Web aplikacija za dodjelu poklona

Novak, David

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

WEB APLIKACIJA ZA DODJELU POKLONA

Završni rad

David Novak

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 20.09.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime studenta:	David Novak
Studij, smjer:	Preddiplomski stručni studij Računarstvo
Mat. br. studenta, godina upisa:	AR 4680, 26.07.2018.
OIB studenta:	93979567102
Mentor:	Marina Peko
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Robert Šojo
Član Povjerenstva 1:	Marina Peko
Član Povjerenstva 2:	Dr. sc. Krešimir Romić
Naslov završnog rada:	Web aplikacija za dodjelu poklona
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	Izraditi web aplikaciju za izbor poklona, aplikacija ima 2 role: admin i korisnik. Admin unosi darove između kojih korisnik može izabrati 1 za svako dijete, prema dobnim skupinama. Osigurati komunikaciju i rad s bazom podataka. Tema rezervirana za: David Novak
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina

Datum prijedloga ocjene mentora:	20.09.2021.
Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 15.10.2021.

Ime i prezime studenta:	David Novak
Studij:	Preddiplomski stručni studij Računarstvo
Mat. br. studenta, godina upisa:	AR 4680, 26.07.2018.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za dodjelu poklona**

izrađen pod vodstvom mentora Marina Peko

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __David Novak_____, OIB: __93979567102_____, student/ica Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek na studiju _Preddiplomski stručni studij Računarstvo_, kao autor/ica ocjenskog rada pod naslovom: _Web aplikacija za dodjelu poklona**,**

dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti
b) studentima/icama i djelatnicima/ama ustanove
c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 15.10.2021._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

SADRŽAJ

1. UVOD

Uporaba internetskih trgovina postalo je svakodnevna pojava. Internetske trgovine olakšavaju kupovinu i pronalazak proizvoda na vrlo brz i siguran način. Web aplikacija za odabir poklona, koja je predstavljena u ovom radu, nije internetska trgovina no pomoću nje se rješava problem pri dodijeli poklona. Ova aplikacija omogućava odabir poklona roditelja za svoju djecu te ju možemo poistovjetiti s internetskom trgovinom. Upotrebom ove aplikacije olakšava se proces zapisivanja podataka kao što su: popis poklona, narudžbe, ime i prezime korisnika, osnovne informacije i slično. Iako rade na sličnom sustavu, dodjela poklona tj. sami pokloni se ne naplaćuju već aplikacija služi isključivo kao brži i sigurniji pristup odabiru željenih poklona. Naravno, primjena ovakve aplikacije je široka te se ne mora ograničiti samo na tvrtke već se može koristiti u svim situacijama gdje se želi skratiti vrijeme popisivanja želja za poklone, dostave i prikaza dostupnih poklona za odabir. Također aplikacija sve svoje podatke sprema u digitalnu bazu podataka te se tako može izbjeći neželjeni gubitak narudžbi te se samim time smanjuje mogućnost gubitka narudžbe pojedinog korisnika. Aplikacija je izrađena u tehnologijama koje su široko korištene u izradi web aplikacija. Ovaj rad će u narednim poglavljima objašnjavati na koji način aplikacija funkcionira, što je sve potrebno za izradu ovakve aplikacije, koje su tehnologija korištene te kratak opis istih. Također, bit će prikazani dijelovi kôda koji predstavljaju neke od važnijih funkcionalnosti aplikacije. Glavni dio rada podijeljen je u tri dijela:

1. pozadinski sustav (engl. *backend*) u kojem se opisuju sve korištene tehnologije i postupci pri izradi logičkog dijela aplikacije
2. dizajn sučelja (engl. *frontend*) u kojem se opisuje izgled aplikacije te njeni elementi
3. tehnologije i postupci korišteni pri izradi sučelja.

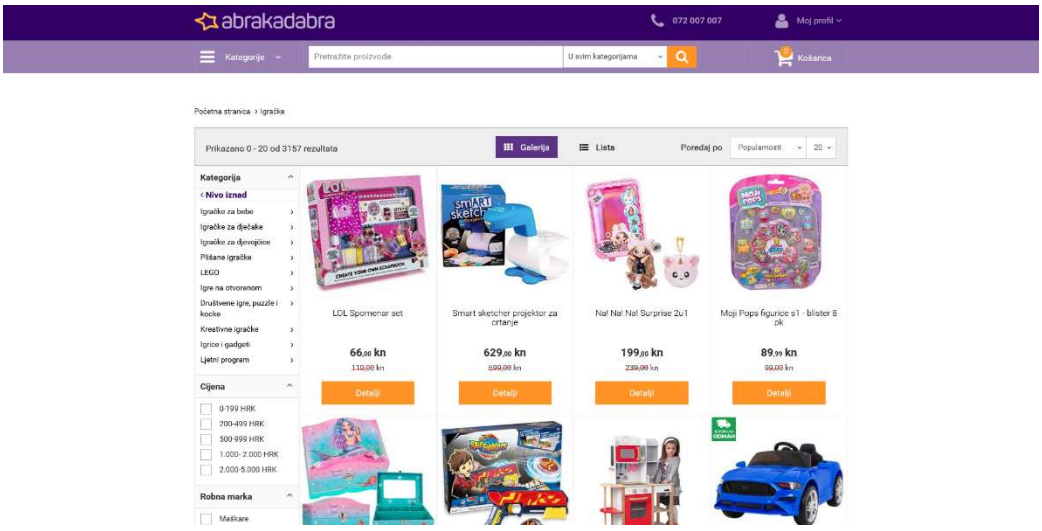
1.1. Zadatak završnog rada

Zadatak ovog završnog rada je bio izraditi web aplikaciju za izbor poklona, sama aplikacija ima dvije uloge: admin i korisnik. Admin unosi poklone između kojih korisnik može izabrati 1 za svako dijete, prema dobnim skupinama. Također neophodno je osigurati komunikaciju i rad s bazom podataka.

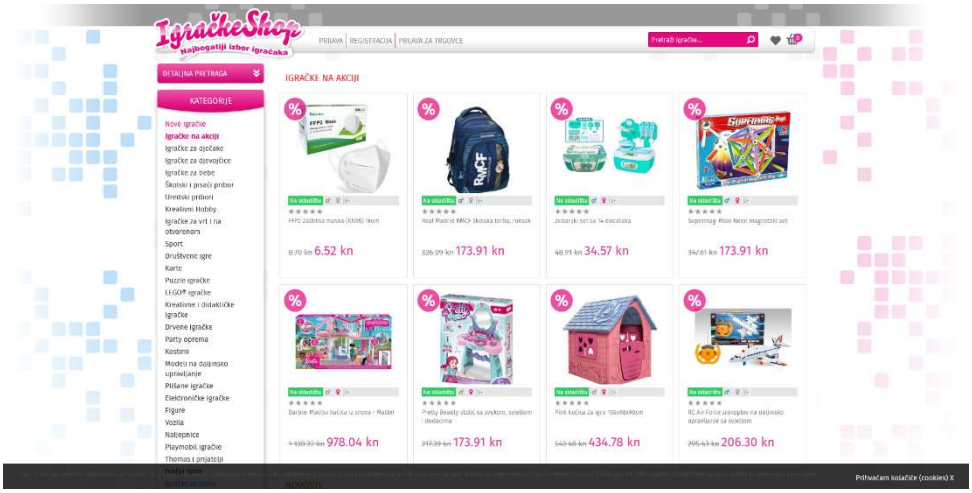
2. O WEB APLIKACIJI ZA DODJELU POKLONA

Ova aplikacija je po svojim glavnim karakteristikama uporabe vrlo slična već postojećim internetskim trgovinama koje svakodnevno koristi vrlo veliki broj ljudi. Sve ovakve web aplikacije imaju sličan scenarij: korisnik pregledava artikle (produkte) te ih ima mogućnost kategorizirati, korisnik odabire željeni proizvod (postavlja ga u virtualnu košaricu) i kada je završio kupovinu upućuje ga se na virtualnu blagajnu gdje odabire način plaćanja, adresu dostave i slično. Web aplikacija za odabir poklona funkcionira vrlo slično, tj. izvedba aplikacije, vrlo je slična takvim internetskim trgovinama no nema implementiranu virtualnu blagajnu. Svrha ovakve aplikacije je olakšati odabir poklona koji se već nalaze u sustavu te su namijenjeni kao poklon, a ne kao proizvod koji će se naplaćivati. Korisnik jednostavno pretraživajući i filtrirajući poklone odabire željeni proizvod za svoje dijete te na kraju dobiva potvrdu o narudžbi. Nakon toga korisniku se taj poklon automatski dodjeljuje te on ima mogućnost preuzimanja tog poklona osim u iznimnim slučajevima gdje bi mu se poklon dostavio na kućnu adresu. Kao najbliže primjere koji imaju sličan pristup i ima sličnu primjenu moguće je izdvojiti internetske trgovine BabyCenter, abrakadabra te IgračkeShop koje su upravo specijalizirane za prodaju dječjih igrački (Slika 2.1., Slika 2.2. i Slika 2.3.). Sve ove aplikacije imaju sljedeće mogućnosti: pretraga poklona, filtriranje poklona, odabir poklona te narudžbu poklona. Ove aplikacije nemaju ograničenja za broj narudžbi jer služe kao internetska trgovina te se svaki poklon naplaćuje za razliku od web aplikacije za dodjelu poklona na kojoj su svi pokloni besplatni.

Slika 2.1. BabyCenter internetska trgovina [1].



Slika 2.2. Abrakadabra internetska trgovina [2].



Slika 2.3. IgračkeShop internetska trgovina [3].

