BDPROJEKT_ALBINSKI_KONIOR

Skład grupy

- Piotr Albiński
- Adam Konior

Temat projektu

- Pizzeria

Informację o wykorzystywanym SZBD i technologii realizacji projektu

- MongoDB
- Node.js

Wstępny opis realizacji zadania.

Nasza pizzeria ma 4 głównych aktorów: admin, employee, deliverer, customer. Admin zarządza menu oraz ma wgląd do statystyk pizzerii. Employee i Deliverer należą do kolekcji Worker. Oni odpowiedzialni są za realizowanie zamówienia. Customer zamawia pizzę z dowozem lub na wynos. Wszystkie role, żeby korzystać z usług pizzerii muszą posiadać konto (kolekcja user).

Spis treści

Proste operacje CRUD

Operacje o charakterze transakcyjnym

Inne operacje

Zapytania raportujące

- getCurrentOrders (dla pracownika)

Wstępne informacje:

- Korzystamy z frameworka Express oraz z mongoose
- Korzystamy z tokenów JWT do zablokowania nieautoryzowanego dostępu do zasobów naszej bazy.

Oto nasz główny plik, który uruchamiany:

app

Konfigurejemy tutaj ścieżki do endpointów, middleware, połączenie z bazą.

```
const express = require('express');
const cors = require('cors');
const AdminRouter = require('./server/routers/AdminRouter.js');
const ClientRouter = require('./server/routers/ClientRouter.js');
const EmployeeRouter = require('./server/routers/EmployeeRouter');
const UserRouter = require('./server/routers/UserRouter');
const app = express();
const port = process.env.PORT || 9000;
const errorHandler = require("./server/middleware/errorHandler");
const dotenv = require('dotenv').config();
const connectDb = require('./server/config/db.Connection');
connectDb();


app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cors());
app.use("/admin", AdminRouter);
app.use("/client", ClientRouter);
app.use("/employee", EmployeeRouter);
app.use("/user", UserRouter);
app.use(errorHandler); // do obsługi błędów na endpointach
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

.env

W poniższym pliku zapisujemy stałe potrzebne do uruchomienia naszego serwera.

```
PORT=...
CONNECTION_STRING=...
ACCESS_TOKEN_SECRET=...
```

db.Connections

Plik z konfiguracją połączenia z MongoDB.

```
const mongoose = require('mongoose');

const connectDb = async () => {
    try {
        const connect = await mongoose.connect(process.env.CONNECTION_STRING);
        console.log("Database connected", connect.connection.host, connect.connection.name);
    } catch (error) {
        console.log(error);
        process.exit(1);
    }
}

module.exports = connectDb;
```

errorHandler

Middleware. Ułatwia on obsługę błędów w programie. W naszych controllerach obsługuje on wyjątki z `throw`.

```
const { constants } = require('../Constants');
const errorHandler = (err, req, res, next) => {
    const statusCode = res.statusCode ? res.statusCode : 500;
    switch(statusCode) {
        case constants.VALIDATION_ERROR:
            res.json({title: "Validation failed", message: err.message, stackTrace: err.stack});
            break;
        case constants.UNAUTHORIZED:
            res.json({title: "Unauthorized", message: err.message, stackTrace: err.stack});
            break;
        case constants.FORBIDDEN:
            res.json({title: "Forbidden", message: err.message, stackTrace: err.stack});
            break;
```

```
        case constants.NOT_FOUND:
            res.json({title: "Validation failed", message: err.message, stackTrace: err.stack});
            break;
        case constants.SERVER_ERROR:
            res.json({title: "Server error", message: err.message, stackTrace: err.stack});
            break;
        default:
            res.json({title: "Error", message: err.message, stackTrace: err.stack});
            break;
    }
}
module.exports = errorHandler;
```

Constants

Stałe używane do obsługi błędów.

```
exports.constants = {
    VALIDATION_ERROR: 400,
    UNAUTHORIZED: 401,
    FORBIDDEN: 403,
    NOT_FOUND: 404,
    SERVER_ERROR: 500
}
```

Oto nasze routery:

AdminRouter

Plik odpowiedzialnych za obsługę funkcjonalności Admina w systemie.

```
const express = require("express");
const router = express.Router();
const { addPizza,
    addIngredient,
    addDiscount,
    registerWorker,
    createOrUpdateDeliveryPrice,
    bestRatedEmployees,
    mostBeneficialPizzasLastYear,
    mostGenerousClients
    } = require("../controllers/AdminController");
const validateToken = require("../middleware/validateToken");
const authorizeAdmin = require("../middleware/authorizeAdmin");

router.post("/add_pizza", validateToken, authorizeAdmin, addPizza);
router.post("/add_ingredient", validateToken, authorizeAdmin, addIngredient);
router.post("/add_discount", validateToken, authorizeAdmin, addDiscount);
router.post("/register_worker", validateToken, authorizeAdmin, registerWorker);
router.patch("/update_delivery_price", validateToken, authorizeAdmin, createOrUpdateDeliveryPrice);
router.get("/best_rated_employees", validateToken, authorizeAdmin, bestRatedEmployees);
router.get("/most_beneficial_pizzas_last_year", validateToken, authorizeAdmin, mostBeneficialPizzasLastYear);
router.get("/most_generous_clients", validateToken, authorizeAdmin, mostGenerousClients);

module.exports = router;
```

ClientRouter:

Router odpowiedzialny za obsługę funckcjonalności klienta.

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

const { getAvailablePizzas,
  makeOrder,
  rateOrder,
  getOrderHistory,
  ratePizza} = require("../controllers/ClientController");

const validateToken = require("../middleware/validateToken");
const authorizeClient = require("../middleware/authorizeClient");

router.get("/available_pizzas", validateToken, authorizeClient, getAvailablePizzas);
router.post("/make_order", validateToken, authorizeClient, makeOrder);
router.patch("/rate_order", validateToken, authorizeClient, rateOrder);
router.get("/order_history", validateToken, authorizeClient, getOrderHistory);
router.patch("/rate_pizza", validateToken, authorizeClient, ratePizza);

module.exports = router;
```

EmployeeRouter:

Router odpowiedzialny za obsługę funckcjonalności pracownika (kucharza i dostawcy).

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

const { updateIngredientStatus,
  changeOrderStatus, getCurrentOrders} = require("../controllers/EmployeeController");

const validateToken = require("../middleware/validateToken");
const authorizeWorker = require("../middleware/authorizeWorker");

router.patch("/update_ingredient_status", validateToken, authorizeWorker, updateIngredientStatus);
router.patch("/change_order_status", validateToken, authorizeWorker, changeOrderStatus);
router.get("/get_current_orders", validateToken, authorizeWorker, getCurrentOrders);
module.exports = router;
```

UserRouter:

Router odpowiedzialny za obsługę funckcjonalności logowania i rejestracji.

```
const express = require('express');
const router = express.Router();
const {registerUser,
  loginUser,
  currentUser,
  deleteUser,
  changePassword, changeAddress} = require('../controllers/UserController');
const validateToken = require('../middleware/validateToken');


router.post('/register', registerUser);
router.post('/login', loginUser);
router.get('/current', validateToken, currentUser);
router.delete('/delete', validateToken, deleteUser);
router.patch('/change_password', validateToken, changePassword);
router.put('/change_address', validateToken, changeAddress);

module.exports = router;
```

validateToken:

Middleware służący do weryfikacji tokenu JWT.

```
const asyncHandler  = require('express-async-handler');
const jwt = require('jsonwebtoken');

const validateToken = asyncHandler(async (req, res, next) => {
    let token;
    let authHeader = req.headers.Authorization || req.headers.authorization;
    if (authHeader && authHeader.startsWith('Bearer')) {
        token = authHeader.split(' ')[1];
        jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, decoded) => {
            if (err) {
                res.status(401);
                throw new Error("Invalid token");
            }
            req.user = decoded.user;
            next();

        });
        if (!token) {
            res.status(401);
            throw new Error("Token is missing.");
        }
    }
    else {
        res.status(401);
        throw new Error("Invalid token");
    }
});

module.exports = validateToken;
```

authorizeAdmin

Middleware weryfikujący rolę administratora.

```
const asyncHandler  = require('express-async-handler');


const authorizeAdmin = asyncHandler(async (req, res, next) => {
    if (req.user.role !== "admin") {
```

```
        res.status(403);
        throw new Error(`You are not authorized to view this page as an  ${req.user.role}`);
    }
    next();
});
```

authorizeClient

Middleware weryfikujący rolę klienta.

```
const asyncHandler  = require('express-async-handler');


const authorizeClient = asyncHandler(async (req, res, next) => {
    if (req.user.role !== "client") {
        res.status(403);
        throw new Error(`You are not authorized to view this page as a ${req.user.role}`);
    }
    next();
});

module.exports = authorizeClient;
```

authorizeWorker

Middleware weryfikujący rolę pracownika.

```
const asyncHandler  = require('express-async-handler');


const authorizeWorker = asyncHandler(async (req, res, next) => {
  if (req.user.role !== "worker") {
    res.status(403);
    throw new Error(`You are not authorized to view this page as a ${req.user.role}`);
  }
  next();
});

module.exports = authorizeWorker;
```

Schemat bazy

adminvars

```
const mongoose = require('mongoose');
const AdminVarsSchema = new mongoose.Schema({
    delivery_price: {
        type: Number,
        required: true
    }
});

module.exports = mongoose.model('AdminVars', AdminVarsSchema);
```

```
    _id: ObjectId('66559bed6415145dde9b75f7')
    __v : 0
    delivery_price : 7
```

orders

```
const mongoose = require('mongoose');
const addressSchema = require('./Address');
const gradeSchema = require('./Grade');
const orderSchema = new mongoose.Schema({
    client_id: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Clients',
        required: true
    },
    employee_id: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Workers',
        required: true
    },
    deliverer_id: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Workers',
        required: false
    },
    pizzas: [
        {
```

```
            pizza_id: {
                type: mongoose.Schema.Types.ObjectId,
                ref: 'Pizzas',
                required: true
            },
            current_price: {
                type: Number,
                required: true
            },
            count: {
                type: Number,
                required: true
            },
            discount: {
                type: Number,
                min: 0,
                max: 1,
                default: 0
            }
        }
    ],
    client_address: {
        type: addressSchema,
        required: true
    },
    order_notes: {
        type: String,
        required: false
    },
    order_date: {
        type: Date,
        required: true
    },
    grade : {
        type: gradeSchema,
        required: false
    },
    status: {
        type: String,
        required: true,
        enum: ['0', '1','-1', '2', '3.1', '3.2', '4', '-4']
    },
    to_deliver: {
        type: Boolean,
        required: true
    },
    total_price: {
        with_discount: {
            type: Number,
            required: true
        },
        without_discount: {
            type: Number,
            required: true
        },
        delivery_price: {
            type: Number,
            required: true
        }
    },
    discount_id: {
        type: mongoose.Schema.Types.ObjectId,
        required: false
    }
}, {timestamps: true});

module.exports = mongoose.model('Orders', orderSchema);
```

```
_id: ObjectId('665f8c7dce009ebd0db5b0d0')
client_id : ObjectId('665f8bbace009ebd0db5b0c5')
employee_id : ObjectId('665f6b6e9a2187d098b0f923')
▾ pizzas : Array (2)
  ▾ 0: Object
      pizza_id : ObjectId('665341399e20fac0feea68b1')
      current_price : 33
      count : 2
      discount : 0.1
      _id : ObjectId('665f8c7dce009ebd0db5b0d1')
  ▾ 1: Object
      pizza_id : ObjectId('6653505680295801 87e349a5')
      current_price : 38
      count : 3
      discount : 0.1
      _id : ObjectId('665f8c7dce009ebd0db5b0d2')
▾ client_address : Object
    city : "Kraków"
    street : "Zielona 25A"
    district : "Krowodrza"
    zip_code : "11-782"
  order_notes : "Poproszę smaczne pizze!"
  order_date : 2024-06-04T21:46:00.000+00:00
  status : "0"
  to_deliver : true
▾ total_price : Object
    with_discount : 169
    without_discount : 187
    delivery_price : 7
  discount_id : ObjectId('665b181040847514b5ffd696')
  createdAt : 2024-06-04T21:51:57.885+00:00
  updatedAt : 2024-06-04T21:51:57.885+00:00
  __v : 0
```


clients

```
const mongoose = require('mongoose');
const addressSchema = require('./Address');

const clientSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    phone: {
        type: String,
        required: true
    },
    address: {
        type: addressSchema,
        required: true
    },
    order_count: {
        type: Number,
        default: 0
    },
    discount_saved: {
        type: Number,
        default: 0
    },
    grades: {
        type: [{ pizza_id: {
                    type: mongoose.Schema.Types.ObjectId,
                    ref: 'Pizzas',
                    required: true
                },
                stars: {
                    type: Number,
                    required: true,
                    min: 1,
                    max: 6}}],
        default: []
    },
    current_orders: {
        type: [{type: mongoose.Schema.Types.ObjectId, ref: 'Orders'}],
        default: []
    },
    orders_history: {
        type: [{type: mongoose.Schema.Types.ObjectId, ref: 'Orders'}],
        default: []
    }
});

clientSchema.index({current_orders: 1});
clientSchema.index({orders_history: 1});

module.exports = mongoose.model('Clients', clientSchema);
```

```
_id: ObjectId('665f8bbace009ebd0db5b0c5')
name : "Zygryd Chowańczyk"
phone : "654321222"
address : Object
  city : "Kraków"
  street : "Zielona 25A"
  district : "Krowodrza"
  zip_code : "11-782"
order_count : 0
discount_saved : 0
current_orders : Array (empty)
orders_history : Array (empty)
grades : Array (empty)
__v : 0
```


pizzas

```
const mongoose = require("mongoose");
const pizzaSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    menu_number: {
        type: Number,
        required: true
    },
    ingredients: {
        type:[{
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Ingredients'
        }],
        default: []
    },
    price: {
        type: Number,
        required: true
    },
    available: {
        type: Boolean,
        required: true
    },
    grades: {
        points_sum: {
            type: Number,
            default: 0
        },
        grade_count: {
            type: Number,
            default: 0
        }
    }
});

module.exports = mongoose.model("Pizzas", pizzaSchema);
```


```
_id: ObjectId('665341399e20fac0feea68b1')
name : "Margherita"
menu_number : 1
ingredients : Array (2)
  0: ObjectId('6651df91a21cc664a510a379')
  1: ObjectId('66533a9b3a36fde49aa46580')
price : 33
available : true
grades : Object
  points_sum : 0
  grade_count : 0
__v : 0
```


ingredients

```
const mongoose = require('mongoose');
const ingredientSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    vegan: {
        type: Boolean,
        required: true
    },
    vegetarian: {
        type: Boolean,
        required: true
    },
    available: {
        type: Boolean,
        required: true
    }
});

module.exports = mongoose.model('Ingredients', ingredientSchema);
```

```
_id: ObjectId('6651df91a21cc664a510a379')
name : "Sos pomidorowy"
vegan : true
vegetarian : true
available : true
__v : 0
```

workers

```
const mongoose = require('mongoose');
const addressSchema = require('./Address');
const workerSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    worker_type: {
        type: String,
        required: true,
        enum: ["employee", "deliverer"]
    },
    salary: {
        type: Number,
        required: true
    },
    phone: {
        type: String,
        required: true
    },
    address: {
        type: addressSchema,
        required: true
    },
    status: {
        type: String,
        required: true,
        enum: ["active", "inactive"]
    },
    current_orders: {
        type: [{type: mongoose.Schema.Types.ObjectId, ref: 'Orders'}],
        default: []
    },
    orders_history: {
        type: [{type: mongoose.Schema.Types.ObjectId, ref: 'Orders'}],
        default: []
    }

});

workerSchema.index({current_orders: 1});
workerSchema.index({orders_history: 1});

module.exports = mongoose.model('Workers', workerSchema);
```

```
_id: ObjectId('665f6c4f9a2187d098b0f92f')
name : "Mariusz Kowalski"
worker_type : "deliverer"
salary : 5600
phone : "124144239"
▾ address : Object
    city : "Kraków"
    street : "Zdrowa 42C"
    district : "Kraków Płaszów"
    zip_code : "22-456"
  status : "active"
▾ current_orders : Array (empty)
▾ orders_history : Array (empty)
  __v : 0
```

users (połączone przez _id z clients i workers, w users istnieje też jedno konto admina)

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    },
    role: {
        type: String,
        required: true
    }
}, {timestamps: true});
```

```
module.exports = mongoose.model('Users', userSchema);
```

```
_id: ObjectId('665f8bbace009ebd0db5b0c5')
email : "zygfryd@gmail.com"
password : "$2b$10$GiIJzK2yoTdry/KMlb6jmOEGWrCNkYVeJb9saIQJ1l/v2MOhRfw6G"
role : "client"
createdAt : 2024-06-04T21:48:42.438+00:00
updatedAt : 2024-06-04T21:48:42.438+00:00
__v : 0
```

gradeSchema:

```
const mongoose = require('mongoose');
const gradeSchema = new mongoose.Schema({
    _id: false,
    grade_food: {
        type: Number,
        min: 1,
        max: 6,
        required: true
    },
    grade_delivery: {
        type: Number,
        min: 1,
        max: 6,
        required: true
    },
    comment: {
        type: String,
        required: false,
        maxlength: 255
    }
});

module.exports = gradeSchema;
```

```
_id: ObjectId('665f8c7dce009ebd0db5b0d0')
client_id : ObjectId('665f8bbace009ebd0db5b0c5')
employee_id : ObjectId('665f6b6e9a2187d098b0f923')
▸ pizzas : Array (2)
▸ client_address : Object
order_notes : "Poproszę smaczne pizze!"
order_date : 2024-06-04T21:46:00.000+00:00
status : "4"
to_deliver : true
▸ total_price : Object
discount_id : ObjectId('665b181040847514b5ffd696')
createdAt : 2024-06-04T21:51:57.885+00:00
updatedAt : 2024-06-04T22:01:41.884+00:00
__v : 0
deliverer_id : ObjectId('665f6bfc9a2187d098b0f92b')
▾ grade : Object
    grade_food : 4
    grade_delivery : 3
    comment : "Smaczna pizza"
```

Nie istnieje jako samodzielny dokument w bazie, poniżej w orderze:

addressSchema:

```
const mongoose = require('mongoose');
const addressSchema = new mongoose.Schema({
    _id: false,
    city: {
        type: String,
        required: true
    },
    street: {
        type: String,
        required: true
    },
    district: {
        type: String,
        required: false
    },
    zip_code: {
        type: String,
        required: true
    }
});

module.exports = addressSchema;
```

```
▾ client_address : Object
    city : "Kraków"
    street : "Zielona 25A"
    district : "Krowodrza"
    zip_code : "11-782"
```

Dodatkowe operacje, które są proste i szybkie dzięki naszemu modelowi:

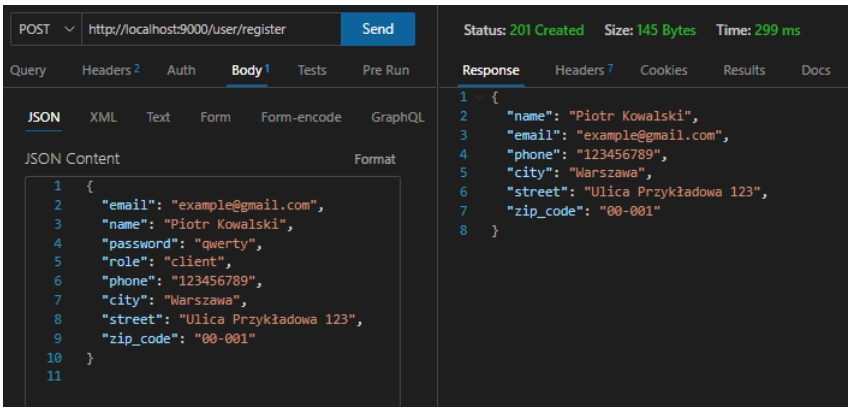
- wyświetlenie ile klient zaoszczędził na zniżkach
- wyświetlenie ile zamówień złożył dany klient
- wyświetlenie ocen, jakie wystawił dany klient
- sprawdzenie dostępności pizzy
- sprawdzenie obecnych zamówień przypisanych pracownikowi i jego historii zamówień
- wyświetlenie obecnych zamówień danego klienta i jego historii zamówień

Proste operacje CRUD - ten etap projektu wykonujemy na kolekcji users

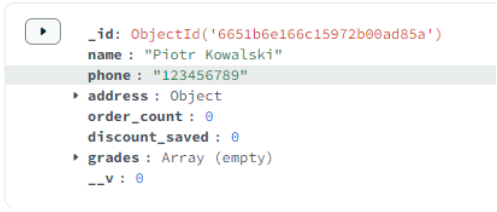
registerClient (create)

Tworzymy konto dla użytkownika w naszej bazie. Podajemy podstawowe potrzebne dane. Wykorzystujemy transakcję.

```
const registerClient = asyncHandler(async (req, res, next) => {
    const session = await mongoose.startSession();
    await session.startTransaction();
    try {
        const { email, name, password, role, phone, city, street, zip_code, district} = req.body;

        if (!name || !email || !password || !role || !phone || !city || !street || !zip_code || !district) {
            throw new Error("Please fill in all fields");
        }

        const userAvailable = await User.findOne({ email }, null, { session }); // sprawdzamy czy istnieje użytkownik o podanym emailu
        if (userAvailable) {
            throw new Error("User already exists");
        }
        const hashedPassword = await bcrypt.hash(password, 10);

        const user = await User.create([{ //tworzymy użytkownika
            email,
            password: hashedPassword,
            role
        }], { session });

        if (user && role === "client") {
            const user_id = user[0]._id;
            const client = await Client.create([{ //tworzymy klienta
                _id: user_id,
                name,
                phone,
                address: { city, street, zip_code, district}
            }], { session });

            if (client) {
                await session.commitTransaction(); //użytkownik i klient poprawnie stworzeni, zatwierdzamy transakcję
                res.status(201).json({
                    _id: client[0].user_id,
                    name: client[0].name,
                    email: user[0].email,
                    phone: client[0].phone,
                    city: client[0].address.city,
                    street: client[0].address.street,
                    zip_code: client[0].address.zip_code,
                    district: client[0].address.district
                });
            }
        } else {
            throw new Error("Invalid user data or role");
        }

    } catch (error) {
        res.status(400);
        await session.abortTransaction();
        next(error);
    } finally {
        await session.endSession();
    }
});
```



clients

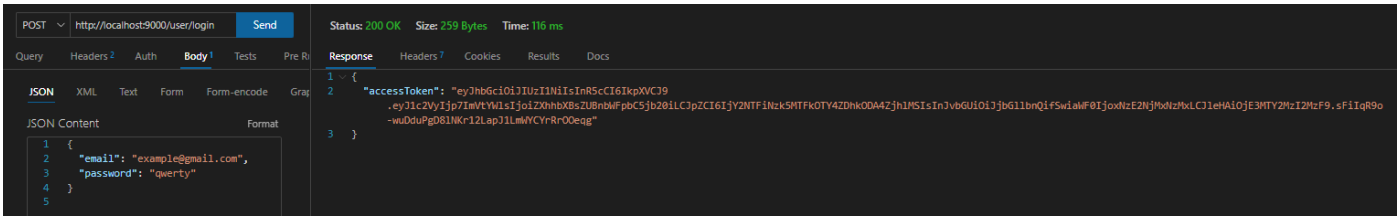


users

loginUser

Logujemy użytkownika. Tworzymy klucz JWT potrzebny do dostępu dla innych endpointów.

```
const loginUser = asyncHandler(async (req, res) => {
    const {email, password} = req.body;
    if(!email || !password){
        res.status(400);
        throw new Error("Please fill in all fields");
    }
    const user = await User.findOne({ email });
    if (user && (await bcrypt.compare(password, user.password))) {
        const accessToken = jwt.sign({user: {email: user.email, id: user.id, role: user.role}},
            process.env.ACCESS_TOKEN_SECRET, {expiresIn: "7d"});
        res.status(200).json({accessToken});
    }
    else {
        res.status(401);
        throw new Error("Invalid email or password");

    };
});
```

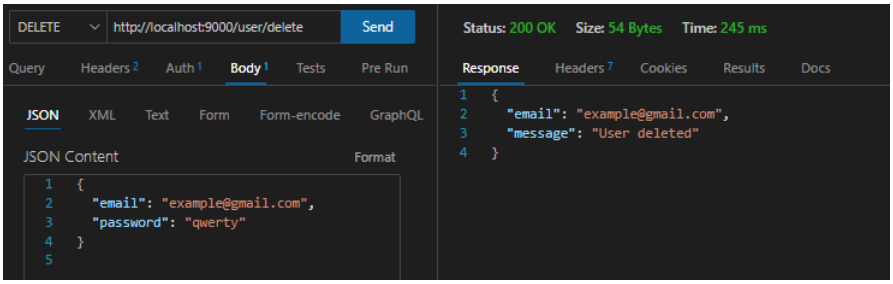


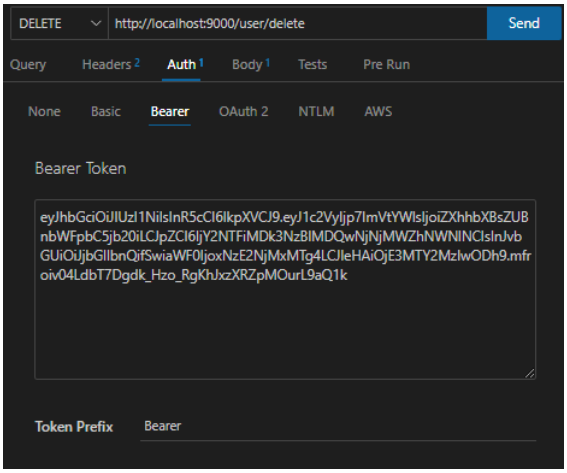
deleteUser (delete)

Podobnie jak tworzyliśmy użytkownika, tutaj też potrzebujemy zastosować transakcję.

```
const deleteUser = asyncHandler(async (req, res) => {
    const session = await mongoose.startSession();
    await session.startTransaction();
    try{
        const {email, password} = req.body;
        if(!email || !password){
            res.status(400);
            throw new Error("Please fill in all fields");
        }
        const user = await User.findOne({ email }, { session });
        if (user && (await bcrypt.compare(password, user.password))) {
            await User.deleteOne({email});
            if (user.role === "client") {
                await Client.deleteOne({_id: user.id});
            } else if (user.role === "worker") {
                await Worker.deleteOne({_id: user.id});
            }
            await session.commitTransaction();
            res.status(200).json({email, message: "User deleted"});
        }
        else {
            res.status(401);
```

```
            throw new Error("There is no user with this email or password is incorrect");

        }
    }
    catch (error) {
        await session.abortTransaction();
        next(error);
    } finally {
        await session.endSession();
    }
});
```

