Razvoj i implementacija veb aplikacije za kupovinu i prodaju kolekcionarskih modela

Ognjen Arsenijević, dr Nemanja Radosavljević

***Sadržaj* — Rad prikazuje razvoj i implementaciju veb aplikacije za kupovinu i prodaju kolekcionarskih modela. Obuhvaćeni su uvod i definisanje problema, analiza tržišta, opis korišćenih tehnologija, specifikacija sistema sa funkcionalnim i nefunkcionalnim zahtevima, arhitektura sistema, model podataka, sigurnosni mehanizmi, implementacija klijentske i serverske strane, prikaz korisničkog interfejsa i testiranje sistema.**

***Ključne reči* — autentifikacija, baza podataka, Express, Node.js, REST API, testiranje, kupovina i prodaja, kolekcionarski modeli, razvoj softvera, veb aplikacija, Vue.js**

I. Uvod

CILJ ovog rada je prikaz razvoja i implementacije veb aplikacije za kupovinu i prodaju kolekcionarskih modela, uključujući minijaturne modele automobila, motora, vozova, kamiona, brodova i drugih prevoznih sredstava. Rad prikazuje definisanje problema i analizu postojećih rešenja na tržištu, ukazujući na potrebu za razvojem ovakve aplikacije.

Organizovan je u više poglavlja: prvo je dat pregled funkcionalnih i nefunkcionalnih zahteva sistema, zatim opis arhitekture i modela podataka, prikaz sigurnosnih mehanizama i dizajnerskih odluka, detaljna implementacija klijentske i serverske strane, prezentacija korisničkog interfejsa i testiranje sistema. Ovakvom strukturom je omogućeno sistematsko predstavljanje procesa razvoja i ključnih rešenja koja su

Ognjen Arsenijević, Šabačka 14, 11120 Beograd, Srbija; (telefon: 381-69-1152014; e-mail: ognjen1arsenijevic@gmail.com).

Dr Nemanja Radosavljević, Računarski fakultet, Kneza Mihaila 6, 11102 Beograd, Srbija; (e-mail: nradosavljevic@raf.rs).

implementirana u aplikaciji. Rad se oslanja na tehničku dokumentaciju, standarde iz oblasti softverskog inženjerstva i relevantnu literaturu.

II. MOTIVACIJA I ANALIZA TRŽIŠTA

Pronalaženje i kupovina kolekcionarskih modela često je otežana zbog nepovezanog tržišta i nedostatka centralizovanog sistema za pregled i pretragu ponude. Postoje specijalizovane prodavnice, opšte platforme za oglašavanje, društvene mreže i sajmovi, ali nijedno rešenje ne objedinjuje sve modele i ne pruža efikasnu pretragu.

Specijalizovane prodavnice nude ograničenu ponudu i ne dozvoljavaju drugim prodavcima da postavljaju oglase. Opšte platforme omogućavaju samostalno oglašavanje, ali različiti formati opisa otežavaju preciznu pretragu. Društvene mreže omogućavaju direktnu i fleksibilnu komunikaciju između kupaca i prodavaca, ali ne garantuju standardizaciju podataka niti ažurnost oglasa. Sajmovi kolekcionarskih modela omogućavaju pregled modela uživo, ali se održavaju retko i uglavnom u većim gradovima, što ograničava pristup široj publici.

Veb aplikacija *Kolekcionarski modeli* razvijena je kao specijalizovana platforma koja objedinjuje prednosti postojećih rešenja i omogućava standardizovano oglašavanje, efikasnu pretragu, pouzdanu komunikaciju i pokriva sve faze procesa trgovine, čime predstavlja originalni doprinos razvoju digitalne trgovine kolekcionarskim modelima na domaćem tržištu.

III. ANALIZA I SPECIFIKACIJA SISTEMA

A. Funkcionalni zahtevi

Funkcionalni zahtevi su izjave o uslugama koje sistem treba da pruži, načinu na koji sistem treba da reaguje na određene korisničke akcije, kao i o ponašanju sistema u specifičnim situacijama [1].

Jasno definisani funkcionalni zahtevi predstavljaju neophodan preduslov za uspešan razvoj softverskih rešenja. Oni služe kao osnova za dizajn, implementaciju i testiranje sistema, a istovremeno omogućavaju bolje razumevanje potreba krajnjih korisnika. Funkcionalni zahtevi predstavljaju deo specifikacije softverskih zahteva. Preporuke za formulisanje specifikacije softverskih zahteva sadržane su u IEEE standardu 830-1998, koji ističe da zahtevi treba da budu tačni, nedvosmisleni, potpuni, dosledni, rangirani po važnosti ili stabilnosti, proverljivi, promenljivi, i mogući za praćenje [2].

U kontekstu veb aplikacije *Kolekcionarski modeli* funkcionalni zahtevi definišu osnovne operacije koje sistem mora da podrži kako bi omogućio

efikasno oglašavanje, pretragu i kupovinu kolekcionarskih modela. Ovi zahtevi obuhvataju tri tipa funkcionalnosti: funkcionalnosti dostupne svim korisnicima, funkcionalnosti dostupne kupcima i funkcionalnosti dostupne prodavcima.

B. Nefunkcionalni zahtevi

Nefunkcionalni zahtevi predstavljaju ograničenja usluga ili funkcija koje softverski sistem mora da ispunjava. Obuhvataju vremenska ograničenja, zahteve u vezi sa procesom razvoja, kao i ograničenja nametnuta standardima. Za razliku od funkcionalnih zahteva, nefunkcionalni zahtevi se najčešće odnose na sistem kao celinu, a ne na pojedinačne funkcionalnosti ili usluge [1].

Važno je istaći da granica između funkcionalnih i nefunkcionalnih zahteva nije uvek jasna, jer neki zahtevi koji se na prvi pogled mogu svrstati u nefunkcionalne, kao što su zahtevi vezani za sigurnost, mogu sadržati i funkcionalne elemente, poput autentifikacije korisnika.

