WYPOŻYCZALNIA KSIĄŻEK

Projekt - Bazy Danych 2

Marcin Żurawel
Jakub Świątek

Aplikacja służąca do wypożyczania książek na określoną liczbę dni. Umożliwia rejestrację oraz logowanie się użytkowników oraz umożliwia zaplanowanie wypożyczenia oraz potwierdzenie go w odpowiednim miejscu w aplikacji. Udostępnia również panel admina do śledzenia wypożyczeń użytkowników.
Aplikacja korzysta z technologii NodeJS, ExpressJS do backendu oraz frameworka React do obsługi części klienckiej.
Użyta baza danych to MongoDB.

1. Kolekcje

a) products (server/src/models/product.ts) - kolekcja przechowująca dokumenty reprezentujące produkty (książki)

1. **_id** pole typu string z dokładnym id pozycji w bazie
2. **name** pole typu string opisujące tytuł książki
3. **description** pole typu string; krótki opis pozycji; autor
4. **quantity** dostępna ilość pozycji w wypożyczalni
5. **imageUrl** ścieżka do zdjęcia okładki danej pozycji

```ts
import { Document, Schema, model } from "mongoose";

interface Product extends Document {
    _id: string;
    name: string;
    description?: string;
    quantity: number;
    imageUrl?: string;
}

const productSchema = new Schema<Product>({
    name: { type: String, required: true },
    description: { type: String, required: false },
    quantity: { type: Number, required: true },
    imageUrl: {
        type: String,
        required: false,
    },
});

const ProductModel = model<Product>("Product", productSchema);

export { Product, ProductModel };
```

b) rentals (server/src/models/rental.ts) - kolekcja przechowująca informacje o wypożyczeniu

1. **_id** dokładne id wypożyczenia w bazie
2. **clientId** id wypożyczającego klienta
3. **productId** id wypożyczonego produktu
4. **quantity** ilość wypożyczonych produktów na raz
5. **idPending** czy oczekuje na potwierdzenie
6. **borrowDate** dokładny czas wypożyczenia
7. **dueDate** data zwrotu produktu
8. **fine** naliczona kara
9. **ifProlonged** czy przedłużone
10. **productName** nazwa wypożyczonego produktu

```typescript
import { Document, Schema, model } from "mongoose";

interface Rental extends Document {
    _id: string;
    clientId: Schema.Types.ObjectId;
    productId: Schema.Types.ObjectId;
    productName: string;
    quantity: number;
    isPending: boolean;
    borrowDate?: Date;
    dueDate?: Date;
    returnDate?: Date;
    fine?: number;
    ifProlonged?: boolean;
}

const rentalSchema = new Schema<Rental>({
    clientId: {
        type: Schema.Types.ObjectId,
        ref: "User",
        required: true,
    },
    productId: {
        type: Schema.Types.ObjectId,
        ref: "Product",
        required: true,
    },
    productName: { type: String, required: true },
    quantity: { type: Number, required: true },
    isPending: { type: Boolean, required: true },
    borrowDate: { type: Date, required: false },
    dueDate: { type: Date, required: false },
    returnDate: { type: Date, required: false },
    fine: { type: Number, default: 0 },
    ifProlonged: { type: Boolean, default: false },
});

const RentalModel = model<Rental>("Rental", rentalSchema);

export { Rental, RentalModel };
```

c) users (server/src/models/user.ts) - kolekcja przechowująca dane do użytkownikach
   1. **_id** dokładne id użytkownika w bazie
   2. **username** unikalna nazwa użytkownika używana do rejestracji i logowania
   3. **password** hasło do logowania użytkownika
   4. **role** rola użytkownika w systemie (user | admin)

```ts
import { Schema, model } from "mongoose";

interface User extends Document {
    _id: string;
    username: string;
    password: string;
    role: "admin" | "user";
}

const userSchema = new Schema<User>({
    username: { type: String, required: true },
    password: { type: String, required: true },
    role: { type: String, enum: ["admin", "user"], default: "user" },
});

const UserModel = model<User>("User", userSchema);

export { UserModel, User };
```