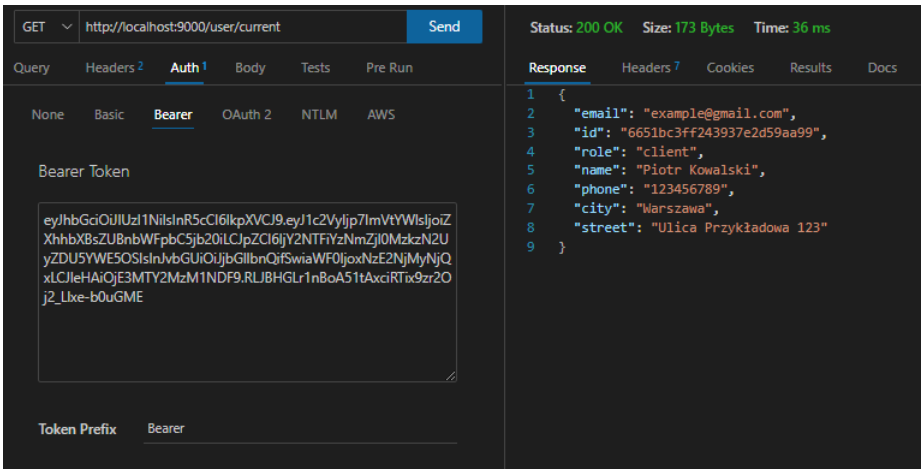




Po wykonaniu tej operacji ten użytkownik znika z obu kolekcji

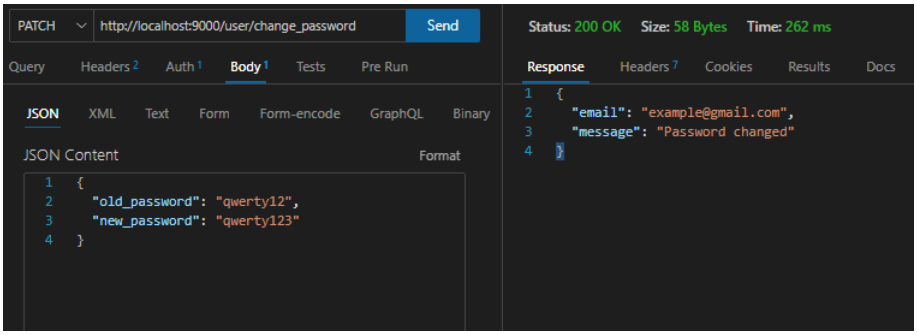
currentUser (read)

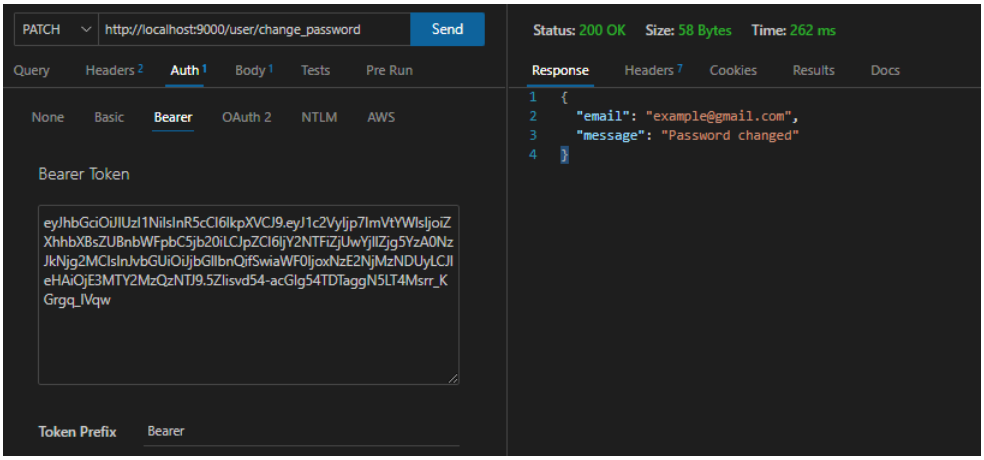
```
const currentUser = asyncHandler(async (req, res) => {
    const {email, id, role} = req.user;
    if(role === "admin"){
        res.status(200).json({email, id, role});
    }
    else if(role === "client"){
        const client = await Client.findOne({_id: id});
        res.status(200).json({email, id, role, name: client.name,
        phone: client.phone, city: client.address.city, street: client.address.street});
    }
    else if(role === "worker"){
        const worker = await Worker.findOne({_id: id});
        res.status(200).json({email, id, role, name: worker.name,
        phone: worker.phone, city: worker.address.city, street: worker.address.street});
    }
    else {
        res.status(401);
        throw new Error("Invalid role");
    }
});
```



changePassword (update)

```
const changePassword = asyncHandler(async (req, res) => {
    const {email, id, role} = req.user;
    const {old_password, new_password} = req.body;
    if(!old_password || !new_password) {
        res.status(400);
        throw new Error("Please fill in all fields");
    }
    const user = await User.findOne({email});
    if (user && (await bcrypt.compare(old_password, user.password))) {
        const hashedPassword = await bcrypt.hash(new_password, 10);
        await User.updateOne({email}, {password: hashedPassword});
        res.status(200).json({email, message: "Password changed"});
    }
    else {
        res.status(401);
        throw new Error("Incorrect old password");
    }

});
```





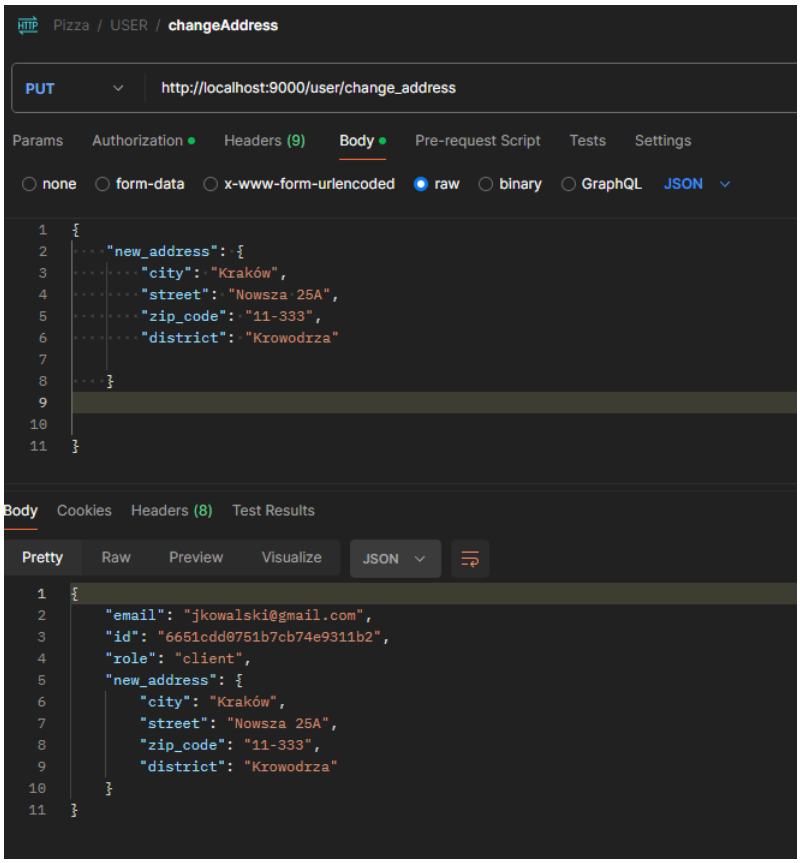
```
_id: ObjectId('6651bf50b9ef89c0472d6860')
email : "example@gmail.com"
password : "$2b$10$nGQ/kSLJ.LsOSBiiaP1dhe5FVYFbRwuAhdoABwjUmKVdVptQdtkzW"
role : "client"
createdAt : 2024-05-25T10:37:04.033+00:00
updatedAt : 2024-05-25T10:43:46.093+00:00
__v : 0
```

po zmianie

```
_id: ObjectId('6651bf50b9ef89c0472d6860')
email : "example@gmail.com"
password : "$2b$10$9wm2xvpFl9OeT4WHWBPAxuyFtUSgsDsr/q/nMCh3U.jqeULRpRgMi"
role : "client"
createdAt : 2024-05-25T10:37:04.033+00:00
updatedAt : 2024-05-25T10:44:13.128+00:00
__v : 0
```

changeAddress

```
const changeAddress = asyncHandler(async (req, res) => {
    const {email, id, role} = req.user;
    const {new_address} = req.body;
    if(!new_address) {
        res.status(400);
        throw new Error("Please fill in all fields");
    }
    if(role === "client"){
        await Client.updateOne({_id: id}, {address: new_address});
        res.status(200).json({email, id, role, new_address});
    }
    else if(role === "worker"){
        await Worker.updateOne({_id: id}, {address: new_address});
        res.status(200).json({email, id, role, new_address});
    }
    else {
        res.status(401);
        throw new Error("Invalid role");
    }

});
```




```
_id: ObjectId('6651cdd0751b7cb74e9311b2')
name : "Jan Kowalski"
phone : "919853444"
address : Object
  city : "Kraków"
  street : "Nowa 25A"
  district : "Krowodrza"
  zip_code : "46-782"
order_count : 8
discount_saved : 29.699999999999996
grades : Array (empty)
__v : 0
current_orders : Array (4)
```

```
_id: ObjectId('6651cdd0751b7cb74e9311b2')
name : "Jan Kowalski"
phone : "919853444"
address : Object
  city : "Kraków"
  street : "Nowsza 25A"
  district : "Krowodrza"
  zip_code : "11-333"
order_count : 8
discount_saved : 29.699999999999996
grades : Array (empty)
__v : 0
current_orders : Array (4)
```

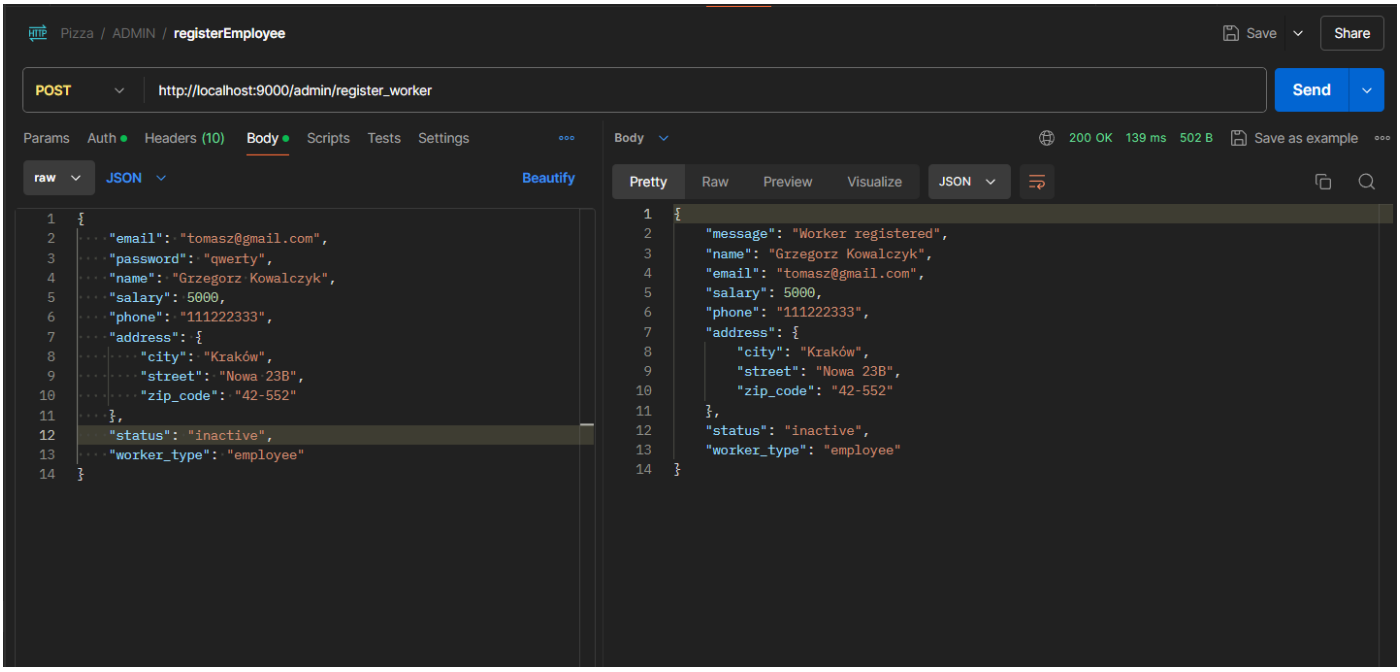

Operacje o charakterze transakcyjnym

registerWorker

dodajemy dokumenty do dwóch różnych kolekcji

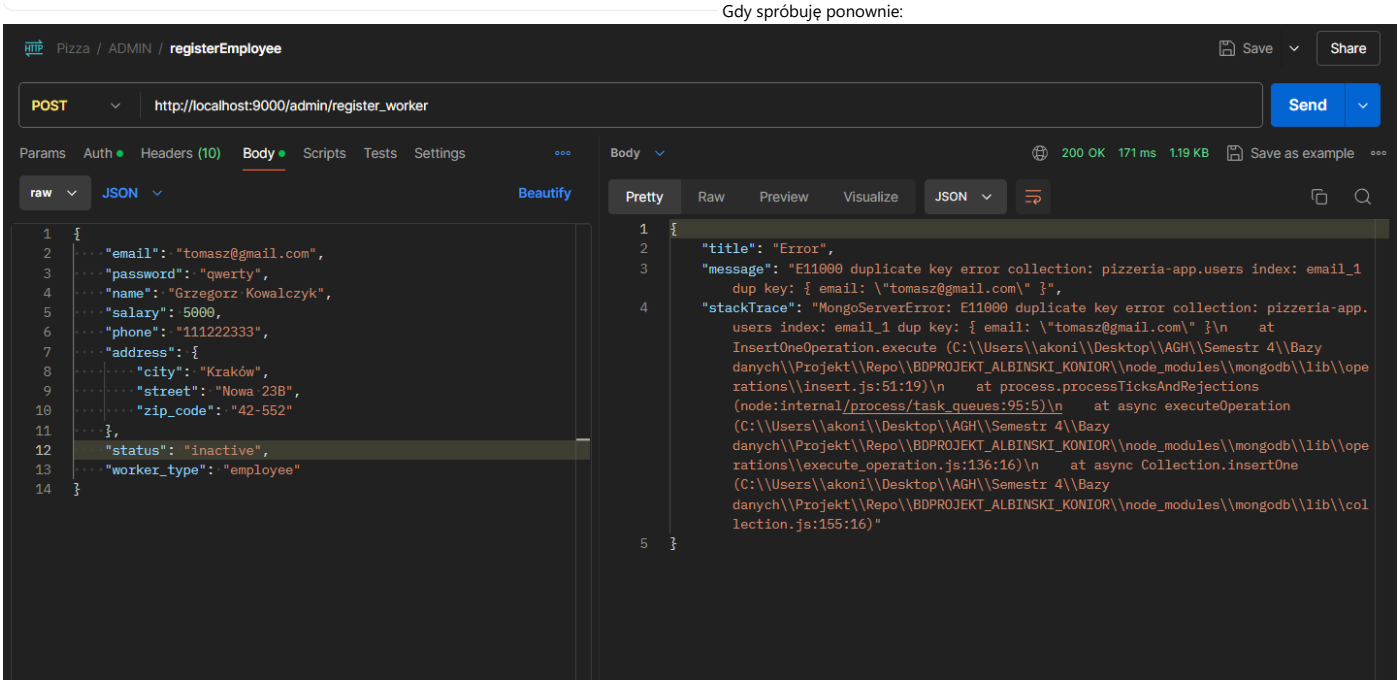
```
const registerWorker = asyncHandler(async (req, res, next) => {
  const session = await mongoose.startSession();
  await session.startTransaction();
  try {
    const { email, password, name, salary, phone, address, status, worker_type } = req.body;
    if (!email || !password || !name || !salary || !phone || !address || !status || !worker_type) {
      res.status(400);
      throw new Error("Please fill in all fields");
    }
    const hashedPassword = await bcrypt.hash(password, 10);
    const user = await User.create([{
      email,
      password: hashedPassword,
      role: "worker"
    }], {session});
    await Worker.create([{
      _id: user[0]._id,
      name,
      worker_type,
      salary,
      phone,
      address,
      status
    }], {session});
    res.status(200).json({
      message: 'Worker registered',
      name,
      email,
      salary,
      phone,
      address,
```

```
      status,
      worker_type
    });
    await session.commitTransaction();
  } catch(err) {
    await session.abortTransaction();
    next(err);
  } finally {
    await session.endSession();
  }
});
```

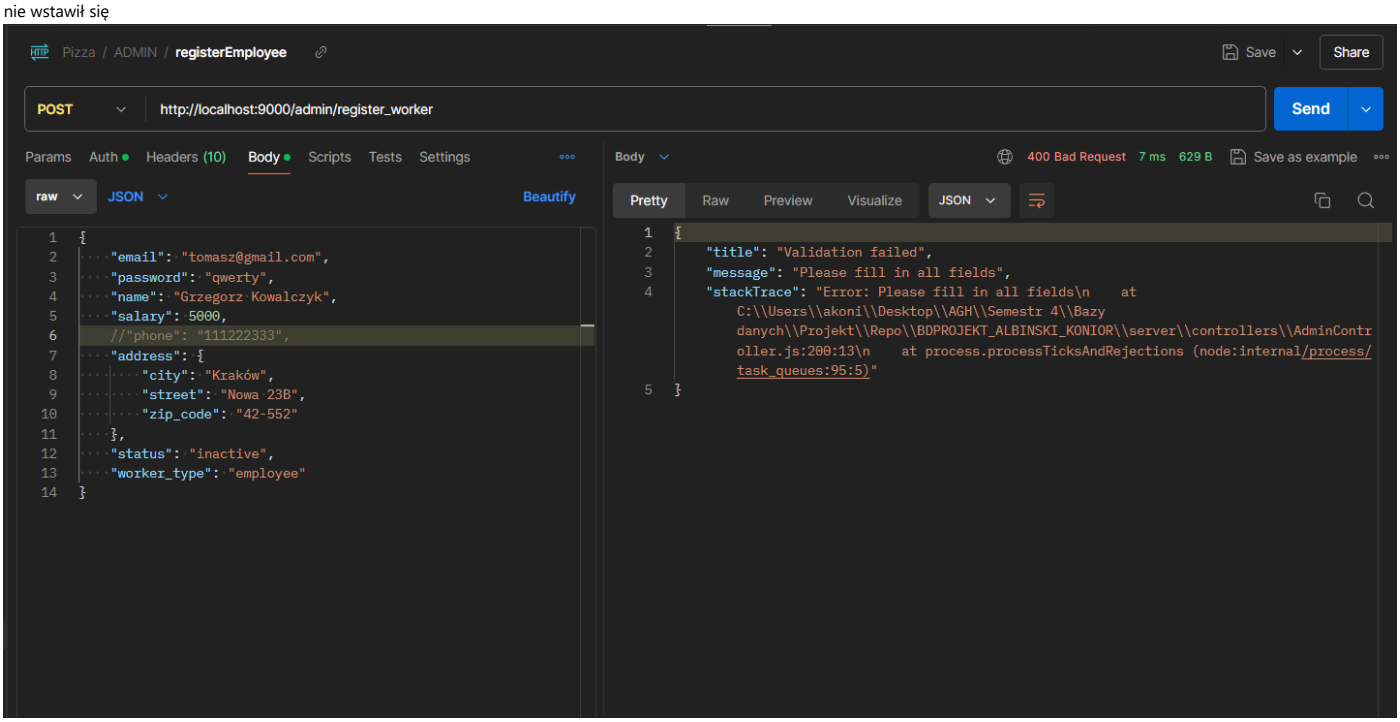



```
_id: ObjectId('6659a61a81f45774254f807f')
name : "Grzegorz Kowalczyk"
worker_type : "employee"
salary : 5000
phone : "111222333"
address : Object
  city : "Kraków"
  street : "Nowa 23B"
  zip_code : "42-552"
  _id : ObjectId('6659a61a81f45774254f8080')
status : "inactive"
current_orders : Array (empty)
orders_history : Array (empty)
__v : 0
```

Gdy spróbuję ponownie:



nie wstawił się



makeOrder

- dodajemy zamówienie do orders, a do pól current_orders w workers i current_orders w clients dodajemy orderId
- Można użyć tylko jednej zniżki na zamówienie, klient wybiera z której zniżki korzysta
- Przekazujemy order_date dla większej elastyczności tworzenia danych testowych (automatycznie pomocne byłyby po prostu timestampsy, czasami jednak chcemy coś wprowadzić starego do bazy)
- Zakładamy, że o każdej porze dnia jest dostępny jakiś dostawca. Nie przyporządkowujemy go przy składaniu zamówienia, lecz gdy pizza jest w przygotowaniu, przy zmianie statusu zamówienia (to znaczy, że jeżeli dostawcy nie są dostępni zamówienie będzie zablokowane na statusie przed ustawieniem dostawcy). Gdy żaden pracownik kuchni nie jest dostępny, nie można złożyć zamówienia. Dostawcy, gdy ustawiają swój status na inactive, oznacza to, że nie przyjmują już więcej zamówień (np. skończyli pracę).

Funkcja pomocnicza sprawdzająca dostępność pizz

```
async function checkPizzasAvailability(basket, res, sessionId) {
  const session = await mongoose.startSession({_id: sessionId});
  const pizzaIds = basket.map(item => item.id);
  const pizzas = await Pizza.find({ _id: { $in: pizzaIds } },null,  {session});
  const unavailablePizzas = pizzas.filter(pizza => !pizza.available);

  if (unavailablePizzas.length > 0) {
    const unavailablePizzaNames = unavailablePizzas.map(pizza => pizza.name).join(', ');
    res.status(400);
    throw new Error(`Pizzas ${unavailablePizzaNames} aren't available. We can't make an order.`);
  }
}
```

Funkcja pomocnicza znajdująca najmniej obciążonego pracownika.

```
async function findEmployee(sessionId) {


  const session = await mongoose.startSession({_id: sessionId});

  const employees = await Worker.find({worker_type: "employee", status: "active"}, null, { session });

  if (employees.length === 0) {
    throw new Error("There are no pizzaiolos available at the moment.");
  }
  const bestEmployee = employees.reduce((best, current) => {
    if (current.current_orders.length < best.current_orders.length) {
      return current;
    } else {
      return best;
    }
  }, employees[0]);
  return bestEmployee;
}
```

Funkcja pomocnicza obliczająca cenę zamówienia.

```
function calculateTotalPrice(basket, to_deliver, delivery_price) {
  let priceWithDiscount = 0;
  let priceWithoutDiscount = 0;
  for (let basketPos of basket) {
    priceWithDiscount += basketPos.current_price * basketPos.count * (1-basketPos.discount);
    priceWithoutDiscount += basketPos.current_price * basketPos.count;
  }
  priceWithDiscount = parseFloat(priceWithDiscount.toFixed(2));
  return to_deliver ?
      { with_discount: priceWithDiscount + delivery_price, without_discount: priceWithoutDiscount + delivery_price } :
      { with_discount: priceWithDiscount, without_discount: priceWithoutDiscount };

}
```

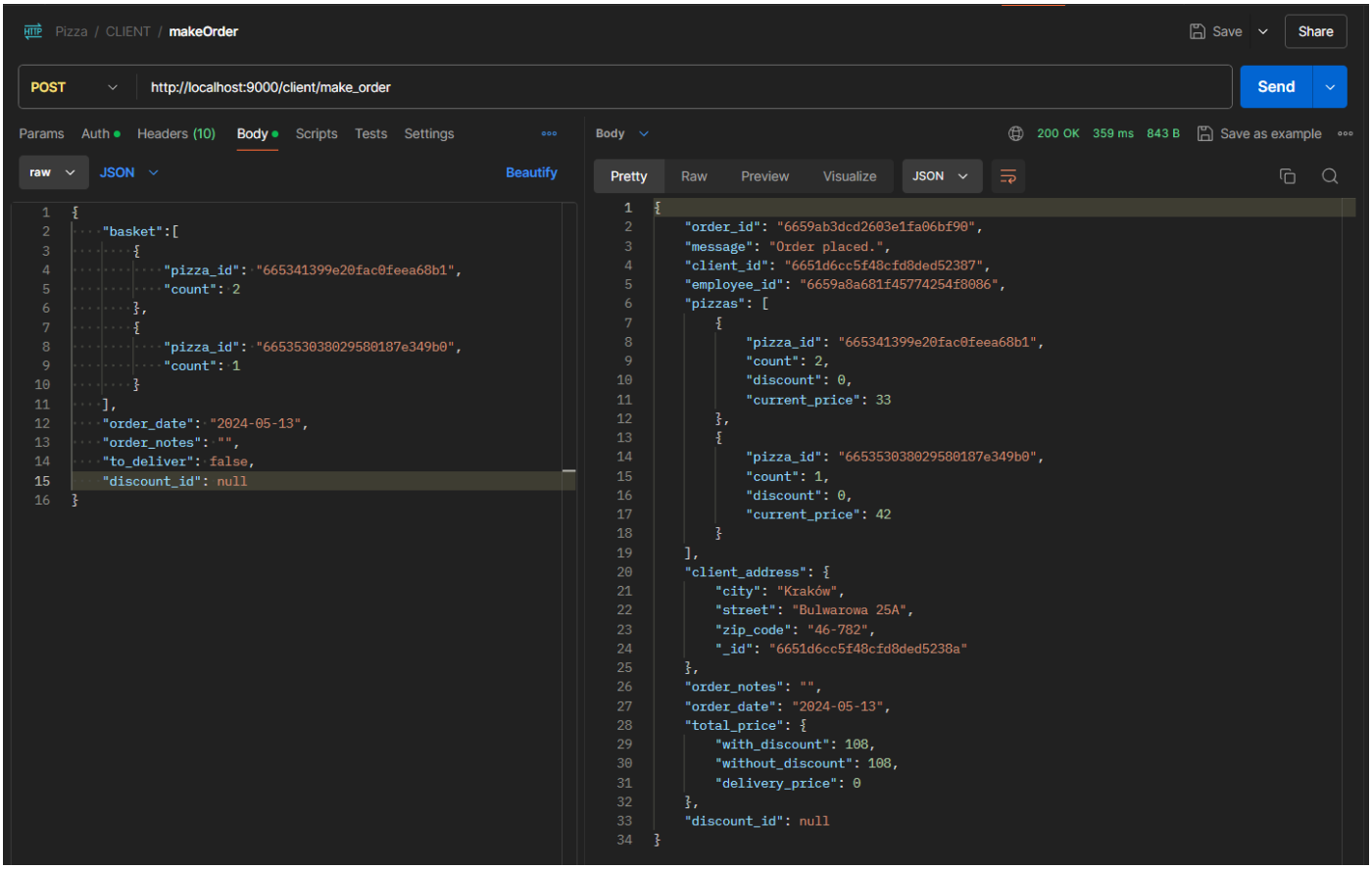
Funkcja pomocnicza sprawdzająca czy data znajduje się w przedziale dat.

```
function isDateBetween(dateToCheck, startDate, endDate) {
  return dateToCheck >= startDate && dateToCheck <= endDate;
}
```

