

Bazy Danych - projekt

Księgarnia internetowa

Mateusz Waga i Witold Strzeboński



A G H

Opis aplikacji

Tematem naszej aplikacji jest księgarnia internetowa. Dla użytkownika dostępne są następujące funkcjonalności: rejestracja, logowanie, zakup książki, przeglądanie historii, dodawanie opinii oraz filtrowanie książek. Natomiast admin ma dodatkowo możliwość dodawania, edytowania i usuwania książek z bazy.

Technologie

- Baza danych: Mongodb
- Backend: Node.js + Express.js
- Frontend: React.js

1. Schemat bazy danych

1.1 Struktura bazy danych

Podczas tworzenia struktury staraliśmy się wykorzystać możliwość bazy MongoDB poprzez zagnieźdżenie dokumentów (embedding). Jednakże zastosowaliśmy też podejście tabelaryczne (referencing) ze względu na dużą redundancję danych. Podejście embedding występuje w kolekcji "Books" poprzez zagnieźdzenie opinii dodawanych przez użytkowników do książek oraz w kolekcji "Users" poprzez zagnieźdzenie adresu użytkownika. Natomiast podejście referencing zastosowaliśmy przy kolekcji "Users" w tablicy history, która przechowuje obiekty zawierające id zakupionych książek użytkownika oraz niezbędne informacje dotyczące każdego zakupu. Dzięki zastosowaniu takiego podejścia nie musimy dla każdego zakupionego produktu przechowywać w historii takich danych jak tytuł, autor i link do zdjęcia kosztem bardziej złożonych zapytań.

1.2 Kolekcja "Books"

title: tytuł książki,
author: autor książki
description: opis książki
onStock: liczba książek na stanie
photo: link do zdjęcia książki
price: cena książki
opinions: tablica zawierające opinie użytkowników na temat książki
 content: treść opini
 rating: ocena od 1 do 5
 author: autor opinii

```
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}
})
```

1.3 Kolekcja “Users”:

email: e-mail użytkownika

password: zaszyfrowane hasło użytkownika

name: nazwa użytkownika

isAdmin: zawiera informację, czy użytkownika ma prawa admina

history: tablica zawierająca książki zakupione przez użytkownika:

bookId: id książki

price: cena, po której zakupiono książkę

quantity: liczba zakupionych egzemplarzy książki

date: data zakupu

address: adres zamieszkania

street: ulica

number: numer domu

place: miejscowość

```
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 Schema = mongoose.Schema
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$10$jw0uRs4YMCyGymv1CSQrQ06PGJgVWOSkKoI7GlF.Mv593Tjx3X.Nm"
name: "Mateusz"
isAdmin: false
- history: Array
  ▾ 0: Object
    bookId: ObjectId('648797cf1eb6e443cala3bbc')
    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('648797cf1eb6e443caba3bbc')
title: "Pan Tadeusz"
author: "Adam Mickiewicz"
description: "Pan Tadeusz, czyli Ostatni zajazd na Litwie – poemat epicki Adama Mickiewicza opisujący powstanie listopadowe. Jeden z najważniejszych dzieł literatury polskiej i europejskiej."
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 > [+] refreshToken > [+] JM  
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

Funkcja 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

Funkcja 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

Funkcja 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

Funkcja 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

Funkcja 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

Funkcja 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

Funkcja najpierw sprawdza, czy w bazie jest wystarczająca liczba egzemplarzy każdej książki zamówionej przez klienta. Jeżeli nie to zwraca komunikat oraz aktualny stan książki niespełniającej warunku. Jeżeli wszystkie zamówione książki spełniają warunek na liczbę egzemplarzy, to funkcja akceptuje transakcję klienta, aktualizuje stan książek w bazie i przechodzi do kolejnej funkcji - addToHistory.

```
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

Jeżeli operacja kupowania książek zakończy się sukcesem, to funkcja dodaje do historii zakupów użytkownika obiekty zawierające id książki, cenę, liczbę egzemplarzy oraz datę zakupu.

```
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

Funkcja jest wywoływana za każdym razem, gdy użytkownik przejdzie do historii. Aby pobrać wszystkie potrzebne informacje o zakupionych książkach funkcja robi joina kolekcji Books do kolekcji Users po id każdej książki.

```
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

Funkcja zwraca książki spełniające filtry zadane przez użytkownika. Aby to wykonać tworzy 2 obiekty: jeden odpowiedzialny za filtry, a drugi - za sortowanie, które są wykorzystywane przy zapytaniu find i sort.

```
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

Funkcja sprawdza, czy podany e-mail nie jest już wykorzystywany przez innego użytkownika. Jeżeli nie, to w oparciu o dane podane w formularzu tworzy nowego użytkownika i dodaje go do bazy.

```
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

Funkcja sprawdza, czy istnieje użytkownik o podanym adresie e-mail i porównuje podane hasło z zaszyfrowanym hasłem w bazie. Jeżeli wszystko się powiedzie, to metoda generuje refresh token i access token, które będą potrzebne przy wykonywaniu autentykowanych operacji i zwraca je wraz z informacjami o użytkowniku.

```
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

Funkcja zwraca id najczęściej kupowanej książki, czyli książki, która została sprzedana w największej liczbie egzemplarzy.

