

ZAVRŠNI RAD

Fitnes aplikacija za praćenje dnevnih aktivnosti treninga

Mentor: Kandidat:

**Prof. dr Milena Bogdanović**  **Aleksa Cekić**

Sadržaj:

[**1. Uvod 3**](#_2fi4sv69hura)

[1.1. Motivacija za izradu završnog rada 3](#_5mswyvw6asz5)

[1.2. Pregled nekih postojećih fitnes aplikacija 4](#_ycculjltpxxy)

[**2. Baze podataka 5**](#_ynj2dorv6sq)

[2.1. Relacione baze podataka 5](#_kcyyug8rihmb)

[2.2. MySQL baza podataka 6](#_gye8ucm5uql0)

[2.3. Tipovi veza 6](#_2nb0hv8a6zde)

[**3. Opis i izbor korišćenih tehnologija i biblioteka 8**](#_6k0f1r6nwi5)

[3.1. React radni okvir za korisnički interfejs 8](#_aqnmca405s1m)

[3.2. Spring Boot radni okvir za logiku aplikacije 9](#_ktwehky2ukw2)

[3.3. MySQL baza podataka za fitness aplikaciju 10](#_fn36byuo1xfu)

[**4. Zahtevi sistema 12**](#_3k6inka6r5lm)

[4.1. Funkcionalni zahtevi 12](#_fffr78bujwt8)

[4.2. Nefunkcionalni zahtevi 13](#_988nwytloke)

[**5. Implementacija 14**](#_ga7l79v7rva3)

[5.1. Slučajevi korišćenja 15](#_sz0fg32md9u7)

[5.2. Arhitektura projekta 20](#_7ykqx33mjqa3)

[5.3. Sistem sekvencijalnih dijagrama 23](#_ca0sx47e2o6s)

[5.4. Baza podataka 27](#_9m4ilhkh76h)

[5.5. Struktura projekta 29](#_dw4ykihu5c4b)

[5.6. Obrada izuzetaka 34](#_6fcuu7b7hffl)

[5.7. Testiranje aplikacije 37](#_df0djqpywa66)

[5.7.1. Jedinično testiranje 38](#_ouy8nol26cni)

[5.7.2. Integraciono testiranje 41](#_ecd8ao47a0te)

[**6. Demonstracija funkcionalnosti aplikacije 45**](#_v0snv4s51r46)

[**7. Uputstvo za korisnike 55**](#_86sk8ibxm1eb)

[**8. Zaključak 56**](#_e4mva5wg552i)

[**9. Literatura 57**](#_v5hv3zrmp2a5)

# Uvod

## Motivacija za izradu završnog rada

Kako je brzina života postala nezaustavljiva, fokus stavljamo na različite, nekada i ne toliko bitne, stvari. Očuvanje fizičkog zdravlja postaje izazov koji zahteva raznovrsne pristupe. Motivacija za izradu “Fitnes aplikacija za praćenje dnevnih aktivnosti treninga” izranja iz potrebe za stvaranjem efikasnog alata koji će podržati pojedince u održavanju zdravih životnih navika.

Veliki broj osoba provodi veliki broj sati obavljajući svoje poslove stacionirano, sedeći u kancelarijama ili od kuće. Ovaj sedentarni način života postaje sveprisutan i donosi sa sobom niz izazova po naše fizičko zdravlje. Svesni smo da je neophodno pronaći balans između zahteva modernog života i brige o sopstvenom zdravlju.

Upravo iz ovog razloga, fitnes aplikacije postaju ključne, pružajući korisnicima alat koji olakšava integrisanje fizičke aktivnosti u njihovu svakodnevnicu. Cilj je prevazići prepreke koje se pojavljuju usled brzog tempa života, obezbeđujući prilagođene treninge koji se lako uklapaju u naše živote ispunjene obavezama. Inicijativa za fizičkom aktivnošću je okrenuta ka omogućavanju svakodnevnog treninga u teretnani.

Aplikacija ne samo da prati aktivnosti korisnika, već pruža i sistem praćenja koji omogućava postizanje ciljeva u vezi sa dnevnikom hrane i nutricionizmom. Ovaj aspekt je od suštinskog značaja, jer korisnicima omogućava jasno praćenje preporučenog dnevnog unosa kalorija. Nakon registracije, aplikacija automatski izračunava masu i brzinu metabolizma korisnika, pružajući im informacije o tome da li se nalaze u kalorijskom deficitu ili suficitu.

Uzimajući u obzir sve ove faktore, motivacija za izradu ove fitnes aplikacije postaje očigledna. Želja da se prevaziđu izazovi modernog života, podrži očuvanje fizičkog zdravlja i podstakne motivacija za redovno vežbanje, pokreće nas ka stvaranju inovativnog rešenja koje će postati neizostavan deo svakodnevnog života mnogih.

Kao student Softverskog Inženjerstva na Fakultetu Informacionih Tehnologija, osećao sam da je izrada ovog završnog rada savršena prilika da pripremim i proširim svoje teorjisko znanje stečeno tokom studija kao i da naučim nešto novo o celokupnom svetu fitnesa.

Ovaj projekat predstavlja spoj mog stručnog interesovanja za softversko inženjerstvo i ličnog angažovanja o očuvanju fizičkog zdravlja. Razvijanje “Fitnes aplikacija za praćenje dnevnih aktivnosti treninga” nike samo tehnički izazov već i prilika da steknem dublje razumevanje potreba korisnika u oblasti fizičkog vežbanja i zdravog života.

## Pregled nekih postojećih fitnes aplikacija

Na tržištu postoji veliki niz fitnes aplikacija koje nude različite funkcionalnosti i prilaze u cilju podrške korisnicima u održavanju aktivnog i zdravog načina života. Ovo su neki od popularnijih fitnes aplikacija:

1. **MyFitnessPal**
   * Služi za praćenje dnevnog unosa hrane, brojanje kalorija, vežbanje i praćenje fizičkih aktivnosti, sadrži veliku bazu podataka sa prehrambenim informacijama.
2. **Lifesum**
   * Praćenje ishrane, planiranje obroka, praćenja unosa vode, vežbe i fizičke aktivnosti.
3. **Nike Training Club**
   * Prilagođeni trening planovi, vežbe od kuće ili u teretani, video uputstva i praćenje napredka.
4. **Strava**
   * Praćenje trčanja, vožnje bicikla, plivanja i drugih aktivnosti, analiza rute, postizanje ciljeva.
5. **Fitbit**
   * Praćenje koraka, praćenje sna, merenje otkucaja srca, praćenje kalorija, trening planovi.

# Baze podataka

Baza podataka je organizovan i struktuiran skup podataka koji se čuva digitalno u računarskom sistemu. Baze podataka omogućavaju efikasno upravljanje, skladištenje i manipulaciju podacima kako bi se zadovoljile različite informacione potrebe organizacija ili pojedinca.

Ciljevi baza podataka su organizacija podataka, efikasno pretraživanje podataka, deljenje, održavanje, sigurnost, prilagođenost i integritet podataka.

One se često koriste u različitim sektorima, a neki od njih su poslovni, naučni, medicniski, obrazovni, itd.

Baze podataka mogu se podeliti na Relacione baze podataka (Relational Database Management System ili RDBMS) i Neracionalne baze podataka.

Razlika između relacionih i nerelacionih baza podataka je u načinu organizacije i povezivanju podataka.

## Relacione baze podataka

Relacione baze podataka predstavljaju ključnu komponentu u IT svetu, pružajući strukturu i mehanizme za efikasno upravljanje podacima. U najpraktičnijem smislu, relaciona baza podataka može se shvatiti kao sistem međusobno povezanih tabela, u kome se teži što racionalnijem i što ekonomičnijem zapisu podatka.

Podaci se čuvaju po **tabelama** koje predstavljaju **entitete**, objekte ili pojave koje želimo opisati podacima. Na primer, tabela “Korisnik” može predstavljati entitet ili entitete koji opisje pojedince u sistemu. Svaka tabela ima kolone koje predstavljaju atribute entiteta. To su karakteristike ili svojstva entiteta. U prethodnom primeru “Korisnik” može imati atribute ili kolone poput “Ime”, “Prezime”, “Email”, “Sifra”, “GodinaRodjenja”. Unutar tabela, redovi sadrže konkretne podatke ili instance entiteta. Svaki red u tabeli “Korisnik” može predstavljati jednog korisnika sa odgovarajućim vrednostima za svaki atribut.

U tabelama takođe postoje **unikatnosti** i **primarni ključevi**. Primarni ključ predstavlja jednu ili više kolona koje jednoznačno identifikuju svaki red u toj tabeli. Ovo omogućava jedinstvenost i referenciranje na podatke. Obično ovim označavamo neke indekse ili ID-jeve. **Sporedni ključ** je primarni ključ iz druge tabele, koristi se za uspostavljanje veza između tabela. Na primer, ako postoji tabela sa nazivom “Rola”, sporedni ključ u tabeli “Korisnik” može biti referenca na primarni ključ u tabeli “Rola”.

**Normalizacija** je proces organizovanja podataka radi eliminisanja redundancije i očuvanja integriteta podataka. Normalizacija vodi do deljenja podataka između više tabela radi postizanja optimalne strukture baze podataka. **Denormalizacija** se primenjuje radi poboljšanja performansi. Preporučuje se da baze podataka koje samo čitaju ostanu denormalizovane. Podaci se organizuju sa dodatnom redundancijom kako bi se smanjilo vreme izvršavanja upita.

Nasuprot relacionim bazama podataka, nerelacione baze podataka dizajnirane su da pruže veću fleksibilnost i agilnost u rukovanju podacima. Ove baze podataka ne koriste tradicionalni model sa tabelama, redovima i kolonama, već koriste različite modele podataka koji odgovaraju različitim potrebama i strukturama podataka.

## MySQL baza podataka

MySQL je popularni sistem za upravljanje relacionim bazama podataka koji se često koristi u različitim vrstama aplikacija, od malih veb sajtova do velikih poslovnih sistema. Jedna od ključnih karakteristika MySQL baza podataka je ta da je on softver otvorenog izvora. Korisnici imaju slobodu da prilagode i deluju na izvorni kod sistema prema svojim potrebama. Ovaj model omogućava zajednici programera da doprinosi razvoju i održavanju sistema. MySQL je poznat po svojoj efikasnosti i visokim performansama, što ga čini popularnim izborom za aplikacije koje zahtevaju brze i pouzdane odgovore na upite. Navedena karakteristika je posebno važna u veb razvoju i poslovnim sistemima sa velikim brojem korisnika. Takođe je multiplatformski i podržava rad na različitim operativnim sistemima, uključujući Linux, Windows i macOS.

Koristi Structured Query Language (SQL) za definisanje i manipulaciju podacima. SQL je standardni jezik za upravljanje relacionim bazama podataka i omogućava korisnicma da izvršavaju različite operacije, uključujući pretragu, dodavanje, ažuriranje i brisanje podataka.

Što se tiče bezbednosti, MySQL ima različite mehanizme zaštite podataka kao što su: autentifikacija korisnika, enkripcija podataka, mogućnost definisanja pristupnih privilegija na nivou tabela i baza podataka. Ove funkcije doprinose očuvanju bezbednosti podataka u sistemu.

## Tipovi veza

U relacionim bazama podataka veze igraju ključnu ulogu u organizaciji podataka, održavanju referencijalnog integriteta i omogućavanju efikasnog čitanja informacija. Veze uspostavljaju odnose između entiteta različitih tabela, stvarajući strukturu podataka koja odražava realne veze i zavisnosti unutar sistema.

Osnovni tipovi veza u relacionim bazama podataka:

1. **Jedan prema Jedan (1:1):**

Ovaj tip veza označava da svaki entitet u jednoj tabeli ima tačno jedan povezani entitet u drugoj tabeli i obrnuto. Na primer, Tabela “Student” može imati 1:1 vezu sa tabelom “Mentor”.

1. **Jedan prema Više (1:N):**

Ovaj tip veza označava da svaki entitet u jednoj tabeli može imati više povezanih entiteta u drugoj tabeli, ali svaki entitet u drugoj tabeli može imati tačno jedan povezani entitet u prvoj tabeli.

1. **Više prema Jednom (N:1):**

Svaki entitet u jednoj tabeli može imati tačno jedan povezani entitet iz druge tabele, ali entitet u drugoj tabeli može imati više povezanih entiteta u prvoj tabeli.

1. **Više prema Više (N:N):**

Ovaj tip veze označava da svaki entitet u jednoj tabeli može imati više povezanih entiteta u drugoj tabeli, i obrnuto. Da bi se implementirala ova veza, često se koristi asocijativni entitet.

1. **Rekurzivna Veza:**

Ova veza označava situaciju kada entitet u jednoj tabeli može biti povezan sa samim sobom. To se obično primenjuje kada postoji hijerarhija ili struktura koja podrazumeva odnos entiteta prema samom sebi, značajno kada se jedan entitet javlja u različitim ulogama u okviru veze sa samim sobom.

Pomoću ovih tipova veza, programeri mogu da modeliraju složene odnose između entiteta i da očuvaju integritet podataka u relacionim bazama. Odabir odgovarajućeg tipa veze zavisi od specifičnih zahteva i strukture podataka u okviru sistema baze podataka.

# Opis i izbor korišćenih tehnologija i biblioteka

## React radni okvir za korisnički interfejs

ReactJs je popularni *JavaScript* radni okvir razvijen od strane Facebook-a za izgradnju modernih korisničkih interfejsa u veb aplikacijama. Ovaj radni okvir se ističe po svojoj efikasnosti, brzini i reaktivnosti, a koristi se za izradu veb aplikacije u okviru projekta “Fitnes aplikacija za praćenje dnevnih aktivnosti treninga”. Fokusira se na jednostavno upravljanje stanjem komponenti kako bi omogućio efikasan prikaz promena u korisničkom interfejsu.

**Komponente** se koriste kao temelj u ReactJS-u, ovo znači da se korisnički interfejs razbija na manje delove koji si ponovno upotrebljivi. Samim tim komponente služe da olakšaju organizaciju koda, održavanje i ponovnu upotrebu. Svaka komponenta sadrži svoje stanje i logiku i može se koristiti za izgradnju složenih korisničkih interfejsa.

**Virtualni DOM** je virtuelna reprezentacija **DOM**-a, u cilju poboljšanja performansi i optimizacije ažuriranja korisničkog interfejsa. Umesto direktnog ažuriranja DOM-a, React upoređuje virtuelni DOM sa stvarnim DOM-om i primenjuje samo promene, umesto da ažurira ceo DOM, što poboljšava strukturu komponenti.

**JSX** ili (*JavaScript XML*) je jezik za opisivanje korisničkog interfejsa. React uvodi JSX koji omogućava pisanje HTML-a direktno unutar JavaScript koda. JSX se kompajlira Reactovim *Babel* kompajlerom i konvertuje JSX u JavaScript.

**Jednosmeran protok podataka** omogućava prenos podataka hijerarhijski od roditeljskih ka komponentama potomcima u jednom pravcu. Ovo se naziva *One-way Data Binidng*. Ovo olakšava praćenje podataka i održava stanje aplikacije doslednim.

**React Router** se koristi za upravljanje rutama aplikacije. Ova biblioteka pruža programerima mogućnost definisanje ruta i njihovo povezivanje sa odgovarajućim komponentama. Umesto klasične zamene celih stranica, React aplikacije koriste router kako bi dinamički učitavale sadržaj komponente u zavisnosti od definisanih ruta. Ovaj pristup čini React tzv. “*single page application*” (aplikacije sa jednom stranicom), što znači da se korisnički interfejs ažurira bez potrebe za ponovnim učitavanjem cele stranice. Ova fleksibilnost omogućava korisnicima lepo iskustvo pregledanja, dok programerima pruža kontrolu i organizaciju putem definisanih ruta.

**Server side rendering** je paradigma koja se takođe može koristiti u React-u, što poboljšava performanse aplikacije i optimizaciju. Ova tehnika omogućava generisanje HTML-a na serverskoj strani prilikom inicijalnog zahteva korisnika.

**Hooks** su funkcionalnosti uvedene kasnije, tek od React 16.8 verzije. One omogućavaju upotrebu stanja i životnog ciklusa komponenata u funkcionalnim komponentama. Ovaj dodatak je veoma značajan za razvoj modernih aplikacija gde pruža efikasne i deklarativne načine rukovanja sa stanjem.

## Spring Boot radni okvir za logiku aplikacije

Spring Boot je radni okvir za izradu *Java* veb aplikacija koji olakšava brz razvoj uz jednostavnu konfiguraciju i povećanje produktivnosti programera. Razvijen od strane Rod Johnson-a, Spring Boot je deo Spring ekosistema i omogućava lakše kreiranje i organizaciju samostalnih Java aplikacija. Sledi dublje razumevanje ključnih karakteristika i komponenti Spring Boot-a:

* **Brza implementacija:** Spring Boot omogućava brz razvoj Java aplikacija zahvaljujući mogućnostima konfiguracije i automatskom podešavanju. Napravljen je tako da se programeri manje bave detaljima infrastrukture i više se fokusiraju na poslovnu logiku.
* **Laka Konfiguracija:** Spring Boot automatski konfiguriše veliki broj stvari, smanjujući potrebu za eksplicitnim podešavanjem. Konfiguracija se može obaviti kroz properties fajlove, YAML fajlove ili putem Java koda.
* **Ugrađen Tomcat Server:** Uz Spring Boot dolazi takođe i ugrađen web server *Tomcat.* Ovo omogućava da se aplikacija može pokrenuti kao samostalna Java aplikacija bez potrebe za eksternim web serverom.
* **Mikroservisna arhitektura:** Spring Boot je idealan za izgradnju mikroservisnih arhitektura. Programeri mogu lako razvijati i povezivati više nezavisnih servisa.
* **Automatsko Podešavanje:** Spring boot koristi mehanizam automatskog podešavanja (auto-configuration) koji konfiguriše aplikaciju na osnovu prisutnih biblioteka i zavisnosti u projektu. Ovo olakšava podešavanje bez potrebe za ručnim konfiguracijama.
* **Spring Ekosistem:** Spring Boot je deo šireg Spring ekosistema koji uključuje: *Spring Framework*, *Spring Data*, *Spring Security* i druge projekte. Ovo pruža širok spektar funkcionalnosti i alata za razvoj različitih vrsta aplikacija.
* **Upravljanje zavisnostima:** Spring Boot koristi ***Maven*** ili ***Gradle*** za upravljanje zavisnostima, što značajno pojednostavljuje dodavanje i ažuriranje zavisnosti u projektu.
* **Testiranje:** Takođe podstiče testiranje aplikacija, daje podršku za različite vrste testova (jedinične, integracione, end-to-end). Olakšava održavanje visokog nivoa kvaliteta.
* **Integracija:** Daje podršku za različite integracije, neke od njih uključuju baze podataka, sistem za poruke, OAuth, RESTful servise, itd.
* **Dokumentacija:** Generiše praktičnu dokumentaciju koja opisuje dostupne REST endpoint, konfiguracije i druge aspekte aplikacije. Ovo značajno olakšava održavanje i razumevanje projekta.

Spring Boot je u poslednjih nekoliko godina postao jedan od veoma popularnih izbora među Java programerima zbog svoje jednostavnosti, efikasnosti i mogućnosti brzog razvoja aplikacija. Ova tehnologija se veoma često koristi u razvoju raznovrsnih aplikacija, od jednostavnih web servisa do kompleksnih mikroservisnih arhitektura pa i servisno-orijentisanih sistema.