Funkcja będą controllerem złożenia zamówienia

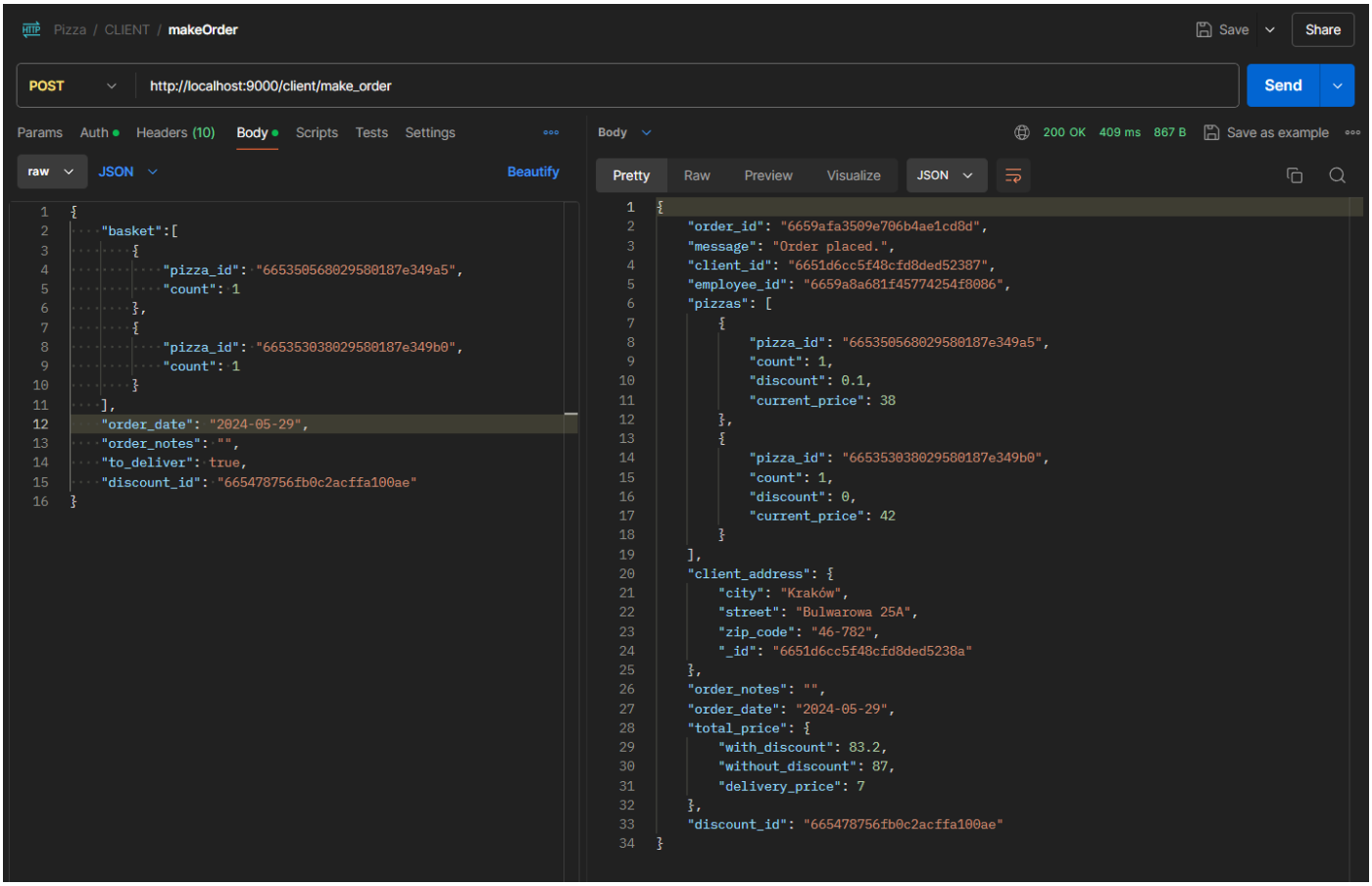
```
const makeOrder = asyncHandler(async (req, res, next) => {
  const session = await mongoose.startSession();
  await session.startTransaction();
  try {
    const {email, id, role} = req.user;
    let { basket, order_date, order_notes, to_deliver, discount_id } = req.body; // basket: [{pizza_id: ObjectId, count: Number}, {pizza_id:
ObjectId, count: Number}, ...]

    if (discount_id) {
      const the_discount = await Discount.findOne({_id: new ObjectId(discount_id)}, null, {session});

      if(!the_discount) {
        res.status(404);
        throw new Error("Discount not found");
      }
      const available = isDateBetween(new Date(order_date), the_discount.start_date, the_discount.end_date);
      if (!available){
        throw new Error("Discount not available");
      }
      for (let basketPos of basket) {
        if (the_discount.pizza_ids.includes(basketPos.pizza_id)) {
          basketPos.discount = the_discount.value;
        }
        else {
          basketPos.discount = 0;
        }
      }

    } else {
      for (let basketPos of basket) {
        basketPos.discount = 0;
      }
    }
    const pizzas = await Pizza.find({_id : {$in: basket.map(item => item.pizza_id)}}, null, {session});
    for (let basketPos of basket) {
      let pizza = pizzas.find((item) => item._id.equals(basketPos.pizza_id));
      basketPos.current_price = pizza.price;
    }
    const vars = await AdminVars.findOne(null, null, {session});
    const {with_discount, without_discount} = calculateTotalPrice(basket, to_deliver, vars.delivery_price);
    if (basket.length === 0) {
      res.status(400);
      throw new Error("We do not accept empty orders");
    }
    const employee = await findEmployee(session.id);
    await checkPizzasAvailability(basket, res, session.id);
    const clientData = await Client.findOne({_id: id},null, {session});
    let order_;
    if (discount_id) {
      order_ = await Order.create([
        {
          client_id: id,
          employee_id: employee._id,
          pizzas: basket,
          client_address: clientData.address,
          order_notes,
          order_date,
          status: '0',
          to_deliver,
          total_price: {with_discount, without_discount, delivery_price: to_deliver ? vars.delivery_price : 0},
          discount_id
        }
      ], { session });
```

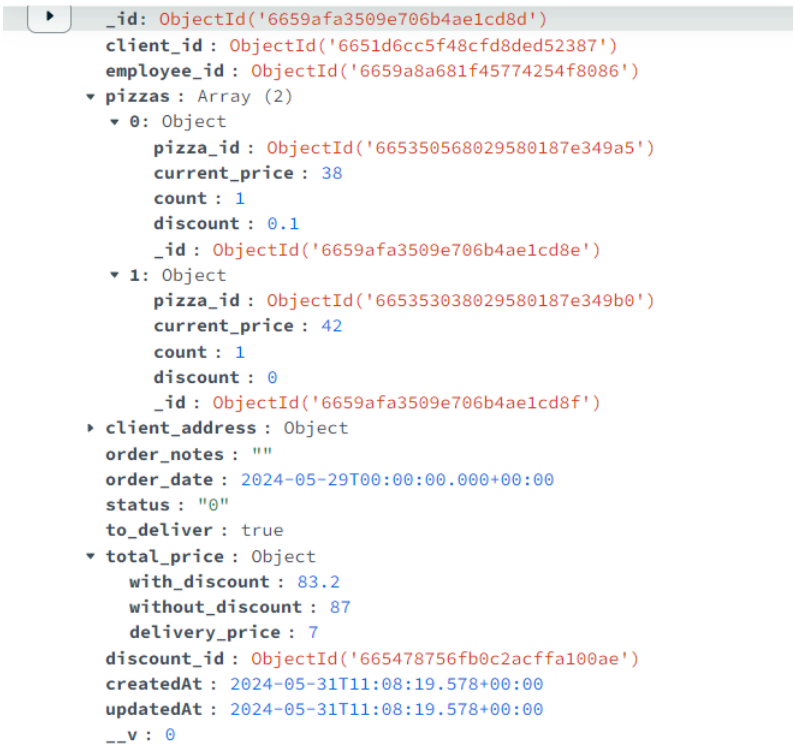
```
    } else {
      order_ = await Order.create([
        {
          client_id: id,
          employee_id: employee._id,
          pizzas: basket,
          client_address: clientData.address,
          order_notes,
          order_date,
          status: '0',
          total_price: {with_discount, without_discount, delivery_price: to_deliver ? vars.delivery_price : 0},
          to_deliver,
          discount_id: null
        }
      ], { session });
    }

    const order = order_[0];
    await Worker.updateOne({_id: employee._id}, {$push: {current_orders: order.id}}, {session});
    await Client.updateOne({_id: id}, {$push: {current_orders: order.id}}, {session});
    await session.commitTransaction();
    res.status(200).json({
      order_id: order._id,
      message: "Order placed.",
      client_id: id,
      employee_id: employee._id,
      pizzas: basket,
      client_address: clientData.address,
      order_notes,
      order_date,
      total_price: {with_discount, without_discount, delivery_price: to_deliver ? vars.delivery_price : 0},
      discount_id
    });
  } catch (err) {
    await session.abortTransaction();
    next(err);
  } finally {
    await session.endSession();
  }
});
```




```
_id: ObjectId('6659ab3dcd2603e1fa06bf90')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('6659a8a681f45774254f8086')
pizzas : Array (2)
  0: Object
      pizza_id : ObjectId('665341399e20fac0feea68b1')
      current_price : 33
      count : 2
      discount : 0
      _id : ObjectId('6659ab3dcd2603e1fa06bf91')
  1: Object
      pizza_id : ObjectId('665353038029580187e349b0')
      current_price : 42
      count : 1
      discount : 0
      _id : ObjectId('6659ab3dcd2603e1fa06bf92')
client_address : Object
order_notes : ""
order_date : 2024-05-13T00:00:00.000+00:00
status : "0"
to_deliver : false
total_price : Object
  with_discount : 108
  without_discount : 108
  delivery_price : 0
discount_id : null
createdAt : 2024-05-31T10:49:33.086+00:00
updatedAt : 2024-05-31T10:49:33.086+00:00
__v : 0
```

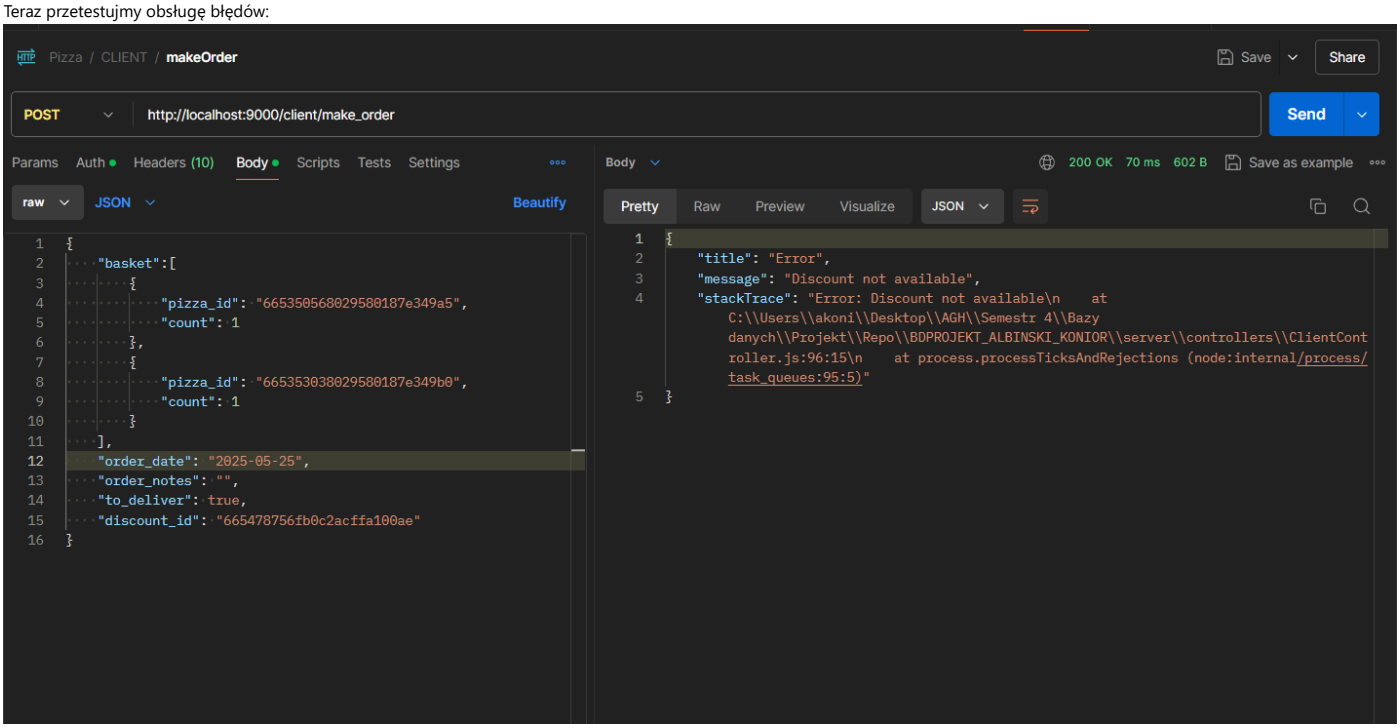
Pojawiło się zamówienie:

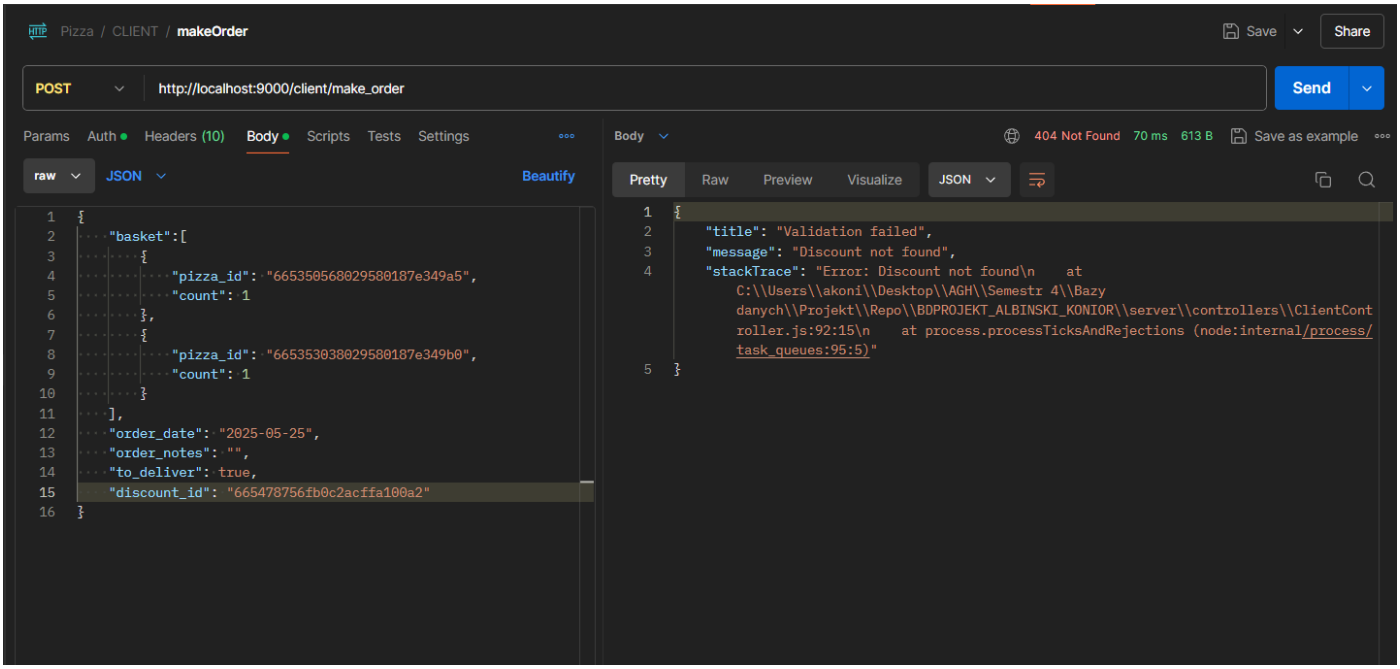




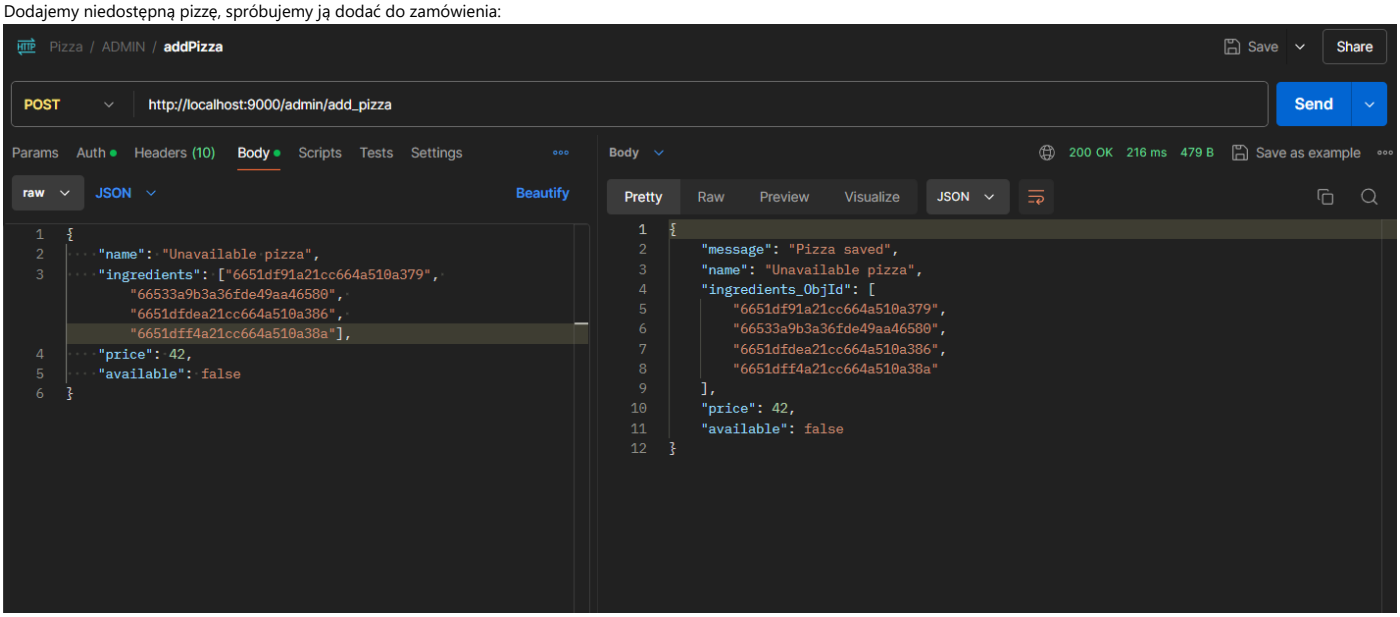
Ze zniżką i z dostawą również się pojawiło:

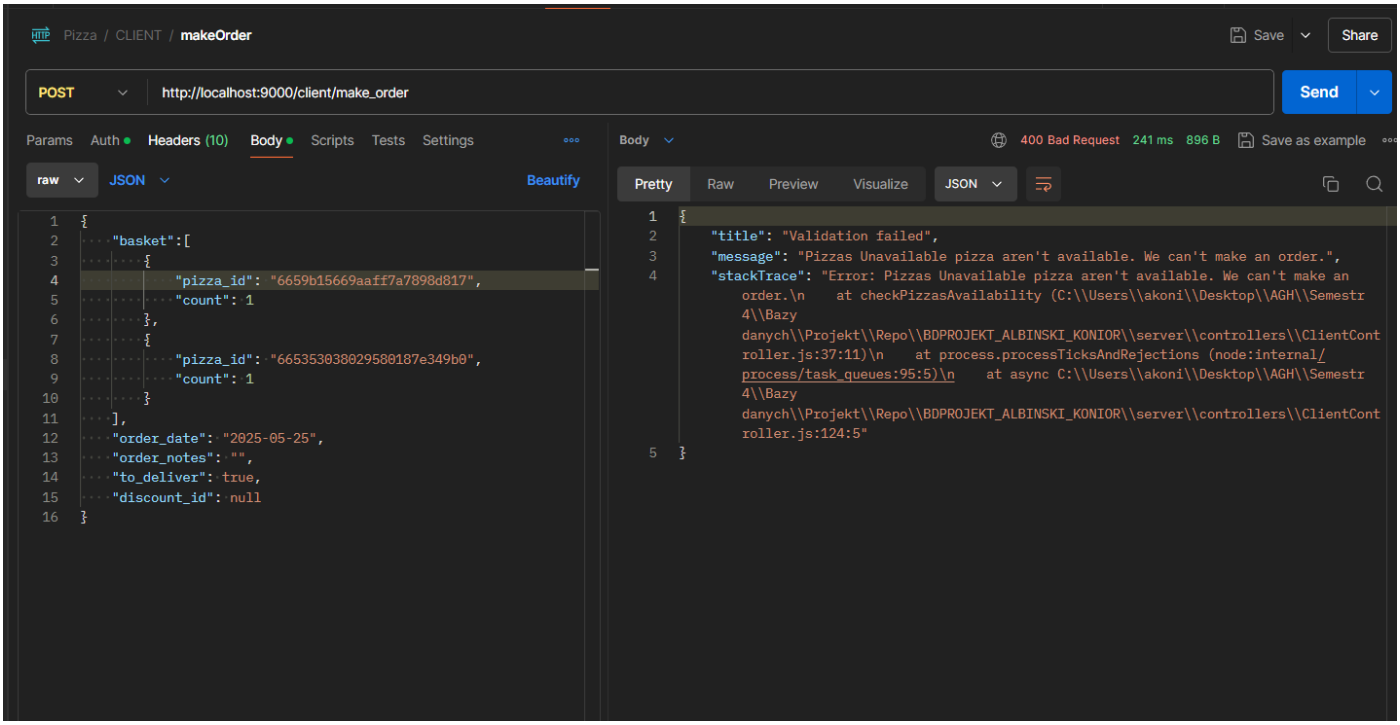
Teraz przetestujmy obsługę błędów:





Dodajemy niedostępną pizzę, spróbujemy ją dodać do zamówienia:



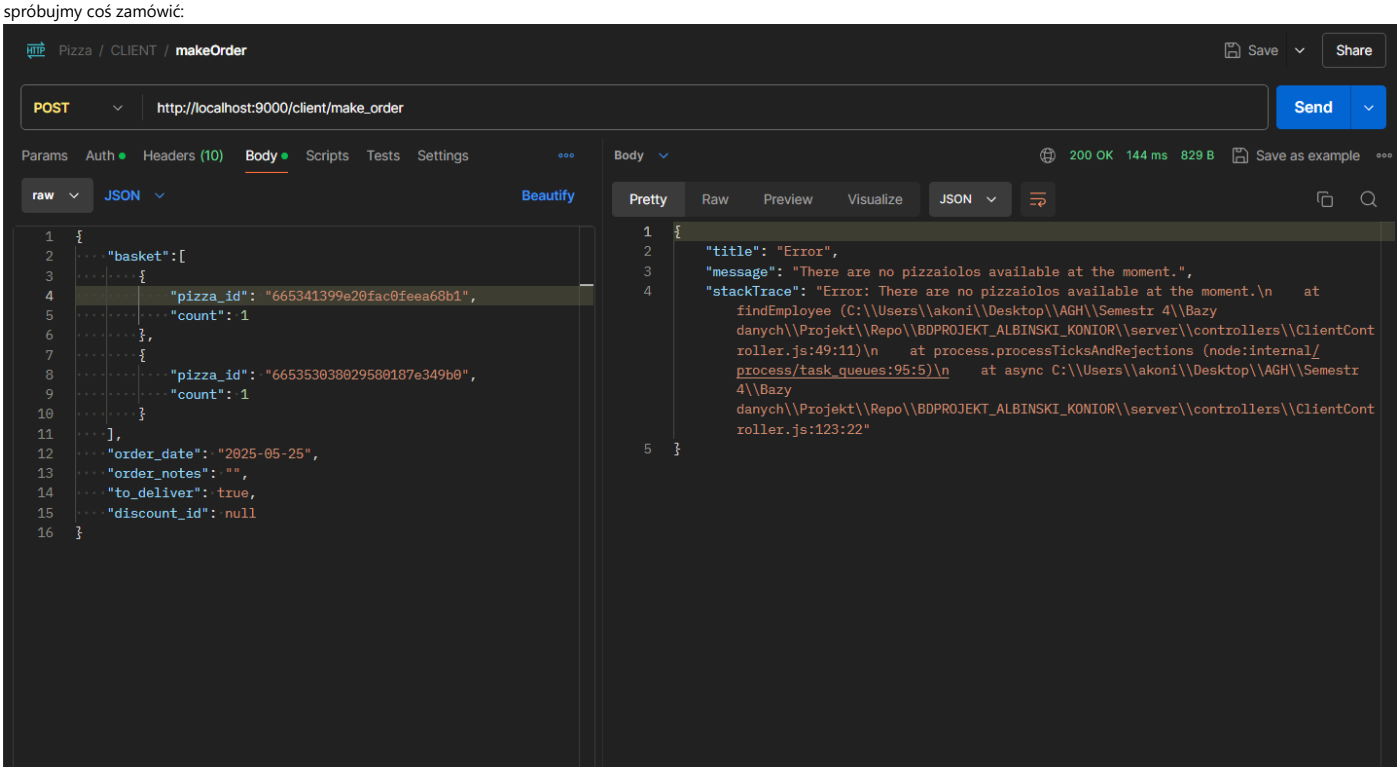



```
_id: ObjectId('6659a61a81f45774254f807f')
name : "Grzegorz Kowalczyk"
worker_type : "employee"
salary : 5000
phone : "111222333"
▸ address : Object
status : "inactive"
▸ current_orders : Array (empty)
▸ orders_history : Array (empty)
__v : 0
```

```
_id: ObjectId('6659a8a681f45774254f8086')
name : "Jan Kowalski"
worker_type : "employee"
salary : 5200
phone : "444555666"
▸ address : Object
status : "inactive"
▸ current_orders : Array (4)
▸ orders_history : Array (empty)
__v : 0
```

