# Razvoj aplikacije Aukcije

Autori:

*Mateo Feltrin, Emmanuel Gordan Fett, Dorian Lulić, Dominik Pilat, Goran Vrban*

# Opis aplikacije

Web aplikacija Aukcije služi za oglašavanje predmeta na aukciji. Korisniku se nudi opcija za registraciju i prijavu.

Svi mogu pregledavati aukcije. Bilo koji registrirani korisnik može sudjelovati u aukciji: oglasiti neki predmet (uslugu) i licitirati za neki oglašeni predmet. Omogućeno je filtriranje i pretraživanje aukcija odnosno predmeta.

Na početnoj stranici korisnik može pregledavati sve trenutne aukcije koje izabire u slučaju želje za licitacijom te mu se nudi mogućnost sortiranja prikazanih predmeta prema određenim kategorijama - namještaj, automobili, laptop, mobiteli i ostalo.

Kod oglašavanja predmeta registrirani korisnik unosi naziv predmeta koji postavlja na aukciju, početnu cijenu, odabire kategoriju u koju se predmet svrstava, dodaje sliku, aktivno razdoblje aukcije (datum i vrijeme početka i završetka aukcije) te opis predmeta.

Za licitiranje predmeta korisnik odabire neki od oglašenih predmeta za koji želi ponuditi određenu cijenu te ulazi u prikaz tog predmeta. Kod prikaza predmeta na licitaciji vidljivi su podaci: naziv predmeta, slika, opis predmeta, početno i završno vrijeme aukcije te početna i trenutna cijena aukcije. Klikom na “Ponuda” korisniku se otvara mogućnost unošenja svote za koju želi povisiti trenutnu cijenu aukcije te se tako cijena ponudama stalno povećava do isteka aukcije.

Prijavljeni korisnik može otići na svoj profil te su tamo vidljivi njegovi podaci: ime i prezime, datum rođenja, e-mail. Prijavljeni korisnik unutar „moj profil“ stranice može izmijeniti svoje podatke (ime, prezime, adresu, email, lozinku), Također, može pregledavati svoje predmete na aukciji – koje je on postavio – te ih eventualno urediti ili obrisati.

Administrator može upravljati korisnicima, izmjeniti njihove podatke ili ih obrisati, također može dodavati nove kategorije predmeta

# ANALIZA

## Funkcionalni zahtjevi

A diagram of a diagram

Description automatically generated

Slika UML dijagram korištenja

|  |  |  |
| --- | --- | --- |
| **Funkcionalnost** | Opis | Realizacija |
| **Registracija korisnika i prijava u sustav** | Omogućuje novim korisnicima registraciju u aplikaciji za aukcije. Korisnici se mogu prijaviti putem svog e-maila i lozinke koja je stvorena prilikom registracije. | Realizirano |
| **Mogućnost oglašavanja predmeta (usluge) na aukciji** | Omogućuje registriranom korisniku postavljanje oglasa za prodaju predmeta ili usluge putem aplikacije za aukcije. Korisnik može navesti naziv predmeta ili usluge, opis, sliku, aktivno razdoblje aukcije, kategoriju, svrhu i početnu cijenu. | Realizirano |
| **Mogućnost licitiranja za predmet (uslugu) oglašen na aukciji** | Omogućuje registriranim korisnicima da licitiraju za predmet ili uslugu oglašenu na aukciji. Korisnik može unijeti željeni iznos za koji želi podići aukciju. | Realizirano |
| **Postavljanje podataka o predmetu (naziv, opis, slika, aktivno razdoblje aukcije, početna cijena i kategorija) pri oglašavanju** | Omogućuje registriranim korisnicima postavljanje podataka o predmetu koji se prodaje, uključujući naziv, opis, sliku predmeta, vrijeme aktivnosti aukcije, kategoriju i početnu cijenu. | Realizirano |
| **Prikaz trenutne/postignute cijene za predmet (uslugu) na aukciji** | Omogućuje prikaz trenutne ili postignute cijene predmeta ili usluge na aukciji, tako da korisnici mogu vidjeti koliko je trenutno ponuđeno za predmet ili uslugu. | Realizirano |
| **Prikaz zadnje izmjene cijene za predmet na aukciji** | Omogućuje prikaz zadnje izmjene cijene za predmet ili uslugu na aukciji, tako da korisnici mogu vidjeti koliko se cijena promijenila od prethodne ponude. | Realizirano |
| **Prikaz pojedinosti aukcije** | Omogućuje prikaz pojedinosti o aukciji ulaskom na pojedinu aukciju. | Realizirano |
| **Sustav licitacije koji omogućuje registriranim korisnicima sudjelovanje u licitaciji za predmet na aukciji** | Omogućuje korisnicima sudjelovanje u licitaciji za predmet ili uslugu na aukciji, aplikacija osigurava sustav licitacije koji osigurava fer i transparentan proces licitacije. Korisnici mogu unijeti svoje ponude i vidjeti trenutni vodeći iznos. Aplikacija prati sve ponude i automatski povećava vodeću ponudu kada drugi korisnik ponudi višu cijenu. Licitacija se završava kada istekne aktivno razdoblje aukcije ili kada nema novih ponuda u određenom razdoblju. | Realizirano |
| **Mogućnost pregledavanja prethodnih aukcija** | Omogućuje korisnicima da pregledaju prethodne aukcije koje su završene. Korisnici mogu vidjeti koja su bila predmeti ili usluge na aukciji i koja je bila konačna cijena. | Realizirano/ treba realizirati (ograničenje zbog postavki DBMS-a nismo u mogućnosti postaviti event za završetak aukcija |
| **Sortiranje po kategorijama, cijeni, nazivu, vremenu isteka** | Omogućuje korisnicima da na početnoj stranici, klikom na gumb određene kategorije sortiraju predmete u toj kategoriji te se oni prikazuju na početnoj stranici. | Realizirano |
| **Pregled svih aukcija** | Na početnoj stranici nudi se pregled svih predmeta koji su na aukciji | Realizirano |
| **Prikaz vlastitih aukcija na profilu** | Omogućuje korisniku odlazak na svoj profil gdje su vidljivi njegovi podaci te sve postavljene aukcije. | Realizirano |
| **Uređivanje postavljene aukcije na profilu** | Omogućuje korisniku uređivanje postavljene aukcije na svom profilu. | Realizirano |
| **Brisanje vlastite aukcije na profilu** | Omogućuje korisniku brisanje postavljene aukcije. | Realizirano |
| **Upravljanje korisnicima od strane administratora** | Administrator može vidjeti sve registrirane korisnike, izmjenjivati podatke i brisati korisnike | Realizirano |
| **Upravljanje kategorijama proizvoda od strane administratora** | Administrator može vidjeti sve kategorije, izmjenjivati nazive, dodavati nove kategorije i brisati postojeće | Realizirano |
| **Izmjena korisničkih podataka od strane korisnika** | Korisnik unutar profila može vidjeti i izmijeniti svoje podatke | Realizirano |

## Korisničke priče i kriteriji prihvatljivosti

Kao korisnik želim moći se prijaviti u sustav, tako da mogu imati povijest kupnje (transakcije).

* Sustav treba pružiti mogućnost prijave u sustav.
* U sustav se mogu prijaviti samo korisnici koji su registrirani već u sustav.

Kao korisnik želim moći se registrirat u sustav, tako da mogu objavljivat oglase, kupovat, pregled povijesti kupnje.

* Sustav treba omogućiti bilo kojem korisniku registraciju u sustav.

Kao registrirani kupac, želim imati mogućnost pretraživanja aukcija prema ključnoj riječi ili kategoriji, tako da mogu lako pronaći artikle za koje želim licitirati.

* Sustav treba pružiti mogućnost pretraživanja aukcija na temelju ključnih riječi ili kategorija.

Kao registrirani kupac, želim imati mogućnost filtriranja aukcija po kategoriji, tako da mogu lako pronaći artikle za koje želim licitirati.

* Sustav treba pružiti mogućnost filtriranja aukcija na temelju kategorije.

Kao prijavljeni korisnik želim imati mogućnost objave predmeta koji želim prodati, također postaviti početnu cijenu, početak i kraj aukcije, postavljanje slika, opis predmeta. Kako bih mogao prodati taj predmet

* Susta treba pružiti mogućnost objave predmeta i njegove početne cijene, početak i kraj aukcije predmeta, slike i opisa

Kao prijavljeni korisnik želim moći pregledati sve objavljene i trenutno aktivne oglase, kako bih mogao pronaći oglas koji me zanima.

* Sustav treba pružiti mogućnost pregleda svih trenutno aktivnih oglasa

Kao korisnik (registrirani ili ne) želim moći izvršiti sortiranje oglasa po cijenu (rastuće, padajuće, najnovije, najstarije), vrijeme isteka (najduže, najkraće).

Kao registrirani korisnik želim moći uređivati vlastite podatke kao što su: ime, prezime, e-mail, lozinku računa i adresu.

Kao registrirani korisnik želim moći pregledati sve svoje prijašnje kupnje, odnosno aukcije na kojima sam bio imao najveću ponudu.