## MySQL baza podataka za fitness aplikaciju

*MySQL* je popularni open-source sistem za upravljanje relacionim bazama podataka (*RDBMS*) kao što je ranije opsiano, koji koristi *SQL (Structured Query Language)* jezik za skladištenje, definisanje i manipulaciju podacima. Ova baza podataka se koristi u projektu “Fitnes aplikacija za praćenje dnevnih aktivnosti treninga” za čuvanje i manipulaciju podacima. U nastavku sledi pregled ključnih karakteristika *MySQL* baza podatka sa *SQL* jezikom:

* **SQL jezik:** *SQL* jezik koji *MySQL* koristi omogućava jednostavno definisanje upita za čitanje, pisanje, ažuriranje i brisanje podatka, olakšavajući izgradnju kompleksnih upita za praćenje aktivnosti i napretka korisnika.
* **Relacioni Model**: *MySQL* je relacioni DBMS, što znači da su podaci u pazi podataka organizovani u tabelama sa definisanim relacijama između njih. Ova struktura omogućava efikasno skladištenje i pretragu podataka.
* **Performanse:** *MySQL* je poznat po brzim performansama, ovo je ključno za fitnes aplikaciju gde je potrebno efikasno čuvanje i prikupljanje podataka o treninzima i aktivnostima
* **Transakcije:** *MySQL* pruža podršku za transakcije, što znači da obezbeđuje doslednosti podataka. Ovo je ključno kada korisnici unose i ažuriraju podatke o svojim treninzima i dnevniku hrane.
* **Višekorisnički Pristup:** Više korisnika može pristupiti *MySQL* bazi istovremeno, pri čemu se upravljanje pristupom i bezbednošću može kontrolisati putem privilegija i autentifikacije.
* **Podrška za Transakcije:** *MySQL* podržava transakcije, što znači da se operacije nad bazom podataka mogu izvršavati kao jedna logička jedinica. Ako se transakcija neuspešno izvrši, sve promene se mogu poništiti (rollback), čime se čuva konzistentnost podataka.
* **Indeksi:** Definisanje indeksa je omogućeno u *MySQL* bazi nad tabelama radi brze i organizovane pretrage podataka. Indeksi poboljšavaju performanse upita, posebno u slučaju velikih baza podatka.
* **Backup i Oporavak:** *MySQL* pruža alate za kreiranje sigurnosnih kopija baze podataka i obnovu podataka u slučaju gubitka ili oštećenja.
* **Skalabilnost:** *MySQL* se može skalirati kako bi se nosio sa rastućim brojem podataka i zahteva. Ovo se postiže horizontalnim ili vertikalnim skaliranjem, kao i korišćenjem klastera i replikacija.
* **Open Source i Community Support:** *MySQL* je open-source projekat sa aktivnom zajednicom programera i korisnika. To znači da postoje brojni resursi, forumi i dokumentacija dostupni za podršku i rešavanje problema.

MySQL baza podataka igra ključnu ulogu u ovom projektu, jer omogućava skladištenje i upravljanje podacima vezanim za vežbe, treninge, aktivnosti, hranu, obroke, nutricionizam, postignuća.

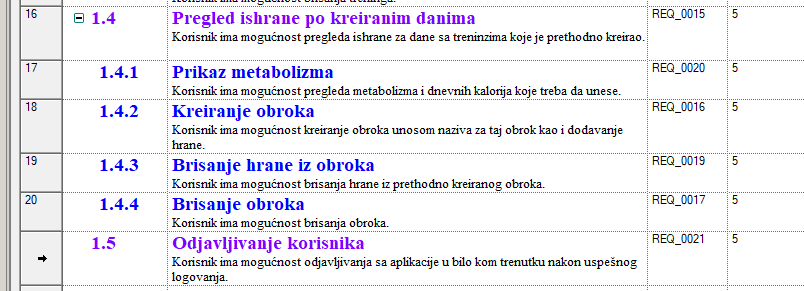
# Zahtevi sistema

## Funkcionalni zahtevi

Funkcionalni zahtevi definišu kako bi softverski sistem trebalo da se ponaša u određenim uslovima. Opisuju zahteve kako sistem treba da se ponaša i pruža funkcionalnosti korisnicima. To uključuje mogućnost unosa informacija o korisnicima, treninzima i ishrane.



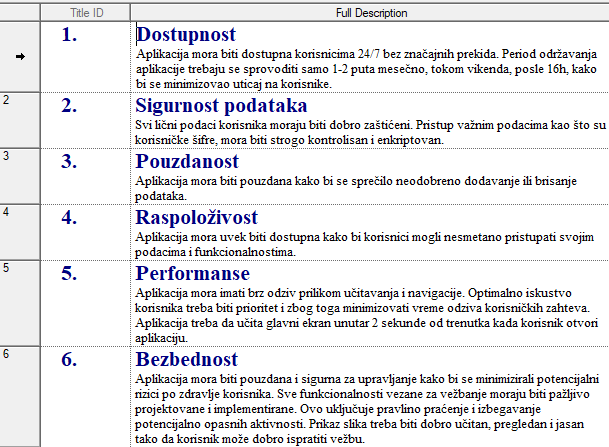
Slika 4.1.1 - Funkcionalni zahtevi slika 1 [izvor: autor]



Slika 4.1.2 - Funkcionalni zahtevi slika 2 [izvor: autor]

## Nefunkcionalni zahtevi

Nefunkcionalni zahtevi definišu karakteristike sistema koje ne definišu direktno njegove funkcionalnosti. Ovi zahtevi se više odnose na sistemske performanse, bezbednost, dostupnost, integritet i druge aspekte. Važni su za održavanje kvaliteta, pouzdanosti sistema i zadovoljenje korisničkih potreba.



Slika 4.2.1 - Nefunkcionalni zahtevi [izvor: autor]

# Implementacija

Deo “Implementacija” fokusira se na realizaciju dizajna i funkcionalnosti fitness aplikacije. Ova sekcija pruža dublji uvid u tehničke detalje procesa razvoja, alate koji će se koristiti i postupke implementacije kako bi se postigao funkcionalan i održiv sistem.

PowerDesigner je alat za modeliranje podataka i arhitekture koji pomaže organizacijama u planiranju, dizajniranju i implementaciji informacionih sistema. Ovaj program se koristi za podršku različitim fazama razvoja softvera uključujući analizu, dizajn i implementaciju. Ovaj projekat će, korišćenjem PowerDesignera, modelovati različite aspekte informacione arhitekture. Preko njega možemo detaljno definisati entitete, relacije i ograničenja baza podataka, pružajući jasnu sliku o organizaciji podataka unutar sistema. Takođe koristiće se za modelovanje procesa, slučajeve korišćenja, arhitekture, sistem sekvencijalnih dijagrama, itd.

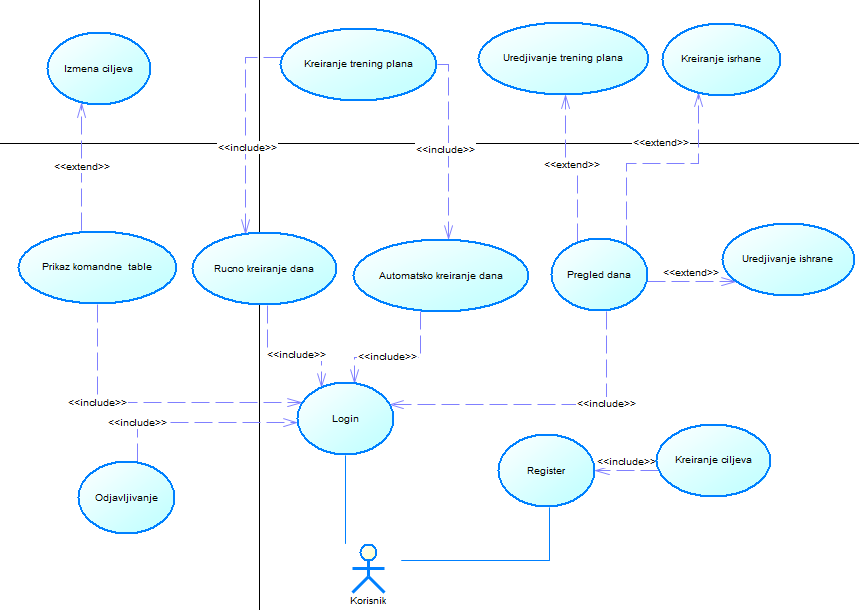
Arhitektura projekta za ovu fitness aplikaciju pažljivo će razmotriti organizaciju komponenti sistema, kao i način njihove međusobne komunikacije. U analizi arhitekture, posebno ćemo istražiti kako su Spring Boot, ReactJS i MySQL baza podataka implementirani kako bi se stvorio sveobuhvatan i performantan sistem.

Koristimo Spring Boot za izradu backend-a kako bi pružio efikasnu osnovu za brz i bezbedan razvoj, dok ReactJS omogućava izgradnju dinamičnog i “user-friendly” interfejsa.

Struktura projekta će biti pažljivo opisana, uključujući organizaciju fajlova, direktorijuma i modula unutar aplikacije. Na primer, Spring Boot moduli će biti organizovani prema slojevima kontrolera, servisa i repozitorijumima, obezbeđujući jasnu separaciju odgovornosti. Ovo takođe možemo i nazvati servisno-orijentisanim stilom. Frontend deo biće strukturiran u komponente i kontejnere kako bi se postigla reaktivnost i lako održavanje.

Odluke vezane za arhitekturu, imajući u vidu performanse i skalabilnost biće naglašene, uzimajući u obzir kako Spring Boot, ReactJS i MySQL zajedno doprinose celokupnom doživljaju korisnika i funkcionalnosti fitness aplikacije. Sama ta analiza arhitekture omogućiće nam da bolje razumemo kako arhitektonske odluke utiču na efikasnost i održivost ove fitness aplikacije.

## Slučajevi korišćenja



Slika 5.1.1 - Prikaz dijagrama slučajeva korišćenja [izvor: autor]

| Slučaj korišćenja : | Login | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da pristupi aplikaciji. | | |
| Preduslovi : | Korisnik ima registrovan nalog i ima pristup korisničkoj aplikaciji. | | |
| Normalni tok : | 1. Korisnik otvara aplikaciju u pregledaču. 2. Korisnik na login stranici unosi svoje korisničko ime i šifru. 3. Korisnik se uspešno loguje na aplikaciju i pristupa joj. [EX1] | | |
| Izuzeci : | [EX1]: Ukoliko korisnik unese pogrešnu kombinaciju username-a i šifre, dobija odgovarajuće obaveštenje o lošim kredicijalima. | | |

| Slučaj korišćenja : | Register | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da postane član aplikacije. | | |
| Preduslovi : | Korisnik nema prethodno kreiran nalog, i želi postati novi član web aplikacije. | | |
| Normalni tok : | 1. Korisnik otvara aplikaciju u pregledaču. 2. Korisnik klikće na dugme “Or become a member today!”. 3. Korisnik unosi svoje kredicijale u sva polja koja su prikazana na ekranu (username, email, password, confirm password, first name, last name, height, weight, age, gender). 4. Korisnik klikće na dugme “Sign up”. [EX1] 5. Uspešno se otvara novi dialog gde korisnik treba da kreira svoj cilj. | | |
| Izuzeci : | [EX1]: Ukoliko korisnik ne unese sve podatke koje se traže, dugme će nestati i prikazaće se greška na ekranu. | | |

| Slučaj korišćenja : | Kreiranje ciljeva | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da postane član aplikacije. | | |
| Preduslovi : | Korisnik je uneo sve ispravne podatke u polja, prilikom registracije i kliknuo je na dugme “Sign up”. | | |
| Normalni tok : | 1. Otvara se dialog gde korisnik mora da unese svoj cilj (goal) 2. Korisnik popunjava sva polja koja su prikazana u dialogu (ciljana težina, tip, koliko puta nedeljno bi trenirao) 3. Korisnik klikće na dugme “Save”. [EX1] 4. Korisnik se uspešno registruje na sajtu. | | |
| Izuzeci : | [EX1]: Ukoliko korisnik unese pogrešne podatke, kao što su broj dana kojim bi trenirao (veći od 6 a manji od 1), izbaciće grešku i neće kreirati novog korisnika. | | |

| Slučaj korišćenja : | Prikaz kontrolne table | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik je pristupio aplikaciji i želi da vidi sumirane podatke. | | |
| Preduslovi : | Korisnik ima pristup korisničkoj aplikaciji. | | |
| Normalni tok : | 1. Korisnik je kliknuo da se uloguje na aplikaciju i navigira do sekcije “Dashboard”. 2. Aplikacija pokazuje sumirane podatke kao što su ishrana i uneta hrana 3. Aplikacija prikazuje korisniku ciljeve koje korisnik može izmeniti ako želi. 4. Aplikacija prikazuje korisniku izračunatu masu na osnovu podatka koje je uneo (visina, težina, broj godina) | | |
| Izuzeci : | / | | |

| Slučaj korišćenja : | Izmena ciljeva | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da izmeni ciljeve koje je postavio prilikom registracije. | | |
| Preduslovi : | Korisnik ima pristup korisničkoj aplikaciji i ulogovan je na svoj nalog. | | |
| Normalni tok : | 1. Korisnik je klinuo da se uloguje na aplikaciju i navigira do sekcije “Dashboard”. 2. Aplikacija prikazuje korisniku sekciju sa ciljevima koje je uneo prilikom registracije. 3. Korisnik popunjava polja koja želi da izmeni:    1. Ciljana težina    2. Koliko puta bi trenirao nedeljno.    3. Konačni cilj 4. Korisnik klikće na dugme izmeni [EX1] 5. Aplikacija obaveštava korisnika da je uspešno izmenio cilj. | | |
| Izuzeci : | [EX1]: Korisnik prilikom unošenja dnevnih treninga unese broj koji je ispod 1 ili iznad 6. Aplikacija obaveštava korisnika da je to neispravno. | | |

| Slučaj korišćenja : | Ručno kreiranje dana | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da ručno kreira aktivan dan sa planom treninga. | | |
| Preduslovi : | Korisnik ima pristup aplikaciji. | | |
| Normalni tok : | 1. Korisnik navigira na stranicu “Exercise”. 2. Korisnik klikće na dugme “Create Manual Plan”. 3. Otvara se dijalog sa poljima za kreiranje dana sa trening planom. 4. Korisnik popunjava polja:    1. Kategorija    2. Lista Vežbi    3. Datum početka vežbe 5. Korisnik klikće na dugme “Create”. [EX1] 6. Aplikacija obaveštava korisnika da je uspešno kreirao aktivan dan sa planom treninga. | | |
| Izuzeci : | [EX1]: Ukoliko korisnik ne popuni odgovarajuća polja izbaciće grešku da nije uspešno kreirao plan. | | |

| Slučaj korišćenja : | Automatsko kreiranje dana | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da preko sistema, automatski kreira dane sa treninzima. | | |
| Preduslovi : | Korisnik ima pristup aplikaciji. | | |
| Normalni tok : | 1. Korisnik navigira na stranicu “Exercise”. 2. Korisnik klikće na dugme “Auto Generate Plan”. 3. Aplikacija obaveštava korisnika da je uspešno kreirao aktivane danove u zavisnosti od koliko dana nedeljno je uneo za trening. | | |
| Izuzeci : | / | | |

| Slučaj korišćenja : | Pregled dana | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da pregleda dane s treninzima. | | |
| Preduslovi : | Korisnik ima pristup aplikaciji i prehtodno je kreirao ručno ili automatski danove sa planovima. | | |
| Normalni tok : | 1. Korisnik navigira do sekcije “Exercise”. 2. Aplikacija prikazuje sve vežbe sortiranim po danima sa tim da se trenutni dan uvek prvi prikazuje na ekranu. | | |
| Izuzeci : | / | | |

| Slučaj korišćenja : | Uredjivanje trening plana | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da izmeni kreirane dane ili da ih obriše. | | |
| Preduslovi : | Korisnik ima pristup aplikaciji i prehtodno je kreirao ručno ili automatski danove sa planovima. | | |
| Normalni tok : | 1. Korisnik navigira do sekcije “Exercise”. 2. Aplikacija prikazuje sve vežbe sortiranim po danima sa tim da se trenutni dan uvek prvi prikazuje na ekranu. 3. Korisnik ima opciju za ažuriranje treninga ili brisanje treninga. 4. Korisnik klikće na dugme “Finish Exercise” ili “Delete Exercise”. 5. Ako je izabrana opcija brisanja trening se briše. 6. Ako je izabrana opcija “Finish”, trening se ažurira. | | |
| Izuzeci : | / | | |

| Slučaj korišćenja : | Kreiranje ishrane | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da vodi računa o ishrani i da kreira obroke | | |
| Preduslovi : | Korisnik ima pristup aplikaciji i prehtodno je kreirao ručno ili automatski danove sa planovima. | | |
| Normalni tok : | 1. Korisnik navigira do sekcije “Food”. 2. Aplikacija prikazuje sekciju sa kalorijama koliko korisnik treba da unese po danu. 3. Korisnik klikće na dugme da kreira novi ili doda postojeći obrok. 4. Korisniku izlazi dialog sa poljima koje treba da popuni da bi kreirao obrok.    1. Naziv obroka    2. Lista hrane 5. Korisnik klikće na dugme i kreira novi obrok. 6. Aplikacija obaveštava korisnika da je uspešno kreirao obrok. | | |
| Izuzeci : | / | | |

| Slučaj korišćenja : | Uredjivanje ishrane | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da izmeni postojeći obrok | | |
| Preduslovi : | Korisnik ima pristup aplikaciji i prehtodno je kreirao ručno ili automatski danove sa planovima. | | |
| Normalni tok : | 1. Korisnik navigira do sekcije “Food”. 2. Korisnik klikće na dugme da izmeni postojeći obrok ili da obriše obrok. 3. Korisniku izlazi dialog gde može da doda ili da obriše dodatu hranu. 4. Korisnik klikće na dugme i ažirira novi obrok. 5. Aplikacija obaveštava korisnika da je uspešno kreirao obrok. 6. Ako je izabrana opcija brisanja obrok se briše. 7. Ako je izabrana opcija ažuriranja, obrok se ažurira. | | |
| Izuzeci : | / | | |

| Slučaj korišćenja : | Odjavljivanje | | |
| --- | --- | --- | --- |
| Pokretač : | Korisnik | | |
| Opis : | Korisnik želi da se izloguje. | | |
| Preduslovi : | Korisnik je ulogovan na aplikaciji. | | |
| Normalni tok : | 1. Korisnik klikće na dugme “Logout”. 2. Aplikacija prebacuje korisnika sa glavne strane na Login stranu. | | |
| Izuzeci : | / | | |

## Arhitektura projekta

*REST* (Representional State Transfer) je arhitektonski stil softvera koji vodi dizajn i razvoj arhitekture za kreiranje različitih veb servisa. Cilj *REST-a* je da olakša komunikaciju između klijenta i servera putem standardizovanog protokola. Ovaj arhitektonski stil naglašava skalabilnost interakcija između komponenata, uniformne interfejse, nezavisno razmeštanje komponenata i stvaranje slojevite arhitekture kako bi se omogućilo keširanje komponenata radi smanjenja čekanja koju korisnici primete, primenjivanje bezbednosti i inkapsulacija starih sistema.

“*RESTful*” aplikacije su aplikacije koje poštuju principe i prakse definisane *REST* arhitekturom, iako se ovaj termin češće povezuje sa dizajnom *API*-ja baziranih na HTTP-u. Znači aplikacija se smatra “*RESTful*” ako pravilno primenjuje *REST* principe, uključujući upotrebu resursa, identifikaciju resursa putem URI-ja, korišćenje standardnih HTTP metoda i slanje podataka u standardnim formatima kao što su JSON (*JavaScript Object Notation*) ili XML (*eXtensible Markup Language*).

*REST* *API* (Represantional State Transfer Application Programming Interface) je interfejs koji omogućava komunikaciju između različitih softverskih sistema putem HTTP metoda. Ovaj interfejs se oslanja na princpipe *REST* arhitekture kako bi omogućio jednostavnu, efikasnu i skalabilnu komunikaciju između klijenta i servera.

Ključne karakteristik *REST API*-ja uključuju:

* **Resursi (Resources)**: *REST API* se bazira na resursima, ti resursi predstavljaju entitete i objekte sa određenim stanjem ili podacima. Svaki resurs ima svoje jedinstveno mesto u hijerarhiji URI-ja.
* **HTTP Metode (*HTTP Methods*):** *REST API* koristi standardne HTTP metode za rad sa resursima. Najčešće korišćene metode su:
  + **GET:** Uzima informacije o resursu.
  + **POST:** Kreira novi resurs.
  + **PUT:** Ažurira postojeći resurs.
  + **DELETE:** Briše resurs.
* **Reprezentovano Stanje (Representational State):** Resursi se predstavljaju u određenom stanju preko formata, kao što su *JSON* ili *XML*. Klijentske aplikacije koriste ove formate kako bi prikazivale ili ažurirale stanje resursa.
* **Bez stanja (Stateless):** *REST API* je protokol bez stanja. Ovo znači da svaki zahtev od klijenta serveru sadrži sve potrebne informacije za obradu tog zahteva. Server ne čuva stanje klijenta između zahteva.
* **Uniformni Interfejs:** *REST API* koristi uniformni interfejs koji olakšava razumevanje i upotrebu API-ja. To uključuje korišćenje standardnih HTTP metoda, jednostavnih URI-ja, i jasno definisane operacije za svaki resurs.
* **Jedinstveni URI:** Svaki resurs ima jedinstveni URI koji ga identifikuje. Recimo URI za dobivanje informacija o korisniku po id-ju može izgledati kao “/users/id/123”.