Zmieńmy wszystkim pracownikom status na inactive: I

spróbujmy coś zamówić:



changeOrderStatus

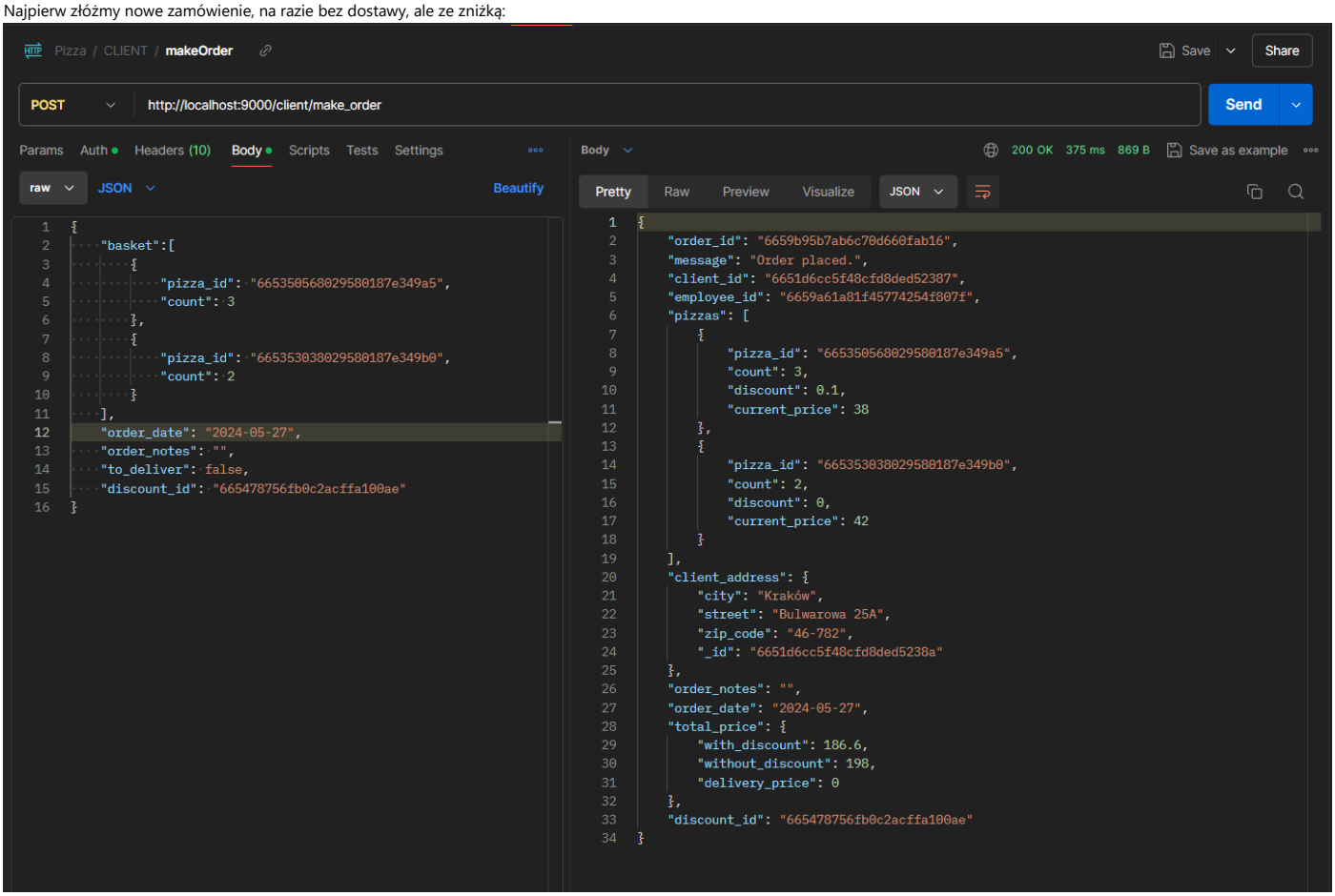
Statusy zamówienia:

- 0 - zamówienie wprowadzone do bazy, przypisany pracownik
- 1 - zamówienie przyjęte przez przypisanego pracownika
- -1 odrzucono zamówienie
- 2 Pizza w przygotowaniu, oczekiwanie na dostawcę, jeśli wybrano zamówienie z dostawą, w przeciwnym razie oczekiwanie na odbiór przez klienta
- 3.1 pizza odebrana przez dostawcę
- 3.2 pizza odebrana przez klienta
- 4 pizza dostarczona
- -4 problemy przy dostawie

```
const changeOrderStatus = asyncHandler(async (req, res, next) => {
  const session = await mongoose.startSession();
  await session.startTransaction();
  try {
    const {new_status, order_id} = req.body;
    const orderId = new ObjectId(order_id);
    const order = await Order.findOne({_id: orderId}, null, {session});
    if (!order) {
      res.status(400);
      throw new Error("Order doesn't exist");
    }
    if (new_status === '1') {
      if (order.status === '0') {
        await Order.updateOne({_id: orderId}, {status: new_status}, {session});
      } else {
        throw new Error(`Invalid new status. Current status is: ${order.status}`);
      }
    } else if (new_status === "2") {
      if (order.status === "1") {
        if (order.to_deliver) {
          const deliverer = await findDeliverer(session.id);
          if (deliverer) {
            await Order.updateOne({_id: orderId}, {status: new_status, deliverer_id: deliverer._id}, {session});
            await Worker.updateOne({_id: deliverer._id}, {$push: {current_orders: orderId}}, {session});
          }
        }
        else {
          await Order.updateOne({_id: orderId}, {status: new_status}, {session});
        }
      } else {
        throw new Error(`Invalid new status. Current status is: ${order.status}`);
      }
    } else if (new_status === '3.1') {
      if (!order.to_deliver) {
        throw new Error(`Invalid new status. Collection in person. This order's to_deliver is set to ${order.to_deliver}`);
      }
      if (order.status === "2") {
        await Order.updateOne({_id: orderId}, {status: new_status}, {session});
      } else {
```

```
        throw new Error(`Invalid new status. Current status is: ${order.status}`);
      }
    } else if (new_status === '3.2') {
      if (order.status === "2") {
        await Order.updateOne({_id: orderId}, {status: new_status}, {session});
        await Client.updateOne({_id: order.client_id}, {$inc: {order_count: 1}}, {session});
        await Worker.updateOne({_id: order.employee_id},
          {$pull: {current_orders: orderId}, $push: {orders_history: orderId}}, {session});
        await Client.updateOne({_id: order.client_id},
            {$pull: {current_orders: orderId}, $push: {orders_history: orderId}}, {session});
        const the_order = await Order.findOne({_id: orderId}, {total_price: 1, discount_id: 1}, {session});
        let saved_amount = the_order.total_price.without_discount - the_order.total_price.with_discount;
        saved_amount = parseFloat(saved_amount.toFixed(2));

        if (saved_amount > 0) {
          await Client.updateOne({_id: order.client_id}, {$inc: {discount_saved: saved_amount}}, {session});
        }
        if (the_order.discount_id) {
          await Discount.updateOne({_id: the_order.discount_id}, {$inc: {used_count: 1}}, {session});
        }
      } else {
        throw new Error(`Invalid new status. Current status is: ${order.status}`);
      }

    } else if (new_status === '-1') {
      await Order.updateOne({_id: orderId}, {status: new_status}, {session});
      await Worker.updateOne({_id: order.employee_id}, {
        $pull: {current_orders: orderId},
        $push: {orders_history: orderId}
      }, {session});
      await Client.updateOne({_id: order.client_id},
          {$pull: {current_orders: orderId}, $push: {orders_history: orderId}}, {session});
    } else if (new_status === '4') {
      if (order.status === "3.1") {
        await Order.updateOne({_id: orderId}, {status: new_status}, {session});
        await Client.updateOne({_id: order.client_id}, {$inc: {order_count: 1}}, {session});
        await Worker.updateMany({_id: {$in: [order.employee_id, order.deliverer_id]}},
          {$pull: {current_orders: orderId}, $push: {orders_history: orderId}}, {session});
        await Client.updateOne({_id: order.client_id},
            {$pull: {current_orders: orderId}, $push: {orders_history: orderId}}, {session});
        const the_order = await Order.findOne({_id: orderId}, {total_price: 1, discount_id: 1}, {session});
        let saved_amount = the_order.total_price.without_discount - the_order.total_price.with_discount;
        saved_amount = parseFloat(saved_amount.toFixed(2));
        if (saved_amount > 0) {
          await Client.updateOne({_id: order.client_id}, {$inc: {discount_saved: saved_amount}}, {session});
        }
        if (the_order.discount_id) {
          await Discount.updateOne({_id: the_order.discount_id}, {$inc: {used_count: 1}}, {session});
        }
      } else {
        throw new Error(`Invalid new status. Current status is: ${order.status}`);
      }
    }  else if (new_status === '-4') {
      await Order.updateOne({_id: orderId}, {status: new_status}, {session});
      await Worker.updateMany({_id: {$in: [order.employee_id, order.deliverer_id]}},
          {$pull: {current_orders: orderId}, $push: {orders_history: orderId}}, {session});
      await Client.updateOne({_id: order.client_id},
          {$pull: {current_orders: orderId}, $push: {orders_history: orderId}}, {session});
    }
    else {
        throw new Error(`Invalid new status: ${new_status}`);
    }
    await session.commitTransaction();
    res.status(201).json({message: `Order status set to ${new_status}`});
  }
  catch(err) {
    await session.abortTransaction();
    next(err);
  } finally {
    await session.endSession();
  }
});
```

Najpierw złóżmy nowe zamówienie, na razie bez dostawy, ale ze zniżką:

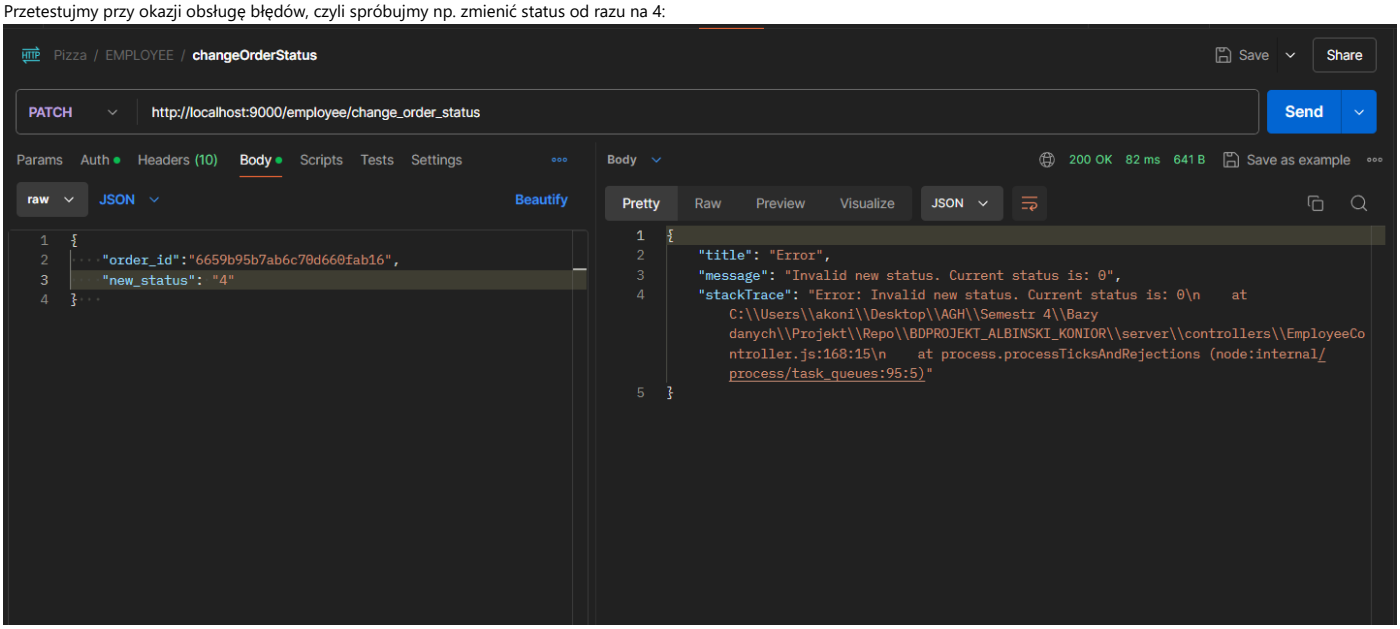



```
_id: ObjectId('6651d6cc5f48cfd8ded52387')
name : "Dariusz Nowak"
phone : "919853444"
▸ address : Object
order_count : 6
discount_saved : 22.8
▸ grades : Array (empty)
__v : 0
▾ current_orders : Array (7)
   0: ObjectId('6659a90481f45774254f8094')
   1: ObjectId('6659a99481f45774254f80a0')
   2: ObjectId('6659ab3dcd2603e1fa06bf90')
   3: ObjectId('6659af45cd2603e1fa06bf9c')
   4: ObjectId('6659afa3509e706b4ae1cd8d')
   5: ObjectId('6659b1b169aaff7a7898d81e')
   6: ObjectId('6659b95b7ab6c70d660fab16')
```

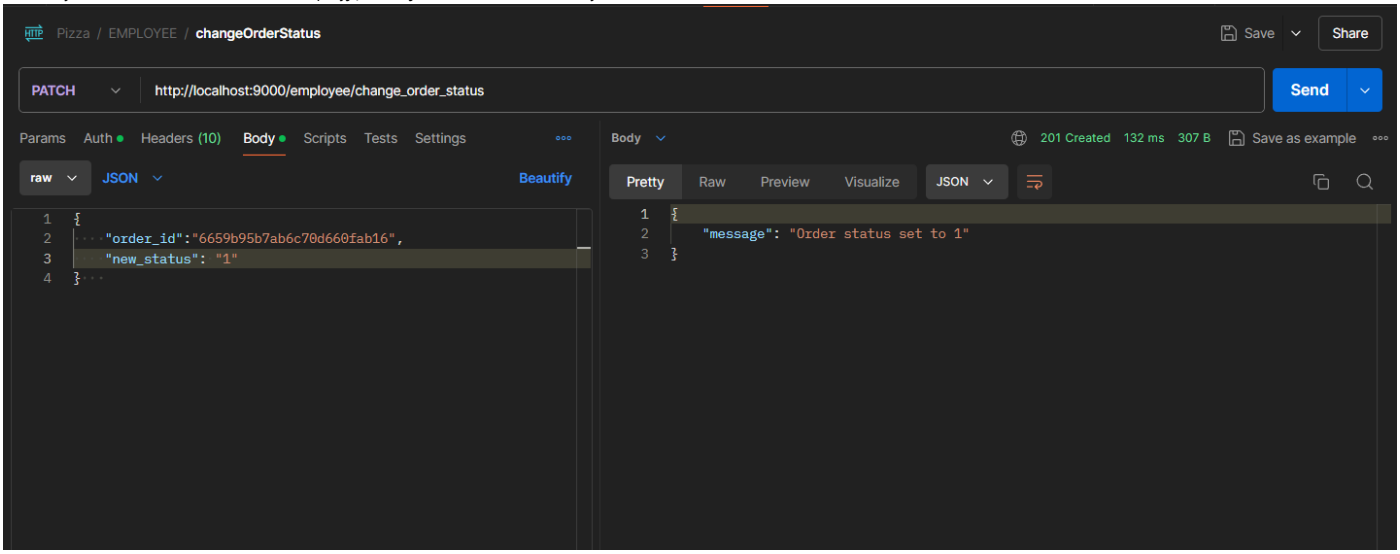
Dodało się również pole w current_orders w clients: (Jako 6. pole) i w workers:

```
_id: ObjectId('6659a61a81f45774254f807f')
name : "Grzegorz Kowalczyk"
worker_type : "employee"
salary : 5000
phone : "111222333"
▸ address : Object
status : "active"
▾ current_orders : Array (1)
   0: ObjectId('6659b95b7ab6c70d660fab16')
▸ orders_history : Array (empty)
__v : 0
```

Przetestujmy przy okazji obsługę błędów, czyli spróbujmy np. zmienić status od razu na 4:

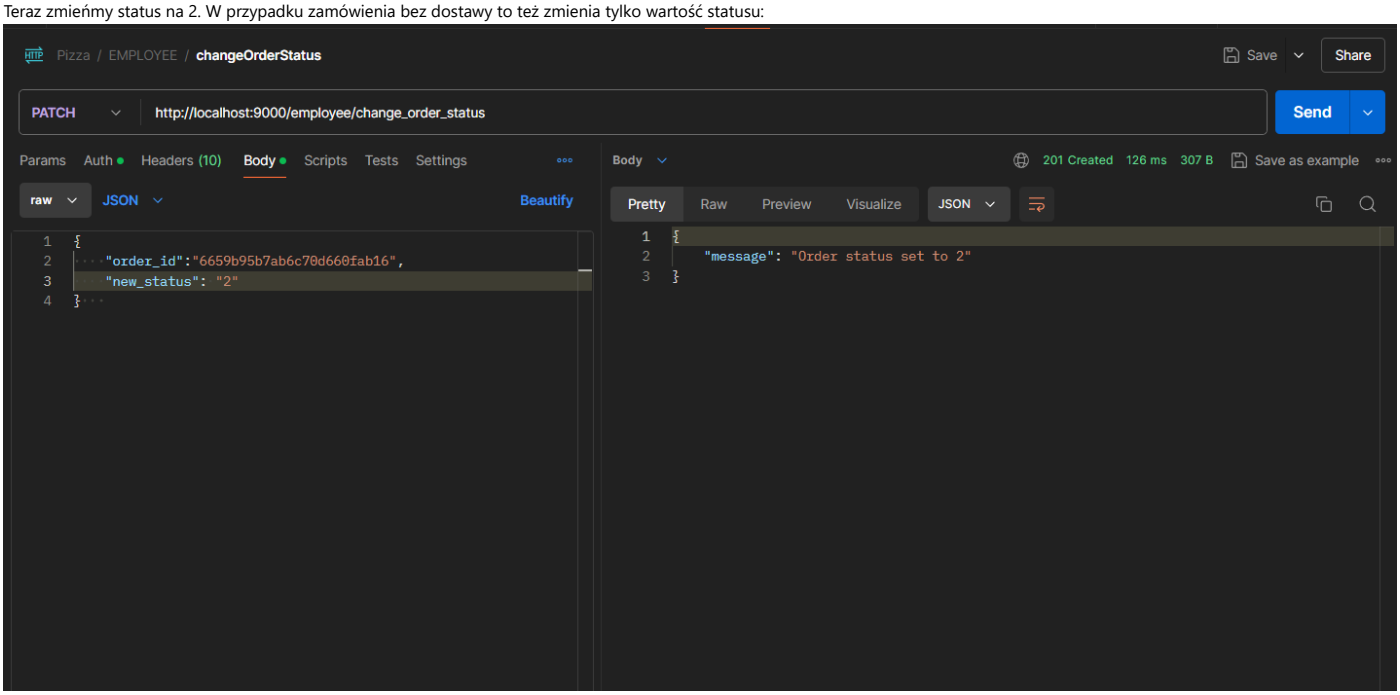


Zmieńmy na razie status na 1(zamówienie przyjęte). To jeszcze nic nie zmienia, tylko wartość statusu:



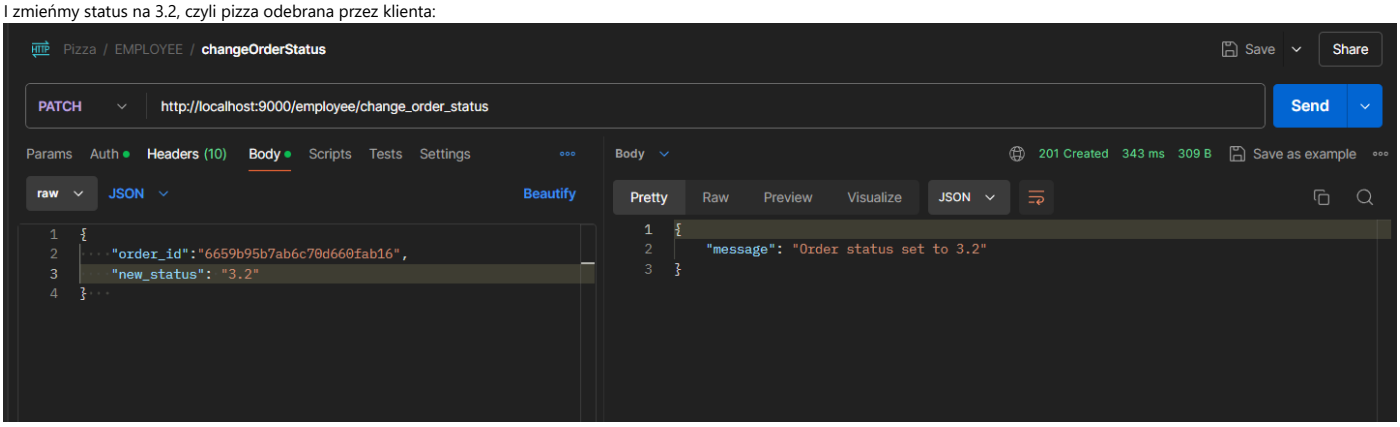

```
_id: ObjectId('6659b95b7ab6c70d660fab16')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('6659a61a81f45774254f807f')
▸ pizzas : Array (2)
▸ client_address : Object
order_notes : ""
order_date : 2024-05-27T00:00:00.000+00:00
status : "1"
to_deliver : false
▸ total_price : Object
discount_id : ObjectId('665478756fb0c2acffa100ae')
createdAt : 2024-05-31T11:49:47.288+00:00
updatedAt : 2024-05-31T12:01:34.199+00:00
__v : 0
```

Teraz zmieńmy status na 2. W przypadku zamówienia bez dostawy to też zmienia tylko wartość statusu:




```
_id: ObjectId('6659b95b7ab6c70d660fab16')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('6659a61a81f45774254f807f')
▸ pizzas : Array (2)
▸ client_address : Object
order_notes : ""
order_date : 2024-05-27T00:00:00.000+00:00
status : "2"
to_deliver : false
▸ total_price : Object
discount_id : ObjectId('665478756fb0c2acffa100ae')
createdAt : 2024-05-31T11:49:47.288+00:00
updatedAt : 2024-05-31T12:03:25.652+00:00
__v : 0
```


I zmieńmy status na 3.2, czyli pizza odebrana przez klienta:




```
_id: ObjectId('6659b95b7ab6c70d660fab16')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('6659a61a81f45774254f807f')
▸ pizzas : Array (2)
▸ client_address : Object
order_notes : ""
order_date : 2024-05-27T00:00:00.000+00:00
status : "3.2"
to_deliver : false
▸ total_price : Object
discount_id : ObjectId('665478756fb0c2acffa100ae')
createdAt : 2024-05-31T11:49:47.288+00:00
updatedAt : 2024-05-31T12:04:43.050+00:00
__v : 0
```


Zamówienie przeniosło się do orders_history klienta oraz zwiększyła się wartość

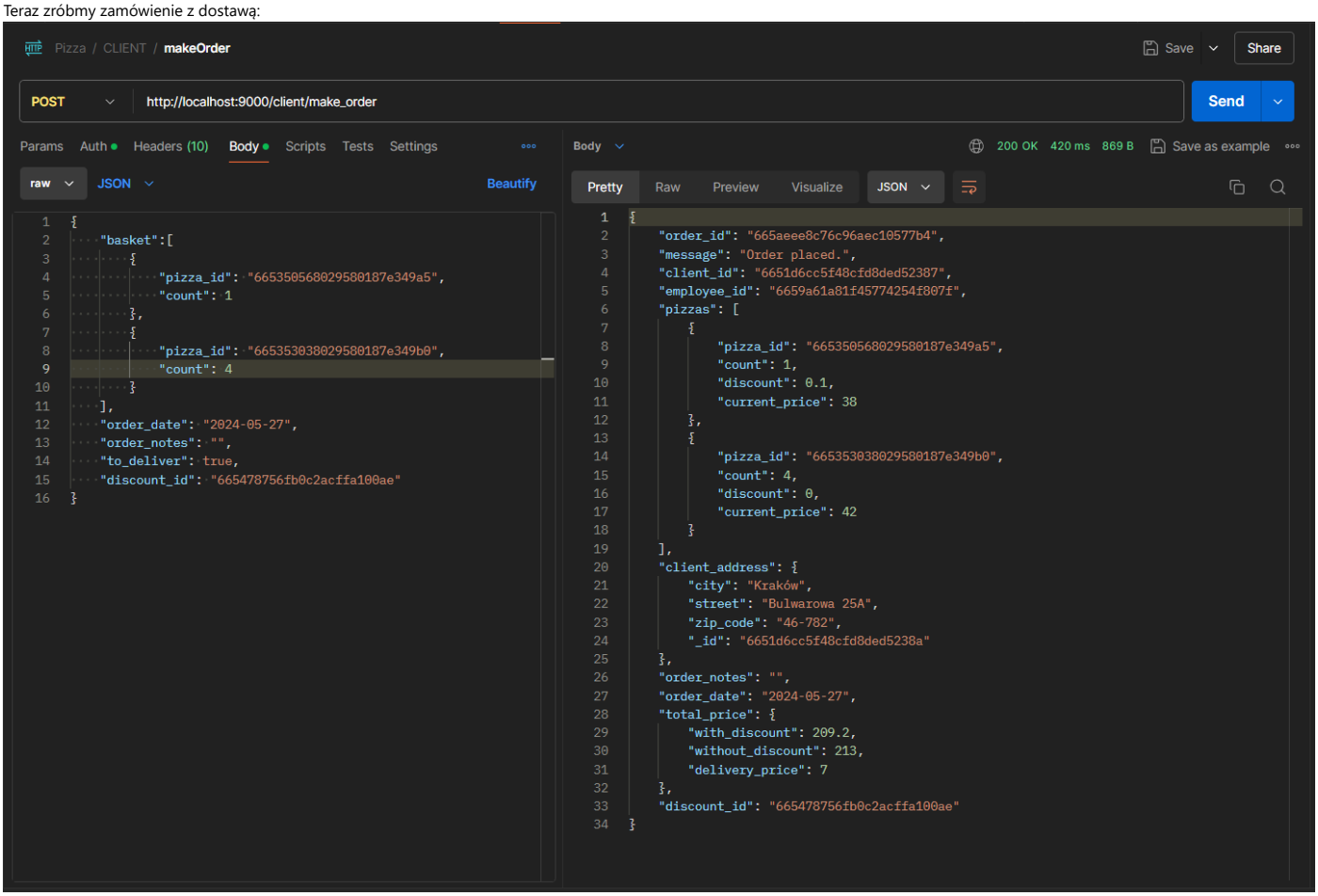
```
_id: ObjectId('6651d6cc5f48cfd8ded52387')
name : "Dariusz Nowak"
phone : "919853444"
address : Object
order_count : 7
discount_saved : 34.2
grades : Array (empty)
__v : 0
current_orders : Array (6)
  0: ObjectId('6659a90481f45774254f8094')
  1: ObjectId('6659a99481f45774254f80a0')
  2: ObjectId('6659ab3dcd2603e1fa06bf90')
  3: ObjectId('6659af45cd2603e1fa06bf9c')
  4: ObjectId('6659afa3509e706b4ae1cd8d')
  5: ObjectId('6659b1b169aaff7a7898d81e')
orders_history : Array (1)
  0: ObjectId('6659b95b7ab6c70d660fab16')
```


discount_saved, czyli pola mówiącego ile dany klient zaoszczędził na zniżkach: To samo


```
_id: ObjectId('6659a61a81f45774254f807f')
name : "Grzegorz Kowalczyk"
worker_type : "employee"
salary : 5000
phone : "111222333"
address : Object
status : "active"
current_orders : Array (empty)
orders_history : Array (1)
  0: ObjectId('6659b95b7ab6c70d660fab16')
__v : 0
```

