

Izrada web aplikacije za stvaranje liste želja

Janeš, David

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:195:474725>

Rights / Prava: [In copyright](#)/[Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-09-13**



Repository / Repozitorij:

[Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository](#)



Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Sveučilišni diplomski studij Informatika – modul Poslovna informatika

David Janeš

Izrada web aplikacije za stvaranje liste želja

Diplomski rad

Mentor: Doc. dr. sc. Vanja Slavuj

Rijeka, prosinac 2022.

Rijeka, 5.5.2022.

Zadatak za diplomski rad

Pristupnik: David Janeš

Naziv diplomskog rada: Izrada web aplikacije za stvaranje liste želja

Naziv diplomskog rada na eng. jeziku: Wishlist web application development

Sadržaj zadatka: U diplomskom radu cilj je prikazati cjelokupan postupak izrade web aplikacije koja se koristi za poslovne svrhe (na odabranom primjeru aplikacije za stvaranje liste želja). Pritom je potrebno posebno proučiti i opisati sljedeća okruženja i njihovu primjenu: Django (besplatni web okvir otvorenog koda baziran na Pythonu) i Bootstrap (besplatni okvir za razvoj responzivnih web sjedišta ponajprije namijenjenih mobilnim uređajima). U radu će se prikazati izrada baze podataka koja podupire rad aplikacije, izrada korisničkog sučelja, ostvarivanje različitih (korisničkih) funkcionalnosti poput stvaranja korisničkog računa, autentikacija korisnika, te izmjena korisničkih podataka, izgradnja API-a, te ostalih funkcionalnosti vezanih uz samu namjenu aplikacije (npr., pregled, stvaranje, izmjena i brisanje liste želja, pretraživanje i prikaz liste po stranicama, i sl.).

Mentor:

Doc. dr. sc. Vanja Slavuj

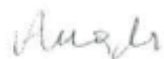


Komentor:

/

Voditeljica za diplomske radove:

Izv. prof. dr. sc. Ana Meštrović



Zadatak preuzet: 9.5.2022.

David Janeš



(potpis pristupnika)

Sadržaj

Sažetak	4
Ključne riječi	4
1. Uvod	5
2. Opis okruženja – web okviri	6
2.1. Django (web okvir).....	6
2.1.1. Povijest web okvira Django	7
2.1.2. Kako izgleda Django kod?	8
2.2. Bootstrap (web okvir).....	9
2.2.1. Povijest web okvira Bootstrap.....	11
2.3. JavaScript.....	11
2.3.1. JavaScript na strani klijenta.....	12
2.3.2. Prednosti JavaScript-a	12
2.3.3. Nedostatci JavaScript-a	13
3. Specifikacija aplikacije	14
4. Izrada web aplikacije za stvaranje lista želja.....	16
4.1. Stvaranje Django aplikacije.....	16
4.2. Izrada baze podataka	18
4.2.1. Relacijski model baze podataka	18
4.2.2. Model liste želja	19
4.2.3. Model proizvoda i model novčane valute	20
4.2.4. Model korisničkog računa.....	22
4.2.5. Stvaranje i registracija izgrađenih modela	22
4.3. Statične datoteke i instalacija početne teme	24
4.4. Izrada početne stranice	26
4.4.1. Navigacijska traka	30
4.4.2. Podnožje stranice.....	32
4.5. Korisnička autentifikacija i izgradnja pripadnih stranica	33
4.5.1. Registracija korisnika	33
4.5.2. Prijava i odjava korisnika	37
4.5.3. Izrada stranice korisničkog računa	39
4.5.4. Skočne poruke	44

4.5.5. Ponovno postavljanje lozinke.....	45
4.6. Stranica lista želja.....	55
4.6.1. Unos proizvoda na listu želja	61
4.6.2. Pretraga.....	69
4.6.3. Sortiranje proizvoda na listi	70
4.6.4. Paginacija	71
4.7. Otklanjanje grešaka, priprema i objava u produkciju.....	73
4.7.1. Priprema za objavu u produkciju	74
4.7.2. Objava u produkciju	74
5. Zaključak.....	76
Literatura	77
Popis slika	79
Prilozi	82

Sažetak

U ovome diplomskom radu prikazan je cjelokupni postupak izrade web aplikacije za stvaranje lista želja koristeći okruženja Django i Bootstrap. U prvom dijelu rada opisani su besplatni web okviri Django, Bootstrap i JavaScript programski jezik, njihova povijest, razvoj i princip rada. U glavnom dijelu rada detaljno je opisan postupak izrade aplikacije te svih njenih funkcionalnosti uz popratne slike ekrana i kodova. Na kraju rada dan je zaključak te osobni osvrt na korištenje dvaju web okvira.

Ključne riječi

Bootstrap, Django, dinamične i responzivne web aplikacije, JavaScript, lista želja, web okvir

1. Uvod

Mnoge internetske trgovine uz dodavanje artikla u košaricu u kojoj se obavlja proces kupnje proizvoda sadrže i mjesto za spremanje istih u svrhu lakšeg pronalaska u budućnosti. To mjesto obično se naziva lista želja (engl. *wishlist*). Koncept liste želja sličan je konceptu košarice, ali bez mogućnosti plaćanja, odnosno obavljanja procesa plaćanja. Na liste želja korisnici mogu spremati artikle koje žele ili planiraju kupiti u budućnosti te vidjeti ukupnu cijenu svih proizvoda na listi.

Iako većina današnjih internetskih trgovina već sadrže mogućnost stvaranja liste želja, glavni problem je taj što se na pojedinoj internetskoj trgovini i pripadnom mjestu za stvaranje liste mogu spremati proizvodi isključivo iz kataloga te trgovine. Glavna motivacija aplikacije stvorene u sklopu ovog diplomskog rada je riješiti navedeni problem, odnosno stvoriti univerzalno mjesto za stvaranje lista želja na koje se mogu unijeti proizvodi iz bilo koje internetske trgovine uz unos pripadnih informacija o proizvodu.

Aplikacija je zamišljena na način da se može koristiti u privatne svrhe, odnosno od strane individualnih korisnika, no mogu ju koristiti i poslovne organizacije ako je potrebno izraditi popis proizvoda koje je potrebno nabaviti, a oni se nalaze na različitim lokacijama, te vidjeti ukupnu cijenu.

Aplikacija je izrađena koristeći besplatne web okvire Django i Bootstrap, te u manjim količinama koristeći JavaScript. U slijedećem poglavlju opisana su korištena okruženja, njihova povijest i princip rada. U trećem poglavlju definirana je specifikacija aplikacije. U četvrtom, glavnom i najširem poglavlju, detaljno je opisana izrada cjelokupne web aplikacije od samog stvaranja projekta te izrade baze podataka pa sve do objave u produkciju, uz popratne slike ekrana i rezultata koda. U zadnjem poglavlju dan je zaključak i osobni osvrt na rad u okruženjima Django i Bootstrap.

2. Opis okruženja – web okviri

U ovome poglavlju definirat će se web okviri, odnosno programska okruženja korištena u izradi web aplikacije za stvaranje liste želja, provesti kroz povijest istih te opisati njihove funkcionalnosti. Korištena okruženja su web okviri Django i Bootstrap te programski jezik JavaScript.

2.1. Django (web okvir)

Django je *back-end* web okvir visoke razine (engl. *high-level*¹) baziran na programskom jeziku Python koji omogućuje brz razvoj zaštićenih web stranica koje je vrlo lako održavati. Budući da je izgrađen od strane iskusnih programera, nudi pregledni i pragmatični dizajn te osigurava bezbrižni razvoj web aplikacija bez potrebe za smišljanjem novih rješenja. Alat je besplatan i otvorenog koda (engl. *open-source*²) koji podupire velika aktivna zajednica, kao i opširna, detaljna i ažurna dokumentacija [1].

Django pomaže u razvoju softvera koji je:

- Potpun,
- Svestran,
- Zaštićen,
- Skalabilan,
- Održiv,
- Prijenosan.

Budući da je sve sadržano u jednom paketu, Django pruža sve što je programerima potrebno za realizaciju programskih rješenja. Svaki pojedini dio web okvira funkcionira besprijekorno s drugim, prati konzistentno načelo dizajna te sadrži ekstenzivnu dokumentaciju [2]. Na Slici 1 prikazan je primjer dokumentacije.

Quick example

This example model defines a **Person**, which has a **first_name** and **last_name**:

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

first_name and **last_name** are fields of the model. Each field is specified as a class attribute, and each attribute maps to a database column.

Slika 1. Primjer Django dokumentacije

Django se može koristiti za izgradnju gotovo bilo kakvog tipa web sjedišta, odnosno web aplikacije. Od menadžmentskih sustava pa sve do društvenih mreža ili stranica dnevnih novosti. Podržava bilo koji *front-end* web okvir te nudi ispis sadržaja u gotovo bilo kojem formatu

¹*High-Level* okruženje je bilo koje okruženje koje omogućuje brz i jednostavan razvoj računalnog rješenja te je pretežito neovisno o hardveru računala

²*Open-source* softver ili programski jezik je besplatno dostupan te se može dijeliti i prilagođavati prema potrebi

uključujući HTML, RSS, JSON, XML i druge. Također, iako nudi mnoštvo izbora za razne funkcionalnosti, nudi i mogućnost proširenja drugim komponentama prema potrebi kao što je npr. PostgreSQL baza podataka [2].

Sigurnost i zaštita kod Django sprječava česte programerske pogreške jer je izvedena na način da je korištenjem tog dijela web okvira aplikacija automatski zaštićena. Primjerice, nudi zaštićeni način upravljanja korisničkim računima i zaporkama izbjegavajući česte pogreške kao što su spremanje informacija u kolačiće (engl. *cookies*) sesije ili direktno spremanje lozinke, a ne *hash*³ lozinke. Django omogućuje zaštitu od sigurnosnih napada prema zadanim postavkama, uključujući SQL injekcije *cross-site* skriptiranje i krivotvorenje zahtjeva [2].

Svaki dio Django arhitekture neovisan je o drugome, što znači da se može mijenjati i prilagođavati prema potrebi. To također znači da se može skalirati prilikom povećanja prometa i korištenja aplikacije dodavajući hardver na bilo kojoj razini kao što su razina aplikacijskih servera, servera baze podataka i slično. Neka i od najvećih web sjedišta današnjice, koja koriste Django kao *back-end* jezgru, uspješno su skalirale prema zahtjevima i potrebama (npr. Instagram i Disqus) [2].

Django kod pisan je koristeći načela i uzorke dizajna koji omogućuju pisanje održivog koda kojeg je moguće i ponovo iskoristiti. Smanjena je količina koda te je izbjegnuta nepotrebna repeticija prilikom samog pisanja koda. Moguće je i grupirati neke povezane funkcionalnosti u module [2].

Django web okvir baziran je na programskom jeziku Python koji je podržan na mnogo platformi. To znači da nije neophodno koristiti određenu platformu te se može koristiti na sustavima kao što su Linux, Windows i macOS. Također, Django je podržan na mnogo poslužitelja za dijeljenje web sjedišta koji nude specifičnu infrastrukturu i dokumentaciju za dijeljenje Django web aplikacija [2].

2.1.1. Povijest web okvira Django

Django web okvir razvijen je između 2003. i 2005. godine od strane tima web programera koji je izvorno programirao i održavao web sjedišta dnevnih novosti, posebice web sjedište Lawrence [3]. Nakon razvoja mnogo web sjedišta spomenuti tim web programera počeo je izdvajati određene dijelove koda, odnosno uzorke koda koje je moguće ponovo koristiti. Takav uobičajen i ponavljajući kod razvio se u generički web okvir koji je pokrenut kao Django projekt otvorenog koda u srpnju 2005. godine [2].

Kao takav nastavio se razvijati i poboljšavati gdje je, od svoje prve objave verzije 1.0 u rujnu 2008. godine do najnovije objave verzije 4.0 2022. godine, svakom objavom uključio neku novu funkcionalnost i ispravke pogrešaka (engl. *bug fixes*), kao što su novi tipovi baza podataka, predlošci i dodatci generičkih funkcija i klasa koje smanjuju količinu pisanog koda [2].

³Hash lozinke je vrijednost fiksne duljine stvorena slanjem lozinke kroz kriptografsku funkciju.

Django je danas uspješan, suradnički projekt otvorenog koda s mnogo korisnika i doprinositelja diljem svijeta. Dok i dalje sadrži neke izvorne značajke, Django se razvio u svestrani web okvir pomoću kojeg je moguće izgraditi bilo kakav tip web sjedišta, odnosno web aplikacije [2].

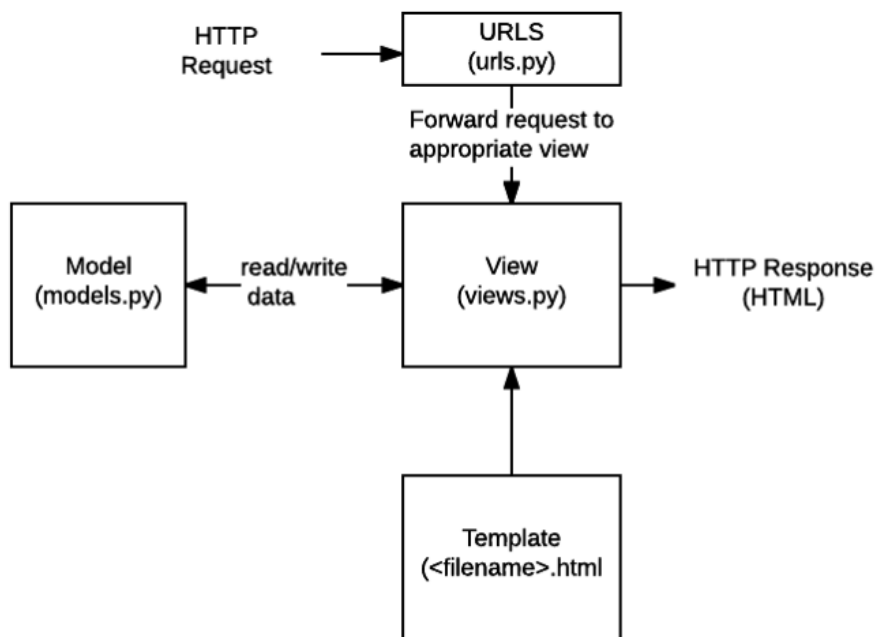
2.1.2. Kako izgleda Django kod?

Kod tradicionalnih podatkovnih web sjedišta, web aplikacija čeka HTTP zahtjev od strane klijenta putem nekog web preglednika. Nakon primitka zahtjeva, aplikacija obrađuje taj zahtjev baziran na URL-u te eventualno POST i GET podacima. Ovisno o tome što je potrebno napraviti, aplikacija čita ili zapisuje podatke iz, odnosno u bazu podataka u svrhu zadovoljavanja zahtjeva. Nakon toga aplikacija vraća odgovor klijentu u obliku dinamički stvorene HTML stranice koju web preglednik tada prikazuje [2].

Programski kod Django web aplikacije uglavnom je grupiran na način da obrađuje pojedine korake u posebnim datotekama, a glavne grupacije su sljedeće:

- URL-ovi,
- Pregledi (engl. *views*),
- Modeli (engl. *models*),
- Predlošci (engl. *templates*).

Na Slici 2 prikazana je shema Django koda.



Slika 2. Shema Django koda [2]

Iako je svaki URL zahtjev moguće obraditi jednom funkcijom, puno više prostora za održavanje nudi se stvaranjem posebne funkcije pregleda za obradu pojedinog resursa. *URL mapper* koristi se za preusmjeravanje HTTP zahtjeva na odgovarajući pregled baziran na URL zahtjevu. *URL mapper* također može proslijediti i određeni uzorak znakova ili znamenki koje se mogu pojaviti u URL-u u funkciju pregleda kao zasebni podatak [2].

Pregled je funkcija koja obrađuje zahtjeve, dakle zaprima HTTP zahtjeve i šalje HTTP odgovore. Pregledi pristupaju podacima, odnosno obrađuju podatke koji su potrebni da bi se zahtjev zadovoljio putem modela te oblikuju odgovor prikazan u predlošcima [2].

Modeli su Python objekti koji definiraju strukturu podataka aplikacije te omogućuju CRUD operacije nad bazom podataka [2]. CRUD je u području programiranja akronim za radnje stvaranja, čitanja, ažuriranja i brisanja. Dakle, CRUD operacije su [4]:

- Stvaranje (engl. *create*),
- Čitanje (engl. *read*),
- Ažuriranje (engl. *update*),
- Brisanje (engl. *delete*).

Funkcija stvaranja omogućuje korisnicima stvaranje novih zapisa ili instanci modela unutar baze podataka. Funkcija čitanja omogućuje korisnicima pretragu i dohvaćanje pojedinih zapisa unutar baze podataka. Funkcija ažuriranja koristi se za uređivanje postojećih zapisa ili informacija instance modela unutar baze podataka. Funkcija brisanja omogućuje korisnicima brisanje zapisa ili instance modela unutar baze podataka. Razlikuju se dva način brisanja zapisa: trajno brisanje (engl. *hard delete*), čime se zapis potpuno briše iz baze, te označavanje zapisa za brisanje (engl. *soft delete*), čime sam zapis ostaje u bazi, ali s određenom oznakom da je izbrisan [4].

Predlošci su datoteke tekstualnog formata koje definiraju strukturu i raspored stranice (npr. HTML stranica). Sadrže predodređena mjesta koja se ovisno o zahtjevu ispune s podacima iz modela što je ostvarivo u relaciji s pregledom. Predložak može definirati strukturu bilo kakvog tipa datoteke i ne mora nužno biti HTML [2].

2.2. Bootstrap (web okvir)

Bootstrap je bogati i vrlo fleksibilni *front-end* web okvir koji pruža besplatnu kolekciju alata za izgradnju web sjedišta i web aplikacija. Koristi HTML, CSS i JavaScript komponente, što omogućuje brz i jednostavan razvoj responzivnih, primarno mobilnih, web sjedišta. Alat je besplatan, otvorenog koda te se dijeli i održava na GitHub-u [5].

Alat uključuje slijedeće [5]:

- Globalne stilove, fluidni mrežasti sustav i raspored stranice te responzivni dizajn;
- Bazični CSS kod koji sadrži korijen HTML elemenata za tipografiju, tablice, obrasce, gumbe, slike i mnoge druge koji se mogu prilagoditi i nadograditi raznim klasama;
- Opsežnu listu komponenata za višekratnu uporabu kao što su padajuće liste, navigacija, trake napretka, skočni okviri, ikone, značke te mnogo drugih;
- JavaScript dodatke kao što je jQuery dodatak za prijelazne efekte, klizače, karusele i mnogo drugih;
- Visoku razinu prilagodljivosti što omogućuje programerima stvaranje vlastite verzije Bootstrap komponenti, klasa i dodataka.

Neke od glavnih prednosti web okvira Bootstrap su:

- Jednostavna instalacija i uporaba,
- Konzistentnost i kompatibilnost s mnogo web preglednika,
- Primarno mobilni pristup,
- Responzivni raspored mreže,
- Široka paleta mogućnosti,
- Često ažuriranje.

Bootstrap koristi dva popularna CSS formata: Less⁴ (engl. *leaner style sheets*) i Sass⁵ (engl. *syntactically awesome stylesheets*). No, također pruža i običnu CSS datoteku koju je moguće preuzeti, raspakirati i referencirati u zaglavlju HTML datoteke. U svega nekoliko koraka omogućen je potpuni pristup cijelom web okviru koji programeri mogu koristiti prema zadanim postavkama ili prilagoditi prema potrebi [5].

Jedna od glavnih ideja alata je stvaranje centraliziranog i višenamjenskog koda te uniformiranje krajnjeg rezultata na svim platformama, odnosno uređajima. Dodatno, alat podržava najnovije verzije svih popularnih web preglednika, ali se također primjereno prilagođava i starijim verzijama istih što omogućuje besprijekoran rad neovisan o platformi [5].

Počevši od verzije alata 2.0, web okvir podržava potpuno responzivni web dizajn što znači da se web stranice dinamički prilagođavaju tipu i veličini uređaja koji se koristi za pregled [5].

Najvažnija značajka Bootstrap web okvira je korištenje responzivnog sustava mreže koja stranicu dijeli na 12 simetričnih stupaca. Takvim sustavom je lako ostvariva raspodjela stranice u nekoliko različitih odjeljaka. Stupci se mogu definirati pomoću nekoliko različitih klasa, ovisno o veličini ekrana, a te klase su: *col-xs*, *col-sm*, *col-md* i *col-lg*. Sve klase definirane su u sinkronizaciji s razlučivošću uređaja te sve što je potrebno od strane programera je primijeniti ih prilikom razvoja elemenata [5].

Bootstrap pruža veliki niz prilagodljivih HTML i CSS komponenti, JavaScript dodataka, obrazaca, responzivnih mreža od 12 stupaca i drugih komponenti. Potrebno je svega nekoliko promjena u klasama za ostvariti željeni rezultat. Programeri imaju besplatni pristup bilo kojoj klasi ili komponenti koje dinamički mogu koristiti i prilagođavati bez narušavanja performansi i preglednosti koda [5].

Bootstrap je jedan od najažuriranih web okvira dostupnih na webu. Ako se pojavi bilo koja vrsta problema, odnosno pogreške, Bootstrap-ov tim programera u kratkom roku identificira i riješi problem uz objavu nove verzije. Zbog takvog načina poslovanja korisnici uvijek mogu biti sigurni da rade s najnovijim i sigurnim alatom koji je podržan bogatom i opsežnom dokumentacijom [5]. Na Slici 3 prikazan je primjer Bootstrap dokumentacije.

⁴Less je paleta CSS koda koja pomaže u pisanju preglednog, optimiziranog, dinamičnog i agilnog koda [6]

⁵Sass je bogati dodatak za CSS koji pomaže u stvaranju i održavanju preglednog koda za višekratnu upotrebu [7]

Sizing

Set heights using classes like `.form-control-lg` and `.form-control-sm`.



Slika 3. Primjer Bootstrap dokumentacije

2.2.1. Povijest web okvira Bootstrap

Bootstrap, izvornog naziva Twitter Blueprint, razvijen je od strane Marka Otta i Jacoba Thorntona unutar tvrtke Twitter kao web okvir kojim se potiče konzistentnost korištenja alata za razvoj. Prije Bootstrap-a korištene su razne biblioteke za razvoj korisničkog sučelja što je dovelo do nekonzistentnosti i teškog održavanja [8].

Mala skupina programera i Mark Otto udružili su se i izgradili novi interni alat u kojemu su vidjeli priliku za postići i nešto više. Nekoliko mjeseci od početka razvoja izgrađena je vrlo rana verzija Bootstrapa kao način za dokumentiranje i dijeljenje uzoraka dizajna unutar tvrtke. Nakon još nekoliko mjeseci, mnogo drugih programera Twittera također počinje pridonositi projektu. Tada je projekt preimenovan iz Twitter Blueprint u Bootstrap i objavljen kao projekt otvorenog koda 19. kolovoza 2011. godine. U siječnju 2012. godine, objavljen je Bootstrap 2 u kojem je dodana mreža od 12 stupaca i komponente responzivnog dizajna. Bootstrap 3 objavljen je 19. kolovoza 2013. godine, a u kojem je predstavljen primarno mobilni pristup dizajna. Dana 29. listopada 2014. godine objavljeno je da je Bootstrap 4 u razvoju te je prva verzija objavljena 19. kolovoza 2015. godine [8].

Najnovija verzija, Bootstrap 5, fokusirana je na poboljšanje verzije 4 s vrlo malo velikih promjena. Nadograđene su već postojeće mogućnosti i komponente, uklonjena podrška za starije web preglednike te je jQuery zamijenjen s običnim JavaScript-om. Bootstrap je postao jedan od najpopularnijih *front-end* web okvira kao i općenito projekata otvorenog koda [9].

2.3. JavaScript

JavaScript (skraćeno JS) je dinamični programski jezik koji se najčešće koristi kao dio web stranica čija implementacija omogućuje skripti sa strane klijenta interakciju s korisnikom, te stvaranje dinamičnih stranica. Interpretacijski je programski jezik s objektno-orijentiranim mogućnostima [10].

JavaScript se izvorno nazivao LiveScript kada se pojavio 1995. godine u sklopu objave web preglednika Netscape 2.0, no Netscape je kasnije promijenio njegovo ime u JavaScript. Izvorna

svrha, funkcionalnost i podrška programskog jezika JavaScript bila je ugrađena u Netscape, Internet Explorer i druge web preglednike [10].

Glavne značajke JavaScript jezika su [10]:

- Interpretacijski programski jezik s niskom razinom upotrebe radne memorije i procesora;
- Dizajniran za izgradnju mrežno orijentiranih aplikacija;
- Komplementaran i integriran s Java programskim jezikom;
- Komplementaran i integriran s HTML-om;
- Podržan na više platformi (engl. *cross-platform*).

2.3.1. JavaScript na strani klijenta

Najzastupljeniji oblik jezika je JavaScript na strani klijenta (engl. *client-side*). Skripta JavaScript-a uključena je i referencirana unutar HTML dokumenta, koju zatim interpretira web preglednik. Web stranica ne mora nužno biti statična HTML datoteka, već ona može uključivati i programe koji su u interakciji s korisnikom, upravljaju web preglednikom i dinamički stvaraju HTML sadržaj [10].

Mehanizam JavaScript-a na strani klijenta pruža mnoge prednosti nad tradicionalnim skriptama sa strane poslužitelja. Primjerice, korištenjem JS-a moguće je provjeriti je li korisnik unio ispravnu E-mail adresu u obrazac. JS kod izvrši se kada korisnik pošalje zahtjev samo ako su svi unosi, ispravni te se tada obrazac prosljeđuje do web poslužitelja. JS se također može koristiti za dohvaćanje događaja pokrenutih od strane korisnika, kao što su klikovi na gumbе, navigacijske poveznice ili bilo koje druge radnje koju korisnik izvrši [10].

2.3.2. Prednosti JavaScript-a

Jedna od najveći prednosti JS-a je ta što nije potrebno kupiti neki od skupih alata za razvoj. Dovoljan je običan uređivač teksta za početi s radom. Također, budući da je riječ o interpretacijskom jeziku u sklopu web preglednika, nije potrebno kupiti ni kompajler [10].

Neke od ostalih prednosti JS-a su slijedeće:

- Manje interakcije s poslužiteljem;
- Vraćanje povratne informacije istog trena;
- Povećana interaktivnost;
- Bogatija korisnička sučelja.

Mogućnost potvrde korisničkog unosa prije slanja zahtjeva poslužitelju rezultira smanjenjem i uštedom prometa. Time se ujedno smanjuje i opterećenost poslužitelja.

Korisnici ne moraju čekati da se stranica osvježi kako bi vidjeli jesu li zaboravili unijeti neku informaciju u obrazac.

JS-om je moguće izgraditi dinamična korisnička sučelja koja reagiraju na određeni način prilikom korisnikova prelaska kursorom preko istih ili aktivacijom putem tipkovnice.

JavaScript se također može koristiti za uključanje povuci-i-spusti (engl. *drag-and-drop*) komponentata, raznih klizače i mnogo drugih značajki koje posjetiteljima web sjedišta pružaju bogato korisničko sučelje [10].

2.3.3. Nedostatci JavaScript-a

Iako JavaScript nudi bogatu paletu mogućnosti, ne može se smatrati punopravnim programski jezikom zbog nedostatka slijedećih funkcionalnosti [10]:

- JavaScript na strani klijenta ne dozvoljava čitanje i pisanje datoteka - razlog tome je primarno sigurnost;
- Ne može se koristiti za mrežne aplikacije jer ne pruža podršku za iste;
- Ne može koristiti više od jedne jezgre procesora.

3. Specifikacija aplikacije

U ovom poglavlju definirana je specifikacija aplikacije, opisani su korisnički zahtjevi te metoda razvoja.

Web aplikacija nazvana je *MyWishlist* iza koje je izvorna ideja stvoriti univerzalnu platformu za stvaranje lista želja. Mnoge internetske trgovine uz košaricu za kupovinu sadrže i vlastiti odjeljak za stvaranje liste želja. Međutim, u tu listu moguće je spremati isključivo proizvode ponuđene u katalogu te trgovine. Aplikacija *MyWishlist* zamišljena je kao rješenje tog problema te se u nju mogu spremati proizvodi s bilo koje internetske trgovine i referencirati na istu korištenjem poveznica.