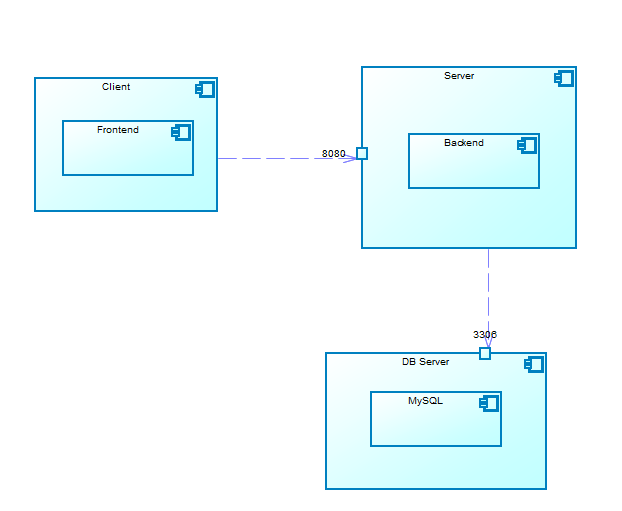
*MVC* (Model-View-Controller) je arhitekturni šablon u softverskom dizajnu koji se često koristi za organizaciju komponenti u aplikacijama, posebno onima koje imaju korisnički interfejs. Ovaj šablon se sastoji od tri osnovne komponente: *Model*, *View* i *Controller*.

* **Model:** Model predstavlja strukturu podataka i biznis logiku aplikacije. Ovaj sloj aplikacije odgovoran je za manipulaciju podacima, pružanje metoda za pristup ili upravljanje i ažuriranje podataka. Ova komponenta se ne bavi prikazivanjem podataka ili interakcijom sa korisnikom, već samom logikom podataka. Na primer, u veb aplikaciji, model bi mogao da predstavlja bazu podataka i kod za pristup toj bazi ili entitetima.
* **View:** Pogled je odgovoran za prikazivanje podataka korisniku. Ovo je korisnički interfejs koji korisnici vide i interaguju sa njim. Ovaj sloj predstavlja grafičke element i prezentacije podataka koja ne sadrži nikakvu poslovnu logiku, već samo prikazuje informacije koje dobija od Modela. View se koristi za predstavljanje informacija na korisničkom ekranu na način koji je razumljiv korisnicima.
* **Controller:** Kontroler je posrednik između modela i pogleda. On obrađuje korisničke zahteve, izvršava odgovarajuće akcije i ažurira Model ili View na osnovu korisničkih interakcija (na primer, klik na dugme). Služi da prosleđuje te zahteve modelu ili pogledu, zavisno od situacije. Controller reaguje na HTTP zahteve korisnika i ažurira model ili promenjene podatke šalje pogledu kako bi se osvežio prikaz.

MVC arhitektura promoviše razdvajanje odgovornosti između komponenata aplikacije, što je čini modularnom, skalabilnom i lakšom za održavanje. Ovo omogućava timovima različitih stručnosti da rade zajedno na projektu, gde dizajneri mogu da rade na pogledu, programeri na kontrolerima i modelima, a svi zajedno da stvaraju funkcionalnu aplikaciju.

Arhitektura projekta predstavlja organizaciju, strukturu i način na koji su komponente sistema raspoređene i povezane kako bi se postigao cilj projekta. Ova struktura obuhvata kako će sistem biti dizajniran, implementiran i održavan tokom vremena. Arhitektura ima ključnu ulogu u obezbeđivanju usklađenosti sistema sa zahtevima i postizanju zadovoljavajućih performansi, skalabilnosti i održivosti. Ključni elementi arhitekture projekta obuhvataju:

* **Klijentska aplikacija:** Klijentska aplikacija se sastoji od ReactJs aplikacije, ovo predstavlja korisnički interfejs. Pomoću nje korisnici vide i interaguju sa podacima.
* **Serverska aplikacija:** Serverska aplikacija se sastoji od Spring Boot aplikacije. Ona je odgovorna za obradu zahteva koje šalju klijentske aplikacije. Sadrži poslovnu logiku i preko nje se pristupa bazi podataka ili drugim spoljnim resursima.
* **Baza podataka:** Baza podataka je skladište podataka koje čuva informacije koje aplikacija koristi. U slučaju fitness aplikacije, ovo je relaciona baza podataka MySQL.

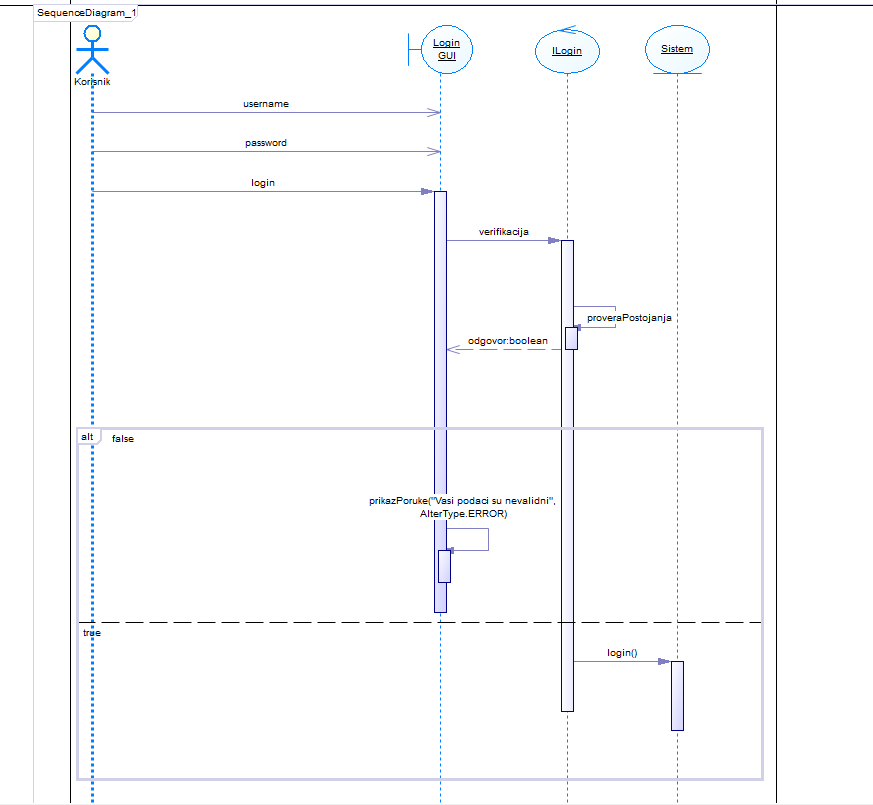


Slika 5.2.1 - Klijent-server arhitektura [izvor: autor]

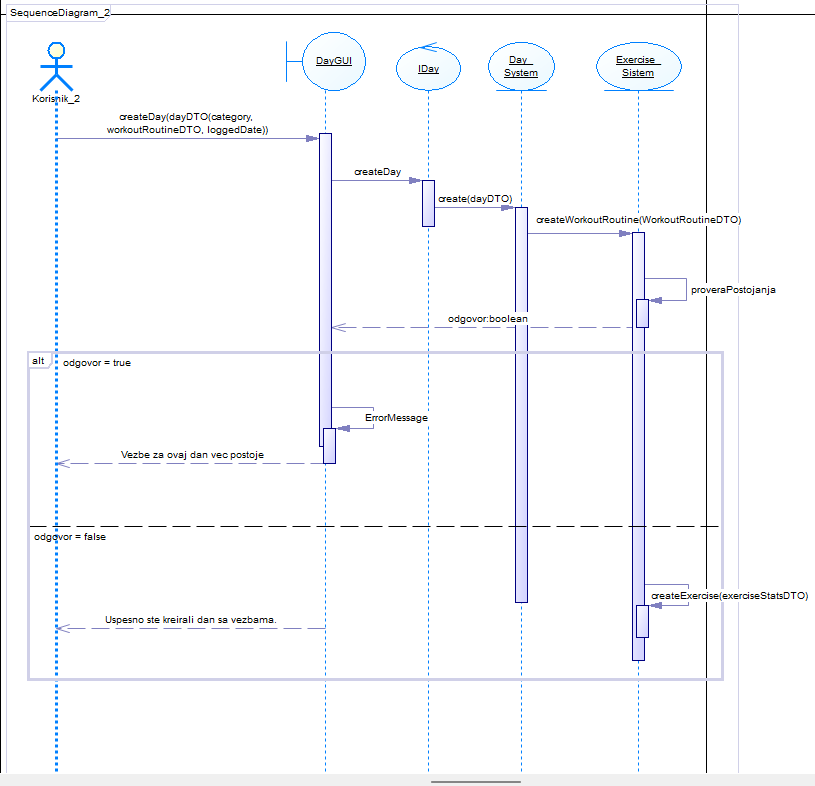
## Sistem sekvencijalnih dijagrama

*Sistem sekvencijalnih dijagrama* (takođe poznati kao Dijagrami Sekvenci) predstavlja vrstu *UML* (Unified Modeling Language) dijagrama koji modeluje redosled interakcija između različitih entiteta tokom određenog vremenskog perioda. Ovakvi dijagrami pomažu za vizualizaciju toka događaja i komunikacije između entiteta tokom izvršavanja određenog scenarija, u kontekstu određenog slučaja korišćenja. Crtanje sekvencijalnih dijagrama za svaki slučaj korišćenja nije uvek obavezan, ali može biti koristan u određenim slučajevima. Odluka o tome da li je potrebno nacrtati sekvencijalni dijagram za određeni slučaj korišćenja zavisi od potreba projekta, složenosti interakcija i preferencija tima.

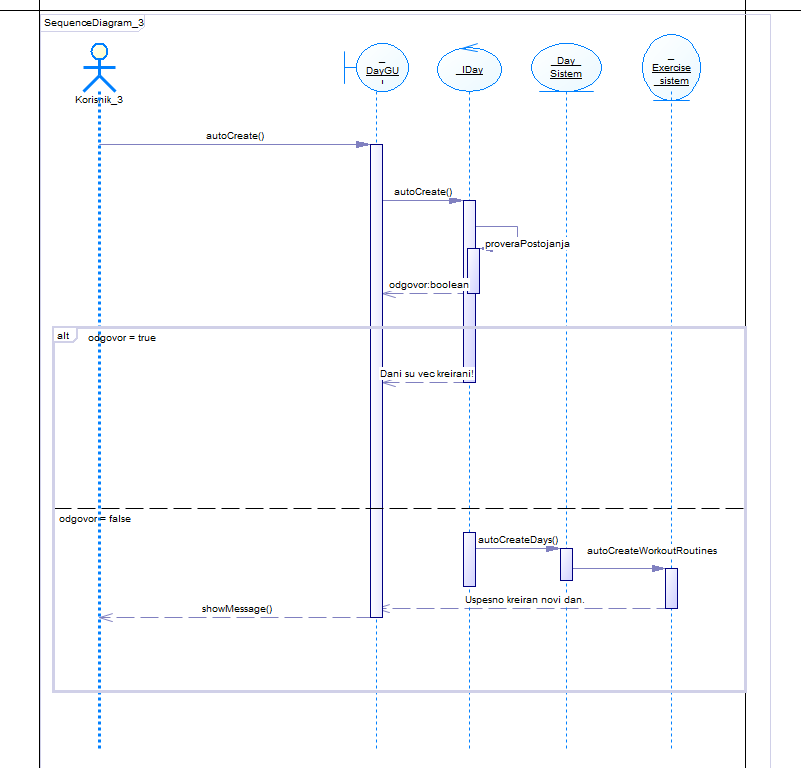
Radi primera možemo videti dijagrame sekvenci za slučajeve gde je potrebno kreirati dane sa vežbama automatski ili ručno, kao i logovanje:



Slika 5.3.1 - Dijagram sekvenci za login [izvor: autor]



Slika 5.3.2 - Dijagram sekvenci za ručno kreiranje dana [izvor: autor]



Slika 5.3.3 - Dijagram sekvenci za automatsko kreiranje dana [izvor: autor]

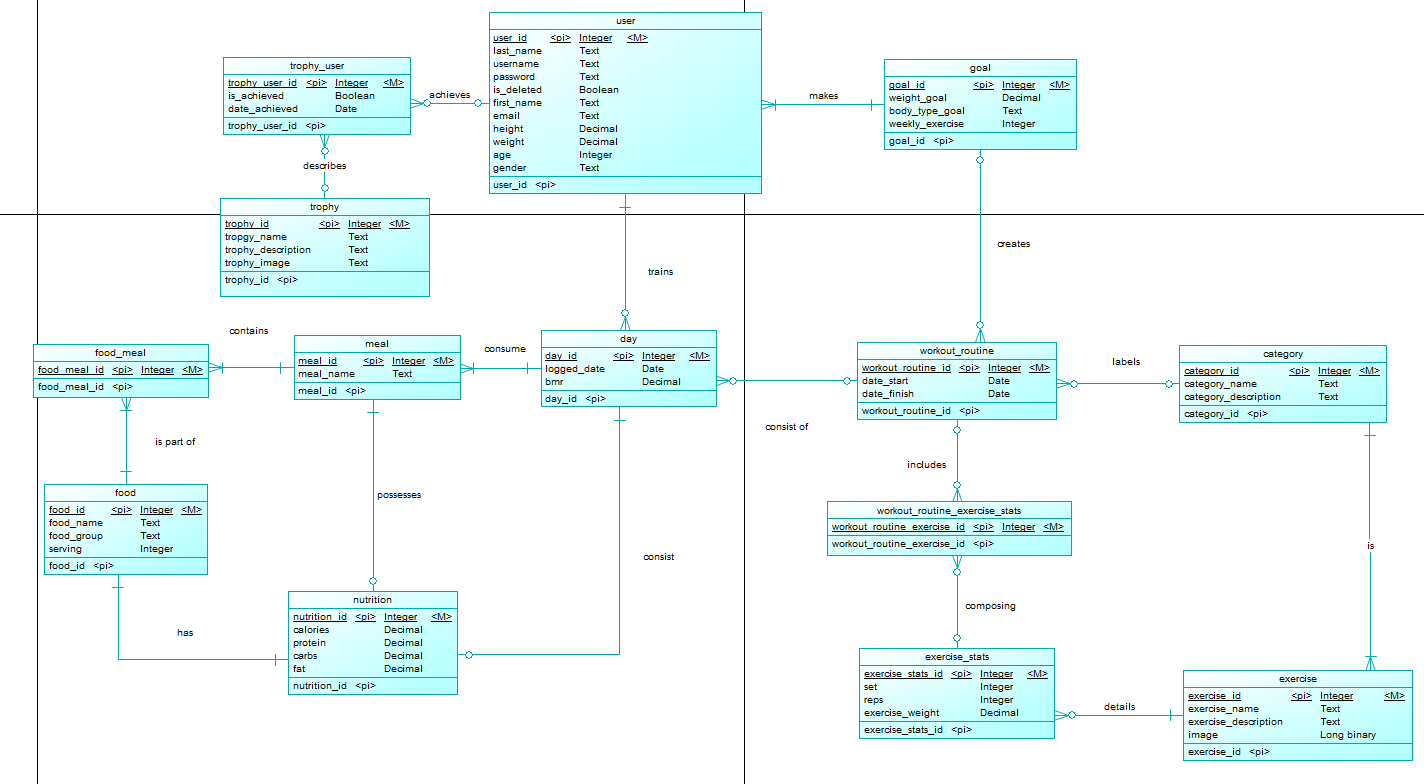
Kao što možemo videti na slikama dijagrami sekvencija služe da prikažu redosled, ili flow kako se kreiraju ručno ili automatski dani ili kako se korisnik loguje. Ovi dijagrami igraju ključnu ulogu u ilustraciji toka informacija i međusobne komunikacije između različitih entiteta sistema tokom izvršavanja specifičnih scenarija. Odavde možemo zaključiti kako sistem reaguje i šta korisniku vraća.

## Baza podataka

U ovom segmentu možemo videti kako je baza vizuelno dizajnirana korišćenjem PowerDesigner-a. Koristimo tri tipa dijagrama prilikom dizajniranja:

* **Conceptual (Konceptualni Dijagram):**

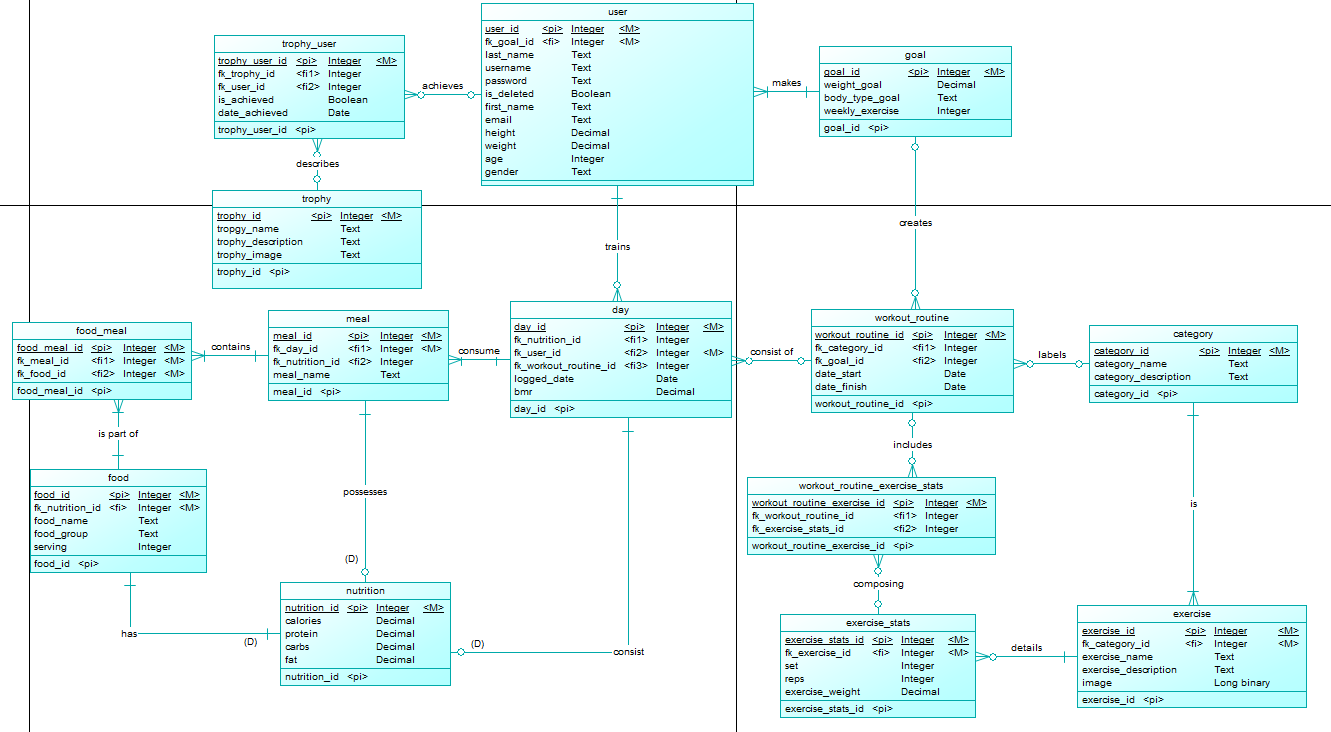
Konceptualni dijagram fokusira se na prikazivanje visokog nivoa apstrakcije i konceptualnih veza između entiteta. Fokus je na “šta” baza podataka treba da čuva, bez ulaženja u tehničke detalje o tome “kako” će nešto biti sačuvano. Ovaj dijagram često koristi entitetsko-relacioni model (*ERM*) kako bi se ilustrovali entiteti, njihovi atributi i veze između njih, bez obzira na tehničke detalje implementacije.



Slika 5.4.1 - Konceptualni model [izvor: autor]

* **Logical (Logički Dijagram):**

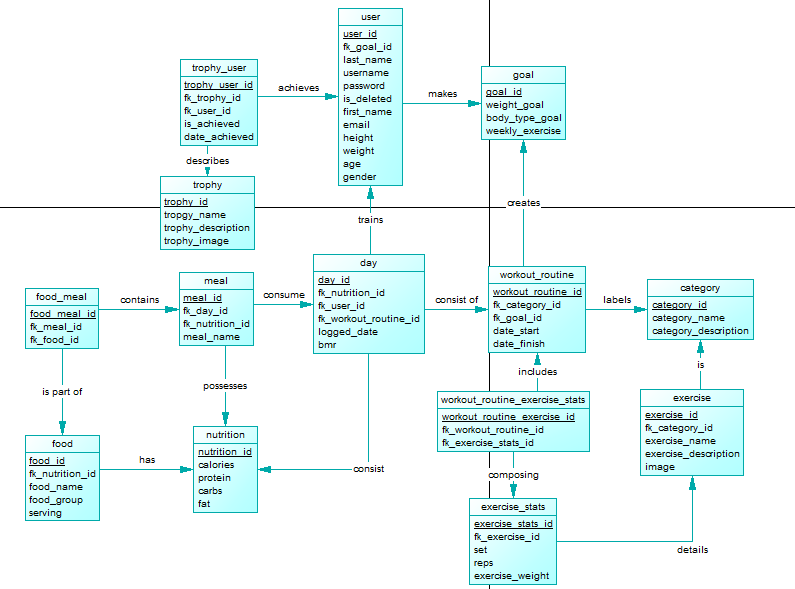
Logički dijagram prelazi na detaljniji nivo i odnosi se na organizaciju podataka u samoj bazi. Prikazivanje entiteta, atributa i veza se izvršava pomoću njega korišćenjem *Entity-Relationship Diagram-a* (*ERD*) dodajući detalje o ključevima (primarnim i stranim), indeksima i ograničenjima. Pomoću ovog dijagrama se prikazuje struktura baze podataka i kako se entiteti povezuju, ali još uvek ne ulazi u tehničke detalje vezane za implementaciju.