Nefunkcionalni zahtevi su ključni za definisanje ukupnog kvaliteta softvera i značajno utiču na njegov dizajn, implementaciju i testiranje. U veb aplikaciji *Kolekcionarski modeli*, glavni nefunkcionalni zahtevi obuhvataju sledeće oblasti:

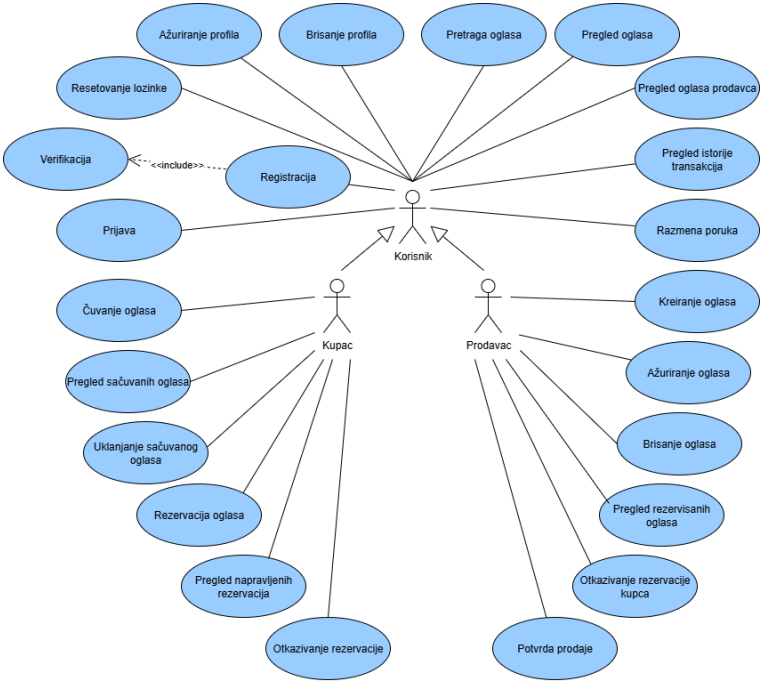
- **Sigurnost**: zaštita lozinki enkripcijom, komunikacija preko HTTPS protokola, verifikacija korisničkog naloga mejlom, sprečavanje duplikata korisničkih imena i mejlova.
- **Performanse**: brzo izvršavanje ključnih korisničkih operacija, uključujući registraciju, prijavu i pretragu oglasa.
- **Pouzdanost i dostupnost**: stabilan rad sistema i očuvanje integriteta podataka.
- **Upotrebljivost**: pregledan i intuitivan interfejs, prilagođen različitim uređajima.
- **Održavanje**: modularna arhitektura koda radi lakšeg proširenja i ispravki.

C. Slučajevi korišćenja

Slučaj korišćenja predstavlja ugovor između zainteresovanih strana sistema o njegovom ponašanju. Slučaj korišćenja opisuje kako se sistem ponaša u različitim uslovima kao odgovor na zahteve primarnog aktera, koji pokreće interakciju sa sistemom radi ostvarivanja određenog cilja. Različite sekvence ponašanja, ili scenariji, mogu se razviti u zavisnosti od zahteva i okolnosti, a slučaj korišćenja objedinjuje te različite scenarije u jedinstveni opis. Slučajevi korišćenja služe kao sredstvo komunikacije između svih

učesnika u razvoju sistema, često predstavljajući zahteve i poslovne procese na razumljiv i jasan način [3].

Za potrebe ove veb aplikacije, slučajevi korišćenja služe kao osnovni alat za analizu i definisanje svih ključnih interakcija korisnika sa sistemom. Njihov pregled predstavljen je dijagramom slučajeva korišćenja (Sl. 1).



Sl. 1. Dijagram slučajeva korišćenja.

IV. KORIŠĆENE TEHNOLOGIJE

A. Klijentska strana

Klijentska strana aplikacije realizovana je korišćenjem modernih JavaScript tehnologija koje omogućavaju interaktivni i responzivni korisnički interfejs. Osnovni okvir je **Vue** [4], koji omogućava modularni razvoj komponenti interfejsa, sa dvostranim vezivanjem podataka, što doprinosi dinamičnoj interakciji korisnika i jednostavnijem održavanju koda. Aplikacija koristi arhitekturu jednostranične veb aplikacije, čime se smanjuje vreme čekanja pri navigaciji.

Za upravljanje navigacijom koristi se **Vue-Router** [5], koji omogućava

prikaz odgovarajućih komponenti prema URL putanji bez ponovnog učitavanja stranice, uz podršku za dinamičke parametre, zaštitu ruta i automatska preusmeravanja. **Vuex** [6] centralizuje stanje aplikacije, čuvajući podatke poput autentifikacionog tokena, statusa prijave i informacija o oglasima, dok **Vuex-PersistedState** [7] omogućava trajno čuvanje tih podataka u lokalnoj memoriji pregledača. **Vuex-Router-Sync** [8] sinhronizuje stanje rutiranja sa **Vuex** skladištem, olakšavajući praćenje aktivne rute.

Vuelidate [9] obezbeđuje validaciju korisničkih unosa i omogućava jednostavno definisanje pravila i prikaz poruka o greškama. Za stilizaciju i kreiranje responzivnih komponenti interfejsa korišćeni su **Bootstrap** [10] i **BootstrapVue** [11], što omogućava dosledan i moderan izgled aplikacije.

Komunikacija sa serverskim delom realizovana je preko **Axios** [12] biblioteke, koja omogućava slanje HTTP zahteva ka REST API servisima, obradu odgovora i rukovanje greškama, uz podršku za slanje autentifikacionih tokena i drugih podataka.

B. Serverska strana

Serverska strana aplikacije razvijena je korišćenjem **Node.js** [13] okruženja i **Express** [14] okvira, koji omogućavaju efikasno upravljanje HTTP zahtevima i definisanje REST API ruta. **Node.js** koristi asinhronu arhitekturu zasnovanu na događajima, što omogućava istovremenu obradu velikog broja zahteva, dok **Express** pruža modularnu organizaciju koda i podršku za midlver, autentifikaciju i validaciju.

Autentifikacija korisnika realizovana je pomoću **JWT (JSON Web Token)** [15] standarda i **Passport-JWT** [16] strategije, čime se obezbeđuje kontrolisan pristup resursima bez potrebe za čuvanjem sesija. **Joi** [17] biblioteka korišćena je za validaciju korisničkih unosa, dok su lozinke sigurno heširane pomoću **Bcrypt** [18] biblioteke.

