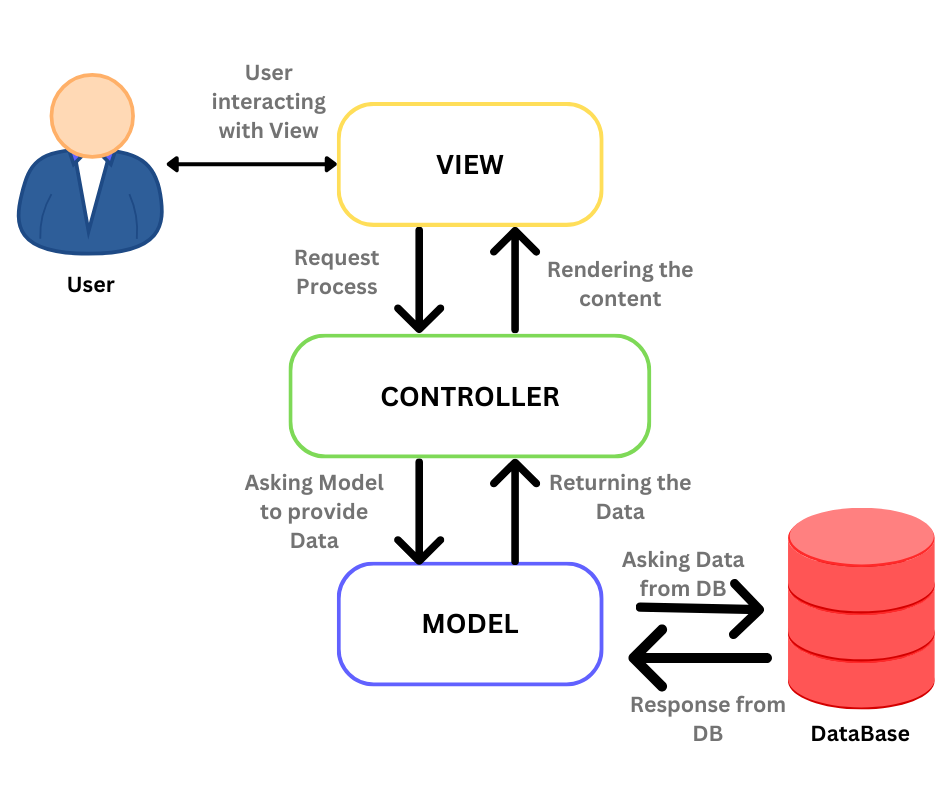
  );

}

### Laravel i React

**React** i **Laravel** često čine snažan tandem za izgradnju modernih web aplikacija, gde Laravel preuzima ulogu robusnog REST API-ja i serverske logike, dok React služi kao dinamični frontend koji interaktivno komunicira sa tim API-jem. U takvom aranžmanu Laravel definiše sve rute unutar api.php (ili, za SPA sa stateful autentifikacijom, unutar web.php uz Sanctum middleware), obrađuje HTTP zahteve, validira podatke, upravlja autentifikacijom i autorizacijom kroz svoje kontrolere i modele. React, sa druge strane, pokreće se u pregledaču kao Single-Page Application, učitavajući jednu HTML stranicu i koristi JavaScript za navigaciju i renderovanje komponenti na osnovu stanja aplikacije (Bhavsar, 2025).

Glavna veza između React-a i Laravel-a je putem **HTTP zahteva** — najčešće **GET**, **POST**, **PUT**, **PATCH** i **DELETE** poziva koje React šalje ka Laravel API endpointima. U React komponentama, najčešće kroz biblioteke poput Axios ili Fetch API, inicira se poziv na rutu. Laravel primaju te zahteve, izvršavaju operacije nad bazom putem Eloquent modela, i vraćaju JSON odgovor. React tada obrađuje dobijene podatke i ažurira stanje UI-ja—bilo da prikazuje listu zahteva, pojedinačni prikaz ili statusne poruke (Bhavsar, 2025).



U **kombinovanom Laravel + React okruženju, MVC patern** se deli između dva sloja tako što Laravel preuzima ulogu Model-Controller dela arhitekture, dok React potpuno preuzima View sloj. To znači da se svi entiteti, njihovi odnosi, validacija i komunikacija sa bazom dešavaju kroz Laravelove Eloquent modele i kontrolere, dok je kompletan korisnički interfejs izgrađen kao hijerarhija React komponenti (Bhavsar, 2025).

Na serverskoj strani, **Model** sloj čine Laravelovi Eloquent modeli – klase koje definišu strukturu tabele, odnose (hasMany, belongsTo i sl.) i eventualne poslovne metode. Svi CRUD zahtevi idu preko **Controller** sloja u Laravelu, gde se validiraju ulazni podaci, izvršavaju transakcije nad bazom i vraćaju JSON odgovori. Ti kontroleri ostaju jedini “izvor istine” za poslovnu logiku i pravila pristupa, pa se u njima nalazi cijela autorizacija, provere uloga i serverske greške.