Slika 5.4.2 - Logički model [izvor: autor]

* **Physical (Fizički Dijagram):**

Ovo je najdetaljniji dijagram od sva tri i usredsređen je na tehničke aspekte implementacije. Fizički dijagram prikazuje konkretnu strukturu baze podataka, uključujući tabele, kolone, tipove podataka, indekse, ključeve i druge tehničke detalje. On direktno odražava podatke i kako će oni biti organizovani i sačuvani na nivou baze podataka, kao i specifičnosti kao što su tipovi podataka i ograničenja.



Slika 5.4.3 - Fizički model [izvor: autor]

## Struktura projekta

Struktura projekta igra ključnu ulogu u upravljanju razvojem aplikacije i organizaciji projekta. Pomoću analize strukture, dobijamo dublji uvid o aplikaciji i njenoj arhitekturi. U ovom delu, detaljno ćemo istražiti strukturu projekta za ReactJS aplikaciju i SpringBoot aplikaciju u ovom Fitnes projektu. Svaka tehnologija ima svoje jedinstvene karakteristike prilikom organizacije koda pomoću koga ćemo bolje razumeti kako su ovi projekti organizovani i kako komponente međusobno komuniciraju.

Što se tiče strukture Spring Boot projekta, možemo videti njegovu strukturu na sledećoj slici:



Slika 5.5.1 - Prikaz strukture backend projekta [izvor: autor]

Direktorijum **src/** služi kao izvorni kod jedne spring boot aplikacije. Organizovan je tako da se u njemu nalaze svi ostali direktorijumi koji se koriste za rad. U **main/** direktorijumu obično se nalaze sve glavne podklase koje služe za razvoj aplikacije. U **java/** direktorijumu se smešta izvorni Java kod aplikacije. Klase i paketi koje se kreiraju trebalo bi da budu organizovani tu. Direktorijum **org.alx.fitnessapp/** je paket gde **org.alx** predstavlja osnovni paket aplikacije. Aplikaciju dalje možemo podeliti u sledeće pakete:

* **config:** Ovde se nalaze klase koje se koriste za konfiguraciju aplikacije. (Spring Bean-ova, itd.)
* **controller:** Ovde se nalaze kontroleri koji obrađuju HTTP zahteve.
* **converter:** Ovde se nalaze konverteri koji prevode entitete u DTO objekte i šalju ili primaju sa frontenda.
* **exception:** Ovde se smeštaju klase koje se bave izuzecima.
* **model:** Ovaj paket sadrži klase koje predstavljaju **entitete** i **DTO**-ove.
* **repository:** Ovde se smeštaju interfejsi koji se koriste za komunikaciju sa bazom.
* **security:** Ovaj paket sadrži klase vezane za sigurnosti aplikacije.
* **service:** Ovde se smeštaju klase i interfejsi koji sadrže poslovnu logiku aplikacije.
* **util:** Ovde se nalaze pomoćne klase koje se mogu koristiti širom aplikacije.
* **validation:** Ovde se smeštaju klase i komponente koje se koriste za validaciju podataka.

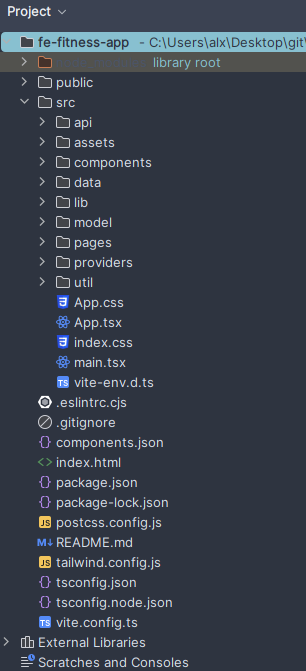
Direktorijum *FitnessAppApplication* je glavna (main) klasa koja ima ključnu ulogu u pokretanju samostalne Spring Boot aplikacije. Ova klasa je centralna tačka za pokretanje Spring Boot kontejnera i pokretanje web servera koji služi aplikaciji. On sadžri metodu **main** koja se izvršava kada se pokrene Java aplikacija i ona poziva **SpringApplication.run()** statičku metodu.

Pored ovoga postoji i direktorijum **resources** koja sadrži resurse poput konfiguracija ili statičkih fajlova i šablona. U njoj se nalazi **application.properties** fajl koji služi za konfiguraciju aplikacije kao i konekciju sa bazom podataka. **Test** direktorijum označava deo aplikacije koji je namenjen testiranju koda i funkcionalnosti.

**Target** direktorijum predstavlja mesto gde se nalaze generisani artifakti za vreme izvršavanja Maven ili Gradle build-a. Ovaj direktorijum sadrži sve što je potrebno za pokretanje aplikacije, uključujući i kompajlirani Java bytecode, zavisnosti (JAR fajlove), resurse i druge artifakte.

Postoje i drugi fajlovi kao što su **mvnw** kao i **mvnw.cmd** i **pom.xml** koji služe kao Maven Wrapper skripte za izvršavanje Mavena, za konfiguraciju Mavena i za lakše instaliranje zavisnosti. Pored njih su tu i fajlovi vezani za *Git* kao što su **.gitignore** i fajl **README.md** i oni služe da ograniče fajlove i direktorijume koji će se slati direktno na git a koje će git da ignoriše kao i da opišu nešto detaljnije o projektu drugim programerima koji se prvi put susreću sa projektom. Ovo opisivanje projekta se radi u *markup* jeziku.

Što se tiče ReactJS aplikacije, na slici možemo videti kako je ona strukturisana:



Slika 5.5.2 - Prikaz strukture frontend projekta [izvor: autor]

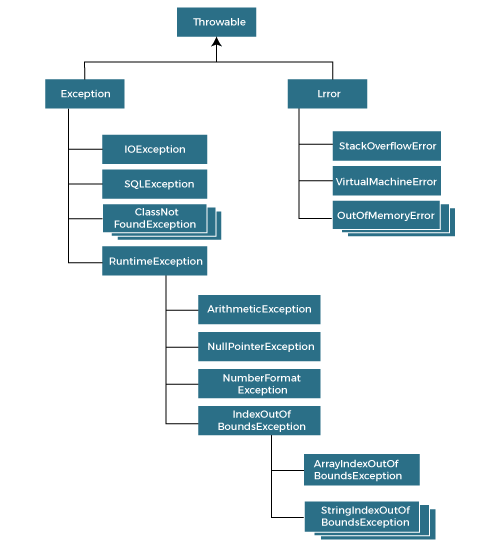
Možemo podeliti strukturu projekta na nekoliko sledećih delova:

* **Korenski direktorijum**
  + **src**: Ovaj direktorijum je ključan i sadrži izvorni kod *React* aplikacije.
  + **public**: Ovde se nalaze statički resursi, kao što su HTML fajlovi, favicon, slike i drugi resursi koje aplikacija koristi. Ovaj direktorijum služi kao izlazni direktorijum za finalnu verziju aplikacije.
  + **node\_modules**: U ovom direktorijumu se smeštaju zavisnosti i biblioteke koje koristi ovaj projekat. Ovaj direktorijum se često generiše automatski prilikom instalacije zavisnosti putem npm-a. (npm install)
  + **package.json**: Ovaj fajl sadrži informacije o projektu, zavisnostima, skriptama za pokretanje, verzije, build i testiranje, kao i druge konfiguracije.
  + **package-lock.json**: Ovaj fajl se koristi za tačno definisanje verzija zavisnosti i osigurava doslednost između različitih instalacija.
  + **assets**: Ovo je direktorijum koji služi za čuvanje lokalnih slika koji će se koristiti za prikazivanje na web aplikaciji.
* **Komponente i Logika**
  + **components**: Ovaj direktorijum služi za smeštanje ponovno upotrebljivih *React* komponenti koje se koriste u različitim delovima aplikacije. Komponente bi trebalo da budu dobro organizovane i ponovno upotrebljive radi olakšavanja održavanja i smanjenja redudantnosti koda.
  + **pages**: Slično kao components, pages direktorijum služi za kreiranje komponenti ali onih koje predstavljaju prave stranice koje korisnici vide, i sa kojima interaguju. Razlikuju se od komponenti iz razloga jer su jedinstvene i čini ih skup različitih komponenti.
  + **lib**: Sadrži utility funkcije koje se koriste širom aplikacije.
* **API**
  + **api**: Služi za implementiranje *API* ruta, korišćenjem *redaxios-a*, u njemu se nalaze različite rute koje gađaju različite *endpoint-e*. Redaxios je biblioteka koja pruža funkcionalnosti za slanje HTTP zahteva koristeći moderni **fetch** API. Ova biblioteka dezajnirana je da bude minimalistična, ali efikasna alterantiva drugim bibliotekama za rad sa HTTP-om.
* **Ostale datoteke**
  + **tsconfig.json**: Ovaj fajl se koristi za konfigurisanje *typescript*-a, ova aplikacija radi korišćenjem typescripta nego klasičnog javascripta.
  + **tsconfig.node.json:** Koristi se za podešavanja opcija *typescript-a*.
  + **postcss.config.js**: Koristi se za konfiguraciju PostCSS, koji je alat za obradu CSS-a.
  + **tailwind.config.js**: Koristi se za konfigurisanjeTailwind CSS okvira, koji pruža set korisnih klasa za brzo stvaranje stilova u HTML-u.
  + **vite.config.ts:** Koristi se za konfigurisanje Vite alata, koji je brz i moderni sistem za razvoj React aplikacija
  + .**gitignore**: Datoteke za git ignorisanje kako bismo zaštitili osetljive informacije i nepotrebne datoteke prilikom verzionisanja.

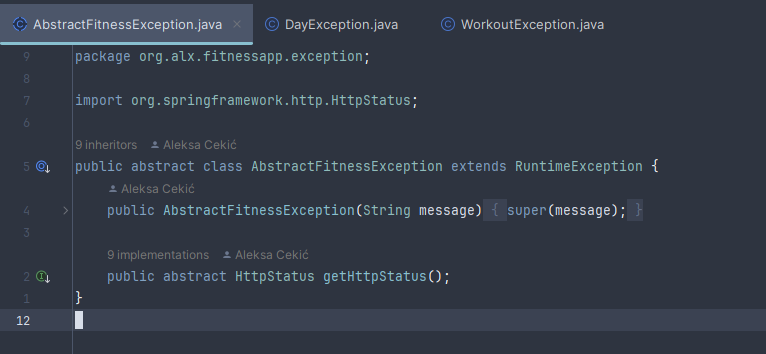
## Obrada izuzetaka

Obrada izuzetaka predstavlja ključni aspekt robusnosti i pouzdanosti softvera. Nepredviđeni ulazi ili nepostojanje resursa kao i raznovrsni faktori mogu izazvati greške koje ugrožavaju normalno funkcionisanje aplikacije. Izuzeci su mehanizam koji omogućavaju programima da reaguju na takve neočekivane situacije, čime se sprečava nepredvidivo ponašanje i omogućava brz i lak oporavak od potencijalnih problema.

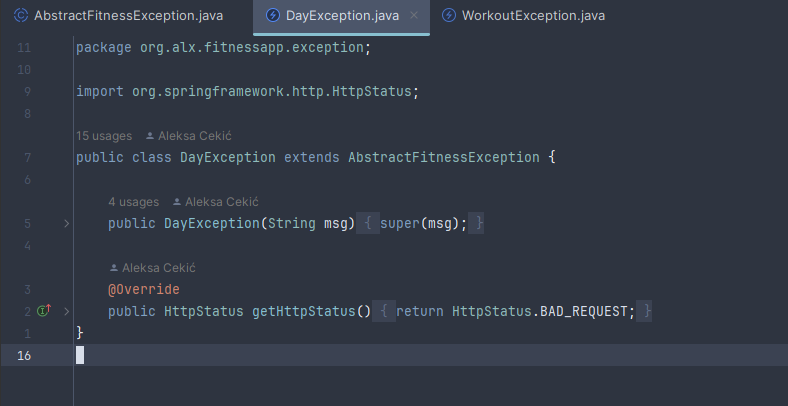
Java i njen Spring Boot radni okvi pružaju snažne mehanizme za njihovo efikasno hvatanje i obradu. U Javi, izuzeci su objekti koji se koriste za signaliziranje grešaka tokom izvođenja programa. Oni mogu nastati iz različitih razloga, uključujući nepravilne ulaze, problem sa resursima ili neočekivano stanje u programu. Java pruža hijerarhiju klasa koje nasleđuju Throwable, a izuzeci mogu biti proveravajući (checked) ili neproveravajući (unchecked). Proveravajući izuzeci moraju biti uhvaćeni ili deklarisani u okviru metode, dok neproveravajući izuzeci nastaju iz nekih ozbiljnijih problema i obično se ne očekuju da budu uhvaćeni.



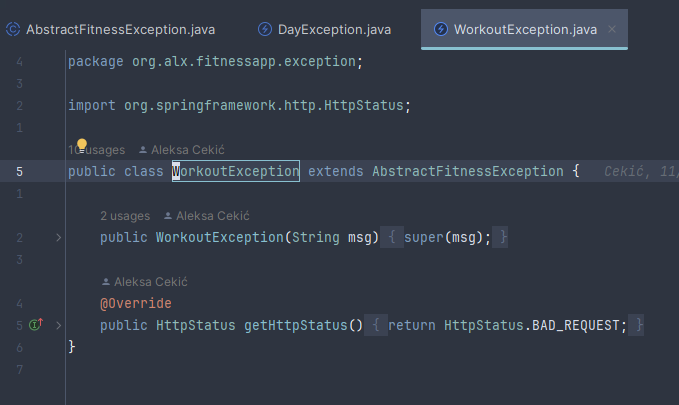
Slika 5.6.1 - Prikaz hierarhije javinih klasa koje nasleđuju Throwable [izvor: Google Images, Javapoint]



Slika 5.6.2 - Prikaz rukovanja izuzecima na backendu prvi deo [izvor: autor]



Slika 5.6.3 - Prikaz rukovanja izuzecima na backendu drugi deo [izvor: autor]



Slika 5.6.4 - Prikaz rukovanja izuzecima na backendu treći deo [izvor: autor]

Na backendu smo obradili izuzetke tako što smo kreirali sve prilagođene klase koje nasleđuju **AbstractFitnessException.** *AbstractFitnessException* je apstraktna klasa koja nasleđuje *RunetimeException,* u njoj se nalazi apstraktna metoda *getHttpStatus*(). Ta HttpStatus metoda služi da sve klase koje nasleđuju ovu apstraktnu klasu imaju mogućnost da setuju koji je njihov HTTP Status sa svojim kodovima kao što su *BadRequest (400)*, *OK (200)*, *Forbidden (403)*, *Unauthorized (401)*, itd.

Obrada izuzetaka na frontend služi za održavanje stablinih i korisnički prijateljskih web aplikacija. U React-u rukovanje izuzecima smo izvršili korišćenjem **try-catch** **blokova** i logovanjem grešaka:



Slika 5.6.5 - Prikaz rukovanja izuzecima na frontendu [izvor: autor]

## Testiranje aplikacije

Testiranje aplikacije je ključna faza u razvoju softvera koja ima za cilj validaciju i verifikaciju funkcionalnosti sistema kako bi se obezbedio kvalitet i pouzdanost softvera. Postoje različiti nivoi testiranja, od koga su dva osnovna *jedinično testiranje* i *integraciono testiranje*.

Spring Boot pruža niz alata i praksi koje omogućavaju efikasno i kvalitetno testiranje. Alati koji su korišćeni u ovom projektu su *Mockito* i *JUnit 5*.

**Mocikto** je popularna biblioteka za kreiranje i rad sa mock objektima u okviru jediničnih testova. Mockicto omogućava simulaciju ponašanja objekata, kao na primer servisi, repozitorijumi ili baze podataka. Ovo je ključno za izolovanje jedinica koda i garantovanje da se testiraju samo određeni delovi aplikacije. Ključne karakterisitke mockita su:

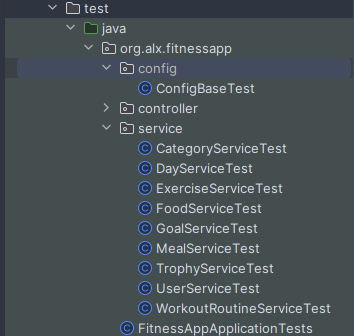
* **Mock Objekti:** Omogućava kreiranje mock objekata koji simuliraju stvarne objekte, ali ne zahtevaju stvarne implementacije.
* **Verifikacija Poziva:** Pruža metode za proveru da li su određene metode pozvane sa određenim argumentima, što omogućava proveru interakcija između objekata.
* **Stubiranje Metoda:** Definisanje ponašanja mock objekata, uključujući vraćanje vrednosti ili izazivanje izuzetaka, kako bi se simuliralo stvarno ponašanje.
* **Argument Matchers:** Omogućava precizno podešavanje očekivanih argumenata poziva metoda.
* **Annotations:** Mockito koristi anotacije kao što su **@Mock**, **@InjectMocks**,i **@Spy** kako bi olakšao integraciju sa *JUnit*-om.

**JUnit 5** je okvir za jedinično testiranje u javi. Osnovna svrha je provera ispravnosti jedinica koda, kao što su metode, klase ili čak celih paketa. JUnit olakšava pisanje i izvršavanje testova, označavajući metode anotacijom “@Test”. Ove označene metode se koriste kao test slučajevi. Ključne karakteristike Junit-a su:

* **@Test**: Anotacije koje obeležavaju metode koje treba izvršiti kao test case-ove.
* **Assertions**: Pruža različite metode asertacije (npr. *assertEquals*, *assertTrue*, itd.) za proveru očekivanih rezultata.
* **Setup metode**: Anotacije poput **@BeforeEach** i **@AfterEach** omogućavaju izvršavanje određenih metoda pre i posle svakog testa, što pomaže u postavljanju i čišćenju okoline za testiranje.
* **Test suites:** Grupisanje test case-ova u logičke celine pomoću **@RunWith** i **@Suite** anotacija.

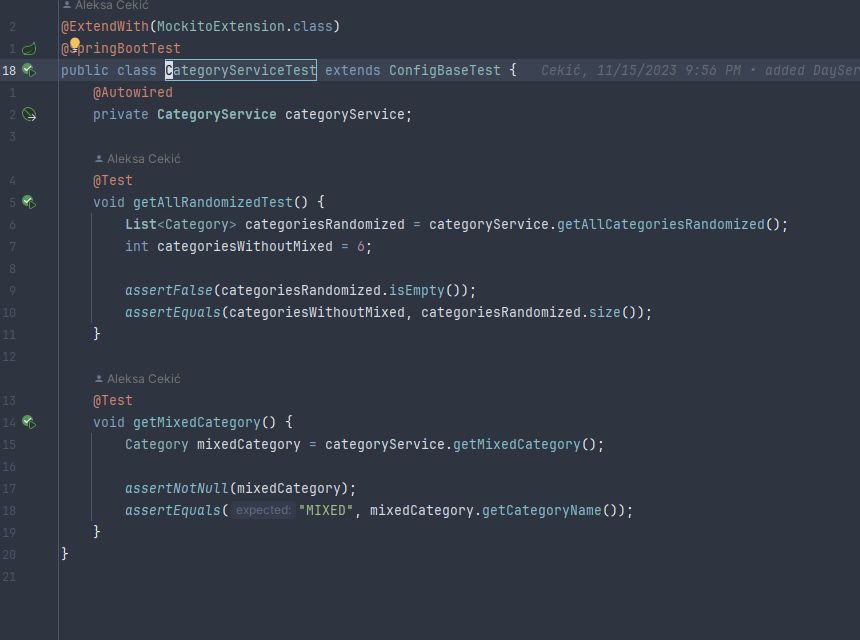
### Jedinično testiranje

Glavni fokus jediničnog testiranja je provera ispravnosti pojedinačnih jedinica koda, kao što su funkcije, metode ili klase. Cilj je identifikovati greške na najmanjem mogućem nivou, omogućavajući programerima da testiraju svaku komponentu zasebno. Jedinice koda se testiraju sa različitim ulazima, a rezultati se upoređuju sa očekivanim vrednostima. Prednosti jediničnog testiranja uključuju brzo otkrivanje grešaka, lakše održavanje koda i poboljšanje dizajna softvera.