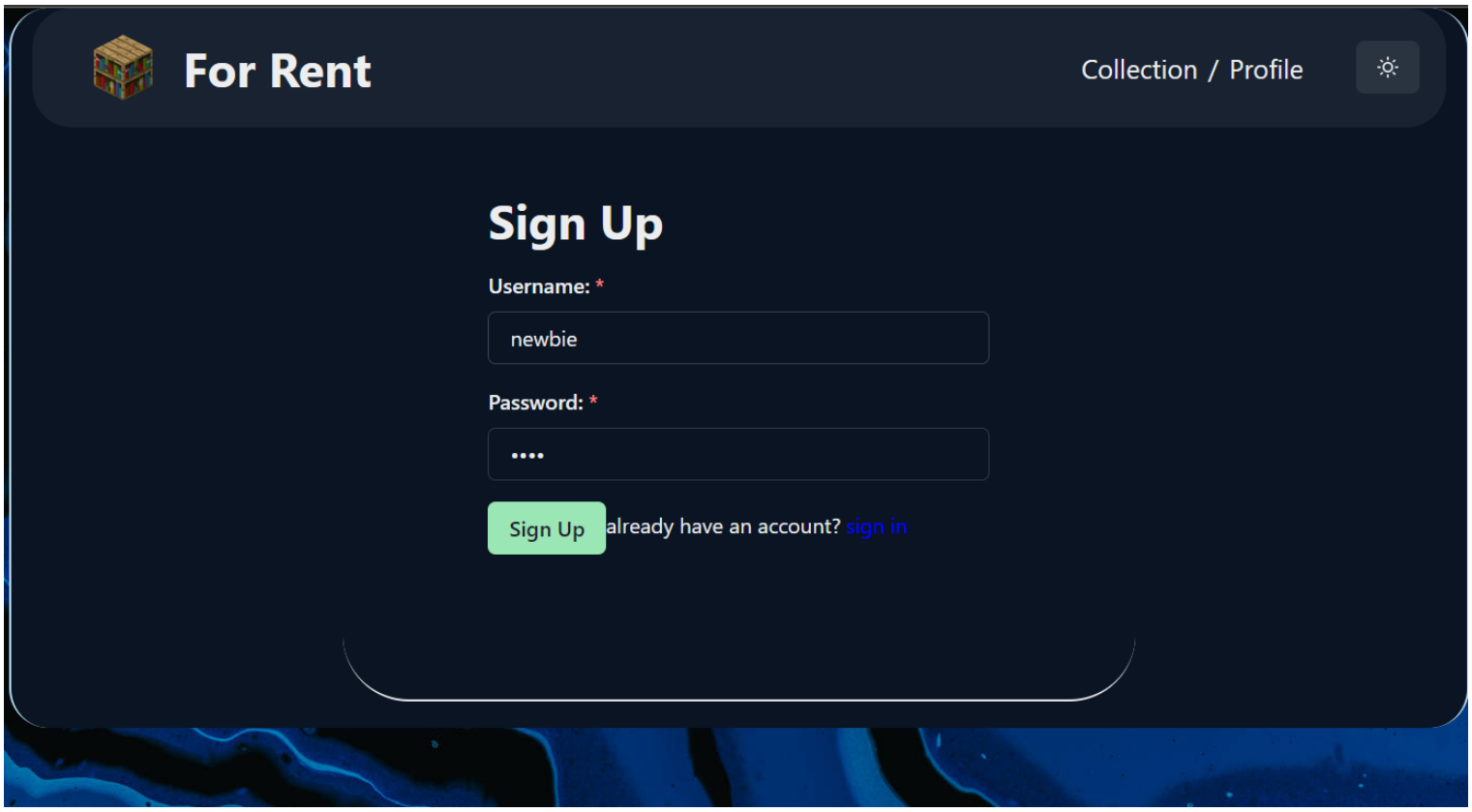
2. Endpointy

1) Route'y użytkownika
    a) /user/signup
    b) /user/sigin
    c) /user/rentals/:id

```
import { Router, Request, Response } from "express";
import { getUserRentals, signin, signup } from "../controllers/userController";
const {
    checkDuplicateUsername,
    verifyToken,
} = require("../middleware/authentication");
const router = Router();

router.post("/signup", checkDuplicateUsername, signup);

router.post("/signin", signin);

router.get("/rentals/:id", verifyToken, getUserRentals);

export default router;
```

