Razvoj RESTful web aplikacije za praćenje troškova korištenjem Djanga i Reacta

Car, Marin

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Razvoj RESTful web aplikacije za praćenje troškova korištenjem Djanga i Reacta

Rijeka, rujan 2024.

Marin Car

0069082973

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Razvoj RESTful web aplikacije za praćenje troškova korištenjem Djanga i Reacta

Mentor: Izv. prof. dr. sc. Marko Gulić

Rijeka, rujan 2024.

Marin Car

0069082973

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 04.04.2024.

Zavod: Zavod za računarstvo

Predmet: Razvoj web aplikacija

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Marin Car (0069082973)**

Studij: Sveučilišni prijediplomski studij računarstva (2035)

Zadatak: **Razvoj RESTful web aplikacije za praćenje troškova korištenjem Djanga i Reacta / Development of a RESTful web application for budget management using Django and React**

Opis zadatka:

Razviti RESTful web aplikaciju za praćenje svakodnevnih troškova. Aplikacija omogućuje funkcionalnosti dodavanja, brisanja i mijenjanja troškova i prihoda prema svojim potrebama. Dodatno, aplikacija omogućuje grafički prikaz troškova, pružajući korisnicima vizualni uvid u njihove financijske obrasce kao i organizaciju troškova i prihoda po kategorijama. Dodatno, aplikacija omogućuje komunikaciju korisnika s ChatGPT-om radi pružanja financijskih savjeta. Također, treba implementirati cjelovitu autentifikaciju unutar spomenute web aplikacije. Za razvoj klijentskog dijela aplikacije treba koristiti React java script radni okvir. Za razvoj poslužiteljskog dijela web aplikacije treba koristiti Django uz PostgreSQL sustav za upravljanje bazom podataka.

Rad mora biti napisan prema Uputama za pisanja diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 22.03.2024.

Predsjednik povjerenstva za
završni ispit:
prof. dr. sc. Miroslav Joler

Mentor:
doc. dr. sc. Marko Gulić

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad

Rijeka, rujan, 2024.

Marin Car

Zahvala

Iskrenu zahvalnost izražavam svima koji su me podržavali tijekom studija i izrade ovog rada: obitelji, prijateljima, kolegama i mentoru.

Sadržaj

1 UVOD

U suvremenom digitalnom dobu, upravljanje osobnim financijama postalo je ključno za postizanje financijske stabilnosti i ostvarivanje dugoročnih ciljeva. U ovome radu predstavljena je inovativna aplikacija za budžetiranje koja je dizajnirana s ciljem pružanja sveobuhvatnog rješenja za praćenje i optimizaciju osobnih financija korisnika. Aplikacija integrira niz naprednih funkcionalnosti koje omogućuju korisnicima učinkovito upravljanje prihodima i rashodima, vizualizaciju financijskih podataka te primanje personaliziranih savjeta.

Ključne funkcionalnosti aplikacije uključuju robustan sustav autentifikacije za nove i postojeće korisnike, koji osigurava sigurnost osobnih podataka. Implementiran je i sustav za resetiranje zaboravljenih lozinki koristeći jednokratne lozinke (engl. *one-time password, skraćeno OTP*), čime se dodatno povećava sigurnost i pristupačnost aplikacije. Korisnici imaju mogućnost dodavanja, uklanjanja i izmjene postojećih unosa prihoda i rashoda, što omogućuje precizno praćenje financijskih tokova. Središnji element aplikacije je glavna nadzorna ploča (engl. *dashboard*) koja pruža vizualnu reprezentaciju financijskih podataka putem raznovrsnih grafikona, omogućujući korisnicima brz i intuitivan pregled njihovog financijskog stanja. Implementiran je i sustav obavijesti koji upozorava korisnike kada njihovi troškovi prekoračuju mjesečni proračun, potičući tako financijsku disciplinu. Za potrebe detaljnije analize i izvještavanja, aplikacija nudi funkcionalnost izvoza podataka o prihodima i rashodima u PDF ili Excel formatu. Inovativni element predstavlja osobni AI asistent, koji, koristeći trenutne podatke o prihodima i rashodima korisnika, pruža personalizirane financijske savjete.

Pozadinski dio (engl. *backend*) aplikacije izgrađen je korištenjem *Django REST* okvira, što omogućuje razvoj skalabilnog i sigurnog *API*-ja. *Django REST* okvir pruža robusnu osnovu za rukovanje *HTTP* zahtjevima, autentifikaciju korisnika i upravljanje bazom podataka, istovremeno olakšavajući implementaciju *RESTful* arhitekture. Korisničko sučelje (engl. *frontend*) razvijeno je pomoću *React* biblioteke, omogućujući stvaranje dinamičkog i interaktivnog korisničkog iskustva. *React*, kao deklarativna *JavaScript* biblioteka za izgradnju korisničkih sučelja, omogućuje efikasno renderiranje komponenti i upravljanje stanjem aplikacije, rezultirajući brzim i responzivnim sučeljem. Podaci korisnika pohranjuju se u *PostgreSQL* bazi podataka, osiguravajući pouzdano i skalabilno rješenje za upravljanje strukturiranim podacima. *PostgreSQL*, kao napredni

sustav za upravljanje relacijskim bazama podataka otvorenog koda, pruža robustan temelj za pohranu i dohvaćanje financijskih podataka korisnika.

Ovaj rad detaljno će istražiti arhitekturu, implementaciju i učinkovitost predstavljene aplikacije za budžetiranje, analizirajući njezin utjecaj na financijsko planiranje i odlučivanje korisnika. Rad je strukturiran u nekoliko ključnih poglavlja koja sustavno obrađuju tematiku razvoja i implementacije web aplikacije za praćenje troškova. Drugo poglavlje pruža detaljan uvid u tehnološke platforme i alate korištene u razvoju aplikacije. U trećem poglavlju predstavlja se iscrpna analiza funkcionalnosti razvijene aplikacije, dok se u četvrtom poglavlju fokus premješta na praktičnu razinu, uključujući reprezentativne primjere programskog koda. Zaključak je iznesen u posljednjem poglavlju.

2 TEHNOLOGIJE

U ovome poglavlju pruža se detaljan uvid u tehnološke platforme i alate korištene u razvoju aplikacije.

2.1 HTML

HTML, što je akronim za *HyperText Markup Language* [1], temeljni je jezik za stvaranje web stranica. Ovaj jezik koristi se za strukturiranje i prezentaciju sadržaja na *World Wide Webu*. HTML nije programski jezik u tradicionalnom smislu, već jezik za označavanje koji definira strukturu i semantiku web dokumenta.

HTML se sastoji od niza elemenata, koji se označavaju pomoću oznaka (eng. *tags*). Ove oznake okružuju različite dijelove sadržaja kako bi ga prikazale na određeni način ili izvršile određenu funkciju. Primjerice, oznaka `<p>` označava odlomak, dok `<h1>` označava naslov prve razine. HTML elementi mogu sadržavati *atribute* koji pružaju dodatne informacije o elementu i mogu modificirati njegovo ponašanje ili izgled.

Kroz godine, HTML je evoluirao kroz nekoliko verzija, pri čemu je trenutno aktualna verzija HTML5. Ova verzija donijela je značajna poboljšanja u smislu semantike, multimedijskih mogućnosti i poboljšane integracije s drugim web tehnologijama. HTML radi u sinergiji s *Cascading Style Sheets* (CSS) za stiliziranje i *JavaScriptom* za dinamičku funkcionalnost, tvoreći tako temelj moderne web tehnologije.

2.2 CSS

CSS (*engl. Cascading Style Sheets*) [2] predstavlja stilski jezik koji se koristi za opisivanje prezentacije dokumenta napisanog u *markup* jeziku, najčešće HTML-u. Ovaj jezik omogućuje odvajanje strukture i sadržaja web stranice od njezina vizualnog prikaza, što značajno unapređuje fleksibilnost i kontrolu u procesu web dizajna.

Funkcionalnost CSS-a temelji se na selektorima i deklaracijama. Selektori identificiraju specifične elemente HTML dokumenta na koje se primjenjuju stilska pravila, dok deklaracije definiraju sama pravila. Svaka deklaracija sastoji se od svojstva i vrijednosti, određujući kako će se odabrani element prikazati u web pregledniku. CSS podržava kaskadnost, što znači da se stilovi mogu nasljeđivati i preklapati prema specificiranim pravilima prioriteta.

CSS pruža širok spektar mogućnosti za kontrolu izgleda web stranica, uključujući definiranje boja, fontova, margina, pozicioniranje elemenata, kao i stvaranje responzivnih dizajna (*engl. responsive designs*) koji se prilagođavaju različitim veličinama zaslona. Napredne funkcionalnosti CSS-a obuhvaćaju *animacije*, *prijelaze* i *transformacije*, omogućujući stvaranje dinamičnih i interaktivnih korisničkih sučelja bez potrebe za skriptnim jezicima.

2.3 JSX

JSX (*engl. JavaScript XML*) [3] je sintaksno proširenje za *JavaScript* koje omogućuje pisanje HTMLsličnog koda unutar *JavaScript* datoteka. Primarno razvijen za *React* biblioteku, JSX pruža deklarativni pristup opisivanju korisničkog sučelja, kombinirajući *JavaScript* s intuitivnošću *markup* jezika.

JSX transformira XML-slične strukture u *JavaScript* objekte koje *React* može interpretirati i renderirati. Ova transformacija se obično izvodi pomoću alata za transpilaciju, rezultirajući čitljivijim i ekspresivnijim kodom od ekvivalentnog čistog *JavaScripta*.

Temeljna značajka JSX-a je koncept komponenti (*engl. components*) kao gradivnih blokova sučelja. Komponente mogu sadržavati druge komponente, stvarajući hijerarhijsku strukturu aplikacije. JSX također omogućuje interpolaciju *JavaScripta* unutar *markup* struktura, pružajući visoku razinu fleksibilnosti u definiranju korisničkog sučelja. Iako je usko povezan s *React* ekosistemom, JSX-ova upotreba nije ograničena samo na *React*, što dodatno učvršćuje njegovu poziciju kao važnog alata u suvremenom web razvoju.

2.4 NPM

NPM (*engl. Node Package Manager*) [4] je standardni upravitelj paketima za *Node.js* okruženje, ključan u ekosustavu modernog web razvoja. Funkcionira kao repozitorij otvorenog koda koji sadrži brojne softverske pakete, pružajući sučelje naredbenog retka za instalaciju, ažuriranje i upravljanje paketima i ovisnostima projekta.

Središnji koncept NPM-a je *package.json* datoteka, koja služi kao manifest projekta, sadržavajući metapodatke, ovisnosti i konfiguracijske postavke. NPM koristi ove informacije za automatizaciju procesa instalacije i upravljanja ovisnostima, olakšavajući reproducibilnost razvojnog okruženja.

NPM podržava semantičko verzioniranje (*engl. semantic versioning*), omogućujući preciznu kontrolu nad verzijama paketa. Također služi kao platforma za objavljivanje i distribuciju *JavaScript* koda, doprinoseći rastu i razvoju *JavaScript* ekosustava i potičući suradnju među programerima.

2.5 React

React [5] je otvorena *JavaScript* biblioteka za izgradnju korisničkih sučelja, razvijena od strane Facebooka. Temelji se na konceptu komponenti (*engl. components*), omogućujući razvoj skalabilnih web aplikacija.

Ključna značajka *Reacta* je virtualni DOM (*engl. Virtual DOM*), koji optimizira proces renderiranja korištenjem algoritma usklađivanja (*engl. reconciliation algorithm*). *React* implementira jednosmjerni tok podataka (*engl. one-way data flow*) kroz stanje (*engl. state*) i svojstva (*engl. props*) komponenti, pojednostavljujući praćenje promjena u aplikaciji.

React se ističe svojom deklarativnom prirodom, gdje programeri definiraju željeno stanje sučelja, a *React* upravlja ažuriranjem i renderiranjem. Ekosustav *Reacta* uključuje alate poput *Redux*a i *React Router*a, a podržava i razvoj mobilnih aplikacija putem *React Native* okvira.

Zbog svoje fleksibilnosti i efikasnosti, *React* je postao vodeći izbor za razvoj modernih web aplikacija, privlačeći veliku zajednicu programera i kontinuirano evoluirajući.

2.5.1 Axios

Axios [6] je JavaScript biblioteka otvorenog koda za izvršavanje HTTP zahtjeva u web aplikacijama, dizajnirana za uporabu u pregledniku i Node.js okruženju. Temelji se na Promise-ima, omogućujući elegantno rukovanje asinkronim operacijama, a ključne funkcionalnosti uključuju automatsku transformaciju JSON podataka, interceptore (*engl. interceptors*) za modifikaciju zahtjeva i odgovora, te poništavanje zahtjeva.

Implementirajući izomorfni (*engl. isomorphic*) dizajn, Axios pruža konzistentno iskustvo u različitim okruženjima. Zbog svoje jednostavnosti i fleksibilnosti, široko je prihvaćen u zajednici web programera kao alternativa nativnom *Fetch API*-ju ili starijim

bibliotekama, što ga čini popularnim izborom za upravljanje mrežnim zahtjevima u modernim web aplikacijama.

2.5.2 Material UI

Material UI [7] je popularni *React* okvir otvorenog koda koji implementira principe dizajna temeljenog na *Material Design* sustavu, razvijenom od strane Googlea. Ovaj okvir omogućuje izradu modernih, estetski usklađenih i responzivnih korisničkih sučelja za web aplikacije, koristeći gotove komponente kao što su gumbi, kartice, obrasci, navigacija i tipografija. *Material UI* komponente dolaze unaprijed stilizirane prema smjernicama *Material Design-a*, pružajući dosljedan vizualni identitet, dok je istovremeno omogućena prilagodba kako bi se zadovoljile specifične potrebe projekta. Korištenje ovog okvira značajno ubrzava razvoj aplikacija, jer smanjuje potrebu za pisanjem prilagođenog koda za dizajn i omogućuje jednostavnu integraciju komponenti.