Kao registrirani korisnik želim moći postaviti svoju ponudu na objavljeni oglas.

Kao registrirani korisnik želim moći urediti informacije o postavljenom oglasu i obrisati istoga.

Kao korisnik ( i registrirani) želim vidjeti aktualne aukcije, odnosno one koje još uvijek traju, a ne želim vidjeti istekle aukcije.

Kao registrirani korisnik želim moći objaviti vlastiti oglas.

Kupac može pregledati detalje o pojedinom predmetu na aukciji - naziv, sliku, opis, početnu cijenu, trenutnu cijenu, početak aukcije i kad završava.

* Sustav treba prikazivati sve relevantne informacije o predmetu na aukciji, uključujući naziv, sliku, opis, početnu cijenu, trenutnu cijenu, datum i vrijeme početka aukcije te datum i vrijeme završetka aukcije.

Administrator može pregledati sve registrirane korisnike i radit CRUD s njima. Kako bi mogao upravljati svim korisnicima

* Sustav treba prikazati na administratov zahtjev sve registrirane korisnike i omogućiti mu CRUD operacije nad njima

A screenshot of a computer

Description automatically generated with low confidence

Slika Dijagram aktivnosti postavljanja aukcije

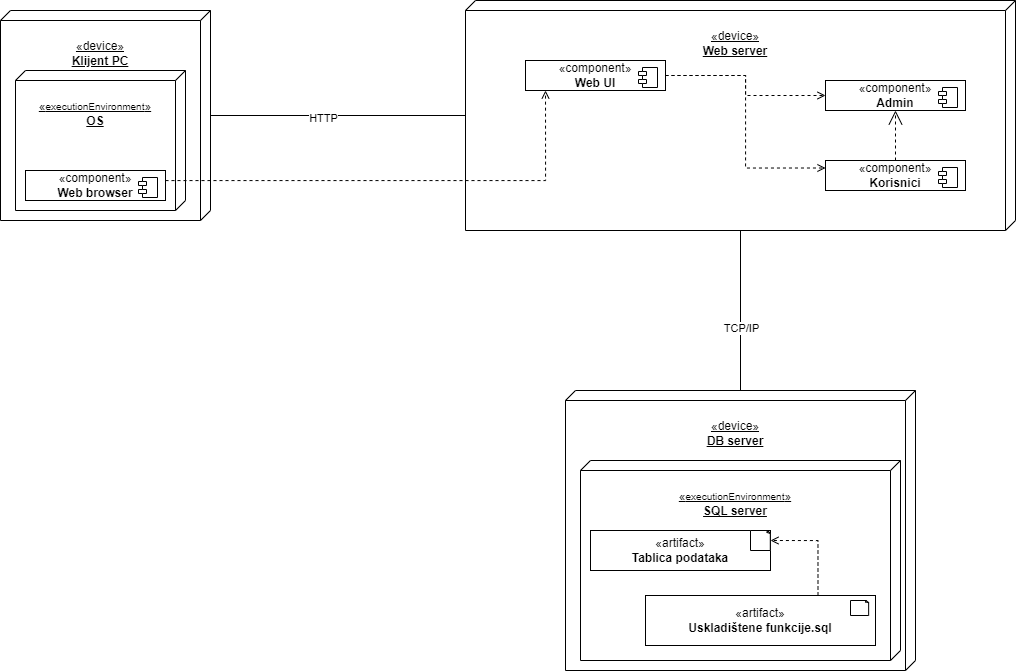
A screenshot of a black background

Description automatically generated with low confidence

Slika Dijagram aktivnosti pregleda aukcija

# ARHITEKTURA SUSTAVA

Dijagram nudi implementacijski pogled na sistem, softverske i hardverske komponente. Najvažniji su čvorovi device i executionEnvironment. Klijent PC predstavlja hardver koji je potreban da se određene softverske komponente vrte ili hardver koji će koristiti klijent za pristup aplikaciji Aukcije. Može to biti i laptop, naravno. Web server je čvor koji služi za prikaz aplikacije, a DB server služi za bazu podataka. To su 3 hardverske komponente.



Slika : UML dijagram rasporeda

Čvorovi su povezani komunikacijskim linkovima kako bi se vidjelo tko s kime komunicira. Klijent komunicira s web serverom jer nema potrebe za komunikaciju s DB serverom, a web server komunicira s DB serverom. Komunikacija klijenta s web serverom se obavlja pomoću HTTP-a(može HTTPS za sigurnost), a web servera s DB serverom se obavlja pomoću TCP/IP protokola.

OS predstavlja operacijski sustav koji je neophodan za pristup aplikaciji. Može biti Linux, Windows itd. Rješenje koje se koristi kako bi se hostirala baza je SQL server pod DB server.

Komponenta koja se mora koristiti na klijentu da bi se pristupilo web aplikaciji je web browser i nalazi se kod Klijent PC. Dakle, pokreće se web browser s određenim OS-om i to se sve nalazi kod računala klijenta. Admin komponenta se tiče administracije aplikacije i postavki, a Korisnici komponenta ovisi o Admin komponenti(isprekidana linija). Označuje da promjene jedne komponente utječu na promjene druge komponente. Uskladištene funkcije ili stored procedures služi kao SQL datoteka. Koriste Tablica podataka za pristup podacima i manipulaciju podacima. Označuje se isprekidanom linijom. Uskladištene funkcije su ovisne o Tablice podataka.

Ono što treba biti dostupno na svakom web serveru je web UI tj. user interface ili korisničko sučelje gdje se nalaze html-ovi. Ovisan je o Admin i Korisnici. Različit je i za oboje.

Ovisnost PC i web servera je moguća samo kod komponenti koje služe za prikaz korisniku tj. klijentu, a to su web browser i web UI. Web browser je ovisan o web UI komponenti.

## Tehnologije za razvoj aplikacije

* Klijentski dio aplikacije - Quasar.js razvojni okvir
* Poslužiteljski dio aplikacije (engl. backend) - Node.js i Express razvojni okviri
* Upiti prema poslužiteljskoj strani aplikacije
* API koji će na temelju zahtjeva od strane klijenta slati podatke

## Klijentski razvojni okvir Quasar

Quasar je klijentski razvojni okvir otvorenog koda za izradu modernih web aplikacija i mobilnih aplikacija. On se temelji na Vue.js, popularnom JavaScript okviru za izgradnju korisničkog sučelja.

Quasar pruža širok spektar komponenti i funkcionalnosti koje olakšavaju razvoj klijentske strane aplikacije. To uključuje gotove komponente za različite elemente korisničkog sučelja poput gumba, obrazaca, kartica, navigacije i još mnogo toga. Ove komponente su prilagodljive i pružaju mogućnost stiliziranja i prilagođavanja prema potrebama aplikacije.

Jedna od ključnih značajki Quasara je mogućnost izgradnje iste aplikacije za web, mobilne uređaje i desktop. Koristeći Quasar, možete stvoriti aplikaciju koja se može izvoditi u pregledniku, kao instalirana mobilna aplikacija na iOS-u i Androidu te kao izvršna datoteka na stolnom računalu. To pruža fleksibilnost i jednostavnost pri razvoju i distribuciji aplikacija na različitim platformama.

Quasar je moćan klijentski razvojni okvir koji olakšava izgradnju modernih web aplikacija i mobilnih aplikacija. Pruža bogate komponente, podršku za više platformi i integraciju s raznim alatima, što ga čini popularnim izborom za razvoj interaktivnih i responzivnih aplikacija.

## Poslužiteljski razvojni okviri Node.js i Express

Node.js je JavaScript okruženje koje korisniku omogućuje pokretanje svog koda na poslužitelju (izvan web preglednika). Node ima vlastiti upravitelj paketa tj. NPM (Node Package Manager) koji omogućuje programeru odabir između tisuća besplatnih paketa (node modula) za preuzimanje. Neke od prednosti Node.js-a su sljedeće:

• JavaScript radno okruženje otvorenog koda - omogućava da se JavaScript kod  
pokreće na serveru.

• Skalabilnost - omogućava skaliranje aplikacije horizontalno preko nekoliko servera ili vertikalno tako da se optimiziraju performance jednog servera

• Brzina – Koristi Chrome-ov V8 JavaScript engine napisan u C++ programskom jeziku

Express.js koji radi unutar Node.js poslužitelja. Express.js sebe predstavlja kao "brz, samostalan, minimalistički web okvir za Node.js", i to je doista točno ono što jest. Express je okvir Node.js-a. Umjesto pisanja koda pomoću Node.js, može se jednostavnije pisati poslužiteljski kod pomoću Expressa. Express pomaže u dizajniranju API-ja (Application Programming Interface), koji su ključan koncept svake backend tehnologije. Iz React.js aplikacije izradom HTTP zahtjeva (GET, POST, PUT, DELETE) povezujemo se s Express.js funkcijama koje pokreću poslužiteljsku stranu aplikacije. Neke od prednosti korištenja Expressa su sljedeće: Učinkovit, brzo i skalabilan

Kombinacija Node.js i Express omogućuje programerima izgradnju brzih, skalabilnih i efikasnih poslužiteljskih aplikacija. Node.js pruža performanse i fleksibilnost, dok Express olakšava upravljanje rutama, obradu zahtjeva i izgradnju web aplikacija. Ova kombinacija je popularna u svijetu razvoja web aplikacija i često se koristi za izgradnju API-ja, mikroservisa i web aplikacija općenito.