Rejestracja nowego użytkownika

```
const handleSignup = async () => {
    try {
        const response = await fetch(apiKey + "/user/signup", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ username, password }),
        });

        if (response.ok) {
            navigate("/profile");
        } else {
            const errorData = await response.json();
            setError(errorData.message);
        }
    } catch (error) {
        console.error("An error occurred while signing up:", error);
    }
};
```



Efekty:

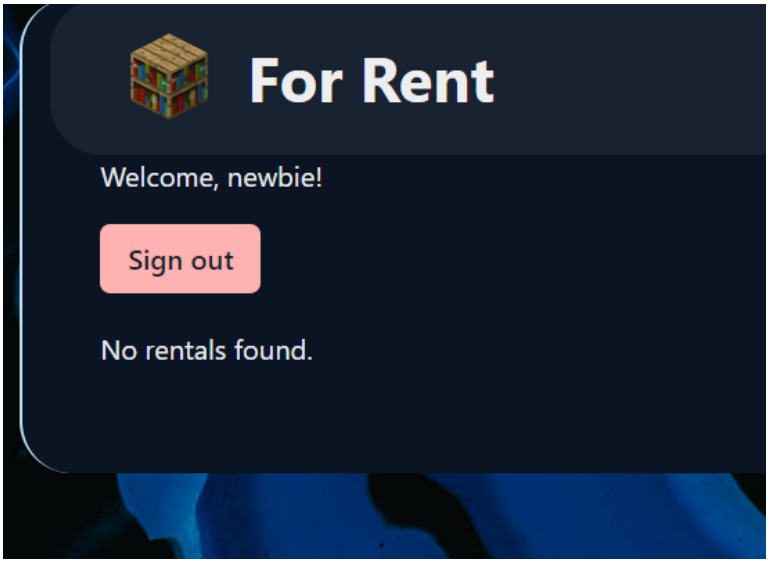
```
_id: ObjectId('6491b19962685ab9bd8d4e0b')
username: "ziutek"
password: "$2a$08$Sod.BhkedJRf8vSyEWKi4eDG4ZkKUD0.yjB8E8Dt3qX2ees5oea/2"
role: "admin"
__v: 0
```

```
_id: ObjectId('6492fc1003e9b21903d064d8')
username: "newbie"
password: "$2a$08$SyRSFf1b9BQhZv9dywQLDuTGBhz0ZEETTkPji4X1dbRPsxihHBC4K"
role: "user"
__v: 0
```