Jedna od ključnih prednosti *Material UI-a* je njegova fleksibilnost i proširivost. Programeri mogu lako mijenjati zadane stilove korištenjem *CSS-in-JS* pristupa, što omogućuje dinamičku prilagodbu stilova unutar *JavaScript* koda.

Pored toga, *Material UI* integrira sustav mreže (*engl. grid system*) i responzivnog dizajna, omogućujući optimalnu prilagodbu aplikacija različitim veličinama zaslona. Ovaj okvir također nudi odličnu dokumentaciju i široku zajednicu korisnika, što olakšava rješavanje problema i proširivanje funkcionalnosti. *Material UI* predstavlja idealan izbor za razvoj složenih web aplikacija s modernim i dosljednim korisničkim sučeljem.

2.5.3 ReCharts

Recharts [8] je biblioteka koja se temelji na *React* ekosustavu i koristi komponente specifične za *React* za izradu vizualizacija podataka. Ova knjižnica nudi deklarativan način definiranja grafikona, što znači da se grafikoni grade pomoću JSX sintakse unutar *React* komponenti. To omogućuje jednostavno upravljanje stanjem i ažuriranje grafova unutar *React* aplikacija. *Recharts* je fleksibilan, proširiv i dobro integriran s *React* aplikacijama, ali se oslanja na *React* okruženje, što znači da se ne može koristiti izvan njega. Njegova snaga leži u jednostavnosti i prilagodljivosti unutar *React* arhitekture, s naglaskom na interaktivnost i modularnost.

2.6 Python

Python [9] je višenamjenski programski jezik visoke razine, poznat po čitljivoj sintaksi i jednostavnosti korištenja. Razvijen 1991. godine, ističe se filozofijom *"The Zen of Python"*, naglašavajući važnost jasnog koda.

Jezik podržava višestruke programske paradigme, uključujući objektno-orijentirano (*engl. object-oriented*) i funkcionalno programiranje. Opsežna standardna biblioteka i *PyPI* (*engl. Python Package Index*) pružaju širok spektar funkcionalnosti, posebice u područjima znanstvenog računarstva (*engl. scientific computing*) i strojnog učenja (*engl. machine learning*).

Python koristi dinamičko tipiziranje (*engl. dynamic typing*) i automatsko upravljanje memorijom (*engl. automatic memory management*), olakšavajući razvoj, ali potencijalno utječući na performanse. Aktivna zajednica doprinosi kontinuiranom razvoju i širenju primjene jezika u različitim domenama, od web razvoja do znanstvenih istraživanja.

2.7 Django

Django [10] je visoko razvijeni web okvir (*engl. web framework*) otvorenog koda, napisan u *Pythonu*. Slijedi princip *batteries included*, pružajući sveobuhvatan skup alata za brz razvoj sigurnih web aplikacija.

Temelji se na *MVT* (*engl. Model-View-Template*) arhitekturi, varijaciji *MVC* (*engl. Model-View-Controller*) obrasca. Njegov *ORM* (*engl. Object-Relational Mapping*) sustav omogućuje apstrakciju baze podataka i rad s podacima kroz *Python* objekte.

Ključne značajke uključuju automatski generirano administrativno sučelje i podršku za internacionalizaciju. *Django*ova sveobuhvatnost, skalabilnost i aktivna zajednica čine ga popularnim izborom za razvoj web aplikacija različitih veličina i kompleksnosti.

2.8 Django REST Framework

Django REST Framework (DRF) [11] je moćna i fleksibilna nadogradnja *Django* web okvira, specijalizirana za izgradnju *Web API-ja* (*engl. Web Application Programming Interfaces*). Ovaj alat omogućuje programerima brz razvoj robusnih, skalabilnih i sigurnih *RESTful API-ja* (*engl. Representational State Transfer APIs*), istovremeno zadržavajući eleganciju i jednostavnost korištenja karakterističnu za *Django*.

U srži *DRF*-a nalazi se koncept *serializer*a, koji pruža mehanizam za pretvaranje složenih podatkovnih tipova, poput *Django* modela i *QuerySet*ova, u formate pogodne za *JSON*, *XML* ili druge tipove sadržaja. *Serializer*i također omogućuju deserijalizaciju, pretvarajući primljene podatke natrag u složene tipove, olakšavajući validaciju i obradu ulaznih podataka.

DRF implementira *class-based views* i *viewsets*, koji pružaju visoku razinu apstrakcije za uobičajene operacije *API*-ja. Ovi koncepti omogućuju programerima definiranje logike za različite HTTP metode (GET, POST, PUT, DELETE) na konzistentan i ponovljiv način. Uz to, DRF nudi generičke *views* i *mixins*, koji dodatno pojednostavljuju implementaciju standardnih CRUD (*engl. Create, Read, Update, Delete*) operacija.

Značajna prednost *DRF*-a je njegova podrška za različite vrste autentifikacije i autorizacije, uključujući *token-based* autentifikaciju, *OAuth* i *JWT* (*engl. JSON Web Tokens*). Ovo omogućuje razvoj sigurnih *API*-ja s granularnom kontrolom pristupa.

DRF također pruža bogato, interaktivno API pregledavanje (*engl. API browsing*) putem web sučelja, što znatno olakšava testiranje i dokumentiranje *API*-ja tijekom razvoja. Ova značajka, uz podršku za automatsko generiranje shema i dokumentacije, čini *DRF* izuzetno korisnim alatom u razvoju modernih web aplikacija.

2.9 OpenAI

OpenAI Python [12] biblioteka je službeno sučelje za programiranje aplikacija koje omogućuje integraciju naprednih *OpenAI* modela umjetne inteligencije u *Python* aplikacije. Ova biblioteka omogućuje razvojnim inženjerima jednostavan i učinkovit pristup širokom spektru jezičnih modela, kao i drugih usluga umjetne inteligencije *(engl. Artificial Intelligence,* skraćeno *AI)* koje *OpenAI* nudi. S naglaskom na jednostavnost

korištenja, pruža alat koji olakšava implementaciju složenih zadataka obrade prirodnog jezika, automatizacije kodiranja i drugih *AI* funkcionalnosti.

Ključne funkcionalnosti uključuju generiranje teksta (*engl. text generation*), omogućujući stvaranje koherentnog i relevantnog teksta koristeći modele poput GPT-3 i GPT-4, te dovršavanje koda (*engl. code completion*) koje automatski generira programski kod u različitim jezicima. Biblioteka također podržava *embeddings,* koji pretvaraju tekst u vektorske reprezentacije, korisne za semantičku pretragu i analizu sličnosti, kao i *fine-tuning*, što omogućuje prilagodbu modela specifičnim potrebama. Implementacija asinkronog programiranja (*engl. asynchronous programming*) poboljšava učinkovitost pri rukovanju mrežnim zahtjevima, dok su ugrađeni mehanizmi za rukovanje pogreškama i ograničenje protoka (*engl. rate limiting*) ključni za optimalno korištenje API-ja.

2.10 PostgreSQL

PostgreSQL [13] je napredni objektno-relacijski sustav za upravljanje bazama podataka (*engl. object-relational database management system*, ORDBMS) otvorenog koda. Ističe se robusnom arhitekturom i podrškom za napredne funkcionalnosti poput složenih upita, stranih ključeva (*engl. foreign keys*), okidača (*engl. triggers*) i transakcijske cjelovitosti (*engl. transactional integrity*).

Podržava složene podatkovne strukture, uključujući korisnički definirane tipove (*engl. user-defined types*) i nasljeđivanje tablica (*engl. table inheritance*). *PostgreSQL* je visoko usklađen sa *SQL standardom,* implementirajući napredne značajke poput rekurzivnih upita (*engl. recursive queries*) i analitičkih funkcija (*engl. window functions*).

U pogledu skalabilnosti, *PostgreSQL* nudi replikaciju (*engl. replication*) i paralelno izvršavanje upita (*engl. parallel query execution*). Zbog svoje robusnosti i fleksibilnosti, *PostgreSQL* je popularan izbor za širok raspon primjena, od malih projekata do *enterprise* sustava, posebice u područjima koja zahtijevaju visoku pouzdanost i integritet podataka.

2.11 pgAdmin

pgAdmin [14] je sveobuhvatna platforma otvorenog koda za administraciju i razvoj *PostgreSQL* baza podataka. Pruža grafičko sučelje (*engl. Graphical User Interface*, GUI) koje olakšava interakciju s bazama podataka.

Ključne značajke uključuju preglednik objekata (*engl. Object Browser*) za navigaciju kroz objekte baze podataka i napredni *SQL editor* s automatskim dovršavanjem (*engl. auto-completion*). *pgAdmin* podržava administrativne funkcije poput upravljanja korisnicima (*engl. user management*) i nadzora performansi (*engl. performance monitoring*).

Alat nudi mogućnosti za modeliranje podataka (*engl. data modeling*) i izvoz/uvoz podataka (*engl. data import/export*). Novije verzije (*pgAdmin 4*) razvijene su kao web aplikacija, podržavajući desktop i serverske načine rada.

Zbog svoje sveobuhvatnosti i jednostavnosti korištenja, *pgAdmin* je postao neizostavni alat u *PostgreSQL* ekosustavu, široko prihvaćen u različitim okruženjima za upravljanje bazama podataka.

2.12 Visual Studio Code

Visual Studio Code (VS Code) [15] je besplatni, višeplatformski (*engl. cross-platform*) uređivač izvornog koda razvijen od strane Microsofta. Temelji se na *Electron* okviru i ističe se modularnim dizajnom i ekstenzivnim sustavom proširenja (*engl. extension system*).

Ključne značajke uključuju napredno sintaksno isticanje (*engl. syntax highlighting*), inteligentno automatsko dovršavanje koda (*engl. IntelliSense*) i podršku za debugiranje (*engl. debugging*). *VS Code* pruža snažnu integraciju s sustavima za kontrolu verzija (*engl. version control systems*), posebice s *Git*-om.

Uređivač podržava razvoj u različitim programskim jezicima (*engl. multi-language development*) i implementira koncept radnih prostora (*engl. workspaces*) za efikasno upravljanje projektima. Dodatno, *VS Code* omogućuje udaljeno razvijanje (*engl. remote development*).

Zbog svoje fleksibilnosti i performansi, *Visual Studio Code* postao je popularan alat za razvoj softvera u raznim okruženjima, od akademskih do profesionalnih.

2.13 Git

Git [16] je distribuirani sustav za kontrolu verzija (*engl. distributed version control system*, skraćeno DVCS) otvorenog koda, dizajniran za praćenje promjena u izvornom kodu tijekom razvoja softvera. Razvio ga je Linus Torvalds 2005. godine s ciljem upravljanja razvojem *Linux* kernela, a od tada je postao standard u industriji softverskog inženjerstva.

Temeljna arhitektura *Git*-a počiva na konceptu spremišta (*engl. repository*), koje sadrži cjelokupnu povijest projekta. Svako spremište uključuje kompletnu kopiju projekta, omogućujući programerima rad bez stalne mrežne veze. *Git* koristi snimke (*engl. snapshots*) umjesto razlika između datoteka, što rezultira bržim operacijama i omogućuje efikasno grananje (*engl. branching*) i spajanje (*engl. merging*).

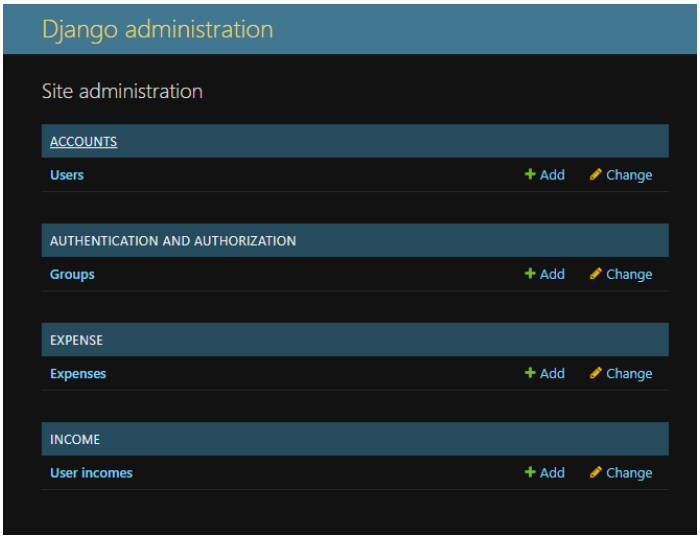
3 WEB APLIKACIJA ZA PRAĆENJE TROŠKOVA

Ovo poglavlje pruža sveobuhvatan pregled i detaljnu analizu ključnih funkcionalnosti aplikacije. Svaka funkcionalnost bit će temeljito objašnjena, s naglaskom na njezinu ulogu i implementaciju u cjelokupnom sustavu.

3.1 Django administracijsko sučelje

U slici 3.1 prikazano je administrativno sučelje koje dolazi ugrađeno s *Django* okvirom. Ova značajka, poznata kao *Django Admin*, predstavlja moćan alat za upravljanje podacima aplikacije bez potrebe za izradom posebnog korisničkog sučelja. Izabrana je standardna implementacija ovog sučelja.