Za rukovanje fajlovima i otpremanje slika korišćen je **Multer** [19], a **Morgan** [20] je korišćen za logovanje HTTP zahteva. Upravljanje konfiguracionim parametrima omogućava **Dotenv** [21] biblioteka, koja učitava vrednosti iz fajla okruženja van izvornog koda, dok **CORS (Cross-Origin Resource Sharing)** [22] omogućava siguran pristup serveru sa različitih domena. Parsiranje JSON podataka realizovano je uz **Body-Parser** [23], čime su podaci klijenta odmah dostupni za dalju obradu.

C. Eksterni servisi

Aplikacija koristi eksterne servise za funkcionalnosti poput obrade slika i slanja elektronske pošte, čime se rasterećuje glavni server i obezbeđuje

skalabilnost sistema.

Cloudinary [24] se koristi za čuvanje, optimizaciju i isporuku slika oglasa, dok **Mailgun** [25] omogućava pouzdano slanje verifikacionih i mejlova za resetovanje lozinke korisnika.

D. Rad sa bazom podataka

Za upravljanje podacima aplikacije korišćena je relaciona baza podataka MySQL, koja obuhvata entitete kao što su korisnici, oglasi, rezervacije i poruke. Za rad sa bazom korišćen je **Sequelize** [26], ORM (Object-Relational Mapping) alat koji omogućava jednostavno upravljanje podacima korišćenjem objekata i JavaScript sintakse. Baza je hostovana na platformi **Filess.io** [27], koja obezbeđuje udaljeni pristup i jednostavno upravljanje bazama.

E. Testiranje

Testiranje je ključno za proveru ispravnosti i pouzdanosti sistema. Za testiranje serverske strane korišćeni su **Jest** [28] i **Supertest** [29], dok je za testiranje kompletne aplikacije korišćen **Cypress** [30], koji omogućava simulaciju interakcije korisnika i vizuelno praćenje izvršenja testova.

V. ARHITEKTURA SISTEMA

A. Postavljanje sistema

Klijentski sloj postavljen je na **Netlify** [31], koji omogućava jednostavno hostovanje statičkog sadržaja i automatsko ažuriranje aplikacije.

Serverski sloj hostovan je na **Render** [32] platformi, koja pruža podršku za **Node.js** aplikacije, automatsko pokretanje servera i pristup putem interneta.

Baza podataka se nalazi na servisu **Filess.io** i serverski sloj komunicira sa njom koristeći **Sequelize**.

Komunikacija između svih komponenti odvija se putem interneta korišćenjem HTTP protokola i JSON formata podataka.

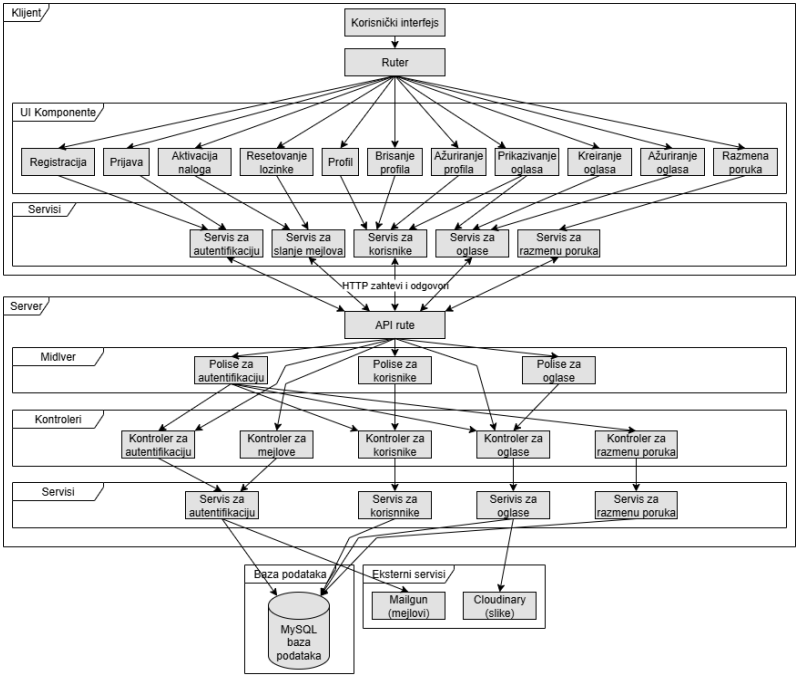
B. Prikaz arhitekture

Arhitektura veb aplikacije *Kolekcionarski modeli* (Sl. 2) zasnovana je na klijent-server principu, gde klijentski sloj prikazuje interfejs i omogućava interakciju sa korisnikom, a serverski sloj obrađuje zahteve, upravlja podacima i sprovodi poslovnu logiku. Takva organizacija omogućava skalabilnost, bolju bezbednost i jednostavnije održavanje sistema.

Serverski deo razvijen je kao modularni monolit, što predstavlja kompromis između klasičnog monolita i mikroservisa. Modularnost

omogućava jasnu podelu funkcionalnosti u zasebne module, čime se olakšava održavanje, testiranje i buduće proširenje sistema, a zadržava se jednostavnost implementacije i upravljanja.

Komunikacija između klijenta i servera odvija se u JSON formatu, uz autentifikaciju korisnika preko JWT tokena.



Sl. 2. Prikaz arhitekture aplikacije.

C. Klijentska i serverska logika

Klijentska logika fokusirana je na prezentaciju i interakciju sa korisnikom, prikupljanje podataka i iniciranje zahteva ka serveru.

Serverska logika odgovorna je za obradu zahteva, sprovođenje poslovnih pravila i komunikaciju sa bazom i eksternim servisima. Midlver slojevi omogućavaju autentifikaciju, autorizaciju i validaciju podataka, dok kontroleri posreduju između HTTP zahteva i servisa.

D. Komunikacija sa bazom podataka

Serverski sloj koristi **Sequelize** biblioteku za rad sa MySQL bazom. Modeli omogućavaju kreiranje, čitanje, izmenu i brisanje podataka, dok relacije između modela olakšavaju dohvatanje povezanih podataka.