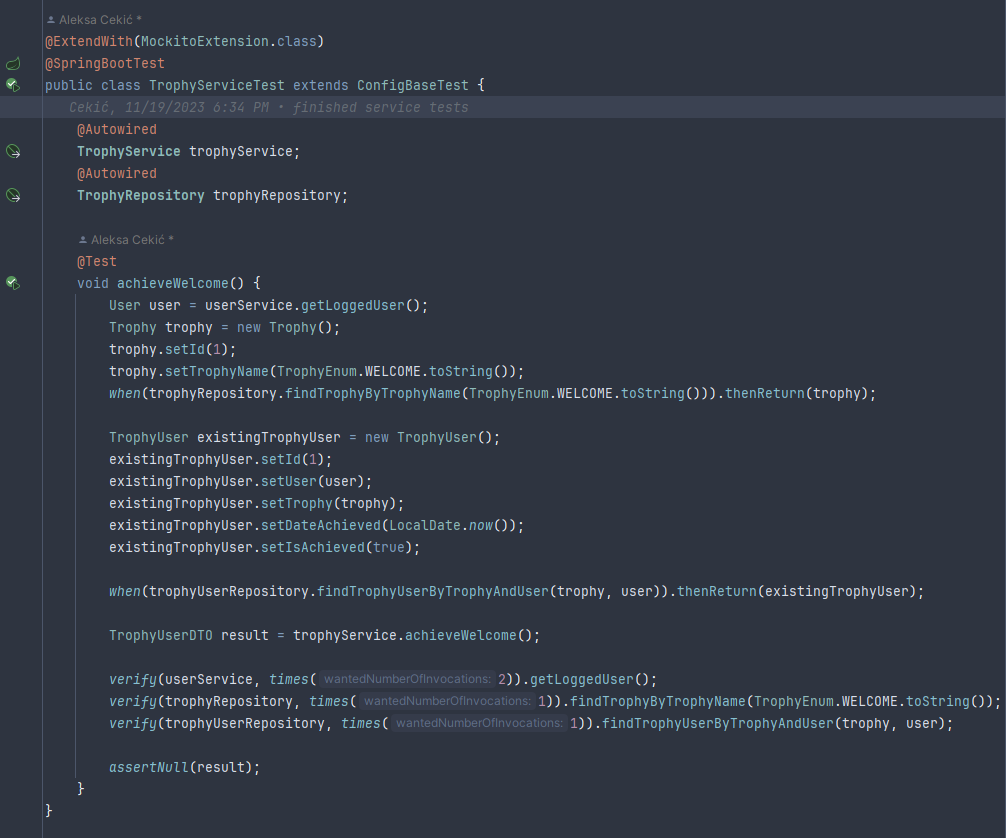


Slika 5.7.1.1 - Prikaz strukture test paketa [izvor: autor]

Ovde možemo videti strukturu test paketa i kako su podeljeni po servisima. Svaka od ovih test klasa predstavlja skup jediničnih testova koji ciljaju određeni servis unutar Spring Boot aplikacije. Ova organizacija omogućava programerima da održe jasnu strukturu testova, olakšavajući održavanje i upravljanje tokom vremena. Na daljim slikama možemo videti primer kako jedinični testovi izgledaju:



Slika 5.7.1.2 - Prikaz jedne test klase[izvor: autor]



Slika 5.7.1.3 - Prikaz druge test klase koja koristi Mockito [izvor: autor]

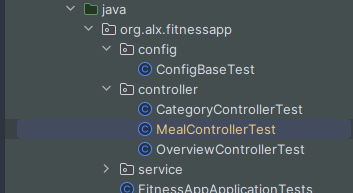


Slika 5.7.1.4 - Prikaz izvršenja jediničnih testova [izvor: autor]

Svi ovi servisni testovi čine ključni deo strategije testiranja Spring Boot aplikacija, obezbeđujući visok nivo kvaliteta, stabilnosti i pouzdanosti softvera.

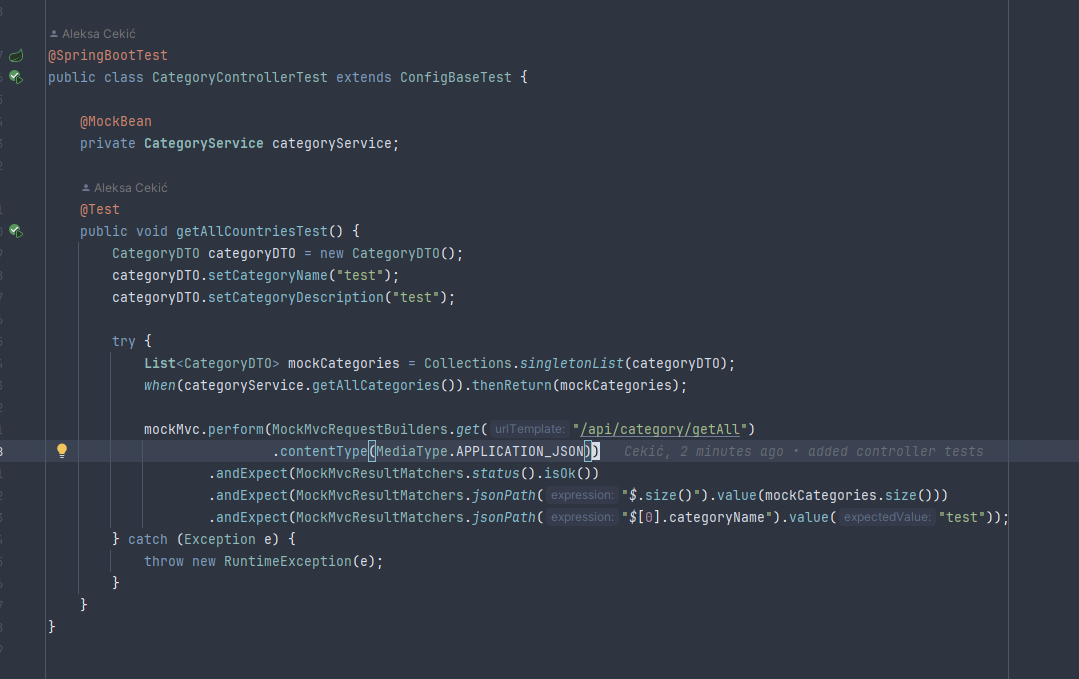
### Integraciono testiranje

Cilj Integracionog testiranja je provera ispravnosti komunikacije i interakcije između različitih jedinica koda. Umesto fokusiranja na pojedinačne komponente, ovo testiranje istražuje kako se komponente povezuju i sarađuju. Ovo uključuje testiranje interfejsa, baza podataka, API poziva i drugih aspekata integracije. Integraciono testiranje identifikuje probleme koji mogu nastati kada se komponente integrišu i pruža sigurnost u vezi sa ispravnošću sistema u celini.

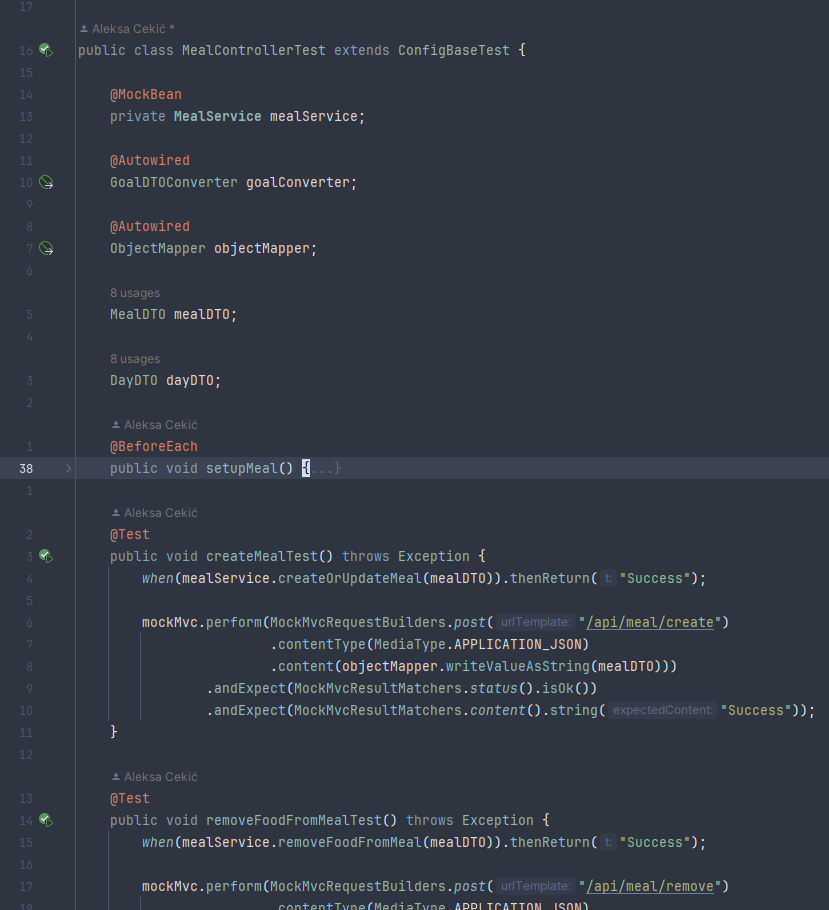


Slika 5.7.2.1 - Prikaz strukture kontroler testova [izvor: autor]

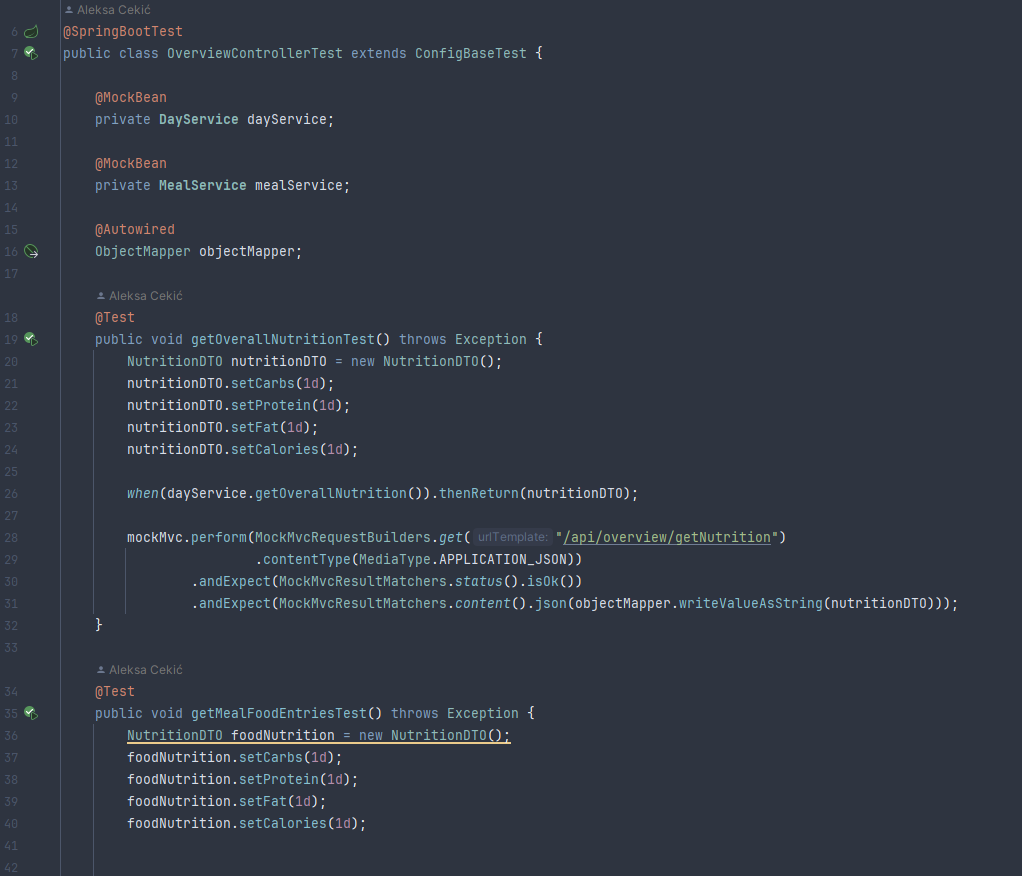
U ovom projektu, integraciono testiranje se izvršava na nivou API poziva, tako da će se korišćenjem JUnita i Mockita testirati kontroleri i njihovi API pozivi. Predstavljeni testovi su sličniji kombinaciji testova jedinica i integracije. Radi prikaza koristićemo sledeće:



Slika 5.7.2.2 - Prikaz testova za kategoriju [izvor: autor]



Slika 5.7.2.3 - Prikaz testova za Obrok kontroler [izvor: autor]



Slika 5.7.2.4 - Prikaz testova za Overview [izvor: autor]

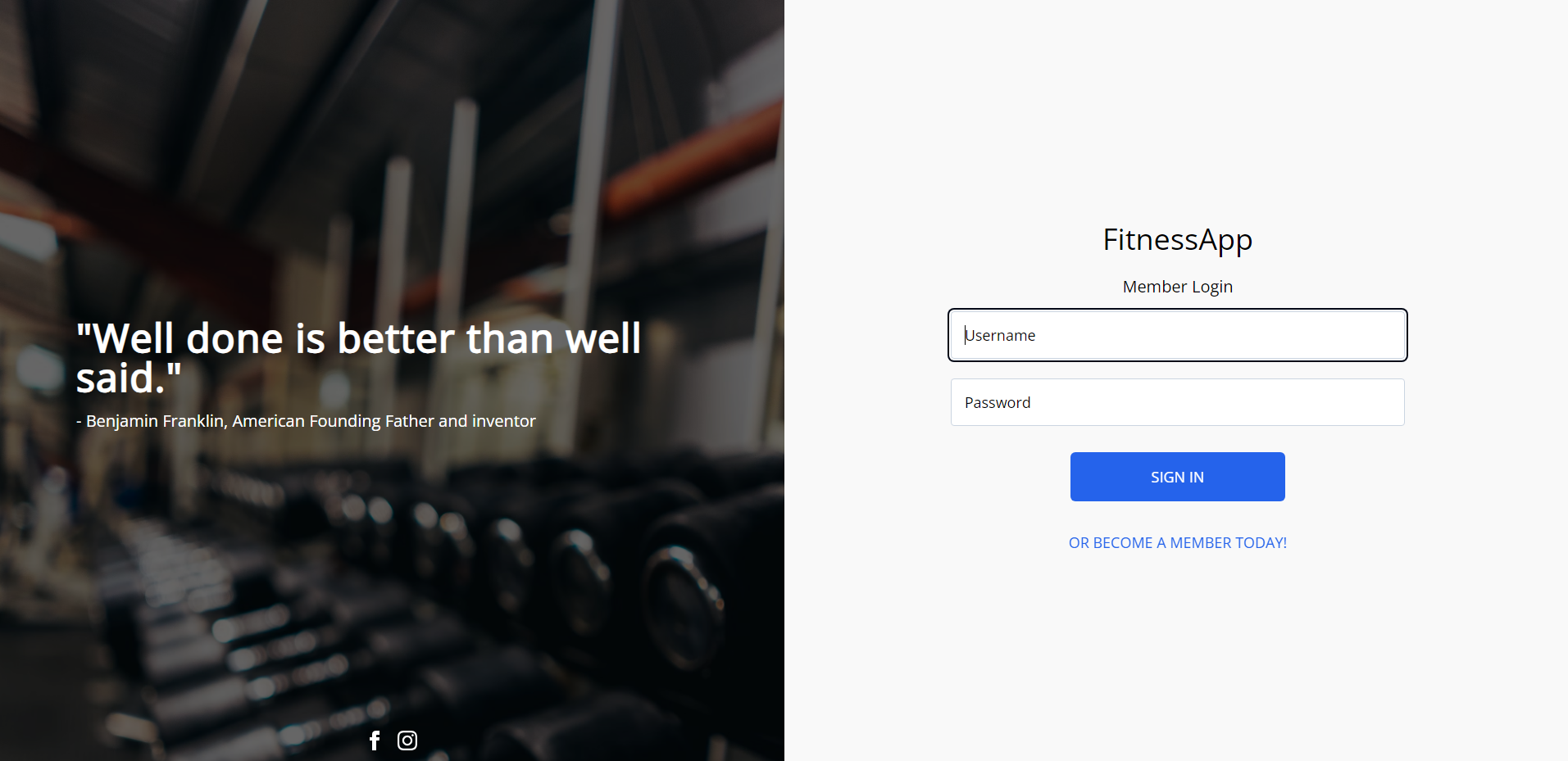
Na osnovu predstavljenih integracionih i jediničnih testova, možemo zaključiti da su implementirani testovi pažljivo osmišljeni kako bi obezbedili celovitu i kvalitetnu proveru funkcionalnosti sistema. Unit testovi pružaju detaljnu proveru ispravnosti pojedinačnih komponenti, što je od suštinskog značaja za otkrivanje i ispravljanje grešaka na nivou koda. S druge strane, integracioni testovi omogućavaju proveru kako te komponente funkcionišu zajedno, identifikujući potencijalne problematične interakcije. Iako su testovi predstavljeni kao kombinacija jedinica i integracije, važno je naglasiti da su oni prilagođeni specifičnostima sistema i zahtevima projekta. Ova prilagodljivost omogućava efikasno testiranje različitih aspekata sistema, čime se obezbeđuje sveobuhvatna provera njegove funkcionalnosti. Takođe ovde radimo sa mokiranim podacima umesto sa stvarnim podacima iz baze.

# Demonstracija funkcionalnosti aplikacije

U ovom segmentu, sprovešćemo demonstraciju aplikacije koja će sveobuhvatno prikazati korisničko iskustvo, obuhvatajući ključne faze životnog ciklusa aplikacije, počevši od registracije i prijave korisnika, pa sve do kreiranja vežbi i sastavljanja obroka.

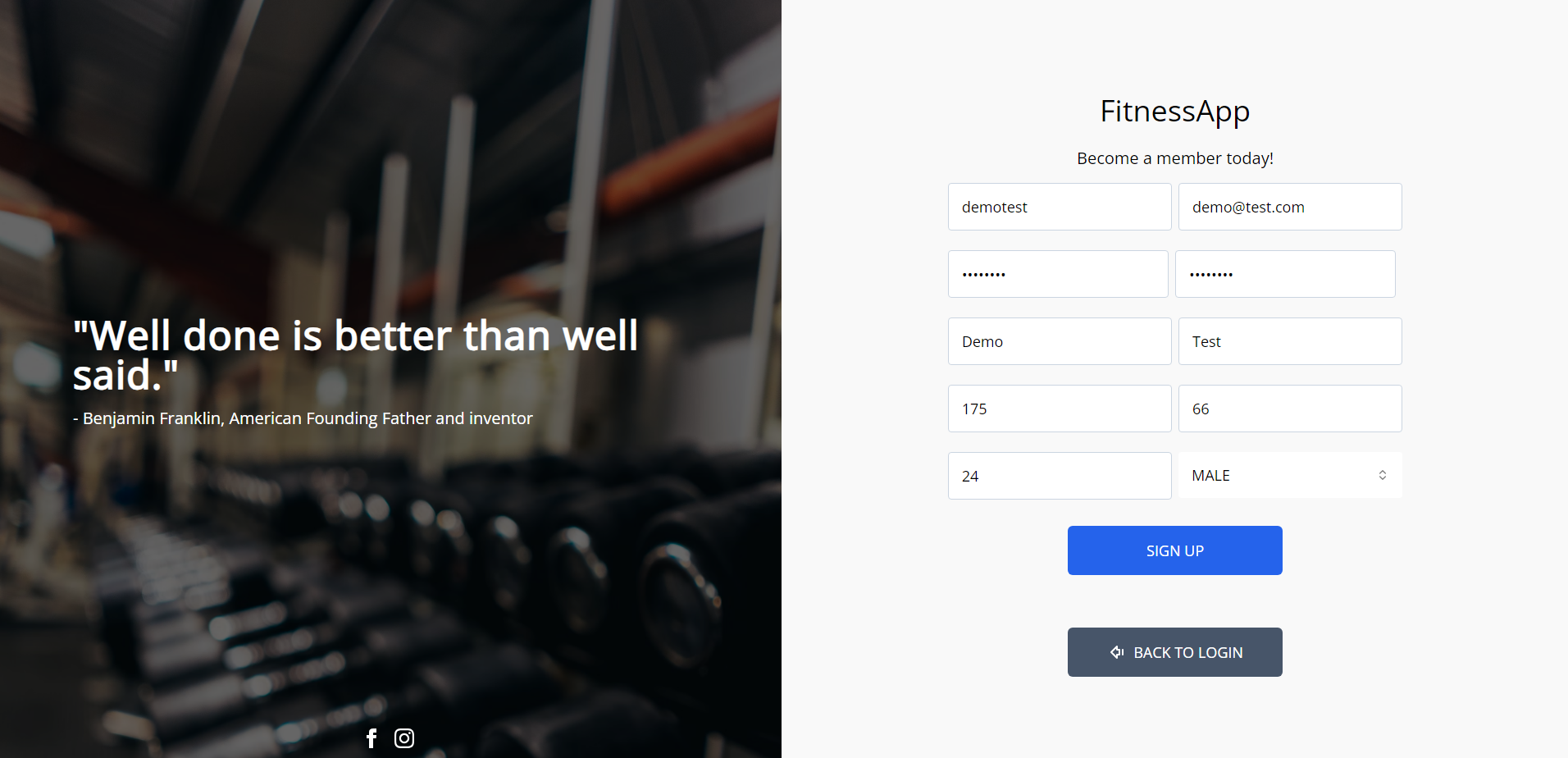
Počinjemo od osnovnih koraka, registracije i prijave korisnika. Ovi početni koraci obezbeđuju personalizovan pristup, čineći iskustvo korisnika jednostavnim i prilagodljivim.