Razvoj aplikacije *MyWishlist* koristi agilnu metodologiju, odnosno u više navrata tijekom razvoja prikupljane su povratne informacije korisnika te su time oblikovani i korisnički zahtjevi. Primarno je aplikacija zamišljena kao rješenje za osobne potrebe, no nju mogu koristiti i određene tvrtke ili obrti koji žele sastaviti listu proizvoda koje je potrebno nabaviti, a ti se proizvodi nalaze na različitim web sjedištima, te čuvati sve na jednome mjestu.

Za korištenje ove aplikacije potreban je uređaj koji podržava bilo koji od web preglednika te internetska veza, a konkretne funkcionalnosti su slijedeće:

- Korisnicima je ponuđeno jedinstveno sučelje za svakog pojedinog korisnika – svaki korisnik može vidjeti i uređivati isključivo vlastite liste;
- Za korištenje svih funkcionalnosti koje web aplikacija nudi potrebno je stvoriti korisnički račun i prijaviti se;
- Korisnicima je dostupan dinamički prikaz svih stranica, ovisno o tome je li korisnik prijavljen, ima li listu želja, koliko proizvoda na listi ima, uređuje li listu ili stvara novu, odabranom sortiranju, zadnjem uređivanju, pretraživanju i trenutnoj lokaciji;
- Korisnici mogu kreirati vlastiti korisnički račun i prijaviti se;
- Korisnici mogu izmijeniti ili nadopuniti informacije korisničkog računa;
- Korisnici mogu poništiti i ponovo postaviti lozinku prilikom prijave (potrebna funkcionalna E-mail adresa);
- Korisnici mogu ponovo postaviti lozinku u postavkama korisničkog računa;
- Aplikacija temeljito upravlja i kontrolira greške prilikom unosa i slanja obrasca;
- Korisnici mogu koristiti jednostavnu i intuitivno dizajniranu navigaciju za kretanje po aplikaciji;
- Korisnici mogu stvarati, čitati, ažurirati i brisati liste želja;
- Korisnici mogu stvarati, čitati, ažurirati i brisati proizvode, odnosno informacije o proizvodima unutar pojedinih lista želja;
- Korisnici mogu pretraživati liste želja po nazivu;
- Korisnici mogu sortirati proizvode padajuće ili rastuće na određenoj listi po nazivu, cijeni, količini i prioritetu;
- Korisnici mogu koristiti paginaciju za pretraživanje lista koje su dostupne, ako je s određenim korisničkim računom povezano više od 10 lista želja;

- Korisnicima je unaprijed zadano nekoliko novčanih valuta koje je na određenoj listi potrebno izjednačiti da bi se prikazao ukupni izračun;
- Korisnicima je skočnim prozorom prikazana poruka uspjeha, neuspjeha ili upozorenja prilikom pojedinog slanja obrasca.

4. Izrada web aplikacije za stvaranje lista želja

U ovom poglavlju detaljno je opisan postupak izrade web aplikacije za stvaranje lista želja koristeći okruženja Django, Bootstrap i JavaScript. Opisan je cjelokupni postupak, počevši od samog stvaranja Django projekta, izrade modela i baze podataka, pa sve do objave aplikacije u produkciju.

4.1. Stvaranje Django aplikacije

Prije svega, da bi se započeo rad na Django aplikaciji potreban je uređivač programskog koda te pristup naredbenoj liniji. Za uređivač programskog koda odabran je Visual Studio Code unutar kojeg je također moguć i pristup naredbenoj liniji. Dakle, sve radnje obavljene su unutar sučelja alata Visual Studio Code.

Zatim, instaliran je Python na računalo kojeg je moguće preuzeti na službenom web sjedištu (<https://www.python.org/>) [11]. Nakon instalacije Pythona instaliran je i sam Django jednostavnom naredbom unutar naredbene linije i stvorenog virtualnog okruženja (Slika 4).

```
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> pip install django
Requirement already satisfied: django in c:\users\davidjanes\desktop\full stack\mywishlist\.venv\lib\site-packages (4.0.3)
Requirement already satisfied: asgiref<4,>=3.4.1 in c:\users\davidjanes\desktop\full stack\mywishlist\.venv\lib\site-packages (from django) (3.5.0)
Requirement already satisfied: sqlparse>=0.2.2 in c:\users\davidjanes\desktop\full stack\mywishlist\.venv\lib\site-packages (from django) (0.4.2)
Requirement already satisfied: tzdata in c:\users\davidjanes\desktop\full stack\mywishlist\.venv\lib\site-packages (from django) (2022.1)
```

Slika 4. Instalacija Django web okvira

Slijedećom jednostavnom naredbom prikazanom na Slici 5, stvoren je projekt *mywishlist*.

```
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> django-admin startproject mywishlist
```

Slika 5. Stvaranje Django projekta

Zatim su, unutar projekta *mywishlist*, stvorene dvije aplikacije koje su prema specifikaciji potrebne: *account* i *wishlists*. One sadrže sve dijelove koda kao što su URL-ovi, pregledi, modeli, obrasci te ostale pomoćne datoteke. Na Slici 6 nalazi se prikaz stvaranja tih dviju aplikacija.

```
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> python manage.py startapp account
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> python manage.py startapp wishlists
```

Slika 6. Stvaranje dviju aplikacija unutar projekta

Nakon uspješnog stvaranja navedenih dvaju aplikacija, a da bi one bile povezane i funkcionalne unutar projekta, u datoteci *settings.py* dodane su u listu `INSTALLED_APPS` (Slika 7).

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'wishlists.apps.WishlistsConfig',
    'account.apps.AccountConfig',
]
```

Slika 7. Povezivanje aplikacija s projektom

Također, stvorene su i dvije nove datoteke *urls.py*, jedna unutar mape *account* i jedna unutar mape *wishlists*, koje su potrebne za generiranje URL adresa svih pripadajućih HTML stranica tih dviju aplikacija. U slijedećem dijelu koda, prikazanim na Slici 8, novostvorene datoteke povezane s glavnom *urls.py* datotekom projekta u mapi *mywishlist*.

```

> .env
  > account
  > _pycache_
  > migrations
  > templates
  > _init_.py
  > admin.py
  > apps.py
  > forms.py
  > models.py
  > singals.py
  > tests.py
  > urls.py
  > views.py
  > api
  > mywishlist
  > _pycache_
  > _init_.py
  > asgi.py
  > forms.py
  > settings.py
  > urls.py
  > views.py
  > wsgi.py
  > static
  > templates
  > wishlists
  > _pycache_
  > migrations
  > templates
  > _init_.py
  > admin.py
  > apps.py
  > forms.py
  > models.py
  > tests.py
  > urls.py
  > utils.py
  > views.py

urls.py
mywishlist > urls.py > ...
1 from django.contrib import admin
2 from django.urls import path, include
3 from django.conf import settings
4 from django.conf.urls.static import static
5 from . import views
6
7 from django.contrib.auth import views as auth_views
8
9
10 urlpatterns = [
11     path('admin/', admin.site.urls),
12     path('', views.home_view, name="home"),
13     path('home/', views.home_view, name="home"),
14     path('', include('wishlists.urls')),
15     path('', include('account.urls')),
]

account > urls.py > ...
1 from django.urls import path
2 from . import views
3
4 from django.contrib.auth import views as auth_views
5
6 urlpatterns = [
7
8     path('account/', views.account_view, name="account"),
9
10 ]

wishlists > urls.py > ...
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('wishlists/', views.get_all_wishlists, name="wishlists"),
6     path('wishlist/<str:pk>', views.get_wishlist_details, name="wishlist"),
7 ]
  
```

Slika 8. Prikaz poveznice url.py datoteka između aplikacija

Prije izrade modela i baze podataka stvorena je još i mapa za spremanje generalnih HTML predložaka, odnosno predložaka koji nisu specifični za pojedinu aplikaciju unutar projekta. Dakle, stvorena je mapa *templates* u korijenskoj mapi projekta. Da bi Django mogao dohvatiti datoteke iz navedene mape, u datoteci *settings.py* potrebno je dodati dio koda prikazanog na Slici 9.

```

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'templates'),
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
  
```

Slika 9. Povezivanje Djanga s mapom za HTML predloške

4.2. Izrada baze podataka

Za početak je korištena baza SQLite koja dolazi zajedno s Django prilikom stvaranja projekta, te je prema zadanim postavkama već konfigurirana za korištenje. Praktična je i jednostavna za korištenje te omogućuje obavljanje svih testova potrebnih prilikom izrade modela. Budući da Django sam izradi sve potrebne početne tablice, sve što je potrebno da bi se one i izradile je pokrenuti sljedeću naredbu prikazanu na Slici 10.

```
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> python manage.py migrate
```

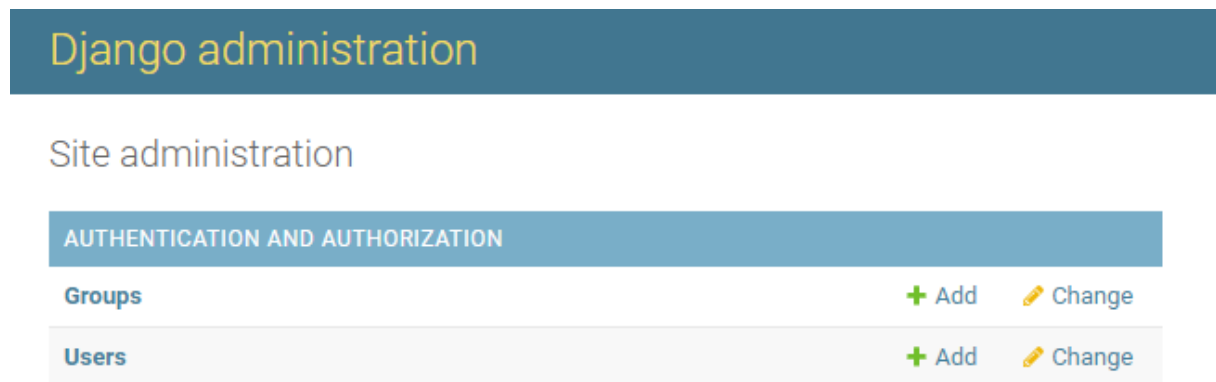
Slika 10. Naredba za izradu početnih tablica

Da bi pristup administracijskom sučelju bio moguć, potrebno je stvoriti korisnika. Slijedećom naredbom i ispunjavanjem informacija zahtjeva, prikazanim na Slici 11, stvoren je korisnik s potpunim ovlastima (engl. *superuser*).

```
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> python manage.py createsuperuser
Username (leave blank to use 'davidjanes'): admin
Email address: admin@gmail.com
Password:
Password (again):
```

Slika 11. Stvaranje korisnika s potpunim ovlastima

Nakon stvaranja korisnika s potpunim ovlastima, omogućena je prijava s prethodno unesenim podacima u administracijsko sučelje u kojem je moguć pregled svih trenutnih tablica u bazi podataka, kao i njihova izmjena, pisanje i brisanje (CRUD). Na Slici 12 prikazano je administracijsko sučelje Django aplikacije.



Slika 12. Administracijsko sučelje Django aplikacije

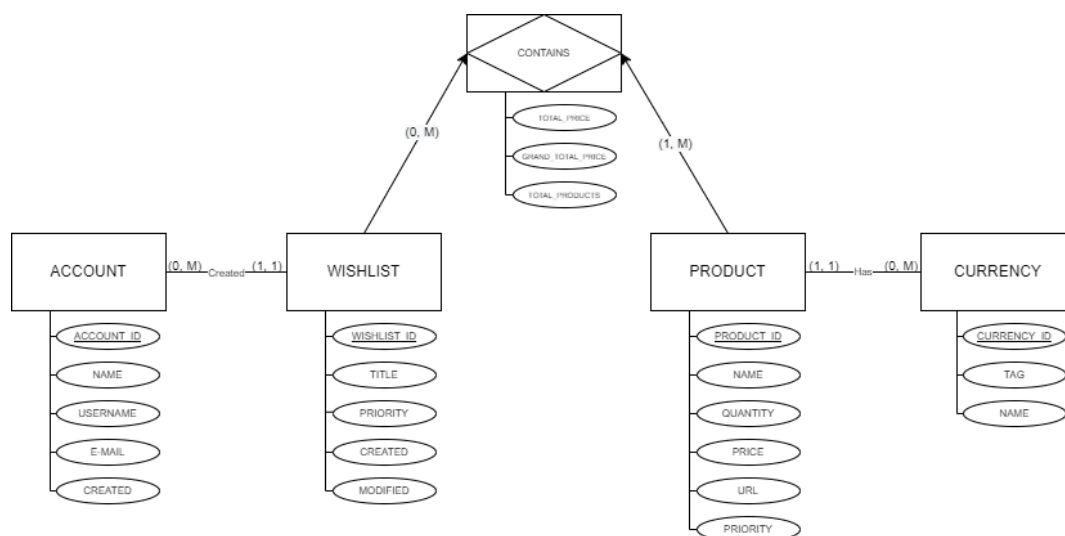
4.2.1. Relacijski model baze podataka

Relacijski model baze prema specifikaciji je prilično jednostavan te se sastoji od svega nekoliko relacija:

- Account (account_ID, name, username, e-mail, created);
- Wishlist (wishlist_ID, title, priority, created, modified, account_ID)
- Product (product_ID, name, quantity, price, url, priority, currency_ID, wishlist_ID)
- Currency (currency_ID, tag, name)

- Contains (total_price, grand_total_price, total_products)

Korisnički račun može imati nijednu ili više lista želja, dok pojedina lista želja može pripadati samo jednom korisničkom računu. Lista želja može sadržavati nijedan ili više proizvoda, dok jedan proizvod može pripadati jednoj ili više lista. Nemoguće je unijeti proizvod bez da pripada nekoj listi. U konačnici, proizvod može u određenom trenutku imati samo jednu novčanu valutu, dok jedna novčana valuta može pripadati nijednom ili više proizvoda. Na Slici 13 prikazan je dijagram entiteti-veze baze izrađen koristeći pristup MIRIS [12] u alatu *diagrams.net* [13].



Slika 13. Dijagram entiteti-veze

4.2.2. Model liste želja

Definicija modela u Django web okviru odrađuje se unutar *models.py* datoteke. Svaki model je definirana Python klasa koja nasljeđuje svojstva iz pred definiranih Django modela. Dakle, stvorena je klasa *Wishlist* te su definirani određeni atributi, a to su: *id*, *owner*, *title*, *priority*, *created* i *modified*.

Atribut *ID* označuje primarni ključ koji se generira pri svakom stvaranju nove instance modela što je definirano svojstvima *unique=True* i *primary_key=True*. Također, to polje ne može se uređivati (*editable=False*) te je zapisano u formatu *uuid4* (*default=uuid.uuid4*). *Title* je polje definirano kao znakovno polje ograničeno na 200 znakova (*max_length=200*). Polje *priority* definirano je na nešto drukčiji način, odnosno na način koji će u korisničkom sučelju, prema zadanim postavkama, prikazati padajuću listu s pred definiranim izborima. Ti izbori definirani su kao pod klasa unutar klase *Wishlist* (*PriorityChoices*). Nadalje, dodani su atributi *created* i *modified* koji označavaju vrijeme stvaranja instance te datum posljednjeg uređivanja. Svojstvom *auto_now_add=True* atributu se pridružuje trenutno vrijeme prilikom stvaranja instance, a svojstvom *auto_now=True* pridružuje se trenutno vrijeme prilikom svakog spremanja, odnosno ažuriranja instance.

Što se tiče atributa *owner*, definiran je kao vanjski ključ modela *Account*. Sintaksa je prilično jednostavna te Django automatski povezuje ta dva modela pripadnim ključevima. Budući da je riječ o modelu iz druge aplikacije unutar Django projekta, potrebno je napraviti uvoz tog

modela slijedećom naredbom: `from account.models import Account`. Također, polje je definirano kao polje koje može biti prazno te se prilikom brisanja korisničkog računa brišu i sve povezane liste želja (`on_delete=models.CASCADE`).

Klasi *Wishlist* također je dodano i nekoliko različitih svojstva kao što je zadani poredak podataka prema datumu posljednje izmjene te dva izračuna, od kojih je jedan ukupni broj proizvoda, a drugi ukupna cijena svih proizvoda na listi. Prilikom definiranja spomenutih funkcija pridruženo je svojstvo `@property`, što omogućuje poziv tih funkcija u pregledima na isti način kao što se pozivaju i atributi, te bazična kontrola grešaka pomoću `try` i `except` funkcija. Na Slici 14 prikazana je cjelokupna definicija modela *Wishlist*.

```
class Wishlist(models.Model):

    class PriorityChoices(models.IntegerChoices):
        low = 1, _('Low')
        medium = 3, _('Medium')
        high = 5, _('High')

    id = models.UUIDField(default=uuid.uuid4, unique=True, primary_key=True, editable=False)
    owner = models.ForeignKey(Account, null=True, blank=True, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    priority = models.IntegerField(choices=PriorityChoices.choices, default=PriorityChoices.low)
    created = models.DateTimeField(auto_now_add=True)
    modified = models.DateTimeField(auto_now=True)

    class Meta:
        ordering = ['-modified']

    def __str__(self):
        return self.title

    @property
    def total_products(self):
        try:
            products = self.product_set.all()
            return products.count()
        except AttributeError:
            return 0

    @property
    def grand_total_price(self):
        try:
            products = self.product_set.all()
            total = 0
            for product in products:
                total += product.total_price
            return total
        except AttributeError:
            return 0
```

Slika 14. Definicija modela *Wishlist*

4.2.3. Model proizvoda i model novčane valute

Model proizvoda definiran je na sličan način kao i model liste želja: definirana je klasa s određenim atributima, a to su: *id*, *name*, *quantity*, *price*, *currency*, *url*, *priority* i *wishlist*.

Atribut *ID* označuje primarni ključ modela, *name* znakovno polje s najviše 2000 znakova, *quantity* je definirano kao *integer* polje sa zadanom vrijednošću 1 (*default=1*), *url* je znakovno polje koje nije nužno popunjeno, te *priority* koje, kao i kod modela liste želja, sadrži pred-definirane izbore.

Atributi *currency* i *wishlist* definirani su kao vanjski ključevi koji tvore poveznicu na modele *Currency* i *Wishlist*.

Modelu proizvoda također su pridružena i određena svojstva kao što je spremanje roditelja, odnosno modela liste želja prilikom svakog spremanja pojedinog proizvoda što je potrebno da bi se ispravno vodilo vrijeme izmjene svake liste. Drugo svojstvo je funkcija koja radi izračun ukupne cijene pojedinog proizvoda na temelju jedinične cijene i količine. Na Slici 15 prikazana je cjelokupna definicija modela *Product*.

```
class Product(models.Model):

    class PriorityChoices(models.IntegerChoices):
        low = 1, _('Low')
        medium = 3, _('Medium')
        high = 5, _('High')

    id = models.UUIDField(default=uuid.uuid4, unique=True, primary_key=True, editable=False)
    name = models.CharField(max_length=2000)
    quantity = models.IntegerField(default=1)
    price = models.DecimalField(max_digits=9, decimal_places=2, default=0.00)
    currency = models.ForeignKey('Currency', on_delete=models.CASCADE)
    url = models.CharField(max_length=2000, blank=True, null=True)
    priority = models.IntegerField(choices=PriorityChoices.choices, default=PriorityChoices.low)
    wishlist = models.ForeignKey('Wishlist', null=True, blank=True, on_delete=models.CASCADE)

    def __str__(self):
        return self.name

    def save(self):
        super(Product, self).save()
        self.wishlist.save()

    @property
    def total_price(self):
        return self.price * self.quantity
```

Slika 15. Definicija modela *Product*

Model novčane valute jednostavan je model koji sadrži svega nekoliko atributa, a to su: *id*, *tag* i *name*. Atribut *ID*, kao i kod ostalih modela, znakovno je polje od 16 znakova koje označuje primarni ključ modela, polje *tag* je znakovno polje ograničeno na 3 znaka (npr. HRK, EUR i slično) te *name*, što je znakovno polje ograničeno na 200 znakova. Budući da je riječ o jednostavnim podacima pred-definiranim od strane administratora, nisu potrebna stoga ni definirana posebna dodatna svojstva. Na Slici 16 prikazana je cjelokupna definicija modela *Currency*.

```

class Currency(models.Model):
    id = models.UUIDField(default=uuid.uuid4, unique=True, primary_key=True, editable=False)
    tag = models.CharField(max_length=3)
    name = models.CharField(max_length=200)

    def __str__(self):
        return self.tag

```

Slika 16. Definicija modela *Currency*

4.2.4. Model korisničkog računa

Model *Account* definiran je u *models.py* datoteci unutar *account* aplikacije Django projekta. Na taj način sav relevantni kod drži se na jednom mjestu, odnosno u jednoj mapi. Kao i ostali modeli, također je definiran kao Python klasa s pripadnim atributima, a to su: *id*, *user*, *name*, *username*, *email* i *created*.

Atribut *ID* označuje primarni ključ modela te je također definiran kao *uuid4* polje. *Name* i *username* su znakovna polja ograničena na 200 znakova, no korisničko ime mora biti jedinstveno. *Email* je e-mail polje također ograničeno na 200 znakova te mora biti jedinstveno, dok polje *created* pridružuje vrijeme stvaranja instance.

Budući da Django prema zadanim postavkama sadrži već pred-definirani model korisnika, model *Account* stvoren je kao proširenje na već zadani model povezan atributom *user*, što je vanjski ključ. Jedna instanca modela *Account* može pripadati jednoj i samo jednoj instanci modela *User* i obratno. Da bi to bilo moguće, prije svega iz web okvira uvezen je model *User* uvezen te je zatim definirana klasa. Na Slici 17 prikazana je cjelokupna definicija modela *Account*.

```

from django.db import models
from django.contrib.auth.models import User
import uuid

User._meta.get_field('email')._unique=True

class Account(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, null=True, blank=True)
    name = models.CharField(max_length=200, null=True, blank=True)
    username = models.CharField(max_length=200, null=True, blank=True, unique=True)
    email = models.EmailField(max_length=200, null=True, blank=True, unique=True)
    created = models.DateTimeField(auto_now_add=True)
    id = models.UUIDField(default=uuid.uuid4, unique=True, primary_key=True, editable=False)

    def __str__(self):
        return str(self.user.username)

```

Slika 17. Definicija modela *Account*

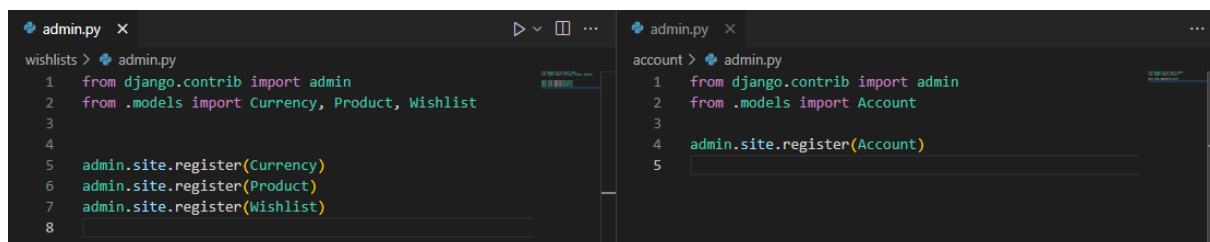
4.2.5. Stvaranje i registracija izgrađenih modela

Nakon kodiranja Python klasa, odnosno Django modela, izvršene su dvije naredbe u naredbenoj liniji pomoću kojih Django generira tablice na temelju definicije klasa. Naredba *makemigration* generira SQL naredbe koje služe za stvaranje tablica, dok naredba *migrate* izvršava te SQL naredbe. Na Slici 18 prikazano je izvršenje navedenih naredbi.


```
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> python manage.py makemigrations
Migrations for 'wishlists':
  wishlists\migrations\0002_alter_wishlist_owner.py
  - Alter field owner on wishlist
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> python manage.py migrate
Operations to perform:
  Apply all migrations: account, admin, auth, contenttypes, sessions, wishlists
Running migrations:
  Applying wishlists.0002_alter_wishlist_owner... OK
```

Slika 18. Izvršenje migracijskih naredbi

Posljednji korak potreban da bi stvorene tablice, odnosno modeli, uspješno bili povezani s Django projektom je registracija stvorenih modela u datoteci `admin.py` unutar odgovarajuće Django aplikacije. Dakle, model `Account` registriran je u datoteci `admin.py` `account` aplikacije, a modeli `Wishlist`, `Product` i `Currency` registrirani su unutar datoteke `admin.py` `wishlists` aplikacije (Slika 19).

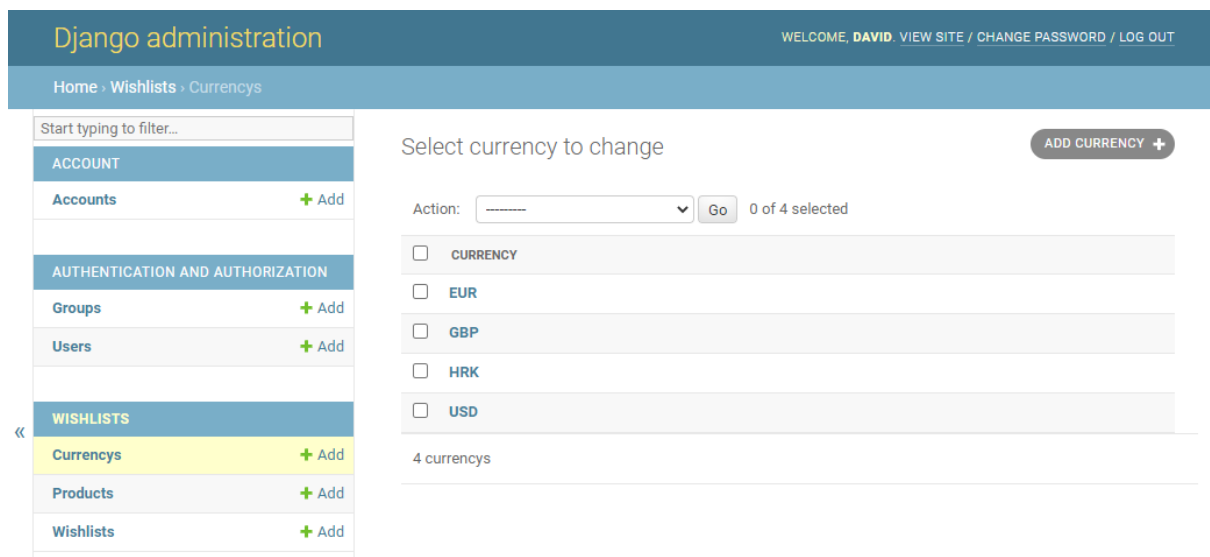


```
wishlists > admin.py
1 from django.contrib import admin
2 from .models import Currency, Product, Wishlist
3
4
5 admin.site.register(Currency)
6 admin.site.register(Product)
7 admin.site.register(Wishlist)
8

account > admin.py
1 from django.contrib import admin
2 from .models import Account
3
4 admin.site.register(Account)
5
```

Slika 19. Registracija modela

Nakon registracije modela, njima se može pristupiti unutar Django administracijskog sučelja te je, budući da je riječ od pred definiranim vrijednostima od strane administratora, u tablicu `Currency` uneseno nekoliko početnih vrijednosti. Navedeno je prikazano na Slici 20.

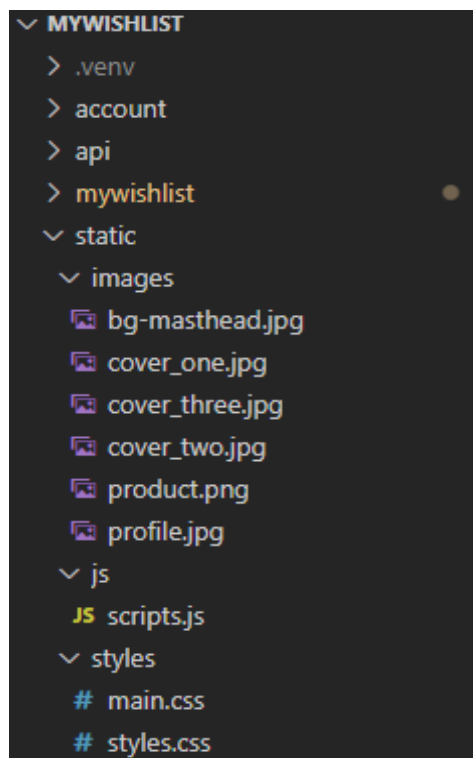


Slika 20. Administracijsko sučelje nakon izgradnje modela i unos početnih vrijednosti

4.3. Statične datoteke i instalacija početne teme

Nakon uspješnog uspostavljanja baze podataka, u slijedećem koraku definirano je mjesto za pohranu statičnih datoteka te je odabrana prvotna tema odnosno dizajn aplikacije koji je kasnije znatno prilagođen.

Za početak, u korijenskoj mapi Django projekta stvorena je nova mapa *static* te tri podmape *images*, *js* i *styles*. U mapi *images* spremljene su sve slike koje aplikacija koristi, što uključuje pozadinske slike, logo tipove te bilo koje druge slikovne datoteke koje služe dizajnu aplikacije. U mapi *js* te datoteci *scripts.js* pohranjene su sve JavaScript varijable i funkcije koje se koriste u aplikaciji, dok je u mapi *styles* te datotekama *main.css* i *styles.css* pohranjen cjelokupni CSS kod koji uključuje Bootstrap web okvir, kao i neke ručno definirane stilove. Na Slici 21 prikazan je navedeni raspored mapa.



Slika 21. Raspored mapa Django projekta

Da bi se Django povezoao s novostvorenim mapama i mogao čitati sadržaj datoteka iz istih, potrebno je urediti nekoliko postavki u *settings.py*, referencirati datoteke i učitati *static* u samom HTML dokumentu. Na Slici 22 prikazan je dio koda koji povezuje Django sa statičnim datotekama.