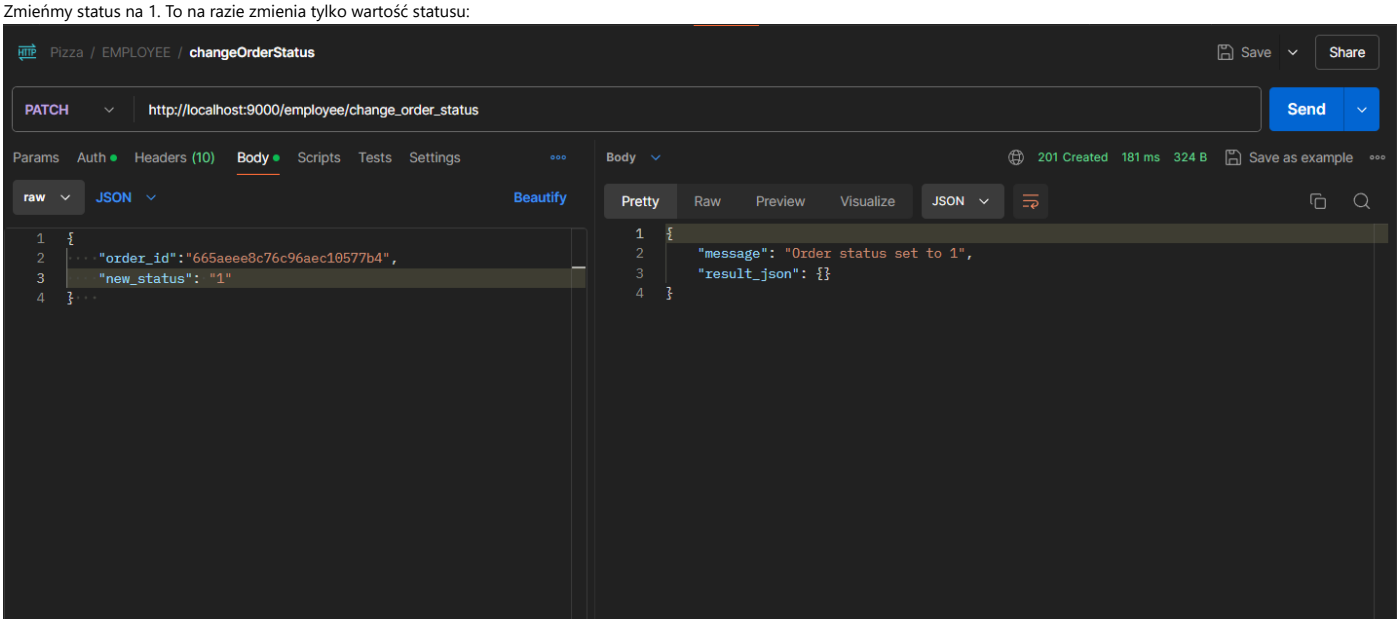

w przypadku pracownika(zamówienie przeniosło się do orders_history):

Teraz zróbmy zamówienie z dostawą:



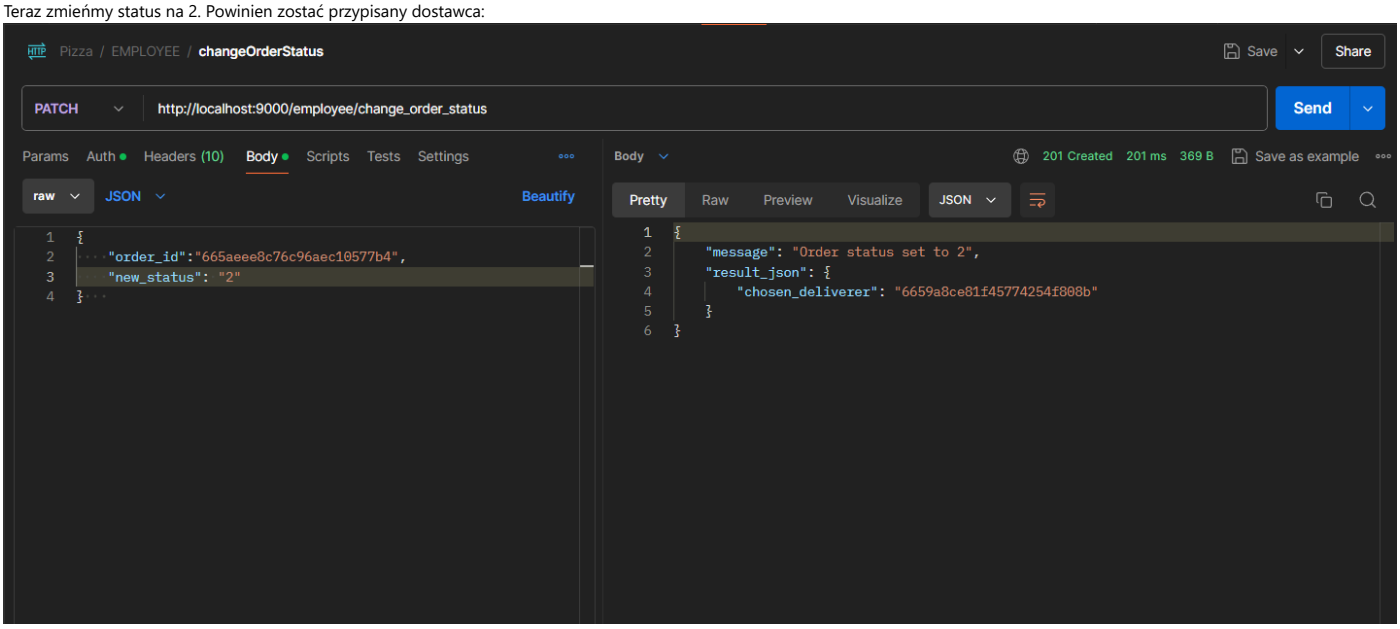
```
_id: ObjectId('665aeee8c76c96aec10577b4')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('6659a61a81f45774254f807f')
pizzas : Array (2)
  0: Object
    pizza_id : ObjectId('665350568029580187e349a5')
    current_price : 38
    count : 1
    discount : 0.1
    _id : ObjectId('665aeee8c76c96aec10577b5')
  1: Object
    pizza_id : ObjectId('665353038029580187e349b0')
    current_price : 42
    count : 4
    discount : 0
    _id : ObjectId('665aeee8c76c96aec10577b6')
client_address : Object
order_notes : ""
order_date : 2024-05-27T00:00:00.000+00:00
status : "0"
to_deliver : true
total_price : Object
  with_discount : 209.2
  without_discount : 213
  delivery_price : 7
discount_id : ObjectId('665478756fb0c2acffa100ae')
createdAt : 2024-06-01T09:50:32.026+00:00
updatedAt : 2024-06-01T09:50:32.026+00:00
__v : 0
```


Zmieńmy status na 1. To na razie zmienia tylko wartość statusu:



```
_id: ObjectId('665aeee8c76c96aec10577b4')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('6659a61a81f45774254f807f')
▾ pizzas : Array (2)
  ▾ 0: Object
      pizza_id : ObjectId('665350568029580187e349a5')
      current_price : 38
      count : 1
      discount : 0.1
      _id : ObjectId('665aeee8c76c96aec10577b5')
  ▾ 1: Object
      pizza_id : ObjectId('665353038029580187e349b0')
      current_price : 42
      count : 4
      discount : 0
      _id : ObjectId('665aeee8c76c96aec10577b6')
▸ client_address : Object
order_notes : ""
order_date : 2024-05-27T00:00:00.000+00:00
status : "1"
to_deliver : true
▸ total_price : Object
discount_id : ObjectId('665478756fb0c2acffa100ae')
createdAt : 2024-06-01T09:50:32.026+00:00
updatedAt : 2024-06-01T09:52:43.866+00:00
__v : 0
```

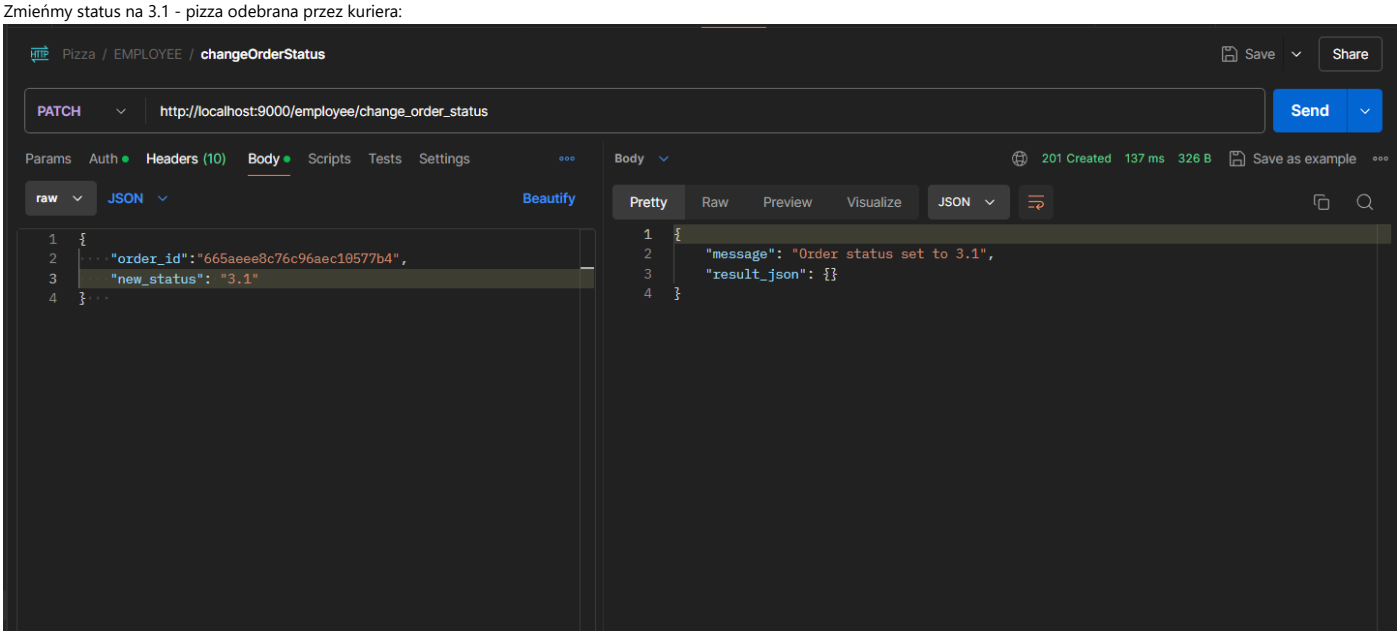

Teraz zmieńmy status na 2. Powinien zostać przypisany dostawca:



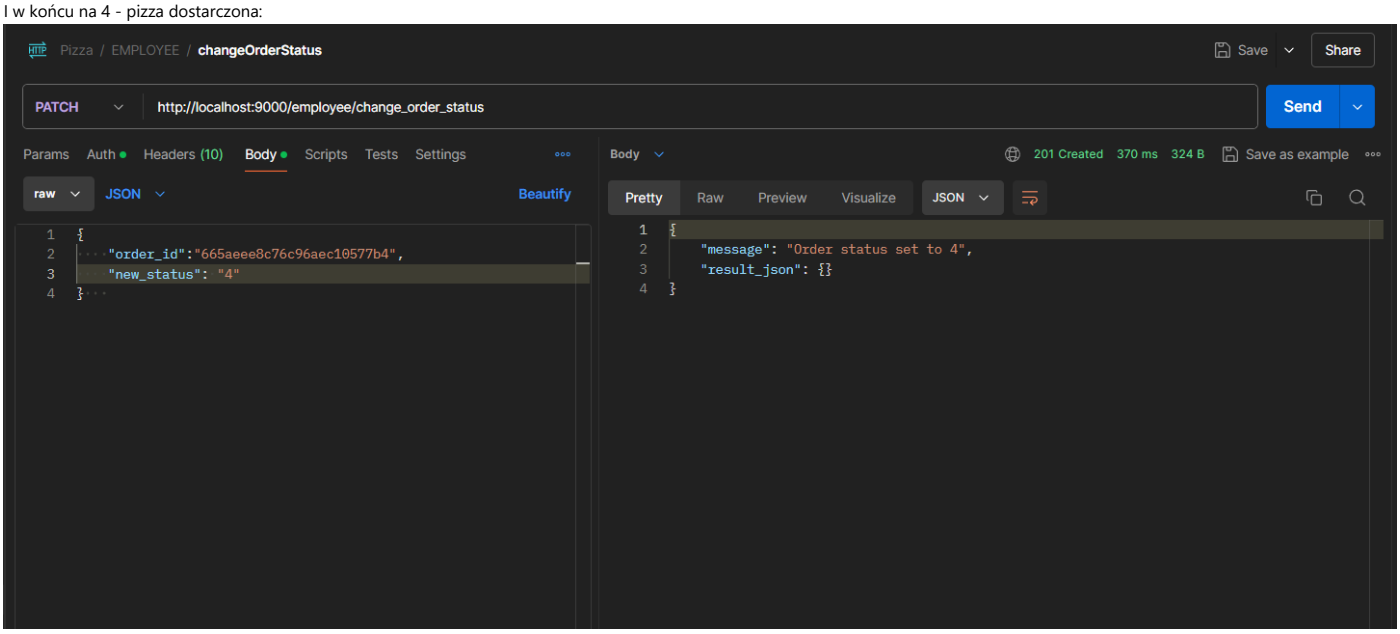
```
_id: ObjectId('665aeee8c76c96aec10577b4')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('6659a61a81f45774254f807f')
pizzas : Array (2)
  0: Object
      pizza_id : ObjectId('665350568029580187e349a5')
      current_price : 38
      count : 1
      discount : 0.1
      _id : ObjectId('665aeee8c76c96aec10577b5')
  1: Object
      pizza_id : ObjectId('665353038029580187e349b0')
      current_price : 42
      count : 4
      discount : 0
      _id : ObjectId('665aeee8c76c96aec10577b6')
client_address : Object
order_notes : ""
order_date : 2024-05-27T00:00:00.000+00:00
status : "2"
to_deliver : true
total_price : Object
discount_id : ObjectId('665478756fb0c2acffa100ae')
createdAt : 2024-06-01T09:50:32.026+00:00
updatedAt : 2024-06-01T09:55:12.544+00:00
__v : 0
deliverer_id : ObjectId('6659a8ce81f45774254f808b')
```


Status się zaktualizował i na końcu dokumentu dodało się pole deliverer_id.

Zmieńmy status na 3.1 - pizza odebrana przez kuriera:



I w końcu na 4 - pizza dostarczona:




```
_id: ObjectId('665aeee8c76c96aec10577b4')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('6659a61a81f45774254f807f')
pizzas : Array (2)
  0: Object
    pizza_id : ObjectId('6653505680295801 87e349a5')
    current_price : 38
    count : 1
    discount : 0.1
    _id : ObjectId('665aeee8c76c96aec10577b5')
  1: Object
    pizza_id : ObjectId('6653530380295801 87e349b0')
    current_price : 42
    count : 4
    discount : 0
    _id : ObjectId('665aeee8c76c96aec10577b6')
client_address : Object
order_notes : ""
order_date : 2024-05-27T00:00:00.000+00:00
status : "4"
to_deliver : true
total_price : Object
discount_id : ObjectId('665478756fb0c2acffa100ae')
createdAt : 2024-06-01T09:50:32.026+00:00
updatedAt : 2024-06-01T10:06:27.309+00:00
__v : 0
deliverer_id : ObjectId('6659a8ce81f45774254f808b')
```

```
_id: ObjectId('6659a8ce81f45774254f808b')
name : "Anna Nowak"
worker_type : "deliverer"
salary : 5300
phone : "777888999"
address : Object
status : "active"
current_orders : Array (empty)
orders_history : Array (1)
  0: ObjectId('665aeee8c76c96aec10577b4')
__v : 0
```

```
_id: ObjectId('6651d6cc5f48cfd8ded52387')
name : "Dariusz Nowak"
phone : "919853444"
address : Object
order_count : 8
discount_saved : 38
grades : Array (empty)
__v : 0
current_orders : Array (6)
orders_history : Array (2)
  0: ObjectId('6659b95b7ab6c70d660fab16')
  1: ObjectId('665aeee8c76c96aec10577b4')
```

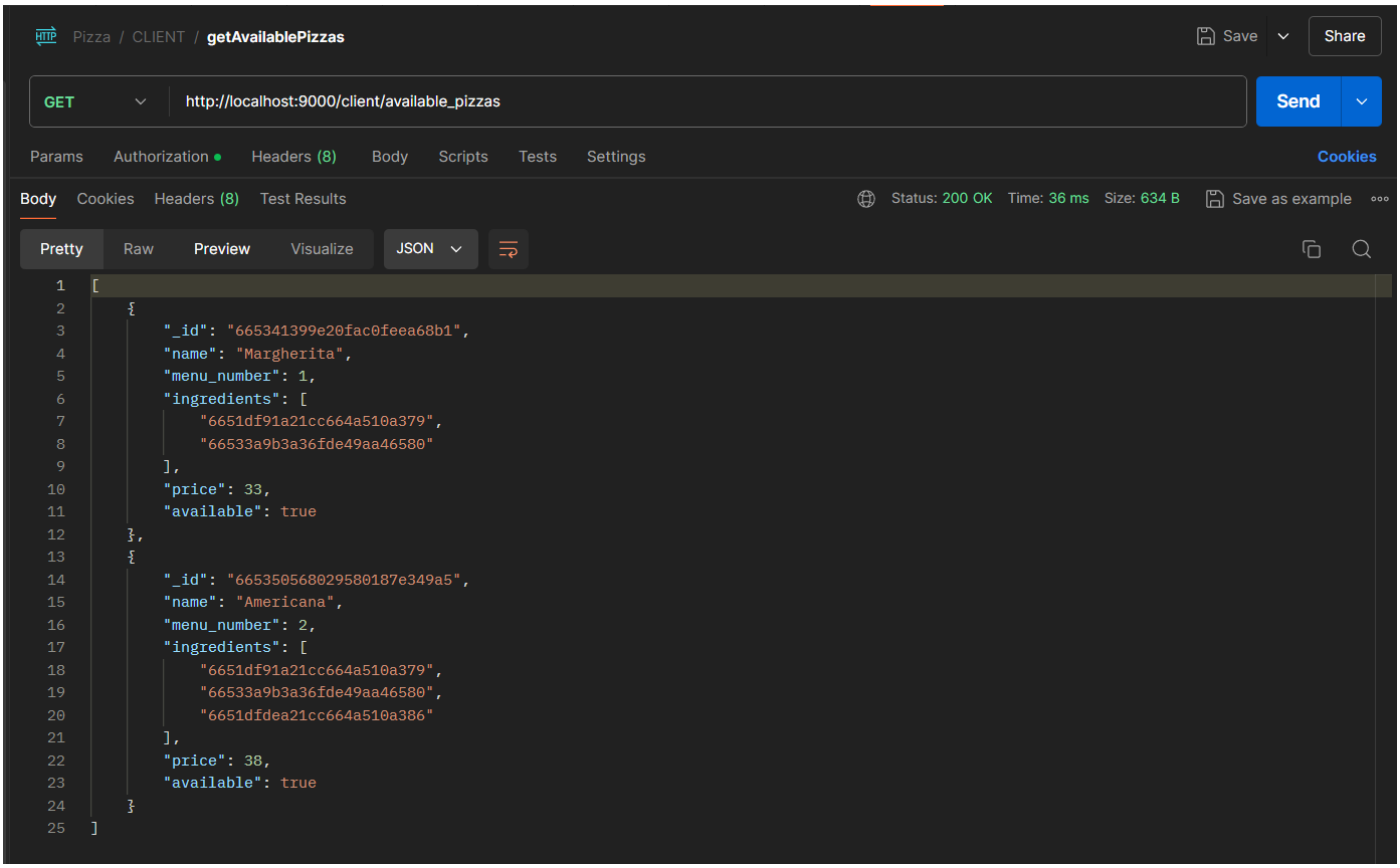

updateIngredientStatus

nie tylko zmieniamy status obecności składnika na stanie, ale także ustawiamy odpowiednio pole available w pizzas, jeśli obecność lub nieobecność jakiegoś składnika powoduje, że już pizza jest dostępna lub niedostępna.

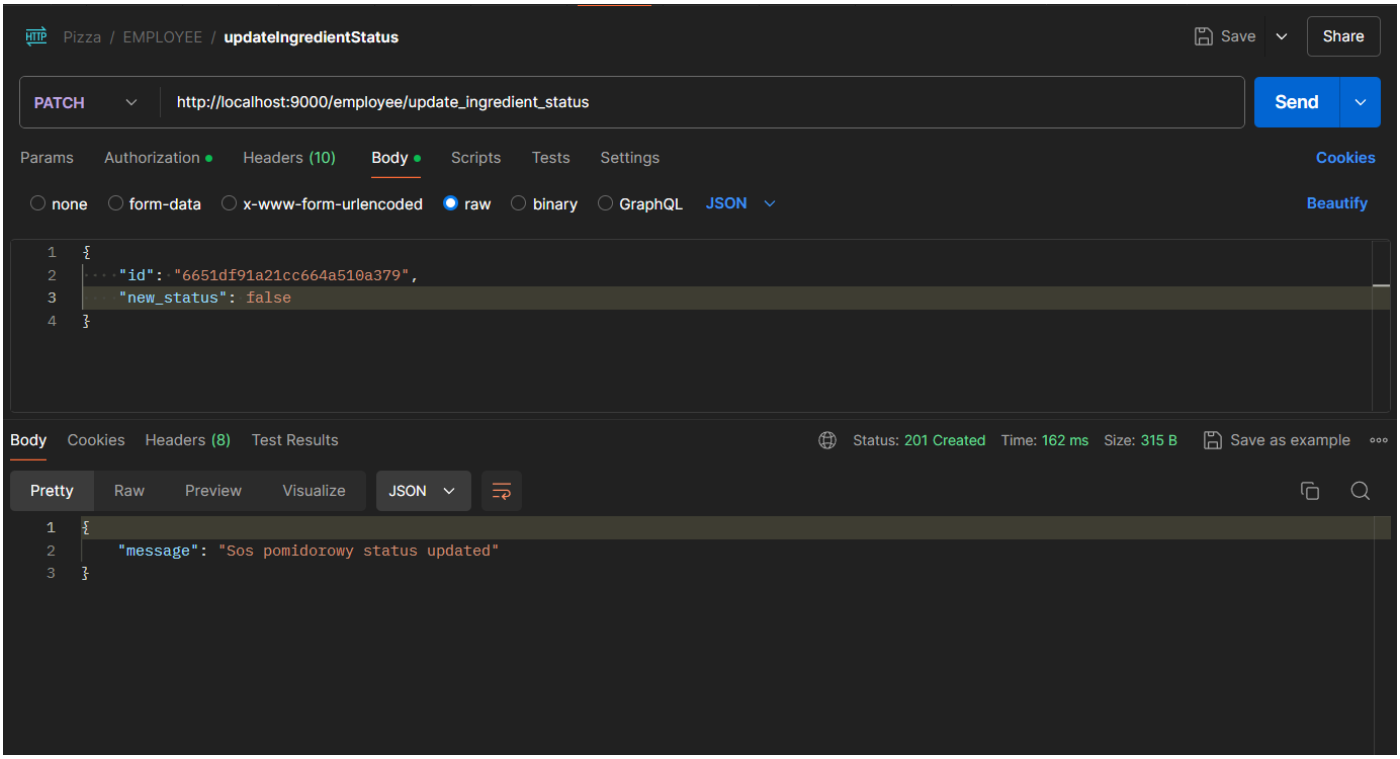
```
const updateIngredientStatus = asyncHandler(async (req, res, next) => {
  const session = await mongoose.startSession();
  await session.startTransaction();
  try {
    let { id, new_status } = req.body;
    id = new ObjectId(id);
    const the_ingredient = await Ingredient.findOne({_id: id});
```

```
    if (!the_ingredient) {
      res.status(400);
      throw new Error("Ingredient doesn't exist");
    }
    await Ingredient.updateOne({_id: id}, {available: new_status}, { session });
    const pizzasWithIngredient = await Pizza.aggregate([
      { $match: { ingredients: id } },
    ], { session });

    for (const pizza of pizzasWithIngredient) {
      let allOtherIngredientsAvailable = true;
      if (new_status){
        const otherIngredients = pizza.ingredients.filter(ingredient_id => !ingredient_id.equals(id));

        allOtherIngredientsAvailable = await Ingredient.countDocuments(
            { _id: { $in: otherIngredients }, available: true },
            { session }
        ) === otherIngredients.length;
      }
      if (allOtherIngredientsAvailable) {
        await Pizza.updateOne(
            { _id: pizza._id },
            { $set: { available: new_status } },
            { session }
        );
      }
    }
    await session.commitTransaction();
    res.status(201).json({message: `${the_ingredient.name} status updated`});
  } catch(err) {
    await session.abortTransaction();
    next(err);
  } finally {
    await session.endSession();
  }
});
```