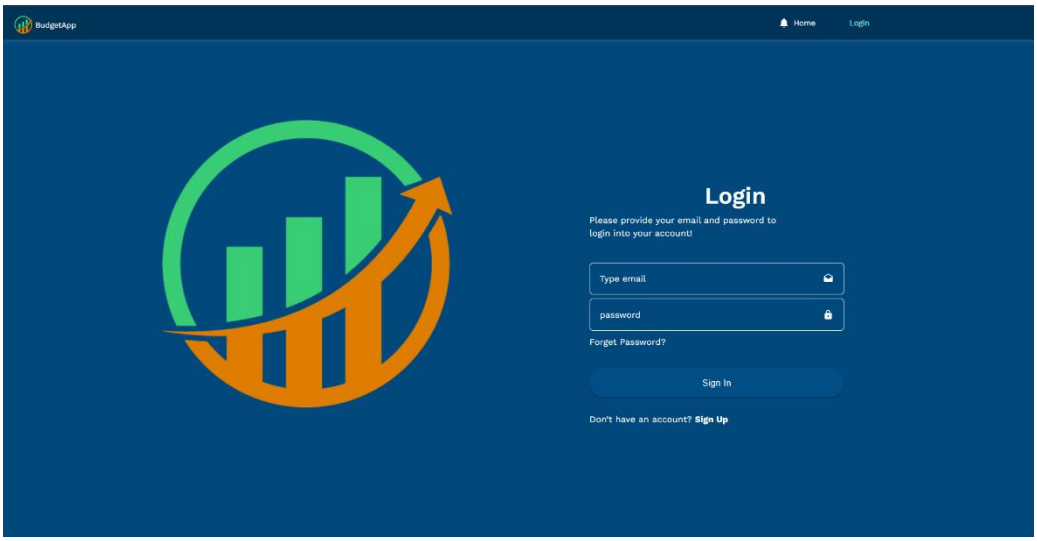
Django Admin sučelje automatski se generira na temelju definiranih modela u aplikaciji. U ovom slučaju, vidljivo je da su kreirani modeli za korisnike, grupe, troškove i prihode, što odgovara strukturi aplikacije za osobno budžetiranje. Ova ugrađena funkcionalnost značajno ubrzava razvoj aplikacije, omogućujući programerima da se usredotoče na poslovnu logiku, dok *Django* preuzima brigu o administrativnim zadacima. Važno je napomenuti da, iako je ovo sučelje iznimno korisno tijekom razvoja i za administrativne zadatke, obično nije namijenjeno krajnjim korisnicima aplikacije.



Slika 3.1. Django administracijsko sučelje

3.2 Početni zaslon aplikacije

Na slici 3.2 prikazan je autentifikacijski zaslon aplikacije. Obrazac uključuje polja za unos e-pošte i lozinke, opciju za zaboravljenu lozinku te gumb za prijavu (*engl. Sign In*). Ispod obrasca nalazi se poveznica za registraciju novih korisnika (*engl. Sign Up*). Ovakav dizajn sučelja omogućuje korisnicima jednostavan pristup svojim financijskim podacima, istovremeno naglašavajući važnost sigurnosti kroz proces autentifikacije.

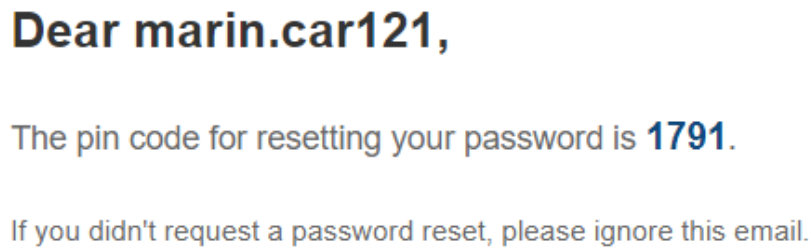


Slika 3.2 Početni zaslon aplikacije

3.3 Zaboravljena lozinka i OTP verifikacija

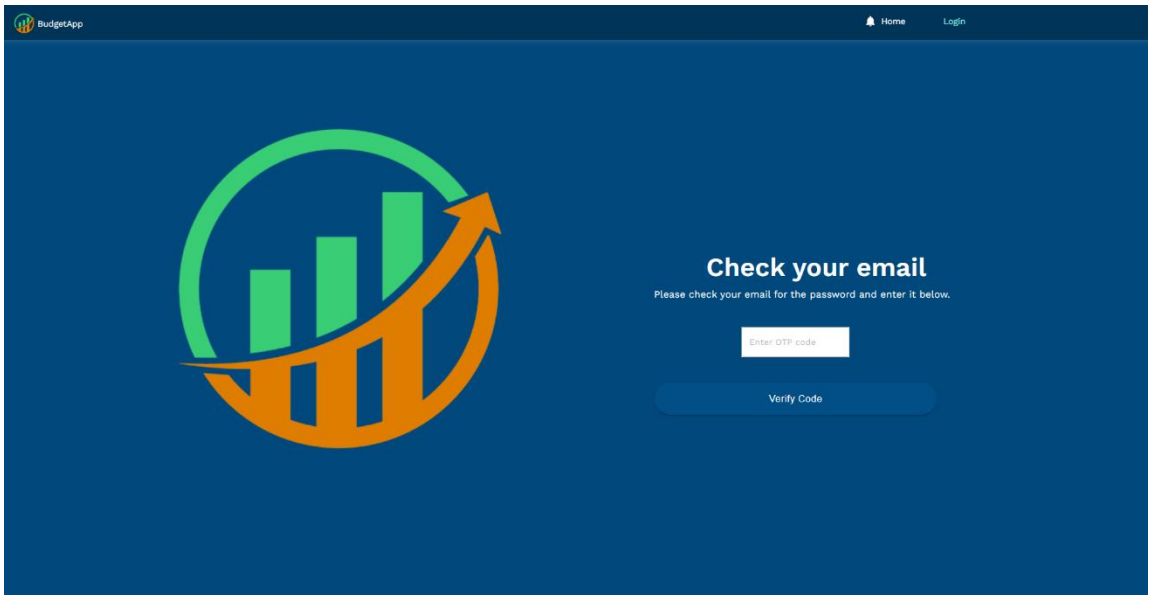
Nakon što je korisnik navigirao na *Forgot Password?* poveznicu pokreće se postupak OTP verifikacije.

OTP verifikacija, ili verifikacija putem jednokratne lozinke (*engl. One-Time Password*), predstavlja napredni mehanizam autentifikacije koji značajno unapređuje sigurnost digitalnih sustava. Ovaj proces temelji se na generiranju jedinstvenog, vremenski ograničenog koda koji se korisniku dostavlja putem sekundarnog komunikacijskog kanala (npr. SMS, e-pošta ili mobilna aplikacija). U slici 3.3 prikazan je primjer email poruke sa generiranim kodom.



Slika 3.3 Primjer maila za OTP verifikaciju

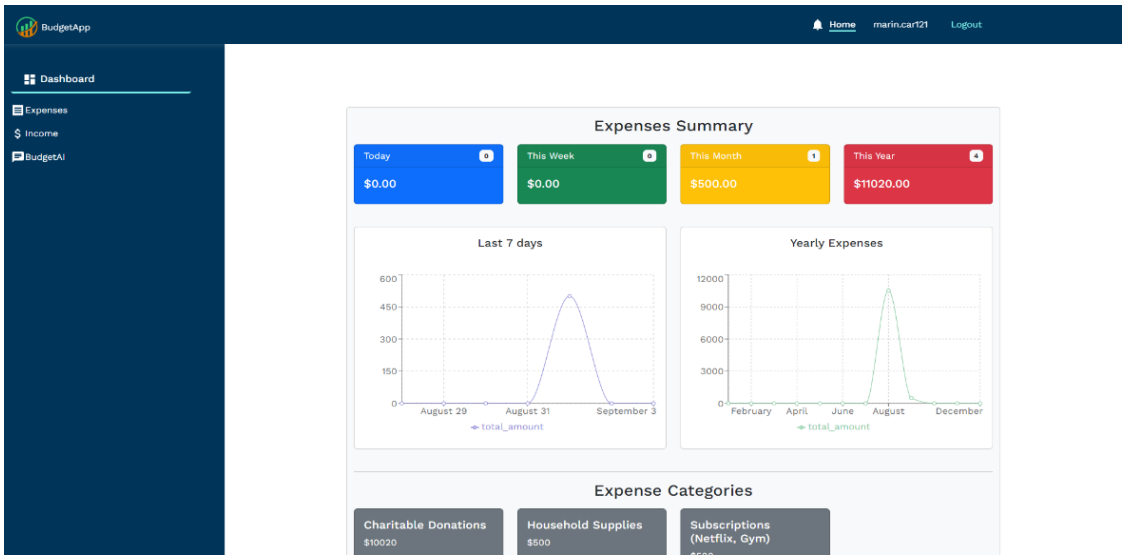
Korisnik zatim unosi primljeni kod u predviđeno polje, nakon čega sustav provjerava njegovu valjanost. Na slici 3.4 prikazan je zaslon za unos istoga. Implementacija OTP verifikacije dodaje dodatni sloj sigurnosti, smanjujući rizik od neovlaštenog pristupa čak i u slučaju kompromitiranja primarnih pristupnih podataka. Ovaj mehanizam nalazi široku primjenu u zaštiti osjetljivih operacija poput financijskih transakcija ili pristupa povjerljivim informacijama, zahtijevajući pažljivo balansiranje između sigurnosnih zahtjeva i korisničkog iskustva.



Slika 3.4 Zaboravljena lozinka i OTP verifikacija

3.4 Glavno korisničko sučelje

Nakon uspješne autentifikacije, korisnik je upućen na glavno sučelje aplikacije (engl. *Dashboard)*. Sučelje (slika 3.5) je strukturirano u nekoliko ključnih segmenata koji pružaju sveobuhvatan pregled korisnikovih financijskih aktivnosti. U gornjem dijelu sučelja nalazi se navigacijska traka s opcijama za pristup početnoj stranici, korisničkom profilu te odjavljivanju iz sustava. Prisutna je i ikona zvona s indikatorom obavijesti, u slučaju da je korisnik prešao zadani budžet.



Slika 3.5 Glavno korisničko sučelje – Dashboard

Lijevi dio sučelja zauzima vertikalna navigacijska, koja sadrži četiri primarne navigacijske opcije: *Dashboard*, *Expenses* (Troškovi), *Income* (Prihodi) i *BudgetAI*, gdje je implementirana komunikacija sa *ChatGPT*-om.

Centralni dio sučelja posvećen je prikazu financijskih podataka. Na vrhu se nalazi sažetak troškova (engl. *Expenses Summary*) prikazan kroz četiri interaktivne kartice koje predstavljaju troškove za različita vremenska razdoblja: danas, ovaj tjedan, ovaj mjesec i ovu godinu. Svaka kartica vizualno je diferencirana bojom i prikazuje ukupan iznos troškova te broj transakcija. Stranica također nudi nekoliko grafičkih pregleda podataka prikazanih na slici 3.6.

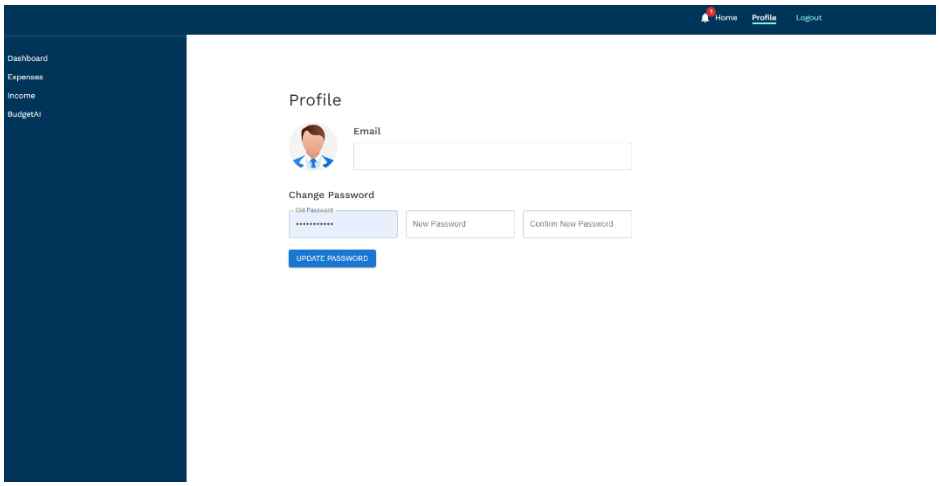
Slika 3.6 Grafički prikaz podataka

Ispod sažetka nalaze se tri grafička prikaza: tjedni i godišnji pregled troškova i prihoda, te vizualizaciju po kategorijama. Ovi grafikoni implementirani su koristeći napredne biblioteke za vizualizaciju podataka, *ReCharts*.

Ovakvo sučelje demonstrira efektivnu primjenu *React* komponenti za stvaranje interaktivnog i informativnog korisničkog iskustva, s naglaskom na vizualnu prezentaciju kompleksnih financijskih podataka kroz intuitivne grafičke elemente.