E. Sigurnosni aspekti

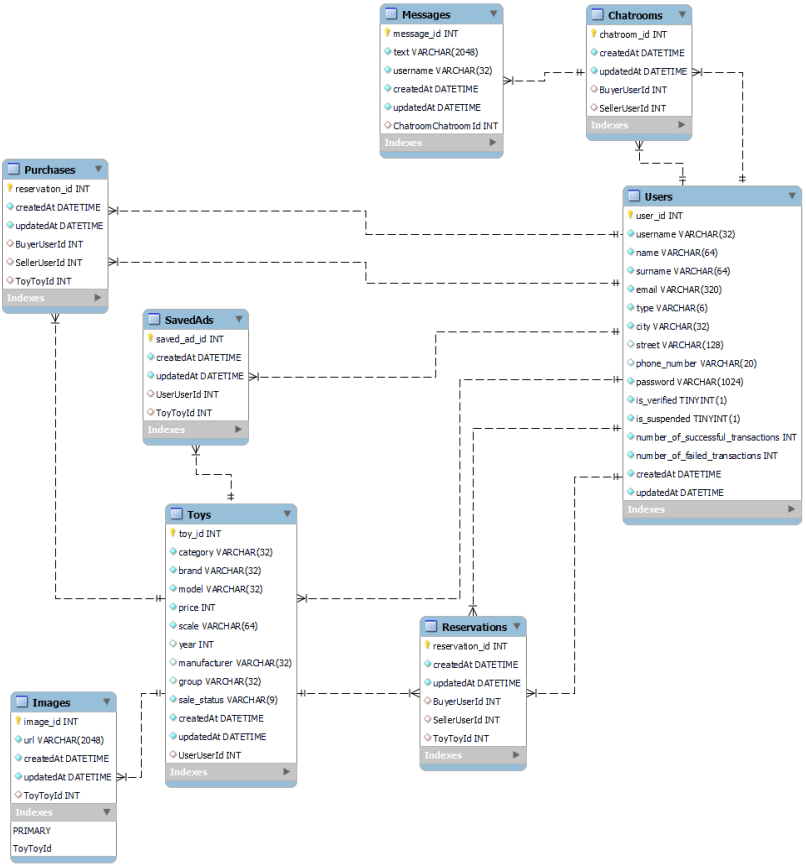
Sigurnosni aspekti ključni su za zaštitu podataka korisnika i sprečavanje neautorizovanog pristupa.

Autentifikacija se vrši pomoću JWT tokena, koji omogućava proveru identiteta bez čuvanja sesija na serveru.

Promenljive okruženja koriste se za čuvanje osetljivih podataka, što smanjuje rizik od kompromitovanja informacija.

Sequelize i midlver slojevi pružaju dodatnu zaštitu od SQL injekcija i nevalidnih unosa, što doprinosi stabilnosti i sigurnosti sistema.

VI. MODEL PODATAKA



Sl. 3. Dijagram entiteta i veza.

Model podataka aplikacije *Kolekcionarski modeli* dizajniran je da podrži sve ključne funkcionalnosti sistema, obezbedi bezbedno čuvanje informacija i jasno definiše veze između entiteta. Baza podataka sadrži osam ključnih entiteta: korisnike, oglase, slike, sačuvane oglase, rezervacije, kupovine, sobe za poruke i poruke.

Dijagram entiteta i veza (Sl. 3) prikazuje međusobne relacije između ovih entiteta, koje osiguravaju integritet podataka i pravilno funkcionisanje svih poslovnih procesa u aplikaciji.

Ograničenja i pravila integriteta sprečavaju nelogične situacije, kao što su dvostruka prodaja ili nepostojanje referenciranih zapisa. Ovakav dizajn osigurava konzistentnost, pouzdanost i skalabilnost sistema, uz jasno mapiranje poslovnih aktivnosti na konkretne entitete baze podataka.

VII. Implementacija ključnih funkcionalnosti

A. Implementacija klijentske strane

Klijentski deo aplikacije sastoji se od **Vue** komponenti, **Vuex** skladišta za globalno upravljanje stanjima, **Vue** rutera za navigaciju i servisa za komunikaciju sa serverskim API interfejsom.

Vue komponente obično sadrže sledeće delove: šablon, podatke, validacije, metode i inicijalizacionu funkciju.

Šablon (Sl. 4) definiše vizuelni izgled i raspored elemenata. Na primer, komponenta za kreiranje oglasa sadrži formu sa poljima za kategoriju, marku, model, razmeru, cenu, sliku i opcione podatke.

```
<b-form-group label="Kategorija:" label-for="input-category">
  <b-form-select
    id="input-category"
    v-model="$v.form.category.$model"
    :options="form.categories"
    :state="validateState('category')"
    required
  ></b-form-select>
  <b-form-invalid-feedback>
    Kategorija nije validna.
  </b-form-invalid-feedback>
</b-form-group>
```

Sl. 4. Deo koda šablona za unos kategorije i prikaz greške.

Svako polje forme je povezano sa podacima iz **data** objekta, čime se ostvaruje dvosmerna veza između korisničkog unosa i stanja komponente. **Data** funkcija (Sl. 5) vraća reaktivni objekat sa svim potrebnim vrednostima i pomoćnim promenljivama za kontrolu statusa unosa i prikaz grešaka ili poruka o uspehu.

```
data () {
  return {
    error: null,
    form: {
        category: '',
        categories: [
            { text: 'Izaberi kategoriju', value: '', disabled: true},
            { text: 'Auto', value: 'car' },
            { text: 'Motor', value: 'motorcycle' },
            { text: 'Karting', value: 'karting' },
            { text: 'Kaciga', value: 'helmet' },
            { text: 'Kamion', value: 'truck' },
            { text: 'Autobus', value: 'bus' },
            { text: 'Avion', value: 'plane' },
            { text: 'Voz', value: 'train' },
            { text: 'Brod', value: 'boat' },
            { text: 'Tramvaj', value: 'tram' },
            { text: 'Bicikl', value: 'bicycle' },
            { text: 'Tenk', value: 'tank' },
        ],
        brand: '',
        model: '',
        scale: '',
        price: null,
        image: null,
        year: null,
        manufacturer: '',
        group: ''
    },
    showDismissibleAlertError: false,
    dismissibleAlertMessageError: '',
    showDismissibleAlertSuccess: false,
    dismissibleAlertMessageSuccess: ''
  }
},
```

Sl. 5. data funkcija za kreiranje oglasa.