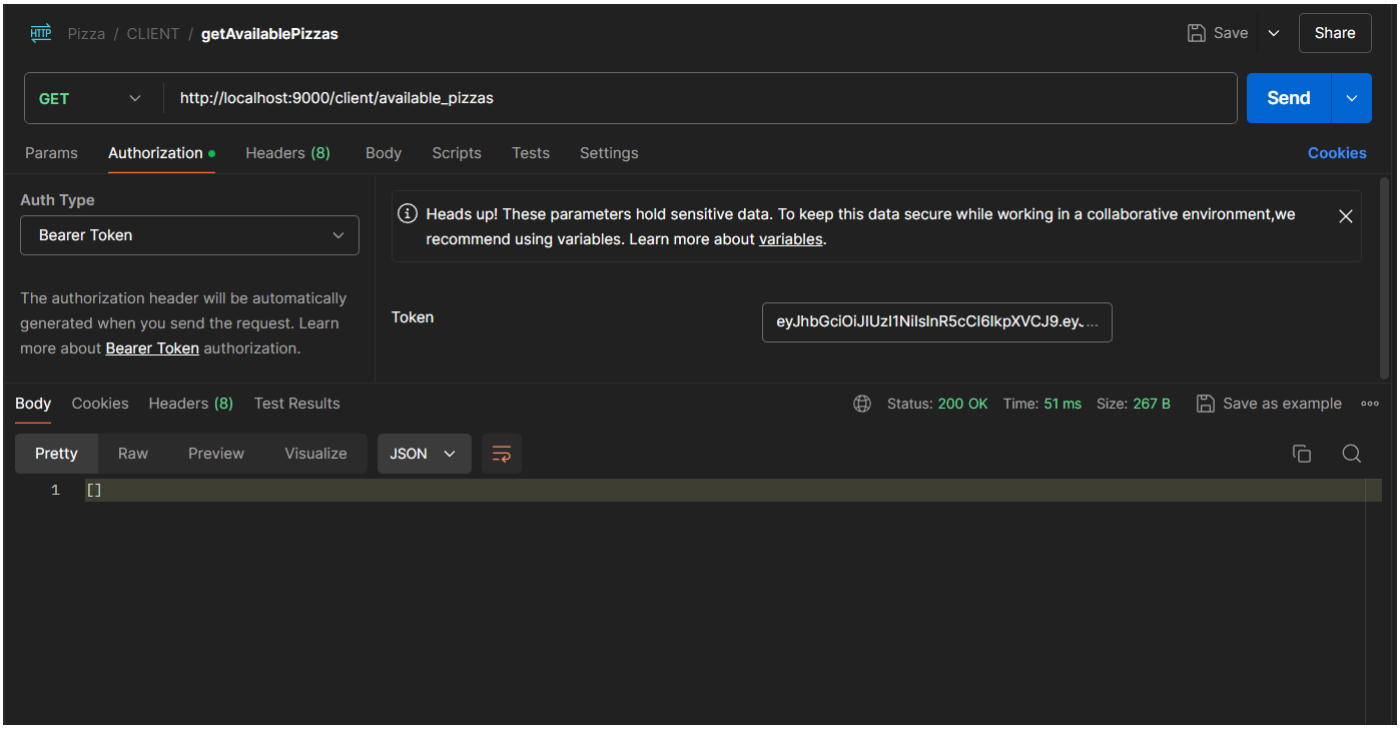
```
_id: ObjectId('6651df91a21cc664a510a379')
name : "Sos pomidorowy"
vegan : true
vegetarian : true
available : true
__v : 0
```

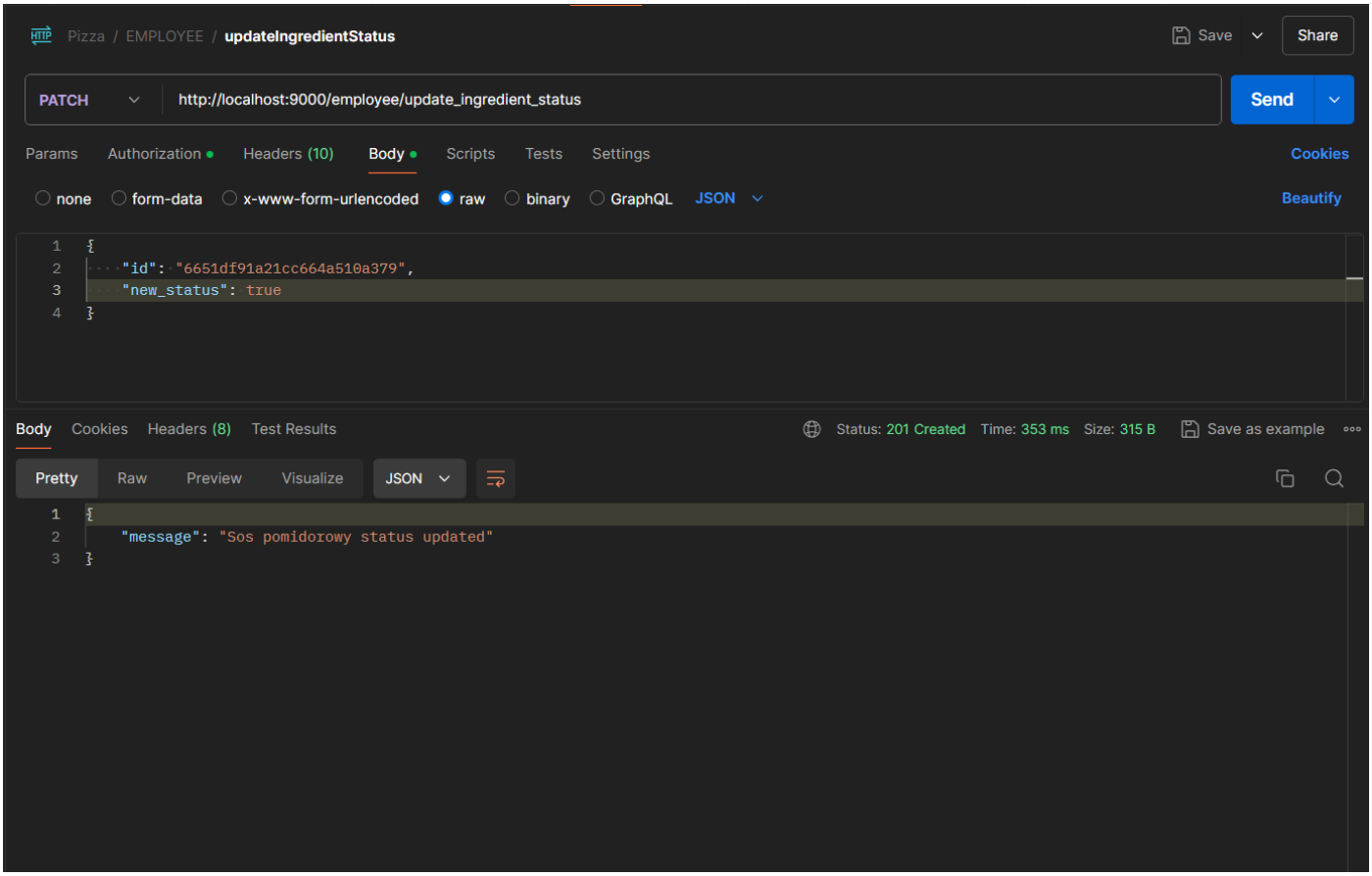


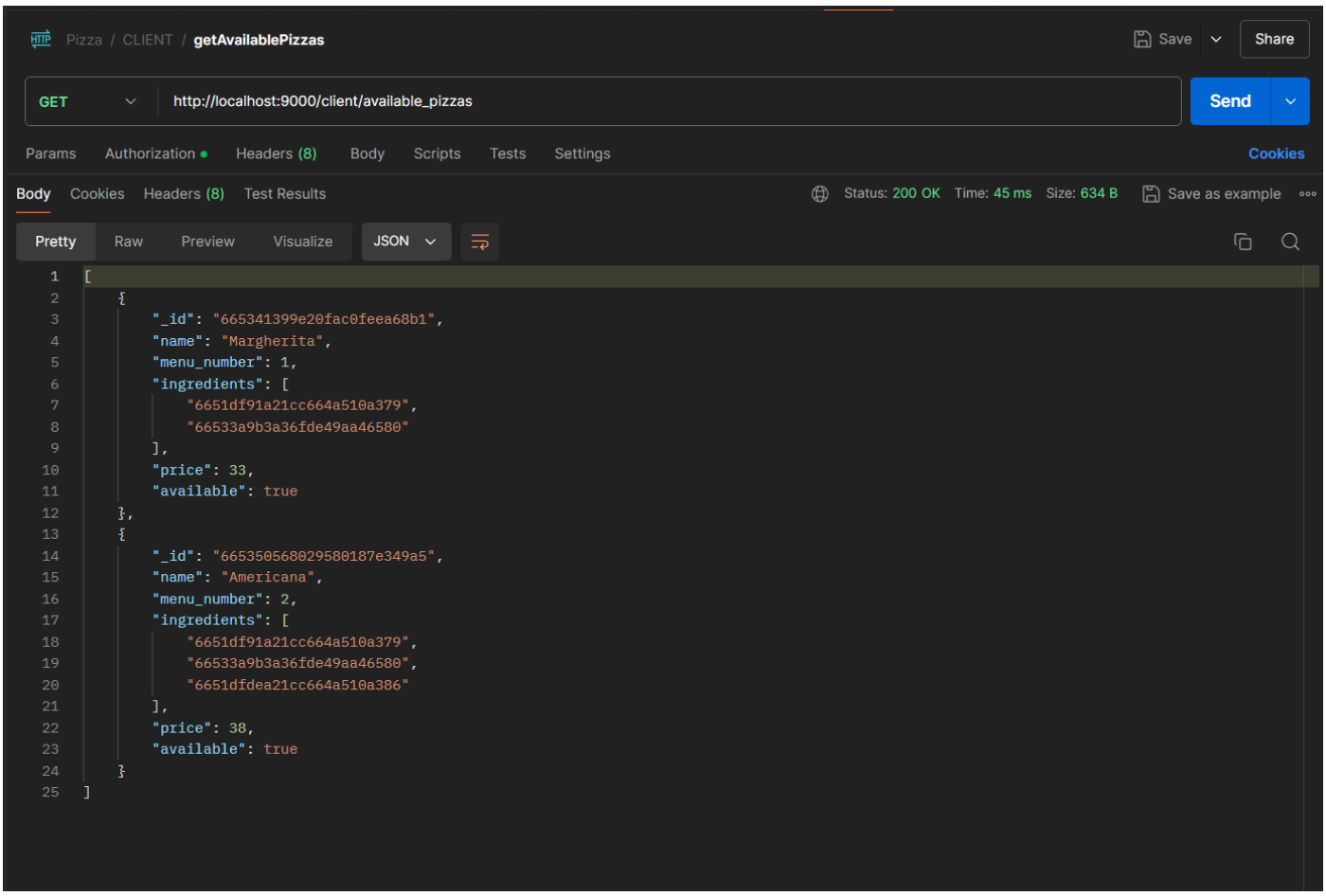
(Obie pizze zawierają Sos pomidorowy)




```
_id: ObjectId('6651df91a21cc664a510a379')
name : "Sos pomidorowy"
vegan : true
vegetarian : true
available : false
__v : 0
```



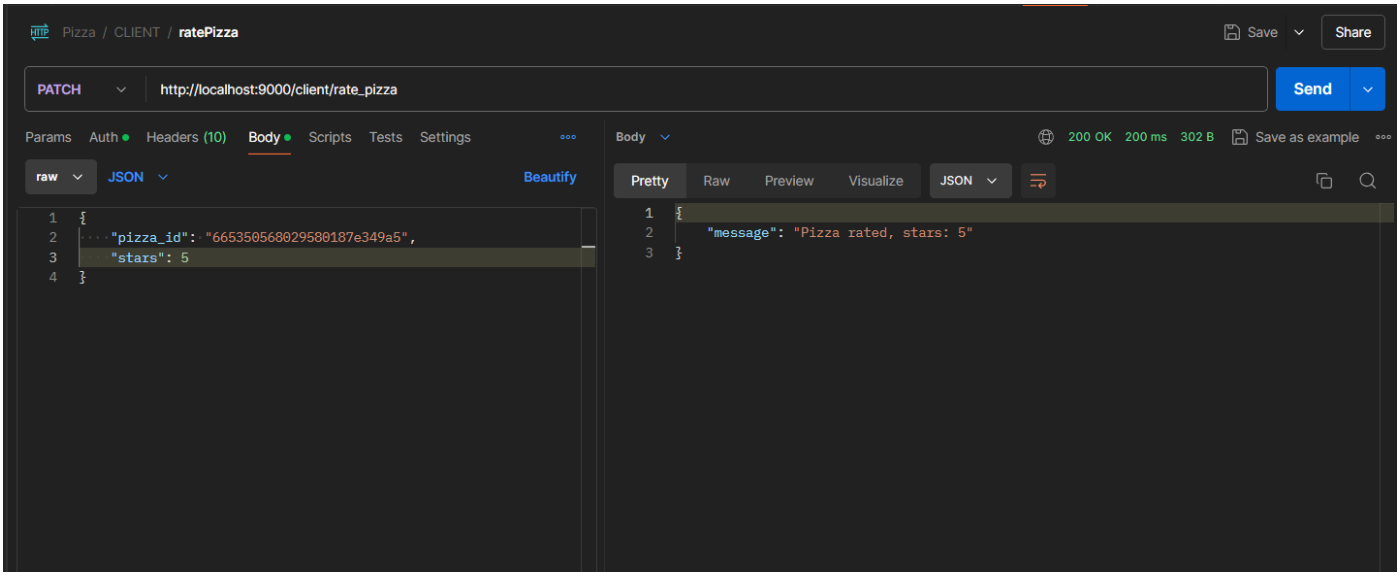




ratePizza

```
const ratePizza = asyncHandler(async (req, res, next) => {
  const session = await mongoose.startSession();
  await session.startTransaction();
  try {
    const {email, id, role} = req.user;
```

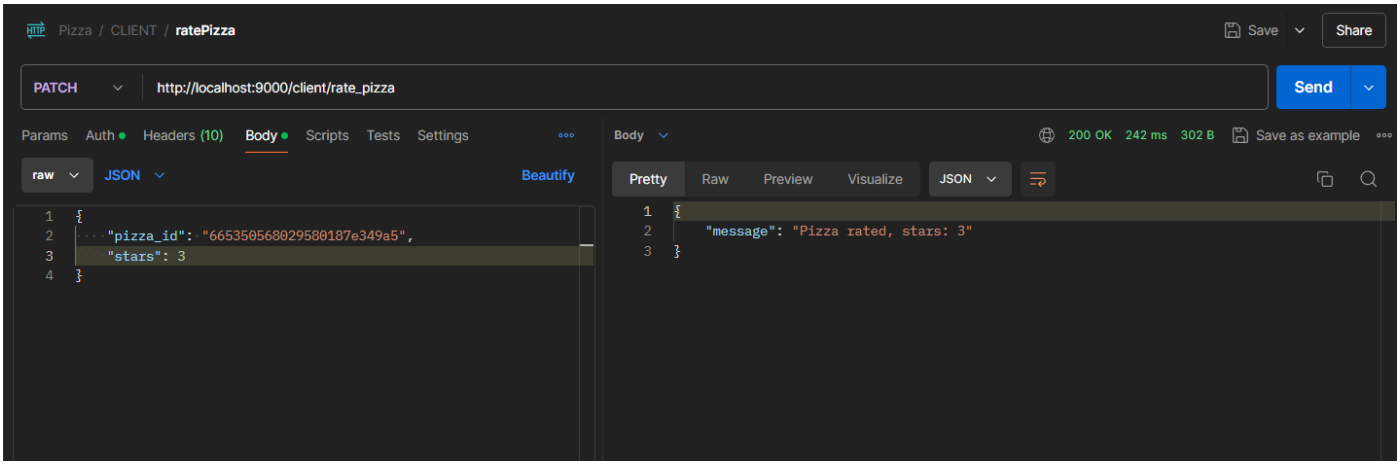
```
    const {pizza_id, stars} = req.body;
    const id_ObjId = new ObjectId(id);
    const pizza_id_ObjId = new ObjectId(pizza_id);
    const existingOrderWithThisPizza = await Order.findOne({
      client_id: id_ObjId,
      status: {$in: ['3.2', '4']},
      "pizzas.pizza_id": pizza_id_ObjId
    }, null, {session});
    if (!existingOrderWithThisPizza) {
      res.status(400);
      throw new Error("You haven't yet finished an order with this pizza");
    }
    const existingGrade = await Client.findOne({_id: id_ObjId, "grades.pizza_id": pizza_id_ObjId});
    if (existingGrade) {
      const currentStars = existingGrade.grades.find((item) => item.pizza_id = pizza_id_ObjId).stars;
      await Client.updateOne({_id: id_ObjId}, {$pull: {grades: {pizza_id: pizza_id_ObjId}}}, {session});
      await Pizza.updateOne({_id: pizza_id_ObjId}, {$inc: {"grades.points_sum": -currentStars, "grades.grade_count": -1}}, {session});
    }
    await Pizza.updateOne({_id: pizza_id_ObjId}, {$inc: {"grades.points_sum": stars, "grades.grade_count": 1}}, {session});
    await Client.updateOne({_id: id_ObjId}, {$push: {grades: {pizza_id: pizza_id_ObjId, stars}}}, {runValidators: true, session});
    await session.commitTransaction();
    res.status(200).json({
      message: `Pizza rated, stars: ${stars}`
    })
  } catch(error) {
    await session.abortTransaction();
    next(error);
  } finally {
    await session.endSession();
  }
});
```




```
_id: ObjectId('6651d6cc5f48cfd8ded52387')
name : "Dariusz Nowak"
phone : "919853444"
address : Object
order_count : 10
discount_saved : 41.8
grades : Array (2)
  0: Object
  1: Object
    pizza_id : ObjectId('665350568029580187e349a5')
    stars : 5
    _id : ObjectId('665b22bd827b7a5db56f4a3f')
__v : 0
current_orders : Array (4)
orders_history : Array (4)
```

```
_id: ObjectId('665350568029580187e349a5')
name : "Americana"
menu_number : 2
ingredients : Array (3)
price : 38
available : true
grades : Object
  points_sum : 5
  grade_count : 1
__v : 0
```

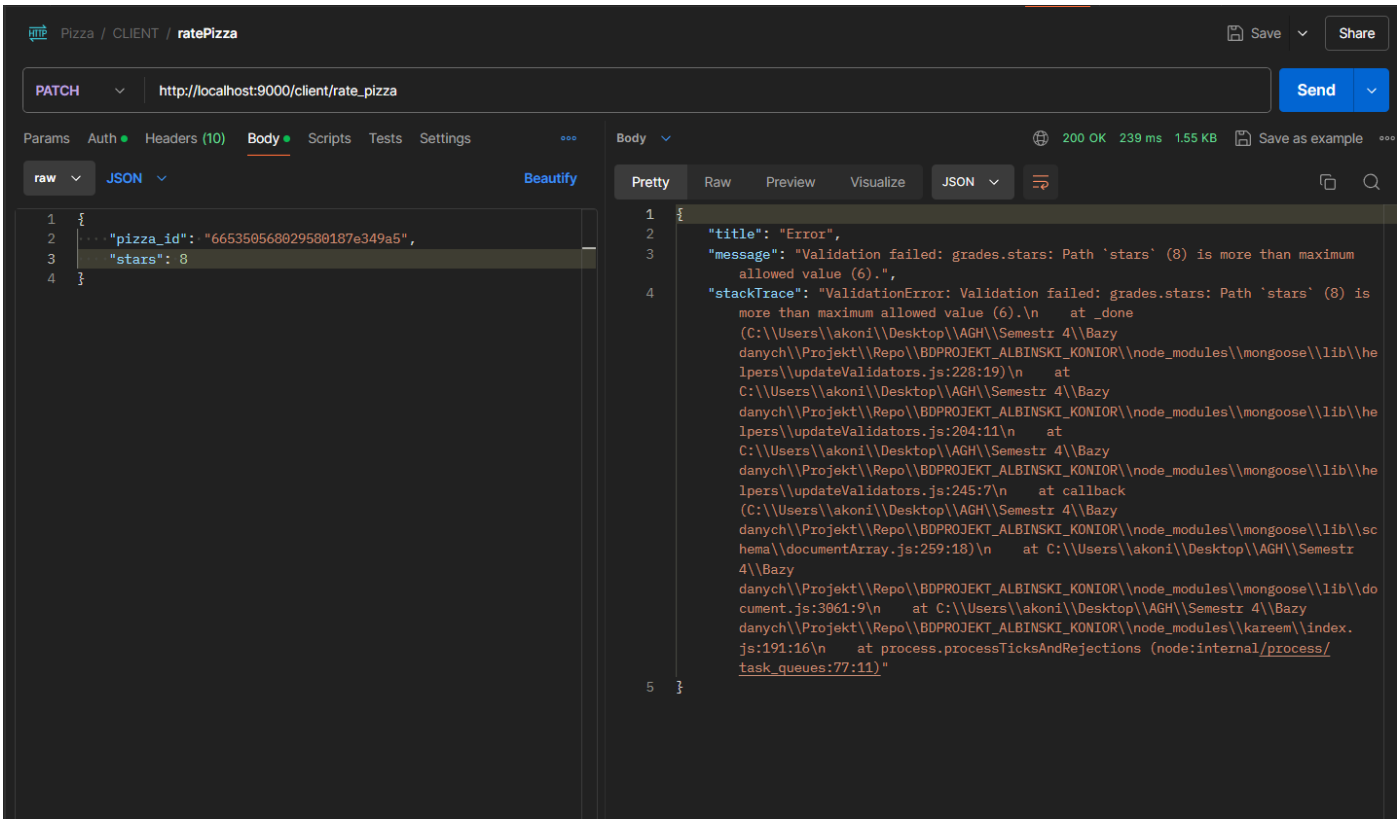
Zmieńmy ocenę(tego samego klienta, tej samej pizzy) na inną:




```
_id: ObjectId('665350568029580187e349a5')
name : "Americana"
menu_number : 2
▸ ingredients : Array (3)
price : 38
available : true
▾ grades : Object
    points_sum : 3
    grade_count : 1
__v : 0
```

```
_id: ObjectId('6651d6cc5f48cfd8ded52387')
name : "Dariusz Nowak"
phone : "919853444"
▸ address : Object
order_count : 10
discount_saved : 41.8
▾ grades : Array (2)
  ▸ 0: Object
  ▾ 1: Object
      pizza_id : ObjectId('665350568029580187e349a5')
      stars : 3
      _id : ObjectId('665b2320827b7a5db56f4a4a')
__v : 0
▸ current_orders : Array (4)
▸ orders_history : Array (4)
```

Przetestujmy błędy: Przetestujmy transakcję. W kolekcji clients mamy constraint na `grades.stars`, aby było w zakresie 1-6. Spróbujmy ustawić ocenę na np. 8. Zobaczmy, czy zaktualizuje się points_sum w pizzy(to zmieniamy jako pierwsze) - nie powinno.

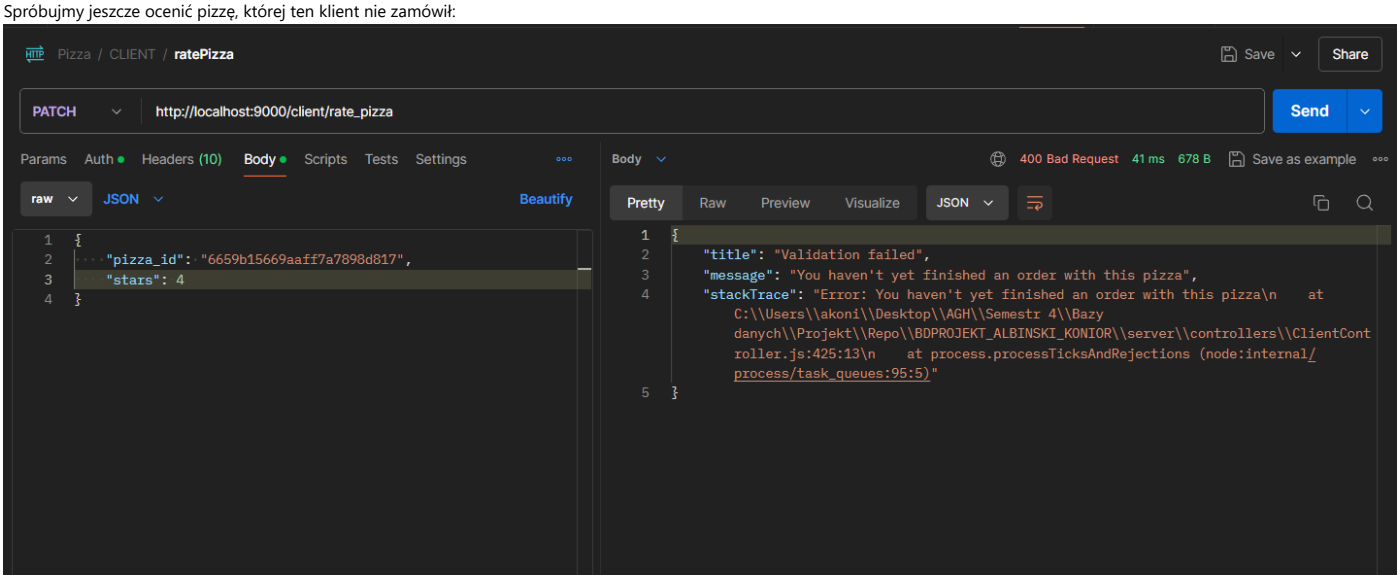



```
_id: ObjectId('665350568029580187e349a5')
name : "Americana"
menu_number : 2
▸ ingredients : Array (3)
price : 38
available : true
▾ grades : Object
    points_sum : 3
    grade_count : 1
__v : 0
```

Nie zaktualizowało się. Nie znikła także ocena z kolekcji clients:

```
_id: ObjectId('6651d6cc5f48cfd8ded52387')
name : "Dariusz Nowak"
phone : "919853444"
▸ address : Object
order_count : 10
discount_saved : 41.8
▾ grades : Array (2)
  ▸ 0: Object
  ▾ 1: Object
      pizza_id : ObjectId('665350568029580187e349a5')
      stars : 3
      _id : ObjectId('665b2320827b7a5db56f4a4a')
__v : 0
▸ current_orders : Array (4)
▸ orders_history : Array (4)
```

Spróbujmy jeszcze ocenić pizzę, której ten klient nie zamówił:



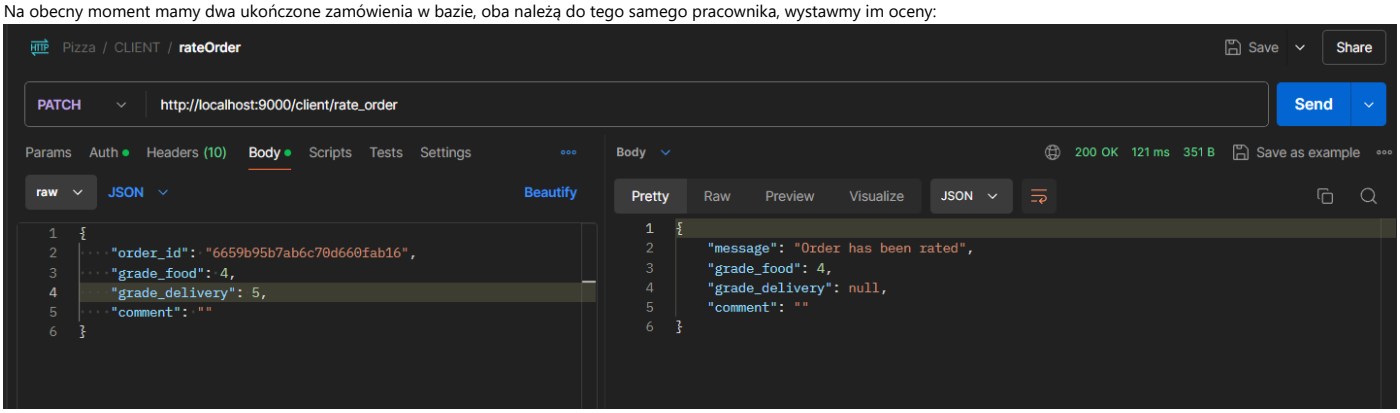
bestRatedEmployees

Najlepiej oceniani pracownicy kuchni(w danym okresie): Pracownicy są oceniani w kolekcji orders. Mamy tam pole `grade`, które zawiera pola `grade_food`, `grade_delivery`, `comment`.