Na klijentskoj strani, **View** sloj je kompletno izgrađen u Reactu. Svaka stranica, modal ili lista zahteva predstavlja zasebnu JSX komponentu – hijerarhijski uvedene u App.js preko React Router-a. Te komponente ne sadrže nikakvu direktnu logiku za pristup bazi ili validaciju na nivou objekata, već koriste HTTP zahteve (putem Axios-a ili fetch-a) da komuniciraju sa Laravel API-jem. Kada stignu JSON podaci, React state se ažurira i JSX template ponovo renderuje UI u potpunosti, a sve interakcije i animacije ostaju unutar React ekosistema.

Ovakva podela omogućava jasnu separaciju odgovornosti: backend tim može da radi isključivo u Laravelu, brinući se o modelima i kontrolerima, bez brige kako će se podaci prikazati; frontend tim gradi i osvežava React komponente, fokusirajući se na korisničko iskustvo. Laravel funkcioniše kao “headless” API server, dok React postaje jedini HTML/CSS/JS koji pregledač zapravo učitava i prikazuje. Zajedno, oni pružaju robusnu, lako održivu full-stack arhitekturu u kojoj MVC patern ostaje konzistentan – ali raspodeljen na dva specijalizovana sloja.

Autentifikacija je još jedan ključni aspekt njihove integracije. Laravel Sanctum omogućava laganu, token-baziranu autentifikaciju prilagođenu SPA aplikacijama. Tokom prijave, React šalje korisničke kredencijale na Laravelovu rutu /api/login, nakon čega Laravel generiše i vraća access\_token. React čuva taj token (npr. u sessionStorage ili localStorage) i prilaže ga u Authorization zaglavlju svakog narednog zahteva. Laravel potom dekodira i validira token na svakoj zaštićenoj ruti, čime omogućava siguran prenos podataka između frontend i backend sloja.

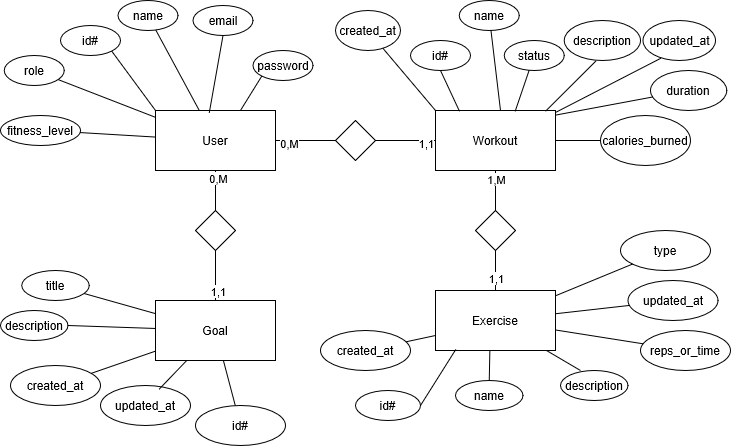
U toku razvoja, Laravel Mix (ili Vite, u novijim verzijama) može služiti za „spajanje“ React koda unutar Laravel projekta. Frontend React fajlovi se nalaze u resources/js/, kompajliraju i izdvajaju u publikovani public/js, dok Laravel može poslužiti i samu SPA stranicu koristeći view('app') ili middleware koji vraća jedinstveni HTML fajl za sve ne-API zahteve. Ovo omogućava lakše upravljanje paketima i zajedničkim asset-ima, kao i hot module replacement tokom razvoja (npm run watch ili npm start).

Konačno, React i Laravel zajedno donose jasnu separaciju odgovornosti: React je zadužen za korisnički interfejs, interakciju i trenutno ažuriranje prikaza, dok Laravel obezbeđuje poslovnu logiku, trajno čuvanje podataka i sigurnost. Ovakav „headless“ pristup omogućava agilniji razvoj oba sloja — frontend tim može nezavisno iterirati na React komponentama, a backend tim može razvijati i skalirati API sloj, uz minimalnu međusobnu zavisnost u kodu.

### Projektovanje strukture softverskog sistema

U **fazi projektovanja strukture softverskog sistema** definiše se tro-slojna arhitektura koja razdvaja korisnički interfejs, aplikacionu logiku i skladište podataka; za svaki entitet iz konceptualnog modela kreiraju se domenske klase koje implementiraju zajednički interfejs za DB-broker, zatim se za svaku poslovnu operaciju (poput kreiranja ili brisanja zapisa) izrađuju posebne klase sistemskih operacija koje obezbeđuju transakcijsko izvršenje i rollback u slučaju greške, a DB-broker samostalno upravlja vezom, transakcijama i materijalizacijom objekata; na kraju se fizičko skladište podataka realizuje prevodom domenskih klasa u relacione tabele sa definisanim primarnim i stranima ključevima, tipovima atributa i podrazumevanim vrednostima, čime se dobija potpuna specifikacija i dizajn slojeva, ponašanja i baze za dalju implementaciju i testiranje.

### Modeli

**Laravel Eloquent modeli** su PHP klase koje preslikavaju (mapiraju) tabele iz vaše baze podataka u objekte u aplikaciji. Svaki model nasleđuje osnovne Eloquent funkcionalnosti, omogućava vam da pišete lekseme poput $user = User::find(1); umesto ručnog pisanja SQL upita, a isto tako definišete i **veze (relations)** između različitih modela (Laravel, 2025).