## Web servisi i API

Web servisi su softverske komponente koje omogućuju razmjenu podataka i funkcionalnosti između različitih aplikacija putem interneta. Oni omogućuju komunikaciju i integraciju između različitih sistema, nezavisno o platformi ili tehnologiji koju koriste.

API (Application Programming Interface) je skup definiranih metoda i pravila koje omogućuju komunikaciju između različitih softverskih aplikacija. API-ji definraju načine na koje jedna aplikacija može da pristupi i koristi funkcionalnosti ili podatke druge aplikacije. Oni omogućuju programerima da integriraju funkcionalnosti drugih aplikacija u svoje projekte, bez potrebe za razumijevanjem detalja implementacije. API-ji se često koriste u kombinaciji sa web servisima. Web servisi obično pružaju funkcionalnosti i podatke putem API-ja, omogućavajući drugim aplikacijama da ih koriste putem standardnih HTTP (Hypertext Transfer Protocol) zahtjeva. API-ji definiraju dostupne rute, parametre i formate podataka koji se koriste prilikom komunikacije s web servisima. Kroz web servise i API-je, aplikacije mogu razmjenjivati podatke, izvršavati određene funkcionalnosti ili čak koristiti funkcionalnosti drugih aplikacija kako bi proširile svoje mogućnosti.

# AUTENTIFIKACIJA I AUTORIZACIJA BACKENDA JWT-OM

Prvo je potrebno instalirati jwt, pokretanjem slijedeće naredbe u terminalu:

npm install jsonwebtoken

I importati ga u ovom slučaju u „index.js“

const jwt = require("jsonwebtoken");

Slijedeći je korak unutar „index.js“ stranice kreirati login endpoint koji u slučaju naše aplikacije za aukcije izgleda ovako:

app.post("/login", function (req, res) {

  const data = req.body;

  const email = data.email;

  const password = data.password;

  connection.query("SELECT \* FROM korisnik WHERE email\_korisnika = ?", [email], function (err, result) {

    if (err) {

      res.status(500).json({ success: false, message: "Internal server error" });

    } else if (result.length > 0) {

      // Compare passwords

      bcrypt.compare(password, result[0].lozinka\_korisnika, function (err, bcryptRes) {

        if (bcryptRes) {

          // Generate JWT token

          const token = jwt.sign({ id: result[0].id\_korisnika, email: result[0].email\_korisnika, uloga: result[0].uloga }, config.secret);

          res.status(200).json({ success: true, message: "Login successful", token: token });

        } else {

          res.status(401).json({ success: false, message: "Invalid email or password " });

        }

      });

    } else {

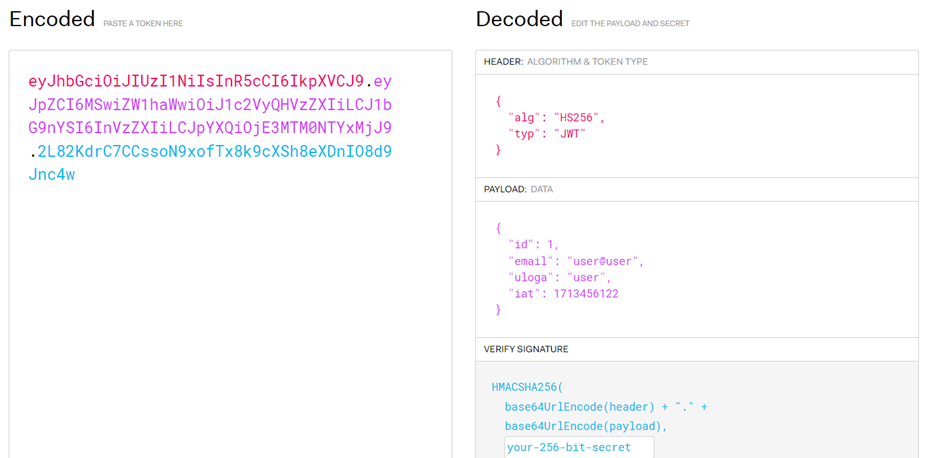
      res.status(401).json({ success: false, message: "Invalid email or password" });

    }

  });

});

Objašnjenje gornjeg koda: prvo se pošalje querry i usporede email i lozinka s podacima u bazi. bycrypt dio je za heashiranje lozinke. Nakon što je potrvrđeno da je korisnik unio dobre podatke kreira se token pomoću „jwt.sign()“ funkcije. Parametri u vitičastim zagradama su parametri koje želimo pohraniti u samom tokenu. To su obićno podaci kao id korisnika i uloga. Podaci koje token nosi se mogu vidjeti na ovoj poveznici: <https://jwt.io/> . I oni se mogu vidjeti u payload dijelu, slika ispod pokazuje to.



Slika JWT sadrzaj tokena

U jwt.sign se na kraju stavlja config.secret, odnosno skriveni ključ koji se koristi za enkriptiranje tokena. Za njega je potrebno kreirati novu datoteku, npr. „auth.config.js“ čiji sadržaj izgleda ovako:

module.exports = {

  secret: "NAPIŠI NEKI STRING ZA KLJUČ OVDJE",

};

Za Secret se može napsati bilo kakav string. Taj config je zatim potrebno importat u datoteku „index.js“

const config = require("../aukcije-server/auth.config.js");

Sada kada generiranje tokena radi potrebno je zaštiti endpointove da samo korisnici s validnim tokenima i oni koji imaju određenu ulogu mogu dobiti odgovore na svoje requestove. Zato prvo treba napraviti novu datoteku npr. „authJwt.js“ i u njoj napisati slijedeće:

const jwt = require("jsonwebtoken");

const config = require("../aukcije-server/auth.config.js");

verifyTokenAdmin = (req, res, next) => {

  /\*   let token = req.headers["x-access-token"]; \*/

  const authHeader = req.headers["authorization"];

  const token = authHeader && authHeader.split(" ")[1];

  if (!token) {

    return res.status(403).send({

      message: "No token provided!",

    });

  }

  jwt.verify(token, config.secret, (err, decoded) => {

    if (err) {

      return res.status(401).send({

        message: "Unauthorized!",

      });

    }

    req.userId = decoded.id;

    /\*     req.role = decoded.uloga; // Assign the role to the request object \*/

    if (decoded.uloga === "admin") {

      next();

    } else {

      res.status(403).send({

        message: "Require Admin Role!",

      });

    }

  });

};

verifyTokenUser = (req, res, next) => {

  /\*   let token = req.headers["x-access-token"]; \*/

  const authHeader = req.headers["authorization"];

  const token = authHeader && authHeader.split(" ")[1];

  if (!token) {

    return res.status(403).send({

      message: "No token provided!",

    });

  }

  jwt.verify(token, config.secret, (err, decoded) => {

    if (err) {

      return res.status(401).send({

        message: "Unauthorized!",

      });

    }

    req.userId = decoded.id;

  });

  next();

};

const authJwt = {

  verifyTokenAdmin: verifyTokenAdmin,

  verifyTokenUser: verifyTokenUser,

};

module.exports = authJwt;

U ovoj datoteci postoje dvije metode „verifyTokenAdmin()“ i „verifyTokenUser()“ Prva provjerava, prvo da li je token dan u headeru, zatim validnost tokena pomoću:

 jwt.verify(token, config.secret, (err, decoded)

Ako je validan onda još provijeri pomoću slijedeće if petlje ima li korisnik ulogu admin

if (decoded.uloga === "admin") {

      next();

    } else {

      res.status(403).send({

        message: "Require Admin Role!",

      });

    }

„verifyTokenUser()“ metoda samo provjerava valjanost tokena bez provjere uloge.

const jwt = require("jsonwebtoken");

const config = require("../backend-api/auth.config");

//PRI POZIVU FUNKCIJE ROLE NAPISAT OVAKO "verifyToken("admin, recepcionar")"

verifyToken = (roles) => (req, res, next) => {

  // Retrieve the token from the authorization header

  const authHeader = req.headers["authorization"];

  const token = authHeader && authHeader.split(" ")[1];

  // If no token is provided, return an error

  if (!token) {

    return res.status(403).json({

      message: "No token provided!",

    });

  }

  // Verify the token using the secret key

  jwt.verify(token, config.secret, (err, decoded) => {

    if (err) {

      return res.status(401).json({

        message: "Unauthorized!",

      });

    }

    // Assign the userId and role to the request object

    req.userId = decoded.id;

    const userRole = decoded.uloga;

    // Split the input roles string by commas and trim whitespace

    const rolesArray = roles.split(",").map((role) => role.trim());

    // Check if the user's role is in the list of provided roles

    if (rolesArray.includes(userRole)) {

      // Proceed to the next middleware or endpoint handler

      next();

    } else {

      // Send a 403 response if the user's role is not allowed

      res.status(403).json({

        message: `Require one of the following roles: ${roles}`,

      });

    }

  });

};

const authJwt = {

  verifyToken,

};

module.exports = authJwt;

U ovom primjeru se pri pozivu napiše ime uloge koja se traži pa je metoda fleksibilnija.