3. FUNKCIONALNOSTI POZADINSKOG SUSTAVA WEB APLIKACIJE ZA DODJELU POKLONA

Za izradu ove web aplikacije odabrana je tehnologija pod nazivom Node.js koji se zasniva na JavaScript skriptnom programskom jeziku [4]. Sam JavaScript se izvršava u web pregledniku na strani korisnika te on omogućava sve funkcionalnosti i interakciju s web aplikacijom [5]. JavaScript je jedan od temeljnih jezika koji se koristi za razvijanje web stranica i aplikacija. Može se reći kako je napravljen po uzoru na Java programski jezik zbog lakšeg korištenja, ali nema funkcionalnosti kao Java. Naime JavaScript nije objektno orijentiran programski jezik već njegov temelj leži na prototipu Jave i tu prestaje svaka povezanost s tim programskim jezikom. Ta dva jezika ne dijele zajednička obilježja. Jedna od zanimljivosti koja kruži programerskim svijetom jest da je JavaScript jezik dobio takvo ime samo iz razloga jer je Java u vrijeme pojavljivanja JavaScript-a bila vrlo popularna i marketinški se pokušalo svratiti pažnju na JavaScript. Trenutno, JavaScript je jedan od najpopularnijih jezika u svijetu te prema GitHub-ovom izvješću Octoverse [6] najviše repozitorija se sadrži upravo od JavaScript jezika, više nego bilo kojeg drugog jezika.

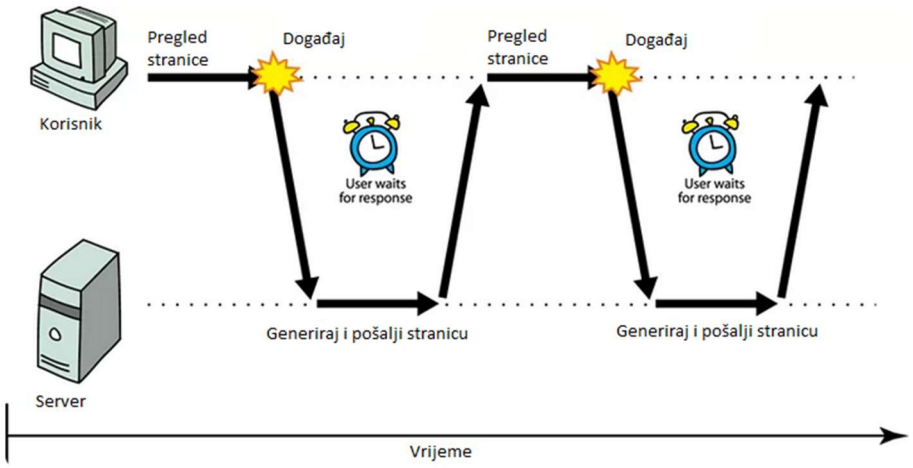
3.1. Node.js

Node.js je pozadinsko JavaScript razvojno okruženje koje se pokreće na V8 JavaScript tehnologiji i izvršava JavaScript kôd izvan internet pretraživača. V8 tehnologija je Google-ov JavaScript i WebAssembly mehanizam otvorenog kôda koji je napisan u C++ programskom jeziku. Node.js programeru dopušta upotrebu JavaScript kôda kako bi napisali alate za pisanje skripti koje će se izvoditi na serverskoj strani kako bi proizveli dinamične i interaktivne web stranice [7]. Sam Node.js se pojavio uz pomoć njegovog kreatora Ryan Dahl-a 2009 godine te je napisan u C, C++ i JavaScript-u. Može se pokretati na mnoštvu operativnih sustava kao što su Windows, macOS, Linux, SmartOS i sl. Node.js jedna je od najpoznatijih JavaScript platformi te se može reći kako programeri/tvrtke odabiru baš tu platformu kao temelj za izradu svih svojih web aplikacija. Node.js omogućava upotrebu jedinstvenog jezika između sučelja i pozadinskog sustava. U biti možemo reći kako bi se pomoću Node.js-a mogla realizirati cijela web aplikacija (što nije slučaj u izradi ovog završnog rada).

3.1.1. Karakteristike Node.js-a

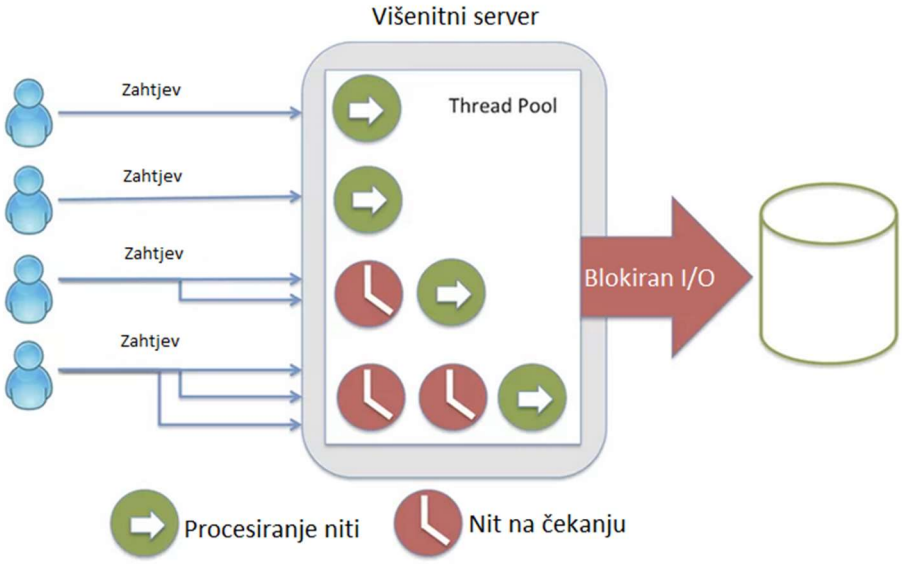
Tradicionalno programiranje (Slika 3.1.) obavljalo se sinkrono što bi u prijevodu značilo da se izvršava pojedina linija kôda dok sustav čeka rezultat. Zatim se rezultat procesira te se izvršavanje programa nastavlja. Možemo zaključiti kako je ovo izvedivo i čak prihvatljivo u manjim sustavima

gdje se ne zahtijeva brzina. Međutim, kada govorimo o velikim sustavima koji se izvršavaju u stvarnom vremenu s vrlo velikim brojem korisnika, može doći do velikih zastoja koji na današnjem nivou razvitka web tehnologija nisu prihvatljivi [8].

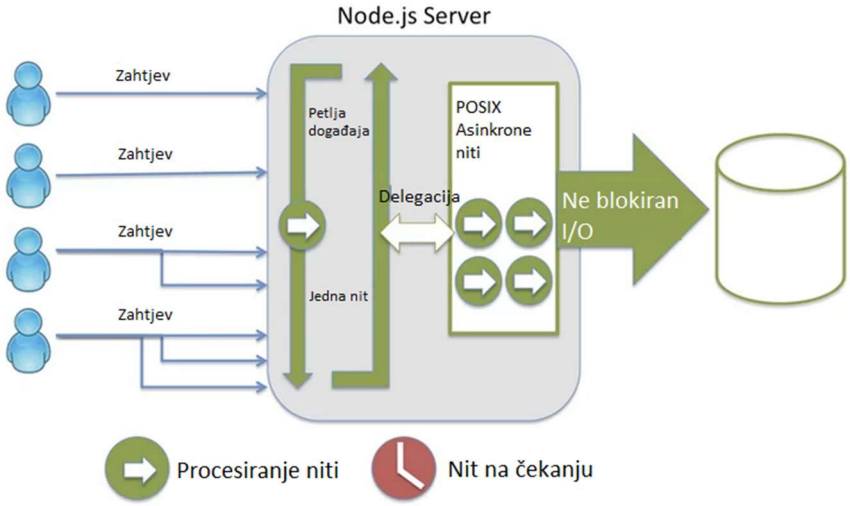


Slika 3.1. Prikaz sinkronog izvođenja programskog kôda [8].

Pojavom jezika poput Jave i C#, rješenje za ovaj problem pojavilo se uvođenjem niti izvršavanja (engl. *Thread*). Međutim i takva vrsta programiranja može izazvati probleme. Na primjer, niti pokušavaju pristupiti resursima u isto vrijeme te može doći do stanja u kojem se niti međusobno blokiraju i sustav se jednostavno sruši jer nema dovoljno resursa (Slika 3.2.). Node.js tj. JavaScript na kojem se Node.js temelji ima drukčiji pristup takvom izazovu. Pri izvođenju programa uvijek će se izvršavati samo jedna nit. Kada se npr. čita iz baze, što je spora I/O operacija, program neće čekati već će nastaviti izvršavanje nadolazeće linije kôda. Kada se I/O operacija vrati pokrenut će se tzv. funkcija povratnog poziva (engl. *callback function*) i rezultat će se procesirati (Slika 3.3.). Node.js ima mogućnost jednostavnog asinkronog događajem pokrenutog (engl. *event-driven*) modela programiranja pri izradi web aplikacija.



Slika 3.2. Server s više niti (thread-ova) [8].



Slika 3.3. Node.js server [8].

3.1.2. NPM: Node Package Manager

Jedna od najbitnijih karakteristika Node.js-a jest njegova odlična ugrađena podrška za upravljanje paketima koji se mogu koristiti za razvijanje aplikacije. NPM dolazi sa svakom Node.js instalacijom te se pomoću njega može instalirati mnoštvo paketa za izradu aplikacije. NPM uklanja potrebu za ručnim instaliranjem softvera koji je potreban za rad aplikacije te također on ima mogućnost bavljenja ažuriranjem, konfiguriranjem i uklanjanjem svih komponenti. Važno je napomenuti da se NPM koristi preko terminala te zahtjeva poznavanje određenih komandi koje koristi sam NPM. On koristi ugniježđeno drvo ovisnosti (engl. *nested dependency tree*). Takvo "drvo" podrazumijeva da se za svaki paket koji se instalira automatski instalira i njegova ovisnost (engl. *dependency*) tako da jedna ovisnost može biti učitana za različite programe.

3.1.3. Instalacija NPM-a i rad s paketima

Instalacija NPM-a je vrlo jednostavna i dogodi se automatski kada se instalira Node.js. Osim instalacije moguće je napraviti ažuriranje NPM-a jer s vremena na vrijeme izlaze novije i poboljšane verzije. To se može napraviti pokretanjem naredbe „npm install -global npm". Nakon što smo uspješno instalirali NPM možemo ga koristiti u našem projektu pokretanjem naredbe za prvu inicijalizaciju upotrebom naredbe „npm init" [9].