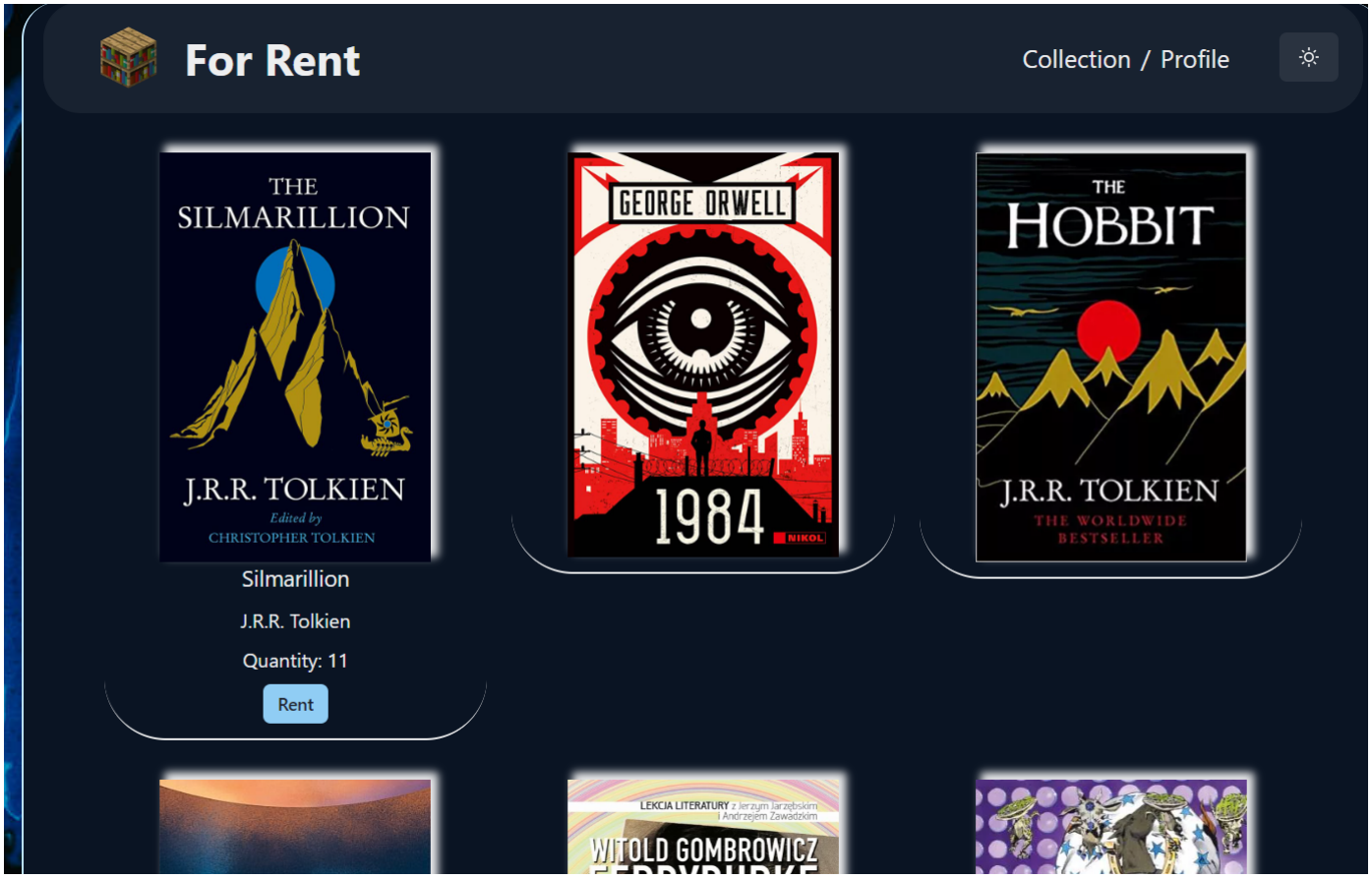
```
const handleLogin = async () => {
    try {
        const response = await fetch(apiKey + "/user/signin", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ username, password }),
        });

        if (response.ok) {
            const data: User = await response.json();
            setUser(data);
            navigate("/profile");
            console.log(data);
        } else {
            const errorData = await response.json();
            setError(errorData.message);
        }
    } catch (error) {
        console.error("An error occurred while logging in:", error);
    }
};
```

Logowanie sprawdza czy użytkownik istnieje w bazie; pobiera informacje o nim, tj. np role.
Jesteśmy już zalogowani jako nowy użytkownik “newbie”