Kad su metode napisane potrebno ih je importati u „index.js“

const authJwt = require("../aukcije-server/authJwt.js");

Sada na endpointu koji želimo osigurati jwt tokenom samo pozovemo metodu za verifikaciju. Metodu pozivamo u samoj deklaraciji endpoita pomoću sa „authJwt.verifyTokenAdmin“. Na primjer na ovom endpointu se provjerava da li je korisnik admin:

app.get("/api/korisnici", authJwt.verifyTokenAdmin, (req, res) => {

  connection.query("SELECT id\_korisnika, ime\_korisnika, prezime\_korisnika, email\_korisnika, adresa\_korisnika FROM korisnik WHERE ime\_korisnika != 'obrisani' AND prezime\_korisnika != 'korisnik'", (error, results) => {

    if (error) throw error;

    res.send(results);

  });

});

## Frontend

U frontendu je potrebno prilikom logina spremiti token u localStorage browsera.

async login() {

      try {

        const response = await axios.post("http://localhost:3000/login", {

          email: this.email\_korisnika,

          password: this.lozinka\_korisnika,

        });

        if (response.data.success) {

          // Save the JWT token to local storage

          localStorage.setItem("token", response.data.token);

          // Redirect to the desired page

          this.$router.push("/Pocetna").then(() => {

            // Refresh the page

            window.location.reload();

          });

        } else {

          // Show error message if login fails

          this.$q.notify({

            color: "negative",

            position: "top",

            message: response.data.message,

            icon: "warning",

          });

        }

      } catch (error) {

        console.error("Login failed:", error);

        this.$q.notify({

          color: "negative",

          position: "top",

          message: "Prijava nije uspjela. Provjerite podatke i pokušajte ponovno.",

          icon: "warning",

        });

      }

    },

Nakon dobivenog pozitivnog odgovora dobiveni token se sprema u local storage pomoću: localStorage.setItem("token", response.data.token);

# DIZAJN SUSTAVA AUKCIJE

## Model podataka za Aukcije

A diagram of a company

Description automatically generated

Slika . Model podataka – EVA[[1]](#footnote-2)

**Tablica entiteta i njihovih opisa**

|  |  |
| --- | --- |
| ENTITETI | OPIS |
| **KORISNIK** | Korisnik aplikacije, njegovi atributi su: id\_korisnika (PK), ime\_korisnika, prezime\_korisnika, e-mail, lozinka, adresa\_korisnika, uloga |
| **PREDMET** | Predmet na aukciji, njegovi atributi su: id\_predmeta(PK), naziv\_predmeta, opis\_predmeta, vrijeme\_pocetka, vrijeme\_zavrsetka, pocetna\_cijena |
| **KATEGORIJA** | Kategorija predmeta, njeni atributi su: id\_kategorije(PK), naziv\_kategorije |
| **PONUDA** | Ponuda za predmet, njeni atributi su: id\_ponude(PK), vrijednost\_ponude, vrijeme\_ponude |
| **SLIKA** | Slike predmeta, njihovi atributi su: Id\_slike(PK), slika |
| **TRANSAKCIJA** | Povezuje korisnika, ponudu i predmet (history kupnji), atributi su: id\_transakcije(PK), iznos\_transakcije, vrijeme transakcije |

Tablica 1. Tablica entiteta i njihovih opisa

**Tablica atributa entiteta i njihovih opisa**

|  |  |
| --- | --- |
| ATRIBUTI ENTITETA KORISNIK | OPIS |
| **id\_korisnika(PK)** | Broj koji identificira korisnika, te je primarni ključ |
| **ime\_korisnika** | Atribut koji predstavlja ime korisnika |
| **prezime\_korisnika** | Atribut koji predstavlja prezime korisnika |
| **e-mail\_korisnika** | Atribut koji predstavlja e-mail adresu korisnika |
| **lozinka\_korisnika** | Atribut koji predstavlja lozinku korisnika |
| **adresa\_korisnika** | Atribut koji predstavlja adresu korisnika |
| ATRIBUTI ENTITETA KATEGORIJA | OPIS |
| **id\_kategorije(PK)** | Broj koji identificira kategoriju, te je primarni ključ |
| **naziv\_kategorije** | Atribut koji predstavlja ime kategorije |
| ATRIBUTI ENTITETA PREDMET | OPIS |
| **id\_predmeta(PK)** | Broj koji identificira predmet, te je primarni ključ |
| **naziv\_predmeta** | Atribut koji predstavlja ime predmeta |
| **opis\_predmeta** | Atribut koji predstavlja opis predmeta |
| **vrijeme\_pocetka** | Atribut koji predstavlja vrijeme početka |
| **vrijeme\_zavrsetka** | Atribut koji predstavlja vrijeme završetka |
| **pocetna\_cijena** | Atribut koji predstavlja početnu cijenu |
| **id\_korisnika(FK1)** | Vanjski ključ, povučen iz entiteta KORISNIK |
| **id\_kategorije(FK2)** | Vanjski ključ, povučen iz entiteta KATEGORIJA |
| ATRIBUTI ENTITETA PONUDA | OPIS |
| **id\_ponude(PK)** | Broj koji identificira ponudu, te je primarni ključ |
| **vrijednost\_ponude** | Atribut koji predstavlja vrijednost ponude |
| **vrijeme\_ponude** | Atribut koji predstavlja vrijeme ponude |
| **id\_korisnika(FK1)** | Vanjski ključ, povučen iz entiteta KORISNIK |
| **id\_predmeta(FK2)** | Vanjski ključ, povučen iz entiteta PREDMET |
| ATRIBUTI ENTITETA SLIKA |  |
| **id\_slike(PK)** | Broj koji identificira sliku, te je primarni ključ |
| **slika** | Spremljena je slika |
| **id\_predmeta(FK1)** | Vanjski ključ, povučen iz entiteta PREDMET |
| ATRIBUTI ENTITETA TRANSAKCIJA |  |
| **id\_transakcije(PK)** | Broj koji identificira transakciju, te je primarni ključ |
| **iznos\_transakcije** | Krajnja cijena kupljenog predmeta na aukciji |
| **vrijeme\_transakcije** | Vrijeme kad je predmet kupljen |
| **id\_korisnika(FK1)** | Vanjski ključ, povučen iz entiteta KORISNIK |
| **id\_predmeta(FK2)** | Vanjski ključ, povučen iz entiteta PREDMET |

Tablica . Tablica atributa i njihovih opisa

**Tablica veza i njihovih opisa**

|  |  |
| --- | --- |
| VEZE | OPIS |
| **KORISNIK - PREDMET** | Korisnik postavlja jedan ili više predmeta na aukciju, predmet je postavljen od strane jednog i samo jednog korisnika |
| **KORISNIK - TRANSAKCIJA** | Korisnik može napraviti 1 ili više transakcija, jedna transakcija je obavljena od strane jednog i samo jednog korisnika |
| **KORISNIK - PONUDA** | Korisnik daje jednu ili više ponuda, ponuda je dana od strane jednog i samo jednog korisnika |
| **KATEGORIJA - PREDMET** | Kategorija ima jedan ili više predmeta, predmet ima jednu i samo jednu kategoriju |
| **PREDMET - PONUDA** | Predmet sadrži jednu ili više ponuda, ponuda je dana za jedan i samo jedan predmet |
| **PREDMET - TRANSAKCIJA** | Jedan predmet se može nalaziti na jednoj ili više transakcija, jedna transakcija sadrži jedan i samo jedan predmet |
| **PREDMET - SLIKA** | Jedan predmet može pripadati jedna ili više slika, slika pripada jednom i samo jednom predmetu |

Tablica . Tablica veza i njihovih opisa

**Relacijski model podataka**

**Popis relacijskih shema**

KORISNIK(**id\_korisnika(PK), i**me\_korisnika, prezime\_korisnika, e-mail, lozinka, adresa\_korisnika, uloga)

KATEGORIJA(**id\_kategorije(PK),** naziv\_kategorije )  
PREDMET(**id\_predmeta(PK),** naziv\_predmeta, , opis\_predmeta, vrijeme\_pocetka, vrijeme\_zavrsetka, pocetna\_cijena, id\_korisnika(FK1), id\_kategorije(FK2))

PONUDA( **id\_ponude(PK),** vrijednost\_ponude, vrijeme\_ponude, id\_korisnika(FK1), id\_predmeta(FK2))

SLIKA( **id\_slike(PK),** slika, id\_predmeta (FK1))

TRANSAKCIJA**(id\_transakcije(PK**), iznos\_transakcije, vrijeme\_transakcije, id\_korisnika(FK1), id\_predmeta(FK2))

A diagram of a language

Description automatically generated

Slika . Relacijski model podataka

# Grafičko korisničko sučelje

Na slici 7 je prikazano sučelje za registraciju korisnika. Unosi se ime, prezime, adresa, email te lozinka te se korisnik registrira. Nakon registracije korisnik se može prijaviti unosom korisničkog imena i lozinke, preko sučelja koje je prikazano na slici 8.