Kada je projekt postavljen i NPM instaliran možemo početi s instaliranjem paketa potrebnih za izradu aplikacije. Postoje tri vrste instalacija, a to su: globalna (naredba: „npm install -global nazivPaketa"), lokalna (naredba: „npm install nazivPaketa [flag]") i instalacija svih paketa iz package.json datoteke pri kloniranju projekta s primjerice GitHub-a (naredba: „npm install").

3.1.4. Express.js

Jedan od paketa koji su korišteni za izradu web aplikacije za dodjelu poklona jest Express.js. On je radni okvir (engl. *framework*) tj. skup pravila i funkcija koji su korišteni za implementaciju strukture aplikacije. Radni okvir nije zaseban programski jezik već se on temelji na nekom od temeljnih programskih jezika kao što je u ovom slučaju JavaScript. Možemo reći kako nam radni okvir olakšava pisanje programskog kôda jer omogućava, na čovjeku razumljiviji način, pisati potreban programski kôd te je on lakši za implementirati i čitati. Express je framework Node-a te je dizajniran za stvaranje web aplikacija i API-ja (engl. *Application Programming Interface*). Napisan je u JavaScript-u te je standardni dio Node.js platforme. Express.js kao radni okvir služi isključivo za izgradnju pozadinskog sustava web aplikacije te u suradnji s nekim drugim radnim okvirom ima mogućnost kreiranja web aplikacije kao cjeline [10]. Također kompatibilan je sa svim tipovima operativnih sustava s kojima je kompatibilan i Node.js što ga čini vrlo korisnim

3.2. Baza podataka (MySQL)

Idući korak pri izradi pozadinskog sučelja je bilo dizajniranje baze podataka te implementiranje iste. Za bazu podataka korišten je MySQL, besplatan sustav za upravljanje bazom podataka otvorenog kôda. Optimizirana je kako bi bila brza nauštrb funkcionalnosti, ali ju krasi stabilnost, dobro dokumentirani moduli te ekstenzije [11]. Baze ovakvih vrsta su relacijskog tipa koji se pokazao kao jedan od najboljih načina skladištenja i pretraživanja velikih količina podataka i u suštini predstavlja osnovu svakog informacijskog sustava.

3.2.1. Osnovni pojmovi baze u projektiranju baze podataka

Pri kreiranju baze podataka na početku je potrebno napraviti shemu baze, koja se u kasnijem postupku pretvara u određen broj tablica koje se koriste za pohranjivanje podataka. Osnovni element koji se zapravo pohranjuje u bazu jest entitet te on može predstavljati bilo što: osobu, objekt, službu i slično. Nakon entiteta važan pojam je relacija te ona predstavlja odnos između entiteta. Prema vrsti relacije se mogu podijeliti u 3 grupe: jedan prema jedan, jedan prema više odnosno više prema jedan te više prema više. Svaka tablica se sastoji od stupaca i redaka, stupac predstavlja pohranu pojedinih podataka tj. atributa o entitetu, a redci sadrže sve podatke jednog entiteta. Tablica mora imati primarni ključ koji predstavlja jedinstvenost podatka i samim time jedinstvenost tablice.

3.2.2. Baza podataka web aplikacije za odabir poklona

U aplikaciji MySQL Workbench napisana je skripta za bazu podataka koja je nakon pisanja izvršena i baza podataka je stvorena. Sama aplikacija se sadrži od četiri tablice: User, Gift, UserChild, Wishlist te GiftOrder. Svi dijelovi tablica i cijela skripta baze podataka prikazana je na Slici 3.4.

Tablice su povezane na načine:

- jedan korisnik tj. roditelj može imati više djece, ali svako dijete može imati jednog roditelja
- jedan korisnik može imati jedan popis želja s nekoliko želja
- jedan korisnik može imati više narudžbi (ovisno koliko djece ima)

Na takav način povezivanja osigurane su veze svih podataka koji će biti korišteni u aplikaciji te je baza podataka pripremljena za pisanje pozadinskog sustava aplikacije.

```
CREATE DATABASE ribbon;

CREATE TABLE User(
    Id VARCHAR(36),
    FirstName VARCHAR (50),
    LastName VARCHAR (50),
    Email VARCHAR (50),
    Username VARCHAR (50),
    Password VARCHAR (255),
    CONSTRAINT PK_User_UserId PRIMARY KEY (Id)
);

CREATE TABLE Gift(
    Id VARCHAR(36),
    Name VARCHAR (50),
    Quantity INT,
    Description VARCHAR (255),
    Image VARCHAR (255),
    CONSTRAINT PK_Gift_Id PRIMARY KEY (Id)
);

CREATE TABLE UserChild(
    Id VARCHAR(36),
    UserId VARCHAR(36),
    Age INT,
    Name VARCHAR (50),
    CONSTRAINT PK_UserChild_Id PRIMARY KEY (Id),
    CONSTRAINT FK_UserChild_User_UserId FOREIGN KEY (UserId)
    REFERENCES User(Id) ON DELETE CASCADE
);

CREATE TABLE Wishlist(
    Id VARCHAR(36),
    UserId INT,
    GiftName VARCHAR (50),
    GiftDescription VARCHAR (255),
    GiftImage VARCHAR (255),
    CONSTRAINT PK_Wishlist_Id PRIMARY KEY (Id),
    CONSTRAINT FK_Wishlist_User_UserId FOREIGN KEY (UserId)
    REFERENCES User(Id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Slika 3.4. Primjer skripte baze podataka.

3.3. Programski kôd

U prijašnjim poglavljima opisana je sama aplikacija te na koji način funkcionira. U ovom potpoglavlju bit će pojašnjen sam kôd te programerska strana aplikacije. Važno je napomenuti kako je aplikacija napisana u velikom broju linija kôda te će se u narednim potpoglavljima prikazati samo dijelovi kôda koji su bitni te na određen način zanimljivi. Sama aplikacija je podijeljena u više direktorija i datoteka (Slika 3.5.) te će biti opisan svaki direktorij i njegov sadržaj. Svi direktoriji i datoteke se nazivaju po onome što sadrže. Datoteke se u praksi nazivaju po onome što rade te se na to još doda sufiks koji predstavlja dio aplikacije o kojem se radi. Na primjer za rutu prema poklonima imat ćemo naziv GiftRouter. Upotrebom standardiziranih naziva i rasporeda direktorija postiže se bolje razumijevanje strukture aplikacije.

Slika 3.5. Prikaz rasporeda direktorija.

3.3.1. Index.ts

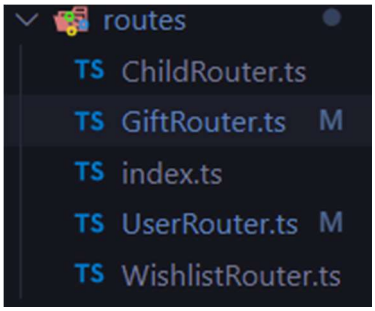
U ovoj datoteci se nalazi temelj aplikacije (Slika 3.6.). Iz nje se aplikacija pokreće te se u njoj nalazi početna točka aplikacije. Između ostalog u njoj se određuje komunikacijska krajnja točka (engl. *port*) na kojem će server tražiti aplikaciju. Također definiraju se određeni uvozi (engl. *import*) modula i datoteka koji se koriste u ovom dijelu aplikacije a to su: rute na kojima će aplikacija funkcionirati te Express.js radni okvir.

```
import express from 'express';
import router from './api/routes/index';
require('dotenv').config();

const app = express();
const PORT = 8000;

app.use(express.json());
app.use(express.urlencoded({
  extended: true
}));

app.use(router);

app.listen(PORT, () => {
  console.log(`⚡[server]: Server is running at https://localhost:${PORT}`);
});
```

Slika 3.6. Prikaz kôda u index.ts datoteci.

3.3.2. *Routes* direktorij

U ovom direktoriju nalaze se sve potrebne rute preko kojih će aplikacija znati pristupati određenom dijelu aplikacije. Ona također ima svoju index.ts datoteku koja sadrži skup svih ruta (Slika 3.8.) koje se nalaze u aplikaciji te ona izravno komunicira sa index.ts datotekom koja se nalazi u korijenskom direktoriju (engl. *root*) aplikacije. Iz nje se napravi izvoz (engl. *export*) glavnog usmjerivača (engl. *router*) koji se tada koristi za pisanje pojedinih ruta prema npr. poklonima, korisniku itd.



Slika 3.7. Datoteke u direktoriju routes

```
import {Router} from 'express';
import giftRouter from './GiftRouter';
import userRouter from './UserRouter';


const router = Router();

router.use("/gifts", giftRouter);
router.use("/users", userRouter);

export default router;
```

Slika 3.8. Primjer index.ts datoteke koja sadrži sve rute

Osim index.ts datoteke tu se nalaze i ostale datoteke (Slika 3.9.) koje sadrže sve potrebne rute preko kojih će se moći implementirati određene funkcionalnosti. Također tu se nalaze i pozivi metoda iz kontrolera (engl. *controller*) kako bi aplikacija mogla prepoznati metode koje mora pozivati kada korisnik želi pristupiti nekoj od ruta.

```
import {Router} from 'express';
import {GiftController} from "../controllers/GiftController";
import auth from '../middlewares/auth';

const giftRouter = Router();
giftRouter.use(auth);

const giftController = new GiftController();

giftRouter.get('/', giftController.GetGifts);

giftRouter.get('/:id', giftController.GetSingleGift);

giftRouter.post('/', giftController.CreateGift);

giftRouter.put("/:id", giftController.UpdateGift);

giftRouter.delete('/:id', giftController.DeleteGift);

export default giftRouter;
```

Slika 3.9. Datoteka GiftRouter.ts i prikaz njenih ruta.

3.3.3. *Models* direktorij i sequelize

Iako je u poglavlju 3.3. navedeno kako je napisana i pokrenuta skripta za bazu podataka potrebno je tu bazu i njene komponente pretvoriti u kôd razumljiv Node.js-u i Express.js-u. Kako bi to bilo moguće korišten je NPM paket pod nazivom sequelize. Taj paket nam omogućava čitljivije i razumljivije pisanje programskog kôda vezanog za baze podataka. Upotrebom sequelize-a nije potrebno pisati MySQL upite na bazu podataka koji ponekad ovisno o kompleksnosti zahtjeva mogu biti vrlo veliki i nečitljivi. On te vrlo velike upite olakša te se upotrebom jedne ili nekoliko ključnih riječi dobije isti rezultat. Kao i svaki dosadašnji direktorij i Models direktorij ima svoju index.ts datoteku iz koje sve počinje. U ovoj index.ts datoteci (Slika 3.10.) se ostvaruje povezivanje s bazom podataka kako bi iz nje bilo moguće čitati podatke i upisivati podatke. Nakon povezivanja na bazu potrebno je napraviti implementaciju tablica iz baze podataka. Ovdje tablice postaju modeli (Slika 3.11.) koji predstavljaju sadržaj i definiraju što se nalazi u tablici.

