Bazy Danych - projekt

Księgarnia internetowa

Mateusz Waga i Witold Strzeboński

1. Schematy modeli

1.1 Book:

```
const bookSchema=new Schema({
    title:{type:String},
    author:{type:String},
    description:{type:String},
    onStock:{type:Number},
    photo:{type:String},
    price:{type:Number},
    opinions:[opinion]
},{timestamps:true})
```

```
const opinion=new Schema({
    content:{type:String},
    rating:{type:Number},
    author:{type:String}
})
```

1.2 User:

```
const userSchema = new Schema({
    email: {type: String},
    password: {type: String},
    name: {type: String},
    isAdmin: {type: Boolean},
    history: [historyElement],
    address:{
        street:{type:String},
        number:{type:String},
        place:{type:String}
    }
}, {timestamps: true})
```

```
const historyElement=new Schema({
    bookId:{type:Object},
    price:{type:Number},
    quantity: {type:Number},
    date: {type:Date}
})
```

2. Przykładowe dane

2.1 User


```
_id: ObjectId('64887d3f1b54f6e90a892cf2')
email: "mateuszwaga@docs.name"
password: "$2a$10$jwOuRs4YMCyGymv1CSQrQO6PGJgVWOSkKoI7GlF.Mv593Tjx3X.Nm"
name: "Mateusz"
isAdmin: false
history: Array
  0: Object
     bookId: ObjectId('648797cf1eb6e443ca1a3bbc')
     price: 40
     quantity: 3
     date: 2023-06-13T14:30:06.476+00:00
     _id: ObjectId('64887d6e1b54f6e90a892cff')
  1: Object
address: Object
   street: "Polna"
   number: "6"
   place: "Warszawa"
createdAt: 2023-06-13T14:29:19.078+00:00
updatedAt: 2023-06-13T14:30:06.478+00:00
__v: 0
```

2.2 Book


```
_id: ObjectId('648797cf1eb6e443ca1a3bbc')
title: "Pan Tadeusz"
author: "Adam Mickiewicz"
description: "Pan Tadeusz, czyli Ostatni zajazd na Litwie – poemat epicki Adama Mick…"
onStock: 29
photo: "uploads\file-1686607823469-818370400"
price: 40
▾ opinions: Array
  ▾ 0: Object
      content: "Polecam książkę"
      rating: 3
      author: "nick1"
      _id: ObjectId('6488778d1b54f6e90a892ca8')
  ▾ 1: Object
      content: "Dobra lektura na ciężkie wieczory"
      rating: 5
      author: "nick2"
      _id: ObjectId('648877b61b54f6e90a892caa')
  ▾ 2: Object
      content: "Odradzam zdecydowanie"
      rating: 1
      author: "nick3"
      _id: ObjectId('648877c21b54f6e90a892cac')
  ▸ 3: Object
createdAt: 2023-06-12T22:10:23.472+00:00
updatedAt: 2023-06-13T14:06:07.499+00:00
__v: 0
```

3. Funkcje

3.0 importy

```
controllers > JS AuthController.js > [e] refreshToken > [M]
const User = require('../models/User')
const bcrypt = require('bcryptjs')
const jwt = require('jsonwebtoken')
const { token } = require('morgan')
```

```
const Books=require("../models/Books")
const Users=require("../models/User")
```

3.1 getAll

zwraca wszystkie książki z bazy

```
const getAll=(req,res,next)=>{
    Books.find()
    .then(response=>{
        res.json({
            ...response
        })
    })
    .catch(err=>{
        res.json({
            message:"Error!"
        })
    })
}
```

3.2 getOne

zwraca książkę o podanym id

```
const getOne=(req,res,next)=>{
    const id=req.body._id
    Books.findById(id)
    .then(response=>{
        res.json({
            ...response._doc
        })
    })
    .catch(err=>{
        res.json({
            message:"Error!"
        })
    })
}
```

3.3 add

dodaje książkę do bazy danych

```
const add=(req,res,next)=>{
    // console.log(req)
    let book=new Books({
        title:req.body.title,
        author:req.body.author,
        description:req.body.description,
        onStock:req.body.onStock,
        photo:req.body.photo? req.body.photo:"",
        price:req.body.price,
        opinions:[]
    })
    if(req.file){
        book.photo=req.file.path
    }
    book.save()
    .then(response=>{
        res.json({...response._doc})
    })
    .catch(err=>{
        res.json({message:"Error!"})
    })
}
```