Validacije (Sl. 6) koriste biblioteku **Vuelidate** za definisanje pravila unosa, uključujući obavezna polja, format i dodatne uslove.

```
validations: {
  form: {
    category: { required },
    brand: { required, valid: regx2 },
    model: { required, valid: regx2 },
    scale: { required, valid: regx1 },
    price: { required, minValue: greaterThanZero },
    image: { required, value: validImage },
    year: { value: validYear },
    manufacturer: { valid: regx2 },
    group: { valid: regx2 }
  }
},
```

Sl. 6. Validaciona pravila za kreiranje oglasa

Metode (Sl. 7) sadrže logiku za obradu korisničkih akcija i komunikaciju sa serverom, uključujući slanje podataka u formatu FormData kada je potrebno preneti slike, dok se ostali podaci šalju u JSON formatu.

Inicijalizaciona funkcija je deo životnog ciklusa Vue komponente koji se izvršava u trenutku kada komponenta postane vidljiva na stranici.

```
async create () {
  this.$v.form.$touch();
  if (this.$v.form.$anyError) {
    return;
  }

  this.showDismissibleAlert = false;

  const formData = new FormData();
  formData.append('category', this.form.category);
  formData.append('brand', this.form.brand);
  formData.append('model', this.form.model);
  formData.append('scale', this.form.scale);
  formData.append('price', this.form.price);
  formData.append('image', this.form.image);

  if (this.form.year !== null && this.form.year !== undefined && this.form.year.length !== 0) {
    formData.append('year', this.form.year);
  }

  if (this.form.manufacturer !== null && this.form.manufacturer !== undefined && this.form.manufacturer.length !== 0) {
    formData.append('manufacturer', this.form.manufacturer);
  }

  if (this.form.group !== null && this.form.group !== undefined && this.form.group.length !== 0) {
    formData.append('group', this.form.group);
  }

  try {
    const response = await AdService.create(formData);
    this.dismissibleAlertMessageSuccess = "Oglas je uspešno kreiran.";
    this.showDismissibleAlertSuccess = true;
  } catch (err) {
    this.error = err.response.data.error;
    this.dismissibleAlertMessageError = this.error;
    this.showDismissibleAlertError = true;
  }
}
```

Sl. 7. Metoda za kreiranje oglasa.

Za navigaciju između stranica iskorišćen je **Vue-Router**. Svaka ruta definiše putanju, komponentu i eventualne uslove pristupa (Sl. 8). Parametri rute se sinhronizuju sa **Vuex** skladištem, a prelazak se može izvršiti deklarativno ili programski. Čuvari navigacije (Sl. 9) osiguravaju kontrolu pristupa stranicama i dinamičko prosleđivanje parametara.

```
{
  path: '/ads',
  name: 'createAd',
  component: CreateAd,
  meta: { requiresAuth: true, role: 'seller' }
},
{
  path: '/sellerAds/:username',
  name: 'sellerAds',
  component: SellerAds,
  meta: { requiresAuth: true, role: 'seller' }
},
{
  path: '/users/:username/edit',
  name: 'editUser',
  component: EditUser,
  meta: { requiresAuth: true }
},
```

Sl. 8. Jedan deo konfiguracije ruta.

```
router.beforeEach((to, from, next) => {
  if (to.matched.some(record => record.meta.requiresAuth)) {
    if (!store.state.isUserLoggedIn) {
      alert('Morate biti prijavljeni da biste pristupili ovoj stranici.')
      return next({ name: 'login' })
    }

    if (to.meta.role && (!store.state.user || store.state.user.role !== to.meta.role)) {
      alert('Nemate dozvolu da pristupite ovoj stranici.')
      return next(false)
    }
  }
  next()
})
```

Sl. 9. Kod čuvara navigacije.

Globalnim stanjem aplikacije upravlja se preko **Vuex** skladišta (Sl. 10), što omogućava centralizovan pristup informacijama o korisniku, tokenu, statusu prijave i trenutno selektovanom oglasu. Mutacije služe za sinhrono menjanje stanja, dok akcije omogućavaju asinhrone operacije.

```
import Vue from 'vue'
import Vuex from 'vuex'
import createPersistedState from 'vuex-persistedstate'

// Allows us to access the store and all the Vue components.
Vue.use(Vuex)

export default new Vuex.Store({
    strict: true,
    plugins: [
        createPersistedState()
    ],
    state: {
        token: null,
        user: null,
        isUserLoggedIn: false,
        ad: null
    },
    mutations: {
        setToken (state, token) {
            state.token = token
            if (token){
                state.isUserLoggedIn = true
            } else {
                state.isUserLoggedIn = false
            }

        },
        setUser (state, user){
            state.user = user
        },
        setAd (state, ad) {
            state.ad = ad
        }
    },
    actions: {
        setToken ({commit}, token){
            commit('setToken',token)
        },
        setUser ({commit}, user){
            commit('setUser', user)
        },
        setAd ({commit}, ad) {
            commit('setAd', ad)
        }
    }
})
```

Sl. 10. Kod Vuex skladišta korišćenog u aplikaciji.

Komponente komuniciraju sa serverom preko servisa koji enkapsuliraju HTTP zahteve koristeći centralizovanu **Axios** instancu (Sl. 11). Zaštićene operacije automatski dodaju JWT token iz **Vuex** skladišta, a asinhroni pozivi se obrađuju pomoću **async/await**. Obrada grešaka vrši se unutar komponenti i prikazuje korisniku kroz vizuelne elemente za poruke ili upozorenja.

```
import axios from 'axios'
import store from '@/store/store'

export default () => {
    return axios.create({
        baseURL: `https://collectabletoyswebapp.onrender.com/`,
        headers: {
            Authorization: `Bearer ${store.state.token}`
        }
    })
}
```

Sl. 11. Konfiguracija Axios instance.

B. Implementacija serverske strane

Serverska strana aplikacije obuhvata sve ključne funkcionalnosti, uključujući autentifikaciju, upravljanje korisnicima i oglasima, razmenu poruka, obradu slika, slanje elektronske pošte i pristup bazi podataka. Kod je organizovan u module prema njihovoj ulozi u sistemu: konfiguracija, rute, kontroleri, servisi, modeli, midlver funkcije i testovi.