```ts
import * as sequelize from "sequelize";
import { UserFactory } from "./UserModel";
import { GiftFactory } from "./GiftModel";
import { ChildFactory } from "./ChildModel";
import { WishlistFactory } from "./WishlistModel";
require('dotenv').config();


const DatabaseName = process.env.DB_NAME || "";
const DatabaseUser = process.env.DB_USER || "";
const DatabasePassword = process.env.DB_PASSWORD || "";
const DatabaseHost = process.env.DB_HOST;

export const dbConfig = new sequelize.Sequelize(
  (DatabaseName),
  (DatabaseUser),
  (DatabasePassword),
  {
    port: Number(process.env.DB_PORT) || 3306,
    host: DatabaseHost,
    dialect: "mysql",
    pool: {
        min: 0,
        max: 5,
        acquire: 30000,
        idle: 10000,
    }
  },
);

export const dbUser = UserFactory(dbConfig);
export const dbGift = GiftFactory(dbConfig);
export const dbChild = ChildFactory(dbConfig);
export const dbWishlist = WishlistFactory(dbConfig);
```

Slika 3.10. Datoteka index.ts i prikaz povezivanja na bazu podataka.

```ts
import { BuildOptions, DataTypes, IntegerDataType, Model, Sequelize } from "sequelize";

export interface GiftAttributes {
  Id: string,
  Name: string,
  Quantity: number,
  Description: string,
  Image: string
}

export interface GiftModel extends Model<GiftAttributes>, GiftAttributes{};
export class Gift extends Model<GiftModel, GiftAttributes>{};

export type GiftStatic = typeof Model & {
  new (values?: object, options?: BuildOptions): GiftModel;
};

export function GiftFactory (sequelize: Sequelize): GiftStatic{
  return <GiftStatic>sequelize.define("gift", {
    Id: {
      type: DataTypes.UUIDV4,
      primaryKey: true
    },
    Name: {
      type: DataTypes.STRING
    },
    Quantity: {
      type: DataTypes.NUMBER
    },
    Description: {
      type: DataTypes.STRING
    },
    Image: {
      type: DataTypes.STRING
    }
  }, {timestamps: false, freezeTableName: true});
}
```

Slika 3.11. Datoteka GiftModel.ts i prikaz definiranja sadržaja tablice Gift.

3.3.4. *Service* direktorij

U Service direktoriju nalaze se datoteke kojima je glavna zadaća definirati logiku metoda i odrediti što one moraju vratiti kao rezultat. Pomoću servisa metoda poziva sequelize metodu te vraća njen rezultat. Uzmimo primjer Get(id) metodu - ona poziva db.Gift.findByPk(id) i vraća njen rezultat. Kada su svi servisi izrađeni potrebno ih je samo instancirati i moguće ih je koristiti. Upotrebom servisa kôd postaje puno čitljiviji i lakše je implementirati metode kada svaka metoda radi točno određeni zadatak. Ovdje se osigurava da funkcija dobije tražene podatke te vrati rezultat.

```
import { dbGift } from "../models";
import { Gift, GiftModel, GiftAttributes } from "../models/GiftModel";

export default class GiftService{
  GetAllGifts = (): Promise<GiftModel[]>=>{
    console.dir(Gift)
    return dbGift.findAll();
  };

  GetGiftById = (id: string): Promise<GiftModel | null>=> {
    return dbGift.findByPk(id);
  };

  InsertGift = (gift: GiftAttributes): Promise<GiftModel> => {
    return dbGift.create(gift);
  };

  UpdateGift = (id: string, gift: GiftAttributes): Promise<[number, GiftModel[]]> => {
    return dbGift.update(gift, {where: {Id: id}});
  };

  DeleteGift = (id: string): Promise<number> => {
    return dbGift.destroy({where:{Id: id}});
  };
}
```

Slika 3.12. Datoteka GiftService.ts i prikaz definiranja logike metoda.

3.3.5. *Controllers* direktorij

Za razliku od servisa, kontroleri ne sadrži nikakvu logiku. Pri pisanju kontrolera poželjno je izbjegavati bilo kakvu logiku. Oni služe samo kako bi pozivali metode iz servisa i vraćali rezultat na klijent s određenim statusnim kôdom. Za pisanje ovih kontrolera korištene su tzv. asinkrone funkcije. Asinkrona funkcija je deklarirana ključnom riječju „async“. Ključne riječi „async“ i „await“ omogućavaju funkcijama asinkrono ponašanje i ponašanje na temelju obećanja (engl. *promise-based*). Takve funkcije napisane su u puno čitljivijem stilu čime se izbjegava eksplicitno pisanje i konfiguriranje lanaca obećanja (engl. *promise chains*). Cijeli kontroler se izvozi te se u konačnici koristi pri definiranju ruta. Prikaz kontrolera prikazan je na slikama 3.13. i 3.14. gdje se

pobliže može vidjeti kako on izgleda te od čega se sastoji. Također na tim slikama prikazane su funkcionalnosti registracije i prijave korisnika. Korišteni su i nizovi provjera kako se ne bi dogodilo da se korisnik s istim korisničkim imenom mogao registrirati u bazu podataka, kako se korisnik ne bi mogao prijaviti s pogrešnom lozinkom i sl.

```
UserSignUp = async(req: Request, res: Response) => {
  const userId = uuidv4();
  const saltRounds = 10;
  console.log(req.body.Username);
  const hashPassword = await bcrypt.hash(req.body.Password, saltRounds);

  const usernameExists = await this.service.GetUserByUsername(
    req.body.Username
  );
  const emailExists = await this.service.GetUserByEmail(
    req.body.Email
  );

  const user ={
    Id: userId,
    FirstName: req.body.FirstName,
    LastName: req.body.LastName,
    Email: req.body.Email,
    Username: req.body.Username,
    Password: hashPassword,
    Admin: req.body.Admin
  }

  if (usernameExists || emailExists){
    res.status(409).json({
      message: "Username or email already exists.",
    });
  }else{
    try{
      const result = await this.service.CreateUser(user);
    res.status(200).json({
      message: "User ${user.Username} created succesfully",
      user: {
        id: result.Id,
        username: result.Username,
        email: result.Email
      },
    });
    }catch(error){
      res.sendStatus(500);
    }
  }
};
```

Slika 3.13. Prikaz registracije korisnika.

```
UserLogin = async(req: Request, res: Response) => {
  const body = req.body;

  const userLogin = await this.service.GetUserByUsername(body.Username);