Na priloženom dijagramu ima 4 modela:

**1. User**

* **Atributi:**  
  id, name, email, password, role, fitness\_level,  
  email\_verified\_at, remember\_token, profile\_picture
* **Veze:**
  + *(implicitno)* ima mnogo treninga → hasMany(Workout::class)
* **Objašnjenje:**  
  Model korisnika čuva osnovne informacije, ulogu i nivo fizičke spremnosti korisnika.  
  Autentikacija se vrši pomoću Laravel Sanctum tokena.

**2. Workout**

* **Atributi:**  
  id, user\_id, name, description, duration, calories\_burned, status,  
  created\_at, updated\_at
* **Veze:**
  + Pripada korisniku → belongsTo(User::class)
  + Ima više vežbi → hasMany(Exercise::class)
* **Objašnjenje:**  
  Trening predstavlja sesiju koju korisnik kreira. Svaki trening je povezan sa korisnikom koji ga je kreirao i može sadržati više vežbi.

**3. Exercise**

* **Atributi:**  
  id, workout\_id, name, description, reps\_or\_time, type,  
  created\_at, updated\_at
* **Veze:**
  + Pripada treningu → belongsTo(Workout::class)
* **Objašnjenje:**  
  Svaka vežba je povezana sa tačno jednim treningom.  
  Polje reps\_or\_time označava broj ponavljanja ili trajanje, dok type definiše tip vežbe (npr. snaga, kardio itd.).

**4. Goal**

* **Atributi:**  
  id, user\_id, title, description, created\_at, updated\_at
* **Veze:**
  + *P*ripada korisniku → belongsTo(User::class)
* **Objašnjenje:**  
  Model ciljeva omogućava korisnicima da postave svoje fitnes ciljeve.  
  Svaki cilj je povezan sa korisnikom koji ga je kreirao.

### Migracije

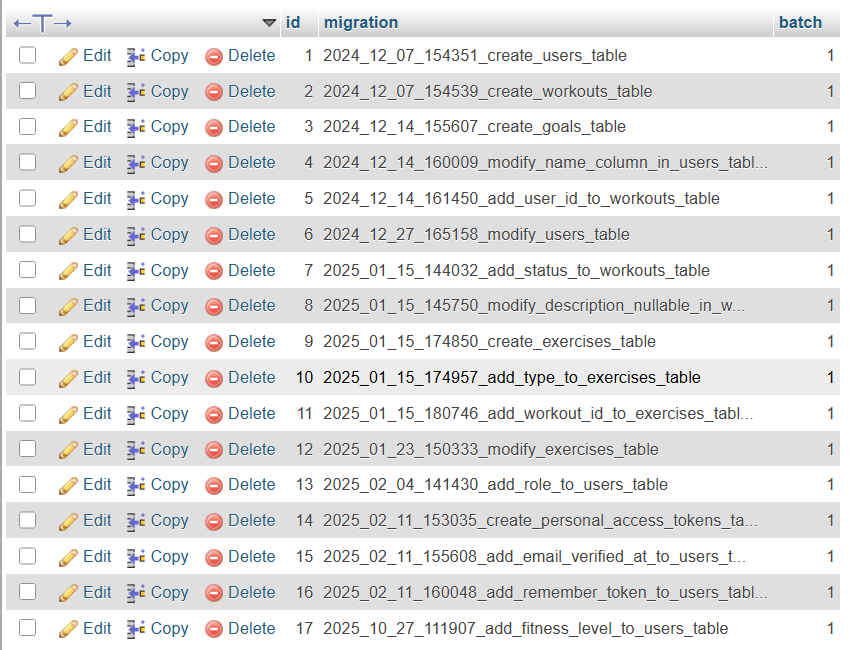
Migracije u Laravelu predstavljaju verzionisane skripte za izmene baze podataka: umesto da ručno

pišemo SQL, svaka migracija definiše strukturu (tabele, kolone, indekse, veze) i omogućava nam da je

primenimo ili poništimo komandom php artisan migrate ili php artisan migrate:rollback. U osnovi, one

čine vaš “shemu” baze čitkom, prenosivom i kontrolisanom kroz Git historiju.

U našem projektu imamo 17 migracija. Od čega neke predstavljaju ispravke prvobitnih logičkih previda, kao i kolone koje su dodate vremenom, kako bi se uklopile sa temom našeg rada.