3.4 update

aktualizuje książkę o podanym id

```
const update=(req,res,next)=>{
    console.log(req)
    const id=req.body._id
    let book={
        title:req.body.title,
        author:req.body.author,
        description:req.body.description,
        onStock:req.body.onStock,
        photo:req.body.photo,
        price:req.body.price
    }
    if(req.file){
        book.photo=req.file.path
    }
    Books.findByIdAndUpdate(id,{$set:book})
    .then(response=>{
        res.json({...book,_id:id})
    })
    .catch(err=>{
        res.json({message:"Error!"})
    })
}
```

3.5 destroy

usuwa książkę o podanym id

```
const destroy=(req,res,next)=>{
    let id=req.body._id
    Books.findByIdAndRemove(id)
    .then(response=>{
        res.json({
            message:"Delete succeed!"
        })
    })
    .catch(err=>{
        res.json({message:"Error!"})
    })
}
```

3.6 addOpinion

dodaje opinie do książki o podanym id

```
const addOpinion=(req,res,next)=>{
    let id=req.body._id
    let opinion={
        content:req.body.content,
        rating:req.body.rating,
        author:req.body.author
    }
    Books.updateOne({_id:id},{$push:{opinions:opinion}})
    .then(response=>{
        res.json({...opinion})
    })
    .catch(err=>{
        res.json({message:"Error!"})
    })
}
```

3.7 buy

kupuje książki z koszyka

```
const buy = (req, res, next) => {
    const promises = []
    for(let book of req.body){
        const promise = Books.findById(book._id)
        promises.push(promise)
    }
    Promise.all(promises)
    .then(responses => {
        for(let i in responses){
            if(responses[i].onStock < req.body[i].quantity){
                return res.json({
                    message: "Not enough units on stock",
                    id: responses[i]._id,
                    onStock: responses[i].onStock
                })
            }
        }

        for(let book of req.body){
            Books.findByIdAndUpdate(book._id, {$inc: {onStock: - book.quantity}}).then()
        }
        next()
    })
    .catch(err => {res.json({message: err})})
}
```

3.8 addToHistory

dodaje zakupione książki do historii użytkownika

```
const addToHistory = (req, res, next) => {
    const date = new Date()
    for(const book of req.body){
        const single = {
            bookId:new mongoose.Types.ObjectId(book._id),
            price: book.price,
            quantity: book.quantity,
            date: date
        }
        Users.updateOne({email: req.user.name}, {$push:{history:single}}).then()
    }
    res.json({message: "Success"})
}
```

3.9 getHistory

zwraca historię zakupów użytkownika o podanym id

```
const getHistory = (req, res, next) => {
    Users.aggregate([
        {$match:{
            email: req.user.name
        }},
        {$unwind:"$history"},
        {$addFields: {
           bookIds: "$history.bookId"
        }},
        {$lookup:{
            from:"books",
            localField:"bookIds",
            foreignField:"_id",
            as:"data"
        }},
        {$unwind:"$data"},
        {$project:{
            name:"$data.name",
            title:"$data.title",
            author:"$data.author",
            price:"$history.price",
            date:"$history.date",
            photo:"$data.photo",
            quantity:"$history.quantity"
        }}])
    .then(response => {
        console.log(response)
        res.json({
            data: [...response],
            message: "Success"
        })
    })
    .catch(err => {
        res.json({message: "Error"})
    })
}
```

3.10 getWithFilters

zwraca książki spełniające wymogi zadane przez filtry

```
const getWithFilters=(req,res,next)=>{
    let filters={}
    let sort={}
    if(req.body.author){
        filters["author"]=req.body.author
    }
    if(req.body.from){
        filters["price"]={$gt:req.body.from}
    }
    if(req.body.to){
        filters["price"]={...filters["price"],$lt:req.body.to}
    }
    if(req.body.sort){
        sort[Object.keys(req.body.sort)[0]]=Object.values(req.body.sort)[0]
    }
    // console.log(filters)
    Books.find(filters,{}).sort(sort)
    .then(response=>{
        res.json({
            ...response
        })
    })
    .catch(er=>{
        res.json({message:"Error!"})
    })
}
```