  if(userLogin){
    const correctPassword = null ? false : await bcrypt.compare(body.password, userLogin.Password);
    if(!correctPassword){
      res.status(401).json({error: "Invalid password"});
    }else{
      const userToken = {
        id: userLogin.Id,
        username: userLogin.Username
      };
      const accessToken = jwt.sign(userToken, ACCESS_TOKEN_SECRET, {
        expiresIn: "2h",
      });
      res.status(200).send({
        message: "${userLogin.Username} succesfully logged in",
        user:{
          id: userLogin.Id,
          username: userLogin.Username
        }, accessToken,
      });
    }
  }else{
    res.status(401).json({error: "Invalid username."});
  }
```

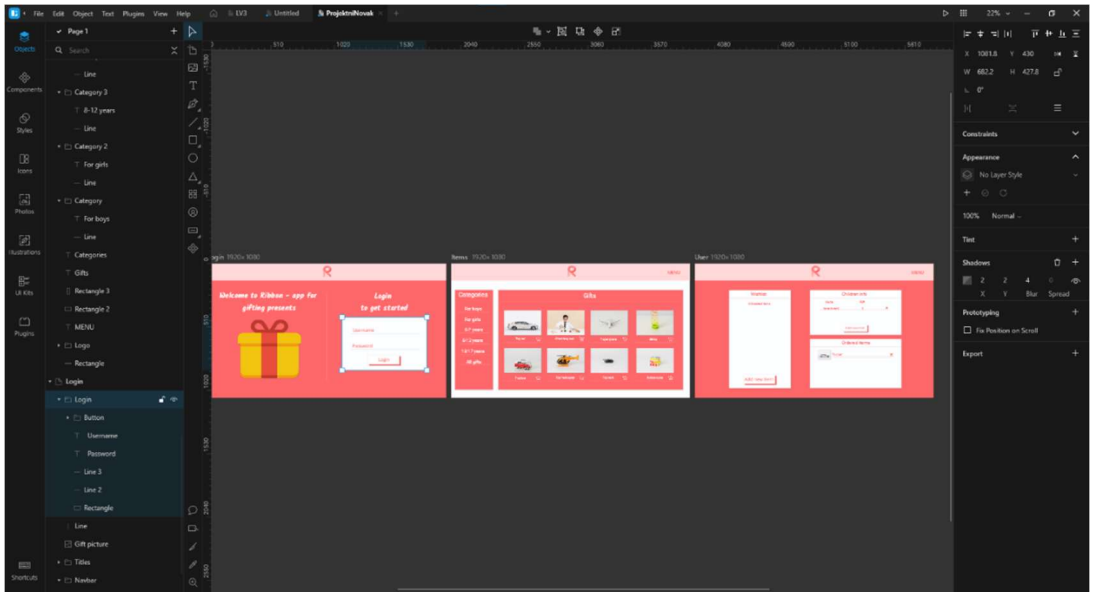
3.14. Prikaz provjere pri prijavi korisnika.

4. DIZAJN SUČELJA WEB APLIKACIJE

Sama aplikacija se sadrži od tri stranice za korisnika i jedne stranice koju vidi samo administrator. Stranice koje vidi korisnik su osmišljene tako da budu što jednostavnije te da korisnik može u vrlo kratkom vremenu pretražiti poklone i odabrati ih. Administratorova zadaća je unositi poklone, registrirati korisnike te kontrolirati narudžbe. Pri registraciji korisnika zadatak mu je generirati korisničke podatke preko kojih će se sam korisnik moći prijaviti u aplikaciju kako bi pristupio odabiru poklona. Na ovaj način se izbjegava zlouporaba ovlasti pri generiranju podataka jer bi korisnik mogao izraditi više korisničkih računa i odabrati više poklona što bi moglo oštetiti stranu koja osigurava poklone za odabir. Korisnik kada pristupi web aplikaciji najprije se mora prijaviti korisničkim podacima koje je dobio od administratora. Jedna od napomena je kako se i korisnik i administrator imaju mogućnost prijaviti preko istog sučelja međutim administrator u bazi podataka ima oznaku da je "admin" te kada se prijavi s tim podacima preusmjerava ga se na administratorovo sučelje.

4.1. Lunacy i Gimp

Nakon izrade pozadinskog sustava započet je rad i na samom dizajnu tj. izradi sučelja web aplikacije. Pri izradi dizajna korišten je program pod nazivom Lunacy. To je moćni besplatni alat baziran na vektorskom dizajnu grafika koji se koristi na operativnom sustavu Windows. Ima mnoštvo mogućnosti kao što su: odabir veličine zaslona, odabir pozadine, upotreba različitih fontova, upotreba već ugrađenih ilustracija ili ikona i sl. Na slici 4.1. prikazano je sučelje programa te prikaz izrađenih stranica ove web aplikacije koje će u daljnjem tekstu biti detaljnije opisane.



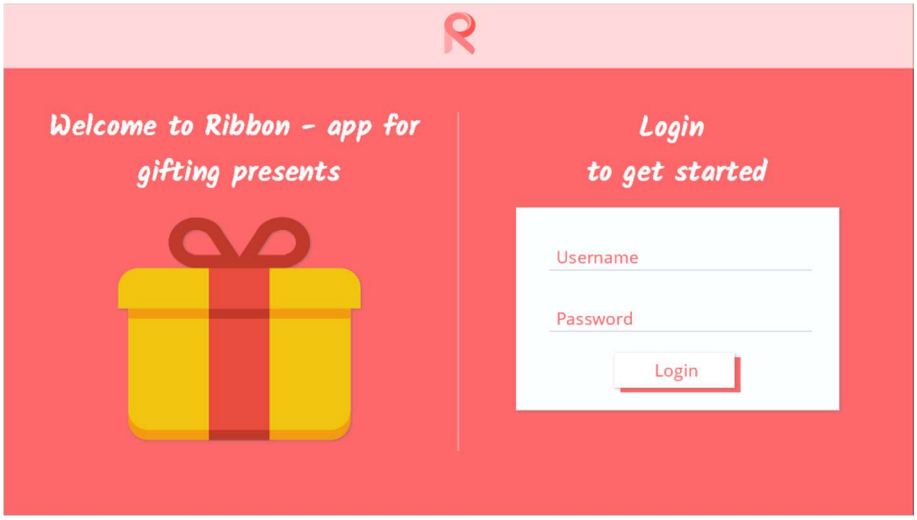
Slika 4.1. Sučelje programa za izradu dizajna Lunacy.

Osim Lunacy-a pri dizajnu je korišten i program Gimp za izradu logo-a aplikacije (Slika 4.2.) [12]. Gimp je također besplatan program za izrađivanje rasterske grafike. Alati korišteni u programu uključuju: odabir boja, filtere, kistove za odabir ili bojanje, transformacijsku selekciju objekata, odabir i stvaranje slojeva slike kao i alate za maskiranje.

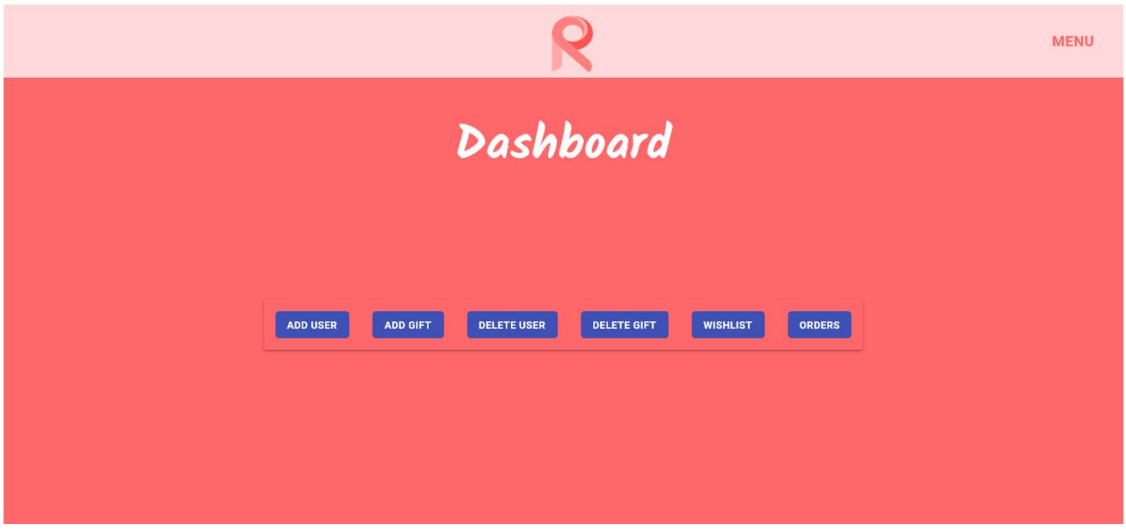
Slika 4.2. Logo web aplikacije.

4.2. Prijava korisnika i administratora

Na stranici za prijavu (Slika 4.3.) nalazi se samo jedna forma koja služi za prijavu korisnika. Ova forma, iako je samo za prijavu, ima dvostruku funkcionalnost. Naime, administrator u bazi podataka ima oznaku „admin“ koja pomaže pri usmjeravanju korisnika nakon prijave. Ako se u formu unese korisničko ime i lozinka koja odgovara administratorovim podacima tada će se on usmjeriti na administratorovu nadzornu ploču (engl. *dashboard*, Slika 4.4.). Administrator ima razne mogućnosti rada nad bazom podataka kao što su: registriranje korisnika, unos poklona, prikaz narudžbi i sl. U drugom slučaju ako podaci odgovaraju „uobičajenom“ korisniku koji nema administratorska prava tada će se on usmjeriti na stranicu za odabir poklona s koje će također moći pristupiti i svojim podacima.



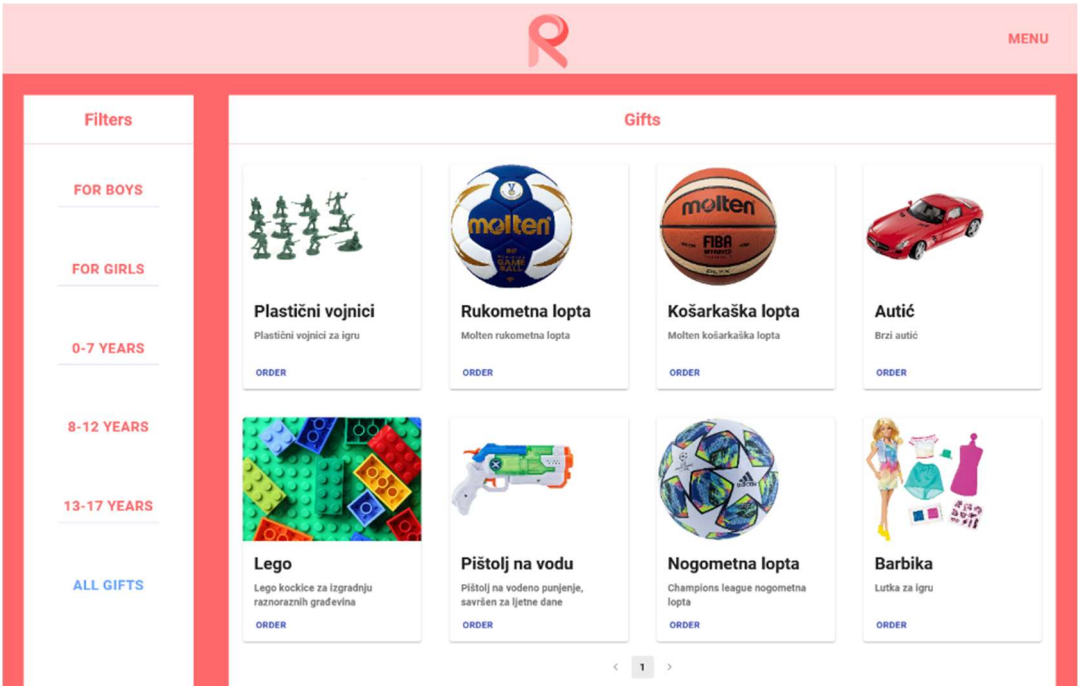
Slika 4.3. Prikaz stranice za prijavu korisnika.



Slika 4.4. Prikaz administratorovog sučelja.

4.3. Stranica za odabir poklona

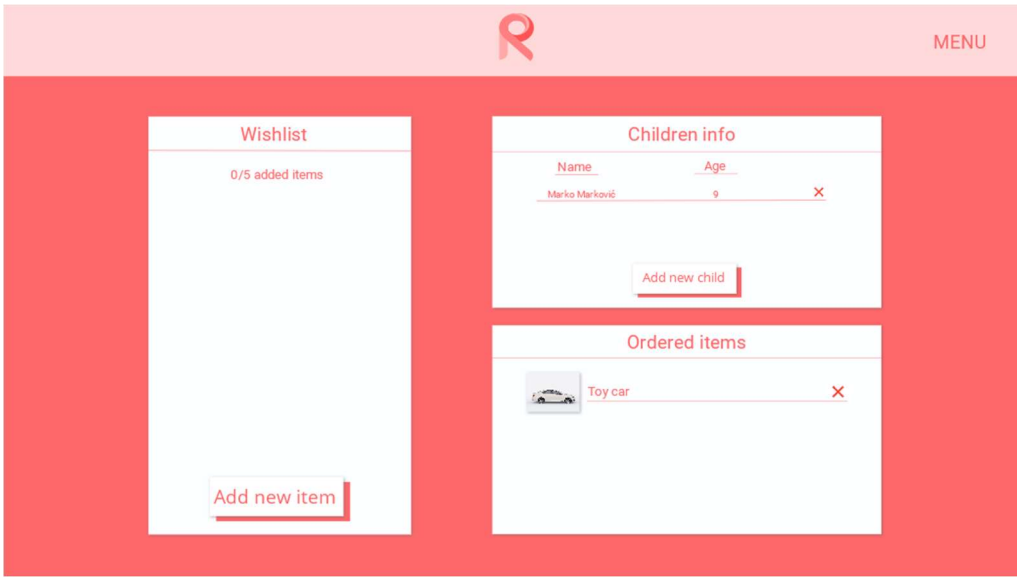
Nakon prijave „uobičajenog“ korisnika prikazat će se stranica za odabir poklona (Slika 4.5.). Na ovoj stranici korisnik ima mogućnost odabira samih poklona koje prethodno može kategorizirati po kategorijama koje se nalaze na lijevoj strani same stranice. Glavni i najveći dio stranice zauzimaju pokloni koje korisnik može odabrati. Pokloni su prikazani u karticama koje se sastoje od: slike, imena, opisa i tipke za narudžbu. Također prikaz poklona po stranici ograničen je na osam, ali ako je potrebno prikazati više poklona implementirana je paginacija. U gornjem desnom uglu nalazi se tipka „MENU“ koja omogućuje korisniku da se odjavi iz aplikacije ili da prijeđe na stranicu sa svojim podacima.



Slika 4.5. Prikaz stranice za odabir poklona.

4.4. Stranica sa osobnim podacima

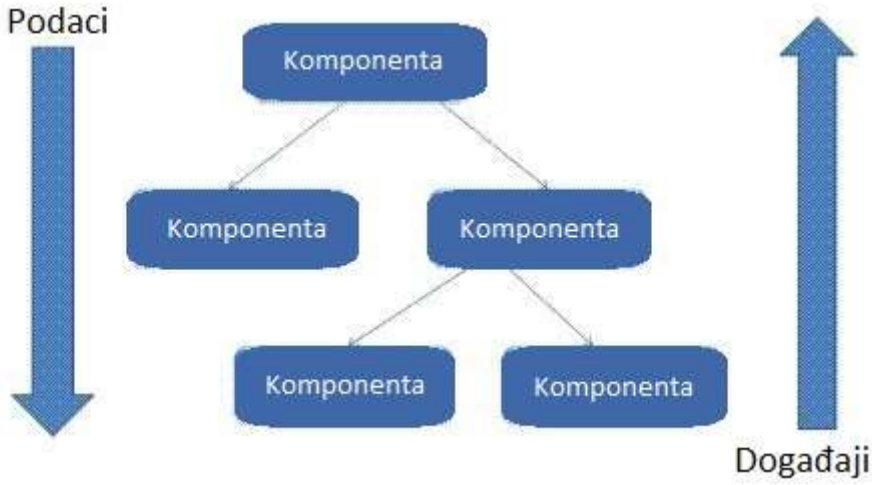
Kada korisnik pristupi stranici s osobnim podacima (Slika 4.6.) prikazane su mu tri komponente: popis želja (engl. *wishlist*), informacije o djeci te informacije o narudžbama. Korisnik ima mogućnost dodavanja maksimalno pet želja na svoj popis koji će biti prikazani na popisu i administratoru. Osim dodavanja želja korisnik ima mogućnost dodavanja imena i godina djeteta te će taj broj dodane djece određivati koliko poklona korisnik može naručiti. Nakon što korisnik unese svu svoju djecu te naruči poklon, korisnik će imati pregled narudžbi na popisu naručenih poklona. Kao na stranici za odabir poklona, i na ovoj stranici postoji gumb „MENU“ pomoću kojeg se korisnik može odjaviti ili se preusmjeriti na stranicu za odabir poklona. Na taj način ostvaruje se povezanost među stranica.



Slika 4.6. Prikaz stranice za prijavu korisnika.

5. FUNKCIONALNOSTI SUČELJA WEB APLIKACIJE ZA DODJELU POKLONA

React je JavaScript biblioteka koja se koristi za izradu korisničkih sučelja. Jedna od glavnih karakteristika React-a je da radi s komponentama koje je moguće ponovno iskorištavati [13]. Sama svrha je da bude brz, skalabilan i jednostavan. Koristi se isključivo za izradu korisničkih sučelja te zahtjeva prethodno izrađen pozadinski sustav kako bi mogao primijeniti sve funkcionalnosti koje web aplikacija zahtjeva. React koristi jednosmjerni protok podataka što znači da se set nepromjenljivih vrijednosti prosljeđuje *renderer-u* komponenti kao svojstvo HTML oznaka. Komponente ne mogu izravno mijenjati svojstva, ali mogu prosljeđivati funkcije povratnih poziva uz čiju pomoć je moguće raditi izmjene. Ovakav proces se naziva „properties flow down; actions flow up“ (Slika 5.1.).



Slika 5.1. Jednosmjerni tok podataka.

React osim ovakvog toka podataka koristi i „*Virtual Document Object Model*“ ili skraćeno „*Virtual DOM“*. To znači da React stvara predmemorijsku strukturu podataka (engl. „*in-memory data structure cache*“) koja proračunava izvršene promjene, a zatim ažurira preglednik s novim i „svježim“ informacijama.

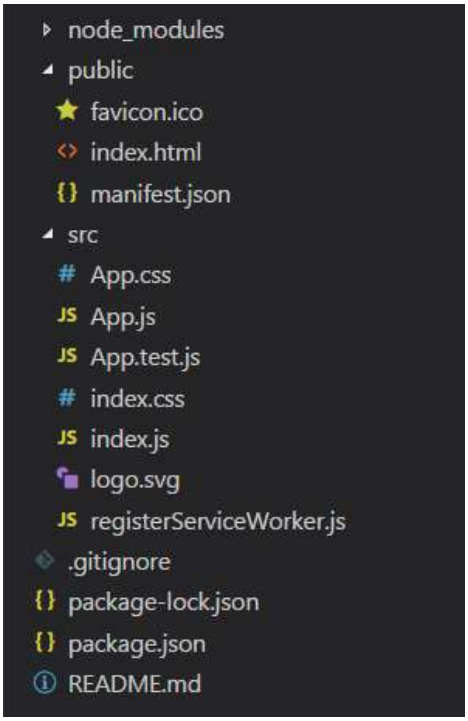
5.1. Glavne karakteristike React JS-a

Karakteristike: [14]

1. Jednostavnost
    - Pristup temeljen na komponentama, dobro definiran životni ciklus i upotreba jednostavnog JavaScript-a čine React vrlo lakim za učenje, izradu web aplikacije te njihovo održavanje. React koristi posebnu sintaksu pod nazivom JSX koja omogućuje miješanje HTML-a s JavaScript-om, ali to nije uvjet. Programer ima mogućnost odabira kojim načinom želi pisati kôd.
2. Lagan za učenje
    - Svatko s osnovnim predznanjem u programiranju vrlo lako može razumjeti React
    - Kako bi počeli programirati u React.js-u dovoljno je osnovno predznanje CSS-a i HTML-a.
3. Kompatibilnost
    - React je vjeran korištenju komponenti kao posebnih dijelova za izradu sučelja te se stoga može koristiti za izradu IOS, Android i web aplikacija.
4. Vezivanje podataka
    - Koristi jednosmjerno vezivanje podataka, a aplikacijska arhitektura pod nazivom Flux kontrolira protok podataka prema komponentama kroz kontrolnu točku. Ovo svojstvo čini laganim otklanjanje pogrešaka u samostalnim komponentama velikih React JS aplikacija.
5. Testabilnost
    - Aplikacije koje koriste React vrlo je jednostavno testirati. React pogledi (engl. „*Views*“) mogu se tretirati kao funkcije stanja, tako da je moguće manipulirati stanjem koje prelazi na React JS pogled te potom pratiti izlaz i pokrenute radnje, događaje, funkcije itd.

5.2. Stvaranje React aplikacije

Stvaranje React aplikacije i njezinog razvojnog okruženja vrlo je jednostavno. Koristeći svojstvo „*Create React App*“ vrlo lako je postaviti samo razvojno okruženje (Slika 5.2.) bez potrebe za samostalnim stvaranjem početnih direktorija, datoteka što znači da nije potrebno samostalno stvaranje početne strukture projekta [14]. Preduvjet za stvaranje React aplikacije jest imati instaliranu određenu verziju Node paketa na računalu jer se za stvaranje koristi NPM (poglavlje 5.2.). Za stvaranje aplikacije koristi se jednostavna naredba koja se pokreće iz terminala: „*npx create-react-app my-app*“. Nakon toga potrebno je pokrenuti aplikaciju upotrebom naredbe: „*npm start*“ te se tada aplikacija učita i automatski pokrene u zadanom internetskom pregledniku. Važno je napomenuti kako *Create React App* ne obrađuje pozadinsku logiku aplikacije ili bazu podataka, on samo stvara „cjevovod“ za stvaranje korisničkog sučelja (engl. *frontend build pipeline*). Takav način stvaranja aplikacije omogućava upotrebu bilo kojeg pozadinskog sustava koji se želi primijeniti.



Slika 5.2. Struktura nakon pokretanja „Create React App“.

5.2.1. Axios

Za komunikaciju s pozadinskim sustavu korišten je paket *Axios* koji se također instalira pomoću NPM-a. On je HTTP klijentska biblioteka koja omogućuje slanje zahtjeva na određenu odredišnu točku. Kada se zahtjev pošalje očekuje se povratna operacija koja odgovara zahtjevu koji je poslan. Na primjer ako je poslan „*GET*“ zahtjev tada se očekuje dobiti podatke koji će biti prikazani na aplikaciji.

Funkcionalnosti Axios-a: [15]

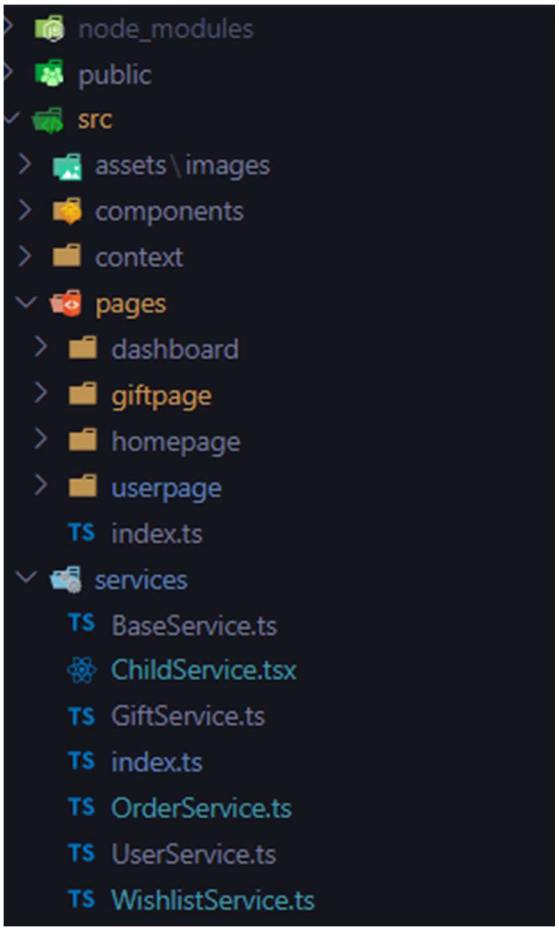
1. Ima vrlo dobre zadane postavke za rad s JSON podacima (format podataka koji se koristi pri komunikaciji između pozadinskog sustava i sučelja)
2. Ima nazive funkcija koje odgovaraju svim HTTP metodama. Npr. za izvođenje „*GET*“ zahtjeva koristi se metoda „ *.get()* “.
3. Čini više s manje kôda. Za pristup traženim JSON podacima potreban je samo jedan poziv „*.then()*“ funkcije.
4. Ima vrlo dobro rješavanje pogrešaka. Axios šalje kôd greški u rasponu od 400 do 500.
5. Može se koristiti na poslužiteljskoj strani kao i na klijentskoj.

5.2.2. Material UI

Za izradu pojedinih komponenti korišten je paket pod nazivom Material UI. Taj paket, koji se instalira pomoću NPM-a, omogućava korištenje unaprijed izrađenih komponenti. Svaka komponenta nakon što se ugradi u kod pomoću ključne riječi „*import*“ moguće ju je koristiti te mijenjati joj izgled.

5.3. Programski kôd i glavne komponente web aplikacije

Na slici 5.3. prikazana je struktura direktorija koja je korištena pri izradi svih funkcionalnosti korisničkog sučelja. Ovakav pristup omogućava bolju preglednost strukture aplikacije te olakšava pristup svim datotekama. U ovom potpoglavlju bit će objašnjen pojedini direktorij te njegove najbitnije komponente.



Slika 5.3. Struktura direktorija.

5.3.1. *Components* direktorij

U ovom direktoriju se nalazi temelj aplikacije. Glavna datoteka je App.tsx koja se koristi za definiranje glavnih ruta aplikacije koje će se koristiti i prikazivati pri upotrebi aplikacije (Slika 5.4.). Također važno je napomenuti kako se pri definiranju ruta potrebno pridržavati imenovanja kao što je bio slučaj pri izradi pozadinskog sustava. Tako se osigurava ispravna komunikacija između pozadinskog sustava i sučelja kada se budu pozivale razne funkcionalnosti.