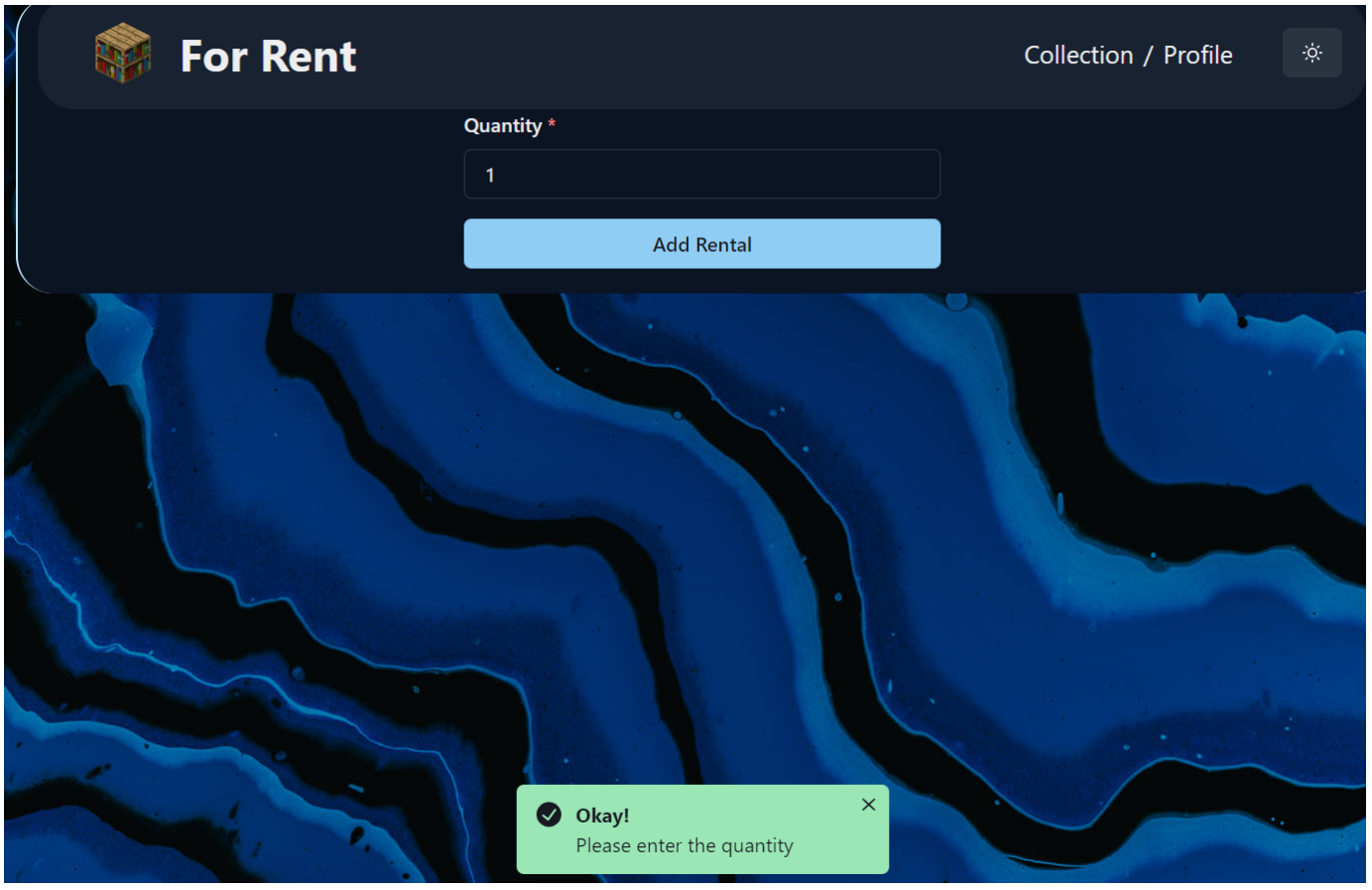
Nie mamy niestety jeszcze nic w naszej bibliotece



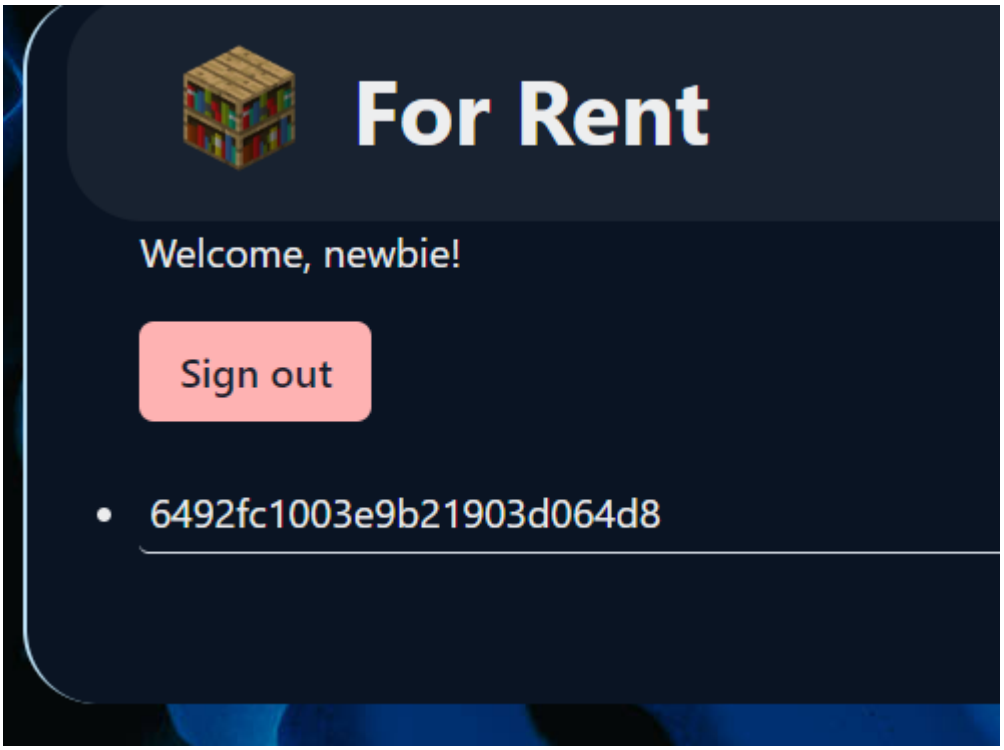
W takim razie wypożyczmy coś!