Za konfiguraciju aplikacije koristi se fajl **config.js** (Sl. 12), koji objedinjuje sve sve ključne parametre, uključujući port servera, podešavanja baze podataka, autentifikaciju, kao i parametre za integraciju sa eksternim servisima za čuvanje slika i slanje mejlova. Vrednosti se učitavaju iz fajla okruženja pomoću biblioteke **Dotenv**, što omogućava lako prilagođavanje različitim okruženjima i čuva osetljive podatke van izvornog koda.

```
require('dotenv').config(); // Load environment variables from .env file
module.exports = {
    port: process.env.PORT || 8081,
    db: {
        database: process.env.DB_NAME || 'collectiblemodels',
        user: process.env.DB_USER || 'root',
        password: process.env.DB_PASSWORD || 'root',
        options: {
            dialect: process.env.DB_DIALECT || 'mysql',
            host: process.env.DB_HOST || 'localhost',
            port: process.env.DB_PORT || 3306
        }
    },
    authentication: {
        jwtSecret: process.env.JWT_SECRET || 'secret'
    },
    cloudinary: {
        cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
        api_key: process.env.CLOUDINARY_API_KEY,
        api_secret: process.env.CLOUDINARY_API_SECRET
    },
    mailgun: {
        domain: process.env.MAILGUN_DOMAIN,
        api_key: process.env.MAILGUN_API_KEY
    }
}
```

Sl. 12. Konfiguracioni fajl.

Glavni fajl **app.js** (Sl. 13) predstavlja ulaznu tačku aplikacije i u njemu se definišu midlver funkcije, uspostavlja konekcija sa bazom, učitava konfiguracija za autentifikaciju i povezuju se sve rute sistema.

```
const express = require('express')
const bodyParser = require('body-parser')
const cors = require('cors')
const morgan = require('morgan')
const {sequelize} = require('./models')
const config = require('./config/config')

const app = express()
app.use(morgan('combined')) // For logs
app.use(bodyParser.json()) // For easy parsing of json requests

const corsOptions = {
    origin: 'https://clinquant-dodol-424960.netlify.app',
    exposedHeaders: 'Location',
    optionsSuccessStatus: 200
}

app.use(cors(corsOptions)) // To allow any client to access this
require('./passport')
require('./routes')(app)

if(process.env.NODE_ENV !== "test") {
    sequelize.sync().then(() => {
        app.listen(config.port)
        console.log(`Server started on port ${config.port}`)
    })
}
```

Sl. 13. Glavni fajl aplikacije.

Rute su organizovane u fajlu **routes.js** (Sl.14), gde se precizno definiše koje HTTP putanje odgovaraju kojim funkcionalnostima. U većini slučajeva se pre poziva kontrolera pozivaju odgovarajuće midlver funkcije, za autentifikaciju, obradu slika ili validaciju podataka.

```
app.post('/ads',
    isAuthenticated,
    AdControllerPolicy.upload.single('image'),
    AdControllerPolicy.createOrUpdate,
    AdController.create)
```

Sl. 14. Primer rute za kreiranje oglasa.

Autentifikacija se oslanja na JWT strategiju putem biblioteke **Passport**. Validnost tokena proverava se midlver funkcijom **isAuthenticated**, koja korisnika postavlja kao **req.user** ako je token ispravan. Validacija podataka (Sl. 15) implementirana je pomoću biblioteke **Joi** i organizovana kroz polise. One definišu pravila za unos, uključujući upotrebu regularnih izraza i dodatnih provera, kao i obradu slika putem **Multer** biblioteke.

```
// Define the schema for user registration validation.
const schema = Joi.object({
    username: Joi.string().required().min(1).max(32).pattern(/^[a-zA-Z0-9_.]+$/),
    name: Joi.string().required().min(1).max(64),
    surname: Joi.string().required().min(1).max(64),
    email: Joi.string().required().email().max(320),
    type: Joi.string().required().min(1).max(6),
    city: Joi.string()
        .required()
        .min(1)
        .max(32)
        .pattern(/^[a-zšđžčćA-ZŠĐŽČĆ]+$/),
    street: Joi.string()
        .max(128)
        .pattern(/^[a-zšđžčćA-ZŠĐŽČĆ0-9]+$/),
    phone_number: Joi.string()
        .max(20)
        .pattern(/^[\+]?[()]?[0-9]{3}[)]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{3,6}$/),
    password: Joi.string()
        .required()
        .pattern(/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,32}$/)
});
```

Sl. 15. Primer validacije iz AuthenticationControllerPolicy polise.

Kontroleri služe kao posrednici između HTTP zahteva i servisnog sloja. Njihova odgovornost je da prihvate podatke iz zahteva, izvrše osnovnu validaciju i pozovu odgovarajuću servisnu metodu. Na taj način se jasno razdvajaju odgovornosti, pri čemu su kontroleri fokusirani na komunikaciju sa klijentom, dok je poslovna logika izdvojena u servisne module. Svaka funkcionalna celina ima svoj kontroler: za autentifikaciju, slanje mejlova, korisnike, oglase i poruke. Kontroleri pozivaju servise i vraćaju odgovarajući odgovor klijentu, kao što je prikazano u primeru registracije (Sl. 16).

```
async register(req, res) {
    try {
        // Call the service to register a new user.
        const { user, token } = await AuthenticationService.registerUser(req.body);

        // Respond with 201 Created, set Location header, and return user data with token.
        res.status(201)
            .location(`users/${user.user_id}`)
            .send({ user, token });
    } catch (err) {
        // Determine status code: use service-provided status or default to 500 for unexpected errors.
        const resStatus = err.status || 500;

        // Determine error message: use service-provided message or default generic message.
        const errMessage = err.message || 'Greška tokom kreiranja korisnika. Pokušaj ponovo.';

        // Send error response.
        res.status(resStatus).send({ error: errMessage });
    }
},
```

Sl. 16. Primer registracije iz AuthenticationController kontrolera.

Servisni sloj predstavlja centralno mesto za implementaciju poslovne logike aplikacije. Servisi primaju podatke od kontrolera, izvršavaju potrebne operacije nad bazom podataka ili eksternim servisima i vraćaju rezultat.