```

STATIC_URL = 'static/'
MEDIA_URL = 'images/'

STATICFILES_DIRS = [
    | BASE_DIR / 'static'
]

STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
MEDIA_ROOT = os.path.join(BASE_DIR, 'static/images')

```

Slika 22. Povezivanje statičnih datoteka

U zadnjem koraku potpunog povezivanja uređena je i *urls.py* datoteka u kojoj je dodan dio koda prikazan na Slici 23, a koji služi za generiranje URL putanje do statičnih datoteka.

```

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)

```

Slika 23. Generiranje URL putanje

Također, budući da statične datoteke funkcioniraju na nešto drukčiji način prilikom objavljivanja u produkciju, u ovoj fazi izrade aplikacije instaliran je i dodatak *whitenoise* koji upravlja statičnim datotekama (naredba prikazana na Slici 24).

```

(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> pip install whitenoise

```

Slika 24. Instalacija dodatka WhiteNoise

Nakon instalacije izmijenjena je i *settings.py* datoteka u odjeljku MIDDLEWARE, gdje je nadodan dio koda prikazan na Slici 25 kako bi se Django i povezo s instaliranim dodatkom.

```

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    'whitenoise.middleware.WhiteNoiseMiddleware',
]

```

Slika 25. Registracija dodatka WhiteNoise

Nakon stvaranja poveznice sa statičnim datotekama, u prethodno stvorenoj mapi *templates* stvorena je datoteka *main.html*, što je glavna datoteka Django aplikacije (odnosno sve druge HTML datoteke stvorene kasnije su proširenje iste). Na web sjedištu *Start Bootstrap* [14] odabran je predložak s već pred-definiranim dizajnom, odnosno prilagođenim, jednostranim

Bootstrap CSS kodom i rasporedom stranice koji se kasnije znatno prilagodio. Predložak se sastojao od jedne HTML i jedne CSS datoteke te je instaliran na način da je prvotno predložak preuzet na lokalno računalo, zatim CSS datoteka, što je zapravo cijeli Bootstrap web okvir, smještena u mapu *styles*, te HTML datoteka smještena u mapu *templates*.

Nakon instalacije Bootstrap predloška, u datoteku *main.html* kopirane su reference na pojedine stilske dokumente iz predloška, odnosno reference na sam Bootstrap kod. Također su i ručno definirane reference na nekoliko različitih Google fontova, Bootstrap i Favicon ikone te je definiran i naslov stranice. Ispod bloka sadržaja u dokumentu je referencirana *scripts.js* datoteka kao i nekoliko drugih JS alata. Budući da su sve ostale datoteke proširenje na datoteku *main.html*, dovoljno je samo na jednom mjestu uključiti reference. Na Slici 26 prikazana je konstrukcija glavne HTML stranice aplikacije.

```
<!DOCTYPE html>
{% load static %}

<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <title>MyWISHLIST - Organize Your Buying Schedule</title>
    <!-- Favicon -->
    <link rel="icon" type="image/x-icon" href="assets/favicon.ico" />
    <!-- Bootstrap Icons -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css" rel="stylesheet" />
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.8.1/font/bootstrap-icons.css">
    <!-- Google fonts -->
    <link href="https://fonts.googleapis.com/css?family=Merriweather+Sans:400,700" rel="stylesheet" />
    <link href="https://fonts.googleapis.com/css?family=Merriweather:400,300,300italic,400italic,700,700italic" rel="stylesheet" type="text/css" />
    <link href="https://fonts.googleapis.com/css2?family=Dancing+Script:wght@700&family=Permanent+Marker&family=Rubik+Bubbles&display=swap" rel="stylesheet">
    <!-- SimpleLightbox plugin CSS -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/SimpleLightbox/2.1.0/simplelightbox.min.css" rel="stylesheet" />
    <!-- Bootstrap -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EV5QTQ3/azpr61Ann3QDgpJLIm9NaoBv21ztcQTFsp3yD65VohhpuuCOMLAsJc" crossorigin="anonymous">
    <!-- Core theme CSS (includes Bootstrap) -->
    <link href="{% static 'styles/styles.css' %}" rel="stylesheet" />
  </head>
  <body>
    {% block content %}
    {% endblock %}
    <!-- Bootstrap core JS -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-Mrcrk6ZMFY1zclA8N1+HtUJVF8sA7MsXsP11yJ0M4YLEuNSFAP+JcXn/tWtIaxVXM" crossorigin="anonymous"></script>
    <!-- SimpleLightbox plugin JS -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/SimpleLightbox/2.1.0/simplelightbox.min.js"></script>
    <!-- jQuery -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.js" integrity="sha512-n/4gH3BAtM30qRbCn6eWmpxCLAHGadJpEbu4xZd47N62oQ+6q7oc3PXstrJYXcbNU1OHdQ17pAP+g5YU8g==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <!-- Core theme JS -->
    <script src="{% static 'js/scripts.js' %}"></script>
  </body>
```

Slika 26. Početni kod *main.html* datoteke

4.4. Izrada početne stranice

Kod izrade početne stranice najprije je stvorena *index.html* datoteka u mapi *templates* u korijenskoj mapi projekta. Stranica je temeljena na preuzetom predlošku [15], odnosno naslovnom dijelu predloška. Izmijenjena je pozadinska slika te boje gumba i teksta manipulirajući uvezeni Bootstrap CSS kod. Također, izmijenjen je i naslov kao i pripadajući tekstualni paragrafi kako bi odgovarali svrsi aplikacije.

U glavnoj *views.py* datoteci definirana je funkcija *home_view* koja vraća potrebne podatke, a to su podatci o korisničkom računu te podatci o pripadnim listama želja. Sve podatke koji se žele prikazati na samoj stranici potrebno je uključiti u varijablu *context*. Na Slici 27 prikazana je *home_view* funkcija.

```

def home_view(request):
    if request.user.is_authenticated:
        account = request.user.account
        wishlists = account.wishlist_set.all()
        context = {
            'wishlists': wishlists,
            'account': account,
        }
    else:
        context = {}
    return render(request, 'index.html', context)

```

Slika 27. Funkcija `home_view`

U slijedećem koraku ispisa podataka na strani klijenta potrebno je definirati i pripadajuću URL putanju koja koristi definiranu funkciju. Dakle, u datoteci `urls.py` dodan je dio koda prikazan na Slici 28.

```

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home_view, name="home"),
    path('home/', views.home_view, name="home"),
    path('', include('wishlists.urls')),
    path('', include('account.urls')),
]

```

Slika 28. Definicija URL putanje

Nakon što je URL definiran, u odgovarajućem HTML dokumentu dohvaćene su informacije vraćene iz funkcije, te je definirano nekoliko uvjeta pomoću kojih stranica postaje dinamična. Svakom korisniku ispisano je jedinstveno odnosno vlastito korisničko ime kao i različit paragraf teksta ovisno o tome ima li korisnik stvorenih lista želja ili je li korisnik prijavljen ili nije. Na Slici 29 prikazan je dio HTML koda naslovne stranice.

```

<!-- Cover Photo -->
<header class="masthead">
  <div class="container px-4 px-lg-5 h-100">
    <div class="row gx-4 gx-lg-5 h-100 align-items-center justify-content-center text-center">
      <div class="col-lg-8 align-self-end">
        <h1 class="text-white font-weight-bold">Your Favorite Place for Creating Wishlists</h1>
        <hr class="divider" />
      </div>
      <div class="col-lg-8 align-self-baseline">
        {% if request.user.is_authenticated %}
        <p class="text-white-75 mb-3">Hi {{account.username}},</p>
        <p class="text-white-75 mb-5">MyWISHLIST can help you create wishlists and better organize your buying schedule!
        {% if not wishlists %}
        Start wishing and some of Your wishlists will be displayed below!</p>
        {% else %}
        Some of the wishlists You already created are displayed below so You can pick up where You left off or create a new wishlist!</p>
        {% endif %}
        <a class="btn btn-primary btn-xl" href="{% url 'create-wishlist' %}">Create wishlist</a>
        {% else %}
        <p class="text-white-75 mb-5">MyWISHLIST can help you create wishlists and better organize your buying schedule! Just create an account, log in and start wishing! Oh, and it's FREE!</p>
        <a class="btn btn-primary btn-xl" href="{% url 'login' %}">Login</a>
        {% endif %}
      </div>
    </div>
  </div>
</header>

```

Slika 29. Dio koda naslovne stranice

Također, ako je korisnik prijavljen te već ima izrađenih lista želja, izgrađen je i prikazan karusel koji prikazuje 9 posljednje uređenih lista, a kojeg je moguće pronaći kao Bootstrap predložak [16]. Prema zadanim postavkama predložak prikazuje samo jednu sliku te je prilagođen da

prikazuje 3 kartice po slajdu i ukupno 3 slajda. Budući da je potrebno stvoriti responzivnu stranicu, HTML kod kartica za velike ekrane (Slika 30) različit je od HTML koda za male ekrane (Slika 31) što je lako ostvarivo jednostavnim korištenjem različitih klasa Bootstrap web okvira.

```

<!-- Cards Large-->
<div class="col-sm-4 d-none d-lg-block">
  <div class="card shadow">
    <div class="pt-4 text-center">
      <a href="{% url 'wishlist' wishlist.id %}"><i class="bi-list-columns fs-1"></i></a>
    </div>
    <div class="card-body">
      <h5 class="card-title text-center"><a href="{% url 'wishlist' wishlist.id %}">{{wishlist.title}}</a></h5>
      <p class="card-text text-muted">Products: {{wishlist.total_products}}</p>
      <p class="card-text text-muted">Total amount: {{wishlist.grand_total_price}}</p>
      <p class="card-text text-muted">Priority: {{wishlist.get_priority_display}}</p>
      <p class="card-text text-muted">Created: {{wishlist.created|date:"M d, Y"}}</p>
      <p class="card-text text-muted">Last modified: {{wishlist.modified|date:"M d, Y"}}</p>
      <div class="text-center">
        <a href="{% url 'wishlist' wishlist.id %}" class="btn btn-primary">View</a>
        <a href="{% url 'update-wishlist' wishlist.id %}" class="btn btn-primary">Edit</a>
        <a href="{% url 'delete-wishlist' wishlist.id %}" class="btn btn-primary">Delete</a>
      </div>
    </div>
  </div>
</div>

```

Slika 30. HTML kod kartice za velike ekrane

```

<!-- Cards Small-->
<div class="col-12 d-lg-none pb-2">
  <div class="card shadow">
    <div class="col-12 pt-2 px-2">
      <div class="row">
        <div class="col-7 text-start">
          <h6 class="card-title text-truncate mb-0"><a href="{% url 'wishlist' wishlist.id %}"><i class="bi-list-nested"></i> {{wishlist.title}}</a></h6>
        </div>
        <div class="col-5 text-end">
          <a class="px-1" href="{% url 'wishlist' wishlist.id %}"><i class="bi bi-search"></i></a>
          <a class="px-1" href="{% url 'update-wishlist' wishlist.id %}"><i class="bi bi-pen"></i></a>
          <a class="px-1" href="{% url 'delete-wishlist' wishlist.id %}"><i class="bi bi-trash"></i></a>
        </div>
      </div>
    </div>
    <div class="row">
      <div class="col-12">
        <div class="card-body py-1">
          <p class="card-text text-muted">Products: {{wishlist.total_products}}</p>
          <p class="card-text text-muted">Total price: {{wishlist.grand_total_price}}</p>
          <div class="row">
            <div class="col-7"><p class="card-text text-muted fs-7">Priority: {{wishlist.get_priority_display}}</p></div>
            <div class="col-5"><p class="card-text text-muted text-end fs-7">{{wishlist.created|date:"M d, Y"}}</p></div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

Slika 31. HTML kod kartice za male ekrane

Iznad karusela smještena je i poveznica koja vodi na pregled svih lista želja. Na Slici 32 prikazan je cjelokupni kod karusela.

```

{% if request.user.is_authenticated %}
{% if wishlists %}
<!-- Wishlist Title -->
<section class="page-section-custom" id="wishlists">
  <div class="container px-4 px-lg-5">
    <h2 class="text-center">My Wishlists</h2>
    <hr class="divider" />
    <h6 class="text-center pt-1"><a href="{% url 'wishlists' %}"><i class="bi bi-search"></i> VIEW ALL</a></h6>

    <!-- Carousel Wrapper -->

    <div id="carouselExampleCaptions" class="carousel slide" data-bs-ride="carousel">

      <div class="carousel-indicators">
        <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="0" class="active" aria-current="true" aria-label="Slide 1"></button>
        <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="1" aria-label="Slide 2"></button>
        <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="2" aria-label="Slide 3"></button>
      </div>

      <div class="carousel-inner">

        <!-- First Slide -->
        <div class="carousel-item active">

          <div class="row p-4">
            {% for wishlist in wishlists|slice:"":9 %}
            <!-- Cards Large -->
            <div class="col-sm-4 d-none d-lg-block"> ...
            </div>

            <!-- Cards Small -->
            <div class="col-12 d-lg-none pb-2"> ...
            </div>
            </div>
            {% if forloop.counter|divisibleby:3 and not forloop.last %}
            </div>
            <div class="carousel-item">
              <div class="row p-4">
                {% endif %}
                {% endfor %}
              </div>
            </div>

          </div>

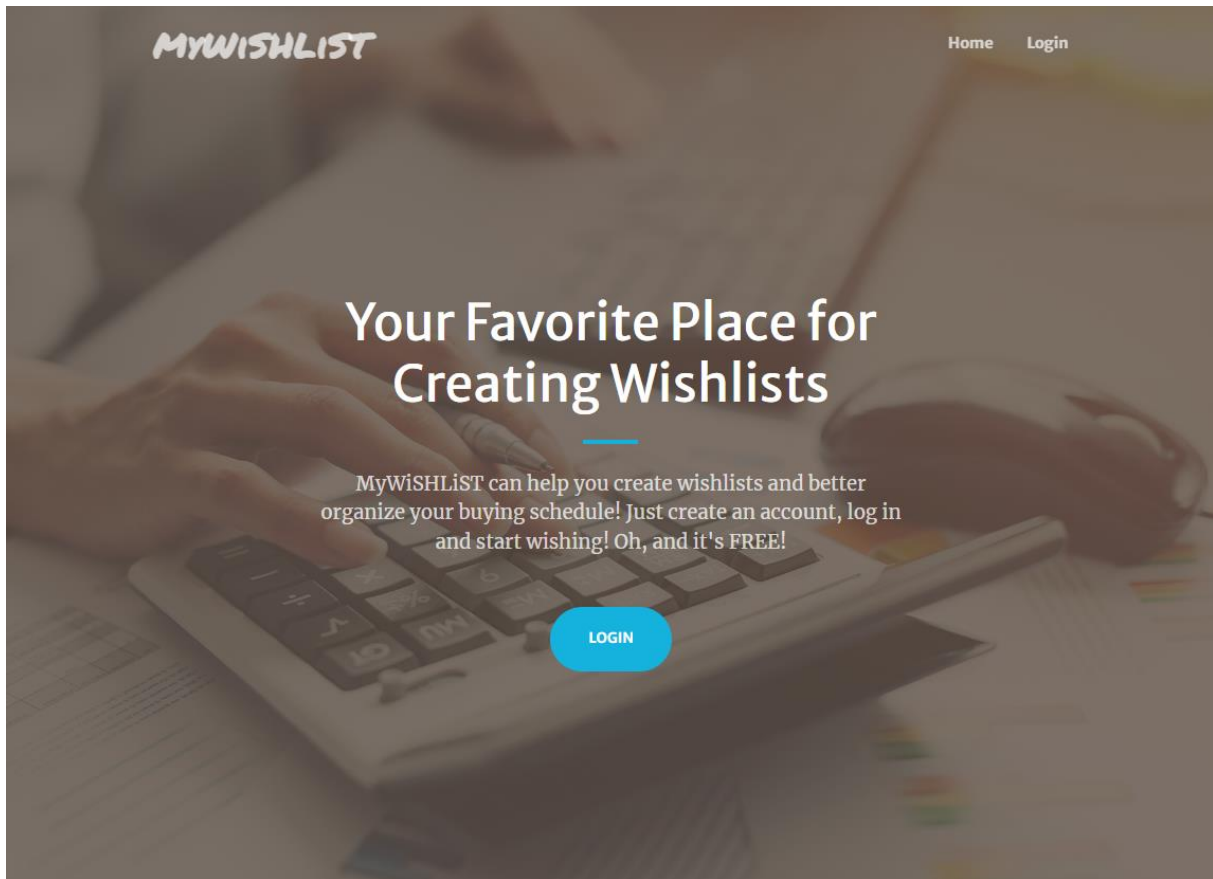
          <button class="carousel-control-prev" type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide="prev">
            <span class="carousel-control-prev-icon" aria-hidden="true"></span>
            <span class="visually-hidden">Previous</span>
          </button>
          <button class="carousel-control-next" type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide="next">
            <span class="carousel-control-next-icon" aria-hidden="true"></span>
            <span class="visually-hidden">Next</span>
          </button>

        </div>
      </div>
    </div>
  </div>
</section>
{% endif %}
{% endif %}

```

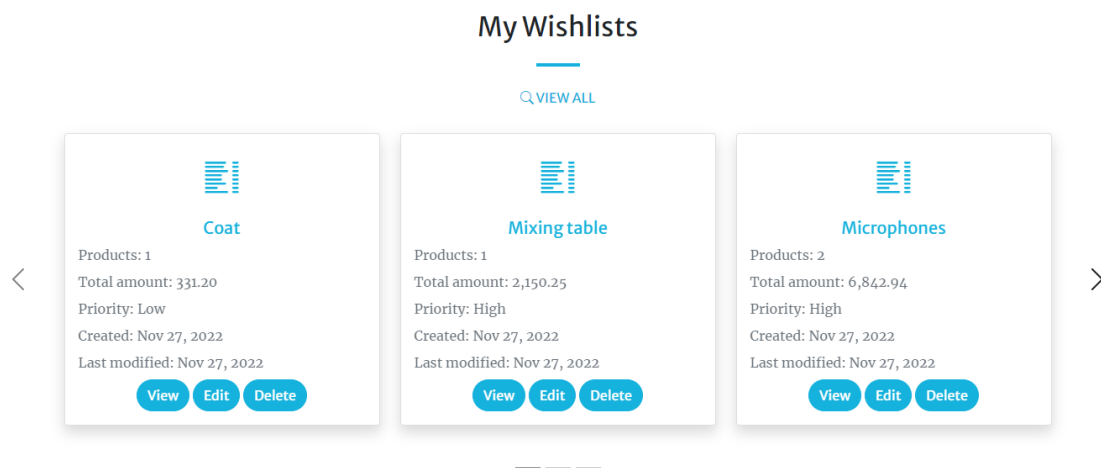
Slika 32. HTML kod karusela

Na Slici 33 prikazan je rezultat koda naslovne stranice u slučaju u kojem korisnik nije prijavljen.



Slika 33. Naslovna stranica - neprijavljen korisnik

Na Slici 34 prikazan je dio naslovne stranice koji se nalazi ispod naslovne slike u slučaju kada je korisnik prijavljen te posjeduje nekoliko lista želja.



Slika 34. Prikaz naslovnice prijavljenog korisnika s nekoliko lista želja

4.4.1. Navigacijska traka

Prilikom izrade navigacijske trake najprije je stvorena *navbar.html* datoteka u mapi *templates* u korijenske mape. Kao i ostale HTML datoteke, proširenje je na datoteku *main.html* te je

novostvorenu datoteku potrebno uključiti u istu. Na Slici 35 prikazan je dio koda kojim je proširena *main.html* datoteka.

```
<body>

    {% include 'navbar.html' %}

    {% block content %}

    {% endblock %}
```

Slika 35. Proširenje *main.html* datoteke

Navigacijska traka također je temeljena na preuzetom predlošku i prilagođena prema potrebi. Sastoji se od logotipa koji je ujedno i poveznica na početnu stranicu te poveznica *Home*, *My wishlists* koja vodi na pregled svih lista želja, *Account* koja vodi na opcije korisničkog računa te poveznica *Log In* ili *Logout* ovisno o tome je li korisnik prijavljen. Poveznice *My wishlists* i *Account* dostupne su samo ako je korisnik prijavljen.

Navigacijskoj traci dodan je efekt promjene boje same trake, a i pripadnih poveznica prilikom listanja prema dolje na svim stranicama koje koriste pozadinsku sliku protegnutu preko cijelog ekrana, pomoću JavaScript koda koji pretražuje trenutnu URL putanju. Navigacija je također responzivna te se automatski prilagođava velikim i malim ekranima. Kad je riječ o malim ekranima, navigacija postaje padajući izbornik. Na Slici 36 prikazan je HTML kod navigacije.

```
<!-- Navigation -->
<nav class="navbar navbar-expand-lg navbar-light fixed-top py-2" id="mainNav">

  <div class="container px-4 px-lg-5">
    <a class="navbar-brand" style="font-family: 'Permanent Marker', cursive; font-size: xx-large; letter-spacing: 1px;" href="{% url 'home' %}">MyWISHLIST</a>
    <button class="navbar-toggler navbar-toggler-right" type="button" data-bs-toggle="collapse" data-bs-target="#navbarResponsive" aria-expanded="false" aria-label="Toggle navigation"><span class="navbar-toggler-icon"></span></button>
    <div class="collapse navbar-collapse" id="navbarResponsive">
      <ul class="navbar-nav ms-auto my-2 my-lg-0">
        {% url 'wishlists' as url_wishlists %}
        {% url 'wishlist' wishlist.id as url_wishlist %}
        <li class="nav-item"><a class="nav-link" href="{% url 'home' %}">Home</a></li>

        {% if request.user.is_authenticated %}
        <li class="nav-item"><a class="nav-link" href="{% if request.path == url_wishlists or request.path == url_wishlist %} active {% endif %}" href="{% url 'wishlists' %}">My wishlists</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'account' %}">Account</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'logout' %}">Logout</a></li>
        {% else %}
        <li class="nav-item"><a class="nav-link" href="{% url 'login' %}">Login</a></li>
        {% endif %}
      </ul>
    </div>
  </div>
</nav>
```

Slika 36. HTML kod navigacije

Na Slici 37 prikazan je JavaScript kod kojim su postignuti željeni efekti kod navigacije.

```

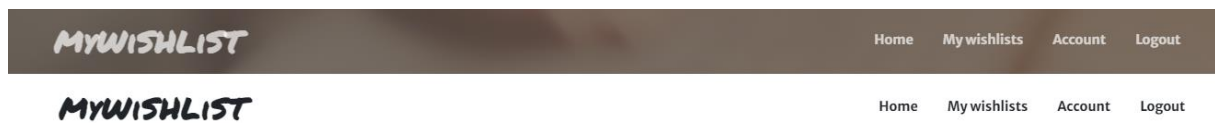
window.addEventListener('DOMContentLoaded', event => {
  // Navbar shrink function
  var pageURL = "https://mywishlistproject.herokuapp.com/home/";
  var pageURL2 = "https://mywishlistproject.herokuapp.com/";
  var pageURL3 = "https://mywishlistproject.herokuapp.com/account/";
  var pageURL4 = "https://mywishlistproject.herokuapp.com/register/";
  var pageURL5 = "https://mywishlistproject.herokuapp.com/login/";
  var pageURL6 = "https://mywishlistproject.herokuapp.com/reset_password/";
  var pageURL7 = "https://mywishlistproject.herokuapp.com/reset_password_sent/";
  var pageURL8 = "https://mywishlistproject.herokuapp.com/reset/MQ/set_password/";
  var pageURL9 = "https://mywishlistproject.herokuapp.com/reset_password_complete/";
  var pageURL10 = "https://mywishlistproject.herokuapp.com/change_password/";
  var pageURL11 = "https://mywishlistproject.herokuapp.com/change_password_done/";
  var navbarShrink = function () {
    const navbarCollapsible = document.querySelector('#mainNav');
    if (!navbarCollapsible) {
      return;
    }
    if (window.scrollY === 0 && [
      window.location.href == pageURL ||
      window.location.href == pageURL2 ||
      window.location.href == pageURL3 ||
      window.location.href == pageURL4 ||
      window.location.href == pageURL5 ||
      window.location.href == pageURL6 ||
      window.location.href == pageURL7 ||
      window.location.href == pageURL8 ||
      window.location.href == pageURL9 ||
      window.location.href == pageURL10 ||
      window.location.href == pageURL11
    ]) {
      navbarCollapsible.classList.remove('navbar-shrink')
    } else {
      navbarCollapsible.classList.add('navbar-shrink')
    }
  };
  console.log(window)
  // Shrink the navbar
  navbarShrink();

  // Shrink the navbar when page is scrolled
  document.addEventListener('scroll', navbarShrink);
}

```

Slika 37. JavaScript kod efekta navigacije

Na Slici 38 prikazan je rezultat koda, odnosno izgled navigacijske trake u početnom stanju, te u stanju listanja prema dolje.



Slika 38. Prikaz navigacijske trake

4.4.2. Podnožje stranice

U mapi *templates* korijenske mape stvorena je datoteka *footer.html* te je nakon bloka sadržaja uključena u *main.html* datoteku čime ju proširuje. Podnožje je izrađeno jednostavno te se sastoji od različite pozadinske boje i odgovarajuće poruke vlasništva stranice. Budući da je uključeno u *main.html* datoteku, podnožje je prisutno na svakoj stranici. Na Slici 39 prikazan je HTML kod podnožja.

```

<footer class="bg-light py-5">
  <div class="container px-4 px-lg-5">
    <div class="small text-center text-muted">Copyright &copy; 2022 - MyWISHLIST</div>
  </div>
</footer>

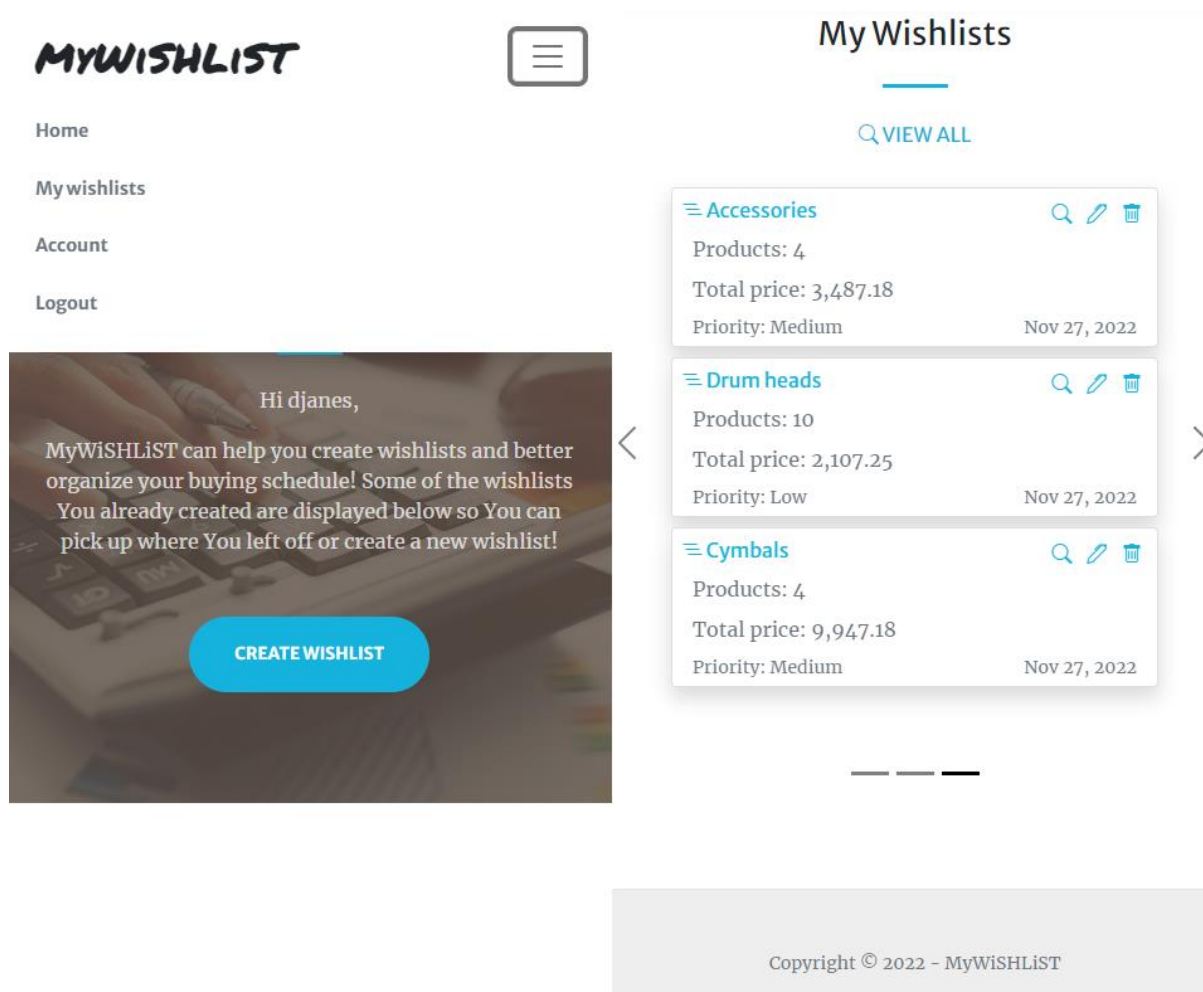
```

Slika 39. HTML kod podnožja

Na Slici 40 prikazan je rezultat, odnosno konačni izgled podnožja.