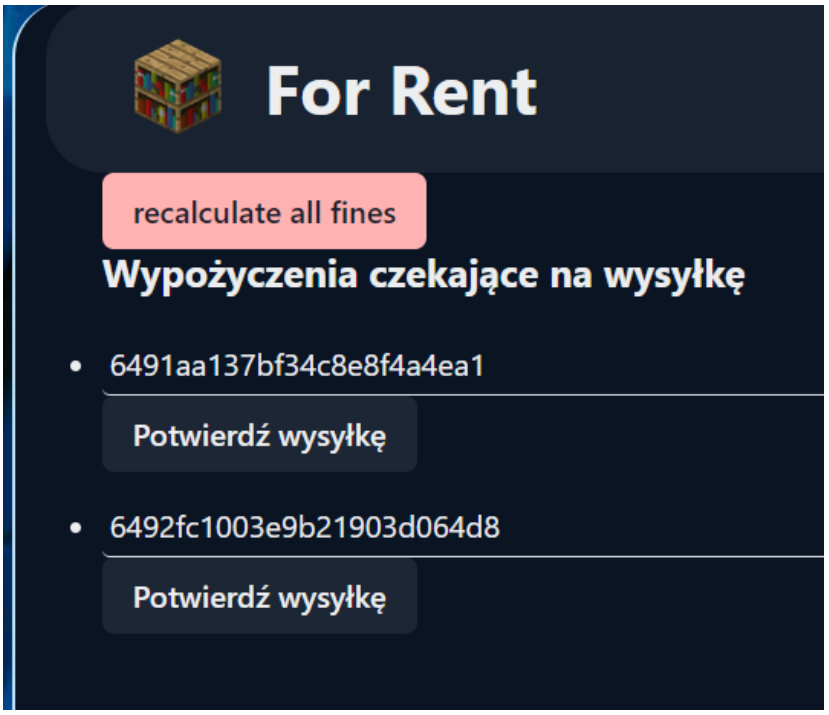
Klikamy przycisk Rent, następnie wpisujemy ile egzemplarzy chcemy i czekamy na potwierdzenie admina



Nasz newbie ma swoje wypożyczenie w historii:



I teraz admin widzi request o wysyłkę od użytkownika
(drugi rekord)



2) Route’y dla wypożyczeń:

```
import express from "express";
import {
    getAllRentals,
    getRentalById,
    createRental,
    updateRental,
    deleteRental,
    getPendingRentals,
} from "../controllers/rentalController";
import { isAdmin } from "../middleware/authentication";
import { finesMiddleware } from "../middleware/finesCalculation";

const router = express.Router();
router.get("/", isAdmin, getAllRentals);
router.get("/fines", isAdmin, finesMiddleware);
router.get("/pending", isAdmin, getPendingRentals);
router.get("/:id", isAdmin, getRentalById);
router.post("/", createRental);
router.put("/:id", isAdmin, updateRental);
router.delete("/:id", isAdmin, deleteRental);

export default router;
```

3) Route’y dla produktów:

```
import express, { Router } from "express";
import {
    getAllProducts,
    createProduct,
    getProductById,
    updateProduct,
    deleteProduct,
} from "../controllers/productController";
import { isAdmin, verifyToken } from "../middleware/authentication";

const router: Router = express.Router();

router.get("/", getAllProducts);

router.post("/", verifyToken, isAdmin, createProduct);

router.get("/:id", getProductById);

router.put("/:id", verifyToken, isAdmin, updateProduct);

router.delete("/:id", verifyToken, isAdmin, deleteProduct);

export default router;
```