3.11 register

rejestruje użytkownika na serwerze

```
const register = (req, res, next) => {
    User.findOne({email: req.body.email})
    .then(user => {
        if(user) {
            res.json({
                message: 'Email is already used',
                result: 'email'
            })
        }
        else {
            bcrypt.hash(req.body.password, 10, function(err, hashedPass) {
                if(err) {
                    res.json({error: err})
                }
                else {
                    let user = new User({
                        email: req.body.email,
                        password: hashedPass,
                        name: req.body.name,
                        isAdmin: false,
                        history:[],
                        address:{
                            street:req.body.street,
                            number:req.body.number,
                            place:req.body.place
                        }
                    })

                    user.save()
                    .then(user => {
                        res.json({
                            message: 'User added successfully!',
                            result: 'registered'
                        })
                    })
                    .catch(error => {
                        res.json({
                            message: 'Error!',
                            result: 'error'
                        })
                    })
                }


            })
        }
    })
    .catch(error=>{
        res.json({
            message: 'Error!',
            result: 'error'
        })
    })
}
```

3.12 logIn

weryfikuje użytkownika oraz generuje refresh token i access token

```
const logIn = (req, res, next) => {
    var email = req.body.email
    var password = req.body.password

    User.findOne({email: email})
    .then(user => {
        if(user) {
            bcrypt.compare(password, user.password, function(err, result) {
                if(err) {
                    res.json({
                        message: 'Error!',
                        result: 'error'
                    })
                }

                if(result) {
                    let token = jwt.sign({name: user.email}, 'secretToken', {expiresIn: '2s'})
                    let refreshToken = jwt.sign({name: user.email}, 'secretRefreshToken', {expiresIn: '24h'})

                    res.json({
                        message: 'Logged in successfully!',
                        token: token,
                        refreshToken: refreshToken,
                        name: user.name,
                        isAdmin: user.isAdmin,
                        result: 'loggedIn'
                    })
                }
                else {
                    res.json({
                        message: 'Wrong password!',
                        result: 'password'
                    })
                }
            })
        }
        else {
            res.json({
                message: 'No user found!',
                result: 'user'
            })
        }
    })
}
```

3.13 getBestSeller

zwraca id najchętniej kupowanej książki

```
const getBestSeller=(req,res,next)=>{
    Users.aggregate([
        {$unwind:"$history"},
        {$addFields: {
           bookIds: "$history.bookId"
        }},
        {$lookup:{
            from:"books",
            localField:"bookIds",
            foreignField:"_id",
            as:"data"
        }},
        {$unwind:"$data"},
        {$group:{
            _id:"$data._id",
            counter:{
                $sum: "$history.quantity"
            }
        }},
        {$sort:{
           counter: -1
        }},
        {$limit: 1},
        {$project:{
            counter: 0
        }}])
        .then(response=>{
            res.json({...response[0]})
        })
        .catch(er=>{
            res.json({message:"Error!"})
    })
}
```

4. Server(Express.js)

4.1 Połączenie z bazą i konfiguracja serwera:

```
const express =  require('express')
const mongoose = require('mongoose')
const morgan = require('morgan')
const bodyPraser = require('body-parser')

const cors=require("cors")

mongoose.connect('mongodb+srv://admin:admin@kurs.hwy4owx.mongodb.net/Projekt_bazy?retryWrites=true&w=majority')
const db= mongoose.connection
const booksRouter=require("./routes/Book.js")
const authRouter = require("./routes/Auth.js")

db.on('error',(err)=>{
    console.log(err)
})

db.once('open',()=>{
    console.log("connect dataBase")
})


const app=express()
app.use(cors())

app.use(morgan('dev'))
app.use(bodyPraser.urlencoded({extended:true}))
app.use(bodyPraser.json())
app.use("/uploads",express.static('uploads'))

const PORT = process.env.port || 3000
app.listen(PORT,()=>{
    console.log(`server running on port ${PORT}`)
})

app.use('/api/books',booksRouter);
app.use('/api/users', authRouter)
```

4.2 Routes

4.2.1 Auth

```
const express = require('express')
const router = express.Router()
const AuthController = require('../controllers/AuthController')

router.post('/register', AuthController.register)
router.post('/login', AuthController.logIn)
router.post('/refresh-token', AuthController.refreshToken)

module.exports = router
```