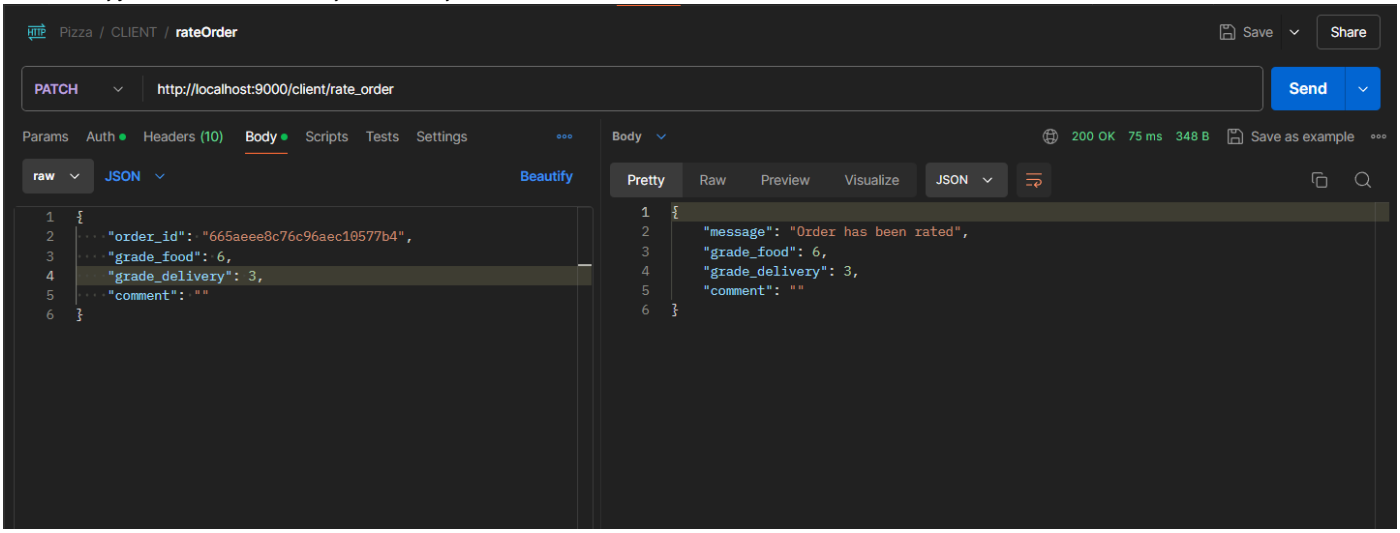
```
const bestRatedEmployees = asyncHandler(async (req, res, next) => {
  try {
    let {limit, date_from, date_to} = req.body;
    if (!limit) {
      throw new Error("Please provide a limit");
    }
    if (!date_from) {
      date_from = new Date(0);
    }
    if (!date_to) {
      date_to = new Date();
    }

    const result = await Order.aggregate([
      {
        $match: {
          "grade": { $exists: true },
          order_date: {
            $gte: date_from,
            $lte: date_to
          }
        }
      },
      {
        $group: { // po prostu grupujemy pracowników i obliczamy dla każdego średnią ocenę grade_food
          _id: {
            employee_id: "$employee_id"
          },
          avg_grade_for_food: { $avg: "$grade.grade_food" }
        }
      },
      {
        $lookup: {
          from: "workers",
          localField: "_id.employee_id",
          foreignField: "_id",
          as: "employee_details"
        }
      },
      {
        $unwind: "$employee_details"
      },
      {
        $project: {
          _id: 0,
          employee_name: "$employee_details.name",
          avg_grade_for_food: 1,
        }
      },
      {
        $sort: { avg_grade_for_food: -1 }
      },
      {
        $limit: limit
      }
    ]);
    res.status(200).json(result);
  } catch(error) {
```

```
    next(error);
  }
});
```

Na obecny moment mamy dwa ukończone zamówienia w bazie, oba należą do tego samego pracownika, wystawmy im oceny:

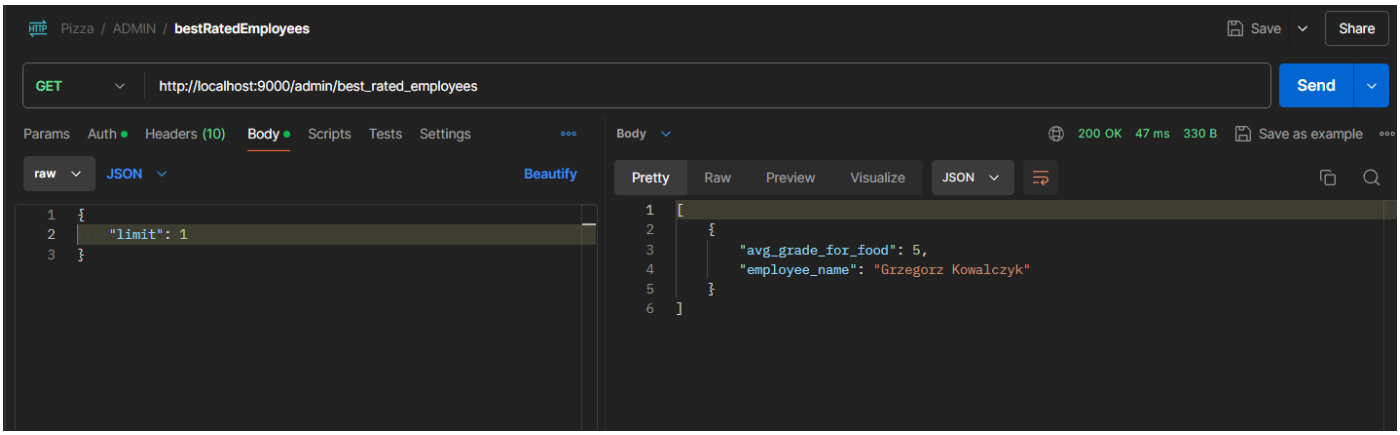


(Grade delivery jest na null, bo zamówienie było bez dostawy)

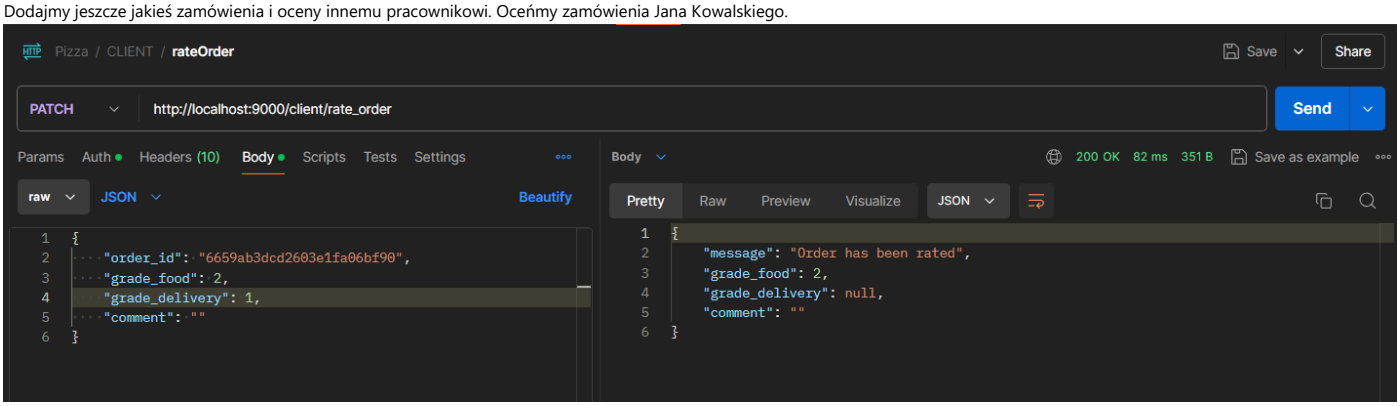


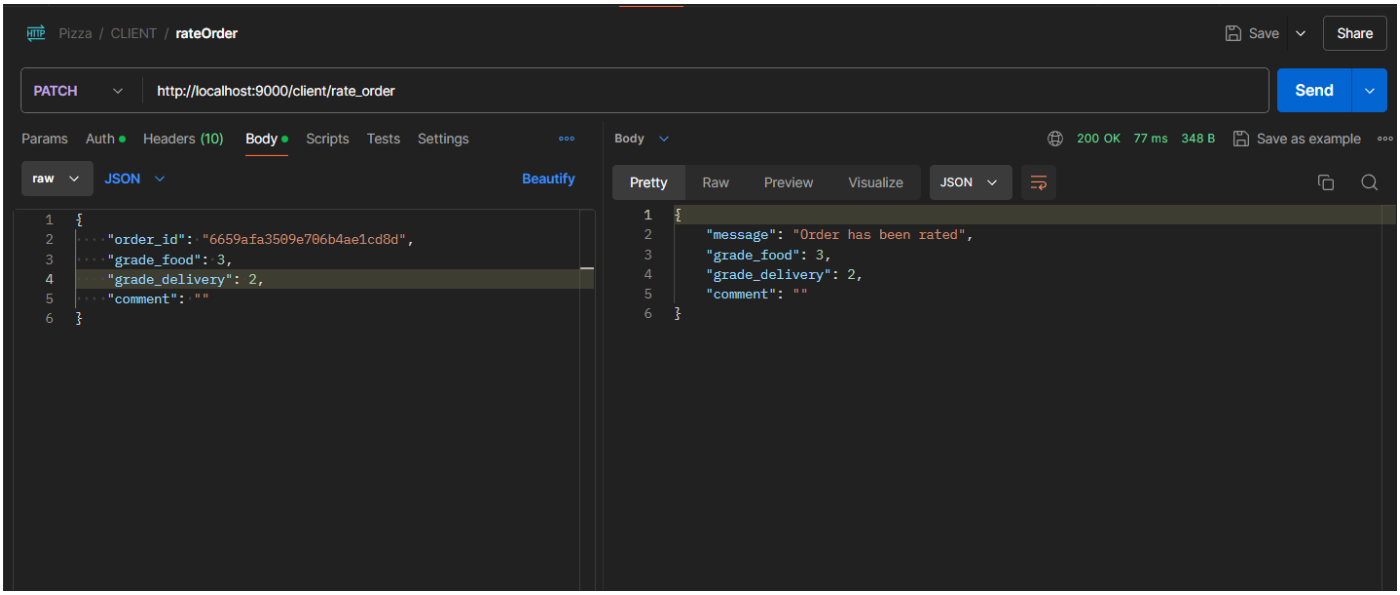

```
__v : 0
▾ grade : Object
    grade_food : 4
    grade_delivery : null
    comment : ""
    _id : ObjectId('665b0036c05e9df639fd7e73')
```

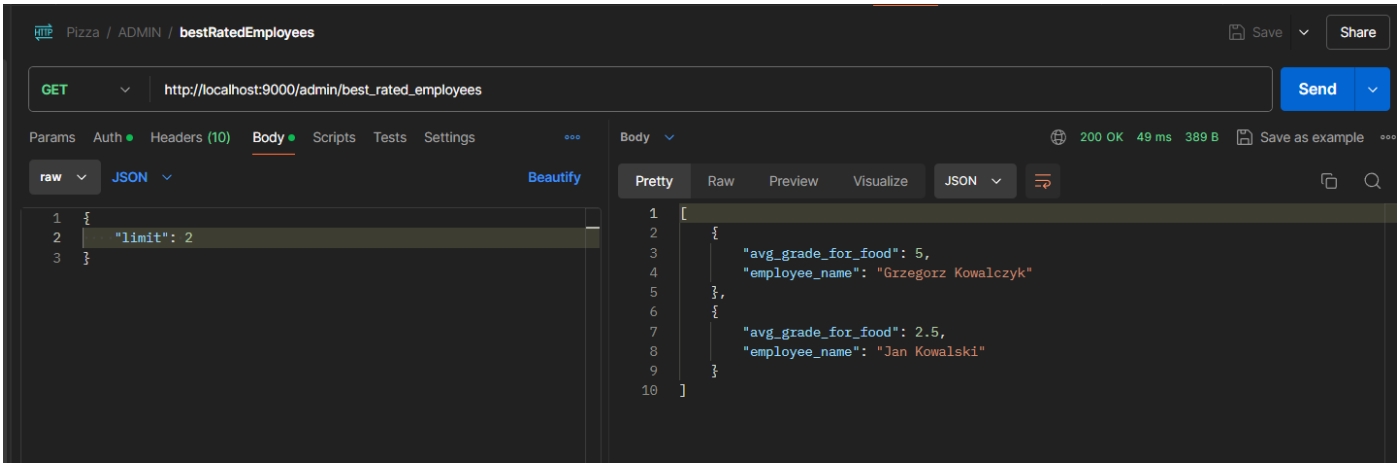
```
_id: ObjectId('665aeee8c76c96aec10577b4')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('6659a61a81f45774254f807f')
▸ pizzas : Array (2)
▸ client_address : Object
order_notes : ""
order_date : 2024-05-27T00:00:00.000+00:00
status : "4"
to_deliver : true
▸ total_price : Object
discount_id : ObjectId('665478756fb0c2acffa100ae')
createdAt : 2024-06-01T09:50:32.026+00:00
updatedAt : 2024-06-01T11:05:06.730+00:00
__v : 0
deliverer_id : ObjectId('6659a8ce81f45774254f808b')
▾ grade : Object
    grade_food : 6
    grade_delivery : 3
    comment : ""
    _id : ObjectId('665b0062c05e9df639fd7e78')
```



Dodajmy jeszcze jakieś zamówienia i oceny innemu pracownikowi. Oceńmy zamówienia Jana Kowalskiego.





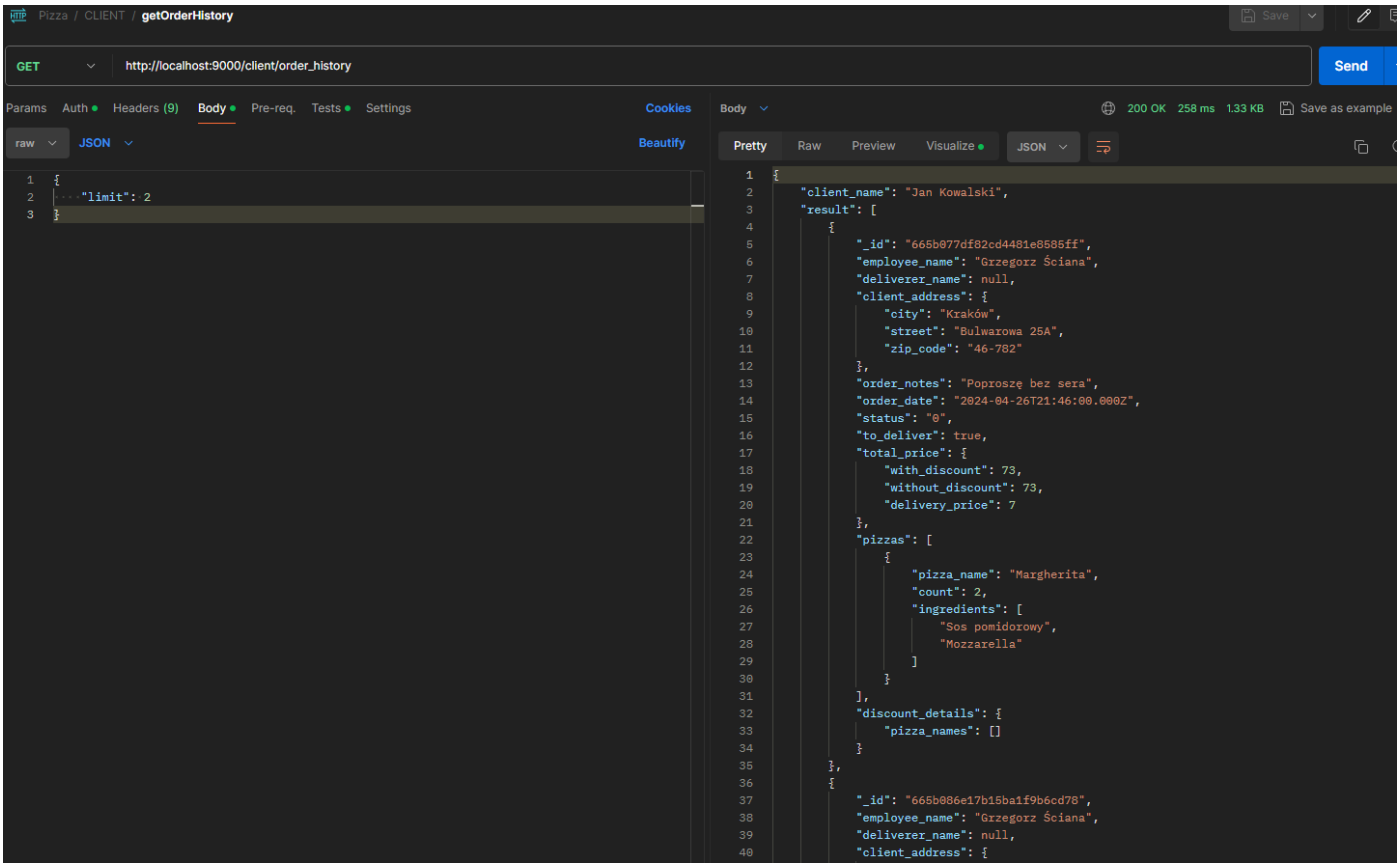


getOrderHistory

```
const getOrderHistory = asyncHandler(async (req, res, next) => {
  try {
```

```
const {email, id, role} = req.user;
let {limit, date_from, date_to} = req.body;
if(!limit) {
  throw new Error("Please provide a limit");
}
if( !date_from ) {
  date_from = new Date(0);
}
if( !date_to ) {
  date_to = new Date();
}
const id_ObjId = new ObjectId(id);
const the_client = await Client.findOne({_id: id_ObjId});
const result = await Order.aggregate([
  {
    $match: {
      "client_id": id_ObjId,
      "order_date": {
        $gte: date_from,
        $lte: date_to
      }
    }
  },
  {
    $sort: { order_date: -1 }
  },
  {
    $limit: limit
  },
  {
    $unwind: "$pizzas"
  },
  {
    $lookup: {
      from: "pizzas",
      localField: "pizzas.pizza_id",
      foreignField: "_id",
      as: "pizza_details"
    }
  },
  {
    $unwind: {
      path: "$pizza_details",
      preserveNullAndEmptyArrays: true
    }
  },
  {
    $lookup: {
      from: "ingredients",
      localField: "pizza_details.ingredients",
      foreignField: "_id",
      as: "ingredient_details"
    }
  },
  {
    $lookup: {
      from: "discounts",
      localField: "discount_id",
      foreignField: "_id",
      as: "discount_details"
    }
  },
  {
    $unwind: {
      path: "$discount_details",
      preserveNullAndEmptyArrays: true
    }
  },
  {
    $lookup: {
      from: "pizzas",
      localField: "discount_details.pizza_ids",
      foreignField: "_id",
      as: "discount_pizza_names"
    }
  },
  {
    $lookup: {
      from: "workers",
      localField: "employee_id",
      foreignField: "_id",
      as: "employee_details"
    }
  },
  {
    $unwind: {
      path: "$employee_details",
      preserveNullAndEmptyArrays: true
    }
  },
```

```
      {
        $lookup: {
          from: "workers",
          localField: "deliverer_id",
          foreignField: "_id",
          as: "deliverer_details"
        }
      },
      {
        $unwind: {
          path: "$deliverer_details",
          preserveNullAndEmptyArrays: true
        }
      },
      {
        $project: {
          "client_address._id": 0
        }
      },
      {
        $group: {
          _id: "$_id",
          employee_name: {$first: "$employee_details.name"},
          deliverer_name: {$first: "$deliverer_details.name"},
          client_address: {$first: "$client_address"},
          order_notes: {$first: "$order_notes"},
          order_date: {$first: "$order_date"},
          status: {$first: "$status"},
          to_deliver: {$first: "$to_deliver"},
          total_price: {$first: "$total_price"},
          pizzas: {
            $push: {
              pizza_price: "$pizzas.price",
              pizza_name: "$pizza_details.name",
              count: "$pizzas.count",
              ingredients: "$ingredient_details.name"
            }
          },
          discount_details: {
            $first: {
              discount_name: "$discount_details.name",
              discount_value: "$discount_details.value",
              pizza_names: "$discount_pizza_names.name"
            }
          }
        }
      }
    ]);
    console.log(result);
    res.status(200).json({
      client_name: the_client.name,
      result,
      date_from,
      date_to
    });
  } catch (error) {
    next(error);
  }
});
```

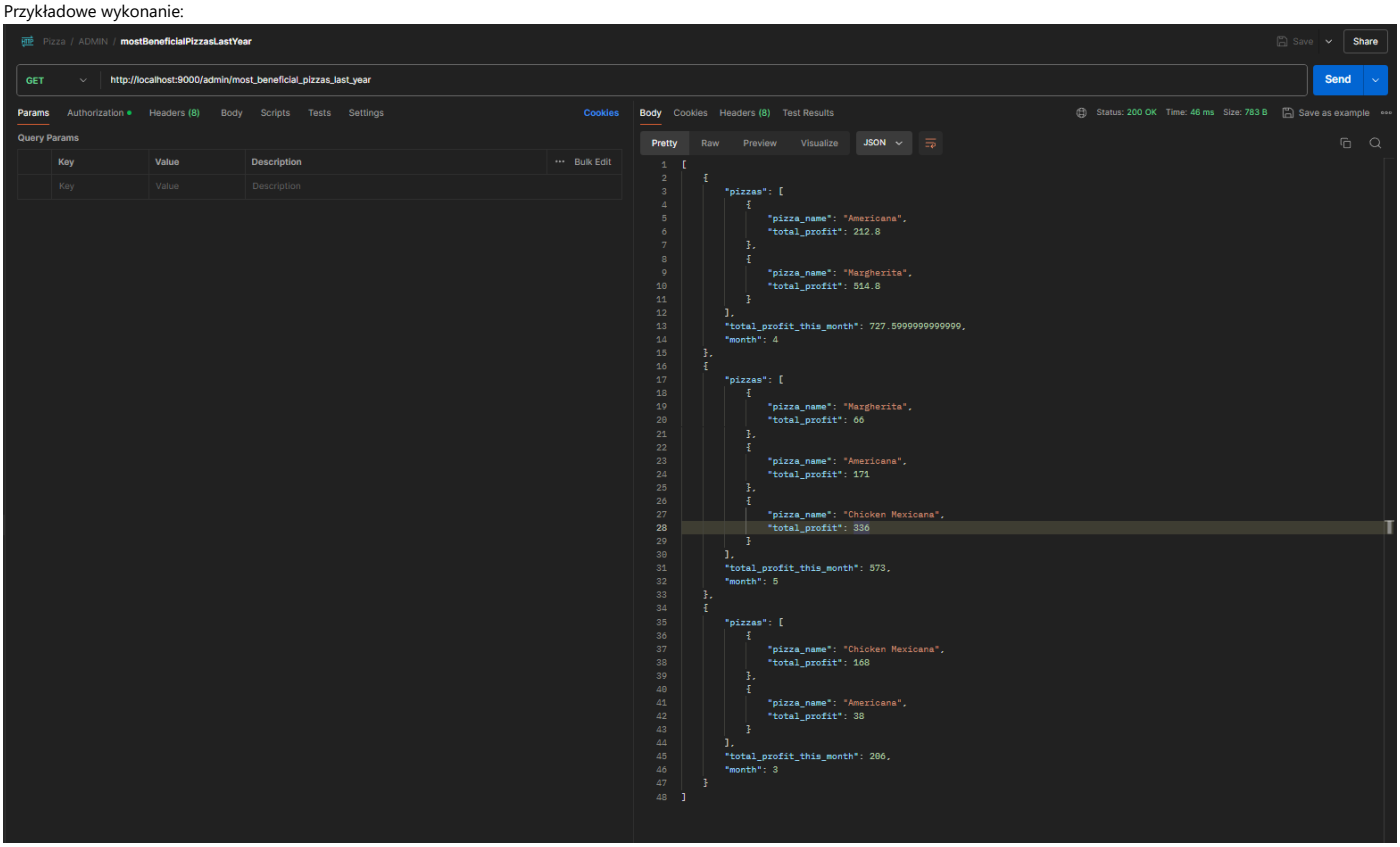


mostBeneficialPizzasLastYear

```
const mostBeneficialPizzasLastYear = asyncHandler(async (req, res, next) => {
  try {
    const now = new Date();
    const oneYearAgo = new Date(now);
    oneYearAgo.setFullYear(oneYearAgo.getFullYear() - 1);

    const result = await Order.aggregate([
      {
        $match: {
          order_date: {
            $gte: oneYearAgo,
            $lte: now
          }
        }
      },
      {
        $project: {
          pizzas: 1,
          order_date: 1
        }
      },
      {
        $unwind: "$pizzas"
      },
      {
        $group: {
          _id: {
            month: { $month: "$order_date" },
            pizza_id: "$pizzas.pizza_id"
          },
          total_profit: {
            $sum: {
              $multiply: [
                "$pizzas.current_price",
                "$pizzas.count",
                { $subtract: [1, "$pizzas.discount"] }
              ]
            }
          }
        }
      },
      {
        $lookup: {
          from: "pizzas",
          localField: "_id.pizza_id",
          foreignField: "_id",
          as: "pizza_details"
        }
```

```
      },
      {
        $unwind: "$pizza_details"
      },
      {
        $group: {
          _id: {
            month: "$_id.month"
          },
          pizzas: {
            $push: {
              pizza_name: "$pizza_details.name",
              total_profit: "$total_profit"
            }
          },
          total_profit_this_month: { $sum: "$total_profit" }
        }
      },
      {
        $project: {
          _id: 0,
          month: "$_id.month",
          pizzas: 1,
          total_profit_this_month: 1
        }
      },
      {
        $sort: {
          total_profit_this_month: -1
        }
      }
    ]);
    res.status(200).json(result);
  } catch(error) {
    next(error);
  }
});
```