Prikazaćemo 9 migracija koje bismo izdvojili:

1. **2024\_12\_07\_154351\_create\_users\_table**  
   Kreira osnovnu tabelu users sa kolonama id, name, email, password, created\_at, updated\_at.  
   Ova migracija postavlja temelj Laravel autentikacije i osnovnih korisničkih podataka.
2. **2024\_12\_07\_154539\_create\_workouts\_table**  
   Podiže tabelu workouts koja sadrži kolone id, name, description, duration, calories\_burned, created\_at, updated\_at.  
   Predstavlja osnovnu strukturu za čuvanje korisničkih treninga.
3. **2024\_12\_14\_155607\_create\_goals\_table**  
   Kreira tabelu goals sa kolonama id, title, description, user\_id, timestamps.  
   Koristi se za praćenje ličnih fitnes ciljeva korisnika.
4. **2025\_01\_15\_174850\_create\_exercises\_table**  
   Dodaje tabelu exercises koja povezuje vežbe sa treningom putem workout\_id.  
   Kolone obuhvataju name, description, reps\_or\_time, type, i timestamps.
5. **2025\_01\_15\_144032\_add\_status\_to\_workouts\_table**  
   Dodaje novu kolonu status u tabelu workouts radi praćenja stanja treninga (npr. “u toku”, “završen”).
6. **2024\_12\_14\_161450\_add\_user\_id\_to\_workouts\_table**  
   U tabelu workouts dodaje kolonu user\_id kao strani ključ, čime se svaki trening direktno povezuje sa korisnikom koji ga je kreirao.
7. **2025\_01\_15\_180746\_add\_workout\_id\_to\_exercises\_table**  
   U tabelu exercises dodaje kolonu workout\_id i uspostavlja vezu između vežbi i treninga (belongsTo odnos).
8. **2025\_10\_27\_111907\_add\_fitness\_level\_to\_users\_table**  
   Proširuje tabelu users novom kolonom fitness\_level koja čuva nivo fizičke spremnosti korisnika (“beginner”, “intermediate”, “expert”).
9. **2025\_02\_11\_153035\_create\_personal\_access\_tokens\_table**  
   Kreira tabelu personal\_access\_tokens koja služi Laravel Sanctum sistemu za API autentikaciju i upravljanje korisničkim tokenima.

Za pregled i upravljanje ovim tabelama vizuelno često koristimo **phpMyAdmin**—web-interfejs koji govori sa **MySQL** serverom (ili MariaDB) putem HTTP-a. MySQL je sam RDBMS: servis koji održava tabele, indekse, relacije i sam izvršava SQL upite. U bazi, **tabela migrations** vodi evidenciju koje su migracije već primenjene (svaki red opisuje fajl migracije i batch broj), pa prilikom migrate skripta zna koje izmene treba da preskoči, a koje tek da izvrši. To nam omogućava sinhronizovanu i bezbednu evoluciju šeme kroz različita razvojna ili produkciona okruženja.

### Projektovanje skladišta podataka

**Relacioni model podataka** je tehnika dizajniranja strukture baze zasnovane na konceptualnom modelu, kojom se opisani entiteti i njihovi odnosi prevode u tabele (relacije) koje čine relacijsku bazu podataka. Svaka tabela ima ime i skup kolona, pri čemu svaka kolona odgovara atributu entiteta, a svaki red (slog) predstavlja konkretan primerak tog entiteta. Primarni ključ, definisan nad jednim ili više atributa tabele, obezbeđuje jedinstvenu identifikaciju svakog reda, dok se strani ključevi koriste za uspostavljanje veza između tabela, čime se održava integritet podataka kroz odnose jedan-prema-jedan, jedan-prema-više ili više-prema-više (Vlajić, 2020).

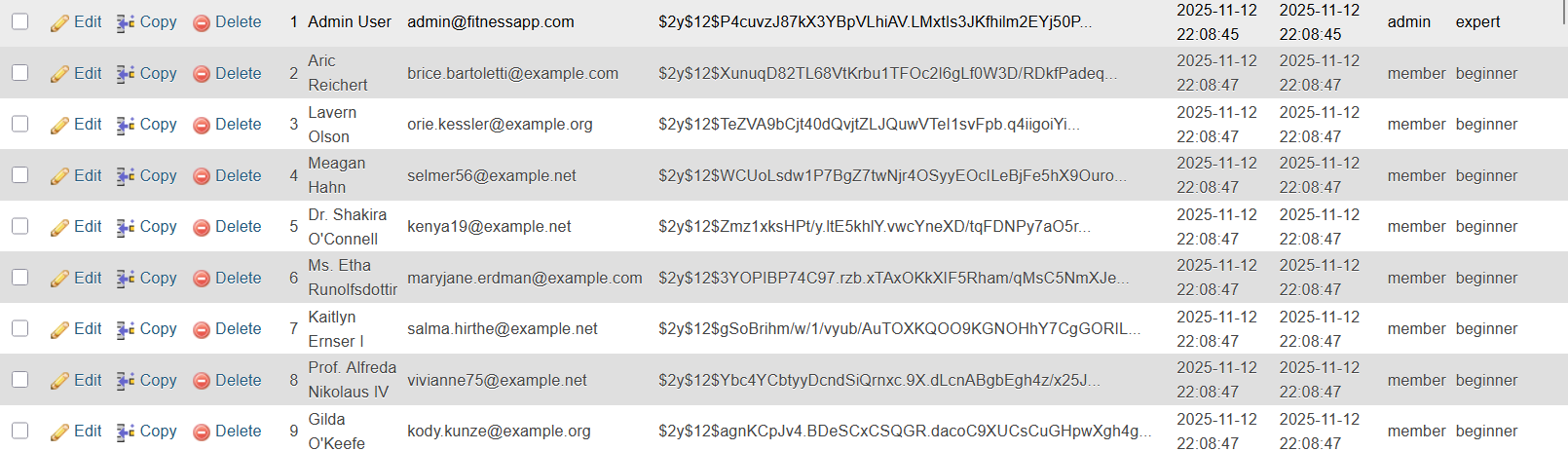
Jedna od ključnih prednosti relacijskog modela je njegova formalna osnova: podaci se mogu manipulisati pomoću standardnog jezika SQL, koji omogućava deklarativno izražavanje upita, ažuriranja i održavanja podataka. Strukturalna ograničenja, poput pravila o kardinalnosti i referencijalne celovitosti, podrazumevano se provode na nivou baze, čime se smanjuje mogućnost nekonzistentnosti ili oštećenja podataka tokom operacija unosa, izmene ili brisanja (Vlajić, 2020).