3.5 Sučelje profila

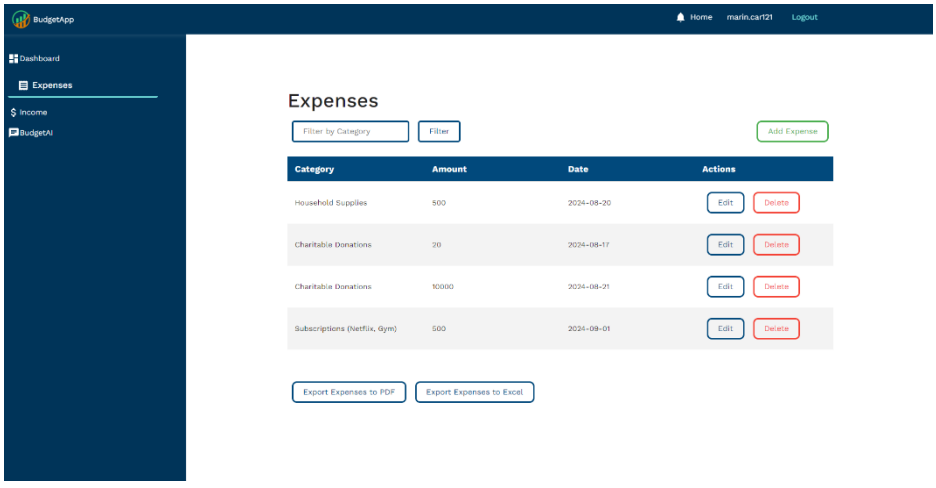
Na slici 3.7 prikazano je *Profile* sučelje, gdje se korisniku pruža mogućnost promijene lozinke. Središnji dio ekrana posvećen je korisničkom profilu. Prikazana je stilizirana ikona korisnika, polje za unos e-pošte te sekcija za promjenu lozinke. Ova sekcija uključuje polja za unos stare lozinke, nove lozinke i potvrdu nove lozinke, uz gumb za potvrdu promjene (engl. *UPDATE PASSWORD*). Ovakav dizajn omogućuje korisnicima jednostavno upravljanje svojim podacima za prijavu, naglašavajući važnost sigurnosti korisničkog računa u aplikaciji.



Slika 3.7 Profile sučelje

3.6 Sučelje Expenses

Na slici broj 3.8 prikazan je ekran za upravljanje troškovima. Sučelje *Expenses* posvećeno je prikazu i upravljanju troškovima. Na vrhu se nalaze opcije za filtriranje po kategoriji i dodavanje novog troška (*engl. Add Expense*). Ispod toga je tablica s prikazom troškova koja sadrži stupce za kategoriju, iznos, datum i akcije. Prikazani su različiti troškovi poput kućanskih potrepština, donacija u dobrotvorne svrhe i pretplata. Za svaki trošak ponuđene su opcije uređivanja (*engl. Edit*) i brisanja (*engl. Delete*).

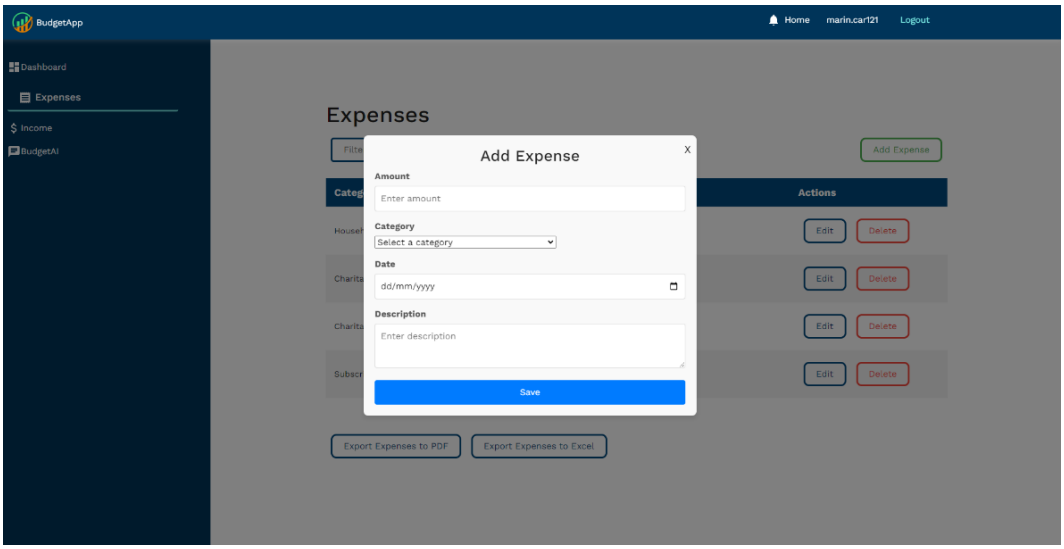


Slika 3.8 Sučelje Expenses

3.6.1 Modalni prozor *Add Expense*

Na slici 3.9 prikazan je modalni prozor za dodavanje troška (engl. *Add Expense)*. Ovaj modalni prozor pojavljuje se iznad zatamnjenog prikaza stranice s postojećim troškovima, čime se korisniku jasno stavlja do znanja da unosi nove podatke. Unutar modalnog prozora nalazi se obrazac za unos detalja o trošku, koji uključuje nekoliko ključnih polja. Prvo polje, omogućuje unos iznosa troška *(engl. Amount)*, dok padajući izbornik Category nudi korisniku opcije za odabir odgovarajuće kategorije troška.

Dalje, tu je polje *Date*, formatirano na način *dd/mm/yyyy*, uz dodatnu ikonu kalendara koja korisnicima olakšava odabir točnog datuma. Na kraju, nalazi se tekstualno polje *Description* gdje korisnik može unijeti kratki opis ili dodatne detalje o trošku. Na dnu modalnog prozora postavljen je gumb *engl. Save*, koji omogućuje korisniku da potvrdi i spremi unos novog troška.



Slika 3.9 „Add Expense“ prozor

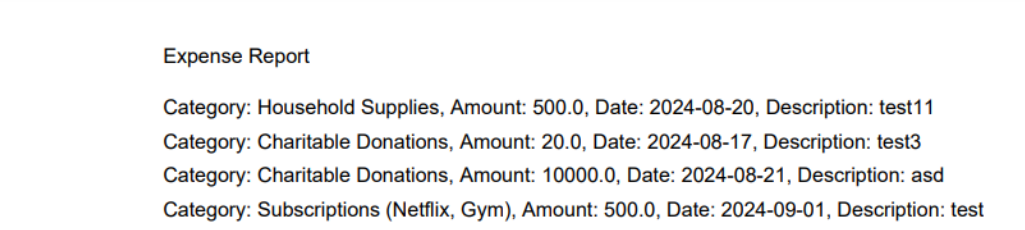
Ovakav dizajn omogućuje korisniku da brzo i jednostavno doda novi trošak bez napuštanja glavnog pregleda troškova. Modalni prozor efikasno fokusira pažnju korisnika na zadatak unosa novog troška, istovremeno pružajući kontekst trenutnog stanja aplikacije u pozadini.

U slučaju da korisnik odabere funkciju *Edit* na nekom od već prisutnih troškova, pokazati će mu se isti modalni prozor no polja će već biti popunjena sa njihovim trenutnim vrijednostima.

3.6.2 Preuzimanje PDF i Excel datoteke

Na dnu *Expenses* sučelja nalaze se gumbi za izvoz troškova u PDF i Excel formate, što omogućuje korisnicima dodatnu analizu i arhiviranje financijskih podataka.

Na slici 3.10 prikazan je primjer preuzete PDF datoteke za unesene troškove.


Expense Report

Category: Household Supplies, Amount: 500.0, Date: 2024-08-20, Description: test11
Category: Charitable Donations, Amount: 20.0, Date: 2024-08-17, Description: test3
Category: Charitable Donations, Amount: 10000.0, Date: 2024-08-21, Description: asd
Category: Subscriptions (Netflix, Gym), Amount: 500.0, Date: 2024-09-01, Description: test

Slika 3.10 Primjer PDF datoteke troškova

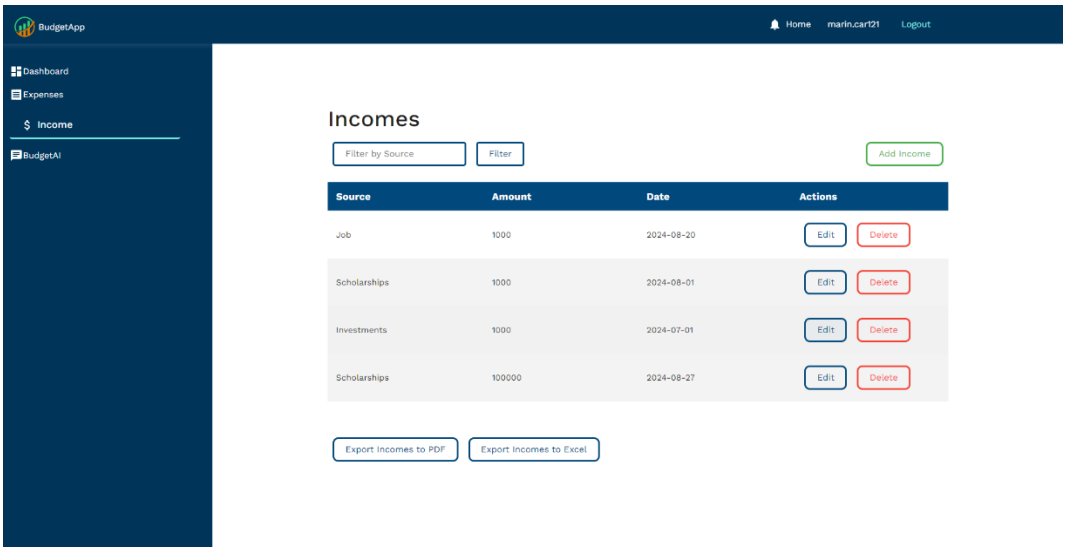
Na slici 3.11 prikazan je primjer preuzete Excel datoteke za unesene troškove.

	A	B	C	D
1	Category	Amount	Date	Description
2	Household Supplies	500	2024-08-20	test11
3	Charitable Donations	20	2024-08-17	test3
4	Charitable Donations	10000	2024-08-21	asd
5				

Slika 3.11 Primjer Excel datoteke troškova

3.7 Sučelje *Income*

Na slici broj 3.12 prikazan je ekran za upravljanje prihodima u aplikaciji. Važno je naglasiti da je funkcionalnost i dizajn stranice za prihode (*engl. Incomes*) u suštini identičan stranici za troškove (*engl. Expenses*) koju smo prethodno analizirali.



Slika 3.12 Sučelje Income

Zbog ove sličnosti, nije potrebno detaljno opisivati svaki element sučelja kao što je to učinjeno za stranicu troškova. Umjesto toga, možemo se usredotočiti na ključne razlike i specifičnosti vezane uz upravljanje prihodima.

Ovakav pristup u dizajnu korisničkog sučelja, gdje se koristi konzistentan izgled i funkcionalnost za slične operacije poput upravljanja troškovima i prihodima, ima nekoliko prednosti. Prvenstveno, poboljšava korisničko iskustvo omogućujući korisnicima da brzo nauče i navigiraju kroz različite dijelove aplikacije. Kada je dizajn sučelja dosljedan, korisnici brže shvaćaju kako koristiti različite funkcionalnosti, što smanjuje vrijeme potrebno za prilagodbu aplikaciji.

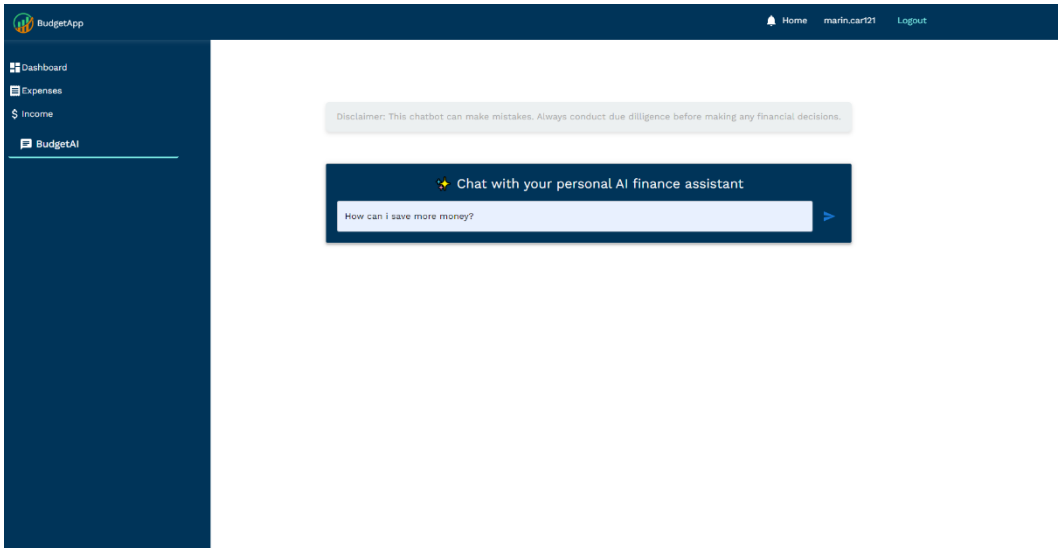
Osim toga, ovaj pristup olakšava održavanje i razvoj aplikacije jer se slični obrasci koda mogu ponovno koristiti. Na taj način, programeri mogu brže i učinkovitije implementirati nove funkcionalnosti ili prilagoditi postojeće. Konačno, konzistentan dizajn osigurava vizualnu koherentnost cijele aplikacije, što doprinosi profesionalnom izgledu i osjećaju, te povećava povjerenje korisnika u stabilnost i kvalitetu aplikacije.