Slika 40. Prikaz podnožja stranice

Budući da je potrebno izgraditi responzivnu web stranicu, sve komponente prikladno se prilagođavaju veličini ekrana. Na Slici 41 prikazana je cjelokupna naslovna stranica s navigacijom i podnožjem na malim ekranima.



Slika 41. Prikaz naslovne stranice na malim ekranima

4.5. Korisnička autentifikacija i izgradnja pripadnih stranica

Nakon izgradnje početne stranice, navigacije i podnožja, izgrađen je i cijeli sustav registracije i prijave korisnika, kao i mogućnost izmjene korisničkih informacija.

4.5.1. Registracija korisnika

U mapi *templates* nadmape *account* stvorena je nova HTML datoteka *login_register.html* koja sadrži cjelokupni kod registracije i prijave korisnika. Na početku je također stvorena i nova pomoćna datoteka *forms.py* u kojoj se nalazi definicija obrasca za registraciju. Obrazac se sastoji od polja *ime*, *prezime*, *korisničko ime*, *e-mail*, *lozinka* i *potvrda lozinke*. Također, sav dizajn polja obrasca definiran je unutar definicije klase istog. Poljima korisničko ime, lozinka i

potvrda lozinke dodane su i pomoćne poruke pravila upisa podataka. Na Slici 42 prikazan je kod registracijskog obrasca.

```
class CustomUserCreationForm(UserCreationForm):
    class Meta:
        model = User
        fields = ['first_name', 'last_name', 'username', 'email', 'password1', 'password2']

    def __init__(self, *args, **kwargs):
        super(CustomUserCreationForm, self).__init__(*args, **kwargs)

        self.fields['first_name'].widget.attrs.update({
            'class': 'form-control',
            'id': 'first_name',
            'type': 'text',
            'placeholder': 'Enter first name...'
        })
        self.fields['last_name'].widget.attrs.update({
            'class': 'form-control',
            'id': 'last_name',
            'type': 'text',
            'placeholder': 'Enter last name...'
        })
        self.fields['username'].widget.attrs.update({
            'class': 'form-control',
            'id': 'username',
            'type': 'text',
            'placeholder': 'Enter username...'
        })
        self.fields['email'].label = 'E-mail'
        self.fields['email'].widget.attrs.update({
            'class': 'form-control',
            'id': 'email',
            'type': 'email',
            'placeholder': 'Enter E-mail...'
        })
        self.fields['password1'].help_text = """<div class="text-start mb-0">Your password can't be too similar to your other personal information.</div>
        Your password must contain at least 8 characters.</div>
        Your password can't be a commonly used password.</div>
        Your password can't be entirely numeric.</div>"""
        self.fields['password1'].widget.attrs.update({
            'class': 'form-control',
            'id': 'password1',
            'type': 'password',
            'placeholder': 'Enter password...'
        })
        self.fields['password2'].label = 'Confirm password'
        self.fields['password2'].widget.attrs.update({
            'class': 'form-control',
            'id': 'password2',
            'type': 'password',
            'placeholder': 'Confirm password...'
        })
    })
```

Slika 42. Definicija obrasca za registraciju

Nakon definiranja obrasca, u datoteci `views.py` u pripadnoj mapi definirana je i funkcija za registraciju koja koristi novostvorenu definiciju obrasca te je dodano i nekoliko poruka o uspjehu ili pogrešci obrade zahtjeva. Na Slici 43 prikazana je funkcija obrade registracijskog obrasca.

```
def registerUser(request):
    page = 'register'
    form = CustomUserCreationForm()

    if request.method == 'POST':
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            user = form.save(commit=False)
            user.username = user.username.lower()
            user.save()

            messages.success(request, 'Registered successfully!')

            login(request, user)
            return redirect('home')
        else:
            messages.error(request, 'An error has occurred during registration.')

    context = {
        'page': page,
        'form': form,
    }
    return render(request, 'account/login_register.html', context)
```

Slika 43. Funkcija registracijskog obrasca

Budući da je model *Account* izgrađen kao proširenje pred definiranog Django korisničkog modela, stvorena je i nova pomoćna datoteka *signals.py* u kojoj je definirana funkcija poveznice dvaju modela te služi za prijenos podataka iz jednog modela u drugi. Dakle, prilikom stvaranja korisničkog računa, stvara se i instanca Django korisničkog modela. Također, u toj funkciji uključeno je i proširenje pomoću kojeg Django automatski šalje pozdravnu E-mail poruku prilikom registracije korisničkog računa. Na Slici 44 prikazana je funkcija povezivanja dvaju modela.

```
def createAccount(sender, instance, created, **kwargs):
    if created:
        user = instance
        account = Account.objects.create(
            user = user,
            username = user.username,
            email = user.email,
            name = f"{user.first_name} {user.last_name}",
        )

        subject = 'Welcome to MyWISHLIST!'
        message = 'Thank You for creating an account!\nMyWISHLIST will help you create wishlists and better organize your buying schedule!\n\nGet out there and start wishing!\n\nMyWISHLIST Team'

        send_mail(
            subject,
            message,
            settings.EMAIL_HOST_USER,
            [account.email],
            fail_silently=False,
        )
```

Slika 44. Povezivanje dvaju korisničkih modela

Brisanje korisničkih računa u ovom sustavu moguće je isključivo putem administracijske ploče. Brisanjem instance Django korisničkog modela briše se i instanca ručno definiranog modela i obratno. Pripadni kod, također definiran u datoteci *signals.py*, prikazan je na Slici 45.

```
def deleteAccount(sender, instance, **kwargs):
    try:
        user = instance.user
        user.delete()
    except:
        pass

post_save.connect(createAccount, sender=User)
post_save.connect(updateUser, sender=Account)
post_delete.connect(deleteAccount, sender=Account)
```

Slika 45. Kod brisanja instance modela

U zadnjem koraku prije prilagodbe dizajna ovog dijela aplikacije u datoteci *urls.py* u pripadnoj mapi stvorena je URL putanja (Slika 46) koja koristi definiranu funkciju registracije.

```
urlpatterns = [
    path('register/', views.registerUser, name="register"),
```

Slika 46. URL putanja registracijskog obrasca

Dizajn registracijskog obrasca izrađen je po uzoru na naslovnu stranicu kako bi tematski odgovarao aplikaciji. Budući da je izgled polja definiran u samoj definiciji obrasca, potrebno je samo preostali dio izgleda obrasca iskoristiti iz preuzetog predloška. Stvorena je Bootstrap kartica te su na nju smještene polja ispod kojih se također nalazi i mjesto za poruku u slučaju greške pri obradi unesenih informacija. Ako korisnik već ima registrirani korisnički račun, na

dnu obrasca smještena je i poveznica koja vodi na stranicu prijave. Na Slici 47 prikazan je HTML kod registracijskog obrasca.

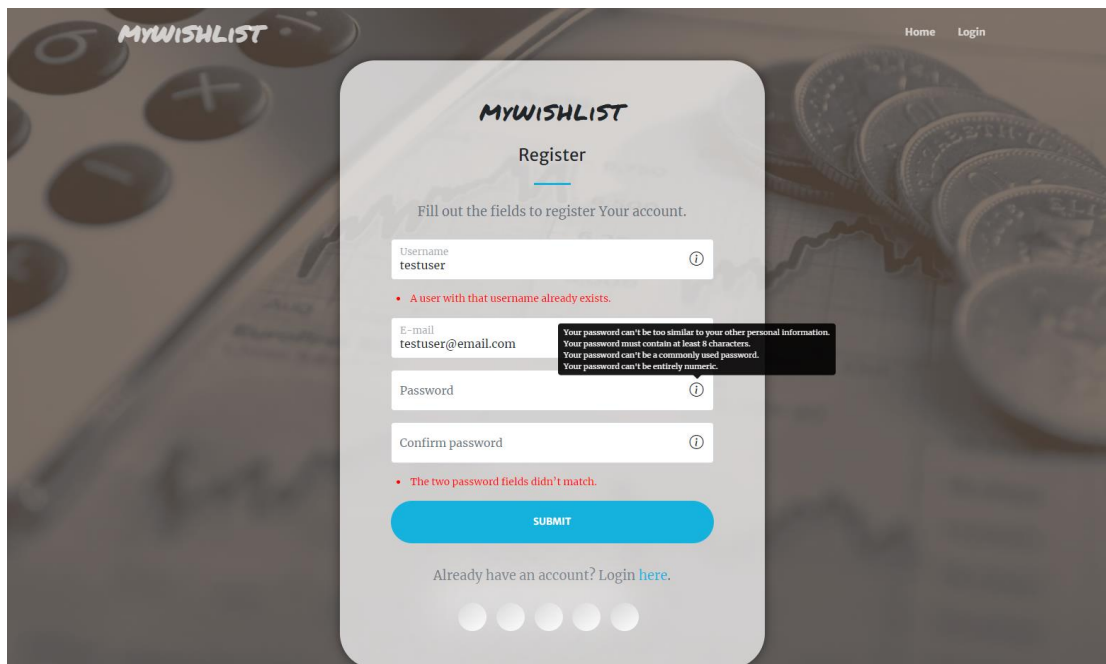
```
{% if page == "register" %}
<!-- REGISTER -->
<header class="masthead-register">
  <div class="container px-4 px-lg-5 h-auto">
    <div class="row gx-4 gx-lg-5 h-auto align-items-center justify-content-center text-center mt-1">

      <div class="card-register p-4 py-4">
        <div class="text-center py-4">
          <a class="text-dark" style="font-family: 'Permanent Marker', cursive; font-size: xx-large; letter-spacing: 1px;" href="{% url 'home' %}">MyWISHLIST</a>
          <h4 class="pt-4">Register</h4>
          <hr class="divider" />
          <p class="text-muted">Fill out the fields to register Your account.</p>
          <div class="px-4 px-lg-5 py-3">
            <form id="CustomUserCreationForm" action="{% url 'register' %}" method="POST">{% csrf_token %}
              <!-- Username input -->
              <div class="form-floating mb-3">
                {{form.username}}
                <span class="form-icon bi bi-info-circle fs-5" data-bs-toggle="tooltip" data-bs-placement="top" title="{{form.username.help_text}}"></span>
                <label for="username" class="text-muted">{{form.username.label}}</label>
              </div>
              <!-- Username errors -->
              {% for error in form.username.errors %}
                <li class="fs-7 mb-3 px-2 text-start" style="color: red;">{{error}}</li>
              {% endfor %}
              <!-- E-mail input -->
              <div class="form-floating mb-3">
                {{form.email}}
                <label for="email" class="text-muted">{{form.email.label}}</label>
              </div>
              <!-- Email errors -->
              {% for error in form.email.errors %}
                <li class="fs-7 mb-3 px-2 text-start" style="color: red;">{{error}}</li>
              {% endfor %}
              <!-- Password input -->
              <div class="form-floating mb-3">
                {{form.password1}}
                <span class="form-icon bi bi-info-circle fs-5" data-bs-toggle="tooltip" data-bs-placement="top" data-bs-html="true" title="{{form.password1.help_text}}"></span>
                <label for="password1" class="text-muted">{{form.password1.label}}</label>
              </div>
              <!-- Password errors -->
              {% for error in form.password1.errors %}
                <li class="fs-7 mb-3 px-2 text-start" style="color: red;">{{error}}</li>
              {% endfor %}
              <!-- Confirm password input -->
              <div class="form-floating mb-3">
                {{form.password2}}
                <span class="form-icon bi bi-info-circle fs-5" data-bs-toggle="tooltip" data-bs-placement="top" data-bs-html="true" title="{{form.password2.help_text}}"></span>
                <label for="password2" class="text-muted">{{form.password2.label}}</label>
              </div>
              <!-- Confirm password errors -->
              {% for error in form.password2.errors %}
                <li class="fs-7 mb-3 px-2 text-start" style="color: red;">{{error}}</li>
              {% endfor %}

              <!-- Submit Button -->
              <div class="d-grid pb-3"><button class="btn btn-primary btn-xl" id="submitButton" type="submit">Submit</button></div>
            </form>
          </div>
          <p class="text-muted">Already have an account? Login <a href="{% url 'login' %}">here</a>.</p>
          <div class="social-buttons mt-4">
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
          </div>
        </div>
      </div>
    </div>
  </div>
</header>
```

Slika 47. HTML kod registracijskog obrasca

Na Slici 48 prikazan je rezultat prethodnog koda, odnosno konačni izgled registracijskog obrasca.



Slika 48. Prikaz registracijskog obrasca

4.5.2. Prijava i odjava korisnika

Za izgradnju prijave korisnika nije potrebno definirati novi obrazac, već se koriste postojeće informacije u bazi. U datoteci `views.py` u pripadnoj mapi definirana je funkcija koja obrađuje zahtjev prijave te posebna funkcija koja obrađuje zahtjev odjave. Unutar funkcije također je uključeno i nekoliko skočnih poruka koje informiraju korisnika o uspjehu ili grešci pri obradi zahtjeva. Na Slici 49 prikazan je kod funkcije obrade zahtjeva prijave i odjave.

```
def loginUser(request):
    page = 'login'

    if request.user.is_authenticated:
        return redirect('home')

    if request.method == "POST":
        username = request.POST['username']
        password = request.POST['password']

        try:
            user = User.objects.get(username=username)
        except:
            messages.error(request, 'Username does not exist.')

        user = authenticate(request, username=username, password=password)

        if user is not None:
            login(request, user)
            messages.success(request, 'Logged in successfully!')
            return redirect('home')
        else:
            messages.error(request, 'Username or password is incorrect.')

    return render(request, 'account/login_register.html')

def logoutUser(request):
    logout(request)
    messages.info(request, 'Successfully logged out.')
    return redirect('home')
```

Slika 49. Funkcije prijave i odjave korisnika

Nadalje, u datoteci `urls.py` u pripadnoj mapi definirane su URL putanje koje koriste definirane funkcije prijave i odjave (Slika 50).

```
urlpatterns = [
    path('login/', views.loginUser, name="login"),
    path('logout/', views.logoutUser, name="logout"),
    path('register/', views.registerUser, name="register"),
```

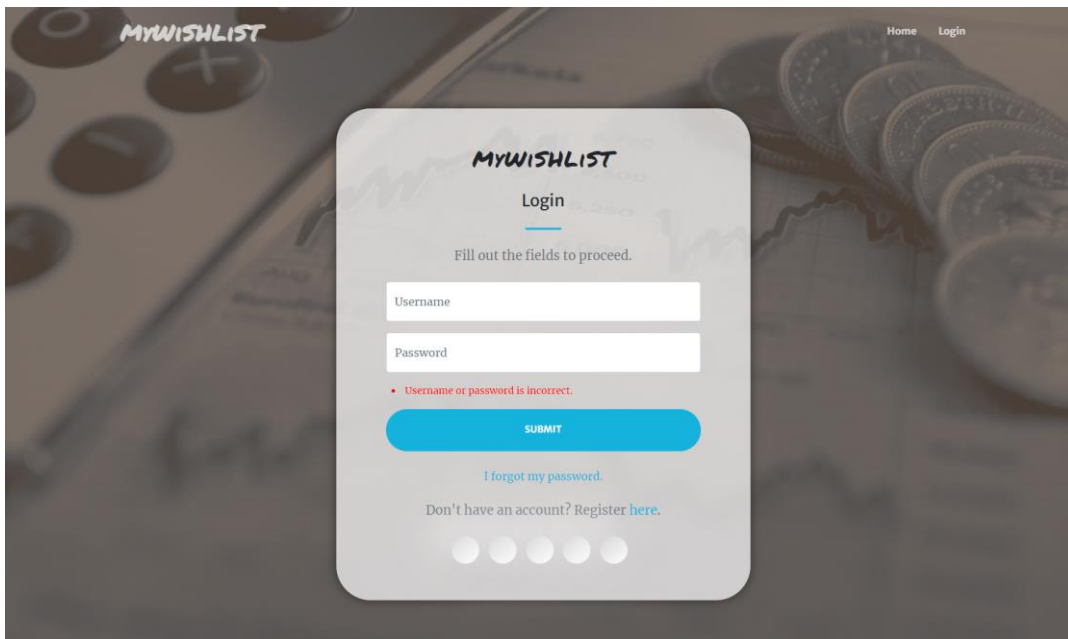
Slika 50. URL putanje prijave i odjave korisnika

Dizajn obrasca prijave također je izrađen po uzoru na naslovnu stranicu kako bi tematski odgovarao aplikaciji. Budući da stranica prijave ne koristi već definirani obrazac, u ovom koraku oblikovana su i polja obrasca. Izrađena je Bootstrap kartica te se obrazac sastoji od polja korisničko ime i lozinka. Ispod polja za unos lozinke nalazi se mjesto za ispis poruka ako dođe do greške prilikom obrade zahtjeva. Na dnu obrasca smještena je poveznica na registracijski obrazac ako korisnik ne posjeduje korisnički račun te poveznica za pokretanje postupka ponovnog postavljanja lozinke ako je korisnik zaboravio lozinku. Što se tiče odjave korisnika, poveznica za odjavu smještena je i prikazana na navigacijskoj traci kada je korisnik prijavljen. Na Slici 51 prikazan je HTML kod obrasca prijave.

```
{% else %}
<!-- LOGIN -->
<header class="masthead-login">
  <div class="container px-4 px-lg-5 h-100">
    <div class="row gx-4 gx-lg-5 h-100 align-items-center justify-content-center text-center">
      <div class="card-register p-4 py-4">
        <div class="text-center py-4">
          <a class="text-dark" style="font-family: 'Permanent Marker', cursive; font-size: xx-large; letter-spacing: 1px;" href="{% url 'home' %}">MYWISHLIST</a>
          <h4 class="pt-4">Login</h4>
          <hr class="divider" />
          <p class="text-muted">Fill out the fields to proceed.</p>
          <div class="px-4 px-lg-5 py-3">
            <form id="CustomUserCreationForm" action="{% url 'login' %}" method="POST"{% csrf_token %}>
              <!-- Username input -->
              <div class="form-floating mb-3">
                <input class="form-control" type="text" id="username" name="username" placeholder="Enter username..." required>
                <label for="username" class="text-muted">Username</label>
              </div>
              <!-- Password input -->
              <div class="form-floating mb-3">
                <input class="form-control" type="password" id="password" name="password" placeholder="Enter password..." required>
                <label for="password" class="text-muted">Password</label>
              </div>
              {% if messages %}
                {% for message in messages %}
                  <li class="fs-7 mb-3 px-2 text-start" style="color: red;">{{ message}}</li>
                {% endfor %}
              {% endif %}
              <!-- Submit Button -->
              <div class="d-grid pb-3"><button class="btn btn-primary btn-xl" id="submitButton" type="submit">Submit</button></div>
            </form>
            <!-- Forgot password -->
            <div class="form-floating py-2">
              <a href="{% url 'reset_password' %}">I forgot my password.</a>
            </div>
          </div>
          <p class="text-muted">Don't have an account? Register <a href="{% url 'register' %}">here</a>.</p>
          <div class="social-buttons mt-4">
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
          </div>
        </div>
      </div>
    </div>
  </div>
</header>
{% endif %}
```

Slika 51. HTML kod obrasca prijave

Na Slici 52 prikazan je rezultat prethodnog koda, odnosno konačni izgled obrasca za prijavu korisnika.



Slika 52. Prikaz obrasca za prijavu korisnika

4.5.3. Izrada stranice korisničkog računa

U mapi *templates* u pripadnoj nadmapi stvorena je datoteka *account.html* koja sadrži prikaz podataka o prijavljenom korisniku te obrazac za izmjenu podataka. U datoteci *forms.py* u pripadnoj mapi stvorena je nova klasa *AccountForm* koja sadrži definiciju obrasca za ažuriranje podataka, a definirana je slično kao i obrazac za registraciju, odnosno isključena su polja prema potrebi. Na Slici 53 prikazana je definicija obrasca za ažuriranje korisničkih informacija.

```
class AccountForm(ModelForm):
    class Meta:
        model = Account
        fields = ['name', 'username', 'email']

    def __init__(self, *args, **kwargs):
        super(AccountForm, self).__init__(*args, **kwargs)
        self.fields['name'].label = 'Full Name'
        self.fields['name'].widget.attrs.update({
            'class': 'form-control',
            'id': 'name',
            'type': 'text',
            'placeholder': 'Enter Your full name...'
        })
        self.fields['username'].widget.attrs.update({
            'class': 'form-control',
            'id': 'username',
            'type': 'text',
            'placeholder': 'Enter username...'
        })
        self.fields['email'].label = 'E-mail'
        self.fields['email'].widget.attrs.update({
            'class': 'form-control',
            'id': 'email',
            'type': 'email',
            'placeholder': 'Enter E-mail...'
        })
    })
```

Slika 53. Definicija obrasca za ažuriranje korisničkih informacija

U datoteci *views.py* u pripadnoj mapi definirana je funkcija koja vraća prikaz informacija korisničkog računa, ali sadrži i prethodno definirani obrazac putem kojeg se one mogu i ažurirati, budući da je stranica zamišljena na način da se i prikaz i ažuriranje odrađuje na istoj stranici. Na Slici 54 prikazana je definicija funkcije obrade prikaza i ažuriranja korisničkog računa.

```
@login_required(login_url="login")
def account_view(request):
    account = request.user.account
    form = AccountForm(instance=account)

    if request.method == 'POST':
        form = AccountForm(request.POST, instance=account)
        if form.is_valid():
            user = form.save(commit=False)
            user.username = user.username.lower()
            user.save()
            messages.success(request, 'Account updated successfully!')
            return redirect('account')
        else:
            messages.error(request, 'An error has occurred.')

    context = {
        'account': account,
        'form': form
    }
    return render(request, 'account/account.html', context)
```

Slika 54. Funkcija prikaza i ažuriranja informacija korisničkog računa

Kao i kod registracije, budući da je model *Account* proširenje zadanog Django korisničkog modela, u datoteci *signals.py* definirana je funkcija povezivanja tih dvaju modela. U modelu *Account* atribut *name* zamišljen je da sadrži puno ime i prezime korisnika, dok se u zadanom Django korisničkom modelu nalaze različita polja za ime i prezime. Koristeći integrirane funkcije razdvajanja i konkatencije spriječene su moguće pogreške u obradama zahtjeva registracije i ažuriranja informacija. Na Slici 55 prikazana je funkcija povezivanja dvaju modela.

```
def updateUser(sender, instance, created, **kwargs):
    account = instance
    user = account.user
    if created == False:
        try:
            user.first_name = account.name.split(' ').pop(0)
            user.last_name = account.name.split(' ').pop(-1)
        except:
            user.first_name = ''
            user.last_name = ''
    user.username = account.username
    user.email = account.email
    user.save()
```

Slika 55. Funkcija povezivanja zadanog i ručno definiranog modela korisničkog računa

U datoteci *urls.py* definirana je URL putanja koja koristi funkciju prikaza i ažuriranja korisničkih informacija, što je prikazano na Slici 56.

```
urlpatterns = [  
    path('login/', views.loginUser, name="login"),  
    path('logout/', views.logoutUser, name="logout"),  
    path('register/', views.registerUser, name="register"),  
    path('account/', views.account view, name="account"),  
]
```

Slika 56. URL putanja stranice korisničkog računa

Dizajn stranice korisničkog računa izgrađen je na sličan način kao i stranice registracije i prijave. Međutim, sastoji se od dvije različite Bootstrap kartice. Na jednoj kartici ispisane su informacije korisničkog računa te poveznica za poništavanje lozinke, a na drugoj je smješten obrazac za ažuriranje istih s popratnim prostorom za ispis grešaka prilikom slanja zahtjeva. Ako dođe do greške, kartica obrasca ostaje aktivna. Na kartici obrasca, budući da je riječ o već postojećim podacima, u poljima istog oni su i prikazani svojstvom *initial*, a pomicanje između te dvije kartice moguće je putem gumba *Edit*. Na Slici 57 prikazan je cjelokupni HTML kod stranice korisničkog računa.

```

<!-- Cover Photo -->
<header class="masthead-custom">
  <div class="container px-4 px-lg-5 h-100">
    <div class="row gx-4 gx-lg-5 h-100 align-items-center justify-content-center text-center">
      <div class="card-account p-4 py-4">
        <div class="text-center">
          <div>
            
          </div>
          <!-- DISPLAY INFO -->
          <div id="displayDIV" class="px-4 px-lg-5 py-2" style="display: block;">
            <button class="btn btn-dark fs-8 mt-3" type="button" onclick="editDIV()">Edit</button>
            {% if not account.user.first_name and not account.user.last_name %}
            <h3 class="mt-3">Full Name</h3>
            {% else %}
            <h3 class="mt-3">{{account.name}}</h3>
            {% endif %}
            <hr class="divider mt-3" />
            <div class="row">
              <div class="col-3 text-start ms-5">
                <p class="fs-6 fw-bold">Username:</p>
              </div>
              <div class="col-7 text-start me-2">
                <p class="fs-6 fst-italic">{{account.username}}</p>
              </div>
              <div class="col-3 text-start ms-5">
                <p class="fs-6 fw-bold">E-mail:</p>
              </div>
              <div class="col-7 text-start me-2">
                <p class="fs-6 fst-italic">{{account.email}}</p>
              </div>
              <div class="col-3 text-start ms-5">
                <p class="fs-6 fw-bold">Password:</p>
              </div>
              <div class="col-7 text-start me-2">
                <a href="{% url 'password_change' %}"><p class="fs-6 fst-italic">Change password</p></a>
              </div>
            </div>
          </div>
          <!-- EDIT INFO -->
          <div id="editDIV" class="px-4 px-lg-5 py-2" style="display: none;">
            <form id="AccountForm" method="POST" action="{% url 'account' %}">{{ csrf_token %}}
            <button class="btn btn-dark fs-8 mt-3" type="submit">Confirm</button>
            <a href="{% url 'account' %}"><button class="btn btn-danger fs-8 mt-3" type="button">Cancel</button></a>
            <p class="text-muted mt-3">Edit Your account information</p>
            <hr class="divider mt-3" />
            <!-- Name input -->
            <div class="form-floating mb-3">
              {{form.name}}
              <label for="name" class="text-muted">{{form.name.label}}</label>
            </div>
            <!-- Username input -->
            <div class="form-floating mb-3">
              {{form.username}}
              <label for="username" class="text-muted">{{form.username.label}}</label>
            </div>
            <!-- Username errors -->
            {% for error in form.username.errors %}
            <li id="error" class="fs-7 mb-3 px-2 text-start" style="color: red;">{{error}}</li>
            {% endfor %}
            <!-- E-mail input -->
            <div class="form-floating">
              {{form.email}}
              <label for="email" class="text-muted">{{form.email.label}}</label>
            </div>
            <!-- Email errors -->
            {% for error in form.email.errors %}
            <li id="error" class="fs-7 mt-3 mb-2 px-2 text-start" style="color: red;">{{error}}</li>
            {% endfor %}
          </form>
        </div>
        <div class="social-buttons mt-4">
          <button class="neo-button">
            </button>
          <button class="neo-button">
            </button>
          <button class="neo-button">
            </button>
          <button class="neo-button">
            </button>
          <button class="neo-button">
            </button>
          <button class="neo-button">
            </button>
          <button class="neo-button">
            </button>
          <button class="neo-button">
            </button>
        </div>
      </div>
    </div>
  </div>
</header>
{% endblock content %}