Pri projektovanju na osnovu relacijskog modela, pored jednostavnih ograničenja na vrednosti atributa (prosta i složena vrednosna ograničenja), važno je definisati i ponašanje sistema prilikom izvršavanja sistemskih operacija – na primer, kaskadno brisanje povezanih slogova pri brisanju nadređenog reda ili poravnanje vrednosti prilikom ažuriranja. Ovaj nivo specificiranja preduslova i postuslova operacija omogućava koordinisano rukovanje kompleksnim promenama podataka, dok se u odeljku strukture softverskog sistema jasno razgraničava uloga baze podataka, kontrolera i aplikacione logike (Vlajić, 2020).

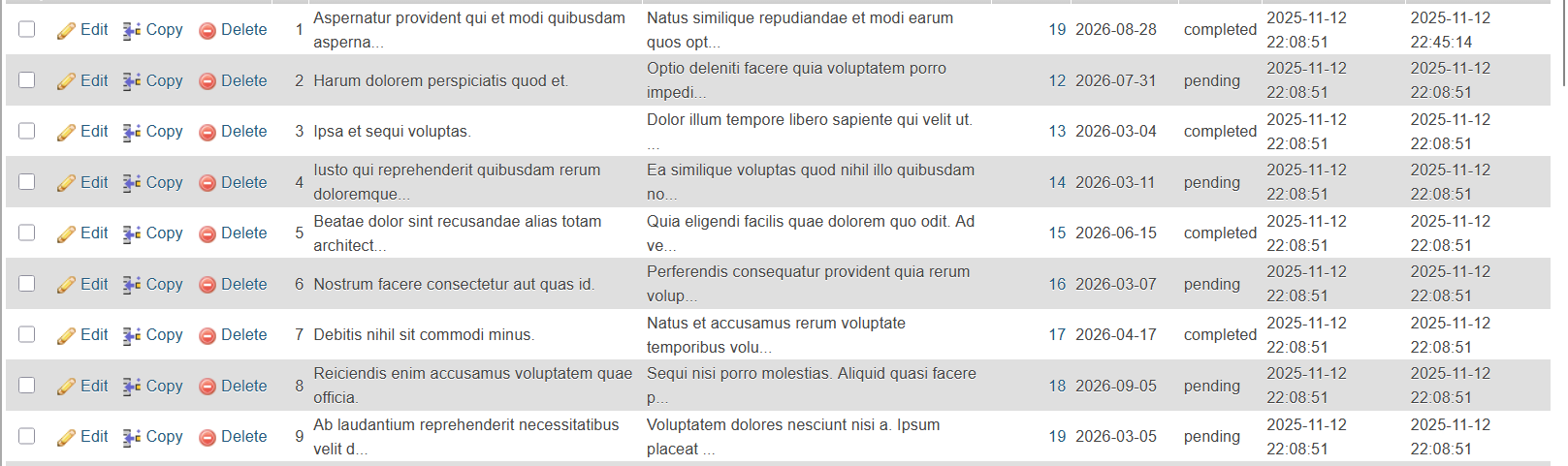
Konačno, relacijski model služi kao temeljna podloga za fizičko projektovanje baze: na osnovu definisanih relacija i ograničenja izrađuju se konkretne SQL DDL naredbe za kreiranje tabela, indeksa i ograničenja. Time se obezbeđuje performansna optimizacija kroz indekse, kao i skalabilnost i sigurnost sistema kroz mehanizme transakcija i zaključavanja koje podržava relacijski SGBD.

Relacioni model **users** tabele:

**User**(id, name, email, password, role, fitness\_level, email\_verified\_at, remember\_token, created\_at, updated\_at)