```
const AppRouting = ({ children }: any) => (
  <Router>
    {children}
    <Switch>
      <PrivateRoute exact path="/dashboard" component={Dashboard} />
      <PrivateRoute exact path="/user" component={UserPage} />
      <PrivateRoute exact path="/gifts" component={GiftPage} />
      <Route exact path="/" component={HomepagePage} />
    </Switch>
  </Router>
);
```

Slika 5.4. Definiranje glavnih ruta za prikaz svih stranica.

Može se reći kako je ova datoteka početna točka aplikacije jer se pomoću nje definira ponašanje aplikacije te njezin tijek. Osim ove datoteke u ovom direktoriju se nalazi i Header.tsx datoteka koja služi za stvaranje zaglavlje stranice (Slika 5.5.) koje će se prikazivati na svakoj stranici te se time izbjegava ponavljanje programskog kôda. Ovakav način stvaranja komponente omogućava nam da iskoristimo komponentu upotrebom HTML oznake: „<Header />“ na bilo kojoj stranici aplikacije. Ovakav pristup se koristi za kreiranje bilo koje komponente na stranici te se ona može reciklirati bilo gdje unutar aplikacije. Također moguće je odrediti različiti izgled zaglavlja kako bi bio prilagođen stranici na kojoj se on prikazuje (Slika 5.6.).

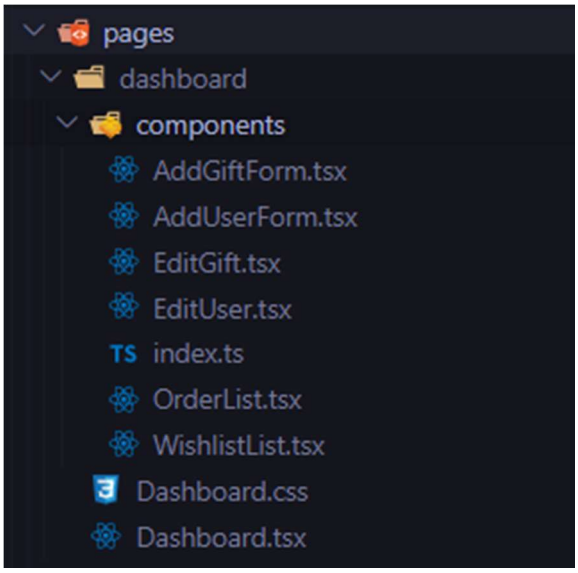
Slika 5.5. Izgled zaglavlja na stranici sa poklonima.