Primenjuju se transakcije (Sl. 17) kada operacija obuhvata više povezanih koraka, rukuje se greškama kroz status kodove i poruke, a validacija se uglavnom oslanja na prethodno izvršene midlvere. Modularna struktura omogućava lako održavanje i proširivanje funkcionalnosti.

```
const transaction = await sequelize.transaction(); // Start a transaction.
try {
    // Create a new purchase record.
    const purchase = await Purchase.create({
        ToyToyId: reservation.ToyToyId,
        SellerUserId: reservation.SellerUserId,
        BuyerUserId: reservation.BuyerUserId
    }, { transaction });

    // Delete the reservation.
    await Reservation.destroy({
        where: { ToyToyId: toy.toy_id, SellerUserId: req.params.id },
        transaction
    });

    // Update the sale status of the toy to "sold".
    await Toy.update(
        { sale_status: "sold" },
        { where: { toy_id: toy.toy_id }, transaction }
    );

    // Increment the number of successful transactions for the seller and buyer.
    await User.increment(
        { number_of_successful_transactions: +1 },
        { where: { user_id: [reservation.SellerUserId, reservation.BuyerUserId] }, transaction }
    );

    // Commit the transaction.
    await transaction.commit();
    return { purchase }
} catch (err) {
    // Roll back the transaction.
    await transaction.rollback();
    // ... error handling
```

Sl. 17. Primer korišćenja transakcije u servisu tokom finalizacije kupovine.

U aplikaciji se koriste sledeći servisi:

- **AuthenticationService** – autentifikacija, aktivacija naloga, slanje mejlova.
- **AdService** – upravljanje oglasima i slikama.
- **UserService** – upravljanje korisnicima i interakcija korisnika sa oglasima.
- **ChatroomService** – slanje i prijem poruka.

Servisi obično prate sledeći tok: prijem podataka, eventualna transformacija, izvršavanje operacija nad bazom, komunikacija sa eksternim servisima i vraćanje rezultata ili greške.

Aplikacija koristi dva ključna eksterna servisa: **Cloudinary** za otpremanje i obradu slika, i **Mailgun** za slanje mejlova čime se omogućava pouzdana obrada fajlova i komunikacija sa korisnicima.

VIII. INTERFEJS I KORISNIČKO ISKUSTVO

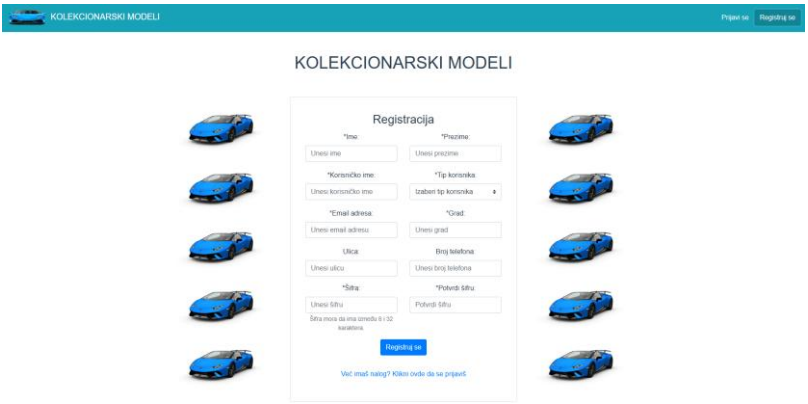
A. Principi dizajna interfejsa

Korisnički interfejs i korisničko iskustvo ključni su za efikasnu, intuitivnu i prijatnu interakciju korisnika sa sistemom. Prilikom razvoja interfejsa primenjeni su sledeći principi:

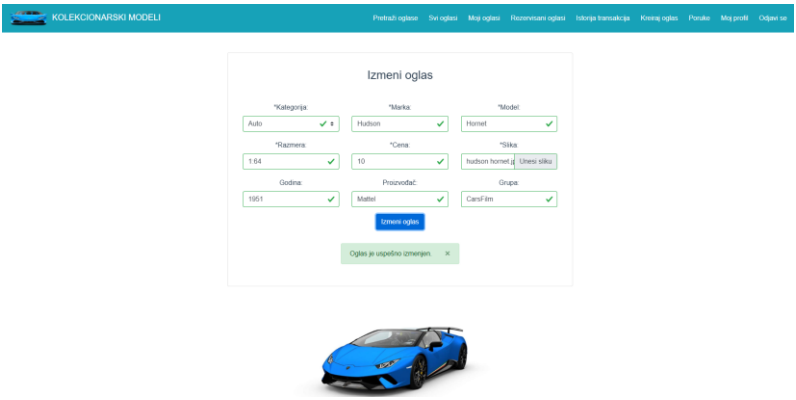
- **Jednostavnost i preglednost** – jasno označene kontrole, forme i navigacioni elementi omogućavaju korisniku da odmah prepozna dostupne funkcionalnosti.
- **Konzistentnost i prilagodljivost** – uniformni vizuelni stil i predvidivo ponašanje elemenata olakšavaju navigaciju kroz aplikaciju i smanjuju mogućnost greške. Interfejs je prilagođen različitim tipovima korisnika, kao što su kupci i prodavci.
- **Vidljivost statusa i povratna informacija** – korisnicima se prikazuju jasne poruke o uspehu ili neuspehu izvršenih akcija, čime se povećava poverenje u sistem.
- **Responzivnost i organizacija sadržaja:** informacije su pregledno raspoređene, čime se omogućava efikasna interakcija bez preopterećenja korisnika.

B. Primeri stranica

Za ilustraciju izgleda aplikacije prikazane su stranica za registraciju (Sl. 18) i stranica za ažuriranje oglasa (Sl. 19).



Sl. 18. Stranica za registraciju.



Sl. 19. Stranica za ažuriranje oglasa u trenutku nakon uspešnog ažuriranja.

IX. Testiranje aplikacije

Testiranje je imalo ključnu ulogu u proveri funkcionalnosti, bezbednosti i otpornosti sistema. Sprovođeni su različiti tipovi testova koji zajedno obuhvataju sve slojeve aplikacije, od pojedinačnih funkcija do kompletne korisničke interakcije.