4.2.2 Book

```
const express = require("express")
const router = express.Router()

const BooksController=require("../controllers/BookController")
const upload = require("../middleware/upload")
const authenticate = require("../middleware/authenticate")
const checkAdminRights = require("../middleware/checkAdminRights")

router.get('/', BooksController.getAll)
router.post('/getOne', BooksController.getOne)
router.post('/add', authenticate, checkAdminRights, upload.single('file'), BooksController.add)
router.post('/destroy', authenticate, checkAdminRights, BooksController.destroy)
router.post('/update', authenticate, checkAdminRights , upload.single('file'), BooksController.update)
router.post('/addOpinion', BooksController.addOpinion)
router.post('/buy', authenticate, BooksController.buy, BooksController.addToHistory)
router.get('/getHistory', authenticate, BooksController.getHistory)
router.post('/getWithFilters',BooksController.getWithFilters)
router.get('/getBestseller',BooksController.getBestSeller)
module.exports = router
```

4.3 Funkcje pomocnicze

4.3.1 storage

funkcja odpowiada za ścieżkę gdzie będą lądować zdjęcia dotyczące książek

```
const path=require("path")
const multer=require("multer")

var storage=multer.diskStorage({
    destination:(req,file,cb)=>{
        cb(null,'uploads/')
    },
    filename:(req,file,cb)=>{
        const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9)
        cb(null, file.fieldname + '-' + uniqueSuffix)
    }
})
```

4.3.2 upload

funkcja odpowiada za dodawanie zdjęcia do serwera

```
var upload=multer({
    storage:storage,
    fileFilter:(req,file,callback)=>{
        if(file.mimetype=="image/png"||file.mimetype=="image/jpg"||file.mimetype=="image/jpeg"){
            callback(null,true)
        }else{
            console.log("Only jpg & png")
            callback(null,false)
        }
    },
    limits:{
        fileSize: 1024*1024*2
    }
})
```

4.3.3 refreshToken

na podstawie refresh tokena generuje nowy access token

```
const refreshToken = (req, res, next) => {
    const refreshToken = req.body.refreshToken
    jwt.verify(refreshToken, 'secretRefreshToken', function(err, decode) {
        if(err) {
            res.json({
                message: 'Error!',
                result: 'error'
            })
        }
        else {
            let token = jwt.sign({name: decode.name}, 'secretToken', {expiresIn: '1h'})
            res.json({
                message: 'Token refreshed',
                token: token,
                refreshToken: refreshToken,
                result: 'refreshed'
            })
        }
    })
}
```

4.3.4 authenticate

sprawdza czy użytkownik jest zalogowany

```
const authenticate = (req, res, next) => {
    try {
        const token = req.headers.authorization.split(' ')[1]
        console.log(token)
        const decode = jwt.verify(token, 'secretToken')
        req.user = decode
        next()
    }
    catch(error) {
        if(error.name == "TokenExpiredError"){
            res.json({message: 'Token expired!'})
        }
        else {
            res.json({message: 'Authentication failed!'})
        }
    }
}
```