```
renderGiftsHeader = () => (
  <div className="menu">
    <div className="menu_item">
      <div ref={this.myRef}>
        <Button
          aria-controls="simple-menu"
          aria-haspopup="true"
          onClick={this.handleClick}
        >
          Menu
        </Button>
        <Menu
          getContentAnchorEl={null}
          id="simple-menu"
          anchorEl={this.myRef.current}
          keepMounted
          open={this.state.menuOpen}
          onClose={this.handleClose}
        >
          <MenuItem onClick={this.handleClose}>
            <Link to="/user">My account</Link>
          </MenuItem>
          <MenuItem onClick={this.handleClose}>Logout</MenuItem>
        </Menu>
      </div>
    </div>
  </div>
);
```

Slika 5.6. Primjer zaglavlja za stranicu sa poklonima.

5.3.2. *Pages* direktorij

U ovom direktoriju nalaze se sve datoteke koje služe za prikaz stranica na web aplikaciji. Ova aplikacija sadrži se od četiri stranice: stranica s poklonima, stranica s osobnim podacima, stranica za prijavu te stranica za administratora. Svaki od pojedinih direktorija sadrži svoju *.tsx* datoteku u kojoj su definirane sve komponente te *.css* datoteku koja služi za definiranje izgleda i poziciju komponenti. Također osim tih datoteka, ako je stranica kompleksna, koristi se i direktorij koji sadrži sve komponente koje se nalaze na stranici kako kôd ne bi bio nepregledan i predugačak (Slika 5.7.).



Slika 5.7. Primjer strukture datoteka za administratorsku ploču.

Za primjer uzeta je stranica administratorske ploče te registracija novog korisnika. Kada administrator stisne gumb za registraciju pojavljuje mu se skočni prozor koji omogućava unos svih potrebnih podataka za korisnika kojeg se želi registrirati. Takav skočni prozor zapravo predstavlja komponentu koja je izrađena u datoteci „AddUserForm“ (Slika 5.8.). Osim izrade komponente potrebno je odrediti glavne varijable te funkcionalnosti s kojima će se povezivati pri pozivanju određenih ruta s pozadinskog sustava (Slika 5.9.).

```
render() {
  return (
    <form onSubmit={(e) => this.onSubmit(e)}>
      <h2 id="Form">Add gift</h2>
      <div>
        <TextField
          required
          label="Gift Name"
          value={this.state.name}
          inputProps={{ maxLength: 50 }}
          onChange={(e) => this.setName(e.target.value)}
        />
      </div>
      <div>
        <TextField
          required
          className="number_input"
          type="number"
          label="Gift Quantity"
          value={this.state.quantity}
          onChange={(e: any) => this.setQuantity(e.target.value)}
        />
      </div>
      <div>
        <TextField
          required
          multiline
          maxRows={10}
          label="Gift Description"
          value={this.state.description}
          inputProps={{ maxLength: 150 }}
          onChange={(e) => this.setDescription(e.target.value)}
        />
      </div>
      <div>
        <TextField
          required
          multiline
          maxRows={10}
          label="Gift Image"
          value={this.state.image}
          onChange={(e) => this.setImage(e.target.value)}
        />
    <div>
      <TextField
        required
        id="standard-select-currency-native"
        select
        label="Gift gender"
        value={this.state.gender}
        onChange={(e: any) => this.setGender(e.target.value)}
        SelectProps={{
          native: true,
        }}
        helperText="Please select gender"
      >
        <option value={0}>For boys</option>
        <option value={1}>For girls</option>
      </TextField>
    </div>
    <div>
      <TextField
        required
        className="number_input"
        type="number"
        label="For age"
        value={this.state.age}
        onChange={(e: any) => this.setAge(e.target.value)}
      />
    </div>
    <div className="submit-button">
      <Button type="submit" variant="contained" color="primary">
        Submit
      </Button>
    </div>
  </form>
```

5.8. Sastavni dijelovi komponente (forme) za registraciju korisnika.