3.8 Sučelje *BudgetAI*

Na slici broj 3.13 prikazan je ekran za interakciju s umjetnom inteligencijom u aplikaciji. U središnjem dijelu ekrana ističu se dva ključna elementa:

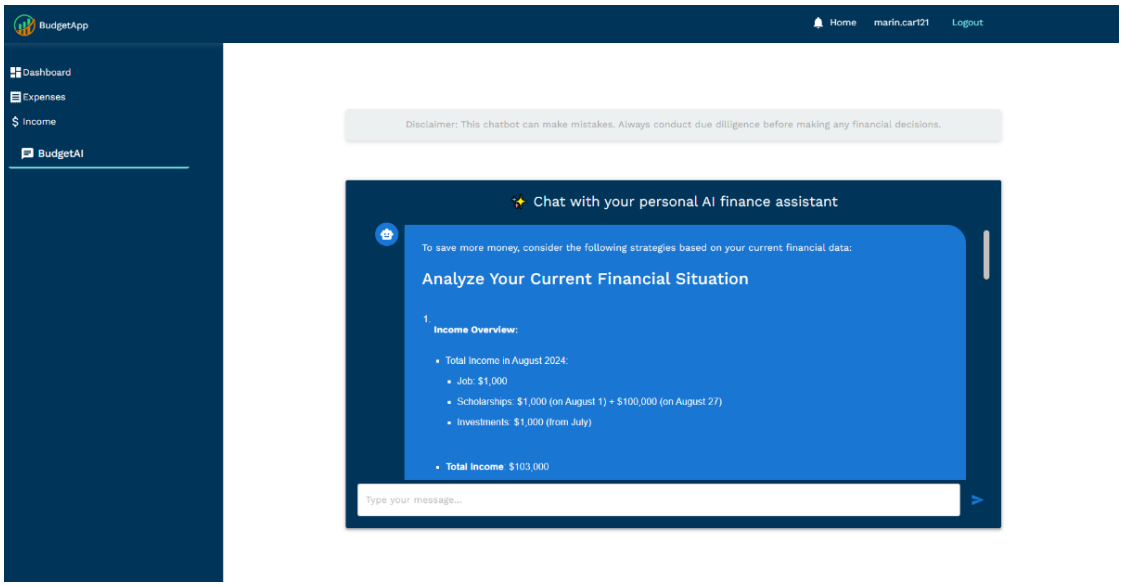
1. Izjava o odricanju odgovornosti (*engl. Disclaimer*) koja upozorava korisnike da *chatbot* može pogriješiti te ih savjetuje da provode vlastitu analizu prije donošenja financijskih odluka. Ova izjava naglašava važnost odgovornog korištenja AI asistenta u kontekstu osobnih financija.

2. Sučelje za chat s AI asistentom za financije. Ovo sučelje sadrži polje za unos teksta gdje korisnici mogu postavljati pitanja ili tražiti savjete.



Slika 3.13 Sučelje BudgetAI

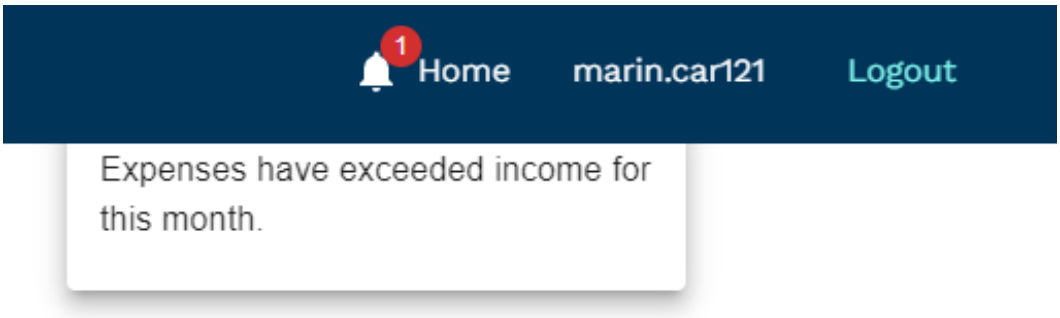
Na slici broj 3.14 prikazan je nastavak interakcije s umjetnom inteligencijom u aplikaciji. U središtu ekrana vidljiv je odgovor AI asistenta na prethodno postavljeno pitanje "Kako mogu uštedjeti više novca?". Ovaj prikaz demonstrira sposobnost AI asistenta da analizira specifične financijske podatke korisnika i pruži personalizirani uvid. Asistent započinje analizu prihodovne strane korisnikovih financija, što je ključni korak u razvoju strategije za uštedu.



Slika 3.14 Primjer odgovora AI asistenta

3.9 Sustav notifikacija za prijelaz budžeta

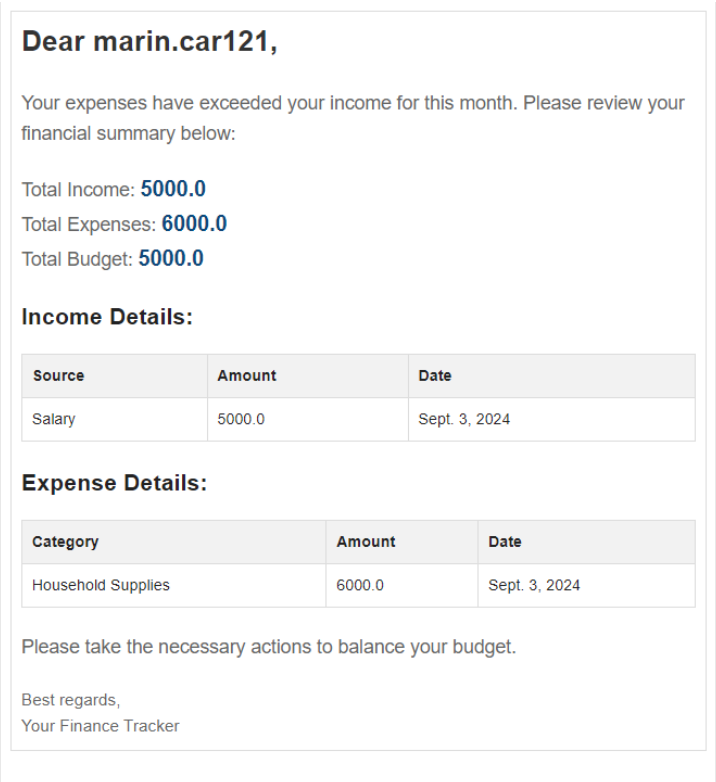
Na slici broj 3.15 prikazan je skočni prozor (*engl. pop-up*) s obavijesti koja se pojavljuje ispod ikone zvona. Ovaj prikaz fokusira se na ključni element obavijesti koji pruža važnu informaciju o korisnikovom financijskom stanju.. Obavijest sadrži sljedeću poruku: *Expenses have exceeded income for this month*. (*hrv.* "Troškovi su premašili prihode za ovaj mjesec.")



Slika 3.15 Primjer notifikacije

Ova obavijest predstavlja ključnu funkcionalnost aplikacije za osobno budžetiranje, upozoravajući korisnika na potencijalnu financijsku neravnotežu. Takva pravovremena upozorenja omogućuju korisniku da brzo reagira i prilagodi svoje financijsko ponašanje kako bi izbjegao daljnje financijske poteškoće.

Na korisnikovu email adresu će također biti poslano upozorenje sa nešto detaljnijim prikazom stanja. Primjer maila je vidljiv na slici 3.16.



Dear marin.car121,

Your expenses have exceeded your income for this month. Please review your financial summary below:

Total Income: **5000.0**
Total Expenses: **6000.0**
Total Budget: **5000.0**

Income Details:

Source	Amount	Date
Salary	5000.0	Sept. 3, 2024

Expense Details:

Category	Amount	Date
Household Supplies	6000.0	Sept. 3, 2024

Please take the necessary actions to balance your budget.

Best regards,
Your Finance Tracker

Slika 3.16 Primjer email notifikacije

Ovakav dizajn obavijesti demonstrira proaktivni pristup aplikacije u pomoći korisnicima da održe zdravo financijsko stanje, kombinirajući praćenje prihoda i rashoda s pravovremenim upozorenjima kada se uoče potencijalni problemi.

4 RAZVOJ I IMPLEMENTACIJA SPECIFIČNIH APLIKACIJSKIH FUNKCIONALNOSTI

U ovom poglavlju detaljno će se razmotriti ključne funkcionalnosti aplikacije kroz analizu relevantnih kodnih isječaka. Cilj je pružiti dubinski uvid u tehničku implementaciju različitih komponenti koje čine osnovu ove aplikacije za osobno budžetiranje. Za svaku od ovih funkcionalnosti, prikazat će se relevantni dijelovi koda, uz detaljna objašnjenja logike, korištenih tehnologija i najboljih praksi u razvoju. Posebna pažnja posvetit će se integraciji *React frontenda* s *Django bekendom*, kao i korištenju *PostgreSQL* baze podataka za učinkovito upravljanje financijskim podacima.

4.1 Datoteke app.js i Config.js

Na slici broj 4.1 prikazan je isječak koda koji definira glavnu *App* funkciju *React* aplikacije koji se nalazi u datoteci *app.js*. Ovaj kod predstavlja strukturu ruta i navigacije unutar aplikacije.

```
22  function App() {
23    return (
24      <SnackbarProvider maxSnack={3}>
25      <div className="App">
26        <BrowserRouter>
27          <Fragment>
28            <NavBars />
29            <Routes>
30              <Route path='/Login' element={<Login />} />
31              <Route path='/Register' element={<Register />} />
32              <Route path='/ForgetPassword' element={<ForgetPassword />} />
33              <Route path='/EnterCode' element={<EnterCode />} />
34              <Route path='/ResetPassword' element={<ResetPassword />} />
35              <Route path='/ResetSuccessfully' element={<ResetSuccessfully />} />
36
37              {/* Private Routes - checks for authentication token */}
38              <Route path="/" element={<PrivateRoute element={DashboardLayout} />}>
39                <Route index element={<PrivateRoute element={DashboardPage} />} />
40                <Route path="expenses" element={<PrivateRoute element={ExpensesPage} />} />
41                <Route path="/profile" element={<PrivateRoute element={ProfilePage} />} />
42                <Route path="income" element={<PrivateRoute element={IncomePage} />} />
43                <Route path="chats" element={<PrivateRoute element={Chatboard} />} />
44              </Route>
45            </Routes>
46          </Fragment>
47        </BrowserRouter>
48      </div>
49      </SnackbarProvider>
50    );
51  }
52
53  export default App;
54
```

Slika 4.1 Datoteka app.js

Ključni elementi ovog koda su:

1. Korištenje *SnackbarProvider* komponente s *maxSnack* postavkom od 3, što omogućuje prikaz do tri obavijesti istovremeno.
2. Implementacija *BrowserRouter* komponente koja omogućuje navigaciju unutar aplikacije.
3. Definicija ruta pomoću *Route* komponenti, koje su podijeljene u dvije glavne kategorije:
    - Javne rute: *Login*, *Register*, *ForgetPassword*, *EnterCode*, *ResetPassword*, i *ResetSuccessfully*.
    - Privatne rute: *Dashboard*, *ExpensesPage*, *ProfilePage*, *IncomePage*, i *Chatboard*.
4. Korištenje *PrivateRoute* komponente za zaštitu ruta koje zahtijevaju autentifikaciju korisnika.
5. Implementacija *Fragment* komponente za grupiranje više elemenata bez dodavanja dodatnog DOM čvora.

Ovakva struktura koda omogućuje učinkovito upravljanje navigacijom unutar aplikacije, jasno odvajajući javno dostupne stranice od onih koje zahtijevaju prijavu korisnika. Korištenje *React Router* biblioteke (*BrowserRouter*, *Route*) osigurava glatko iskustvo jednostranične aplikacije (engl. *Single Page Application*), dok *PrivateRoute* komponenta implementira dodatni sloj sigurnosti.

Na slici 4.2 prikazana je datoteka Settings.js, ključan dio konfiguracije frontend dijela aplikacije za osobno budžetiranje. Ovaj redak koda definira konstantu *API_BASE_URL* koja sadrži osnovni URL adresu API-ja.

```
1   export const  API_BASE_URL = "http://127.0.0.1:8000/api";
```

Slika 4.2 Datoteka Config.js

4.2 Datoteka api.js

Na slici broj 4.3 prikazan je dio sadržaja datoteke api.js, koja implementira ključne funkcionalnosti za komunikaciju s API-jem u aplikaciji za osobno budžetiranje.

Glavne komponente ovog koda su:

1. Uvoz potrebnih modula:
    - *axios* za HTTP zahtjeve
    - *Cookies* za rukovanje kolačićima
    - *API_BASE_URL* iz konfiguracijske datoteke (referenca na sliku 4.2)
2. Kreiranje Axios instance (*API*) s osnovnim URL-om postavljenim na *API_BASE_URL*.
3. Funkcija *setAuthToken*:
    - Dohvaća token iz kolačića
    - Ako token postoji, dodaje ga u zaglavlje *Authorization* svih budućih API zahtjeva
    - Ako token ne postoji, uklanja *Authorization* zaglavlje
4. Funkcija *setupAxiosInterceptors*:
    - Postavlja interceptor za odgovore API-ja
    - Provjerava status kod 401 (neautoriziran)
    - U slučaju neautoriziranog pristupa, uklanja token iz kolačića i preusmjerava korisnika na stranicu za prijavu

```
import axios from 'axios';
import Cookies from 'js-cookie';
import { API_BASE_URL } from '../components/Config';

const API = axios.create({
  baseURL: API_BASE_URL,
});

export const setAuthToken = () => {
  const token = Cookies.get('token');
  if (token) {
    API.defaults.headers.common['Authorization'] = `Bearer ${token}`;
  } else {
    delete API.defaults.headers.common['Authorization'];
  }
};

// interceptor to check for 401 (unauthorized) status code and redirect to login page
export const setupAxiosInterceptors = (navigate) => {
  API.interceptors.response.use(
    response => response,
    error => {
      if (error.response && error.response.status === 401) {
        Cookies.remove('token');
        navigate('/login');
      }
      return Promise.reject(error);
    }
  );
};
```

Slika 4.3 Datoteka API.js

Ova implementacija osigurava konzistentno rukovanje API zahtjevima i autentifikacijom kroz cijelu aplikaciju, što je ključno za sigurnost i pravilno funkcioniranje aplikacije.