```

Slika 57. HTML kod stranice korisničkog računa

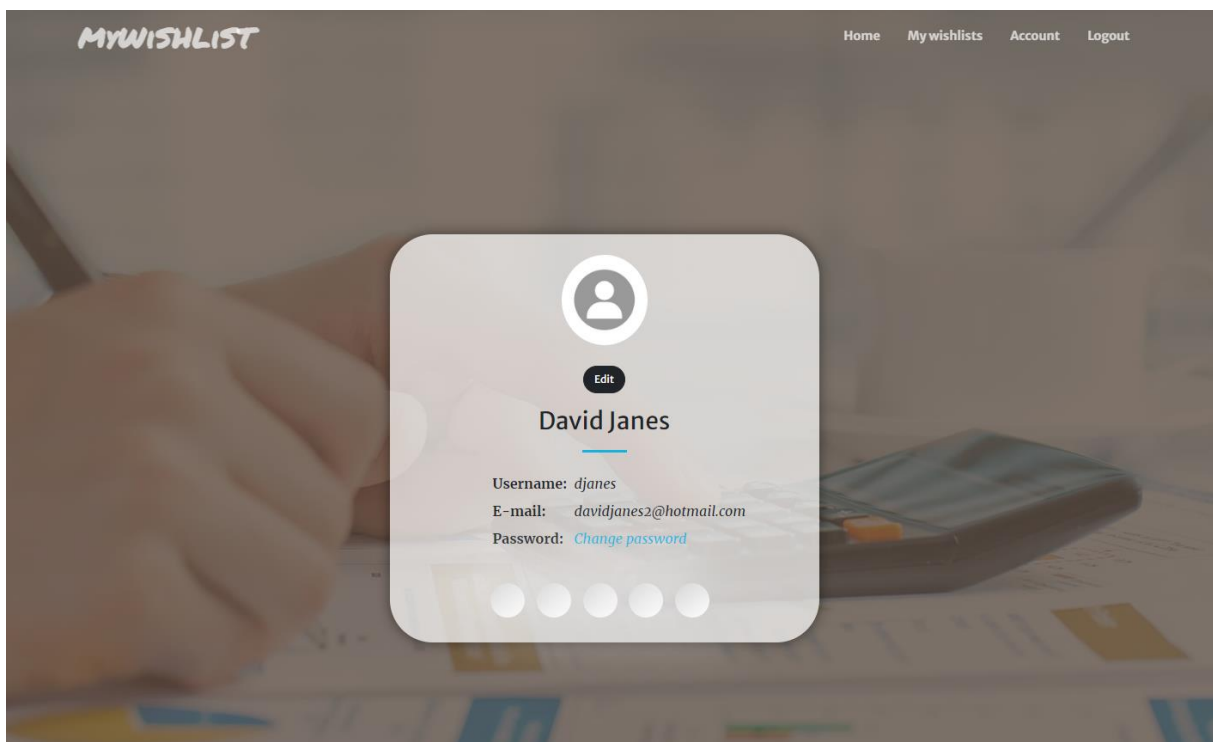
Spomenuto pomicanje među dvama karticama te zadržavanje kartice obrasca kada dođe do pogreške prilikom slanja obrasca ostvareno je pomoću JavaScript koda prikazanog na Slici 58: odgovarajućom definicijom *onClick* svojstva unutar HTML oznake gumba te definiranjem odgovarajuće ID oznake u odjeljcima ispisa pogrešaka.

```
//Edit & preview account switch
function editDIV() {
  document.getElementById("displayDIV").style.display="none";
  document.getElementById("editDIV").style.display="block";
}
//if errors exist keep edit div display
let error = document.getElementById("error");

if (error){
  editDIV();
}
```

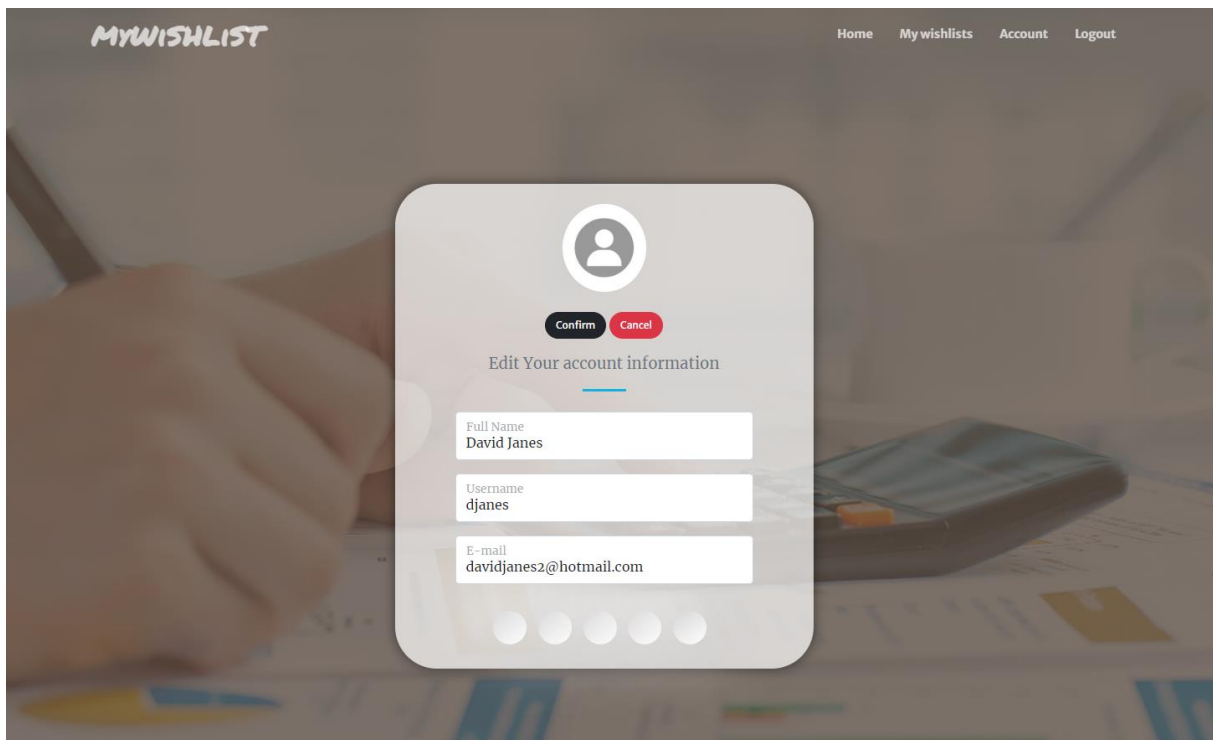
Slika 58. JavaScript kod za kretnju među karticama

Na Slici 59 prikazan je rezultat prethodnog koda, odnosno konačni rezultat kartice ispisa podataka korisničkog računa.



Slika 59. Prikaz stranice korisničkog računa - ispis podataka

Na Slici 60 prikazan je rezultat prethodnog koda, odnosno konačni rezultat kartice obrasca za ažuriranje podataka korisničkog računa.



Slika 60. Prikaz stranice korisničkog računa - obrazac za ažuriranje informacija

4.5.4. Skočne poruke

Da bi korisnici bili sigurni je li se pojedini zahtjev obradio uspješno ili nije, u lijevi donji kut aplikacije smještene su skočne poruke. U funkcijama obrade zahtjeva već prije su dodane poruke uspjeha, neuspjeha ili napomene te je taj dio koda iskorišten za dinamički prikaz poruke u različitim bojama, ovisno o kakvoj poruci je riječ. Iz Bootstrap dokumentacije [16] preuzete su 3 različite varijante poruke. Međutim, u određenom trenutku potrebna je samo jedna kojoj se dinamički mijenjaju svojstva pomoću Django koda. HTML kod poruke smješten je u *main.html* datoteku ispod bloka sadržaja te iznad podnožja kako bi poruke bile prisutne na svim stranicama aplikacije. Na Slici 61 prikazan je HTML kod skočnih poruka.

```
{% endblock %}

{% if messages %}
{% for message in messages %}
<!-- POPUP Alert -->
<div class="alert alert-{% if message.tags == 'info' %}primary{% elif message.tags == 'error' %}danger{% elif message.tags == 'success' %}success{% endif %} alert-dismissible alert-position fade show" role="alert">
  <div class="d-flex flex-row">
    <svg class="bi flex-shrink-0 me-2" width="24" height="24" role="img" aria-label="Info:"><use xlink:href="#info-fill"/></svg>
    <p class="mb-0 pb-0">{{message}}</p>
  </div>
  <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
</div>
{% endfor %}
{% endif %}

{% include 'footer.html' %}
```

Slika 61. HTML kod skočnih poruka

Skočne poruke moguće je ručno sakriti klikom na gumb X, ali im je također dodan i efekt automatskog nestajanja nakon određenog vremena, što je postignuto JavaScript kodom prikazanim na Slici 62.

```

// Alerts fade out
setTimeout(function () {

    // Closing the alert
    $(".alert").delay(5000).slideUp(800, function() {
        $(this).alert('close');
    });
});

```

Slika 62. JavaScript kod za automatsko nestajanje skočnih poruka

Na Slici 63 prikazan je rezultat prethodnog koda, odnosno konačni rezultat izrade skočnih poruka.



Slika 63. Prikaz skočne poruke

4.5.5. Ponovno postavljanje lozinke

Ponovno postavljanje lozinke ostvareno je rutinskim postupkom zadanim unutar Django web okvira. Ostvareno je na dva načina, a prema specifikaciji aplikacije. Jedan način je ponovno postavljanje lozinke prilikom prijave (ako je korisnik zaboravio lozinku) koristeći e-mail adresu računa, a drugi je pritiskom na poveznicu za izmjenu lozinke unutar stranice postavki korisničkog računa (koristeći već postojeću lozinku).

4.5.5.1. Ponovno postavljanje lozinke pomoću e-mail adrese

Prije svega, na uslugama Google-a stvorena je nova Gmail adresa te je generirana lozinka pomoću koje tu adresu mogu koristiti aplikacije treće strane. Zatim je u *settings.py* datoteci dodano nekoliko varijabli pomoću kojih se Django može i spojiti na novostvorenu e-mail adresu. Podatci prijave spremljeni su u posebnu datoteku u svrhu pojačanja sigurnosti te učitani naredbom *os.getenv*. Na Slici 64 prikazane su e-mail postavke aplikacije.

```

# E-mail setup
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = os.getenv('EMAIL_HOST_USER')
EMAIL_HOST_PASSWORD = os.getenv('EMAIL_HOST_PASSWORD')

```

Slika 64. E-mail postavke

Nakon povezivanja Djanga s e-mail adresom, stvoreno je nekoliko novih HTML datoteka u mapi *templates* korijenske mape, a to su *reset.html*, *reset_password.html*, *reset_password_sent.html* i *reset_password_complete.html*. Korištene su zadane Django funkcije za ponovno postavljanje lozinke te su proširene i prilagođene prema potrebi gdje je

primarno riječ o dizajnu. U datoteci *forms.py* u pripadnoj mapi definirana je nova Python klasa koja nasljeđuje zadanu te je prilagođen dizajn. Sadrži samo jedno polje, odnosno polje za unos e-mail adrese, te se nakon unosa pokreće postupak ponovnog postavljanja lozinke. Na Slici 65 prikazana je definicija klase obrasca za unos e-mail adrese.

```
class CustomPasswordResetForm(PasswordResetForm):  
    def __init__(self, *args, **kwargs):  
        super(CustomPasswordResetForm, self).__init__(*args, **kwargs)  
  
        self.fields['email'].label = 'E-mail'  
        self.fields['email'].widget.attrs.update({  
            'class': 'form-control',  
            'id': 'email',  
            'type': 'email',  
            'placeholder': 'Enter E-mail...',  
        })
```

Slika 65. Definicija obrasca za unos e-mail adrese

Na isti način stvorena je i prilagođena klasa odnosno obrazac za unos nove lozinke koji se prikazuje nakon pritiska na poveznicu dobivenu nakon što Django pošalje upute putem e-mail adrese, te su također definirane i pomoćne poruke. Na Slici 66 prikazana je definicija klase obrasca za unos nove lozinke.

```
class CustomSetPasswordForm(SetPasswordForm):  
    def __init__(self, *args, **kwargs):  
        super(CustomSetPasswordForm, self).__init__(*args, **kwargs)  
  
        self.fields['new_password1'].help_text = """<p class="text-start mb-0">Your password can't be too similar to your other personal information.</p><br>  
Your password must contain at least 8 characters.</br>  
Your password can't be a commonly used password.</br>  
Your password can't be entirely numeric.</p>"""  
        self.fields['new_password1'].label = 'New password'  
        self.fields['new_password1'].widget.attrs.update({  
            'class': 'form-control',  
            'id': 'new_password1',  
            'type': 'password',  
            'placeholder': 'Enter new password...'  
        })  
        self.fields['new_password2'].help_text = """<p class="text-start mb-0">Enter the same password as before, for verification.</p>"""  
        self.fields['new_password2'].label = 'New password confirmation'  
        self.fields['new_password2'].widget.attrs.update({  
            'class': 'form-control',  
            'id': 'new_password2',  
            'type': 'password',  
            'placeholder': 'Confirm new password...'  
        })
```

Slika 66. Definicija obrasca za unos nove lozinke

U slijedećem koraku, zadane Django funkcije ponovnog postavljanja lozinke proširene su odnosno izmijenjene unutar datoteke *views.py* u pripadnoj mapi kodom prikazanim na Slici 67.

```
class CustomPasswordResetView(PasswordResetView):  
    form_class = CustomPasswordResetForm  
  
class CustomPasswordResetConfirmView(PasswordResetConfirmView):  
    form_class = CustomSetPasswordForm
```

Slika 67. Proširenje zadanih funkcija ponovnog postavljanja lozinke

Koristeći Django dokumentaciju [17] te novostvorene proširene klase stvorene su i URL putanje unutar `urls.py` datoteke u pripadnoj mapi koje referenciraju na prethodno stvorene HTML datoteke. Navedeno je prikazano na Slici 68.

```
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home_view, name="home"),
    path('home/', views.home_view, name="home"),
    path('', include('wishlists.urls')),
    path('', include('account.urls')),
    path('api/', include('api.urls')),

    path('reset_password/', views.CustomPasswordResetView.as_view(template_name="reset_password.html"), name="reset_password"),
    path('reset_password_sent/', auth_views.PasswordResetDoneView.as_view(template_name="reset_password_sent.html"), name="password_reset_done"),
    path('reset/uidb64/<token>', views.CustomPasswordResetConfirmView.as_view(template_name="reset.html"), name="password_reset_confirm"),
    path('reset_password_complete/', auth_views.PasswordResetCompleteView.as_view(template_name="reset_password_complete.html"), name="password_reset_complete"),
]
```

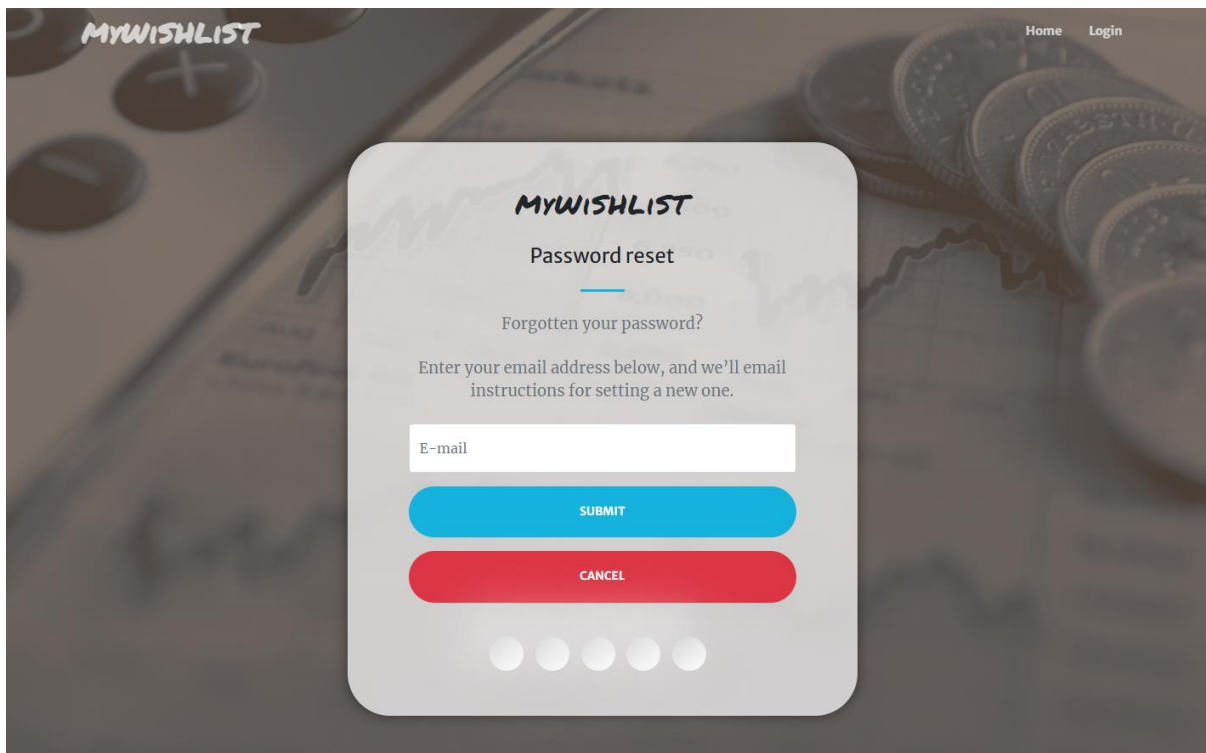
Slika 68. Definicija URL putanja do odgovarajućih HTML stranica

U zadnjem koraku prilagođen je dizajn pripadnih HTML stranica po uzoru na obrazac registracije i prijave u svrhu održavanja teme aplikacije. Na Slici 69 prikazan je HTML kod stranice `reset_password.html`.

```
<!-- RESET -->
<header class="masthead-login">
  <div class="container px-4 px-lg-5 h-100">
    <div class="row gx-4 gx-lg-5 h-100 align-items-center justify-content-center text-center">
      <div class="card-register p-4 py-4">
        <div class="text-center py-4">
          <a class="text-dark" style="font-family: 'Permanent Marker', cursive; font-size: xx-large; letter-spacing: 1px; href="{% url 'home' %}">MyWISHLIST</a>
          <h4 class="pt-4">Password reset</h4>
          <hr class="divider" />
          <p class="text-muted px-4 px-lg-5 pb-3">Forgotten your password?</p>
          <p class="text-muted px-4 px-lg-5">Enter your email address below, and we'll email instructions for setting a new one.</p>
          <div class="px-4 px-lg-5 py-3">
            <form method="POST"{% csrf_token %}>
              <!-- E-mail input -->
              <div class="form-floating mb-3">
                <input type="text" class="form-control" id="email" value="">
                <label for="email" class="text-muted">{{form.email.label}}</label>
              </div>
              <!-- Submit Button -->
              <div class="d-grid pb-3"><button class="btn btn-primary btn-xl" id="submitButton" type="submit">Submit</button></div>
              <div class="d-grid"><button class="btn btn-danger btn-xl" id="cancelButton" onClick="history.go(-1); return false;">Cancel</button></div>
            </form>
          </div>
          <div class="social-buttons mt-4">
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
          </div>
        </div>
      </div>
    </div>
  </div>
</header>
```

Slika 69. HTML kod obrasca za unos e-mail adrese

Na Slici 70 prikazan je konačni rezultat prethodnog koda, odnosno prikaz obrasca za unos e-mail adrese.



Slika 70. Prikaz obrasca za unos E-mail adrese

Na Slici 71 prikazan je HTML kod stranice *password_reset_sent.html* na kojoj se nalazi obavijest o slanju uputa za ponovno postavljanje lozinke na unesenu e-mail adresu.

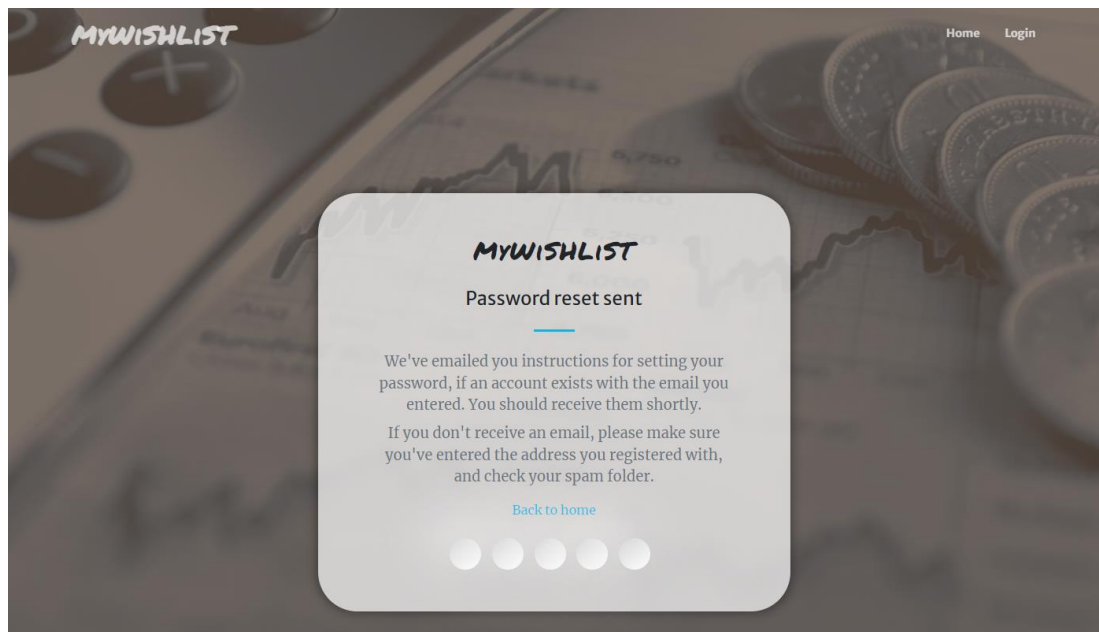
```

<!-- SENT -->
<header class="masthead-login">
  <div class="container px-4 px-lg-5 h-100">
    <div class="row px-4 px-lg-5 h-100 align-items-center justify-content-center text-center">
      <div class="card-register p-4 py-4">
        <div class="text-center py-4">
          <a class="text-dark" style="font-family: 'Permanent Marker', cursive; font-size: xx-large; letter-spacing: 1px;" href="{% url 'home' %}">MYWISHLIST</a>
          <h4 class="pt-4">Password reset sent</h4>
          <hr class="divider" />
          <p class="text-muted px-4 px-lg-5">We've emailed you instructions for setting your password, if an account exists with the email you entered. You should receive them shortly.</p>
          <p class="text-muted px-4 px-lg-5 pb-2">If you don't receive an email, please make sure you've entered the address you registered with, and check your spam folder.</p>
          <a href="{% url 'home' %}">Back to home</a>
          <div class="social-buttons mt-4">
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
          </div>
        </div>
      </div>
    </div>
  </div>
</header>

```

Slika 71. HTML kod stranice obavijesti o slanju uputa

Na Slici 72 prikazan je konačni rezultat prethodnog koda, odnosno prikaz stranice obavijesti o slanju uputa na unesenu e-mail adresu.



Slika 72. Prikaz stranice obavijesti o slanju uputa

Na Slici 73 prikazan je HTML kod obrasca za unos nove lozinke koji također sadrži kontrolu grešaka kao i ostali obrasci aplikacije.

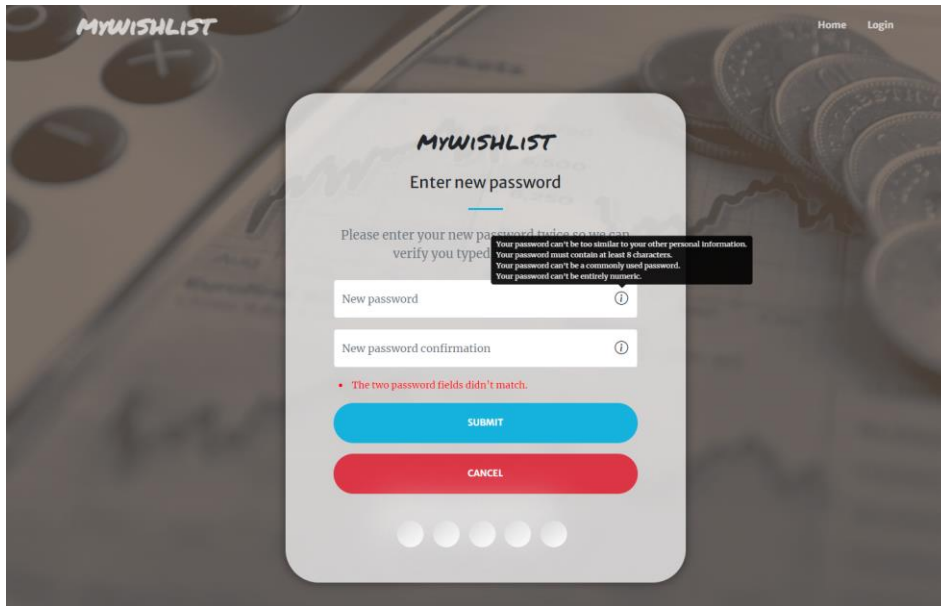
```

<!-- INPUT NEW PASSWORD -->
<header class="masthead-login">
  <div class="container px-4 px-lg-5 h-100">
    <div class="row px-4 px-lg-5 h-100 align-items-center justify-content-center text-center">
      <div class="card-register p-4 py-4">
        <div class="text-center py-4">
          <a class="text-dark" style="font-family: 'Permanent Marker', cursive; font-size: xx-large; letter-spacing: 1px; href="{% url 'home' %}">MYWISHLIST</a>
          <h4 class="pt-4">Enter new password</h4>
          <hr class="divider" />
          <p class="text-muted px-4 px-lg-5">Please enter your new password twice so we can verify you typed it in correctly.</p>
          <div class="px-4 px-lg-5 py-3">
            <form method="POST"{% csrf_token %}>
              <!-- Password input -->
              <div class="form-floating mb-3">
                <input type="password" class="form-control" id="password1">
                <span class="form-icon bi bi-info-circle fs-5" data-bs-toggle="tooltip" data-bs-placement="top" data-bs-html="true" title="{{form.new_password1.help_text}}"></span>
                <label for="password1" class="text-muted">{{form.new_password1.label}}</label>
              </div>
              <!-- Password errors -->
              <div class="text-danger">
                <small>{{form.new_password1.errors}}</small>
              </div>
              <!-- Confirm password input -->
              <div class="form-floating mb-3">
                <input type="password" class="form-control" id="password2">
                <span class="form-icon bi bi-info-circle fs-5" data-bs-toggle="tooltip" data-bs-placement="top" data-bs-html="true" title="{{form.new_password2.help_text}}"></span>
                <label for="password2" class="text-muted">{{form.new_password2.label}}</label>
              </div>
              <!-- Confirm password errors -->
              <div class="text-danger">
                <small>{{form.new_password2.errors}}</small>
              </div>
              <!-- Submit Button -->
              <div class="d-grid pb-3">
                <button type="submit" class="btn btn-primary btn-xl">Submit</button>
                <button type="button" class="btn btn-danger btn-xl">Cancel</button>
              </div>
            </form>
          </div>
          <div class="social-buttons mt-4">
            <button class="neo-button"></button>
            <button class="neo-button"></button>
            <button class="neo-button"></button>
            <button class="neo-button"></button>
            <button class="neo-button"></button>
            <button class="neo-button"></button>
            <button class="neo-button"></button>
          </div>
        </div>
      </div>
    </div>
  </div>
</header>

```

Slika 73. HTML kod obrasca za unos nove lozinke

Na Slici 74 prikazan je konačni rezultat prethodnog koda, odnosno prikaz obrasca za unos nove lozinke.