A screenshot of a computer

Description automatically generated

Slika Registracija

A screenshot of a computer

Description automatically generated

Slika Prijava

Nakon uspješne prijave stvara se sesija preko JWT tokena. Ovisno o ulozi (user ili admin) u hamburger izborniku su dostupne različite stavke. Slika 9 u nastavku prikazuje slučaj ako je administrator prijavljen. Ako je prijavljen običan korisnik, prikazuje mu se sve isto osim gumba „admin dashboard“, ako korisnik nije prijavljen pojavljuju mu se gumbi „registracija, prijava, početna stranica, dodaj\_aukciju“ Neregistrirani korisnik ne može dodavati aukcije tako da će biti vraćen na početnu stranicu te dobiti prigodnu poruku o odbijenom pristupu.

A screenshot of a computer

Description automatically generated

Slika Hamburger menu

Klikom na gumb „Admin dashboard“ otvaraju se pod kategorije „ Kategorije“ i „pregled korisnika“ kao što prikazuje slika 10. Administrator može upravljati svim korisnicima, izmijeniti njihove korisničke podatke ili ih obrisati. Dostupan mu je i pregled svih korisnika. Osim korisnika administrator upravlja i kategorijama proizvoda, može dodavati nove kategorije, izmijeniti ili obrisati postojeće.

A screenshot of a computer

Description automatically generated

Slika Admin dashboard

Na slici 11 prikazana je početna stranica. Ovdje se prikazuju sve aukcije sa osnovnim informacijama – slika, naziv, početna cijena, vrijeme završetka aukcije, preostalo vrijeme aukcije te trenutna cijena. Na vrhu su prikazane moguće kategorije u koje se svrstavaju predmeti kod unosa aukcije. Sadrži pretraživanje i sortiranje po cijeni, abecedno i vremenu isteka

Klikom na kategoriju “Laptop” nude se sve aukcije koje su postavljene u toj kategoriji što se može vidjeti na slici 12.

A screenshot of a computer

Description automatically generated

Slika Početna stranica (prikaz aukcija)

A screenshot of a computer

Description automatically generated

Slika Prikaz predmeta iz kategorije “Laptop”

Slika 13 prikazuje sučelje prikaza aukcije na kojem se vide podaci – naziv proizvoda, slika, opis proizvoda, početna cijena proizvoda, početno vrijeme aukcije, završno vrijeme aukcije te trenutna cijena. Na dnu se nalazi gumb “Ponuda”. Klikom na njega otvara se opcija za povisiti cijenu aukcije može se zadati viša cijena po postocima u odnosu na početnu, ili ponuđivač sam bira iznos koji nudi, prikazano na slici 14.

A screenshot of a computer

Description automatically generated

Slika Prikaz pojedine aukcije

A screenshot of a computer

Description automatically generated

Slika Podizanje cijene predmeta na aukciji za određeni iznos

Slika 13 prikazuje sučelje za postavljanje aukcije. Ovdje se unosi naziv proizvoda, početna cijena, odabire se kategorija, datum i vrijeme početka i završetka aukcije, opis proizvoda te slika predmeta. Nudi se mogućnost “Postavi” koja sprema aukciju ili “Otkaži” koja vraća na početnu stranicu.

A screenshot of a computer

Description automatically generated

Slika Postavljanje aukcije

Na slici 14 prikazano je sučelje profila korisnika. Ovdje se vide podaci korisnika – ime i prezime, datum rođenja, te email korisnika. Ispod se nudi popis predmeta na aukciji koje je postavio taj korisnik. Za svaki predmet nudi se opcija “Uredi” za uređivanje ili “Obriši” za brisanje vlastite aukcije. Ispod vlastitih predmeta nalazi se popis predmeta na koje je prijavljeni korisnik licitirao, odnosno ponude

A screenshot of a computer

Description automatically generated

Slika Profil korisnika

Klikom na „izmjena korisničkih podataka“ otvara se prozor prikazan na slici 15. Korisnik vidi trenutne podatke i može unijeti nove te ih pohraniti klikom na gumb „izmijeni“

A screenshot of a computer

Description automatically generated

Slika Izmjena korisničkih podataka

# Kreiranje API-ja

Uloga serverskog dijela aplikacije je omogućiti komunikaciju između baze podataka i klijentskog dijela aplikacije odnosno omogućiti CRUD (engl. *Create, Read, Update, Delete*) funkcionalnosti putem RESTful arhitekture, te će izvršavati operacije spremanja podataka. Sam serverski dio aplikacije izgrađen je prema RESTful arhitekturi koristeći URI (engl. *Uniform Resource Identifier*) u kombinaciji s HTTP metodama POST, PUT, PATCH, DELETE i GET.

Za izradu serverskog dijela aplikacije korišten je razvojni okvir Express.js, te su same akcije definirane kao putanje (engl. *route*) koristeći HTTP metode. Prije implementacije krajnjeg rješenja potrebno definirane putanje koje će se koristiti za pristup resursima potrebnim za rad web-aplikacije. Krajnje putanje resursa prikazane su tablicom.

Tablica 1 - Krajnje putanje resursa

|  |  |  |
| --- | --- | --- |
| **URL** | **HTTP metoda** | **Akcija** |
| Resurs: Predmet na aukciji | | |
| **http://localhost:3000/** **all-predmet** | **GET** | **Dohvat svih predmeta na aukciji** |
| **http://localhost:3000/api/get-predmet2/:id** | **GET** | **Dohvat jednog predmeta na aukciji** |
| **http://localhost:3000/api/kategorijainfo :id** | **GET** | **Dohvat predmeta prema ID-u kategorije** |
| **http://localhost:3000/** **unosPredmeta** | **POST** | **Unos novog predmeta** |
| **http://localhost:3000/** **api/izmjenaPredmeta/:id** | **PUT** | **Izmjena predmeta** |
| **http://localhost:3000/api/brisanjePredmeta/:id** | **DELETE** | **Brisanje predmeta** |
| Resurs: Korisnik | | |
| **http://localhost:3000/api/korisnici** | **GET** | **Dohvat svih korisnika** |
| **http://localhost:3000/api/izmjenakorisnika/** | **PUT** | **Izmjena korisnika od strane administratora** |
| Resurs: Kategorija predmeta na aukciji | | |
| **http://localhost:3000/** **all-kategorija** | **GET** | **Dohvat svih kategorija** |
| [**http://localhost:3000/get/kategorija-predmet/:id**](http://localhost:3000/get/kategorija-predmet/:id) | **GET** | **Dohvat predmeta prema ID-u kategorije** |
| **http://localhost:3000/api/izmjenaKategorije** | **PUT** | **Izmjena kategorije od strane administratora** |
| **http://localhost:3000/api/deleteKategoriju/:id** | **DELETE** | **Brisanje kategorije** |
| Resurs: Ponuda | | |
| **http://localhost:3000/api/vlastita-ponuda-korisnik/:id** | **GET** | **Dohvat ponuda za određeni predmet** |
| **http://localhost:3000/ api/unostrenutnaponuda** | **POST** | **Unos trenutne ponude za predmet** |
|  |  |  |
| Resurs:Slika | | |
| **http://localhost:3000/api/dodavanjeSlika** | **POST** | **Unos slike predmeta** |
| **http://localhost:3000/api/brisanjeSlike/:id** | **DELETE** | **Brisanje slike** |
| Resurs:Trenutna cijena | | |
| **http://localhost:3000/get-predmet-trenutna-cijena/:id** | **GET** | **Dohvat trenutne cijene za za ponuđeni predmet** |
|  |  |  |

# IMPLEMENTACIJA SUSTAVA

## Prikaz svih predmeta na aukciji

Kako bi prikazali sve predmete na aukciji koristimo „mounted“ metodu koja popuni lokalnu varijablu „items“ sa rezultatima sql upita. „Mounted“ metode se izvršavaju prilikom svakog kreiranja vue komponente, te na taj način dobijamo osvježene podatke prilikom svakog posjeta toj stranici.