Na slici broj 4.4 prikazan je nastavak sadržaja datoteke api.js, koji definira niz funkcija za interakciju s različitim krajnjim točkama API-ja aplikacije za osobno budžetiranje.

```
setAuthToken();
export const getUser = () => API.get('/accounts/user/');
export const fetchExpenses = () => API.get('/expenses/');
export const addExpense = (expense) => API.post('/expenses/', expense);
export const editExpense = (id, expense) => API.put(`/expenses/${id}/`, expense);
export const deleteExpense = (id) => API.delete(`/expenses/${id}/`);
export const searchExpenses = (category) => API.get('/expenses/', { params: { search: category } });
export const fetchIncomes = () => API.get('/income/');
export const addIncome = (income) => API.post('/income/', income);
export const editIncome = (id, income) => API.put(`/income/${id}/`, income);
export const deleteIncome = (id) => API.delete(`/income/${id}/`);
export const searchIncome = (source) => API.get('/income/', { params: { search: source } });
export const exportExpensesPDF = () => '/expenses/pdf/';
export const exportExpensesExcel = () => '/expenses/excel/';
export const exportIncomesPDF = () => '/incomes/pdf/';
export const exportIncomesExcel = () => '/incomes/excel/';
export const fetchNotifications = () => API.get('/check-exceeding-expenses/');
export const postChat = (messages) => API.post('/chat/', { messages });
export const updatePassword = (old_password, password, password1) => API.post('/accounts/user/change_password/',
    old_password, password, password1);
export const fetchDashboardData = async () => {
  try {
    const response = await API.get("/summary/");
    return response.data;
  } catch (error) {
    console.error("Error fetching the dashboard data:", error);
    throw error;
  }
};
```

Slika 4.4 Definiranje krajnjih točaka API-a

Ključne komponente su:

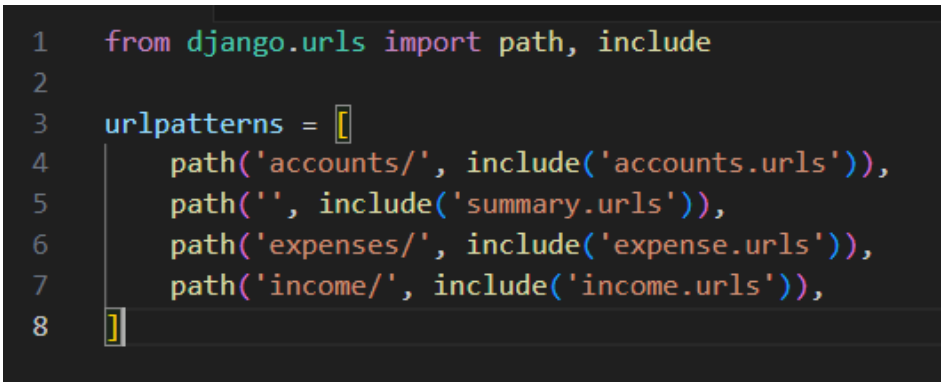
1. Poziv *setAuthToken()* na početku, osiguravajući da je autentifikacijski token postavljen prije bilo kakvih API poziva.
2. Funkcije za upravljanje korisničkim podacima:
    - *getUser*: dohvaća podatke o korisniku
3. Funkcije za upravljanje troškovima:
    - *fetchExpenses*: dohvaća sve troškove
    - *addExpense*: dodaje novi trošak
    - *editExpense*: uređuje postojeći trošak
    - *deleteExpense*: briše trošak

- *searchExpenses*: pretražuje troškove po kategoriji

4. Slične funkcije za upravljanje prihodima (*fetchIncomes*, *addIncome*, itd.)
5. Funkcije za izvoz podataka u PDF i Excel formate
6. *fetchNotifications*: dohvaća obavijesti o prekoračenju troškova
7. *postChat*: šalje poruke chatbotu
8. *updatePassword*: ažurira lozinku korisnika
9. *fetchDashboardData*: asinkrona funkcija koja dohvaća podatke za nadzornu ploču, s rukovanjem pogreškama

4.3 Datoteka urls.py

Na slici broj 4.5 prikazan je isječak koda iz *Django* strane projekta, konkretno iz datoteke urls.py koja se nalazi u glavnom direktoriju projekta. Ovaj kod definira strukturu URL-ova za aplikaciju.



Slika 4.5 Datoteka urls.py

Ova struktura URL-ova predstavlja *backend* dio aplikacije i osigurava da su sve potrebne rute dostupne za API pozive koje smo vidjeli u *frontend* dijelu aplikacije.

- Ruta *accounts/* uključuje URL-ove za prijavu, registraciju i upravljanje korisničkim profilom.

- Ruta za sažetak ('') služe nadzornoj ploči (*Dashboard*)
- Rute *expenses/* i *income/* služe funkcionalnostima za upravljanje troškovima i prihodima koje smo vidjeli u API pozivima.

4.4 Grafički prikaz podataka na *Dashboard*u

Na slici broj 4.6 prikazane su definicije klasa serijalizatora *u Django REST* okviru, koje su ključne za pretvaranje složenih tipova podataka, poput *Django* modela, u formate pogodne za JSON renderiranje. Izabrana je tehnika korištenja serijalizatora za učinkovito rukovanje podacima o troškovima i prihodima u različitim vremenskim okvirima.

```
8   class WeeklyExpenseSerializer(serializers.Serializer):
9       date = serializers.DateField()
10      total_amount = serializers.FloatField()
11
12  class YearlyExpenseSerializer(serializers.Serializer):
13      month = serializers.IntegerField()
14      total_amount = serializers.FloatField()
15
16  class WeeklyIncomeSerializer(serializers.Serializer):
17      date = serializers.DateField()
18      total_amount = serializers.FloatField()
19
20  class YearlyIncomeSerializer(serializers.Serializer):
21      month = serializers.IntegerField()
22      total_amount = serializers.FloatField()
23
```

Slika 4.6 Profile sučelje

Definirane su četiri klase serijalizatora: *WeeklyExpenseSerializer*, *YearlyExpenseSerializer*, *WeeklyIncomeSerializer* i *YearlyIncomeSerializer*. Svaka od ovih klasa nasljeđuje od *serializers.Serializer*, što je osnovna klasa za serijalizaciju u *Django REST* okviru. Klase za tjedne podatke (*WeeklyExpenseSerializer* i *WeeklyIncomeSerializer*) koriste *DateField()* za polje datuma, dok klase za godišnje podatke (*YearlyExpenseSerializer* i *YearlyIncomeSerializer*) koriste *IntegerField()* za

polje mjeseca. Sve klase uključuju *FloatField()* za *total_amount*, što omogućuje precizno bilježenje financijskih iznosa.

Na slici 4.7 prikazane su dvije ključne funkcije za izračun korisničkih troškova i prihoda u aplikaciji za osobno upravljanje proračunom. Implementirane su funkcije calculate_user_expenses i *calculate_user_incomes* koje koriste *Django ORM* (Object-Relational Mapping) za dohvaćanje i agregaciju financijskih podataka.

```
30  def calculate_user_expenses(user):
31      today = timezone.now().date()
32      start_of_week = today - timezone.timedelta(days=today.weekday())
33      start_of_month = today.replace(day=1)
34      start_of_year = today.replace(month=1, day=1)
35
36      total_today = Expense.objects.filter(owner=user, date=today).aggregate(total_amount=Sum('amount'), count=Count('id'))
37      total_this_week = Expense.objects.filter(owner=user, date__gte=start_of_week).aggregate(total_amount=Sum('amount'), count=Count('id'))
38      total_this_month = Expense.objects.filter(owner=user, date__gte=start_of_month).aggregate(total_amount=Sum('amount'), count=Count('id'))
39      total_this_year = Expense.objects.filter(owner=user, date__gte=start_of_year).aggregate(total_amount=Sum('amount'), count=Count('id'))
40
41      return {
42          'total_today': total_today['total_amount'] or 0,
43          'count_today': total_today['count'] or 0,
44          'total_this_week': total_this_week['total_amount'] or 0,
45          'count_this_week': total_this_week['count'] or 0,
46          'total_this_month': total_this_month['total_amount'] or 0,
47          'count_this_month': total_this_month['count'] or 0,
48          'total_this_year': total_this_year['total_amount'] or 0,
49          'count_this_year': total_this_year['count'] or 0,
50      }
51
52
53  def calculate_user_incomes(user): ...
74
```

Slika 4.7 Funkcije za kalkulaciju prihoda i rashoda

Obje funkcije slijede sličnu strukturu. Prvo se određuju vremenski okviri za danas, početak tjedna, mjeseca i godine koristeći Python *datetime* modul i Django *timezone*. Zatim se vrši filtriranje i agregacija podataka iz modela *Expense* i *UserIncome* za različite vremenske periode (danas, ovaj tjednik, ovaj mjesec, ova godina). Koristi se metoda *filter()* za odabir relevantnih zapisa, a *aggregate()* za izračun ukupnih iznosa i broja stavki. Rezultati se vraćaju u obliku rječnika koji sadrži ukupne iznose i broj stavki za svaki vremenski period. Implementacija osigurava da se vraća 0 ako nema podataka, izbjegavajući tako potencijalne pogreške pri obradi praznih rezultata.

Na slici 4.8 prikazane su dvije funkcije koje su ključne za pripremu i formatiranje podataka o troškovima ili prihodima u određenom vremenskom razdoblju. Implementirane su funkcije *get_period_data* i *format_data* koje zajedno omogućuju

fleksibilno prikupljanje i strukturiranje financijskih podataka za tjedne ili godišnje preglede.

```python
def get_period_data(model, user, period='week'):
    today = timezone.now().date()
    if period == 'week':
        start_date = today - timedelta(days=6)
        date_range = [start_date + timedelta(days=i) for i in range(7)]
    else:  # year
        start_date = today.replace(month=1, day=1)
        date_range = range(1, 13)  # months

    group_by = 'date' if period == 'week' else 'date__month'
    end_date = today

    aggregated_data = model.objects.filter(
        owner=user,
        date__range=[start_date, end_date]
    ).values(group_by).annotate(total_amount=Sum('amount'))

    return format_data(aggregated_data, date_range, period)


def format_data(aggregated_data, date_range, period='week'):
    formatted_data = []
    date_key = 'date' if period == 'week' else 'date__month'
    output_key = 'date' if period == 'week' else 'month'

    for date in date_range:
        matching_item = None
        for item in aggregated_data:
            if item[date_key] == date:
                matching_item = item
                break

        total_amount = matching_item['total_amount'] if matching_item else 0

        formatted_data.append({
            output_key: date,
            'total_amount': total_amount
        })

    return formatted_data

```

Slika 4.8 Funkcije get_period_data i format_data

Funkcija *get_period_data* prima model (*Expense* ili *UserIncome*), korisnika i period ('week' ili 'year') kao parametre. Ovisno o odabranom periodu, funkcija postavlja odgovarajući raspon datuma i grupiranje podataka. Za tjedni pregled, generira se lista datuma za proteklih 7 dana, dok se za godišnji pregled koristi raspon od 12 mjeseci. Zatim se koristi *Django ORM* za filtriranje i agregaciju podataka prema zadanim kriterijima. Rezultat se prosljeđuje funkciji *format_data* za daljnju obradu.

Funkcija *format_data* preuzima agregirane podatke, raspon datuma i period, te ih formatira u konzistentan izlazni format. Ova funkcija osigurava da su svi datumi ili mjeseci u zadanom rasponu uključeni u izlaz, čak i ako za njih nema podataka, postavljajući iznos na 0 za te periode. Rezultat je lista rječnika, gdje svaki rječnik sadrži datum (ili mjesec) i odgovarajući ukupni iznos.

Na slici 4.9 prikazana je implementacija klase *DashboardSummaryAPIView* koja nasljeđuje *Django REST* okvir *APIView*. Ova klasa služi za generiranje sveobuhvatnog pregleda financijskih podataka korisnika za nadzornu ploču aplikacije.

```python
class DashboardSummaryAPIView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request, *args, **kwargs):
        todays_date = datetime.today().date()
        six_months_ago = todays_date - timedelta(days=30*6)
        expenses_by_category = Expense.objects.filter(
            owner=request.user,
            date__gte=six_months_ago,
            date__lte=todays_date
        ).values('category').annotate(total_amount=Sum('amount'))

        incomes_by_source = UserIncome.objects.filter(
            owner=request.user,
            date__gte=six_months_ago,
            date__lte=todays_date
        ).values('source').annotate(total_amount=Sum('amount'))

        finalrep_exp = {expense['category']: expense['total_amount'] for expense in expenses_by_category}
        finalrep_inc = {income['source']: income['total_amount'] for income in incomes_by_source}
        user_expenses = calculate_user_expenses(request.user)
        user_incomes = calculate_user_incomes(request.user)

        weekly_expenses = get_period_data(Expense, request.user, 'week')
        weekly_expenses_serializer = WeeklyExpenseSerializer(weekly_expenses, many=True)
        yearly_expenses = get_period_data(Expense, request.user, 'year')
        yearly_expense_serializer = YearlyExpenseSerializer(yearly_expenses, many=True)
        weekly_incomes = get_period_data(UserIncome, request.user, 'week')
        weekly_income_serializer = WeeklyIncomeSerializer(weekly_incomes, many=True)
        yearly_incomes = get_period_data(UserIncome, request.user, 'year')
        yearly_income_serializer = YearlyIncomeSerializer(yearly_incomes, many=True)

        return Response({'expense_category_data': finalrep_exp,
                         'income_source_data': finalrep_inc,
                         'user_expenses': user_expenses, 'user_incomes': user_incomes,
                         'weekly_expenses': weekly_expenses_serializer.data,
                         'yearly_expenses': yearly_expense_serializer.data,
                         'weekly_incomes': weekly_income_serializer.data,
                         'yearly_incomes': yearly_income_serializer.data})
```

Slika 4.9 DashboardSummaryAPIView

Metoda *get* ove klase odgovorna je za prikupljanje i strukturiranje različitih financijskih podataka. Implementacija obuhvaća sljedeće ključne korake:

- Izračunavanje troškova po kategorijama i prihoda po izvorima za posljednjih šest mjeseci koristeći *Django ORM*.