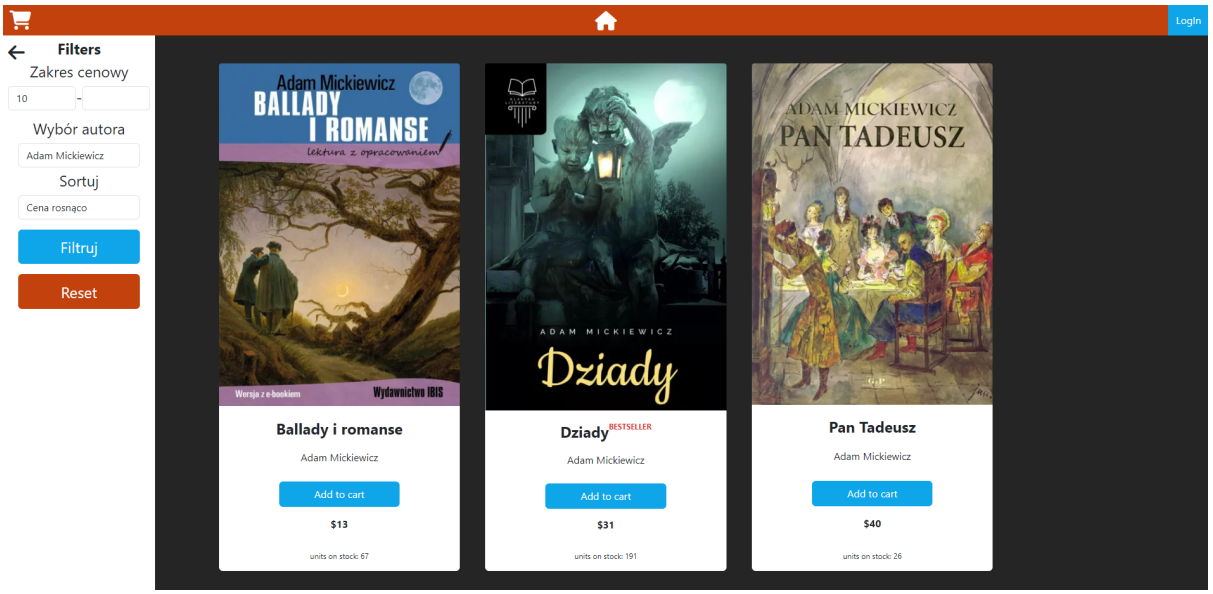
4.3.5 checkAdminRights

sprawdza czy użytkownik jest adminem

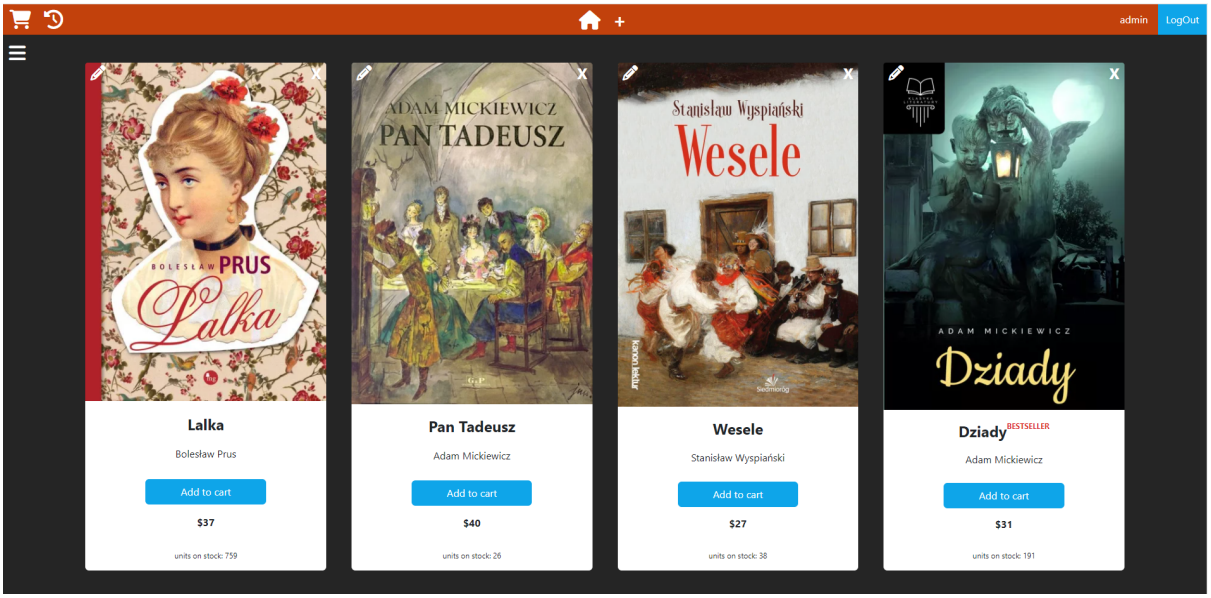
```
const checkAdminRights = (req, res, next) => {
    Users.findOne({email: req.user.name})
    .then(response => {
        if(response._doc.isAdmin){
            next()
        }
        else{
            res.json({message: "No admin rights"})
        }

    })
    .catch(err => {
        res.json({message: "Error"})
    })
}
```