SQL upita (ovaj upit jos dodatno filtrira na način da prikazuje samo trenutne odnosno aktivne aukcije):

SELECT

        p.id\_predmeta,

        p.opis\_predmeta,

        p.naziv\_predmeta,

        p.pocetna\_cijena,

        p.vrijeme\_pocetka,

        p.vrijeme\_zavrsetka,

        CONCAT(

            FLOOR(TIMESTAMPDIFF(SECOND, NOW(), p.vrijeme\_zavrsetka) / (24 \* 3600)),

            ' dana, ',

            TIME\_FORMAT(

                SEC\_TO\_TIME(TIMESTAMPDIFF(SECOND, NOW(), p.vrijeme\_zavrsetka) % (24 \* 3600)),

                '%H:%i:%s'

            )

        ) AS preostalo\_vrijeme,

        (SELECT slika FROM slika WHERE id\_predmeta = p.id\_predmeta LIMIT 1) AS slika,

        COALESCE(MAX(po.vrijednost\_ponude), p.pocetna\_cijena) AS trenutna\_cijena

    FROM predmet p

    LEFT JOIN ponuda po ON p.id\_predmeta = po.id\_predmeta

    WHERE p.vrijeme\_zavrsetka > NOW()

    GROUP BY p.id\_predmeta

    ORDER BY preostalo\_vrijeme DESC;

Dio koda koji je zaslužan za slanje upita na backend i spremanje rezultata u lokalnu varijablu:

const baseUrl = "http://localhost:3000/api/";

export default {

  setup() {

    return {};

  },

  data() {

    return {

      Pretrazivanje: "",

      items: [],

      kategorija: [],

      selectedsortianje: "",

      sortiranje: [

        { label: "Cijena: manja prema većoj", value: "price-asc" },

        { label: "Cijena: veća prema manjoj", value: "price-desc" },

        { label: "Naziv: A do Z", value: "name-asc" },

        { label: "Naziv: Z do A", value: "name-desc" },

        { label: "Vrijeme isteka", value: "expiration" },

      ],

    };

  },

  mounted() {

    // Get the JWT token from local storage

    const token = localStorage.getItem("token");

    // Set up the request headers to include the JWT token

    const headers = { Authorization: `Bearer ${token}` };

    // Make the request to fetch all-predmet

    axios

      .get(baseUrl + "all-predmet", { headers })

      .then((response) => {

        this.items = response.data;

      })

      .catch((error) => {

        console.error("Error fetching all-predmet:", error);

        // Handle errors as needed

      });

    // Fetch all-kategorija without token

    axios

      .get(baseUrl + "all-kategorija", { headers })

      .then((response) => {

        this.kategorija = response.data;

      })

      .catch((error) => {

        console.error("Error fetching all-kategorija:", error);

        // Handle errors as needed

      });

  },

**TO TUDA JE NOVO!!!**

## Prikaz jednog predmeta na aukciji i postavljanje ponude

**SQL Upit za dohvat jednog predmeta preko Id-a.**

Sa SELECT-om označimo koje sve podatke trebamo dohvatiti

TIME\_FORMAT služi za formatiranje dohvaćenog vremena

TIMESTAMPDIFF se koristi za izračunavanje razlike između dvije vremenske jedinice

COALESCE ukoliko je neki član null coalesce vraća sljedeći

FROM koristimo kako bi znali na kojoj tablici se izvršava upit

LEFT\_JOIN neophodan je za dohvaćanje podataka iz više različitih tablica

WHERE uvjet, odnosno u našem slučaju usporedimo id koji šaljemo sa frontenda sa onima iz tablice

SELECT p.naziv\_predmeta, p.id\_predmeta, p.pocetna\_cijena, p.vrijeme\_pocetka, p.vrijeme\_zavrsetka, TIME\_FORMAT( SEC\_TO\_TIME(TIMESTAMPDIFF(SECOND, p.vrijeme\_pocetka, p.vrijeme\_zavrsetka)), '%H:%i:%s' ) AS preostalo\_vrijeme, p.opis\_predmeta, COALESCE(MAX(po.vrijednost\_ponude), p.pocetna\_cijena) AS vrijednost\_ponude, GROUP\_CONCAT(DISTINCT s.slika SEPARATOR '|||') AS slike

FROM predmet p

LEFT JOIN ponuda po ON p.id\_predmeta = po.id\_predmeta

LEFT JOIN slika s ON p.id\_predmeta = s.id\_predmeta

WHERE p.id\_predmeta = ?

GROUP BY p.id\_predmeta

Nakon što na backend stigne GET zahtjev na toj ruti *(/api/get-predmet/:id*), kod izvršava SQL upit na bazi podataka (phpMyAdmin) koristeći primarni ključ predmeta (*id*) koji je dobiven kao parametar. Upit dohvaća informacije o nazivu predmeta, šifri predmeta, slici, početnoj cijeni, vremenu početka, vremenu završetka, preostalom vremenu, opisu predmeta i vrijednosti najviše ponude za taj predmet.

Ako se prilikom izvršavanja SQL upita javi greška, bacit će exception (grešku). Ako nema greške, rezultati SQL upita će se poslati na frontend kao odgovor na zahtjev.

app.get("/api/get-predmet/:id", (req, res) => {

const { id } = req.params;

connection.query(

`SELECT p.naziv\_predmeta, p.id\_predmeta, p.pocetna\_cijena, p.vrijeme\_pocetka, p.vrijeme\_zavrsetka, TIME\_FORMAT( SEC\_TO\_TIME(TIMESTAMPDIFF(SECOND, p.vrijeme\_pocetka, p.vrijeme\_zavrsetka)), '%H:%i:%s' ) AS preostalo\_vrijeme, p.opis\_predmeta, COALESCE(MAX(po.vrijednost\_ponude), p.pocetna\_cijena) AS vrijednost\_ponude, GROUP\_CONCAT(DISTINCT s.slika SEPARATOR '|||') AS slike

FROM predmet p

LEFT JOIN ponuda po ON p.id\_predmeta = po.id\_predmeta

LEFT JOIN slika s ON p.id\_predmeta = s.id\_predmeta

WHERE p.id\_predmeta = ?

GROUP BY p.id\_predmeta`,

[id],

(error, results) => {

if (error) throw error;

if (results.length > 0 && results[0].slike) {

results[0].slike = results[0].slike.split("|||");

}

res.send(results);

}

);

});

*baseUrl* definira osnovni URL na kojem se nalazi API (u našem slučaju *http://localhost:3000/api*).

*data()* je funkcija koja vraća početno stanje podataka komponente. U ovom slučaju, item je prazan niz.

computed je svojstvo koje se koristi za definiranje reaktivnih svojstava koji ovise o drugim podacima.

*mounted()* je hook (metoda) koja se poziva kod samog renderiranja. Ovaj hook koristi Axios za izvršavanje HTTP GET zahtjeva prema određenom URL-u koji se sastoji od *baseUrl* i "*get-predmet/*" + *this.sifra\_predmeta*.

*this.sifra\_predmeta* je nepohodno slati u zahtjevu kako bi SQL upit znao koji nam točno podatci trebaju odnosno za koji predmet it baze.

const baseUrl = "<http://localhost:3000/api/>";

export default {

computed: {

id\_predmeta() {

return this.$route.query.id\_predmeta;

},

startingIndex() {

return 2;

},

},

data() {

return {

item: [],

showDialog: false,

odabranaCijena: "",

prices: [],

predmet: {

id\_ponude: null,

vrijednost\_ponude: null,

vrijeme\_ponude: null,

id\_korisnika: null,

id\_predmeta: null,

},

slike: [],

showSingleImage: false,

index: 1,

};

},

mounted() {

axios.get(baseUrl + "get-predmet/" + this.id\_predmeta, {}).then((response) => {

this.item = response.data[0];

this.item.trenutna\_cijena = this.item.pocetna\_cijena;

if (this.item.slike && this.item.slike.length > 0) {

if (this.item.slike.length === 1) {

// If there's only one image, showSingleImage should be true

this.showSingleImage = true;

this.item.slika = this.item.slike[0];

} else {

// If there are multiple images, showSingleImage should be false

this.showSingleImage = false;

}

}

});

axios.get(baseUrl + "get-predmet-trenutna-cijena/" + this.id\_predmeta, {}).then((response) => {

if (response.data.max\_vrijednost\_ponude != null) {

this.item.trenutna\_cijena = response.data.max\_vrijednost\_ponude;

}

this.prices = [

{

label: "+ 10%: " + (this.item.trenutna\_cijena \* 1.1).toFixed(2) + " $",

value: (this.item.trenutna\_cijena \* 1.1).toFixed(2),

},

{

label: "+ 20%: " + (this.item.trenutna\_cijena \* 1.2).toFixed(2) + " $",

value: (this.item.trenutna\_cijena \* 1.2).toFixed(2),

},

{

label: "+ 30%: " + (this.item.trenutna\_cijena \* 1.3).toFixed(2) + " $",

value: (this.item.trenutna\_cijena \* 1.3).toFixed(2),

},

{

label: "+ 40%: " + (this.item.trenutna\_cijena \* 1.4).toFixed(2) + " $",

value: (this.item.trenutna\_cijena \* 1.4).toFixed(2),

},

{

label: "+ 50%: " + (this.item.trenutna\_cijena \* 1.5).toFixed(2) + " $",

value: (this.item.trenutna\_cijena \* 1.5).toFixed(2),

},

{

label: "+ 100%: " + (this.item.trenutna\_cijena \* 2).toFixed(2) + " $",

value: (this.item.trenutna\_cijena \* 2).toFixed(2),

},

{

label: "Odaberi cijenu sam",

value: "Odaberi cijenu sam",

},

];

});

},

## Postavljanje ponude za predmet na aukciji (biddanje)

API (Application Programming Interface) je skup pravila i protokola koji omogućavaju komunikaciju između različitih softverskih komponenti. U kontekstu web razvoja, API se obično odnosi na sučelje koje omogućava komunikaciju između klijentske strane (frontend) i poslužiteljske strane (backend).