Pristup linku aplikacije automatski će nas preusmeriti na deo za prijavljivanje, gde korisnik može uneti svoje pristupne podatke i pristupiti aplikaciji.



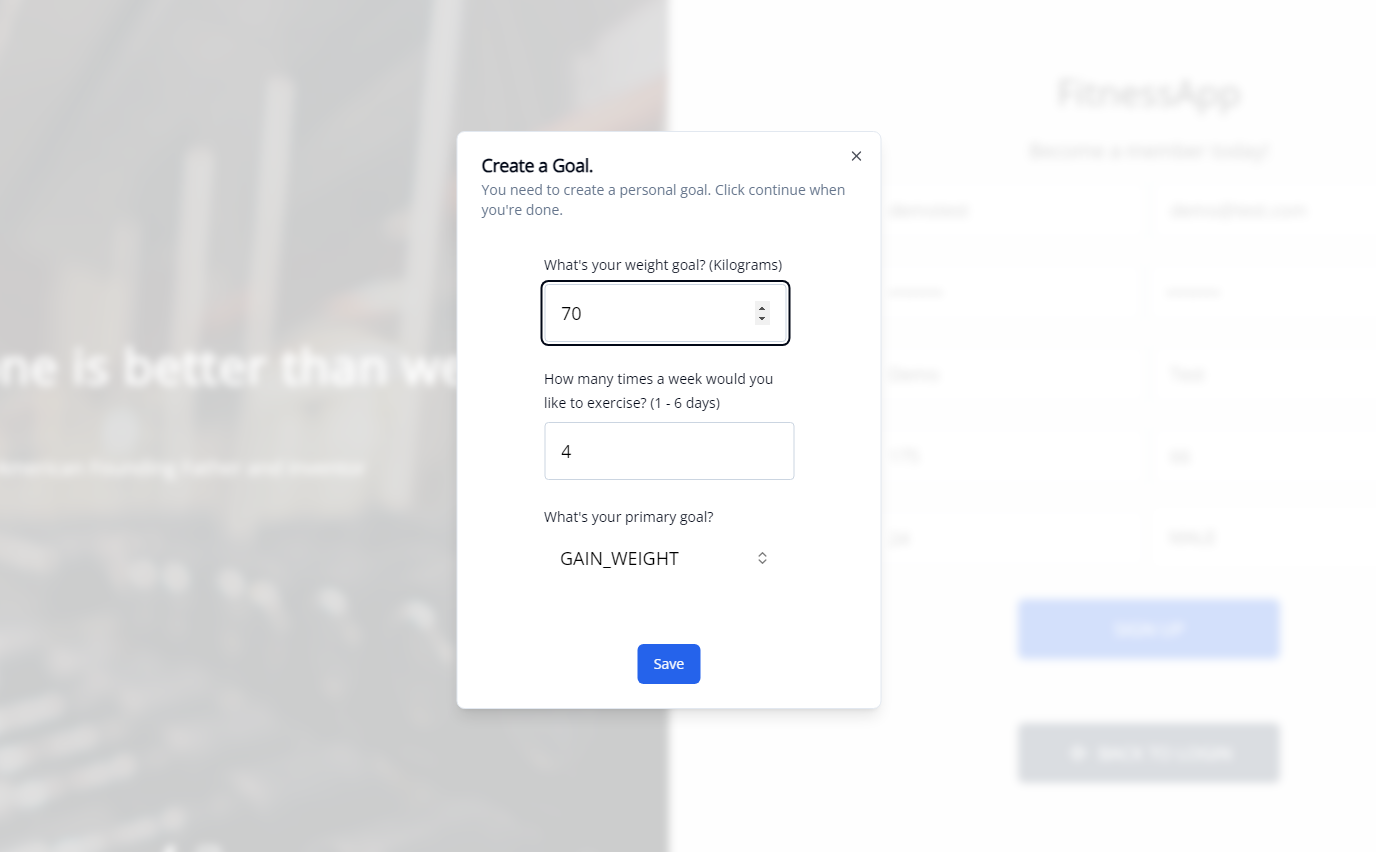
Slika 6.1.1 - Prikaz login forme [izvor: autor]

S druge strane, ukoliko korisnik nema već postojeći nalog, može se prijaviti klikom na opciju “Or become a member today!”, što će ga preusmeriti na deo gde može izvršiti registraciju i postati novi član.



Slika 6.1.2 - Prikaz registracijske forme [izvor: autor]

Nakon što korisnik popuni sva polja sa svojim podacima, može kliknuti na dugme “Sign Up”, što će otvoriti dijalog za postavljanje korisnikovih ciljeva. U slučaju bilo kakve greške prilikom unosa podataka u polja, pojaviće se odgovarajuća obaveštenja, i korisnik neće moći nastaviti dalje dok ne unese tačne podatke.



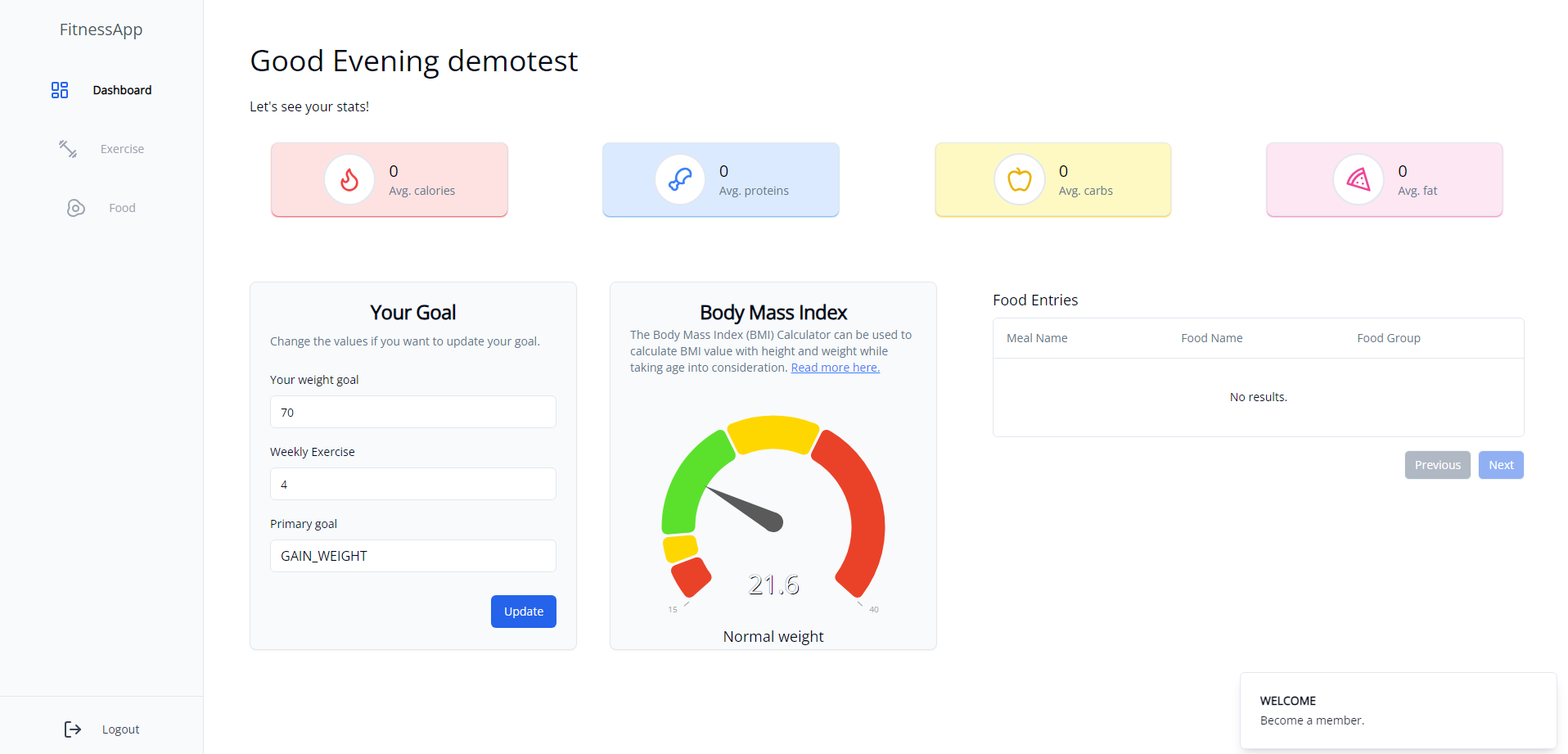
Slika 6.1.2 - Prikaz registracijske forme [izvor: autor]

Korisnik zatim treba da unese svoje ciljeve, uključujući broj treninga nedeljno, ciljanu težinu i glavni cilj. Klikom na dugme 'Save', korisnik će uspešno kreirati svoj nalog, nakon čega može pristupiti prijavi.



Slika 6.1.2 - Prikaz sidebar menija na aplikaciji [izvor: autor]

U bočnom meniju imamo sve dostupne akcije unutar samog sajta. Odavde, korisnik može efikasno odabrati koju akciju želi obaviti, bilo da želi dobiti konačan pregled, pristupiti danima sa vežbama ili pristupiti informacijama o ishrani. Takođe korisnik se može izlogovati sa aplikacije ako to želi.



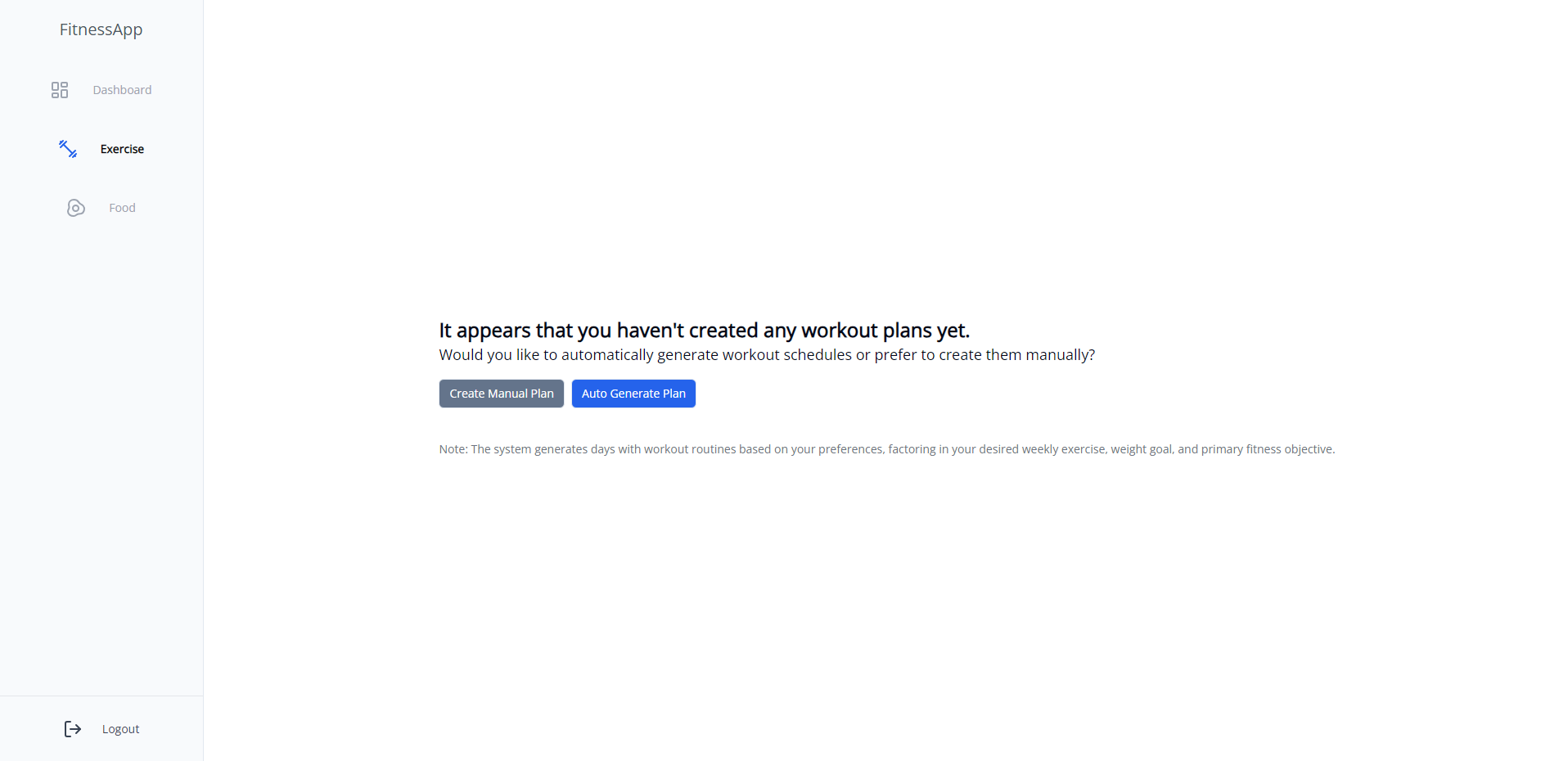
Slika 6.1.3 - Prikaz komandne table [izvor: autor]

Pristupanjem aplikaciji, korisnik može pregledati komandnu tablu sa sveobuhvatnim podacima. Na ovoj tabli, korisnik ima mogućnost promene svojih ciljeva, pregleda izračunate vrednosti indeksa telesne mase (BMI), kao i kompletnog dnevnog unosa hrane, uključujući kalorije, proteine, ugljene hidrate i masti.

Pri prvom pristupu svom nalogu, korisnika će dočekati personalizovana poruka sa trofejnim elementom, koja se nalazi u donjem desnom uglu, pružajući mu dobrodošlicu na platformi.

Što se tiče indeksa telesne mase (BMI), on predstavlja numeričku vrednost koja se koristi za procenu odnosa između težine i visine osobe. Izračunava se deljenjem težine u kilogramima sa kvadratom visine u metrima pomoću formule **BMI = težina(kg) / (visina(m))^2**. Ova vrednost pruža procenu ukupne telesne mase i koristi se kao indikator prekomerne telesne mase ili gojaznosti.

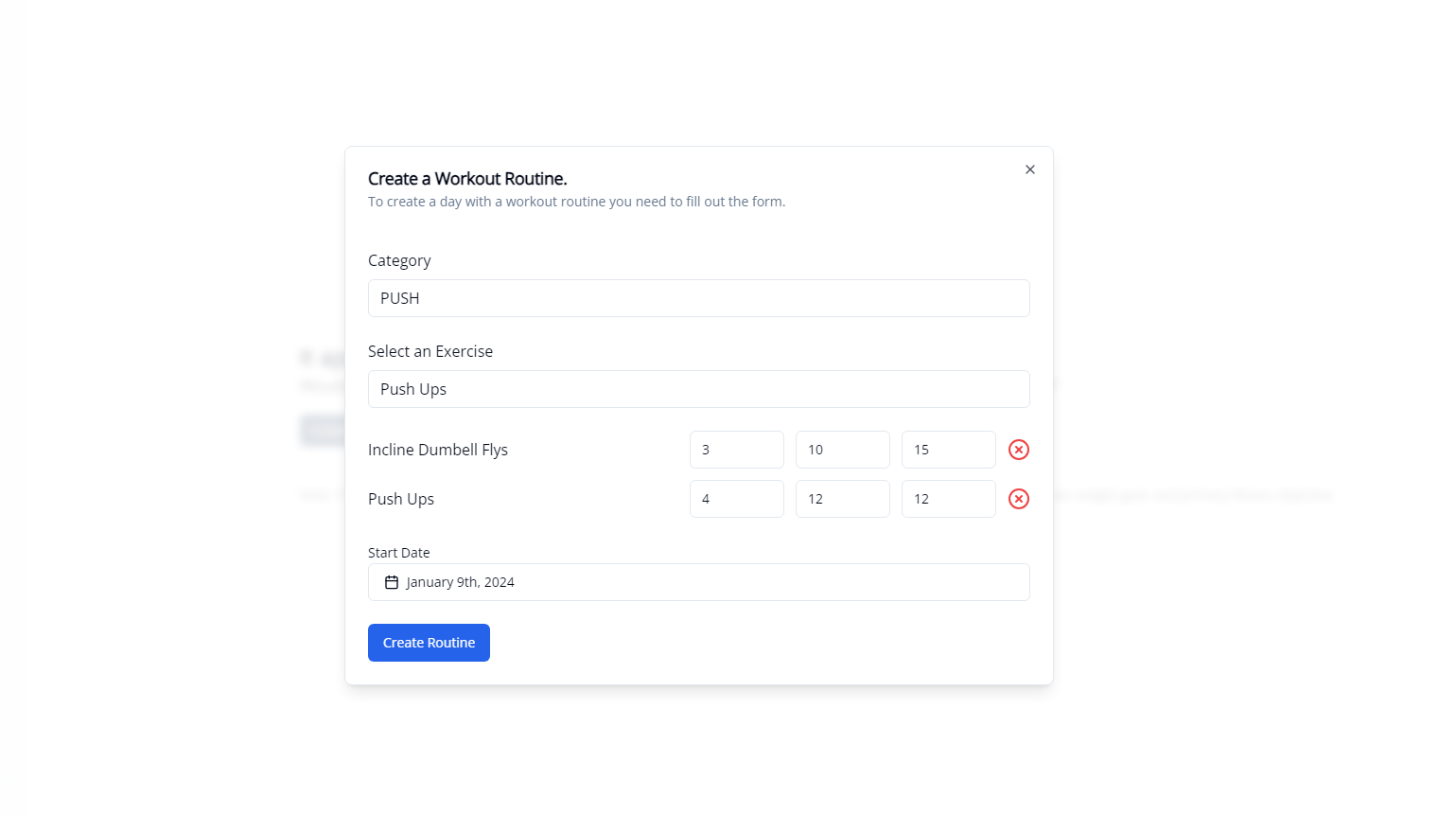
Jedan od ključnih elemenata ove aplikacije je mogućnost korisnicima da prilagode svoj režim vežbanja prema sopstvenim potrebama i ciljevima. Samim tim, kreiranje vežbi možemo podeliti na ručno kreiranje ili automatsko.



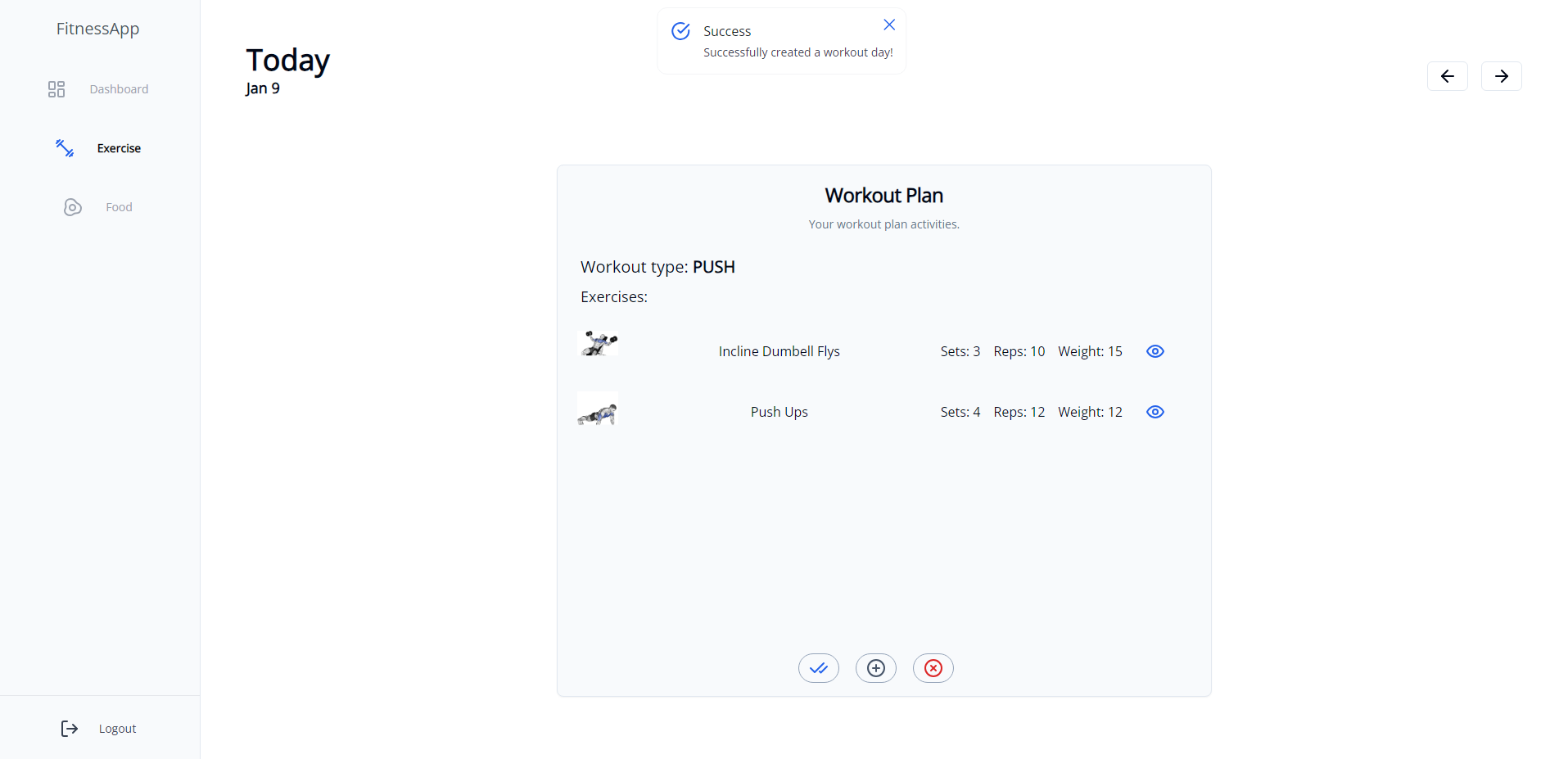
Slika 6.1.4 - Prikaz exercise strane bez kreiranih dana [izvor: autor]

Ručno kreiranje vežbi pruža korisnicima mogućnost da biraju specifične vežbe koje odgovaraju njihovom trenutnom stanju i željama. Ova prilagodljivost omogućava korisnicima da personalizuju svoj trening, čime se postiže optimalan rezultat.