5. Frontend (React.js)

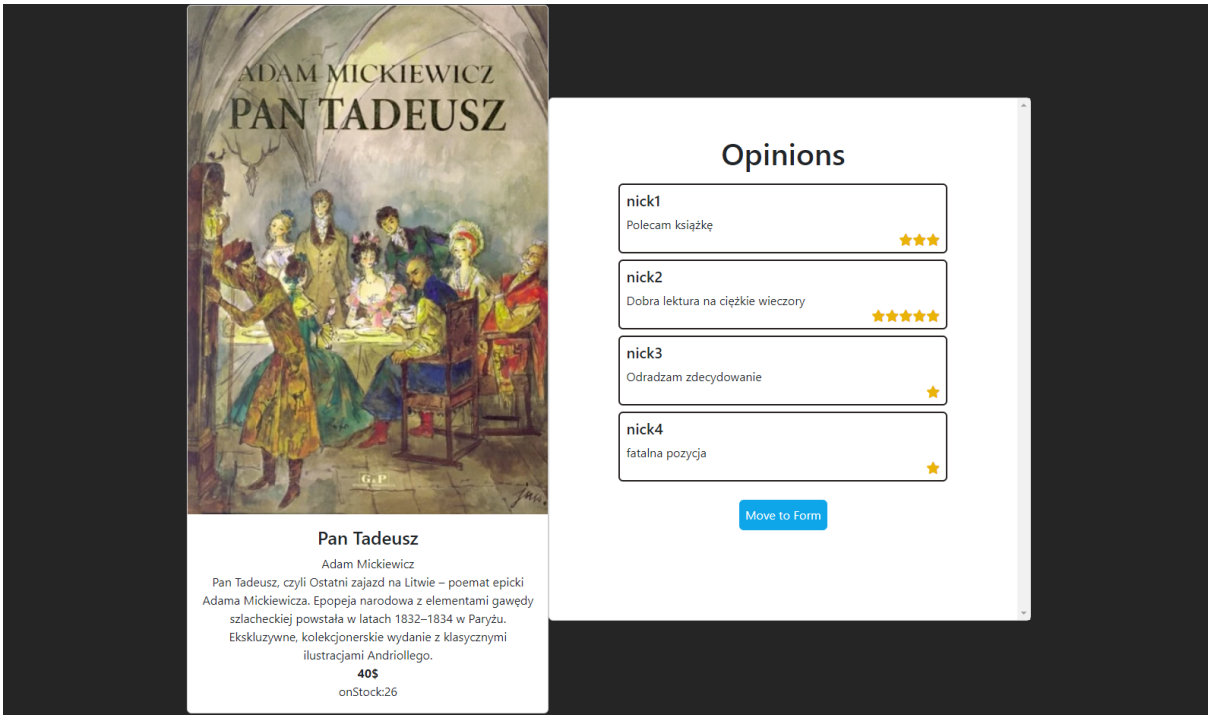
5.1 Strona główna



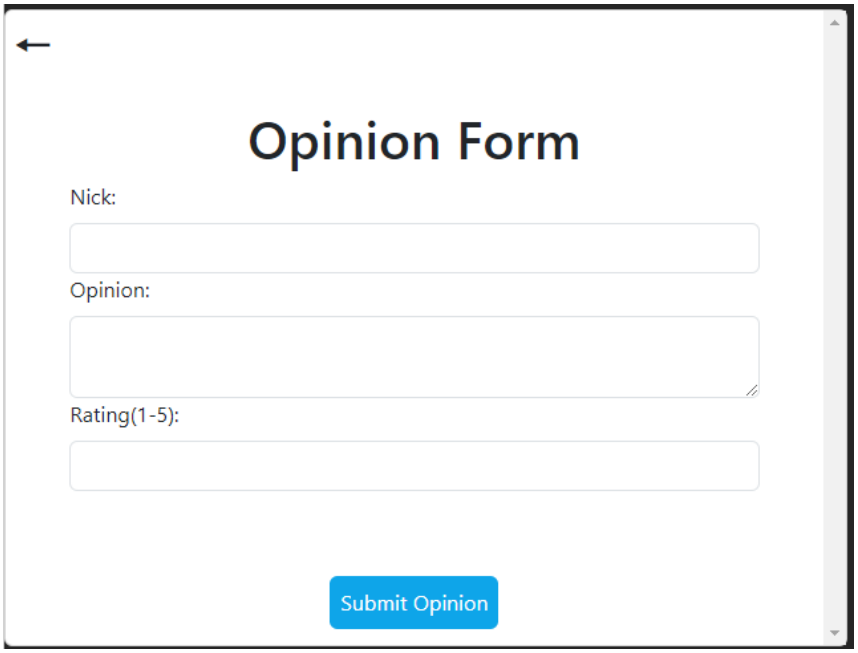
5.2 Widok admina



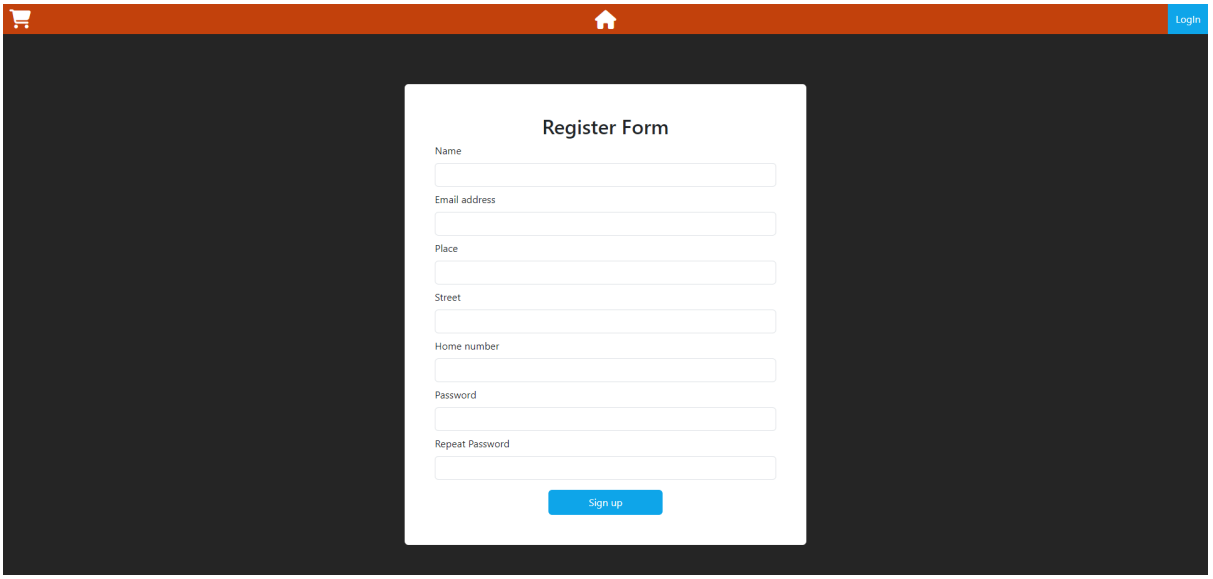