```
class AddUserForm extends React.Component<{}, AddUserFormState> {
  static contextType = TokenContext;
  private userService: UserService;
  userContext: any;
  setUserContext: any;
  constructor(props: any) {
    super(props);
    this.state = {
      firstName: "",
      lastName: "",
      mail: "",
      username: "",
      password: "",
      admin: false,
      token: "",
    };
    this.userService = new UserService();
  }

  clearState = () => this.setState({
      firstName: "",
      lastName: "",
      mail: "",
      username: "",
      password: "",
      admin: false,
      token: "",
  })

  componentDidMount() {
    const [context, setContext] = this.context;
    this.userContext = context;
    this.setUserContext = setContext;
    this.setState({
      token: this.userContext?.token,
    });
  }

setFirstName = (firstName: string) => this.setState({ firstName });
setLastName = (lastName: string) => this.setState({ lastName });
setMail = (mail: string) => this.setState({ mail });
setUsername = (username: string) => this.setState({ username });
setPassword = (password: string) => this.setState({ password });
setAdmin = (admin: boolean) => this.setState({ admin });


onSubmit = async (e: React.FormEvent<HTMLFormElement>) => {
  e.preventDefault();

  const { token, ...dataObject } = this.state;
  const bodyData = {
    FirstName: dataObject.firstName,
    LastName: dataObject.lastName,
    Email: dataObject.mail,
    Username: dataObject.username,
    Password: dataObject.password,
    Admin: dataObject.admin
  }

  try {
    await this.userService.addUser(bodyData);
    this.clearState();
    alert("User has been successfully created.");
  } catch (err: any) {
    if (err?.response?.status === 409 || err?.response?.status === 204) {
      alert(err?.response?.data);
    }
```

5.9. Prikaz varijabli i funkcionalnosti forme za registraciju korisnika.

Osim „dashboard“ direktorija ovdje se nalaze i ostali direktoriji koji predstavljaju stranice koje se izrađuju. Pa tako postoje direktoriji: giftpage, homepage te userpage. Svaki od ovih direktorija ima datoteke u kojima se nalaze komponente te funkcionalnosti kojima se povezuje sučelje za pozadinskim sustavom, slično kao za administratorsku ploču koja je objašnjena gore u tekstu.

5.3.3. *Services* direktorij

Servisi za izradu sučelja predstavljaju povezanost s pozadinskim sustavom. Pomoću servisa moguće je pristupiti svim metodama koje su implementirane kao što je registracija korisnika, unos poklona, uređivanje korisnikovih podataka i slično. Pojedini servis, za komunikaciju s pozadinskim sustavom, koristi rute koje su predefinirane u samom pozadinskom sustavu (Slika 5.11.). Kao glavni servis koristi se datoteka pod nazivom „BaseService.ts“ koja ima definirane metode kojima se čitaju, pišu, mijenjaju ili brišu podaci iz baze podataka (Slika 5.10.). Za pisanje ostalih servisa prvo je potrebno uvesti glavni servis kako bi se mogle koristiti potrebne rute ovisno o onome što se želi izvesti (čitanje, pisanje, promjena ili brisanje).

```
import axios from "axios";

class BaseService {
  public url: URL;
  constructor(private route: string) {
    this.url = new URL(`https://ribbon-app.herokuapp.com/${this.route}`);
  }

  public getAll = (params: any, token: string): Promise<any> => {
    const url = this.url;
    url.search = new URLSearchParams(params).toString();
    return axios.get(url.toString(), {
      headers: {
        "Content-Type": "application/json",
        Authorization: `BEARER ${token}`,
      },
    });
  };

  public getSingle = (id: string, params: any, token: string): Promise<any> => {
    const url = new URL(this.url.toString() + `/${id}`);
    url.search = new URLSearchParams(params).toString();
    return axios.get(url.toString(), {
      headers: {
        "Content-Type": "application/json",
        Authorization: `BEARER ${token}`,
      },
    });
  };

  public create = (body: any, token: string): Promise<any> => {
    const url = this.url;
    return axios.post(url.toString(), body, {
      headers: {
        "Content-Type": "application/json",
        Authorization: `BEARER ${token}`,
      },
    });
  };
```

5.10. Prikaz definiranja glavnih ruta.

```
import { BaseService } from "./";
import axios from "axios";

class GiftService extends BaseService {
  constructor() {
    super("gifts");
  }

  addGift = async (body: any, token: string): Promise<any> => {
    const url = `${this.url.toString()}/`;
    try {
      console.log(url, body);
      return await axios.post(url, body, {
        headers: {
          "Content-Type": "application/json",
          Authorization: `BEARER ${token}`,
        },
      });
    } catch (err) {
      console.error(err);
    }
  };
}

export default GiftService;
```

5.11. Rute za pristupanje funkcionalnostima pri radu s poklonima (dodavanje poklona).

6. ZAKLJUČAK

U ovom završnom radu realizirana je web aplikacija za dodjelu poklona koja je zahtijevala nekoliko glavnih funkcionalnosti. Osim funkcionalnosti postoje dvije glavne uloge, administrator i korisnik, koje su ostvarene kroz provjeru podataka pri unosu forme za prijavu.

Sama aplikacija je sastavljena od dva dijela: pozadinskog sustava i korisničkog sučelja. Ta dva glavna dijela aplikacije izvedena su pomoću dvije JavaScript tehnologije, Node.js i React.js. Osim implementacije svih funkcionalnosti bilo je potrebno i dizajnirati korisničko sučelje kako bi ono olakšalo korištenje, a samim time bilo je potrebno steći potrebna znanja u radu s programima za dizajn sučelja. Sve tehnologije navedene u radu nisu ograničene samo na izradu ovakvih ili sličnih web aplikacija već je moguće izraditi mnoštvo kompleksnih web aplikacija koje imaju puno veći broj funkcionalnosti. Sama web aplikacija za dodjelu poklona vrlo je jednostavna te se može zaključiti kako ovakva web aplikacija znatno olakšava i pojednostavljuje odabir poklona što je u krajnjem slučaju bio cilj.

Kroz ovaj rad dokazano je kako sve odabrane tehnologije mogu funkcionirati zajedno i činiti cjelovitu web aplikaciju. Također, rad je rezultirao uspješno implementiranim odabranim tehnologijama, ali može služiti kao temelj za dodavanje novih funkcionalnosti i korištenje u velikim sustavima gdje se želi pojednostaviti dodjela poklona.

LITERATURA

[1] BabyCenter web trgovina, dostupno na: https://www.babycenter.hr/

[2] abrakadabra web trgovina, dostupno na: https://www.abrakadabra.com/hr-HR/

[3] IgračkeShop web trgovina, dostupno na: https://www.babycenter.hr/

[4] Wikipedia, JavaScript, dostupno na: https://hr.wikipedia.org/wiki/JavaScript

[5] Springboard Blog, The History of JavaScript: Everything You Need to Know, 19.08.2019., dostupno na: https://www.springboard.com/blog/data-science/history-of-javascript/

[6] Octoverse, Top languager over the years, 2020.., dostupno na: https://octoverse.github.com/

[7] Wikipedia, Node.js, 01.07.2015., dostupno na: https://en.wikipedia.org/wiki/Node.js

[8] Popwebdesing, Šta je Node.js?, 12.09.2021., dostupno na: https://www.popwebdesign.net/popart_blog/2015/06/sta-je-node-js/

[9] Webprogramiranje, npm & yarn osnove, dostupno na: https://www.webprogramiranje.org/npm-yarn-osnove/

[10] Wikipedia, Express.js, 29.07.2021., dostupno na: https://en.wikipedia.org/wiki/Express.js

[11] Wikipedia, MySQL, 31.03.2021., dostupno na: https://hr.wikipedia.org/wiki/MySQL

[12] Wikipedia, GIMP, 01.09.2021., dostupno na: https://en.wikipedia.org/wiki/GIMP

[13] N.I., What And Why React.js, C# Corner, 2021., dostupno na: https://www.c-sharpcorner.com/article/what-and-why-reactjs/

[14] React, Tutorial: Intro to React, 2021., dostupno na: https://reactjs.org/tutorial/tutorial.html

[15] R.B., How To Use Axios With React: The Definitive Guide, 2021., dostupno na: https://www.freecodecamp.org/news/how-to-use-axios-with-react/

SAŽETAK

Web aplikacija za dodjelu poklona

Web aplikacija za dodjelu poklona za svrhu je imala olakšati komunikaciju između administratora i korisnika pri dodjeli poklona u velikim tvrtkama ili sličnim sustavima. Administrator ima mogućnost registrirati korisnike te unositi poklone koje će korisnik moći odabrati. Osim unosa i registracije, administrator može uređivati podatke o korisniku ili poklonima. Također, ima pregled svih narudžbi i lista želja koje pojedini korisnik može napraviti. Svi uneseni podaci, od strane administratora ili korisnika, pohranjeni su u digitalnu bazu podataka Korisnik, nakon prijave u aplikaciju, mora unijeti ime i dob djeteta te može odabrati po jedan poklon za pojedino dijete. Korisnik pri odabiru poklona ima filtere po kojima može filtrirati poklone kao što su dob ili spol. Na stranici s osobnim podacima korisnik može vidjeti popis svoje djece, narudžbe koje je napravio te može stvoriti listu želja.

Ključne riječi: administrator, dodjela poklona, korisnik, poklon, web aplikacija

ABSTRACT

Web application for awarding gifts

The purpose of the gift-giving web application was to facilitate communication between administrators and users when giving gifts in large companies or similar systems. The administrator has the ability to register users and enter gifts that the user will be able to choose. In addition to entry and registration, the administrator can edit user data or gifts. It also has an overview of all orders and wishlists that an individual user can make. All entered data, by the administrator or user, are stored in a digital database. The user, after logging in to the application, must enter the name and age of the child and can choose one gift for each child. When choosing a gift, the user has filters by which he can filter gifts such as age or gender. On the personal information page, the user can see a list of their children, the orders they have made and can create a wishlist.

Keywords: administrator, gift, gift giving, user, web application,

ŽIVOTOPIS

Autor ovog završnog rada, David Novak, student je stručnog studija Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Rođen je u Osijeku 1999. godine. Završio je Elektrotehničku i prometnu školu Osijek u Osijeku te ima zvanje Tehničar za računalstvo. U trenutku pisanja ovog rada zaposlen je u tvrtki Samurai Digital u kojoj ima poziciju Service Delivery Manager-a.

_____

Potpis autora