Unošenjem kategorije, datuma početka treninga i liste vežbi sa njihovim statusima, kao što su serije, ponavljanja i težine, korisnik uspešno kreira jedan aktivan dan sa planom treninga.

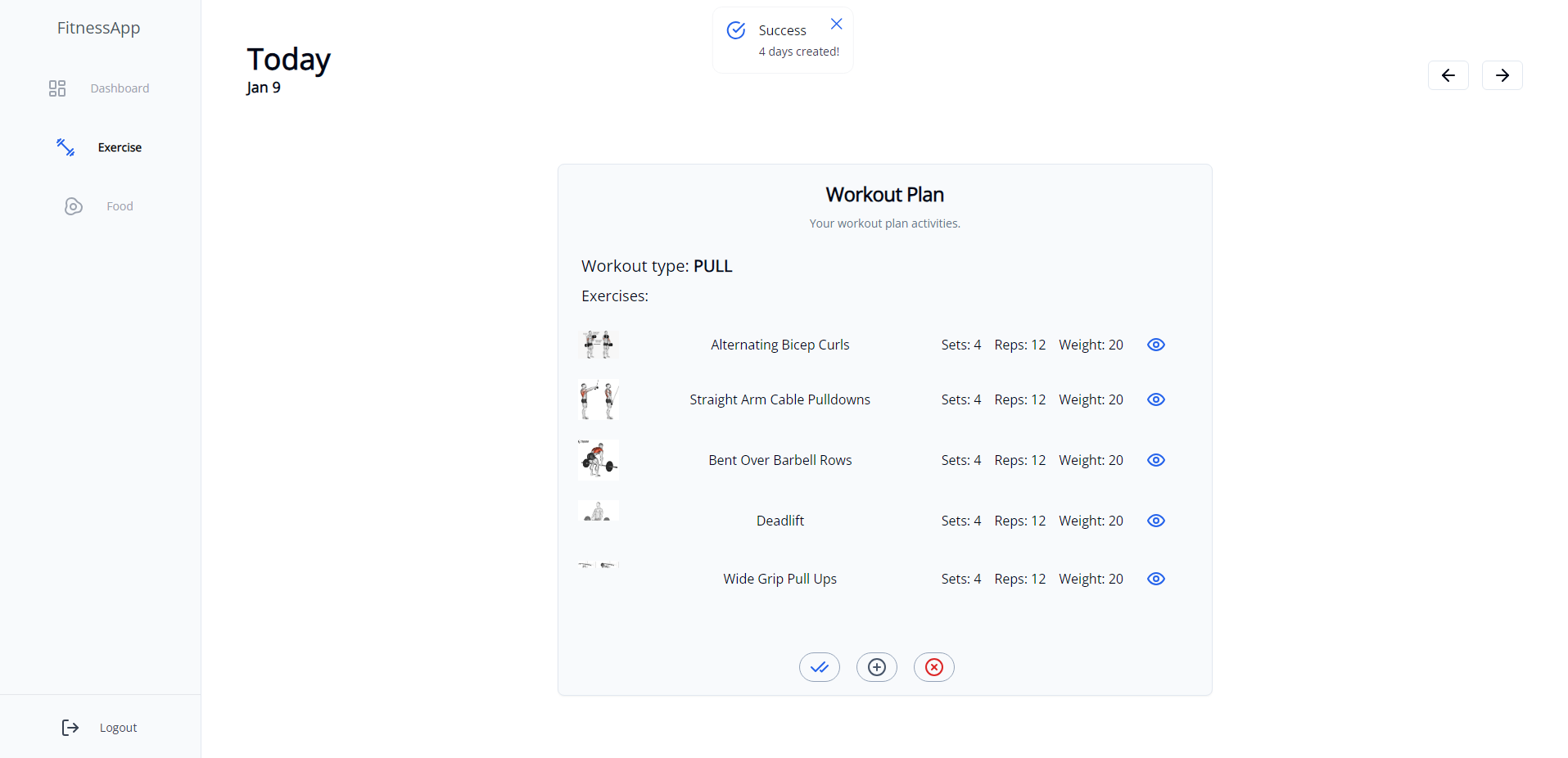


Slika 6.1.5 - Prikaz forme za kreiranje dana sa planom vežbi [izvor: autor]



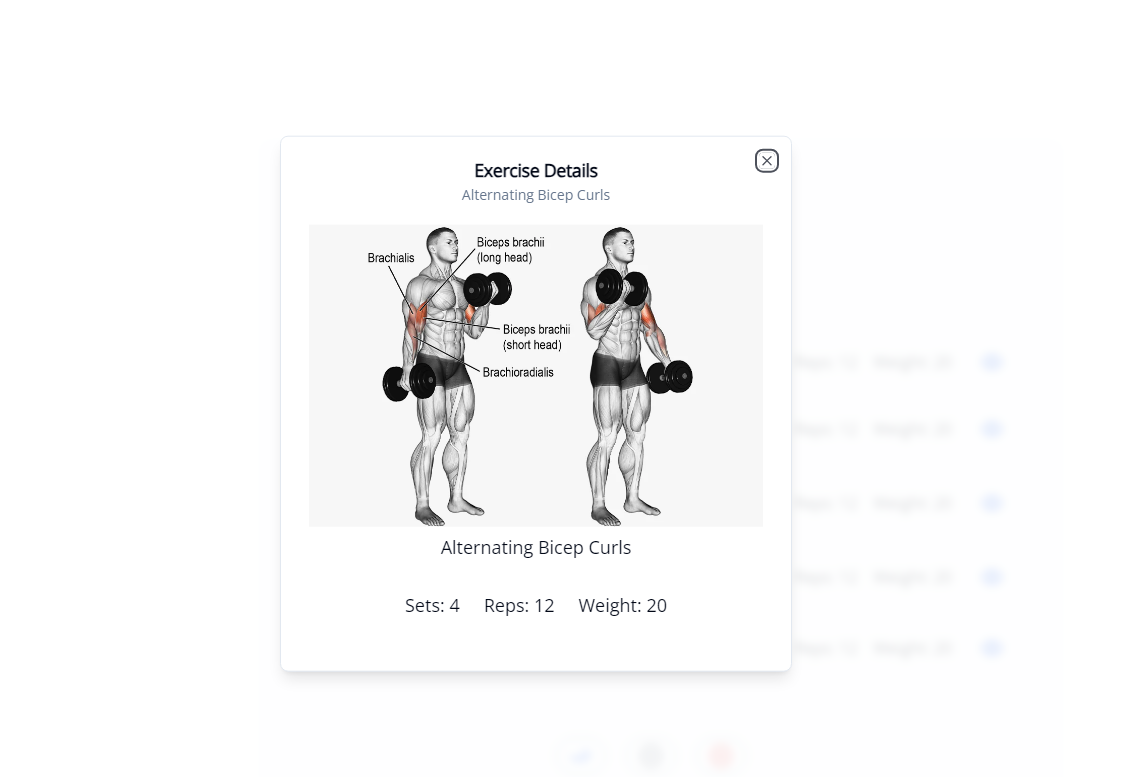
Slika 6.1.6 - Prikaz uspešno kreiranog dana sa planom vežbi [izvor: autor]

Sa druge strane, automatsko kreiranje vežbi koristi algoritme i podatke o korisniku i njegov cilj kako bi predložilo optimalan trening plan. Ova funkcionalnost dodatno pojednostavljuje proces planiranja vežbi, čime se omogućava korisnicima da brže i efikasnije postignu svoje ciljeve.



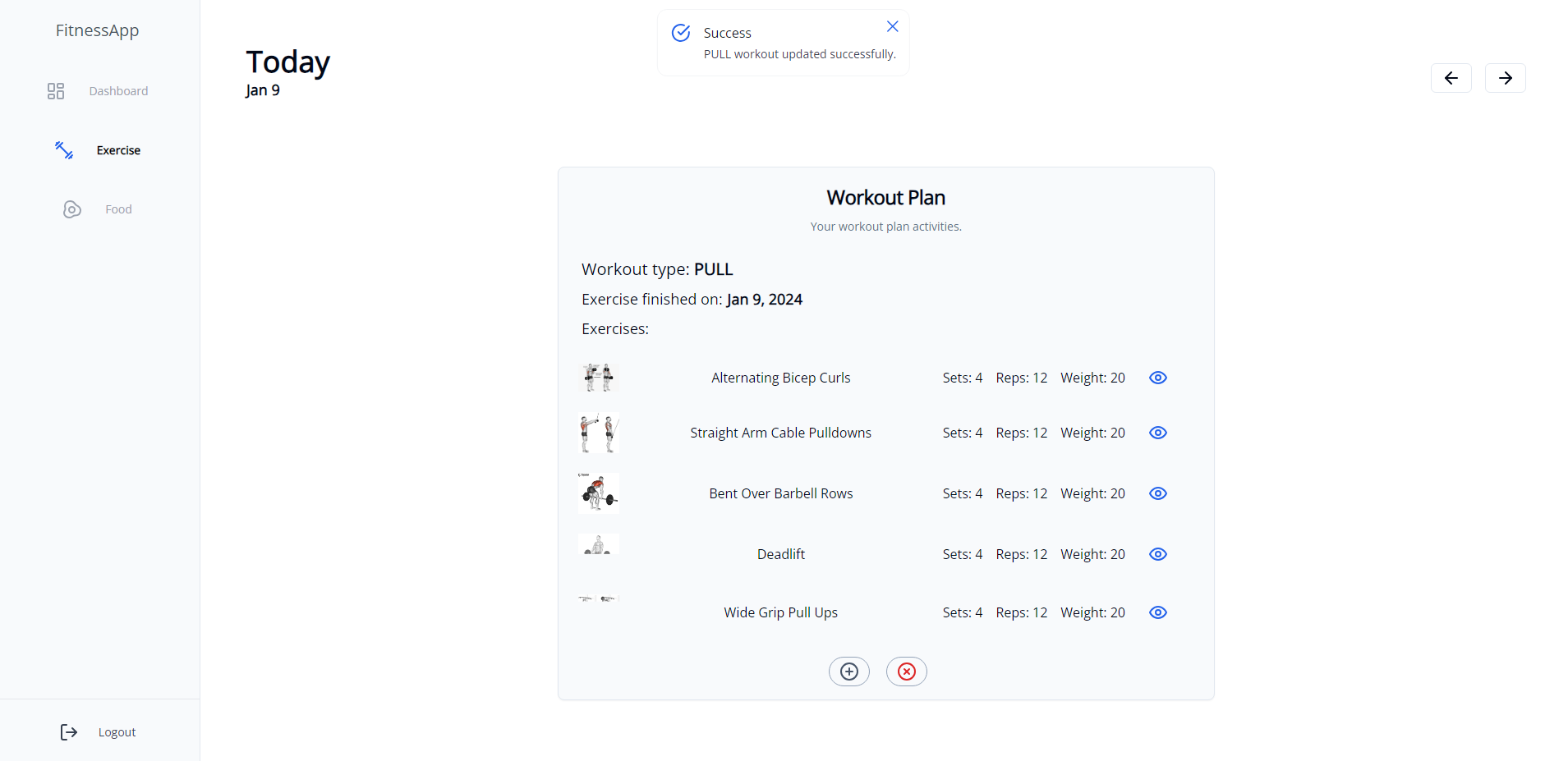
Slika 6.1.7 - Prikaz 4 automatsko kreiranih dana sa planom vežbi [izvor: autor]

U ovom slučaju, sistem kreira 4 dana sa vežbama koje imaju hardcodovane vrednosti za serije, ponavljaje i težinu. Korisnik može detaljnije pogledati vežbu klikom na ikonu desno.



Slika 6.1.7 - Prikaz detalja vežbe [izvor: autor]

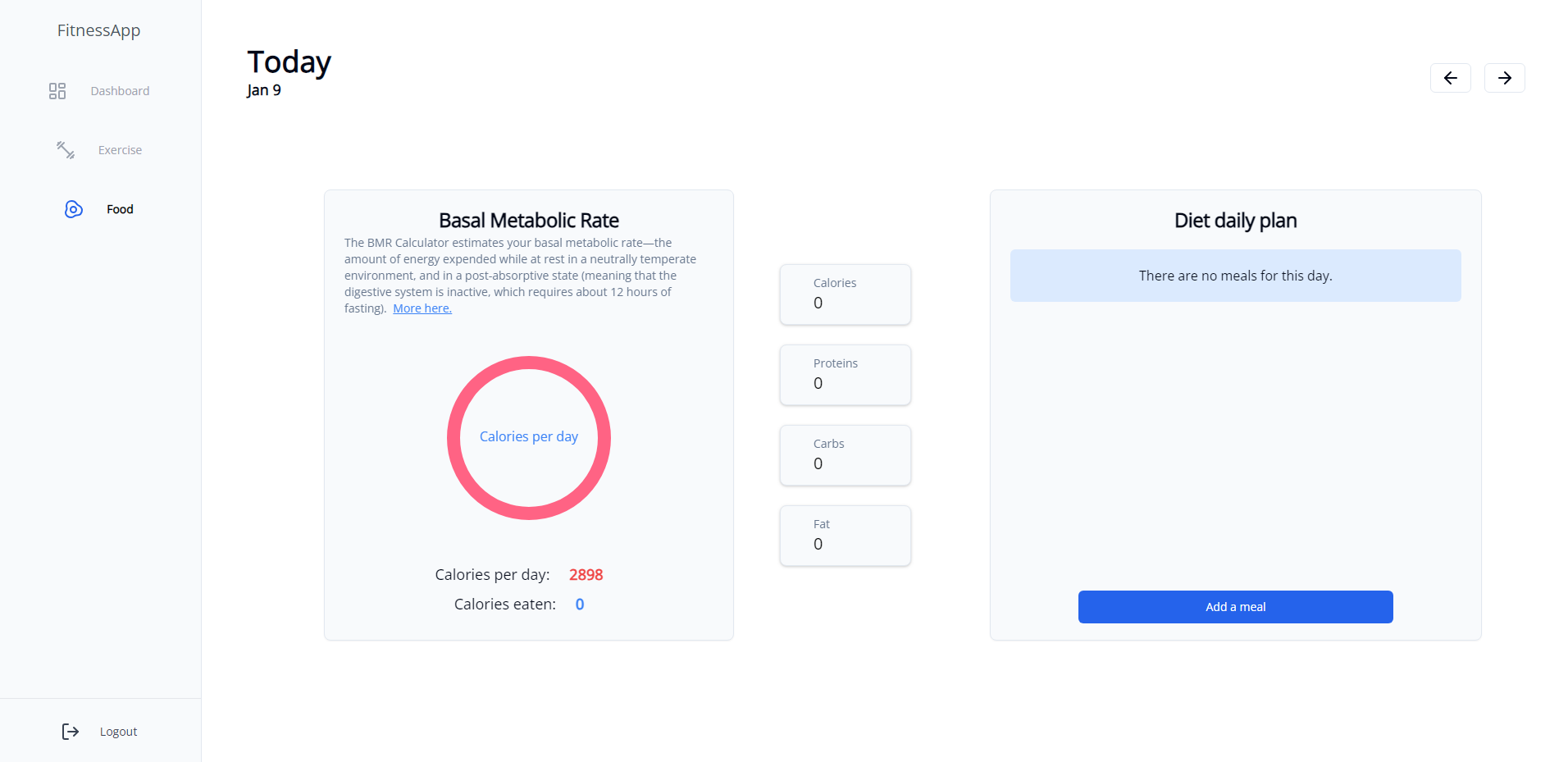
Kada korisnik završi kreiranje vežbe, pritiskom na plavo potvrdno dugme, ažurira se trenutna vežba sa datumom završetka dana kada je vežbu završio. Na slici možemo primetiti dodato polje sa datumom završetka..



Slika 6.1.8 - Prikaz ažuriranja plana vežbi [izvor: autor]

Ako korisnik želi da kreira novi dan sa vežbama, to može učiniti klikom na dugme sa sivom plus ikonom, što će otvoriti isti dijalog kao prilikom ručnog kreiranja vežbe. Klikom na crveno dugme sa simbolom "X", korisnik briše dan sa vežbama.

Pored fitness aspekta, aplikacija pruža i mogućnost praćenja ishrane. Korisnici mogu lako da unose informacije o svojim obrocima, prateći kalorijski unos i nutritivne vrednosti.