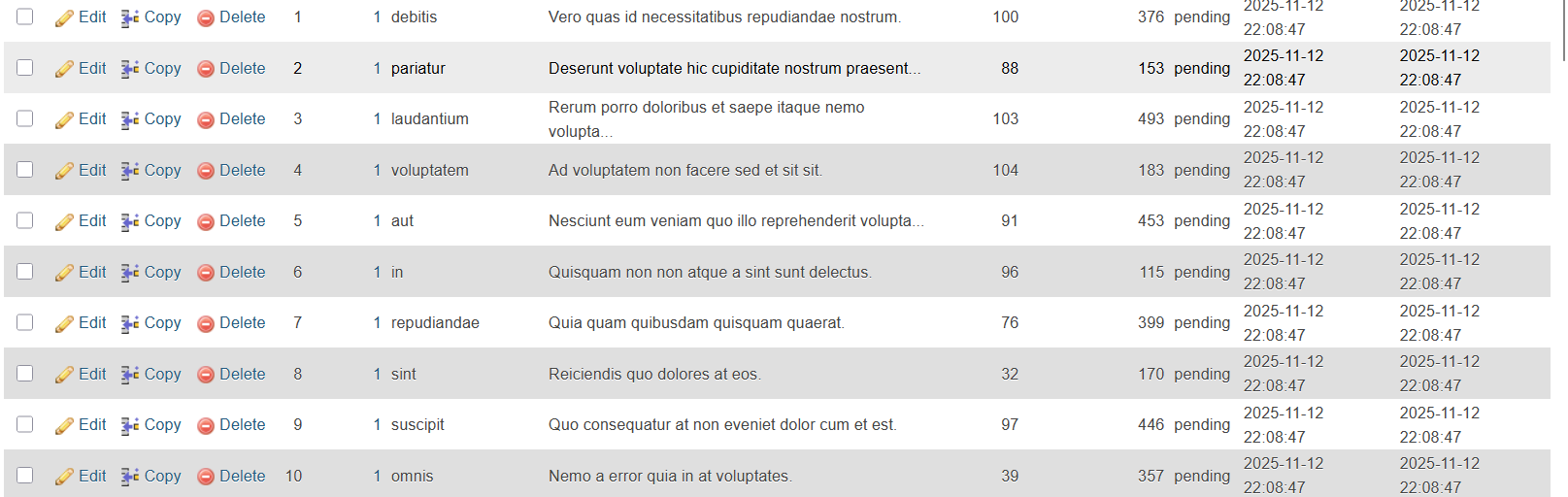
Relacioni model **goals** tabele:

**Goal**(id, user\_id, title, description, status, target\_date, created\_at, updated\_at)

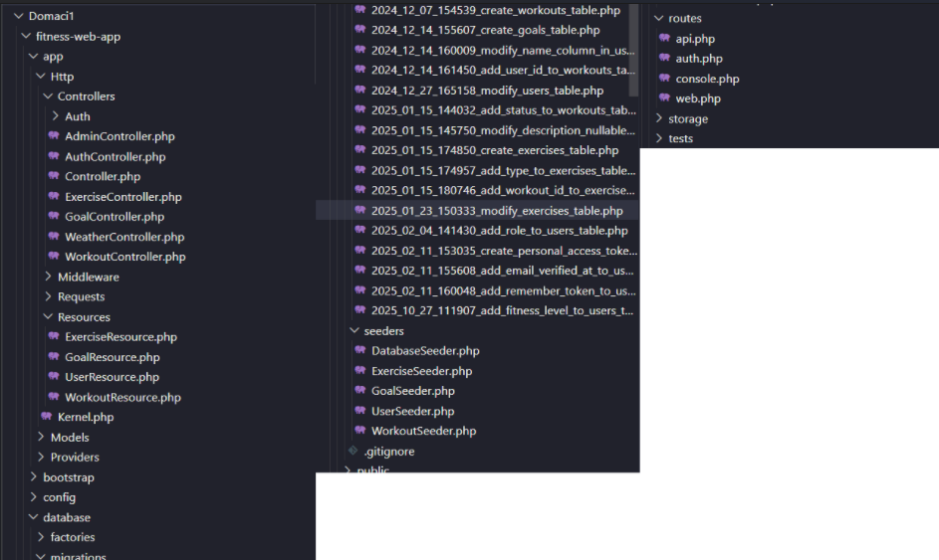
Relacioni model **exercises** tabele:

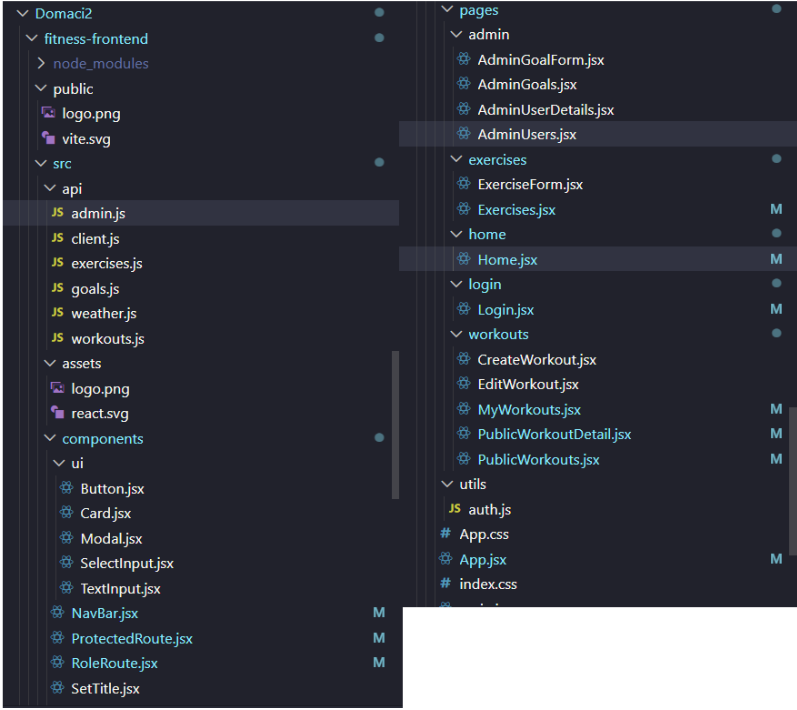
**Exercise**(id, workout\_id, name, description, reps\_or\_time, type, created\_at, updated\_at)