Komunikacija između frontend aplikacije (Vue.js) i API-ja (Express.js) obično se odvija putem HTTP zahtjeva. U našem primjeru HTTP GET zahtjev šalje se s frontend aplikacije za dohvaćanje podataka, a zatim se dobiveni rezultati prikazuju u sučelju korisniku. Također postoji HTTP POST zahtjev koji se šalje kada korisnik potvrdi ponudu.

Uzorak komunikacije između frontend i API-ja u našem primjeru uključuje sljedeće korake:

1. Na frontendu se koristi **axios** biblioteka za izvršavanje HTTP GET ili POST zahtjeva prema odgovarajućim API endpointima.
2. Kada se izvrši HTTP GET zahtjev, frontend prima odgovor s podacima iz API-ja, a ti podaci se mogu koristiti za ažuriranje korisničkog sučelja ili manipulaciju podacima.
3. Kada korisnik potvrdi ponudu, frontend šalje HTTP POST zahtjev s odgovarajućim podacima ponude na API endpoint **/unostrenutnaponuda**. API obrađuje taj zahtjev i vrši umetanje ponude u bazu podataka.

**Frontend kod**

<div class="q-pa-md flex flex-center">

<q-btn label="Ponuda" color="primary" @click="showDialog = true" />

<q-dialog v-model="showDialog">

<q-card style="width: 300px">

<q-card-section>

<div class="text-h6">Ponudi</div>

</q-card-section>

<q-card-section class="q-pt-none">

<q-select outlined v-model="odabranaCijena" :options="prices" label="Odaberi cijenu" />

</q-card-section>

<q-card-actions align="right" class="bg-white text-teal">

<q-btn flat label="Cancel" color="primary" v-close-popup />

<q-btn flat label="Potvrdi ponudu" @click="potvrdiPonudu" />

</q-card-actions>

</q-card>

</q-dialog>

</div>

Navedeni kod predstavlja HTML i JavaScript kod koji implementira dijalog za ponudu u okviru korisničkog sučelja. Ovaj kod koristi Quasar Framework za izradu responzivnog korisničkog interfejsa. Evo kratak opis dijela koda:

1. Događajni handler "onClick" na gumbu "Ponuda" otvara dijalog kada se klikne na njega.
2. Dijalog je definiran pomoću **<q-dialog>** komponente. Kada se postavi v-model atribut na true, dijalog će biti prikazan, a kada se postavi na false, dijalog će biti skriven.
3. **<q-card>** komponenta koristi se za definiranje kartice koja sadrži sadržaj dijaloga.
4. **<q-card-section>** komponenta se koristi za definiranje sekcija unutar kartice. Prva sekcija sadrži naslov "Ponudi".
5. **<q-select>** komponenta koristi se za prikazivanje padajućeg izbornika cijena. Vrijednost odabrane cijene bind-uje se na v-model atribut "odabranaCijena".
6. **<q-card-actions>** komponenta koristi se za definiranje akcija koje se nalaze na dnu kartice. U ovom slučaju, prikazuje se samo gumb "Potvrdi ponudu".
7. Događajni handler "onClick" na gumbu "Potvrdi ponudu" poziva funkciju "potvrdiPonudu".
8. JavaScript kod koji se nalazi ispod HTML dijela koristi Vue.js sintaksu za definiranje funkcija i varijabli koje se koriste u dijalogu.

methods: {

potvrdiPonudu() {

// Get the JWT token from local storage

const token = localStorage.getItem("token");

const headers = { Authorization: `Bearer ${token}` };

const decodedToken = jwtDecode(token);

console.log(this.odabranaCijena.value);

if (this.odabranaCijena) {

// Increase the current price based on the selected value

const selectedPrice = parseInt(this.odabranaCijena.value);

if (selectedPrice > this.item.trenutna\_cijena) {

const currentDate = new Date();

const formattedTime = `${currentDate.getFullYear()}-${currentDate.getMonth() + 1}-${currentDate.getDate()} ${currentDate.getHours()}:${currentDate.getMinutes()}:${currentDate.getSeconds()}`;

const podaciPonude = {

id\_predmeta: this.id\_predmeta,

vrijednost\_ponude: selectedPrice,

vrijeme\_ponude: formattedTime,

id\_korisnika: decodedToken.id,

};

axios

.post("http://localhost:3000/unostrenutnaponuda", podaciPonude, { headers })

.then((response) => {

console.log("New price stored successfully:", response.data);

// Handle the response data

})

.catch((error) => {

console.error("Error storing new price:", error);

// Handle the error

});

// Update the displayed price

this.item.trenutna\_cijena = selectedPrice;

// Close the dialog

this.showDialog = false;

}

}

},

Gore navedeni dio koda predstavlja Vue.js komponentu s definiranim metodama. Ovaj dio koda nadovezuje se na isječak koda prije njega. Kratak opis dodatnog dijela koda:

1. **potvrdiPonudu()** metoda: Ova metoda se poziva kada korisnik klikne na gumb "Potvrdi ponudu". Provjerava se je li **odabranaCijena** postavljena (nije **null**) te provjerava da li je nova cijena veća od trenutne. Ako je odabrana cijena veća od trenutne, njena vrijednost se dodjeljuje vrijednosti ponude, a zatim se postavlja **showDialog** na **false** kako bi se zatvorio dijalog.

Ovaj dio koda proširuje prethodni primjer s funkcionalnostima za dohvaćanje podataka putem Axiosa i prikazivanje odabrane cijene iz padajućeg izbornika.

**Backend kod**

const express = require("express");

const session = require("express-session");

const bcrypt = require("bcrypt");

const cors = require("cors");

const bodyParser = require("body-parser");

const mysql = require("mysql");

const { join } = require("path");

const path = require("path");

const multer = require("multer");

const upload = multer();

const jwt = require("jsonwebtoken");

const config = require("../aukcije-server/auth.config.js");

const authJwt = require("../aukcije-server/authJwt.js");

const app = express();

const port = 3000;

// Parser za JSON podatke

app.use(bodyParser.json());

// Parser za podatke iz formi

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

// Postavke direktorija za statičke datoteke

app.use(express.static(path.join(\_\_dirname, "public")));

app.use(cors({ origin: "\*" }));

const connection = mysql.createConnection({

host: "student.veleri.hr",

user: "iooa-aukcije",

password: "11",

database: "iooa-aukcije1",

});

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

connection.connect();

app.post("/unostrenutnaponuda", authJwt.verifyTokenUser, function (request, response) {

const data = request.body;

const ponuda = [[data.vrijednost\_ponude, data.vrijeme\_ponude, data.id\_korisnika, data.id\_predmeta]];

connection.query("INSERT INTO ponuda (vrijednost\_ponude, vrijeme\_ponude, id\_korisnika, id\_predmeta) VALUES ?", [ponuda], function (error, results, fields) {

if (error) throw error;

return response.send({ error: false, data: results, message: "Dodana je trenutna ponuda." });

});

});

app.get("/api/get-predmet-trenutna-cijena/:id", (req, res) => {

const { id } = req.params;

connection.query("SELECT MAX(`vrijednost\_ponude`) AS max\_vrijednost\_ponude FROM `ponuda` WHERE id\_predmeta = ?", [id], (error, results) => {

if (error) {

return res.status(500).send({ error: "Database error" });

}

if (results && results.length > 0) {

res.send({ max\_vrijednost\_ponude: results[0].max\_vrijednost\_ponude });

} else {

res.status(404).send({ message: "No offers found for the given id\_predmeta" });

}

});

});

app.listen(port, () => {

console.log("Server running at port: " + port);

});

kod predstavlja backend implementaciju aplikacije koristeći Express.js, CORS, Body Parser i MySQL biblioteke. Backend je odgovoran za obradu zahtjeva klijenta, komunikaciju s bazom podataka i slanje odgovora natrag klijentu.

Kratak opis navedenog koda:

1. Uvoz potrebnih biblioteka: `express`, `cors`, `body-parser`, `mysql` i `path`.

2. Inicijalizacija Express aplikacije i postavljanje porta na 3000.

3. Konfiguracija parsera za JSON podatke i podatke iz formi koristeći `body-parser` biblioteku.

4. Postavljanje direktorija za statičke datoteke pomoću `express.static` kako bi se poslužile javne datoteke.

5. Postavljanje CORS konfiguracije pomoću `cors` biblioteke kako bi se omogućili zahtjevi s različitih domena.

6. Stvaranje MySQL konekcije prema određenoj bazi podataka s postavkama za host, korisnika, lozinku i bazu podataka.

7. Pozivanje metode `connect()` kako bi se uspostavila veza s bazom podataka.

8. Definiranje POST endpointa "/unostrenutnaponuda" koji obrađuje zahtjeve za unos trenutne ponude. Podaci se dohvaćaju iz zahtjeva i koriste se za izvršavanje SQL upita putem `connection.query()` metode.