5.3 Widok pojedyńczej wycieczki



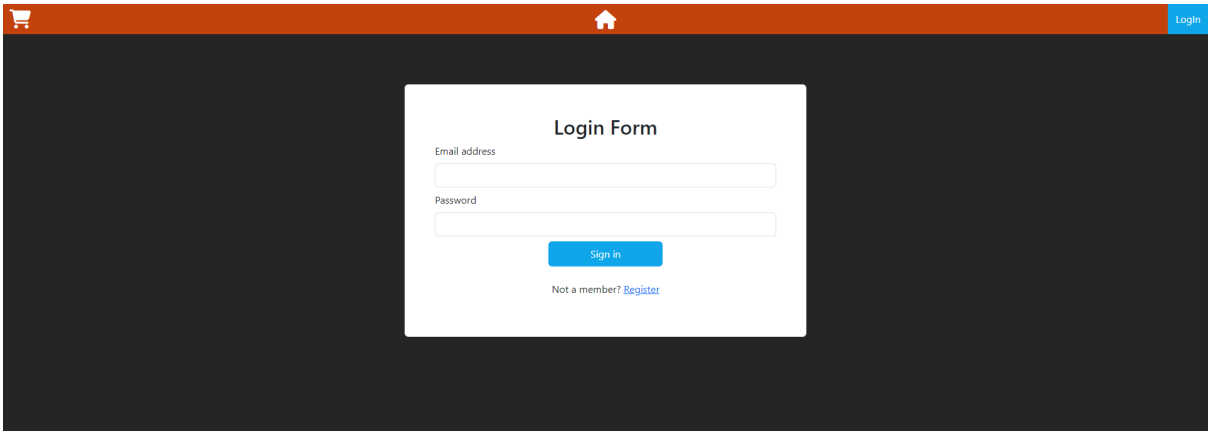
5.4 Formularz dodawania opinii



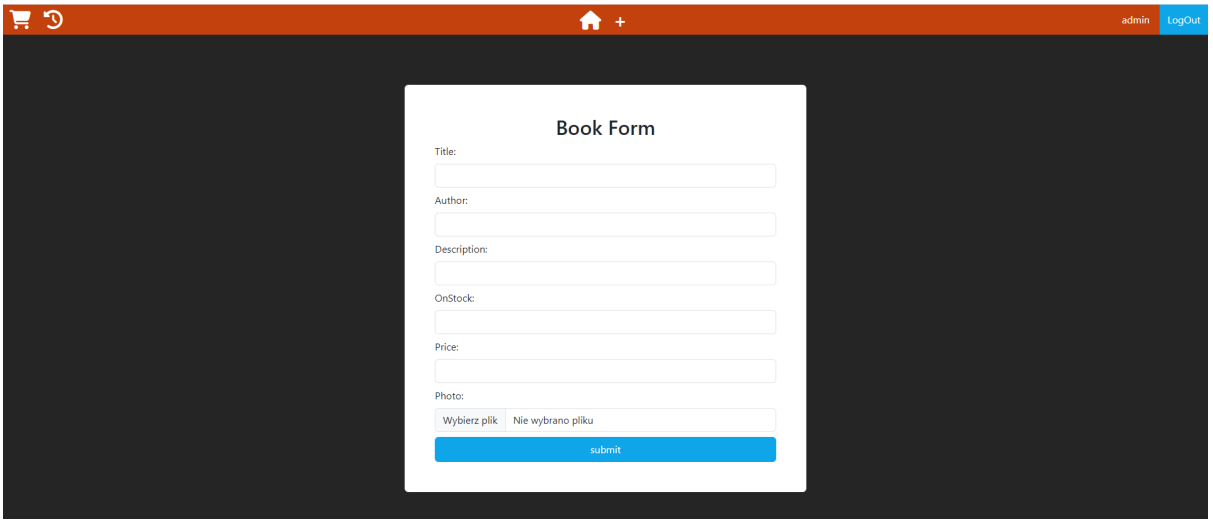