Relacioni model **workouts** tabele:

**Workout**(id, user\_id, name, description, duration, calories\_burned, status, created\_at, updated\_at)

## Faza implementacije

Projekat „Rebel Fitness“ je podeljen na dve jasno odvojene celine: backend Laravel API i frontend React klijent. Laravel aplikacija se nalazi u korenskom direktorijumu backend dela i prati tipičnu strukturu ovog PHP okvira. Direktorijum app/ sadrži poslovnu logiku, dok config/ drži globalna podešavanja aplikacije. U okviru app/Http/Controllers/ grupisani su kontroleri po domenima – posebni kontroleri obrađuju autentikaciju korisnika i rad sa treninzima, vežbama i ciljevima (workouts, exercises, goals), kao i javni servis za vremensku prognozu. Svaki od njih izlaže RESTful API rute za listanje, kreiranje, izmenu i brisanje resursa, uz poštovanje pravila autorizacije. U direktorijumu app/Models/ nalaze se Eloquent modeli User.php, Workout.php, Exercise.php i Goal.php, tako da svaka PHP klasa predstavlja mapiranje na odgovarajuću tabelu u bazi podataka i enkapsulira relacije (npr. korisnik ima više treninga, trening ima više vežbi itd.). API ulazne tačke definišu se u routes/api.php, gde se nalaze rute za registraciju i prijavu korisnika, rad sa treninzima, vežbama, ciljevima i javnu rutu za prognozu vremena. Strukturu baze dodatno opisuju migracije u database/migrations/, dok se inicijalni podaci i test primeri mogu popuniti preko fabrika i seedera u database/factories/ i database/seeders/. Public direktorijum (public/) služi za izlaganje statičkih fajlova i ulazne tačke za frontend build, dok datoteka artisan obezbeđuje komandnu interakciju sa Laravelom (pokretanje migracija, seedera i development servera).

Frontend deo je organizovan u posebnom direktorijumu (npr. fitness-frontend/) i predstavlja samostalnu React aplikaciju pokrenutu uz Vite. U korenu se nalaze package.json, vite.config.\*, public/ i src/. Direktorijum public/ sadrži osnovni HTML šablon (index.html), favicon i logo aplikacije, dok je glavni kod smešten u src/. Tu se nalaze ulazne tačke main.jsx i App.jsx, globalni stilovi (App.css, index.css), kao i podsistemi organizovani po funkcionalnim celinama: src/components/ sadrži generičke komponente kao što su NavBar.jsx, ProtectedRoute.jsx, RoleRoute.jsx, SetTitle.jsx, vidžet za vremensku prognozu (WeatherWidget.jsx) i UI elemente (Button.jsx, Card.jsx, Modal.jsx, TextInput.jsx, SelectInput.jsx). Autentikacija na klijentskoj strani centralizovana je u kontekstu src/context/AuthContext.jsx, koji brine o čuvanju JWT/tokena, trenutnog korisnika i metodama za login, registraciju i guest ulaz. API pozivi prema Laravel serveru enkapsulirani su u src/api/ modulima (npr. fajlovi za rad sa korisnicima, treninzima, vežbama, ciljevima i prognozom), dok se logika za rad sa eksternim servisom vremena nalazi u prilagođenom hook-u useWeather. Stranične komponente su grupisane u direktorijum src/pages/ i dalje podeljene po podfolderima: home/ (početna stranica sa kontakt formom i javnim servisom za vreme), login/ (login i registracija), workouts/ (lista, kreiranje, izmena i detalj treninga), exercises/ (lista i forma za vežbe) i admin/ (administrativne stranice za pregled i uređivanje korisnika i ciljeva). Navigacija se realizuje pomoću React Router-a u App.jsx, gde su definisane javne i zaštićene rute, kao i rute ograničene na određene uloge (guest, member, admin).

Ovakva deoba na Laravel backend i React frontend obezbeđuje jasnu separaciju odgovornosti: backend je zadužen za skladištenje podataka, validaciju, autentikaciju i autorizaciju, dok frontend preuzima prikaz, korisničku interakciju i asinhrone pozive ka API-ju. Zahvaljujući tome, izmene na jednom sloju (npr. dodavanje nove API rute ili migracije za tabelu exercises) ne remete direktno React komponetno stablo, dok vizuelne izmene i prilagođavanje UX-a na klijentu ne zahtevaju redeploy Laravel koda, čime se olakšava razvoj, testiranje i kasnije održavanje aplikacije.