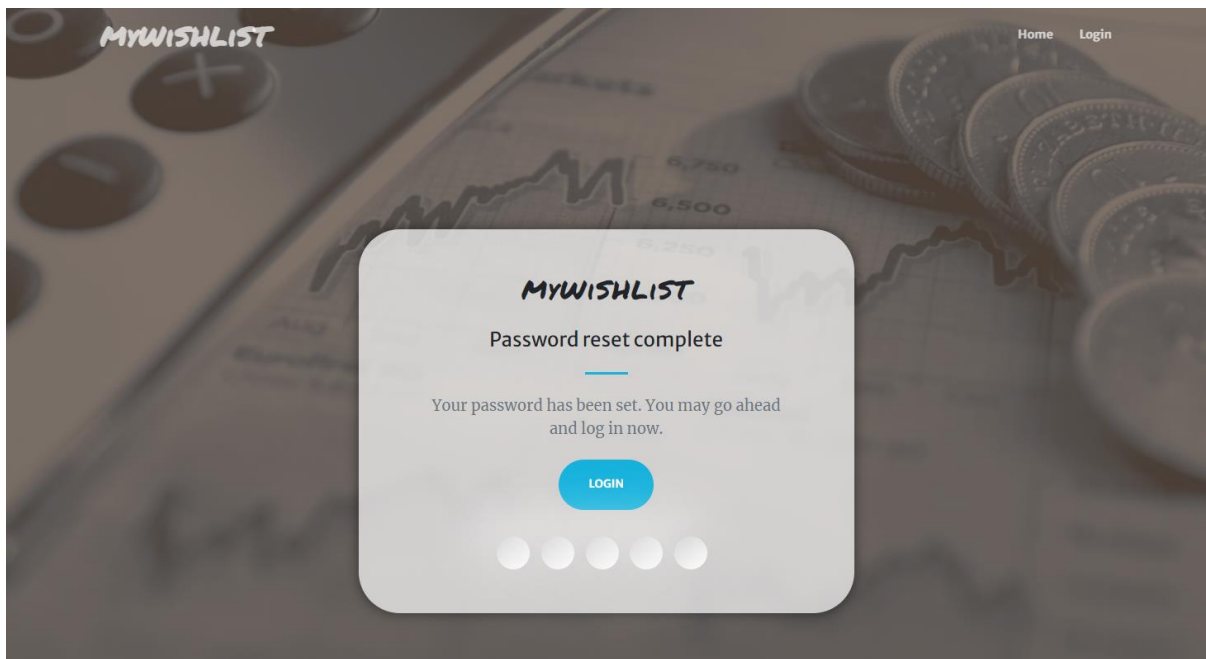
Slika 74. Prikaz obrasca za unos nove lozinke

Na Slici 75 prikazan je HTML kod stranice *password_reset_complete.html*, odnosno posljednje stranice u postupku ponovnog postavljanja lozinke koji sadrži gumb s poveznicom na obrazac za prijavu.

```
<!-- RESET DONE -->
<header class="masthead-login">
  <div class="container px-4 px-lg-5 h-100">
    <div class="row gx-4 gx-lg-5 h-100 align-items-center justify-content-center text-center">
      <div class="card-register p-4 py-4">
        <div class="text-center py-4">
          <a class="text-dark" style="font-family: 'Permanent Marker', cursive; font-size: xx-large; letter-spacing: 1px;" href="{% url 'home' %}">MyWishList</a>
          <h4 class="pt-4">Password reset complete</h4>
          <hr class="divider" />
          <p class="text-muted px-4 px-lg-5 pb-3">Your password has been set. You may go ahead and log in now.</p>
          <a class="btn btn-primary btn-xl mb-2" href="{% url 'login' %}">Login</a>
          <div class="social-buttons mt-4">
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
            <button class="neo-button">
            </button>
          </div>
        </div>
      </div>
    </div>
  </div>
</header>
```

Slika 75. HTML kod potvrđne stranice o izmjeni lozinke

Na Slici 76 prikazan je konačni rezultat prethodnog koda, odnosno prikaz stranice potvrde izmjene lozinke.



Slika 76. Prikaz stranice potvrde o izmjeni lozinke

4.5.5.2. Ponovno postavljanje lozinke pomoću postojeće lozinke

U mapi *templates* u pripadnoj mapi *account* aplikacije stvorene su dvije nove HTML datoteke, a to su *password_change.html* i *password_change_done.html*. Korištene su dvije zadane Django funkcije za ponovno postavljanje lozinke te je jedna proširena, odnosno prilagođena. U datoteci *forms.py* u pripadnoj mapi definirana je nova Python klasa koja nasljeđuje već postojeću funkciju izmjene lozinke u kojoj je definiran dizajn polja kao i pomoćne poruke. Sastoji se od polja za unos postojeće lozinke, polja za unos nove lozinke te polja za ponovni unos nove lozinke. Na Slici 77 prikazana je definicija klase obrasca ponovnog postavljanja lozinke pomoću postojeće lozinke.

```
class CustomPasswordChangeForm>PasswordChangeForm):
    def __init__(self, *args, **kwargs):
        super(CustomPasswordChangeForm, self).__init__(*args, **kwargs)

        self.fields['old_password'].help_text = """<p class="text-start mb-0">Enter your old password.</p>"""
        self.fields['old_password'].label = 'Old password'
        self.fields['old_password'].widget.attrs.update({
            'class': 'form-control',
            'id': 'old_password',
            'type': 'password',
            'placeholder': 'Enter old password...'
        })
        self.fields['new_password1'].help_text = """<p class="text-start mb-0">Your password can't be too similar to your other personal information.</br>
        Your password can't be the same as your old password.</br>
        Your password must contain at least 8 characters.</br>
        Your password can't be a commonly used password.</br>
        Your password can't be entirely numeric.</p>"""
        self.fields['new_password1'].label = 'New password'
        self.fields['new_password1'].widget.attrs.update({
            'class': 'form-control',
            'id': 'new_password1',
            'type': 'password',
            'placeholder': 'Enter new password...'
        })
        self.fields['new_password2'].help_text = """<p class="text-start mb-0">Enter the same password as before, for verification.</p>"""
        self.fields['new_password2'].label = 'New password confirmation'
        self.fields['new_password2'].widget.attrs.update({
            'class': 'form-control',
            'id': 'new_password2',
            'type': 'password',
            'placeholder': 'Confirm new password...'
        })
    })
```

Slika 77. Definicija obrasca za ponovno postavljanje lozinke koristeći postojeću lozinku

Zatim je novostvoreni obrazac povezan na način da proširuje već postojeću funkciju unutar pripadne *views.py* datoteke dijelom koda prikazanim na Slici 78.

```
class CustomPasswordChangeView(PasswordChangeView):  
    form_class = CustomPasswordChangeForm
```

Slika 78. Proširenje zadane klase za ponovno postavljanje lozinke

Nadalje, u pripadnoj datoteci *urls.py* definirane su URL putanje, prikazane na Slici 79, koje koriste stvorene funkcije obrade obrasca.

```
urlpatterns = [  
    path('login/', views.loginUser, name="login"),  
    path('logout/', views.logoutUser, name="logout"),  
    path('register/', views.registerUser, name="register"),  
  
    path('account/', views.account_view, name="account"),  
  
    path('change_password/', views.CustomPasswordChangeView.as_view(template_name="account/password_change.html"), name="password_change"),  
    path('change_password_done/', auth_views.PasswordChangeDoneView.as_view(template_name="account/password_change_done.html"), name="password_change_done"),  
]
```

Slika 79. Definicija URL putanje obrasca ponovnog postavljanja lozinke

U posljednjem koraku prilagođen je dizajn stvorenih HTML datoteka po uzoru na registracijski obrazac, kao i obrazac prijave, kako bi odgovarali ostatku aplikacije. Na Slici 80 prikazan je HTML kod stranice *password_change.html* koja sadrži obrazac za unos nove lozinke, kao i prostor za ispis poruka prilikom greške obrade zahtjeva. Obrascu se može pristupiti putem poveznice na stranici korisničkog računa.

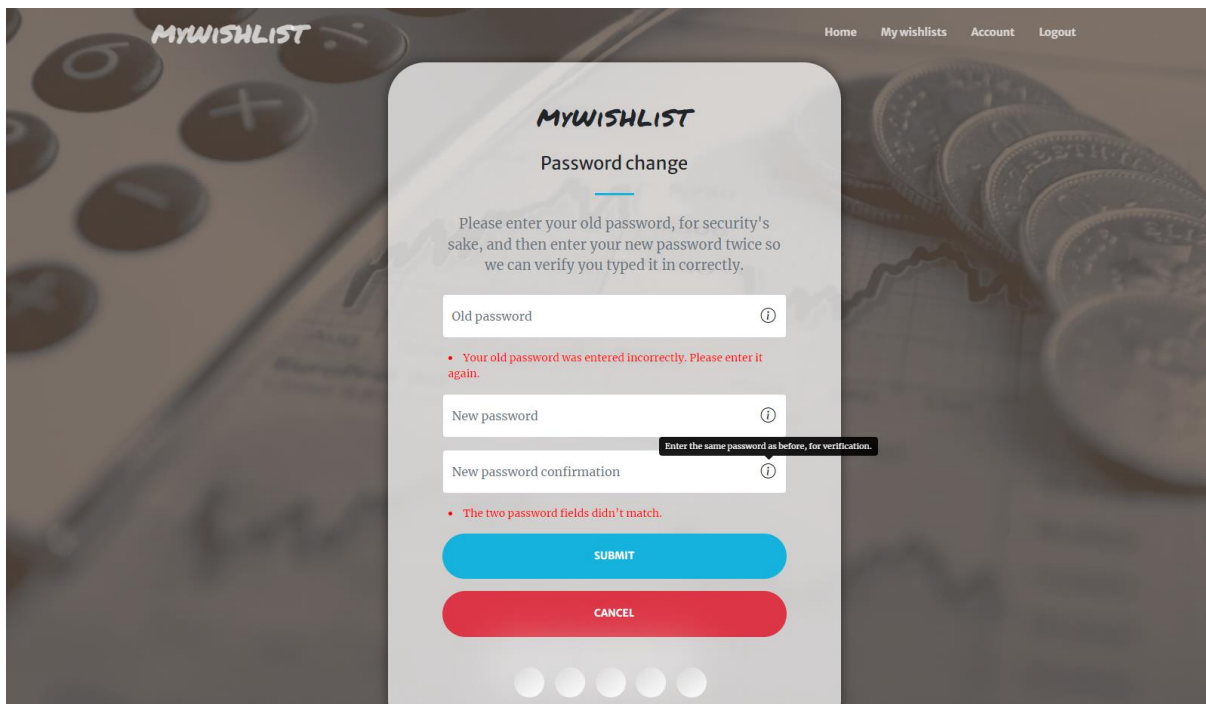
```

<!-- OLD PASS RESET -->
<header class="masthead-register">
  <div class="container px-4 px-lg-5 h-100">
    <div class="row gx-4 gx-lg-5 h-auto align-items-center justify-content-center text-center mt-1">
      <div class="card-register p-4 py-4">
        <div class="text-center py-4">
          <a class="text-dark" style="font-family: 'Permanent Marker', cursive; font-size: xx-large; letter-spacing: 1px;" href="{% url 'home' %}">MyWISHLIST</a>
          <h4 class="pt-4">Password change</h4>
          <hr class="divider" />
          <p class="text-muted px-4 px-lg-5">Please enter your old password, for security's sake, and then enter your new password twice so we can verify you typed it in correctly.</p>
          <div class="px-4 px-lg-5 py-3">
            <form method="POST">{% csrf_token %}
              <!-- OLD Password input -->
              <div class="form-floating mb-3">
                {{ form.old_password }}
                <span class="form-icon bi bi-info-circle fs-5" data-bs-toggle="tooltip" data-bs-placement="top" data-bs-html="true" title="{{ form.old_password.help_text }}"></span>
                <label for="old_password" class="text-muted">{{ form.old_password.label }}</label>
              </div>
              <!-- OLD Password errors -->
              {% for error in form.old_password.errors %}
                <li class="fs-7 mb-3 px-2 text-start" style="color: red;">{{ error }}</li>
              {% endfor %}
              <!-- Password input -->
              <div class="form-floating mb-3">
                {{ form.new_password1 }}
                <span class="form-icon bi bi-info-circle fs-5" data-bs-toggle="tooltip" data-bs-placement="top" data-bs-html="true" title="{{ form.new_password1.help_text }}"></span>
                <label for="new_password1" class="text-muted">{{ form.new_password1.label }}</label>
              </div>
              <!-- Password errors -->
              {% for error in form.new_password1.errors %}
                <li class="fs-7 mb-3 px-2 text-start" style="color: red;">{{ error }}</li>
              {% endfor %}
              <!-- Confirm password input -->
              <div class="form-floating mb-3">
                {{ form.new_password2 }}
                <span class="form-icon bi bi-info-circle fs-5" data-bs-toggle="tooltip" data-bs-placement="top" data-bs-html="true" title="{{ form.new_password2.help_text }}"></span>
                <label for="password2" class="text-muted">{{ form.new_password2.label }}</label>
              </div>
              <!-- Confirm password errors -->
              {% for error in form.new_password2.errors %}
                <li class="fs-7 mb-3 px-2 text-start" style="color: red;">{{ error }}</li>
              {% endfor %}
              <!-- Submit Button -->
              <div class="d-grid pb-3"><button class="btn btn-primary btn-xl" id="submitButton" type="submit">Submit</button></div>
              <div class="d-grid"><a href="{% url 'account' %}" class="btn btn-danger btn-xl" id="cancelButton">Cancel</a></div>
            </form>
          </div>
          <div class="social-buttons mt-4">
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
          </div>
        </div>
      </div>
    </div>
  </div>
</header>

```

Slika 80. HTML kod stranice za unos nove lozinke pomoću postojeće

Na Slici 81 prikazan je konačni rezultat prethodnog koda, odnosno prikaz stranica obrasca za unos nove lozinke pomoći postojeće.



Slika 81. Prikaz stranice obrasca za ponovno postavljanje lozinke

Ako su uneseni ispravni podatci, nakon uspješne obrade obrasca prikazuje se obavijest o potvrdi izmjene lozinke s poveznicom na stranicu korisničkog računa. Na Slici 82 prikazan je HTML kod stranice `password_change_done.html`, odnosno potvrdne stranice izmjene lozinke.

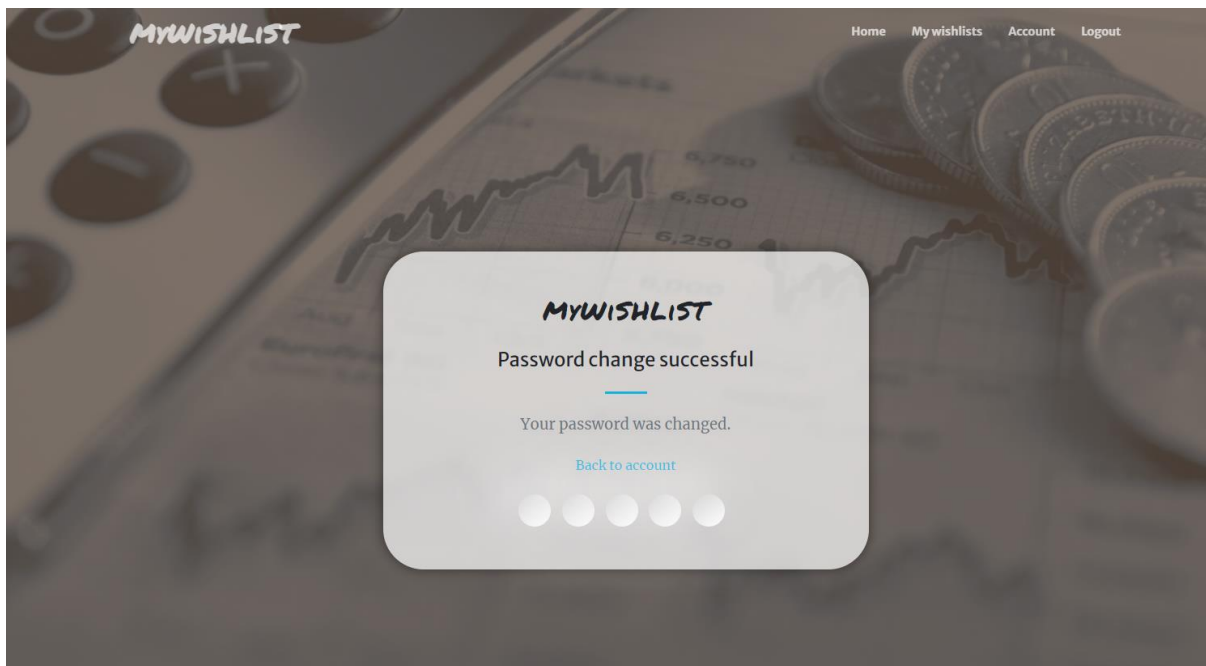
```

<!-- PASS DONE -->
<header class="masthead-login">
  <div class="container px-4 px-lg-5 h-100">
    <div class="row gx-4 gx-lg-5 h-100 align-items-center justify-content-center text-center">
      <div class="card-register p-4 py-4">
        <div class="text-center py-4">
          <a class="text-dark" style="font-family: 'Permanent Marker', cursive; font-size: xx-large; letter-spacing: 1px;" href="{% url 'home' %}">MYWISHLIST</a>
          <h4 class="pt-4">Password change successful</h4>
          <hr class="divider" />
          <p class="text-muted px-4 px-lg-5 pb-3">Your password was changed.</p>
          <a href="{% url 'account' %}">Back to account</a>
          <div class="social-buttons mt-4">
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
            <button class="neo-button">
              </button>
          </div>
        </div>
      </div>
    </div>
  </div>
</header>

```

Slika 82. HTML kod potvrdne stranice izmjene lozinke

Na Slici 83 prikazan je konačni rezultat prethodnog koda, odnosno potvrdne stranice.



Slika 83. Prikaz potvrдне stranice izmjene lozinke

4.6. Stranica lista želja

U mapi *templates* aplikacije *wishlists* stvorena je nova HTML datoteka *wishlists.html* koja sadrži cijeli dizajn i raspored stranice lista želja te HTML datoteka *wishlist_form.html* koja sadrži obrazac za unos nove liste, kao i obrazac za izmjenu postojeće. Stranica je podijeljena na 3 glavna dijela. Prvi dio, s lijeve strane stranice, čini stupac u kojem je smješten gumb za dodavanje novih lista želja. Ispod gumba smještena je traka za pretraživanje lista po imenu, a ispod pretrage smještene su same liste želja. Drugi dio stranice tvore naziv liste, padajući izbornik za sortiranje proizvoda koji postaje funkcionalan tek prilikom odabira liste te gumbi za uređivanje i brisanje liste koji također postaju aktivni prilikom odabira liste. U trećem dijelu stranice stvoreno je mjesto gdje su, prilikom odabira liste, prikazani proizvodi pojedine liste te gumb za dodavanje liste želja koji prilikom odabira liste postaje gumb za dodavanje novog proizvoda.

U datoteci *forms.py* u pripadnoj mapi stvorena je nova klasa u kojoj je definiran obrazac za unos lista kao i dizajn polja istog. Obrazac se sastoji od svega dva polja, a to su naslov i prioritet. Na Slici 84 prikazana je definicija obrasca za unos lista.

```

class WishlistForm(ModelForm):
    class Meta:
        model = Wishlist
        fields = ['title', 'priority']

    def __init__(self, *args, **kwargs):
        super(WishlistForm, self).__init__(*args, **kwargs)

        self.fields['title'].widget.attrs.update({
            'class': 'form-control',
            'id': 'title',
            'type': 'text',
            'placeholder': 'Enter wishlist name...',
        })
        self.fields['priority'].widget.attrs.update({
            'class': 'form-select',
            'id': 'priority',
        })

```

Slika 84. Definicija obrasca za unos lista

U pripadnoj datoteci *views.py* je zatim definirana funkcija koja vraća podatke o listama želja. Na Slici 85 nalazi se prikaz te funkcije.

```

@login_required(login_url="login")
def get_all_wishlists(request):
    account = request.user.account
    wishlists = account.wishlist_set.all()

    context = {
        'wishlists': wishlists,
    }
    return render(request, 'wishlists/wishlists.html', context)

```

Slika 85. Funkcija ispisa lista želja

Također, u istoj datoteci definirane su i funkcije za obradu zahtjeva stvaranja, izmjene i brisanja lista želja. Funkcije za stvaranje i izmjenu koriste prethodno definirani obrazac za unos lista, dok funkcija za brisanje dohvaća i briše listu. Unos lista je izgrađen na način da se liste pridružuju trenutno prijavljenoj instanci korisničkog računa prilikom stvaranja istih, isto kao i prikaz, odnosno prilikom prikaza dohvaćaju se one liste koje su povezane s trenutno prijavljenim korisnikom, što tvori jedinstveno sučelje za svakog korisnika. Definirane su i skočne poruke koje se prikazuju prilikom izvršenja određenog zahtjeva. Na Slici 86 prikazane su funkcije za unos, izmjenu i brisanje lista želja.

```

@login_required(login_url="login")
def create_wishlist(request):
    page = 'create-wishlist'
    account = request.user.account

    form = WishlistForm()

    if request.method == 'POST':
        form = WishlistForm(request.POST)
        if form.is_valid():
            wishlist = form.save(commit=False)
            wishlist.owner = account
            form.save()
            messages.success(request, 'Wishlist created successfully.')
            return redirect('wishlist', pk=wishlist.id)

    context = {'form': form, 'page': page}
    return render(request, 'wishlists/wishlist_form.html', context)

@login_required(login_url="login")
def update_wishlist(request, pk):
    page = 'update-wishlist'
    account = request.user.account
    wishlist = account.wishlist_set.get(id=pk)
    form = WishlistForm(instance = wishlist)

    if request.method == 'POST':
        form = WishlistForm(request.POST, instance=wishlist)
        if form.is_valid():
            form.save()
            messages.success(request, 'Wishlist updated successfully.')
            return redirect('wishlist', pk=wishlist.id)

    context = {'form': form, 'page': page}
    return render(request, 'wishlists/wishlist_form.html', context)

@login_required(login_url="login")
def delete_wishlist(request, pk):
    account = request.user.account
    wishlist = account.wishlist_set.get(id=pk)
    if request.method == 'POST':
        wishlist.delete()
        messages.success(request, 'Wishlist deleted successfully.')
        return redirect('wishlists')

    context = {'object': wishlist}
    return render(request, 'wishlists/delete_template.html', context)

```

Slika 86. Funkcije unosa, izmjene i brisanja lista

Nakon definiranja funkcija, u pripadnoj `urls.py` datoteci dodane su URL putanje, prikazane na Slici 87, koje koriste stvorene funkcije.

```
urlpatterns = [
    path('wishlists/', views.get_all_wishlists, name="wishlists"),
    path('wishlist/<str:pk>', views.get_wishlist_details, name="wishlist"),
    path('create-wishlist/', views.create_wishlist, name="create-wishlist"),
    path('update-wishlist/<str:pk>', views.update_wishlist, name="update-wishlist"),
    path('delete-wishlist/<str:pk>', views.delete_wishlist, name="delete-wishlist"),
]
```

Slika 87. Definicija URL putanje unosa, izmjene i brisanja lista

U datoteci `wishlist_form.html` definiran je cijeli dizajn obrasca za stvaranje, ali i izmjenu lista. Koristeći varijablu `page` iz definiranih funkcija stvorena je dinamična stranica koja se mijenja ovisno o zahtjevu korisnika. Na Slici 88 prikazan je HTML kod obrasca za unos ili brisanje liste.

```
<section class="page-section" id="single-wishlist">
  <div class="container-custom px-4 px-lg-5">
    <div class="row gx-4 gx-lg-5 justify-content-center">
      <div class="col-lg-8 col-xl-6 text-center">
        <h2 class="mt-0">{% if page == 'create-wishlist' %}Add{% elif page == 'update-wishlist' %}Edit{% endif %} Wishlist</h2>
        <hr class="divider" />
        <p class="text-muted mb-5">Fill out the fields to {% if page == 'create-wishlist' %}create{% elif page == 'update-wishlist' %}update{% endif %} your wishlist!</p>
      </div>
    </div>
    <div class="row gx-4 gx-lg-5 justify-content-center mb-5">
      <div class="col-lg-6">
        <!-- FORM -->
        <form id="wishlistForm" action="" method="POST">{% csrf_token %}
          <!-- Name input -->
          <div class="form-floating mb-3">
            {{form.title}}
            <label for="item">Title</label>
          </div>
          <!-- Priority input -->
          <div class="form-floating mb-3">
            {{form.priority}}
            <label for="priority">Priority</label>
          </div>
          <!-- Submit Button -->
          <div class="d-grid pb-3"><button class="btn btn-primary btn-xl" id="submitButton" type="submit">Submit</button></div>
          <div class="d-grid"><button class="btn btn-danger btn-xl" id="cancelButton" onClick="history.go(-1); return false;">Cancel</button></div>
        </form>
      </div>
    </div>
  </div>
</section>
```

Slika 88. HTML kod obrasca za unos i izmjenu liste želja

Na Slici 89 prikazan je konačni rezultat prethodnog koda, odnosno obrazac za unos ili izmjenu liste želja u slučaju zahtjeva novog unosa.

Add Wishlist

Fill out the fields to create your wishlist!

Slika 89. Prikaz obrasca za unos ili izmjenu liste želja

Sučelje stranice lista želja, izuzev pretrage i padajućeg izbornika za sortiranje proizvoda, izgrađeno je bez predložaka isključivo koristeći Bootstrap klase. Na Slici 90 prikazan je HTML kod stranice prikaza lista želja.