```
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. Serwer (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 ustawia ścieżkę do zdjęcia.

```
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

The screenshot shows the main homepage of a bookstore. On the left, there is a sidebar with a shopping cart icon, a 'Filters' button, and several dropdown menus for price range (10-100), author (Adam Mickiewicz), and sorting (Sortuj, Cena rosnąco). Below these are 'Filtruj' and 'Reset' buttons. The main area displays three book cards:

- Ballady i romanse** by Adam Mickiewicz (Wydawnictwo IBIS) - \$13, units on stock: 67. Buttons: Add to cart.
- Dziady** by Adam Mickiewicz (BESTSELLER) - \$31, units on stock: 191. Buttons: Add to cart.
- Pan Tadeusz** by Adam Mickiewicz (G.P.) - \$40, units on stock: 26. Buttons: Add to cart.

At the top right, there are 'Login' and 'Logout' buttons.

5.2 Widok admina

The screenshot shows the administrator's view of the bookstore. At the top right, there are 'admin' and 'LogOut' buttons. The main area displays four book cards:

- Lalka** by Bolesław Prus - \$37, units on stock: 759. Buttons: Add to cart.
- Pan Tadeusz** by Adam Mickiewicz - \$40, units on stock: 26. Buttons: Add to cart.
- Wesele** by Stanisław Wyspiański - \$27, units on stock: 38. Buttons: Add to cart.
- Dziady** by Adam Mickiewicz (BESTSELLER) - \$31, units on stock: 191. Buttons: Add to cart.

At the top left, there is a menu icon (three horizontal lines).

5.3 Widok pojedyńczej wycieczki

The screenshot shows a travel itinerary card for the book 'Pan Tadeusz' by Adam Mickiewicz. At the top is the title 'ADAM MICKIEWICZ PAN TADEUSZ' above a painting of a group of people at a table. Below the painting, the text 'Pan Tadeusz' is repeated, followed by 'Adam Mickiewicz'. A detailed description follows: 'Pan Tadeusz, czyli Ostatni zajazd na Litwie – poemat epicki Adama Mickiewicza. Epopeja narodowa z elementami gawędy szlacheckiej powstała w latach 1832–1834 w Paryżu. Ekskluzywne, kolekcjonerskie wydanie z klasycznymi ilustracjami Andriollego.' It also mentions '40\$' and 'onStock:26'. To the right is a sidebar titled 'Opinions' containing four user reviews:

nick1	Polecam książkę	★★★
nick2	Dobra lektura na ciężkie wieczory	★★★★★
nick3	Odradzam zdecydowanie	★
nick4	fatalna pozycja	★

[Move to Form](#)

5.4 Formularz dodawania opinii

The screenshot shows the 'Opinion Form' page. At the top is a back arrow icon. The title 'Opinion Form' is centered. Below it are three input fields: 'Nick:' with a text input field, 'Opinion:' with a text input field, and 'Rating(1-5):' with a text input field. At the bottom is a blue 'Submit Opinion' button.

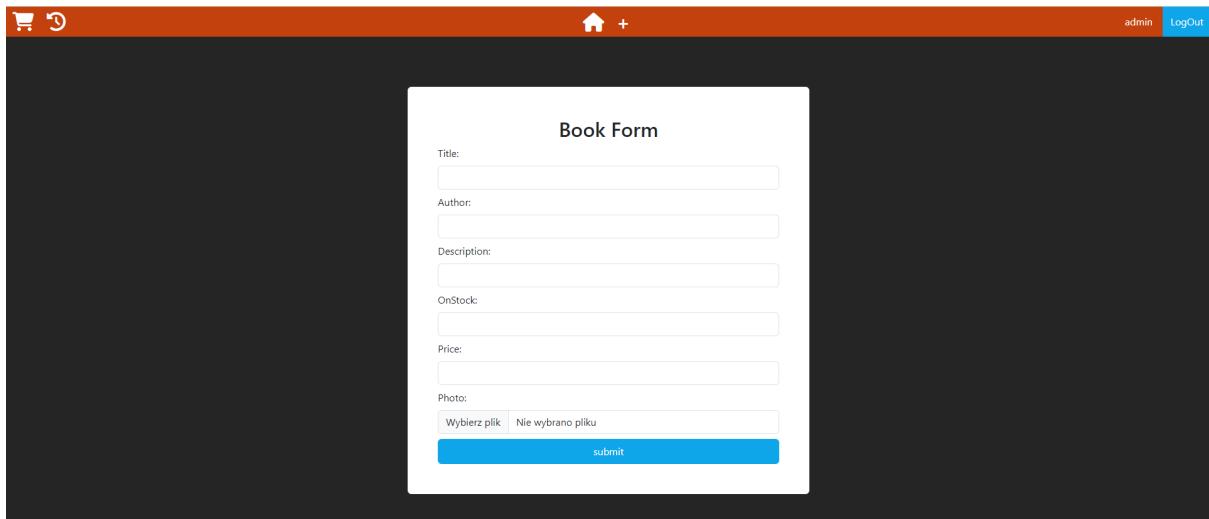
5.5 Rejestracja

The screenshot shows a registration form titled "Register Form". The form consists of several input fields: Name, Email address, Place, Street, Home number, Password, and Repeat Password. A blue "Sign up" button is located at the bottom right of the form. The background of the page is dark, and there are navigation icons at the top.

5.6 Logowanie

The screenshot shows a login form titled "Login Form". It contains two input fields: Email address and Password. Below these fields is a blue "Sign in" button. At the bottom of the form, there is a link that says "Not a member? [Register](#)". The background of the page is dark, and there are navigation icons at the top.

5.7 Formularz dodawania i edycji wycieczki



The screenshot shows a 'Book Form' input field. It contains fields for Title, Author, Description, OnStock, Price, and Photo. There is also a file selection input labeled 'Wybierz plik' (Select file) and a button labeled 'submit'.

5.8 Koszyk

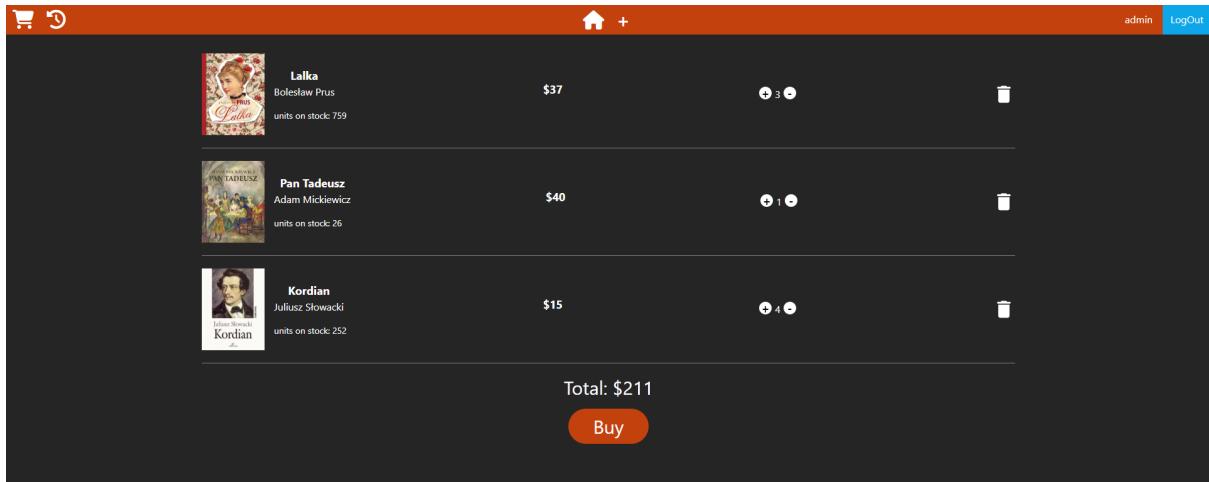


Image	Title	Author	Price	Quantity	Remove
	Lalka	Bolesław Prus	\$37	+ 3	
	Pan Tadeusz	Adam Mickiewicz	\$40	+ 1	
	Kordian	Juliusz Słowacki	\$15	+ 4	

Total: \$211 [Buy](#)

5.9 Historia zamówień użytkownika

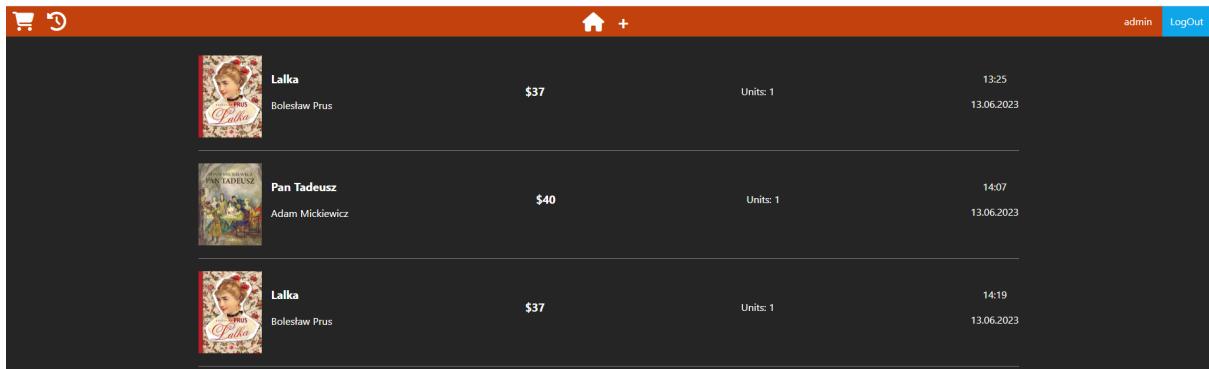


Image	Title	Author	Price	Units	Date
	Lalka	Bolesław Prus	\$37	1 Units: 1	13:25 13.06.2023
	Pan Tadeusz	Adam Mickiewicz	\$40	1 Units: 1	14:07 13.06.2023
	Lalka	Bolesław Prus	\$37	1 Units: 1	14:19 13.06.2023