# ZAKLJUČAK

Kroz izradu i dokumentovanje projekta **„Rebel Fitness“** prikazan je kompletan tok razvoja savremene full-stack veb aplikacije, počev od analize korisničkih zahteva pa sve do završne implementacije i testiranja funkcionalnosti. U uvodnim fazama definisani su ključni akteri sistema – gosti, registrovani članovi i administratori – a njihove potrebe formalizovane su kroz jasno opisane slučajeve korišćenja, dijagrame toka i model relacija nad bazom podataka. Ovakav pristup omogućio je da se pre implementacije precizno razume šta svaki tip korisnika očekuje od aplikacije, bilo da je u pitanju jednostavno istraživanje javnih treninga, upravljanje sopstvenim vežbama ili administracija celog sistema.

Na backend strani, implementacija u Laravelu obezbedila je stabilnu osnovu za rad sa podacima. Korišćeni su modeli poput User, Workout, Exercise i Goal, zajedno sa migracijama koje garantuju doslednu strukturu baze i jasne relacije (npr. korisnik → treninzi → vežbe). Kontroleri su modularno organizovani tako da svaka oblast funkcionalnosti ima sopstveni skup REST API ruta, što pojednostavljuje održavanje i logičko proširenje sistema. Autentikacija i autorizacija bazirane su na token-sistemu, što je omogućilo sigurnu komunikaciju između React klijenta i Laravel servera.

Frontend je razvijen kao potpuno odvojen React klijent, baziran na Vite okruženju radi maksimalne brzine i jednostavnog razvoja. Komponente su organizovane po funkcionalnim celinama (home, login, workouts, exercises, admin), čime je postignuta visoka preglednost i lakša kasnija modifikacija. Korišćenje React konteksta (AuthContext) omogućilo je centralizovanu kontrolu korisničke sesije, dok su zaštitne komponente poput ProtectedRoute i RoleRoute osigurale da svaki korisnik može pristupiti samo sadržaju predviđenom njegovoj ulozi. Dodatno, kroz integraciju javnog API servisa za vremensku prognozu demonstrirana je fleksibilnost sistema u komunikaciji sa eksternim izvorima podataka.

Povezivanjem Laravel API-ja i React klijenta obezbeđena je jasna arhitektonska podela odgovornosti. Backend brine o čuvanju i validaciji podataka, dok frontend donosi interaktivan i responzivan korisnički interfejs. Ovakav pristup znatno olakšava buduće nadogradnje – bilo da se uvode nove rute, dodaju novi entiteti u bazu (npr. kategorije treninga) ili proširuju funkcionalnosti administrativnog panela.

Dokumentacija projekta formira temelj za dugoročno održavanje i dalje unapređivanje „Rebel Fitness“ platforme. U budućem radu planirana su proširenja kao što su notifikacije o napretku, detaljnije analitike treninga, dodatni javni servisi (npr. merenje kalorijske potrošnje) i mogućnost deljenja trening rutine sa drugim korisnicima. Na osnovu jasno definisanih faza, modularne arhitekture i upotrebe pouzdanih tehnologija, ovaj sistem predstavlja skalabilnu i sigurnu osnovu za sve naredne verzije aplikacije.

# LITERATURA

Bhavsar, D. (2025, March 24). *How to Use Laravel with React for Building Modern Web Applications.* Preuzeto sa BACANCY: https://www.bacancytechnology.com/blog/laravel-with-react

Laravel. (2025). *Laravel Official Documentation.* Preuzeto sa Laravel: https://laravel.com/docs/12.x

MDN. (2025). *JavaScript.* Preuzeto sa MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/JavaScript

Meta. (2025). *Quick Start.* Preuzeto sa React: https://react.dev/learn

mohamadayash22. (2025). *real-inspire.* Preuzeto sa GitHub: https://github.com/mohamadayash22/real-inspire

MySQL. (2025). *MySQL Documentation.* Preuzeto sa MySQL: https://dev.mysql.com/doc/

PHP. (2025). *PHP Official DOcumentation.* Preuzeto sa PHP: https://www.php.net/docs.php

phpMyAdmin. (2025). *phpMyAdmin Documentation.* Preuzeto sa phpMyAdmin: https://www.phpmyadmin.net/docs/

Postman. (2025). *Postman Documentation.* Preuzeto sa Postman: https://learning.postman.com/docs/introduction/overview/

RandomUser.me. (2025). *Random User Generator.* Preuzeto sa RandomUser.me: https://randomuser.me/

Red Hat. (2020, May 8). *What is a REST API?* Preuzeto sa Red Hat: https://www.redhat.com/en/topics/api/what-is-a-rest-api

Vlajić, S. (2020, October). *ПРОЈЕКТОВАЊЕ СОФТВЕРА (СКРИПТА - РАДНИ МАТЕРИЈАЛ - 2020).* Preuzeto sa ResearchGate: https://www.researchgate.net/publication/344459754\_PROJEKTOVANE\_SOFTVERA\_SKRIPTA\_-\_RADNI\_MATERIJAL\_-\_2020

W3Schools. (2025). *CSS Tutorial.* Preuzeto sa W3Schools: https://www.w3schools.com/css/

W3Schools. (2025). *HTML Tutorial.* Preuzeto sa W3Schools: https://www.w3schools.com/html/