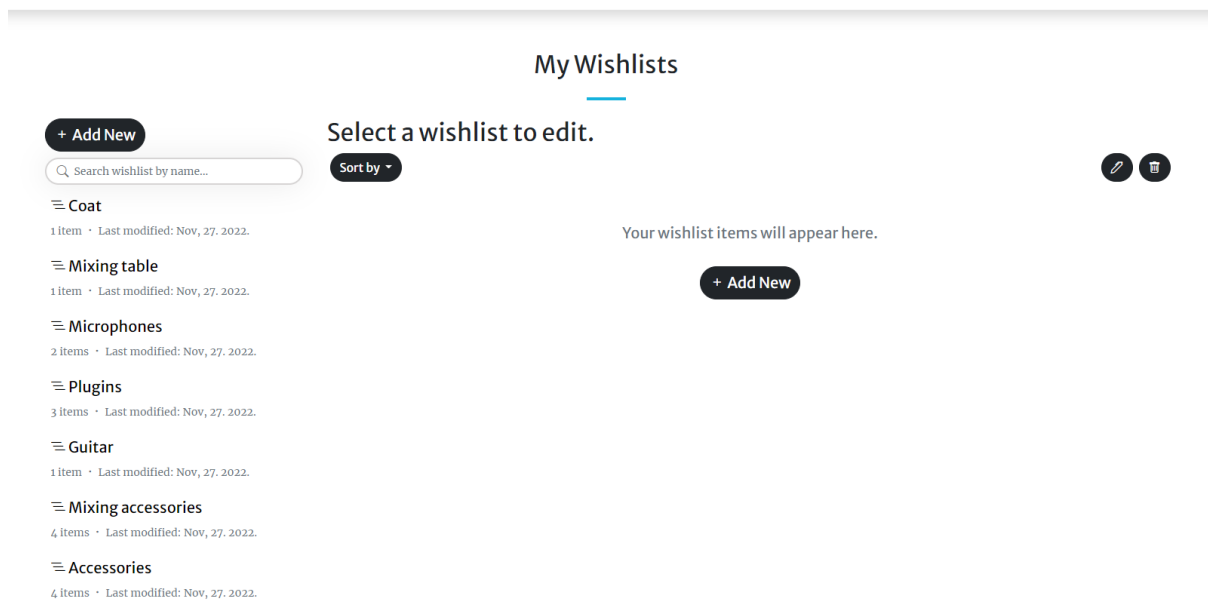
```

<section class="page-section" id="single-wishlist">
  <div class="container-custom px-4 px-lg-5">
    <h2 class="text-center">My Wishlists</h2>
    <hr class="divider" />
    <div class="row">
      <!-- LEFT COLUMN ALL WISHLISTS-->
      <div class="col-lg-3">
        <div class="px-2 mb-2">
          <a href="{% url 'create-wishlist' %}"><button class="btn btn-dark fs-5"><i class="bi bi-plus"></i> Add New</button></a>
        </div>
        <!-- SEARCH BAR -->
        <form id="searchForm" action="{% url 'wishlists' %}" method="GET">
          <div class="row d-flex align-items-center px-2 mb-2">
            <div class="col-md-12">
              <div class="search">
                <button type="submit" class="btn bi bi-search"></button>
                <input type="text" class="form-control fs-7" name="search_query" value="{{search_query}}" placeholder="Search wishlist by name...">
              </div>
            </div>
          </div>
        </form>
        <!-- END SEARCH BAR -->
        <div class="px-2">
          {% if wishlists %}
          {% for wishlist in wishlists %}
          <a href="{% url 'wishlist' wishlist.id %}">
            <div class="p-2 mb-2 card-custom">
              <h5 class="text-truncate text-black"><i class="bi bi-list-nested"></i> {{wishlist.title}}</h5>
              <p class="text-muted mb-0 fs-7 d-inline">{{wishlist.total_products}} item{{wishlist.total_products|pluralize:"s"}}</p>
              <p class="text-muted mb-0 fs-7 d-inline"><i class="bi bi-dot"></i></p>
              <p class="text-muted mb-0 fs-7 d-inline">Last modified: {{wishlist.modified|date:"M, d. Y."}}</p>
            </div>
          </a>
          {% endfor %}
          {% else %}
          <h6 class="text-muted pt-2 ps-1">{{text_message}}</h6>
          {% endif %}
        </div>
      </div>
      <!-- RIGHT COLUMN INDIVIDUAL WISHLIST ITEMS -->
      <div class="col-lg-9">
        <div class="row">
          <div class="col-9">
            <h2>Select a wishlist to edit.</h2>
          </div>
          <div class="pb-5">
            <!-- SORT MENU -->
            <div class="dropdown d-inline float-start mx-1">
              <button class="btn btn-dark dropdown-toggle" type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" aria-expanded="false">
                Sort by
              </button>
              <ul class="dropdown-menu" aria-labelledby="dropdownMenuButton1">
                <li><button class="dropdown-item" name="sort" value="name-asc">Name (A-Z)</button></li>
                <li><button class="dropdown-item" name="sort" value="name-desc">Name (Z-A)</button></li>
                <li><button class="dropdown-item" name="sort" value="price-asc">Price (Low to High)</button></li>
                <li><button class="dropdown-item" name="sort" value="price-desc">Price (High to Low)</button></li>
                <li><button class="dropdown-item" name="sort" value="quantity-asc">Quantity (Low to High)</button></li>
                <li><button class="dropdown-item" name="sort" value="quantity-desc">Quantity (High to Low)</button></li>
                <li><button class="dropdown-item" name="sort" value="priority-asc">Priority (Low to High)</button></li>
                <li><button class="dropdown-item" name="sort" value="priority-desc">Priority (High to Low)</button></li>
              </ul>
            </div>
            <!--END SORT MENU -->
            <button class="btn btn-dark d-inline float-end mx-1"><i class="bi bi-trash"></i></button>
            <button class="btn btn-dark d-inline float-end mx-1"><i class="bi bi-pen"></i></button>
          </div>
          <div class="p-2 d-flex justify-content-center">
            <h5 class="text-muted">Your wishlist items will appear here.</h5>
          </div>
          <div class="d-flex justify-content-center py-3">
            <a href="{% url 'create-wishlist' %}"><button class="btn btn-dark fs-5"><i class="bi bi-plus"></i> Add New</button></a>
          </div>
        </div>
      </div>
    </div>
  </div>
</section>

```

Slika 90. HTML kod stranice lista želja

Na Slici 91 prikazan je konačni rezultat prethodnog koda, odnosno sučelje stranice lista želja.



Slika 91. Prikaz sučelja stranice lista želja

4.6.1. Unos proizvoda na listu želja

U mapi *templates* u pripadnoj mapi aplikacije *wishlists*, kao i kod samih lista želja, stvorena je nova HTML datoteka *wishlist.html* koja je kopija ranije opisane stranice *wishlists.html*, no sadrži i prikaz proizvoda pojedine liste te funkcionalni padajući izbornik za sortiranje kao i ukupni iznos proizvoda. Također, stvorene su HTML datoteke *product_form.html*, koja sadrži obrazac za unos ili izmjenu proizvoda, i *delete_template.html*, što je zajednička datoteka listama i proizvodima za brisanje, ovisno o objektu koji se želi izbrisati.

U datoteci *forms.py* u pripadnoj mapi definirana je nova klasa koja definira obrazac i dizajn polja istog te se obrazac sastoji od polja *naziv*, *cijena*, *količina*, *novčana valuta*, *poveznica* i *prioritet*. Isti obrazac korišten je za stvaranje novog, kao i izmjenu postojećeg proizvoda. Na Slici 92 prikazana je definicija obrasca za unos i izmjenu proizvoda.

```

class ProductForm(ModelForm):
    class Meta:
        model = Product
        fields = ['name', 'price', 'quantity', 'currency', 'url', 'priority']

    def __init__(self, *args, **kwargs):
        super(ProductForm, self).__init__(*args, **kwargs)

        self.fields['name'].widget.attrs.update({
            'class': 'form-control',
            'id': 'item',
            'type': 'text',
            'placeholder': 'Enter item name...'
        })
        self.fields['price'].widget.attrs.update({
            'class': 'form-control',
            'id': 'price',
            'type': 'number',
            'placeholder': 'Enter item price...'
        })
        self.fields['currency'].empty_label = 'Currency'
        self.fields['currency'].widget.attrs.update({
            'class': 'form-select pt-2',
            'id': 'currency',
        })
        self.fields['quantity'].widget.attrs.update({
            'class': 'form-control',
            'id': 'quantity',
            'type': 'number',
            'placeholder': 'Enter item quantity...'
        })
        self.fields['url'].widget.attrs.update({
            'class': 'form-control',
            'id': 'url',
            'type': 'text',
            'placeholder': 'Enter item url...'
        })
        self.fields['priority'].widget.attrs.update({
            'class': 'form-select',
            'id': 'priority',
        })

```

Slika 92. Definicija obrasca za unos i izmjenu proizvoda

Nadalje, u pripadnoj datoteci *views.py* definirane su funkcije za unos, izmjenu i brisanje proizvoda. Funkcije su definirane na način da svaki proizvod koji se dodaje, dodaje se samo trenutno aktivnoj instanci liste prijavljenog korisnika. Funkcije koriste prethodno definirani obrazac za obradu zahtjeva unosa i izmjene, dok funkcija za brisanje dohvaća proizvod i briše ga. Na Slici 93 prikazane su funkcije obrade zahtjeva za unos, izmjenu i brisanje.


```

@login_required(login_url="login")
def create_product(request, pk):
    page = 'create_product'
    wishlist = Wishlist.objects.get(id=pk)
    form = ProductForm()

    if request.method == 'POST':
        form = ProductForm(request.POST)
        product = form.save(commit=False)
        product.wishlist = wishlist
        if form.is_valid():
            product.save()
            messages.success(request, 'Item added successfully.')
            return redirect ('wishlist', pk=wishlist.id)

    context = {
        'form': form,
        'wishlist': wishlist,
        'page': page
    }
    return render(request, 'wishlists/product_form.html', context)

@login_required(login_url="login")
def update_product(request, pk, productPK):
    page = 'update_product'
    wishlist = Wishlist.objects.get(id=pk)
    product = wishlist.product_set.get(id=productPK)
    form = ProductForm(instance=product)

    if request.method == 'POST':
        form = ProductForm(request.POST, instance=product)
        product = form.save(commit=False)
        product.wishlist = wishlist
        if form.is_valid():
            product.save()
            messages.success(request, 'Item updated successfully.')
            return redirect ('wishlist', pk=wishlist.id)

    context = {
        'form': form,
        'wishlist': wishlist,
        'product': product,
        'page': page
    }
    return render(request, 'wishlists/product_form.html', context)

@login_required(login_url="login")
def delete_product(request, pk, productPK):
    wishlist = Wishlist.objects.get(id=pk)
    product = wishlist.product_set.get(id=productPK)
    if request.method == 'POST':
        product.delete()
        messages.success(request, 'Item deleted successfully.')
        return redirect('wishlist', pk=wishlist.id)

    context = {'object': product}
    return render(request, 'wishlists/delete_template.html', context)

```

Slika 93. Funkcije unosa, izmjene i brisanja proizvoda

Također, definirana je i funkcija koja vraća sve detalje liste koji se mogu dohvatiti u HTML datoteci te je time omogućen cjelokupni CRUD. Na Slici 94 prikazana je funkcija obrade prikaza informacija liste i pripadnih proizvoda.

```
@login_required(login_url="login")
def get_wishlist_details(request, pk):
    account = request.user.account
    wishlists = account.wishlist_set.all()

    wishlist = Wishlist.objects.get(id=pk)
    products = wishlist.product_set.all().order_by('name')

    context = {
        'wishlist': wishlist,
        'wishlists': wishlists,
        'products': products,
    }
    return render (request, 'wishlists/wishlist.html', context)
```

Slika 94. Funkcija ispisa podataka liste i pripadnih proizvoda

Nakon definiranja funkcija, u datoteci *urls.py* definirane su i URL putanje, prikazane na Slici 95, koje koriste prethodno definirane funkcije.

```
urlpatterns = [
    path('wishlists/', views.get_all_wishlists, name="wishlists"),
    path('wishlist/<str:pk>', views.get_wishlist_details, name="wishlist"),

    path('create-wishlist/', views.create_wishlist, name="create-wishlist"),
    path('update-wishlist/<str:pk>', views.update_wishlist, name="update-wishlist"),
    path('delete-wishlist/<str:pk>', views.delete_wishlist, name="delete-wishlist"),

    path('wishlist/<str:pk>/create-product/', views.create_product, name="create-product"),
    path('wishlist/<str:pk>/update-product/<str:productPK>', views.update_product, name="update-product"),
    path('wishlist/<str:pk>/delete-product/<str:productPK>', views.delete_product, name="delete-product"),
]
```

Slika 95. Definicija URL putanji pregleda, unosa, izmjene i brisanja proizvoda

U datoteci *wishlist.html* definiran je cjelokupni dizajn stranice lista želja i pripadnih proizvoda. Proizvodi su prikazani u malim Bootstrap karticama te sadrže sve informacije iz obrasca. Na Slici 96 i 97 prikazan je cjelokupni HTML kod stranice pojedine liste želja i pripadnih proizvoda.

```

<section class="page-section" id="single-wishlist">
  <div class="container-custom px-4 px-lg-5">
    <h2 class="text-center">My Wishlists</h2>
    <hr class="divider" />
    <div class="row">
      <!-- LEFT COLUMN ALL WISHLISTS -->
      <div class="col-lg-3">
        <div class="px-2 mb-2">
          <a href="{% url 'create-wishlist' %}"><button class="btn btn-dark fs-5"><i class="bi bi-plus"></i> Add New</button></a>
        </div>
        <!-- SEARCH BAR -->
        <form id="searchForm" action="{% url 'wishlist' wishlist.id %}" method="GET">
          <div class="row d-flex align-items-center px-2 mb-2">
            <div class="col-md-12">
              <div class="search">
                <button type="submit" class="btn bi bi-search"></button>
                <input type="text" class="form-control fs-7" name="search_query" value="{% search_query %}" placeholder="Search wishlist by name...">
              </div>
            </div>
          </div>
        </form>
        <!-- END SEARCH BAR -->
        <div class="px-2">
          <!-- If wishlists -->
          <{% if wishlists %>
            <{% for wishlist in wishlists %>
              <a href="{% url 'wishlist' wishlist.id %}">
                <div class="p-2 mb-2 card-custom {% if request.path == url_wishlist %> active {% endif %}>
                  <h5 class="text-truncate text-black"><i class="bi bi-list-nested"></i> {{wishlist.title}}</h5>
                  <p class="text-muted mb-0 fs-7 d-inline">{{wishlist.total_products}} item{{wishlist.total_products|pluralize:"s"}}</p>
                  <p class="text-muted mb-0 fs-7 d-inline"><i class="bi bi-dot"></i></p>
                  <p class="text-muted mb-0 fs-7 d-inline">Last modified: {{wishlist.modified|date:"M, d. Y." }}</p>
                </div>
              </a>
            <{% endfor %>
          <{% else %>
            <h6 class="text-muted pt-2">{{text_message}}</h6>
          <{% endif %>
        </div>
      </div>
      <!-- RIGHT COLUMN INDIVIDUAL WISHLIST ITEMS -->
      <div class="col-lg-9">
        <div class="row">
          <div class="col-9">
            <h2>{{wishlist.title}}</h2>
          </div>
          <div class="col-3">
            <h6 class="text-end text-muted">{{wishlist.get_priority_display}} priority</h5>
          </div>
        </div>
        <div class="pb-5">
          <!-- SORT MENU -->
          <form action="{% url 'wishlist' wishlist.id %}" method="GET">
            <div class="dropdown d-inline float-start mx-1">
              <button class="btn btn-dark dropdown-toggle" type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" aria-expanded="false">
                Sort by
              </button>
              <ul class="dropdown-menu" aria-labelledby="dropdownMenuButton1">
                <li><button class="dropdown-item" name="sort" value="name-asc">Name (A-Z)</button></li>
                <li><button class="dropdown-item" name="sort" value="name-desc">Name (Z-A)</button></li>
                <li><button class="dropdown-item" name="sort" value="price-asc">Price (Low to High)</button></li>
                <li><button class="dropdown-item" name="sort" value="price-desc">Price (High to Low)</button></li>
                <li><button class="dropdown-item" name="sort" value="quantity-asc">Quantity (Low to High)</button></li>
                <li><button class="dropdown-item" name="sort" value="quantity-desc">Quantity (High to Low)</button></li>
                <li><button class="dropdown-item" name="sort" value="priority-asc">Priority (Low to High)</button></li>
                <li><button class="dropdown-item" name="sort" value="priority-desc">Priority (High to Low)</button></li>
              </ul>
            </div>
          </form>
          <!-- END SORT MENU -->
          <a href="{% url 'delete-wishlist' wishlist.id %}" class="btn btn-dark d-inline float-end mx-1"><i class="bi bi-trash"></i></a>
          <a href="{% url 'update-wishlist' wishlist.id %}" class="btn btn-dark d-inline float-end mx-1"><i class="bi bi-pen"></i></a>
        </div>
      </div>
    </div>
  </div>

```

Slika 96. HTML kod stranice pojedine liste želja

```

<!-- Products -->
{% if wishlist.product_set.all %}
{% for product in products %}
<div class="bg-light p-2 mb-2">
  <div class="row">
    <div class="col-1">
      <p><i class="bi bi-list"></i></p>
      {% if not product.url %}
      <div class="product">
        
      </div>
      {% else %}
      <div class="product-url">
        
      </div>
      {% endif %}
    </div>
    <div class="col-11 ps-5">
      <div class="pb-1">
        <h6 class="d-inline">{{product.name}}</h6>
        <h4 class="d-inline float-end m-0">{{product.currency.tag}} {{product.total_price}}</h4>
      </div>
      <div class="pt-1 pb-4 m-0">
        <p class="float-end text-muted fs-7">{{product.currency.tag}} {{product.price}} per piece</p>
      </div>
      <div class="pb-2 mb-0">
        {% if not product.url %}
        {% else %}
        <a href="{% product.url %}" target="_blank" class="fs-7 overflow-hidden">{{product.url}}</a>
        {% endif %}
      </div>
      <div class="mt-0">
        <p class="d-inline float-start mb-0 mt-2 text-muted fs-7">{{product.get_priority_display}} priority</p>
        <a href="{% url 'delete-product' wishlist.id product.id %}" class="d-inline float-end mx-2" style="color: black; font-size: x-large;"><i class="bi bi-trash"></i></a>
        <a href="{% url 'update-product' wishlist.id product.id %}" class="d-inline float-end mx-2" style="color: black; font-size: x-large;"><i class="bi bi-pen"></i></a>
        <p class="d-inline float-end mb-0 me-4" style="color: black; font-size: x-large;">{{product.quantity}}</p>
      </div>
    </div>
  </div>
</div>
{% endfor %}
{% else %}
<div class="p-2">
  <h5 class="text-muted">There are no items on this wishlist yet.</h5>
</div>
{% endif %}
<div class="d-flex justify-content-center py-3">
  <a href="{% url 'create-product' wishlist.id %}"><button class="btn btn-dark fs-6"><i class="bi bi-plus"></i> Add New</button></a>
</div>
<!-- TOTAL AMOUNT -->
{% if error_message %}
<div class="d-flex flex-row-reverse">
  <h3 class="m-1 d-inline">0</h3><h2 class="fw-bold d-inline">Total: </h2>
</div>
<div class="d-flex flex-row-reverse">
  <p class="text-danger">{{error_message}}</p>
</div>
{% else %}
<div class="d-flex justify-content-end">
  <h2 class="fw-bold">Total: </h2><h3 class="m-1">
    {% if currency_1 == None %}
    {% else %}
    {{currency_1}}
    {% endif %}
    {{wishlist.grand_total_price}}</h3>
  </div>
</div>
</div>
</div>
</div>
</section>

```

Slika 97. HTML kod stranice pojedine liste želja - nastavak

Na Slici 98 prikazan je konačni rezultat prethodnog koda, odnosno prikaz stranice pojedine liste želja.

My Wishlists










+ Add New

Q Search wishlist by name...

- Coat**
1 item · Last modified: Nov, 27, 2022.
- Mixing table**
1 item · Last modified: Nov, 27, 2022.
- Microphones**
2 items · Last modified: Nov, 27, 2022.
- Plugins**
3 items · Last modified: Nov, 27, 2022.
- Guitar**
1 item · Last modified: Nov, 27, 2022.
- Mixing accessories**
4 items · Last modified: Nov, 27, 2022.
- Accessories**
4 items · Last modified: Nov, 27, 2022.

Plugins

Sort by ▾

Item	Priority	Price	Quantity	Actions
 Waves CLA MixHub https://www.thomann.de/intl/waves_cla_mixhub.htm Low priority	Medium priority	HRK 219.03 HRK 219.03 per piece	1	 
 Waves PuigChild Compressor https://www.thomann.de/intl/waves_puigchild_compressor.htm Medium priority	Medium priority	HRK 196.38 HRK 196.38 per piece	1	 
 Waves PuigTec EQs https://www.thomann.de/intl/waves_puigtec_eqs.htm?ref=wil Low priority	Medium priority	HRK 234.14 HRK 234.14 per piece	1	 

Total: HRK 649.55

Slika 98. Prikaz stranice pojedine liste želja

U datoteci *product_form.html* sadržan je cjelokupni dizajn obrasca za unos ili izmjenu proizvoda. Stranica je izgrađena na dinamični način, ovisno o varijabli *page* iz funkcije te Django kodu unutar HTML dokumenta. Gumb *Cancel* vraća korak unazad pomoću definiranja JavaScript svojstva *onClick*. Na Slici 99 prikazan je HTML kod obrasca unosa ili izmjene proizvoda.

```

<section class="page-section" id="contact">
  <div class="container px-4 px-lg-5">
    <div class="row gx-4 gx-lg-5 justify-content-center">
      <div class="col-lg-8 col-xl-6 text-center">
        <h2 class="mt-0">{% if page == 'create_product' %}Add{% elif page == 'update_product' %}Edit{% endif %} item</h2>
        <hr class="divider" />
        <p class="text-muted mb-5">Fill out the fields to {% if page == 'create_product' %}add the item to{% elif page == 'update_product' %}edit the item on{% endif %} your wishlist!</p>
      </div>
    </div>
    <div class="row gx-4 gx-lg-5 justify-content-center mb-5">
      <div class="col-lg-6">
        <!-- FORM -->
        <form id="productForm" action="" method="POST">{% csrf_token %}
          <!-- Name input -->
          <div class="form-floating mb-3">
            <input type="text" class="form-control" value="{{form.name}}"/>
            <label for="item">Item</label>
          </div>
          <div class="row">
            <div class="col-9">
              <!-- Price input -->
              <div class="form-floating mb-3">
                <input type="text" class="form-control" value="{{form.price}}"/>
                <label for="price">Price</label>
              </div>
            </div>
            <div class="col-3">
              <!-- Currency input -->
              <div class="form-floating mb-3">
                <input type="text" class="form-control" value="{{form.currency}}"/>
                <label for="currency">Currency</label>
              </div>
            </div>
          </div>
          <!-- Quantity input -->
          <div class="form-floating mb-3">
            <input type="text" class="form-control" value="{{form.quantity}}"/>
            <label for="quantity">Quantity</label>
          </div>
          <!-- URL input -->
          <div class="form-floating mb-3">
            <input type="text" class="form-control" value="{{form.url}}"/>
            <label for="url">URL</label>
          </div>
          <!-- Priority input -->
          <div class="form-floating mb-3">
            <input type="text" class="form-control" value="{{form.priority}}"/>
            <label for="priority">Priority</label>
          </div>
          <!-- Submit Button -->
          <div class="d-grid pb-3"><button class="btn btn-primary btn-xl" id="submitButton" type="submit">Submit</button></div>
          <div class="d-grid"><button class="btn btn-danger btn-xl" id="cancelButton" onClick="history.go(-1); return false;">Cancel</button></div>
        </form>
      </div>
    </div>
  </div>
</section>

```

Slika 99. HTML kod obrasca za unos ili izmjenu proizvoda

Na Slici 100 prikazan je konačni rezultat prethodnog koda, odnosno prikaz obrasca za unos ili izmjenu proizvoda u slučaju unosa novog proizvoda.

The screenshot shows the 'Add item' form in the MyWishlist application. The form is titled 'Add item' and has a subtitle 'Fill out the fields to add the item to your wishlist!'. The form contains the following fields and controls:

- Item:** A text input field.
- Price:** A text input field with the value '0.0'.
- Currency:** A dropdown menu with a downward arrow.
- Quantity:** A text input field with the value '1'.
- URL:** A text input field.
- Priority:** A dropdown menu with the value 'Low' and a downward arrow.
- Buttons:** Two buttons at the bottom: a blue 'SUBMIT' button and a red 'CANCEL' button.

Slika 100. Prikaz obrasca za unos ili izmjenu proizvoda

U datoteci `delete_template.html` definiran je cjelokupni dizajn zajedničke stranice potvrde brisanja lista želja i proizvoda. Na Slici 101 prikazan je HTML kod stranice potvrde brisanja.

```
<section class="page-section" id="single-wishlist">
  <div class="container-custom px-4 px-lg-5">
    <div class="row gx-4 gx-lg-5 justify-content-center">
      <div class="col-lg-8 col-xl-6 text-center">
        <h2 class="mt-0">Please confirm</h2>
        <hr class="divider" />
      </div>
    </div>
    <div class="row gx-4 gx-lg-5 justify-content-center mb-5">
      <div class="col-lg-6 text-center">
        <form action="" method="POST">{% csrf_token %}
          <p class="mb-3 fs-5">Are you sure you want to delete "{{object}}"?</p>
          <!-- Submit Button-->
          <div class="d-inline"><button class="btn btn-primary btn-xl" id="submitButton" type="submit">Confirm</button></div>
          <div class="d-inline"><button class="btn btn-danger btn-xl" id="cancelButton" onClick="history.go(-1); return false;">Cancel</button></div>
        </form>
      </div>
    </div>
  </div>
</section>
```

Slika 101. HTML kod stranice potvrde brisanja

Na Slici 102 prikazan je rezultat prethodnog koda, odnosno prikaz stranice potvrde brisanja objekta.

Please confirm

Are you sure you want to delete "Waves PuigTec EQs"?

CONFIRM

CANCEL

Slika 102. Prikaz stranice potvrde brisanja objekta

4.6.2. Pretraga

U datotekama *wishlist.html* i *wishlists.html* dodan je dio koda, prikazan na Slici 103, preuzet iz Bootstrap dokumentacije koji čini pretragu. Pretraga je također obrazac, no ne koristi POST, već GET metodu. Definirana je varijabla *search_query* koja se prilikom slanja zahtjeva dohvaća u funkciji ispisa listi i proizvoda.

```
<!-- SEARCH BAR -->
<form id="searchForm" action="{% url 'wishlist' wishlist.id %}" method="GET">
  <div class="row d-flex align-items-center px-2 mb-2">
    <div class="col-md-12">
      <div class="search">
        <button type="submit" class="btn bi bi-search"></button>
        <input type="text" class="form-control fs-7" name="search_query" value="{{search_query}}" placeholder="Search wishlist by name...">
      </div>
    </div>
  </div>
</form>
```

Slika 103. HTML kod pretrage

Koristeći integriranu *icontains* funkciju unutar Djanga te dodavanjem varijable *search_query* u kontekst, omogućeno je pretraživanje listi po nazivu. Ako ukoliko naziv liste sadrži niz znakova unesen u traku pretraživanja na strani klijenta, samo te liste će se i prikazati. Na Slici 104 prikazan je Django kod pretrage.

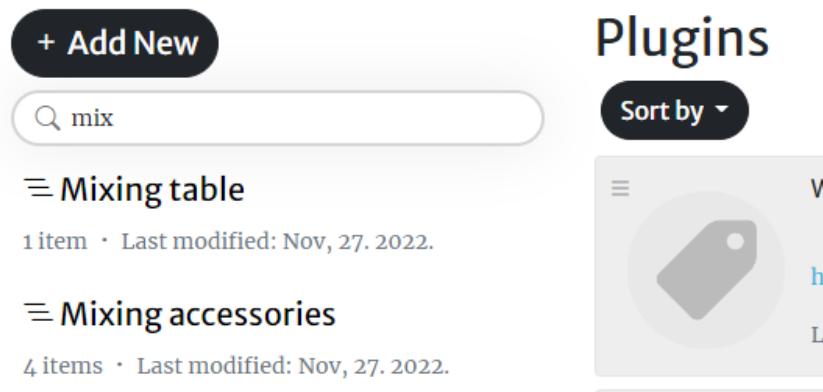
```
search_query = ''

if not wishlists:
    text_message = "You don't have any wishlists."

if request.GET.get('search_query'):
    search_query = request.GET.get('search_query')
    wishlists = account.wishlist_set.filter(title__icontains=search_query)
    if not wishlists:
        text_message = "No wishlists found."
```

Slika 104. Django kod pretrage

Na Slici 105 prikazan je konačni rezultat prethodnih dvaju odsječaka koda, odnosno prikaz pretrage određenog znakovnog niza.



Slika 105. Prikaz pretrage određenog znakovnog niza

4.6.3. Sortiranje proizvoda na listi

U datoteci *wishlist.html* dodan je dio koda, prikazan na Slici 106, koji tvori padajući izbornik preuzet iz Bootstrap dokumentacije te je definirano nekoliko mogućnosti sortiranja.

```
<!-- SORT MENU -->
<form action="{% url 'wishlist' wishlist.id %}" method="GET">
<div class="dropdown d-inline float-start mx-1">
  <button class="btn btn-dark dropdown-toggle" type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" aria-expanded="false">
    Sort by
  </button>
  <ul class="dropdown-menu" aria-labelledby="dropdownMenuButton1">
    <li><button class="dropdown-item" name="sort" value="name-asc">Name (A-Z)</button></li>
    <li><button class="dropdown-item" name="sort" value="name-desc">Name (Z-A)</button></li>
    <li><button class="dropdown-item" name="sort" value="price-asc">Price (Low to High)</button></li>
    <li><button class="dropdown-item" name="sort" value="price-desc">Price (High to Low)</button></li>
    <li><button class="dropdown-item" name="sort" value="quantity-asc">Quantity (Low to High)</button></li>
    <li><button class="dropdown-item" name="sort" value="quantity-desc">Quantity (High to Low)</button></li>
    <li><button class="dropdown-item" name="sort" value="priority-asc">Priority (Low to High)</button></li>
    <li><button class="dropdown-item" name="sort" value="priority-desc">Priority (High to Low)</button></li>
  </ul>
</div>
</form>
<!--END SORT MENU -->
```

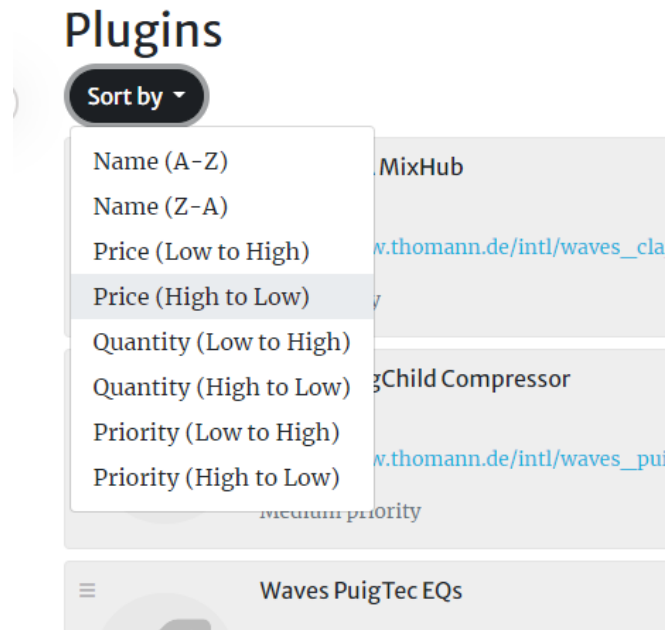
Slika 106. Padajući izbornik mogućnosti sortiranja

Padajući izbornik također je definiran kao obrazac koji koristi GET metodu za slanje vrijednosti varijable *sort* s određenom pridruženom vrijednosti u Django funkciju koja se zatim obrađuje uvjetima prikazanim na Slici 107.

```
#sort conditions
if request.GET.get('sort') == 'name-asc':
    products = wishlist.product_set.all().order_by('name')
elif request.GET.get('sort') == 'name-desc':
    products = wishlist.product_set.all().order_by('-name')
elif request.GET.get('sort') == 'price-asc':
    products = wishlist.product_set.all().order_by('price')
elif request.GET.get('sort') == 'price-desc':
    products = wishlist.product_set.all().order_by('-price')
elif request.GET.get('sort') == 'quantity-asc':
    products = wishlist.product_set.all().order_by('quantity')
elif request.GET.get('sort') == 'quantity-desc':
    products = wishlist.product_set.all().order_by('-quantity')
elif request.GET.get('sort') == 'priority-asc':
    products = wishlist.product_set.all().order_by('priority')
elif request.GET.get('sort') == 'priority-desc':
    products = wishlist.product_set.all().order_by('-priority')
```

Slika 107. Uvjeti različitih izbora sortiranja

Na Slici 108 prikazan je konačni rezultat prethodnih dvaju odsječaka koda, odnosno prikaz mogućnosti sortiranja.