- Generiranje sažetaka troškova i prihoda korisnika pozivanjem funkcija *calculate_user_expenses* i *calculate_user_incomes*.
- Dohvaćanje tjednih i godišnjih podataka o troškovima i prihodima pomoću funkcije *get_period_data*.
- Korištenje odgovarajućih serijalizatora za formatiranje tjednih i godišnjih podataka.

Završni korak metode je vraćanje strukturiranog *Response* objekta koji sadrži sve prikupljene i formatirane podatke. Ovaj objekt uključuje podatke o troškovima po kategorijama, prihodima po izvorima, korisničkim troškovima i prihodima, te tjedne i godišnje preglede troškova i prihoda.

Ovakva implementacija omogućuje klijentu aplikacije da jednim API pozivom dobije sve relevantne financijske podatke potrebne za prikaz na nadzornoj ploči. Time se osigurava učinkovitost i konzistentnost podataka prikazanih korisniku, pružajući sveobuhvatan pregled osobnih financija kroz različite vremenske okvire i kategorije.

Na slici 4.10 prikazana je implementacija *React* komponente *DashboardPage* koja služi kao glavna komponenta za prikaz nadzorne ploče u aplikaciji.

```
const DashboardPage = () => {
    const [expenseData, setExpenseData] = useState({});
    const [expenseDataTime, setExpenseDataTime] = useState({});
    const [expenseDataWeeklyChart, setexpenseDataWeeklyChart] = useState([]);
    const [expenseDataYearlyChart, setexpenseDataYearlyChart] = useState([]);
    const [incomeData, setIncomeData] = useState({});
    const [incomeDataTime, setIncomeDataTime] = useState({});
    const [incomeDataWeeklyChart, setIncomeDataWeeklyChart] = useState([]);
    const [incomeDataYearlyChart, setIncomeDataYearlyChart] = useState([]);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        const fetchData = async () => {
            try {
                const data = await fetchDashboardData();

                setExpenseData(data.expense_category_data);
                setIncomeData(data.income_source_data);
                setExpenseDataTime(data.user_expenses);
                setIncomeDataTime(data.user_incomes);
                setexpenseDataWeeklyChart(data.weekly_expenses);
                setexpenseDataYearlyChart(data.yearly_expenses);
                setIncomeDataWeeklyChart(data.weekly_incomes);
                setIncomeDataYearlyChart(data.yearly_incomes);
                setLoading(false);
            } catch (error) {
                setLoading(false);
                toast.error("Failed to load dashboard data.");
            }
        };

        fetchData();
    }, []);

```

Slika 4.10 React komponenta *DashboardPage.js*

Komponenta koristi *React Hooks* za upravljanje stanjem i efektima. Konkretno, koristi se *useState hook* za definiranje različitih stanja komponente, uključujući podatke o troškovima i prihodima za različite vremenske okvire i grafičke prikaze. Ovaj pristup omogućuje jednostavno upravljanje dinamičkim podacima i prilagođavanje korisničkog sučelja ovisno o stanju aplikacije.

Ključni dijelovi implementacije uključuju definiranje višestrukih stanja pomoću *useState hooka* za različite aspekte podataka nadzorne ploče, te korištenje *useEffect hooka* za dohvaćanje podataka prilikom inicijalizacije komponente. Asinkrona funkcija *fetchData* poziva API *endpoint fetchDashboardData* kako bi dohvatila sve potrebne podatke. Nakon što se podaci uspješno dohvaćaju, različita stanja komponente ažuriraju se pomoću odgovarajućih *setter* funkcija. Također, implementirana je obrada pogrešaka, pri čemu se stanje učitavanja postavlja na *false*, a korisniku se prikazuje obavijest o pogrešci putem funkcije *toast.error*.

Komponenta je dizajnirana da prikaže različite aspekte korisničkih financija, uključujući troškove po kategorijama, prihode po izvorima, te tjedne i godišnje preglede troškova i prihoda. Korištenje *React Hooks*a osigurava optimizirano upravljanje stanjem i životnim ciklusom komponente, što doprinosi boljim performansama i lakšem održavanju koda.

Na slici 4.11 prikazane su dvije funkcije koje služe za formatiranje tjednih podataka o troškovima i prihodima u *React* komponenti nadzorne ploče aplikacije za osobno upravljanje proračunom.

```
119     const formattedWeeklyExpenses = expenseDataWeeklyChart.map((expense) => {
120         const dateObj = new Date(expense.date);
121         const formattedDate = dateObj.toLocaleDateString("en-US", {
122             month: "long",
123             day: "numeric",
124         });
125         return {
126             ...expense,
127             formattedDate,
128             total_amount: parseFloat(expense.total_amount),
129         };
130     });
131
132     const formattedWeeklyIncomes = incomeDataWeeklyChart.map((income) => {
133         const dateObj = new Date(income.date);
134         const formattedDate = dateObj.toLocaleDateString("en-US", {
135             month: "long",
136             day: "numeric",
137         });
138         return {
139             ...income,
140             formattedDate,
141             total_amount: parseFloat(income.total_amount),
142         };
143     });
144
```

Slika 4.11 Formatiranje tjednih podataka o troškovima i prihodima

Funkcija *formattedWeeklyExpenses* obrađuje podatke o tjednim troškovima koristeći nekoliko ključnih koraka. Prvo se koristi metoda *map* na nizu *expenseDataWeeklyChart* kako bi se iteriralo kroz svaki unos troška. Za svaki od tih unosa kreira se novi *Date* objekt temeljen na vrijednosti *expense.date*. Nakon toga, datum se formatira u lokalizirani *string* koristeći metodu *toLocaleDateString*, pri čemu se opcije postavljaju tako da prikazuju puni naziv mjeseca ("*long*") i numerički dan. Na kraju, funkcija vraća novi objekt koji uključuje originalne podatke o trošku, formatirani datum i ukupni iznos koji se pretvara u broj koristeći *parseFloat*.

Funkcija *formattedWeeklyIncomes* slijedi identičnu logiku, ali obrađuje podatke o tjednim prihodima. Također iterira kroz niz *incomeDataWeeklyChart*, kreira *Date* objekt iz *income.date* i formatira datum na isti način kao i za troškove. Rezultat je objekt koji sadrži originalne podatke o prihodu, formatirani datum i ukupni iznos pretvoren u broj.

Na slici 4.12 prikazan je dio *React* komponente koja implementira grafički prikaz tjednih troškova na nadzornoj ploči aplikacije za osobno upravljanje proračunom.

```
<div className="col-md-6">
    <div className="card h-100">
        <h3 className="py-3 px-4 fs-5">
            Last 7 days
        </h3>
        <div className="card-body">
            <ResponsiveContainer
                width="100%"
                height={300}
            >
                <LineChart
                    data={formattedWeeklyExpenses}
                >
                    <CartesianGrid strokeDasharray="4 4" />
                    <XAxis dataKey="formattedDate" />
                    <YAxis />
                    <Tooltip />
                    <Legend />
                    <Line
                        type="monotone"
                        dataKey="total_amount"
                        stroke="#8884d8"
                    />
                </LineChart>
            </ResponsiveContainer>
        </div>
    </div>
```

Slika 4.12 grafički prikaz tjednih troškova

Glavno tijelo kartice (*className="card-body"*) sadrži *ResponsiveContainer* komponentu, iz biblioteke *Recharts*, koja osigurava responzivnost grafa.

Unutar *ResponsiveContainer* nalazi se *LineChart* komponenta koja prikazuje linijski graf:

- Podaci za graf dolaze iz *formattedWeeklyExpenses* varijable.

- *CartesianGrid* komponenta dodaje mrežu grafu s isprekidanim linijama.
- *XAxis* koristi *formattedDate* kao ključ za x-os.
- *YAxis*, *Tooltip* i *Legend* komponente su uključene za dodatne informacije i interaktivnost.
- *Line* komponenta definira samu liniju grafa, koristeći "*monotone*" tip za glatku krivulju, *total_amount* kao ključ podataka i plavu boju ("#8884d8") za liniju.

Ovim načinom implementacije omogućuje se vizualno atraktivan i interaktivan prikaz tjednih troškova korisnika, s responzivnim dizajnom koji se prilagođava različitim veličinama zaslona. Korištenje *Recharts* biblioteke osigurava bogatu funkcionalnost grafa uz relativno jednostavnu implementaciju u *React* okruženju.

4.5 Implementacija umjetne inteligencije

Na slici 4.13 prikazana je implementacija funkcije *chat_with_gpt* koja služi kao API *endpoint* za interakciju s *OpenAI* GPT modelom u kontekstu aplikacije za osobno upravljanje proračunom. Ova funkcija integrira umjetnu inteligenciju za pružanje personaliziranih financijskih analiza korisnicima.

```python
client = OpenAI(api_key=settings.OPENAI_API_KEY)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def chat_with_gpt(request):
    if request.method == "POST":
        data = json.loads(request.body)
        messages = data.get("messages", [])

        if not messages:
            return JsonResponse({"error": "No messages provided"}, status=400)

        try:
            current_date = datetime.now().strftime("%B %d, %Y")

            SYSTEM_PROMPT = f"\
You will provide helpful analyses of the user's financial data. \
Your outputs will be nicely formatted using markdown. \
**IMPORTANT**: DO NOT output tables or LaTeX. \
Current date: {current_date}"

            system_messages = []

            if len(messages) == 1 and messages[0]['role'] == 'user':
                user_financial_data = generate_combined_text(
                    Expense.objects.filter(owner=request.user),
                    UserIncome.objects.filter(owner=request.user)
                )
                system_messages = [
                    {"role": "system", "content": SYSTEM_PROMPT},
                    {"role": "system", "content": user_financial_data},
                ]
                messages = system_messages + messages

            all_messages = messages

            response = client.chat.completions.create(
                model="gpt-4o-mini",
                messages=all_messages,
                max_tokens=1024,
                temperature=0.7,
            )

            assistant_message = response.choices[0].message.content
            # Add the assistant's response to the message history
            messages.append({"role": "assistant", "content": assistant_message})

            return JsonResponse({"messages": messages})

        except Exception as e:
            return JsonResponse({"error": str(e)}, status=500)

    return JsonResponse({"error": "Invalid request method"}, status=405)

```

Slika 4.13 Funkcija chat_with_gpt

Ključni elementi implementacije su:

- Korištenje OpenAI API-ja za komunikaciju s GPT modelom.
- Autentifikacija korisnika pomoću *@permission_classes([IsAuthenticated])* dekoratora.
- Obrada POST zahtjeva s porukama korisnika.

- Generiranje kontekstualnih informacija o financijskim podacima korisnika.
- Formiranje sustava poruka koji uključuje upute za GPT model i financijske podatke korisnika.
- Pozivanje GPT modela s pripremljenim porukama i parametrima.
- Obrada odgovora modela i vraćanje rezultata korisniku.

Funkcija prvo provjerava valjanost ulaznih podataka, zatim generira trenutni datum i postavlja sistemsku poruku koja definira ulogu i ograničenja GPT modela. Ako postoji korisnička poruka, funkcija dohvaća financijske podatke korisnika i dodaje ih u kontekst razgovora.

GPT model se poziva s parametrima koji uključuju model "gpt-4o-mini", maksimalni broj tokena i temperaturu koja utječe na kreativnost odgovora. Odgovor modela se zatim dodaje u povijest poruka i vraća korisniku.

Na slici 4.14 prikazana je implementacija funkcije *generate_combined_text* koja služi za generiranje strukturiranog tekstualnog prikaza financijskih podataka korisnika, uključujući i troškove i prihode.

```
70  def generate_combined_text(expenses, incomes):
71      output = "Expenses:\n"
72      output += "Category, Amount, Date, Description\n"
73      for expense in expenses:
74          output += f"{expense.category}, {expense.amount}, {expense.date}, {expense.description}\n"
75
76      output += "\nIncomes:\n"
77      output += "Source, Amount, Date, Description\n"
78      for income in incomes:
79          output += f"{income.source}, {income.amount}, {income.date}, {income.description}\n"
80
81      return output
82
```

Slika 4.14 Funkcija generate_combined_text

Funkcija radi sljedeće:

- Inicijalizira varijablu *output* s naslovom "*Expenses*:".
- Dodaje zaglavlje stupaca za troškove: *"Category, Amount, Date, Description"*.

- Iterira kroz listu troškova (*expenses*), dodajući svaki trošak u formatirani string s njegovom kategorijom, iznosom, datumom i opisom.
- Dodaje prazan redak i naslov "*Incomes*:".
- Dodaje zaglavlje stupaca za prihode: *"Source, Amount, Date, Description"*.
- Iterira kroz listu prihoda (*incomes*), dodajući svaki prihod u formatirani string s njegovim izvorom, iznosom, datumom i opisom.
- Vraća kompletni generirani tekst.