Grupisani su u sledeće kategorije:

- **Jedinični testovi** – proveravaju rad pojedinačnih funkcija u izolaciji.
- **Integracioni testovi** – ispituju saradnju različitih slojeva (midlver, kontroler, servis, baza).
- **Dimni testovi** – brzo testiraju osnovne funkcionalnosti nakon postavljanja sistema.
- **Bezbednosni testovi** – proveravaju otpornost na ranjivosti kao što su SQL injekcije i neautorizovan pristup.
- **Testovi otpornosti na greške** – simuliraju kvarove poput pada baze i proveravaju reakciju sistema.
- **Testovi kompletne funkcionalnosti** – simuliraju ponašanje korisnika kroz sve slojeve sistema.

Ovakav pristup omogućio je rano otkrivanje grešaka, povećanu stabilnost sistema i pouzdano funkcionisanje u različitim scenarijima.

X. Zaključak

Predstavljena veb aplikacija unapređuje tržište kolekcionarskih modela omogućavanjem jednostavnije kupovine i prodaje i efikasnije komunikacije

između korisnika. Umesto da je ponuda rasuta po nespecijalizovanim platformama, korisnicima se nudi centralizovano i pregledno mesto posvećeno isključivo kolekcionarskim modelima. Sistem obuhvata sve ključne funkcionalnosti potrebne za pouzdano i bezbedno korišćenje, uz jednostavno praćenje aktivnosti unutar aplikacije.

Pored postignutih rezultata, aplikacija pruža osnovu za dalja unapređenja koja bi dodatno poboljšala korisničko iskustvo i efikasnost sistema. To uključuje naprednije opcije pretrage i filtriranja oglasa, omogućavanje dodatnih informacija i više slika po oglasu radi bolje preglednosti, optimizaciju komunikacije u realnom vremenu, kao i razvoj mobilne verzije aplikacije koja bi korisnicima omogućila praktičniji pristup. Ova unapređenja mogu dodatno povećati preglednost, brzinu i udobnost korišćenja, potvrđujući potencijal aplikacije da dugoročno doprinese organizovanijem i transparentnijem tržištu kolekcionarskih modela.

ZAHVALNICA

Zahvaljujem se svom mentoru, prof. Dr Nemanji Radosavljeviću, na posvećenosti, podršci i savetima tokom izrade ovog rada.

LITERATURA

[1] I. Sommerville, „Software Engineering,“ 9th ed., Addison-Wesley, 2011, pp. 84-85.
[2] *IEEE Recommended Practice for Software Requirements Specifications*, IEEE Standard 830, 1998. p. 4.
[3] A. Cockburn, „Writing Effective Use Cases,“ Addison-Wesley, 2001, p. 1.
[4] Vue, [Na mreži], Dostupno: https://vuejs.org/
[5] Vue Router, [Na mreži]. Dostupno: https://router.vuejs.org/
[6] Vuex, [Na mreži]. Dostupno: https://vuex.vuejs.org/.
[7] vuex-persistedstate, GitHub repozitorijum. [Na mreži]. Dostupno: https://github.com/robinvdvleuten/vuex-persistedstate/
[8] vuex-router-sync, GitHub repozitorijum. [Na mreži]. Dostupno: https://github.com/vuejs/vuex-router-sync/
[9] Vuelidate, [Na mreži]. Dostupno: https://vuelidate-next.netlify.app/
[10] Bootstrap, [Na mreži]. Dostupno: https://getbootstrap.com/
[11] BootstrapVue, [Na mreži]. Dostupno: https://bootstrap-vue.org/
[12] Axios, [Na mreži]. Dostupno: https://axios-http.com/
[13] Node.js, [Na mreži]. Dostupno: https://nodejs.org/
[14] Express, [Na mreži]. Dostupno: https://expressjs.com/
[15] JWT.io, „Introduction to JSON Web Tokens,“ Auth0. [Na mreži]. Dostupno: https://jwt.io/introduction/
[16] Passport, [Na mreži]. Dostupno: https://www.passportjs.org/
[17] Joi, [Na mreži]. Dostupno: https://joi.dev/
[18] bcrypt, GitHub repozitorijum. [Na mreži]. Dostupno: https://github.com/kelektiv/node.bcrypt.js/
[19] multer, GitHub repozitorijum. [Na mreži]. Dostupno: https://github.com/expressjs/multer/

[20] morgan, GitHub repozitorijum. [Na mreži]. Dostupno: https://github.com/expressjs/morgan/
[21] dotenv, GitHub repozitorijum. [Na mreži]. Dostupno: https://github.com/motdotla/dotenv/
[22] MDN Web Docs, „Cross-Origin Resource Sharing (CORS),“ [Na mreži]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS/
[23] body-parser, GitHub repozitorijum. [Na mreži]. Dostupno: https://github.com/expressjs/body-parser/
[24] Cloudinary, [Na mreži]. Dostupno: https://cloudinary.com/
[25] Mailgun, [Na mreži]. Dostupno: https://www.mailgun.com/
[26] Sequelize, [Na mreži]. Dostupno: https://sequelize.org/.
[27] Filess.io, [Na mreži]. Dostupno: https://filess.io/
[28] Jest, [Na mreži]. Dostupno: https://jestjs.io/
[29] supertest, GitHub repozitorijum. [Na mreži]. Dostupno: https://github.com/forwardemail/supertest
[30] Cypress, [Na mreži]. Dostupno: https://www.cypress.io/
[31] Netlify, [Na mreži]. Dostupno: https://www.netlify.com/
[32] Render, [Na mreži]. Dostupno: https://render.com/

ABSTRACT

The aim of this work is to create a functional web application for buying and selling collectible models, taking into account the shortcomings of existing solutions, with the purpose of improving user experience and efficiency compared to the current market, as well as to present in detail the process of development and implementation of the proposed solution. The current market situation has been analyzed, highlighting the need for the development of such an application. Particular focus is placed on the application's functionality and the way it is implemented. In addition, the application architecture, design decisions made during development, and a detailed overview of the technologies used are presented. The description also covers how the various functionalities were tested to ensure correctness and reliability. The result of this work is a detailed overview of the development and implementation of the application, along with an assessment of the achieved results, highlighting possibilities for further development and improvement of the solution. The developed application integrates all stages of the trading process into a single solution tailored to the specific niche of collectible models and represents an original contribution in this field, providing users with a practical, reliable, and efficient tool for trading.

DEVELOPMENT AND IMPLEMENTATION OF A WEB APPLICATION FOR BYING AND SELLING COLLECTIBLE MODELS

Ognjen Arsenijević, dr Nemanja Radosavljević