Slika 108. Prikaz mogućnosti sortiranja

4.6.4. Paginacija

U mapi *templates* korijenske mape stvorena je nova datoteka *pagination.html* koja sadrži cjelokupni HTML kod paginacije. U njemu je definirano i nekoliko uvjeta prikaza ovisno o tome postoji li slijedeća ili prethodna stranica. Također, u mapi *templates* pripadne mape aplikacije *wishlists* stvorena je pomoćna datoteka *utils.py* koja služi za očuvanje preglednog koda.

Uvezene su integrirane funkcije paginacije unutar Django web okvira te je, koristeći Django dokumentaciju, definirana funkcija paginacije prikazana na Slici 109.

```

from django.core.paginator import Paginator, PageNotAnInteger, EmptyPage

def paginateWishlists(request, wishlists, results):

    #pagination
    page = request.GET.get('page')
    paginator = Paginator(wishlists, results)

    try:
        wishlists = paginator.page(page)
    except PageNotAnInteger:
        page = 1
        wishlists = paginator.page(page)
    except EmptyPage:
        page = paginator.num_pages
        wishlists = paginator.page(page)

    leftIndex = (int(page) - 2)
    if leftIndex < 1:
        leftIndex = 1

    rightIndex = (int(page) + 3)
    if rightIndex > paginator.num_pages:
        rightIndex = paginator.num_pages + 1

    custom_range = range(leftIndex, rightIndex)

    return custom_range, wishlists

```

Slika 109. Funkcija paginacije

Funkcija je definirana na način da omogućuje prikaz 10 listi na stranici nakon čega se stvara nova stranica sa slijedećim listama. Nakon što je funkcija definirana, ona je i uključena u funkciju prikaza listi želja, kao i u funkciju prikaza pojedine liste kodom prikazanim na Slici 110.

```

custom_range, wishlists = paginateWishlists(request, wishlists, 10)

```

Slika 110. Poziv funkcije paginacije unutar druge funkcije

Na Slici 111 prikazan je HTML kod paginacije koji je uključen u stranice *wishlist.html* te *wishlists.html*.

```

<!-- PAGINATION -->
{% if queryset.has_other_pages %}
<nav aria-label="Page navigation" class="d-flex justify-content-center">
  <ul class="pagination">
    {% if queryset.has_previous %}
    <li class="page-item">
      <a class="page-link" href="?page={{queryset.previous_page_number}}" aria-label="Previous" data-page="{{queryset.previous_page_number}}">
        <span aria-hidden="true">&laquo;</span>
      </a>
    </li>
    {% endif %}

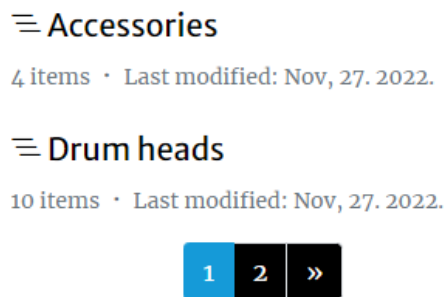
    {% for page in custom_range %}
    {% if page == queryset.number %}
    <li class="page-item active"><a class="page-link" href="?page={{page}}" data-page="{{page}}">{{page}}</a></li>
    {% else %}
    <li class="page-item"><a class="page-link" href="?page={{page}}" data-page="{{page}}">{{page}}</a></li>
    {% endif %}
    {% endfor %}

    {% if queryset.has_next %}
    <li class="page-item">
      <a class="page-link" href="?page={{queryset.next_page_number}}" aria-label="Next" data-page="{{queryset.next_page_number}}">
        <span aria-hidden="true">&raquo;</span>
      </a>
    </li>
    {% endif %}
  </ul>
</nav>
{% endif %}

```

Slika 111. HTML kod paginacije

Na Slici 112 prikazan je konačni rezultat prethodnog koda, odnosno prikaz paginacije u slučaju kada korisnik ima više od 10 lista želja.



Slika 112. Prikaz paginacije

4.7. Otklanjanje grešaka, priprema i objava u produkciju

Prije objave aplikacije u produkciju, sve funkcionalnosti aplikacije temeljito su testirane te je otkrivena pogreška u izračunu ukupnog iznosa proizvoda pojedine liste želja. Ako se na određenoj listi nalaze proizvodi s različitim novčanim valutama, izračun nije točan. U funkciji prikaza liste i pripadnih proizvoda stvorene su dvije nove varijable: *currency_1* i *currency_2*. Varijable dohvaćaju novčane valute svih proizvoda na listi koje se smještaju u listu: jedna ih sortira silazno, a druga uzlazno, te se radi usporedba jednakosti. Ako se vrijednosti varijabli podudaraju, radi se ispis jedne od njih. Na Slici 113 prikazan je kod provjere novčanih valuta.

```
#currency error handling
currency_1 = wishlist.product_set.all().values_list('currency__tag', flat=True).order_by('currency__name').first()
currency_2 = wishlist.product_set.all().values_list('currency__tag', flat=True).order_by('-currency__name').first()

if currency_1 != currency_2:
    error_message = "Please match item currencies to display total value."
```

Slika 113. Kontrola greške novčanih valuta

Na Slici 114 prikazan je povezani HTML kod pripadne stranice koji prikazuje izračun cijene proizvoda ako se valute prethodno definirane varijable podudaraju.

```
<!-- TOTAL AMOUNT -->
{% if error_message %}
<div class="d-flex flex-row-reverse">
  <h3 class="m-1 d-inline"> 0</h3><h2 class="fw-bold d-inline">Total: </h2>
</div>
<div class="d-flex flex-row-reverse">
  <p class="text-danger">{{error_message}}</p>
</div>
{% else %}
<div class="d-flex justify-content-end">
  <h2 class="fw-bold">Total: </h2><h3 class="m-1">
    {% if currency_1 == None %}
    {% else %}
    {{currency_1}}
    {% endif %} {{wishlist.grand_total_price}}</h3>
  {% endif %}
</div>
```

Slika 114. HTML kod prikaza ukupnog iznosa proizvoda liste

4.7.1. Priprema za objavu u produkciju

Budući da zadana SQLite baza podataka nije idealno rješenje, ona je premještena na jednu od AWS usluga koja koristi PostgreSQL sustav, odnosno RDS. Nakon stvaranja RDS instance na AWS-u, u postavkama aplikacije, odnosno u datoteci *settings.py*, izmijenjene su postavke te je Django povezan s RDS-om. Kao i kod e-mail usluge osjetljivi podatci spremljeni su u posebnu datoteku kako ne bi bile izložene prilikom objave koda. Nakon spajanja na bazu, izvršene su naredbe generiranja SQL koda za izradu definiranih tablica te izvršenje istih kao i u početku prilikom korištenja SQLite sustava. Na Slici 115 prikazan je kod postavki baze podataka.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mywishlist',
        'USER': 'djaner',
        'PASSWORD': os.getenv('DB_PASSWORD'),
        'HOST': os.getenv('DB_HOST'),
        'PORT': '5432',
    }
}
```

Slika 115. Postavke RDS baze podataka

Također, u svrhu pojačanja sigurnosti aplikacije isključen je način rada za otkrivanje grešaka kada se trenutna lokacija stranice ne nalazi u lokalnoj mapi, i to pomoću koda prikazanog na Slici 116.

```
if os.getcwd() == '/app':
    DEBUG = False
```

Slika 116. Uvjetovana aktivnost načina rada za otkrivanje grešaka

4.7.2. Objava u produkciju

Odabrana usluga za dijeljenje web stranica je Heroku [18]. Na službenoj stranici stvoren je korisnički račun te novi projekt odgovarajućeg imena. Odabrana je metoda objave putem GitHub-a te je na GitHub-u stvoren novi repozitorij koji će sadržavati cjelokupni kod aplikacije. Unosom repozitorija unutar postavki Heroku projekta GitHub i Heroku su povezani.

Prije svega, instaliran je *gunicorn* koji služi kao posrednik za komunikaciju između web usluge i web aplikacije, te su stvorene datoteke *Procfile*, koja sadrži definiciju o kojoj Django aplikaciji je riječ, te *runtime.txt*, koja sadrži verziju Pythona korištenu unutar aplikacije.

U slijedećem koraku u datoteci *settings.py* varijabli *ALLOWED_HOSTS* dodana je adresa Heroku web stranice prikazana na Slici 117.

```
ALLOWED_HOSTS = ['localhost', '127.0.0.1', 'mywishlistproject.herokuapp.com']
```

Slika 117. Dozvola rada aplikacije na Heroku servisu

Nadalje, instaliran je i sam Heroku te je izrađena tekstualna datoteka *requirements.txt* koja sadrži sve dodatke potrebne za funkcionalni rad stranice te kako bi Heroku mogao i instalirati iste prilikom objave. Na Slici 118 prikazana je naredba stvaranja tekstualne datoteke dodatka.

```
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> pip freeze > requirements.txt
```

Slika 118. Stvaranje datoteke dodatka aplikacije

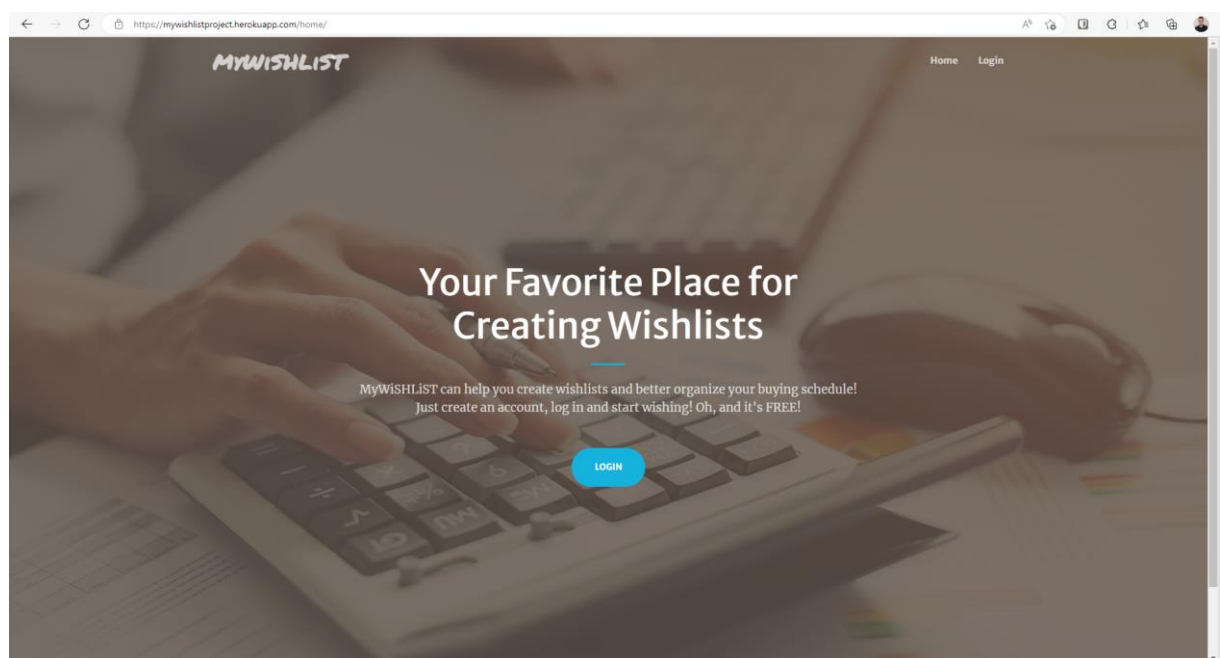
Preteći upute na stranici stvorenog GitHub repozitorija cjelokupni kod aplikacije i prenesen je na glavnu granu istog. Nakon povezivanja repozitorija s lokalnom mapom aplikacije svaku novu verziju koda moguće je prenijeti na repozitorij pomoću tri jednostavne naredbe. Naredba *git add* . priprema napravljene izmjene u korijenskoj mapi za pohranu. Naredba *git commit -m 'comment'* pohranjuje izmjene, a naredba *git push* objavljuje spremljene izmjene na GitHub repozitorij. Navedene naredbe prikazane su na Slici 119.

```
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> git add .|
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> git commit -m 'initial commit'
(.venv) PS C:\Users\DavidJanes\Desktop\Full Stack\MyWishList> git push|
```

Slika 119. Ažuriranje koda na repozitoriju s najnovijim lokalnim kodom

Nakon što je cjelokupni kod prenesen na GitHub repozitorij, a budući da je prethodno isti povezan i s novostvorenim Heroku projektom, unutar postavki Heroku projekta odabrana je opcija ručne objave, a nakon prve objave omogućena i automatska objava prilikom svakog ažuriranja koda na repozitoriju GitHub-a.

U posljednjem koraku unutar postavki Heroku projekta definirane su okolinske varijable koje su se prethodno nalazile u posebnoj datoteci u lokalnoj mapi aplikacije te je aplikacija uspješno objavljena u produkciju. Na Slici 120 nalazi se prikaz stranice na objavljenoj Heroku web adresi.



Slika 120. Prikaz objavljenje aplikacije na Heroku usluzi

5. Zaključak

U ovom radu opisani su besplatni web okviri Django i Bootstrap, njihovi začetci, razvoj te princip na kojem funkcioniraju. Također, ukratko je opisan i JavaScript programski jezik. Detaljno je opisana i prikazana izrada web aplikacije za stvaranje lista želja u kojoj korisnici mogu stvoriti vlastiti korisnički račun, ažurirati podatke računa, ponovno postaviti lozinku na dva različita načina, stvarati, čitati, ažurirati i brisati liste, dodavati proizvode na pojedine liste, kao i ažurirati i brisati iste, pretraživati liste po imenu, sortirati proizvode i drugo. U svakom koraku prikazan je programski kod kao i krajnji rezultat koda. Aplikacija je temeljito testirana u svakom koraku tijekom izrade i dodavanja novih funkcionalnosti, te je objavljena u produkciju na Heroku usluzi.

Za izradu aplikacije potrebno je predznanje Python jezika kao i dobro poznavanje HTML i CSS jezika, međutim i Django i Bootstrap sadrže bogatu dokumentaciju na službenim web sjedištima uz pomoću kojih je jednostavno postići željene rezultate.

Može se zaključiti da je Django prilično intuitivan web okvir koji prema zadanim postavkama sadrži sve što je potrebno za izgradnju bogate i dinamične web aplikacije te ga je vrlo lako proširiti i prilagoditi prema potrebi. Bootstrap je također intuitivan web okvir čijim se jednostavnim korištenjem klasa također vrlo lako mogu postići željeni rezultati. Osobno ne smatram ni jedan ni drugi web okvir presloženim za naučiti te sam vrlo zadovoljan intuitivnosti, brzinom i učinkovitosti tijekom rada koji se može ostvariti koristeći ova dva besplatna web okvira.

Literatura

- [1] Django Software Foundation. „The web framework for perfectionists with deadlines.“ djangoproject.com.
<https://djangoproject.com/> (dohvaćeno Stu. 26, 2022).
- [2] Mozilla Foundation. „Django Introduction – Learn web development.“ [developer.mozilla.org](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction/).
<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction/> (dohvaćeno Stu. 26, 2022).
- [3] Arora, N. „Complete Django History | Python“ [geeksforgeeks.org](https://www.geeksforgeeks.org/complete-django-history-python/).
<https://www.geeksforgeeks.org/complete-django-history-python/> (dohvaćeno Stu. 26, 2022).
- [4] Sumo Logic. „CRUD (create, read, update and delete) - definition & overview“ [sumologic.com](https://www.sumologic.com/glossary/crud/).
<https://www.sumologic.com/glossary/crud/> (dohvaćeno Stu. 26, 2022).
- [5] Quintagroup. „Front-end development with Bootstrap.“ [quintagroup.com](https://quintagroup.com/cms/technology/bootstrap).
<https://quintagroup.com/cms/technology/bootstrap> (dohvaćeno Stu. 26, 2022).
- [6] Quintagroup. „LESS is more agile way to build CSS code.“ [quintagroup.com](https://quintagroup.com/cms/technology/less-css).
<https://quintagroup.com/cms/technology/less-css> (dohvaćeno Stu. 26, 2022).
- [7] Quintagroup. „SASS makes CSS development enjoyable.“ [quintagroup.com](https://quintagroup.com/cms/technology/sass-css).
<https://quintagroup.com/cms/technology/sass-css> (dohvaćeno Stu. 26, 2022).
- [8] Education Ecosystem. „Introduction to Bootstrap Frontend Framework.“ [educationecosystem.com](https://educationecosystem.com/guides/programming/bootstrap/history).
<https://educationecosystem.com/guides/programming/bootstrap/history> (dohvaćeno Stu. 27, 2022).
- [9] Bootstrap. „About - Bootstrap v5.1.“ [getbootstrap.com](https://getbootstrap.com/docs/5.1/about/overview/).
<https://getbootstrap.com/docs/5.1/about/overview/> (dohvaćeno Stu. 27, 2022).
- [10] Tutorials Point. „Javascript - Overview.“ [tutorialspoint.com](https://www.tutorialspoint.com/javascript/javascript_overview.htm).
https://www.tutorialspoint.com/javascript/javascript_overview.htm (dohvaćeno Stu. 27, 2022).
- [11] Python Software Foundation. „Download Python.“ [Python.org](https://www.python.org/downloads/).
<https://www.python.org/downloads/> (dohvaćeno Stu. 28, 2022).
- [12] Pavlić M, *Oblikovanje baza podataka*. Rijeka, 2011.
- [13] JGraph Ltd. „Diagram Software and Flowchart Maker.“ [diagrams.net](https://www.diagrams.net/).
<https://www.diagrams.net/> (dohvaćeno Stu. 28, 2022).
- [14] Start Bootstrap. „Free Bootstrap Themes, Templates, Snippets, and Guides - Start Bootstrap.“ startbootstrap.com.
<https://startbootstrap.com/> (dohvaćeno Stu. 28, 2022).
- [15] Start Bootstrap. „Creative - One Page Bootstrap Theme - Start Bootstrap.“ [startbootstrap.com](https://startbootstrap.com/theme/creative).
<https://startbootstrap.com/theme/creative> (dohvaćeno Stu. 28, 2022).
- [16] Bootstrap. „Introduction · Bootstrap v5.0.“ [getbootstrap.com](https://getbootstrap.com/docs/5.0/).
<https://getbootstrap.com/docs/5.0/> (dohvaćeno Stu. 28, 2022).

- [17] Django Software Foundation. „Django documentation.“ djangoproject.com <https://docs.djangoproject.com/en/4.1/> (dohvačeno Stu. 28, 2022).
- [18] Salesforce. „Cloud Application Platform | Heroku.“ heroku.com <https://www.heroku.com> (dohvačeno Stu. 28, 2022).

Popis slika

Slika 1. Primjer Django dokumentacije	6
Slika 2. Shema Django koda [2].....	8
Slika 3. Primjer Bootstrap dokumentacije.....	11
Slika 4. Instalacija Django web okvira.....	16
Slika 5. Stvaranje Django projekta.....	16
Slika 6. Stvaranje dviju aplikacija unutar projekta.....	16
Slika 7. Povezivanje aplikacija s projektom.....	16
Slika 8. Prikaz poveznice url.py datoteka između aplikacija.....	17
Slika 9. Povezivanje Djanga s mapom za HTML predloške.....	17
Slika 10. Naredba za izradu početnih tablica	18
Slika 11. Stvaranje korisnika s potpunim ovlastima	18
Slika 12. Administracijsko sučelje Django aplikacije.....	18
Slika 13. Dijagram entiteti-veze	19
Slika 14. Definicija modela Wishlist.....	20
Slika 15. Definicija modela Product	21
Slika 16. Definicija modela Currency	22
Slika 17. Definicija modela Account	22
Slika 18. Izvršenje migracijskih naredbi	23
Slika 19. Registracija modela.....	23
Slika 20. Administracijsko sučelje nakon izgradnje modela i unos početnih vrijednosti	23
Slika 21. Raspored mapa Django projekta	24
Slika 22. Povezivanje statičnih datoteka	25
Slika 23. Generiranje URL putanje	25
Slika 24. Instalacija dodatka WhiteNoise	25
Slika 25. Registracija dodatka WhiteNoise.....	25
Slika 26. Početni kod main.html datoteke.....	26
Slika 27. Funkcija home_view	27
Slika 28. Definicija URL putanje.....	27
Slika 29. Dio koda naslovne stranice	27
Slika 30. HTML kod kartice za velike ekrane.....	28
Slika 31. HTML kod kartice za male ekrane.....	28
Slika 32. HTML kod karusela.....	29
Slika 33. Naslovna stranica - neprijavljen korisnik.....	30
Slika 34. Prikaz naslovnice prijavljenog korisnika s nekoliko lista želja	30
Slika 35. Proširenje main.html datoteke.....	31
Slika 36. HTML kod navigacije	31
Slika 37. JavaScript kod efekta navigacije.....	32
Slika 38. Prikaz navigacijske trake	32
Slika 39. HTML kod podnožja.....	32
Slika 40. Prikaz podnožja stranice	33
Slika 41. Prikaz naslovne stranice na malim ekranima.....	33
Slika 42. Definicija obrasca za registraciju	34

Slika 43. Funkcija registracijskog obrasca	34
Slika 44. Povezivanje dvaju korisničkih modela	35
Slika 45. Kod brisanja instance modela	35
Slika 46. URL putanja registracijskog obrasca	35
Slika 47. HTML kod registracijskog obrasca.....	36
Slika 48. Prikaz registracijskog obrasca.....	37
Slika 49. Funkcije prijave i odjave korisnika	37
Slika 50. URL putanje prijave i odjave korisnika	38
Slika 51. HTML kod obrasca prijave	38
Slika 52. Prikaz obrasca za prijavu korisnika.....	39
Slika 53. Definicija obrasca za ažuriranje korisničkih informacija.....	39
Slika 54. Funkcija prikaza i ažuriranja informacija korisničkog računa.....	40
Slika 55. Funkcija povezivanja zadanog i ručno definiranog modela korisničkog računa	40
Slika 56. URL putanja stranice korisničkog računa	41
Slika 57. HTML kod stranice korisničkog računa	42
Slika 58. JavaScript kod za kretnju među karticama.....	43
Slika 59. Prikaz stranice korisničkog računa - ispis podataka	43
Slika 60. Prikaz stranice korisničkog računa - obrazac za ažuriranje informacija.....	44
Slika 61. HTML kod skočnih poruka	44
Slika 62. JavaScript kod za automatsko nestajanje skočnih poruka	45
Slika 63. Prikaz skočne poruke	45
Slika 64. E-mail postavke.....	45
Slika 65. Definicija obrasca za unos e-mail adrese	46
Slika 66. Definicija obrasca za unos nove lozinke	46
Slika 67. Proširenje zadanih funkcija ponovnog postavljanja lozinke.....	46
Slika 68. Definicija URL putanja do odgovarajućih HTML stranica	47
Slika 69. HTML kod obrasca za unos e-mail adrese.....	47
Slika 70. Prikaz obrasca za unos E-mail adrese	48
Slika 71. HTML kod stranice obavijesti o slanju uputa	48
Slika 72. Prikaz stranice obavijesti o slanju uputa	49
Slika 73. HTML kod obrasca za unos nove lozinke.....	49
Slika 74. Prikaz obrasca za unos nove lozinke.....	50
Slika 75. HTML kod potvrdne stranice o izmjeni lozinke	50
Slika 76. Prikaz stranice potvrde o izmjeni lozinke	51
Slika 77. Definicija obrasca za ponovno postavljanje lozinke koristeći postojeću lozinku.....	51
Slika 78. Proširenje zadane klase za ponovno postavljanje lozinke.....	52
Slika 79. Definicija URL putanje obrasca ponovnog postavljanja lozinke.....	52
Slika 80. HTML kod stranice za unos nove lozinke pomoću postojeće	53
Slika 81. Prikaz stranice obrasca za ponovno postavljanje lozinke	54
Slika 82. HTML kod potvrdne stranice izmjene lozinke	54
Slika 83. Prikaz potvrdne stranice izmjene lozinke	55
Slika 84. Definicija obrasca za unos lista.....	56
Slika 85. Funkcija ispisa lista želja	56
Slika 86. Funkcije unosa, izmjene i brisanja lista	57

Slika 87. Definicija URL putanje unosa, izmjene i brisanja lista.....	58
Slika 88. HTML kod obrasca za unos i izmjenu liste želja.....	58
Slika 89. Prikaz obrasca za unos ili izmjenu liste želja.....	59
Slika 90. HTML kod stranice lista želja.....	60
Slika 91. Prikaz sučelja stranice lista želja.....	61
Slika 92. Definicija obrasca za unos i izmjenu proizvoda	62
Slika 93. Funkcije unosa, izmjene i brisanja proizvoda	63
Slika 94. Funkcija ispisa podataka liste i pripadnih proizvoda	64
Slika 95. Definicija URL putanji pregleda, unosa, izmjene i brisanja proizvoda	64
Slika 96. HTML kod stranice pojedine liste želja	65
Slika 97. HTML kod stranice pojedine liste želja - nastavak.....	66
Slika 98. Prikaz stranice pojedine liste želja	67
Slika 99. HTML kod obrasca za unos ili izmjenu proizvoda.....	67
Slika 100. Prikaz obrasca za unos ili izmjenu proizvoda	68
Slika 101. HTML kod stranice potvrde brisanja	68
Slika 102. Prikaz stranice potvrde brisanja objekta	69
Slika 103. HTML kod pretrage	69
Slika 104. Django kod pretrage.....	69
Slika 105. Prikaz pretrage određenog znakovnog niza	70
Slika 106. Padajući izbornik mogućnosti sortiranja.....	70
Slika 107. Uvjeti različitih izbora sortiranja	70
Slika 108. Prikaz mogućnosti sortiranja.....	71
Slika 109. Funkcija paginacije	72
Slika 110. Poziv funkcije paginacije unutar druge funkcije.....	72
Slika 111. HTML kod paginacije	72
Slika 112. Prikaz paginacije	73
Slika 113. Kontrola greške novčanih valuta.....	73
Slika 114. HTML kod prikaza ukupnog iznosa proizvoda liste.....	73
Slika 115. Postavke RDS baze podataka.....	74
Slika 116. Uvjetovana aktivnost načina rada za otkrivanje grešaka	74
Slika 117. Dozvola rada aplikacije na Heroku servisu.....	74
Slika 118. Stvaranje datoteke dodatka aplikacije	75
Slika 119. Ažuriranje koda na repozitoriju s najnovijim lokalnim kodom	75
Slika 120. Prikaz objavljenje aplikacije na Heroku usluzi.....	75

Prilozi

Kod aplikacije moguće je dohvatiti na GitHub repozitoriju na poveznici:
<https://github.com/davidjanes96/myWiSHLiST>

Aplikaciji je moguće pristupiti na sljedećoj poveznici:
<https://mywishlistproject.herokuapp.com/home/>