5.5 Rejestracja



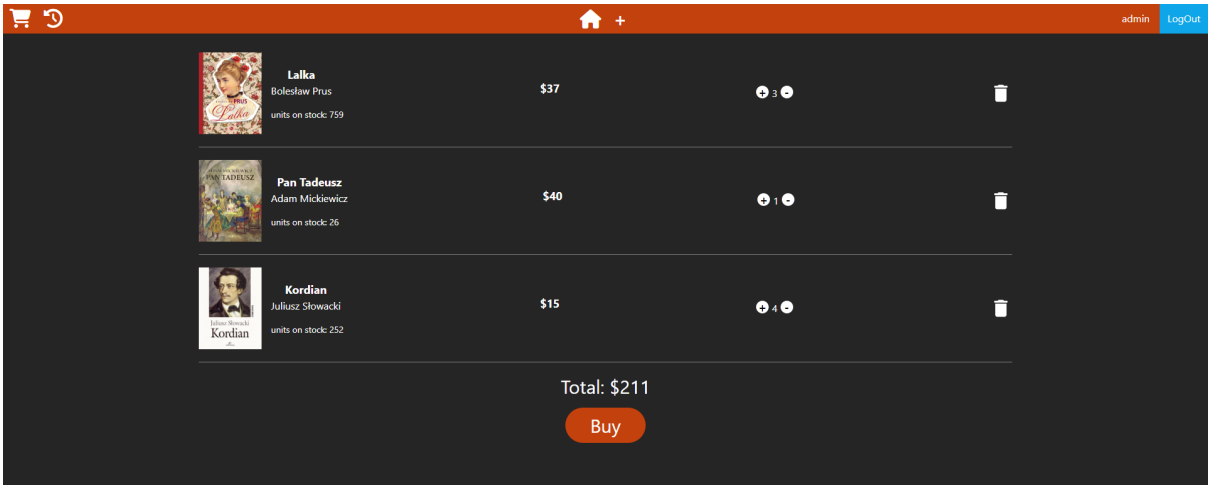
5.6 Logowanie



5.7 Formularz dodawania i edycji wycieczki



5.8 Koszyk



5.9 Historia zamówień użytkownika