9. Definiranje GET endpointa "/api/get-predmet-trenutna-cijena/:id" koji dohvaća određeni predmet i njegovu trenutnu cijenu na temelju ID-a predmeta. Izvršava se SQL upit koji kombinira podatke iz tablice "predmet" i "ponuda" koristeći lijevu vezu. Rezultati se šalju kao odgovor.

10. Pokretanje servera na određenom portu i ispis poruke u konzoli.

Ovaj backend kod omogućava obradu zahtjeva klijenta, komunikaciju s bazom podataka i slanje odgovora natrag klijentu. Važno je napomenuti da se u kodu koriste postavke kao što su host, korisnik, lozinka i baza podataka koje treba prilagoditi specifičnostima projekta. Također, treba provjeriti i osigurati ispravnu konfiguraciju MySQL konekcije.

Opisani kod prikazuje implementaciju dijela korisničkog sučelja i backend logike za komunikaciju s bazom podataka. Ovdje je kratak opis komunikacije između frontenda i API-ja:

1. Frontend komponenta:
   * Događajni handler "onClick" na gumbu "Ponuda" otvara dijalog kada se klikne na njega.
   * Korisnik odabire cijenu iz padajućeg izbornika.
   * Događajni handler "onClick" na gumbu "Potvrdi ponudu" poziva funkciju "potvrdiPonudu".
2. API endpointi:
   * GET "/api/get-predmet-trenutna-cijena/:id" - Dohvaća podatke o određenom predmetu i njegovu trenutnu cijenu na temelju ID-a predmeta.
   * POST "/unostrenutnaponuda" - Unosi trenutnu ponudu u bazu podataka.
3. Frontend komunikacija s API-jem:
   * Kada se komponenta učita (mounted metoda), frontend šalje GET zahtjev prema "/api/get-predmet-trenutna-cijena/:id" endpointu kako bi dobio podatke o odabranom predmetu.
   * Kada korisnik potvrdi ponudu (klikne na gumb "Potvrdi ponudu"), frontend šalje POST zahtjev prema "/unostrenutnaponuda" endpointu s podacima o ponudi (id\_predmeta, vrijednost\_ponude, vrijeme\_ponude, id\_korisnika iz tokena).
4. Backend komunikacija s bazom podataka:
   * Backend prima GET zahtjev na "/api/get-predmet-trenutna-cijena/:id" endpointu i izvršava SQL upit koji dohvaća podatke o predmetu i trenutnoj cijeni iz baze podataka. Rezultati se šalju kao odgovor.
   * Backend prima POST zahtjev na "/unostrenutnaponuda" endpointu i izvršava SQL upit koji unosi trenutnu ponudu u bazu podataka. Ovisno o rezultatu upita, šalje se odgovor natrag frontendu.

**ADAPTIRANO DO OVDJE - emm**

## Unos predmeta za aukciju

API za **postavljanje aukcije** je realiziran putem POST metode na **/unosPredmeta** endpointu.

POST metodu koristimo kada nešto želimo postaviti i/ili ažurirati u bazi podataka. U ovome je to slučaju postavljanje predmeta na aukciju, što znači kako se određeni podaci postavljaju u bazu podataka tj. Nova aukcija, odnosno predmet.

API Endpoint:

* POST /unosPredmeta

**API za unos predmeta na aukciju je implementiran kroz POST rutu koja je definirana u ovom dijelu koda (PostaviAukciju.vue – middleware)**

app.post("/unosPredmeta", upload.none(), authJwt.verifyTokenUser, function (request, response) {

const data = request.body;

const predmet = [[data.naziv\_predmeta, data.opis\_predmeta, data.vrijeme\_pocetka, data.vrijeme\_zavrsetka, data.pocetna\_cijena, data.id\_korisnika, data.id\_kategorije]];

connection.query("INSERT INTO predmet (naziv\_predmeta, opis\_predmeta, vrijeme\_pocetka, vrijeme\_zavrsetka, pocetna\_cijena, id\_korisnika, id\_kategorije) VALUES ?", [predmet], function (error, results) {

if (error) throw error;

const insertedPredmetId = results.insertId;

const keys = Object.keys(data).filter(key => key.startsWith("file"));

let completed = 0;

if (keys.length === 0) {

return response.send({ error: false, message: "Predmet dodan bez slika.", insertedPredmetId: insertedPredmetId });

}

keys.forEach((key) => {

const base64String = data[key];

connection.query("INSERT INTO slika (slika, id\_predmeta) VALUES (?, ?)", [base64String, insertedPredmetId], function (error) {

if (error) throw error;

completed++;

if (completed === keys.length) {

return response.send({ error: false, message: "Predmet i slike su dodani.", insertedPredmetId: insertedPredmetId });

}

});

});

});

});

* 1. **OPIS POST RUTE/API-ja**
  2. ***'App.post'*** – nova HTTP POST ruta
  3. ***'const data'*** = request.body;' - : Ovdje se podaci iz tijela POST zahtjeva (koji su poslani sa klijenta) spremaju u konstantu "data".
  4. ***'predmet = [[data.naziv\_predmeta, ...]]' –*** objekt unutar kojega se spremaju podaci iz requesta sa frontenda
  5. ***'*** ***if (error) throw error;' -*** Ako se prilikom izvršavanja SQL upita dogodi greška.
  6. ***'*** ***return response.send({error: false, message: "Predmet i slike su dodani.", insertedPredmetId: insertedPredmetId });' -*** Kada SQL upit uspješno izvrši, server će odgovoriti na klijentov zahtjev slanjem JSON objekta koji sadrži rezultate upita i poruku koja potvrđuje da je predmet uspješno dodan. U slučaju greške, ovaj dio linije koda neće biti uopće dosegnut.

**SQL upit koji se koristi za unos novog predmeta je:**

*INSERT INTO predmet (naziv\_predmeta, opis\_predmeta, vrijeme\_pocetka, vrijeme\_zavrsetka, pocetna\_cijena, id\_korisnika, id\_kategorije) VALUES ?*

**2.1. OPIS SQL UPITA**

Ovaj SQL upit koristi **INSERT INTO** naredbu, koja se koristi za dodavanje novog zapisa u tablicu **predmet**. Evo što točno svaka komponenta upita znači:

* ***'INSERT INTO predmet'***: Ovaj dio govori SQL-u da ćemo dodavati novi zapis u tablicu nazvanu **predmet**.
* ***'(naziv\_predmeta, opis\_predmeta, vrijeme\_pocetka, vrijeme\_zavrsetka, pocetna\_cijena, id\_korisnika, id\_kategorije)'***: Ovaj dio navodi imena stupaca u tablici **predmet** u koje će se unijeti podaci.
* **'VALUES ?'**: Ovo je parametrizirani dio upita koji predstavlja vrijednosti koje će biti unesene u svaki od gore navedenih stupaca ( *naziv\_predmeta, opis\_predmeta..*.). Umjesto konkretnih vrijednosti, postavljen je upitnik **?**. Taj upitnik omogućuje dodavanje vrijednosti iz varijabli ili iz drugih izvora podataka, odnosno upitnik će biti zamijenjen nizom vrijednosti koji su dobiveni iz POST zahtjeva, odnosno unosom podataka na frontendu, onim redoslijedom kako je definirano u SQL upitu, također ovakav način pisanja SQL upita šiti od mogućeg SQL injectiona.

**Komunikacija frontenda s API-jem:**

Kada korisnik želi postaviti aukciju, korisnik popunjava formu na frontend-u (PostaviAukciju.vue). Nakon što se forma ispunjena, aplikacija šalje POST zahtjev na **/unosPredmeta** endpoint s potrebnim podacima za kreiranje aukcije. Ovi podaci uključuju **naziv\_predmeta**, **opis\_predmeta**, **vrijeme\_pocetka**, **vrijeme\_zavrsetka**, **pocetna\_cijena**, **id\_korisnika** i **id\_kategorije**.

Na backendu, nakon što se primi POST zahtjev, podaci se izvlače iz tijela zahtjeva. Zatim se podaci koriste za izvršavanje SQL upita koji unosi novi redak u tablicu **predmet**.

**Unos slike/a za predmet**

Unutar iste metode provjerava se da li je korisnik odabrao slike za unos u frontendu tada ovisno o tome da li korisnik želi unesti slike za predmet ili ne izvodi se kod za unos slika unutar tablice ***slika***. Ukoliko je korisnik odabrao unos slike tada se izvršava sljedeći SQL upit:

INSERT INTO slika (slika, id\_predmeta) VALUES (?, ?)

Vrijednosti koje se unose su base64String za spremanje slike unutar baze podataka i id unesenog predmeta. Bitno je da se referenciramo na točan id predmeta kako ne bi unesli sliku za krivi predmet. Ukoliko korisnik ne želi unesti sliku za taj predmet ovaj SQL upit se preskače.

Nakon uspješnog izvršavanja upita, server šalje odgovor natrag frontendu s informacijama o rezultatima operacije. Frontend zatim koristi te informacije za obavijest korisnika o uspješnosti operacije.

**DO OVDJE - emm**

1. [↑](#footnote-ref-2)