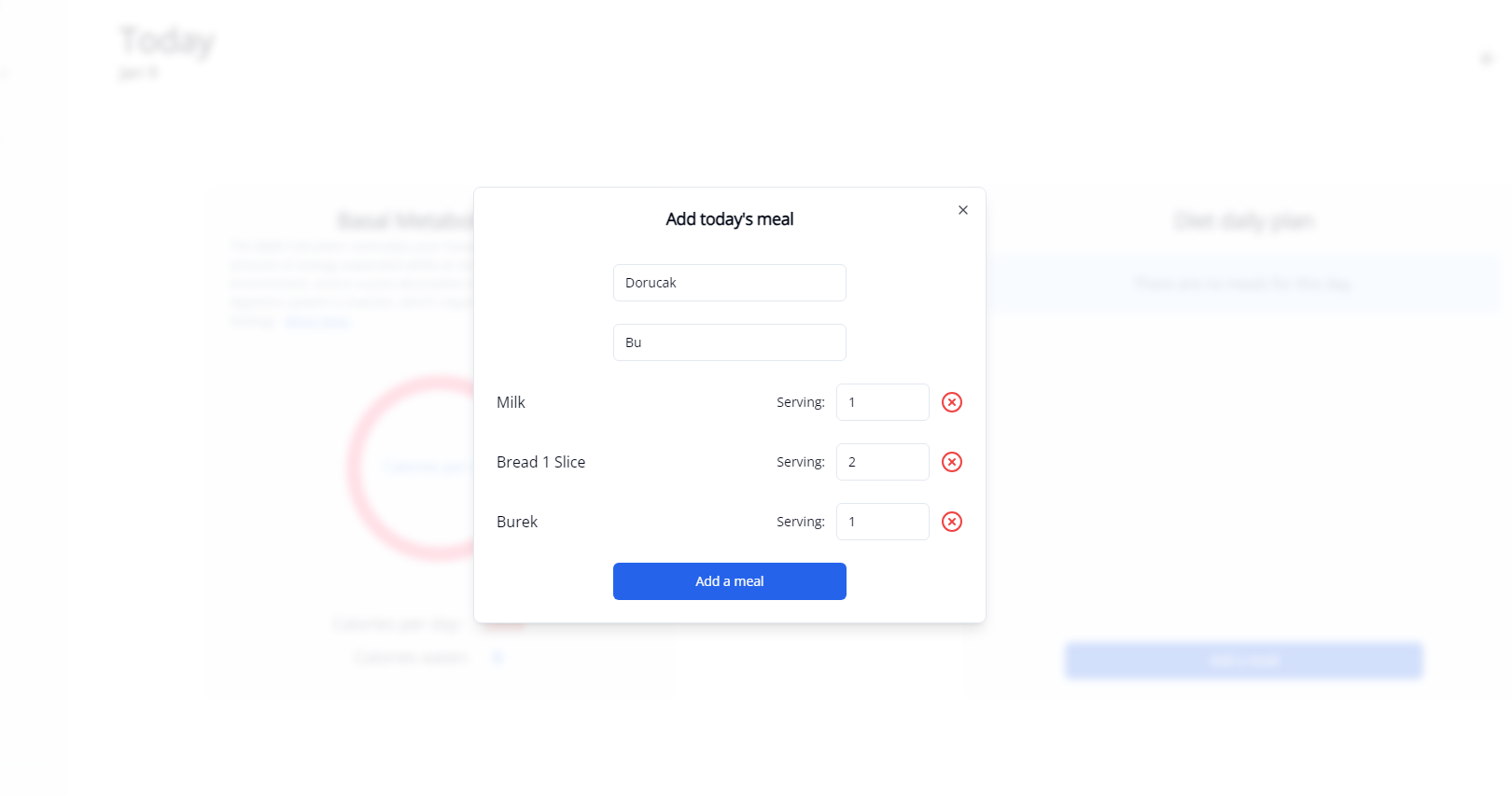


Slika 6.1.9 - Prikaz strane sa ishranom [izvor: autor]

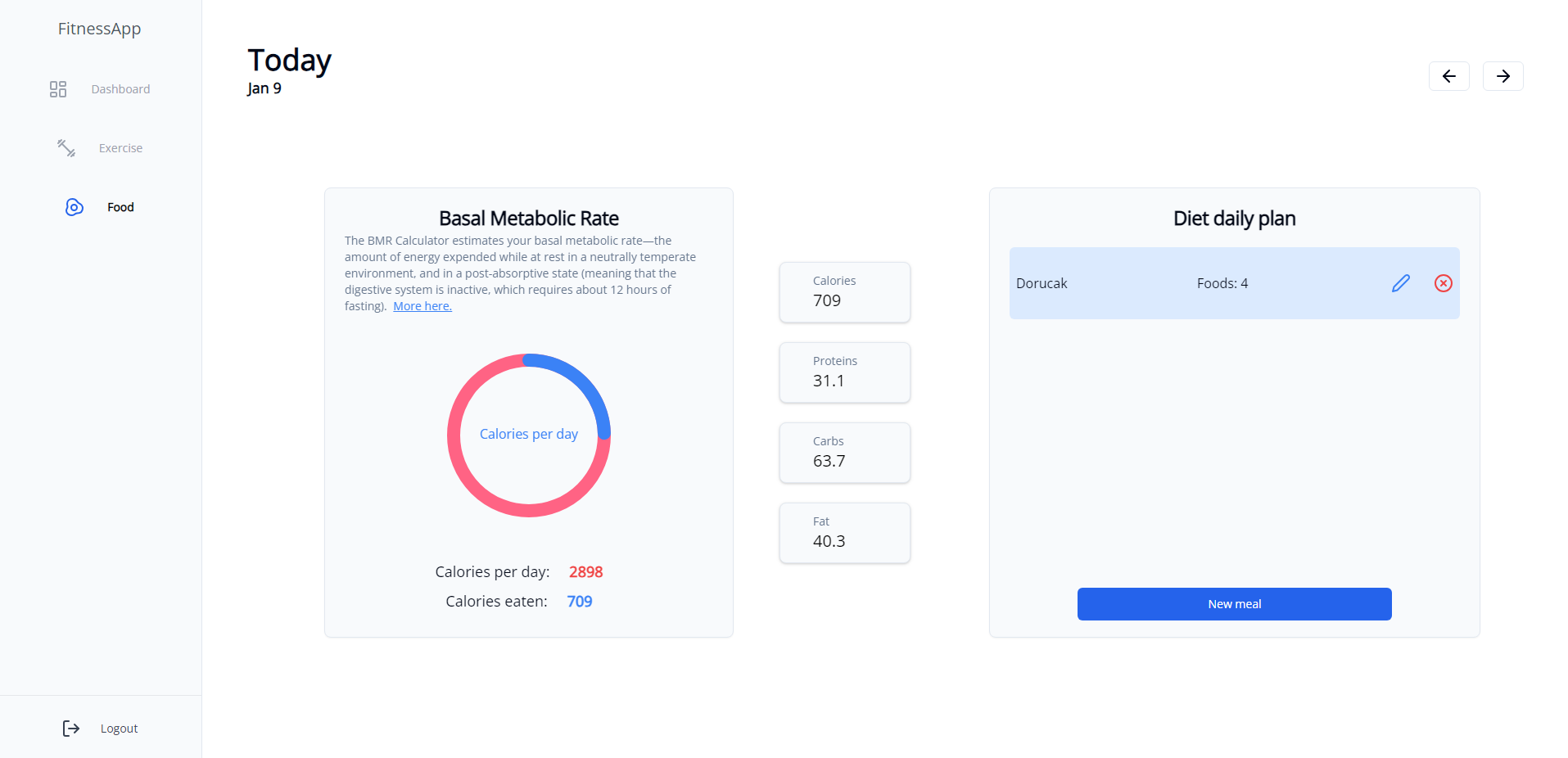
Aplikacija generiše BMR za svakog korisnika na osnovu podataka koje korisnik unosi prilikom registracije, kao što su težina, visina, pol, broj godina i nivo fizičke aktivnosti, tj. koliko puta nedeljno bi trenirao/la.

BMR (Basal Metabolic Rate) predstavlja baznu metaboličku stopu, odnosno energetsku potrošnju tela u potpunom mirovanju, kada je osoba potpuno opuštena, fizički neaktivna, i bez unošenja hrane. BMR predstavlja minimalnu količinu energije koja je potrebna telu da održi osnovne životne funkcije poput održavanja srčanog ritma, disanja, temperature tela i drugih vitalnih procesa.

Na osnovu dobijenog BMR-a, korisniku se preporučuje određeni broj kalorija koje bi trebalo da unese svakog dana. Ovo se postiže kreiranjem obroka i dodavanjem hrane u aplikaciji.

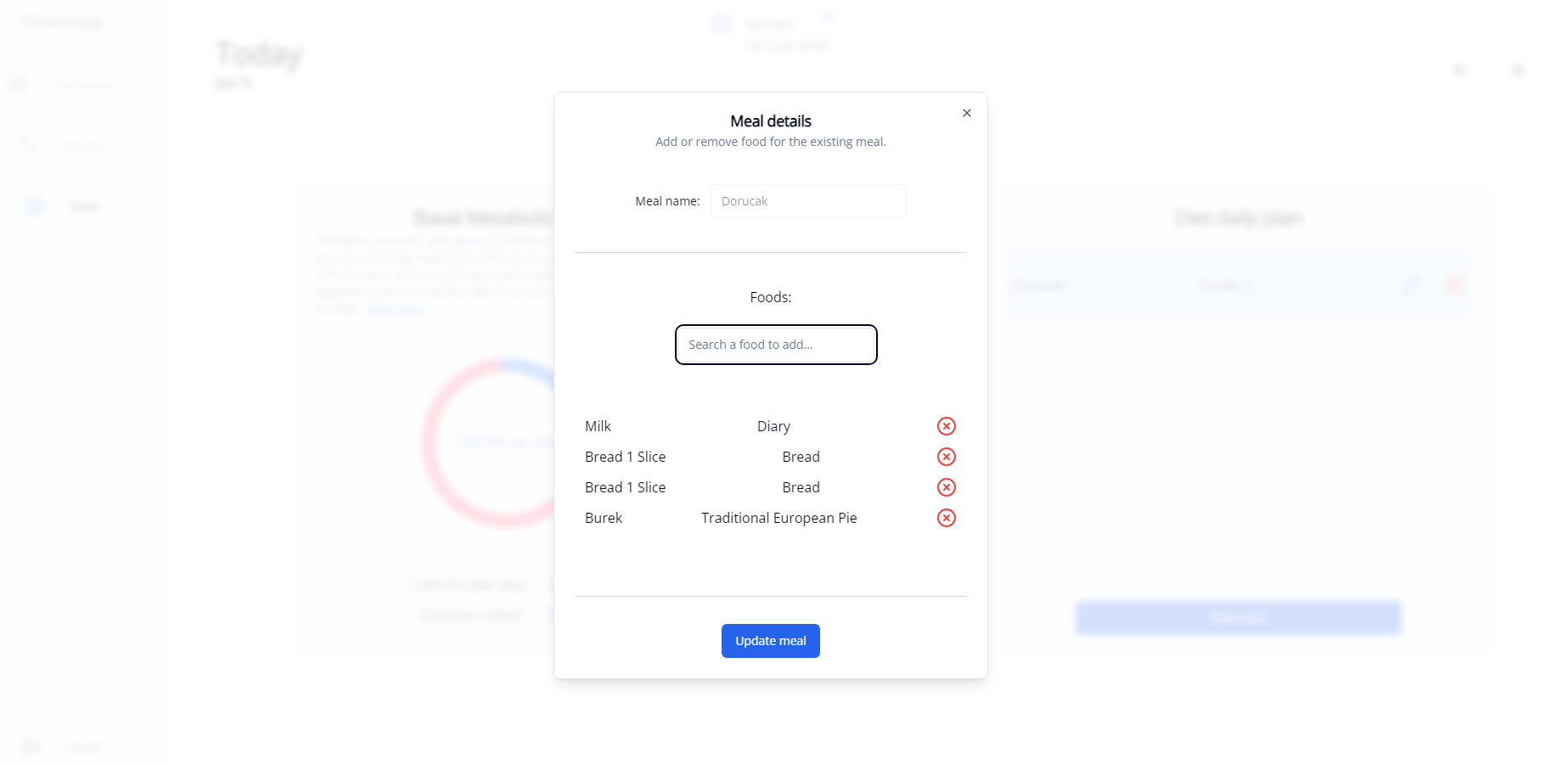


Slika 6.1.10 - Prikaz kreiranje doručka [izvor: autor]



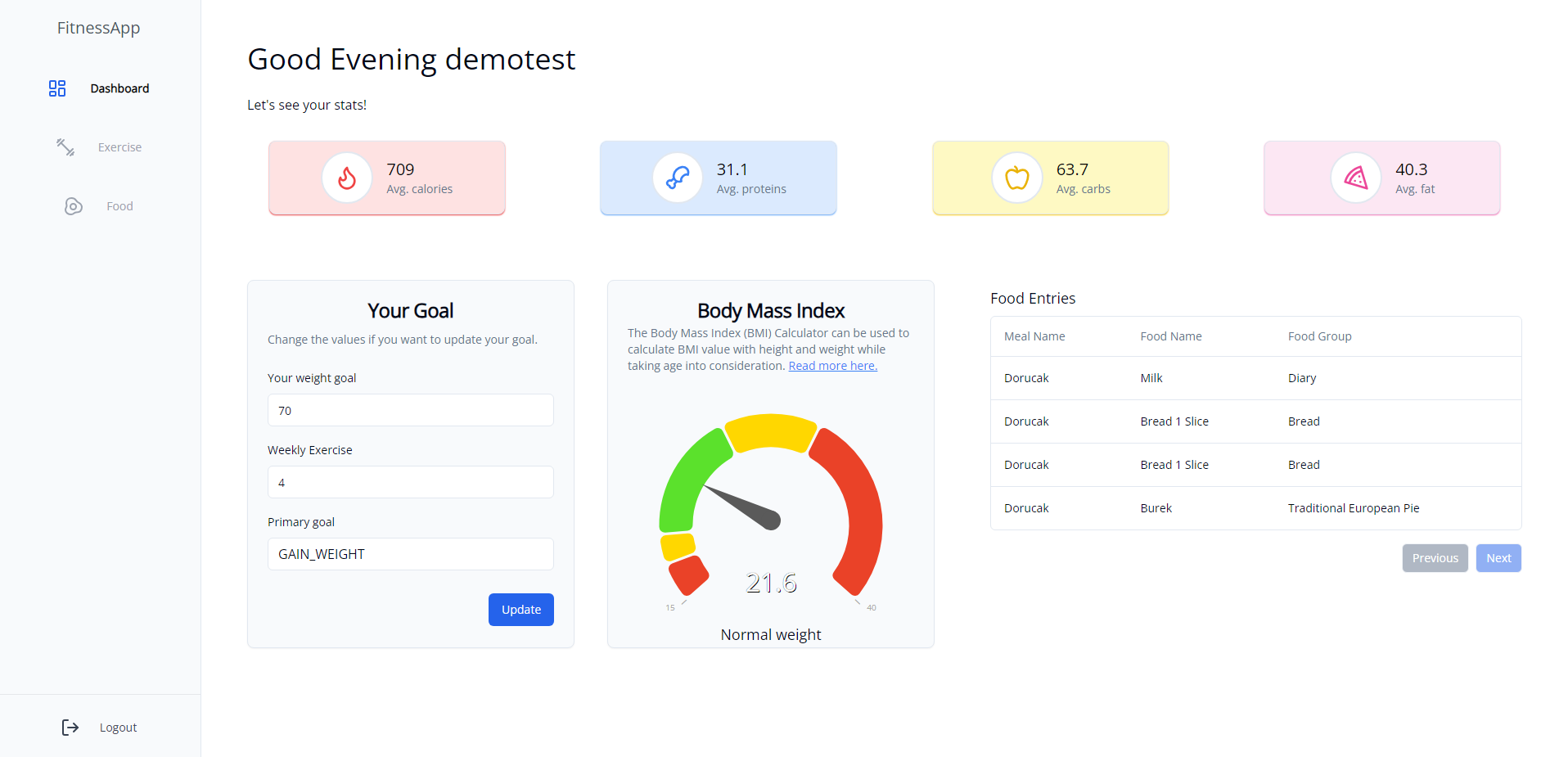
Slika 6.1.11 - Prikaz kako izgleda kada je doručak kreiran [izvor: autor]

Korisnik može obrisati, izmeniti ili kreirati potpuno novo jelo, u zavisnosti od svojih zahteva. Ukoliko odabere izmenu hrane, može dodati nove namirnice ili obrisati postojeće iz obroka.



Slika 6.1.12 - Prikaz izmene doručka [izvor: autor]

Konačno, možemo videti izgled dashboard stranice sa podacima o ishrani koje smo prethodno kreirali.



Slika 6.1.13 - Prikaz komandne table sa kreiranim obrocima [izvor: autor]

Ako je korisnik završio sa treningom ili unosom hrane za određeni dan, može se po želji izlogovati iz aplikacije pritiskom na dugme za odjavljivanje (Logout).

# Uputstvo za korisnike

U okviru ovog uputstva, pružamo detaljne informacije o tome kako lokalno preuzeti i pristupiti Fitness aplikaciji. Pratite sledeće korake kako biste se sigurno i efikasno prijavili na odgovarajuću aplikaciju.

Pre svega, pretpostavlja se da posedujete integrisano razvojno okruženje (IDE) za Java programski jezik, kao i tekstualni uređivač koji može pokrenuti JavaScript projekte kao i instaliran nodejs. Pored toga, trebalo bi da imate radni okvir koji omogućava podizanje MySQL baze, na primer PHPMyAdmin. PHPMyAdmin se lako instalira uz pomoć XAMPP-a. Ako koristite Windows operativni sistem, takođe biste trebali instalirati Git kako biste klonirali projekte.

Projekti se nalaze na Github-u:

* <https://github.com/yxngalex/fe-fitness-app>
* <https://github.com/yxngalex/fitness-app>

Preuzmite ih lokalno na vaš računar klikom na zeleno dugme 'Code'. Zatim, u okviru SSH taba, kopirajte link koji ćete kasnije nalepiti u konzolu uz pomoć **git clone** komande koja bi trebalo da izgleda ovako:

git clone git@github.com:yxngalex/fe-fitness-app.git



kao i

git clone git@github.com:yxngalex/fitness-app.git



Pratite uputstva u konzoli, a kada završite, možete lokalno pristupiti projektima. Baza je exportovana i nalazi se u direktorijumu “***database***”' unutar backend projekta. Importujte bazu u vaše radno okruženje, kao što je XAMPP, i pokrenite server. U datoteci “***application.properties***” na backendu, možete podesiti bazu sa vašim podacima, uključujući URL do baze, port, korisničko ime i lozinku. Ako su svi parametri pravilno postavljeni, možete pokrenuti backend.

Na frontendu, jedino što je potrebno uraditi je izvršiti “**npm install**” kako bi se instalirale sve zavisnosti. Kada se ovaj proces završi, izvršite “**npm run dev**”.

Ukoliko su svi koraci ispravno izvršeni, trebalo bi da imate pristup web sajtu na <http://localhost:5173/> linku.

# Zaključak

Završni rad predstavljа sveobuhvatnu analizu i razvoj fitness aplikacije koja pruža širok spektar funkcionalnosti, uključujući mogućnost ručnog i automatskog kreiranja vežbi, praćenje ishrane, automatsko generisanje *BMI*-a i *BMR*-a. Ona ne samo što omogućava korisnicima personalizaciju vežbi i praćenje ishrane, već i olakšava put ka postizanju njihovih zdravstvenih ciljeva.

Proces razvoja ove aplikacije odvijao se kroz pažljivo proučavanje i implementaciju modernih tehnologija i softverskih praksi, uz dodatni fokus na detaljno proučavanje principa zdravog života, raznolikih tipova treninga i generalnog fokusa na fitnessu. Uzimajući u obzir važnost integrisanja relevantnih informacija o zdravom životu i treninzima, ovaj dodatni aspekt omogućava aplikaciji da pruži sveobuhvatan pristup korisnicima u postizanju njihovih zdravstvenih i fitnes ciljeva. Integracija sa relevantnim znanjem o treninzima i zdravom životu takođe doprinosi unapređenju korisničkog iskustva, pružajući vredne smernice i informacije koje podržavaju celokupni wellness pristup.

Analizirane su ključne komponente poput *REST API*-ja, *MVC* arhitekture i korišćenja popularnih okvira poput *ReactJS*-a i *Spring Boot*-a. Kroz detaljno istraživanje arhitekture projekta, obrađivanje podataka i prikaz informacija na korisničkom portalu, dobili smo uvid u kompleksnost i funkcionalnost ove aplikacije.

Tokom rada na ovom projektu, sticanje znanja o različitim tehnikama i tehnologijama, kao i o fitnessu, generalnom zdravom načinu života i ishrani, bilo je izazovno, ali dragoceno iskustvo. Fleksibilnost u učenju i istraživanju bila je ključna u postizanju uspeha u razvoju ove napredne veb aplikacije.

Napredni koraci u daljem razvoju mogu uključivati dodavanje personalizovanih preporuka i unapređenje opcija analize podataka. Takođe, proširenje podrške na dodatne platforme i uređaje može dalje unaprediti dostupnost aplikacije. Kroz stalno praćenje povratnih informacija korisnika i brzo prilagođavanje promenama u tehnologiji, ova fitness aplikacija ima potencijal da ostane relevantna i konkurentna u dinamičnom svetu digitalnih rešenja.

# Literatura

1. <http://lams.metropolitan.ac.rs:8080/lams/index.do> [Pristupljeno: 11.09.2023]
2. <https://library.ku.ac.ke/wp-content/downloads/2011/08/Bookboon/IT,Programming%20and%20Web/an-introduction-to-relational-database-theory.pdf> [Pristupljeno: 11.09.2023]
3. <https://www.ibm.com/docs/en/mam/7.6.0?topic=structure-database-relationships> [Pristupljeno: 11.09.2023]
4. IT350 - Baze podataka, autorizovani nastavni materijali – Univerzitet Metropolitan Beograd; (Lekcije 01 - 15) [Pristupljeno: 11.09.2023]
5. <https://stackoverflow.com/questions/4048151/what-are-the-options-for-storing-hierarchical-data-in-a-relational-database> [Pristupljeno: 11.09.2023]
6. <http://prof.mau.ac.ir/images/Uploaded_files/Clean%20Architecture_%20A%20Craftsman%E2%80%99s%20Guide%20to%20Software%20Structure%20and%20Design-Pearson%20Education%20(2018)%5B7615523%5D.PDF> [Pristupljeno: 11.09.2023]
7. <https://refactoring.guru/design-patterns> [Pristupljeno: 11.09.2023]
8. <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/> [Pristupljeno: 11.09.2023]
9. <https://react.dev/learn> [Pristupljeno: 11.09.2023]
10. <https://savvycomsoftware.com/fitness-app-development/> [Pristupljeno: 11.09.2023]
11. SE311 - Projektovanje i arhitektura softvera, autorizovani nastavni materijali – Univerzitet Metropolitan Beograd; (Lekcije 01 - 15) [Pristupljeno: 11.09.2023]
12. IT355 - Veb sistemi 2, autorizovani nastavni materijali – Univerzitet Metropolitan Beograd; (Lekcije 01 - 15) [Pristupljeno: 11.09.2023]