Strukturirani prikaz olakšava GPT modelu razumijevanje i analizu financijskih informacija, što rezultira preciznijim i relevantnijim odgovorima na upite korisnika vezane uz njihove osobne financije. Ova funkcija igra ključnu ulogu u pripremi podataka za AI-podržanu analizu i savjetovanje u aplikaciji

Na slici 4.15 prikazana je implementacija *React* komponente *Chatboard* koja služi kao okvir za prikaz *chatbot* sučelja u aplikaciji.

```
4   const Chatboard = () => {
5     return (
6       <div>
7         <p
8           style={{
9             fontFamily: "Roboto, sans-serif",
10            color: "darkgray",
11            backgroundColor: "#ecf0f1",
12            padding: "15px 20px",
13            borderRadius: "8px",
14            marginBottom: "6%",
15            textAlign: "center",
16            boxShadow: "0px 4px 6px rgba(0, 0, 0, 0.1)",
17          }}
18        >
19          Disclaimer: This chatbot can make mistakes. Always conduct due dilligence before making any financial decisions.
20        </p>
21        <ChatWindow />
22      </div>
23    );
24  };
25
26  export default Chatboard;
```

Slika 4.15 React komponenta Chatboard .js

Komponenta je definirana kao funkcijska komponenta bez *propsa*. Vraća *div* element koji sadrži dva glavna dijela: Paragraf *(<p>)* s upozorenjem o mogućim

pogreškama *chatbota* I *ChatWindow* komponentu koja sadrži glavno sučelje za interakciju s *chatbotom*.

Na slici 4.16 prikazana je implementacija dijela *React* komponente *ChatWindow* koja upravlja prikazom i interakcijom s chat sučeljem u aplikaciji.

```
84  const ChatWindow = () => {
85    const [messages, setMessages] = useState([]);
86    const [inputMessage, setInputMessage] = useState("");
87    const [isWaiting, setIsWaiting] = useState(false);
88    const listRef = useRef(null);
89    const lastMessageRef = useRef(null);
90
91    const scrollToLastMessage = () => {
92      if (lastMessageRef.current && listRef.current) {
93        const padding = 20;
94        const scrollPosition = lastMessageRef.current.offsetTop - padding;
95
96        listRef.current.scrollTo({
97          top: scrollPosition,
98          behavior: "smooth"
99        });
100
101       // also scroll to the top of the list with some padding
102       const listRefScrollTopPadding = 75;
103       window.scrollTo({
104         top: listRef.current.offsetTop - listRefScrollTopPadding,
105         behavior: "smooth"
106       });
107     }
108   };
109
```

Slika 4.16 React komponenta *ChatWindow.js*

Ključni elementi implementacije su:

1. Korištenje *React Hooks* za upravljanje stanjem:
    - *useState* za poruke, unos poruke i stanje čekanja
    - *useRef* za reference na DOM elemente (posljednja poruka i lista poruka)
2. Funkcija *scrollToLastMessage* koja omogućuje automatsko pomicanje prikaza na posljednju poruku:
    - Provjerava postojanje referenci na posljednju poruku i listu
    - Izračunava poziciju za scroll uzimajući u obzir padding
    - Koristi *scrollTo* metodu za glatko pomicanje do izračunate pozicije
    - Dodatno pomiče prozor preglednika do vrha liste poruka s određenim *paddingom*

Ova implementacija osigurava dobro korisničko iskustvo kroz nekoliko ključnih funkcionalnosti. Održava stanje razgovora (engl. *messages*), omogućuje unos novih poruka (engl. *inputMessage*) i prikazuje indikator čekanja dok se čeka odgovor (engl. *isWaiting*). Dodatno, automatski pomiče prikaz na najnoviju poruku, čime se osigurava da korisnik uvijek vidi najrelevantniji sadržaj, što rezultira fluidnim i intuitivnim chat sučeljem koje poboljšava interakciju korisnika s AI asistentom za financijske savjete.

Funkcija *scrollToLastMessage* je posebno važna jer osigurava da korisnik ne mora ručno pomicati prikaz kako bi vidio nove poruke, što je ključno za fluidnu interakciju u chat sučelju. Na slici broj 4.17 prikazana je implementacija ključnih funkcionalnosti chat sučelja u *React* komponenti *ChatWindow*:

```
110  useEffect(() => {
111    scrollToLastMessage();
112  }, [messages]);
113
114  const sendMessage = useCallback(async () => {
115    if (inputMessage.trim() === "") return;
116
117    const newUserMessage = { role: "user", content: inputMessage };
118    const updatedMessages = [...messages, newUserMessage];
119
120    setMessages(updatedMessages);
121    setInputMessage("");
122    setIsWaiting(true);
123
124    try {
125      const response = await postChat(updatedMessages);
126      const newMessages = response.data.messages;
127
128      setMessages(newMessages);
129    } catch (error) {
130      console.error("Error communicating with OpenAI API:", error);
131    } finally {
132      setIsWaiting(false);
133    }
134  }, [inputMessage, messages]);
135
136  const handleKeyPress = (event) => {
137    if (event.key === "Enter") {
138      sendMessage();
139    }
140  };
141
142  const displayMessages = messages.filter(message => message.role !== "system");
143
```

Slika 4.17 Funkcije unutar komponente ChatWindow.js

Funkcija *useEffect* osigurava automatsko pomicanje na posljednju poruku nakon svake promjene u listi poruka. Funkcija *sendMessage*, implementirana pomoću

useCallback hooka, upravlja slanjem korisničkih poruka i primanjem odgovora od AI asistenta. Ona provjerava valjanost unosa, ažurira stanje poruka, postavlja indikator čekanja i komunicira s backend API-jem putem *postChat* funkcije. Implementirana je i obrada pogrešaka te završno resetiranje indikatora čekanja. Funkcija *handleKeyPress* omogućuje slanje poruke pritiskom na tipku Enter, dok *displayMessages* filtrira sistemske poruke iz prikaza.

Na slici broj 4.18 prikazana je implementacija korisničkog sučelja *React* komponente *ChatWindow*

```
144   return (
145     <Box sx={boxStyle}>
146       <Typography variant="h5" gutterBottom sx={{ textAlign: "center", color:"#fff" }}>
147         ✨ Chat with your personal AI finance assistant
148       </Typography>
149       <List sx={listStyle} ref={listRef}>
150         {displayMessages.map((message, index) => (
151           <ListItem
152             key={index}
153             sx={listItemStyle}
154             ref={index === displayMessages.length - 1 ? lastMessageRef : null}
155           >
156             <Avatar sx={avatarStyle(message.role)}>
157               {message.role === "user" ? <PersonIcon /> : <SmartToyIcon />}
158             </Avatar>
159             <ListItemText sx={listItemTextStyle(message.role)}
160               primary={
161                 <Typography
162                   variant="body1"
163                   sx={messageTextStyle}
164                   component="div"
165                 >
166                   <ReactMarkdown>
167                     {message.content}
168                   </ReactMarkdown>
169                 </Typography>
170               }
171             />
172           </ListItem>
173         ))}
174       </List>
175       <Box sx={inputBoxStyle}>
176         <TextField
177           fullWidth
178           variant="outlined"
179           value={isWaiting ? "Waiting for response..." : inputMessage}
180           onChange={(e) => setInputMessage(e.target.value)}
181           onKeyDown={handleKeyPress}
182           placeholder="Type your message..."
183           sx={textFieldStyle}
184           disabled={isWaiting}
185         />
186         <IconButton
187           color="primary"
188           onClick={sendMessage}
189           sx={iconButtonStyle}
190           disabled={isWaiting}
191         >
192           <SendIcon />
193         </IconButton>
194       </Box>
195     </Box>
196   );
197 };
198
199 export default ChatWindow;
```

Slika 4.18 Korisničko sučelje Chatwindow.js

Komponenta je strukturirana kao interaktivni chat prozor s prikazom poruka i sučeljem za unos novih poruka. Glavni elementi uključuju naslov *Chat with your personal AI finance assistant*, listu poruka (engl. *List*) koja prikazuje razgovor, te polje za unos (engl. *TextField*) s pripadajućim gumbom za slanje (engl. *IconButton*).

Svaka poruka u listi prikazana je kao *ListItem* s avatarom koji razlikuje korisničke poruke (engl. *PersonIcon*) od odgovora AI asistenta (engl. *SmartToyIcon*). Sadržaj poruke renderira se pomoću *ReactMarkdown* komponente, što omogućuje formatiranje teksta. Implementirano je i uvjetno renderiranje reference na posljednju poruku za potrebe automatskog pomicanja prikaza.

Polje za unos teksta dinamički mijenja svoj prikaz ovisno o stanju *isWaiting*, prikazujući *Waiting for response...* kada je AI asistent aktivan. Gumb za slanje poruke također se onemogućuje tijekom čekanja odgovora. Komponenta koristi različite stilske objekte (*boxStyle*, *listStyle*, *avatarStyle*, itd.) za konzistentan i atraktivan izgled sučelja.

Ovakva implementacija pruža intuitivno i vizualno privlačno sučelje za interakciju s AI asistentom za financije, olakšavajući korisnicima dobivanje personaliziranih financijskih savjeta i analiza unutar aplikacije za upravljanje osobnim financijama.

5 ZAKLJUČAK

U ovom radu predstavljena je inovativna aplikaciju za budžetiranje koja je dizajnirana s ciljem pružanja sveobuhvatnog rješenja za praćenje i optimizaciju osobnih financija korisnika. Tehnologije korištene u ovoj aplikacij, kao što su *React*, *Django REST* i *PostgreSQL*, pokazale su se izuzetno korisnima u pružanju skalabilnog, sigurnog i interaktivnog rješenja za upravljanje osobnim financijama. *React* omogućuje brzo i responzivno korisničko sučelje, dok *Django REST* pruža robustan okvir za razvoj sigurnog API-ja, a *PostgreSQL* osigurava pouzdanu pohranu podataka.

Za budući razvoj aplikacije, moglo bi se uvesti nekoliko poboljšanja i novih funkcionalnosti. Na primjer, dodavanje integracije s vanjskim financijskim institucijama omogućilo bi automatsko praćenje bankovnih transakcija. Također, implementacija naprednijih analitičkih alata i prediktivnih modela, temeljena na umjetnoj inteligenciji, mogla bi korisnicima pružiti dublje uvide u financijske trendove i prilike za optimizaciju troškova. Na taj način, aplikacija bi se mogla značajno obogatiti kroz razne inovacije, poboljšavajući korisničko iskustvo i pružajući dodatnu vrijednost u svakodnevnom upravljanju financijama.

Literatura

[1] HTML, s interneta, https://en.wikipedia.org/wiki/HTML. , kolovoz 2024.

[2] CSS, s interneta, https://en.wikipedia.org/wiki/CSS. , kolovoz 2024.

[3] JSX, s interneta, https://en.wikipedia.org/wiki/JSX_(JavaScript), kolovoz 2024.

[4] NPM, s interneta, https://www.npmjs.com/ , kolovoz 2024.

[5] React, s interneta, https://react.dev/ , kolovoz 2024.

[6] Axios, s interneta, https://axios-http.com/docs/intro , kolovoz 2024.

[7] M. UI, s interneta, https://mui.com/ , kolovoz 2024.

[8] ReCharts, s interneta, https://recharts.org/en-US/, kolovoz 2024.

[9] Python, s interneta, https://www.python.org/ , kolovoz 2024.

[10] Django, s interneta, https://www.djangoproject.com/ , kolovoz 2024.

[11] Django rest framework, s interneta, https://www.django-rest-framework.org/ , kolovoz 2024.

[12] OpenAI Python, s interneta, https://platform.openai.com/docs/libraries, kolovoz 2024.

[13] Postgresql, s interneta, https://www.postgresql.org/ , kolovoz 2024.

[14] Pgadmin, s interneta, https://www.pgadmin.org/ , kolovoz 2024.

[15] V. s. code, s interneta, https://code.visualstudio.com/ , kolovoz 2024.

[16] Git, s interneta, https://git-scm.com/ , kolovoz 2024.

POPIS SLIKA

SAŽETAK

U ovom radu dizajnirana je i implementirana web aplikacija za praćenje osobnih financija. Aplikacija omogućuje funkcionalnosti dodavanja, brisanja i mijenjanja troškova i prihoda prema svojim potrebama. Dodatno, aplikacija omogućuje grafički prikaz troškova, pružajući korisnicima vizualni uvid u njihove financijske obrasce kao i organizaciju troškova i prihoda po kategorijama. Dodatno, aplikacija omogućuje komunikaciju korisnika s ChatGPT-om radi pružanja financijskih savjeta. Također, implementirana je cjelovita autentifikacija unutar spomenute web aplikacije. Za razvoj klijentskog dijela aplikacije korišten je React javascript radni okvir. Za razvoj poslužiteljskog dijela web aplikacije korišten je Django uz PostgreSQL sustav za upravljanje bazom podataka.

Ključne riječi: praćenje troškova, web sustav, Django REST Framework, React, autentikacija

ABSTRACT

In this final paper, a web application for personal finance tracking was designed and implemented. The application provides functionalities for adding, deleting, and modifying expenses and income according to user needs. Additionally, the application offers graphical representations of expenses, giving users a visual insight into their financial patterns as well as organizing expenses and income by categories. Furthermore, the application allows user interaction with ChatGPT to provide financial advice. A comprehensive authentication system has also been implemented within the application. The client-side of the application was developed using the React JavaScript framework, while the server-side was developed with Django and utilizes the PostgreSQL database management system.

Keywords: budget management, web system, Django REST Framework, React, authentication