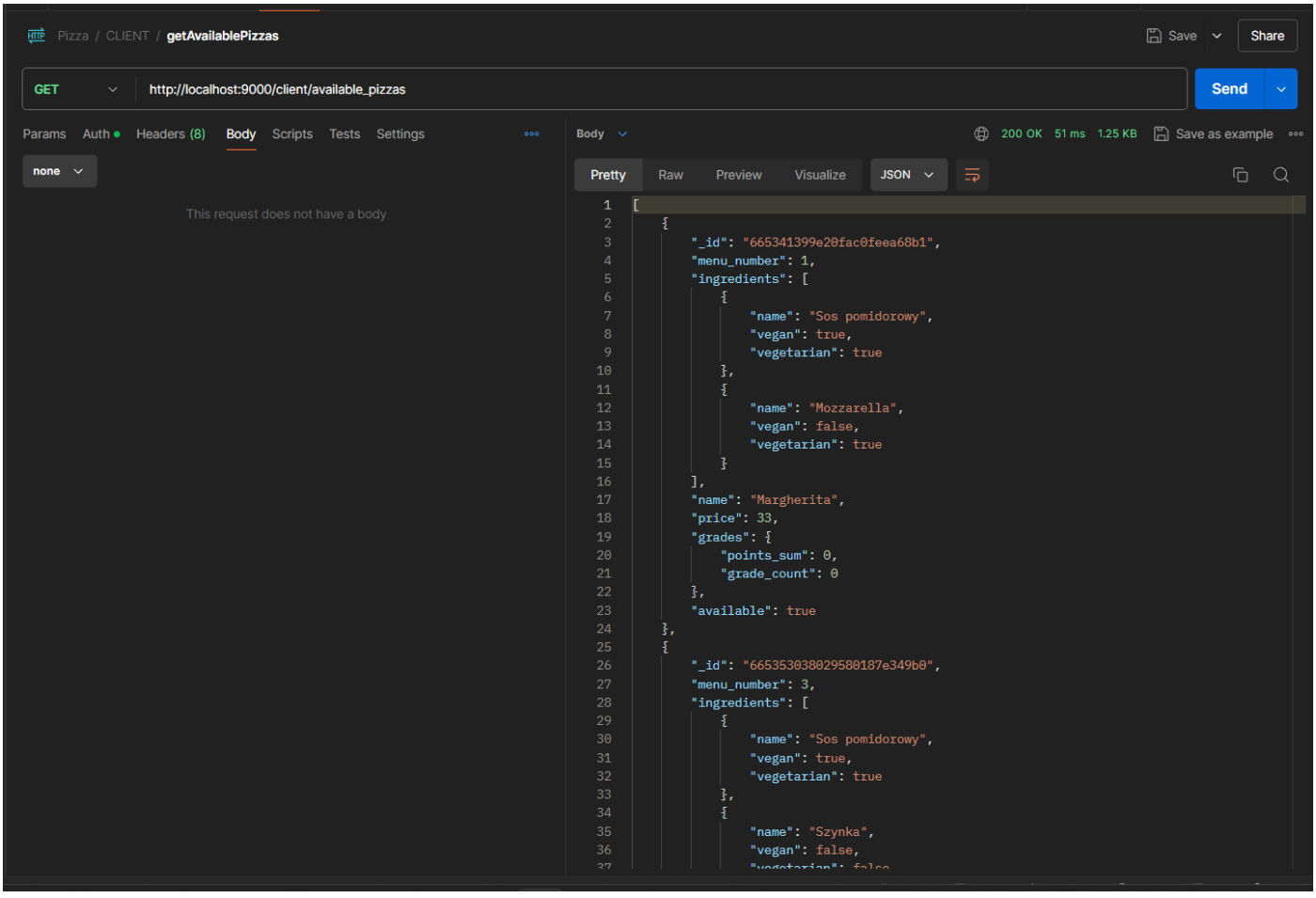
Przykładowe wykonanie:



getAvailablePizzas

```
const getAvailablePizzas = asyncHandler(async (req, res, next) => {
  try {
    const pizzas = await Pizza.aggregate([
      {
        $match: {available: true}
      },
      {
        $lookup: {
          from: "ingredients",
```

```
          localField: "ingredients",
          foreignField: "_id",
          as: "ingredient_details"
        }
      },
      {
        $unwind: "$ingredient_details"
      },
      {
        $group: {
          _id: "$_id",
          menu_number: {$first: "$menu_number"},
          ingredients: {
            $push: {
              name: "$ingredient_details.name",
              vegan: "$ingredient_details.vegan",
              vegetarian: "$ingredient_details.vegetarian"
            }
          },
          name: {$first: "$name"},
          price: {$first: "$price"},
          grades: {$first: "$grades"},
          available: {$first: "$available"}
        }
      }
    ]);
    res.status(200).json(pizzas);
  } catch (err) {
    next(err);
  }
});
```



(Ocen brak - wcześniej ocenialiśmy zamówienia, ocenianie pizz jest realizowane osobno)

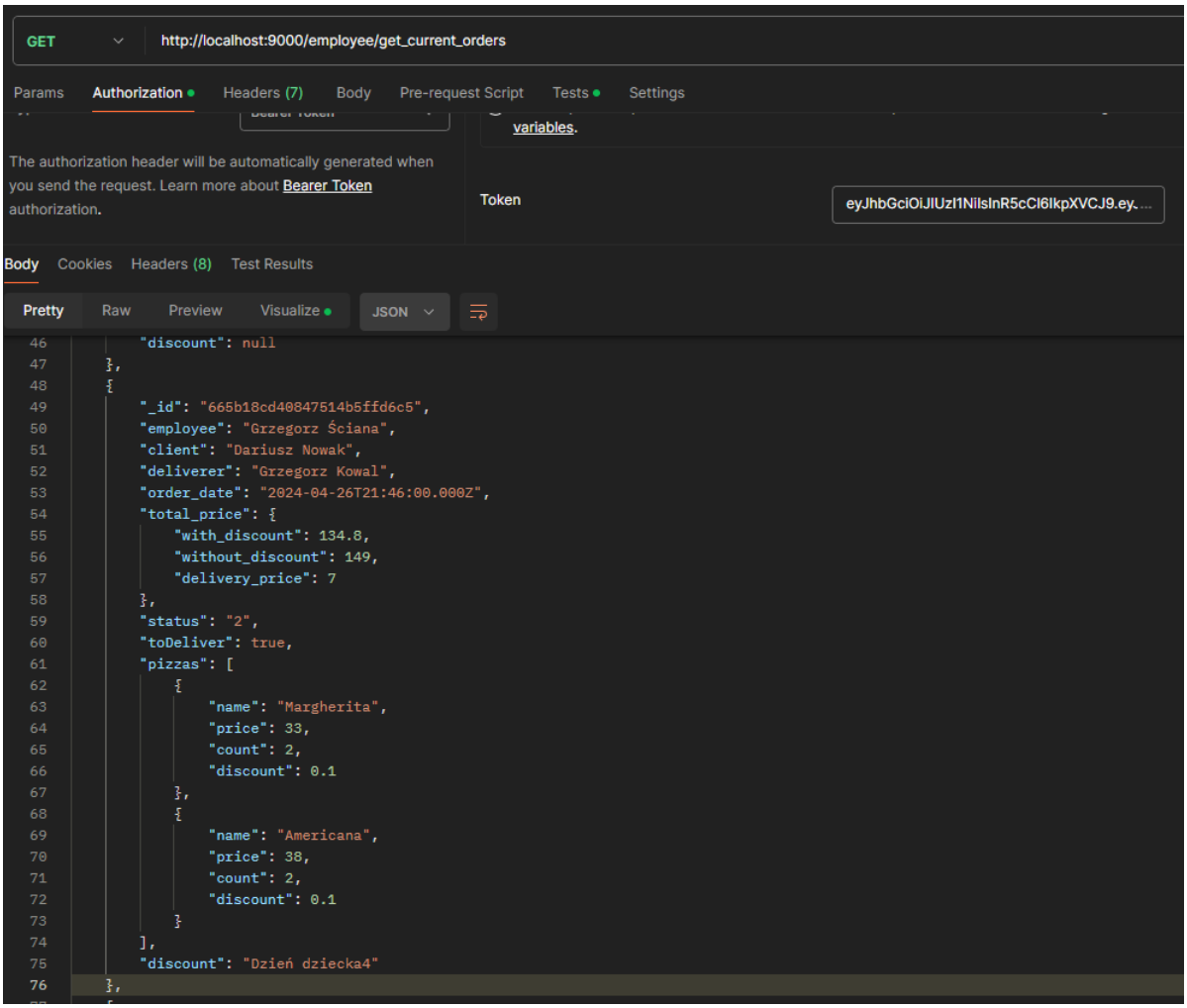
getCurrentOrders (dla pracownika)

Chcemy umożliwić widok pracownikowi obecnie realizowanych przez niego zamówień.

```
const getCurrentOrders = asyncHandler(async (req, res, next) => {
  try {
    const objId = new ObjectId(req.user.id);
    const orders = await Worker.aggregate([
      {
        $match: { _id: objId }
      },
      {
```

```
      $lookup: {
        from: "orders",
        localField: "current_orders",
        foreignField: "_id",
        as: "current_orders"
      }
    },
    {
      $unwind: "$current_orders"
    },
    {
      $lookup: {
        from: "clients",
        localField: "current_orders.client_id",
        foreignField: "_id",
        as: "current_orders.client"
      }
    },
    {
      $unwind: "$current_orders.client"
    },
    {
      $lookup: {
        from: "workers",
        localField: "current_orders.deliverer_id",
        foreignField: "_id",
        as: "current_orders.deliverer"
      }
    },
    {
      $unwind: {
        path: "$current_orders.deliverer",
        preserveNullAndEmptyArrays: true
      }
    },
    {
      $group: {
        _id: "$current_orders._id",
        employee: { $first: "$name" },
        client: { $first: "$current_orders.client.name" },
        deliverer: { $first: "$current_orders.deliverer.name" },
        order_date: { $first: "$current_orders.order_date" },
        pizzas: { $first: "$current_orders.pizzas" },
        total_price: { $first: "$current_orders.total_price" },
        discount_id: { $first: "$current_orders.discount_id" },
        grade: { $first: "$current_orders.grade" },
        status: { $first: "$current_orders.status" },
        toDeliver: { $first: "$current_orders.to_deliver" }
      }
    },
    {
      $lookup: {
        from: "pizzas",
        localField: "pizzas.pizza_id",
        foreignField: "_id",
        as: "pizza_details"
      }
    },
    {
      $lookup: {
        from: "discounts",
        localField: "discount_id",
        foreignField: "_id",
        as: "discount"
      }
    },
    {
      $unwind: {
        path: "$discount",
        preserveNullAndEmptyArrays: true
      }
    },
    {
      $project: {
        _id: 1,
        employee: 1,
        client: 1,
        deliverer: 1,
        order_date: 1,
        status: 1,
        toDeliver: 1,
        pizzas: {
          $map: {
            input: "$pizzas",
            as: "pizzaItem",
            in: {
              name: { $arrayElemAt: ["$pizza_details.name", { $indexOfArray: ["$pizza_details._id", "$$pizzaItem.pizza_id"] }] },
              price: { $arrayElemAt: ["$pizza_details.price", { $indexOfArray: ["$pizza_details._id", "$$pizzaItem.pizza_id"] }] },
              count: "$$pizzaItem.count",
              discount: "$$pizzaItem.discount"
```

```
            }
          }
        },
        total_price: 1,
        discount: { $ifNull: ["$discount.name", null] },
      }
    }
  ]);


  res.status(200).json(orders);

 } catch(err) {
  next(err);
 }

});
```

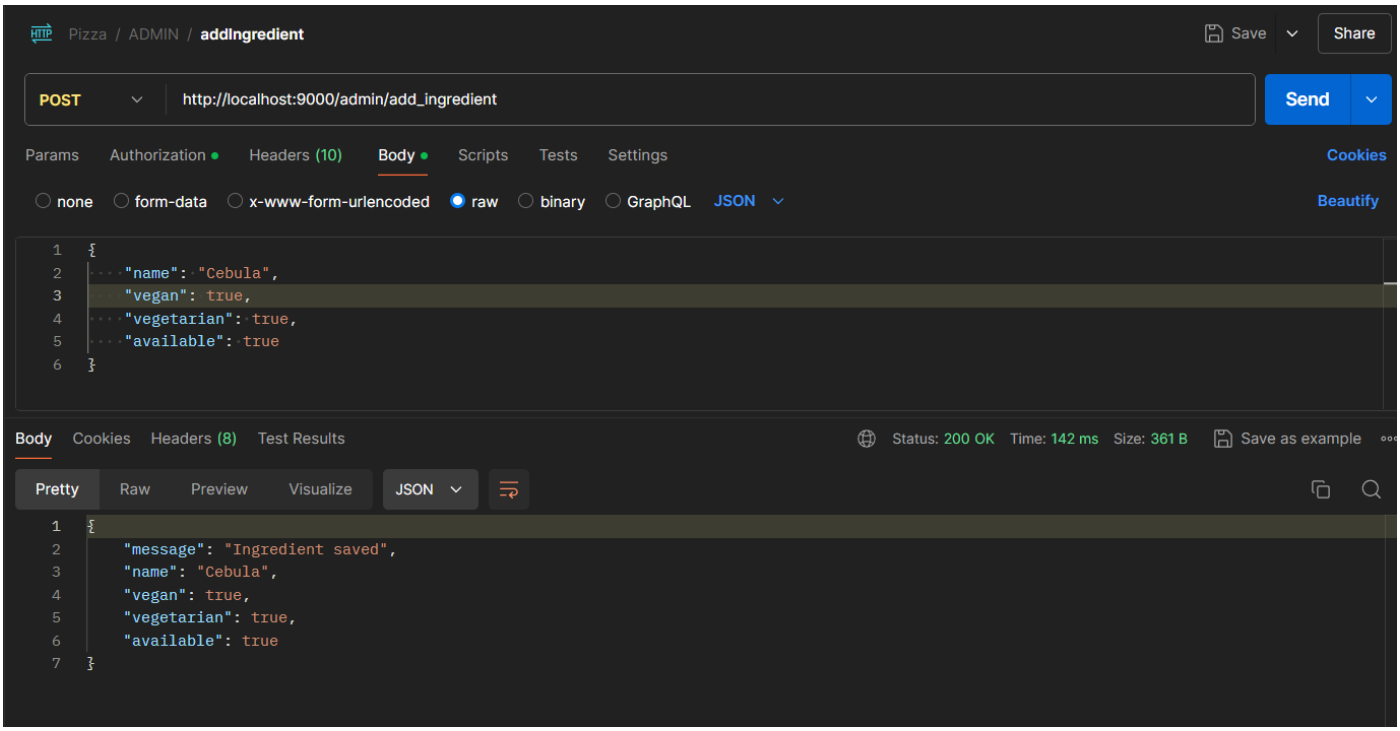


addIngredient

(dodanie nowego składnika pizzy do bazy)

```
const addIngredient = asyncHandler(async (req, res, next) => {
  try {
    const {name, vegan, vegetarian, available} = req.body;
    const existingIngredient = await Ingredient.findOne({name: name});
    if (existingIngredient) {
      res.status(400);
      throw new Error("Ingredient already exists");
    }
    await Ingredient.create({
      name,
      vegan,
      vegetarian,
      available
    });

    res.status(200).json({
      message: 'Ingredient saved',
      name,
      vegan,
```

```
      vegetarian,
      available
    })
  } catch(err) {
    next(err);
  }
});
```

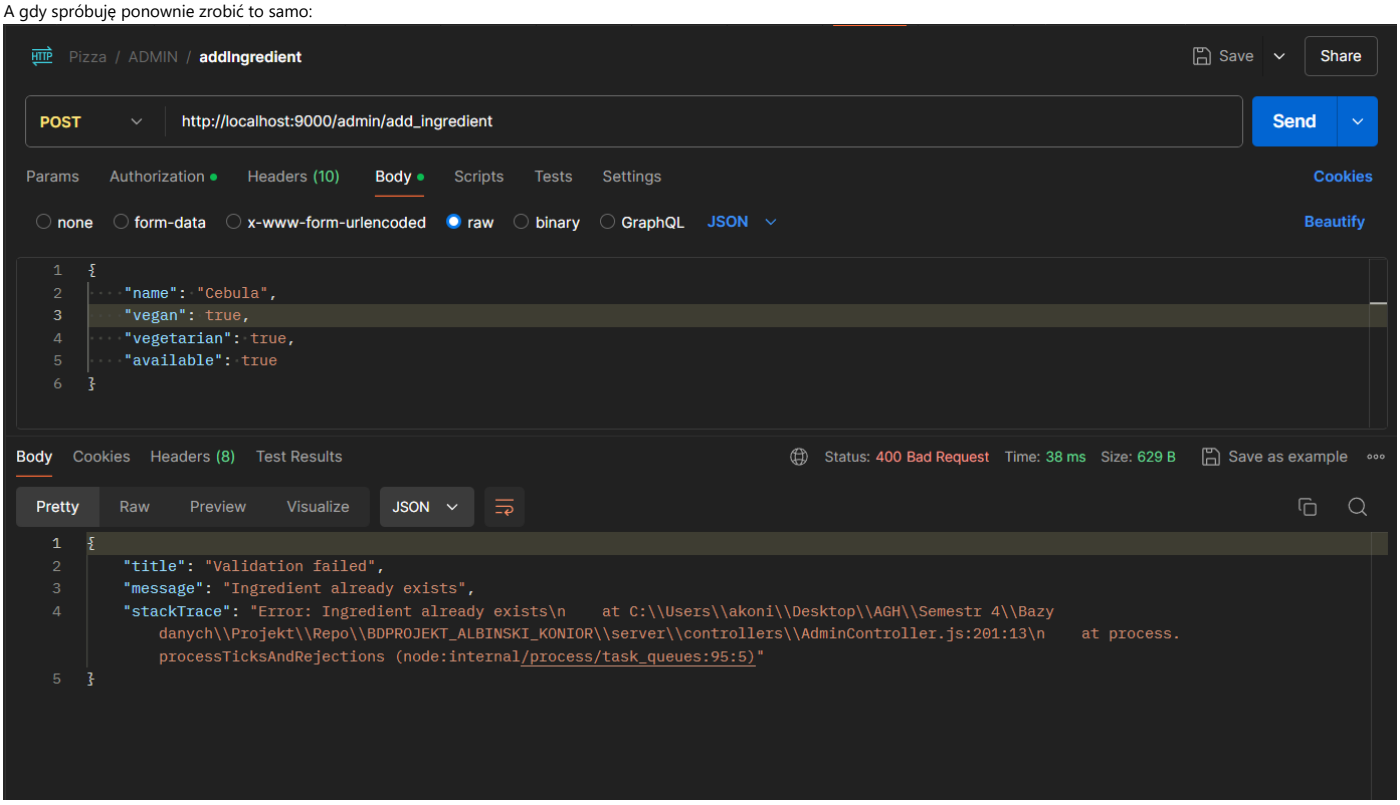



```
_id: ObjectId('665218c0e080e0db66b99c5b')
name : "Papryka"
vegan : true
vegetarian : true
available : false
__v : 0
```

```
_id: ObjectId('66533a9b3a36fde49aa46580')
name : "Mozzarella"
vegan : false
vegetarian : true
available : true
__v : 0
```

```
_id: ObjectId('665352008029580187e349a9')
name : "Cebula"
vegan : true
vegetarian : true
available : true
__v : 0
```

A gdy spróbuję ponownie zrobić to samo:

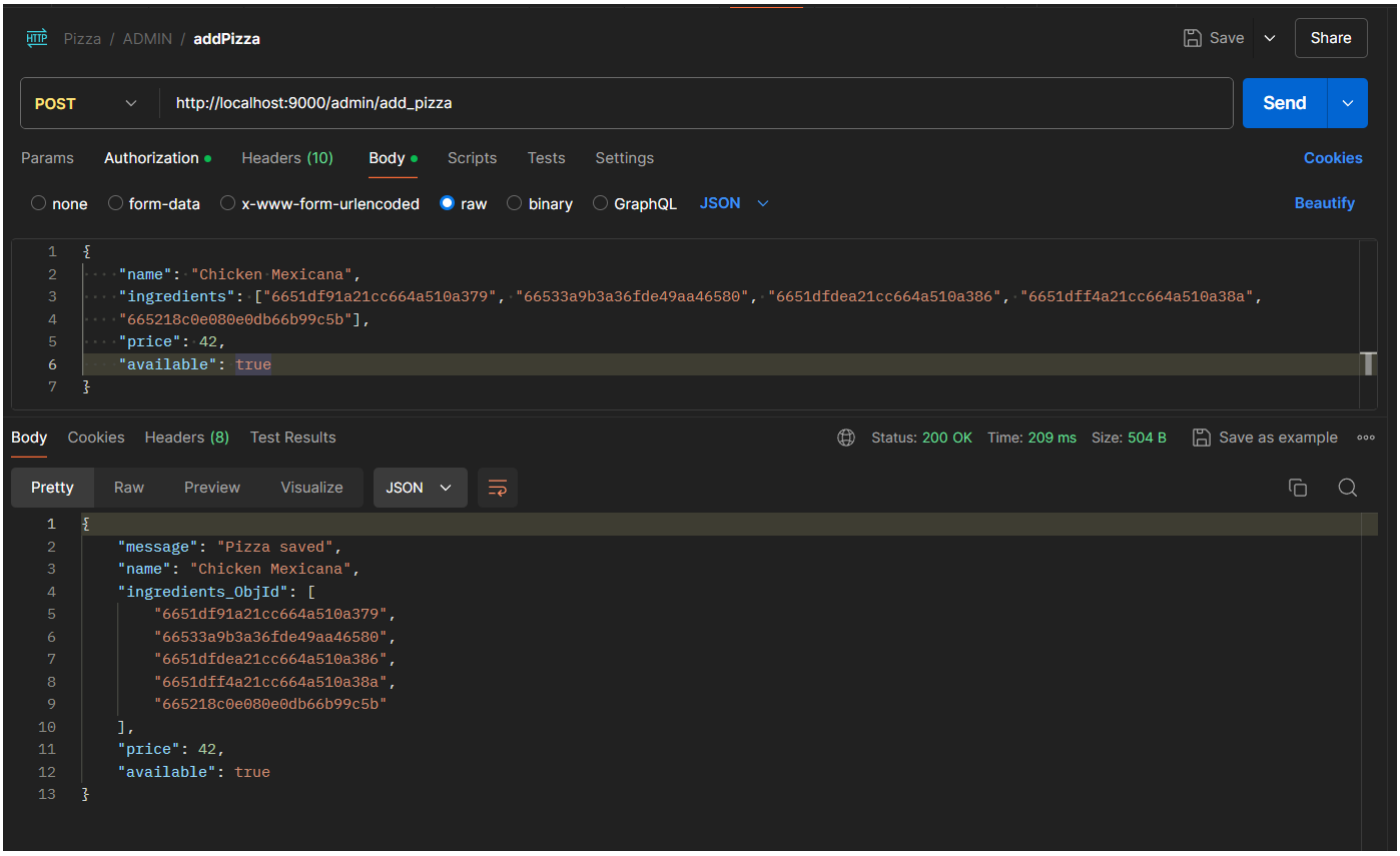


i cebula się nie dodała drugi raz.

addPizza (dodanie nowej pizzy do bazy)

```
const addPizza = asyncHandler(async (req, res, next) => {
  try {
    const {name, ingredients, price, available} = req.body;
    const existingPizzaWithName = await Pizza.findOne({name: name});
    if (existingPizzaWithName) {
      res.status(400);
      throw new Error("There is already a pizza with this name");
    }
    const ingredients_ObjId = ingredients.map((ingredient) => new ObjectId(ingredient));
    const existingPizzaWithIngredients = await Pizza.aggregate([
      {
        $project: {
          isSameIngredients: { $setEquals: ["$ingredients", ingredients_ObjId] }
        }
      },
      {
        $match: {
          isSameIngredients: true
        }
      }
    ]);
    if (existingPizzaWithIngredients.length > 0) {
      res.status(400);
      throw new Error("There is already a pizza with this set of ingredients");
    }
// sprawdzamy, czy wszystkie składniki, które zostały podane, istnieją w bazie w następujący sposób
    // 1) szukamy w bazie wszystkich składników, które zostały podane
    // 2) zliczamy ilość znalezionych w bazie składników
    // 3) sprawdzamy, czy znaleźliśmy tyle składników ile zostało podane(zakładamy, że nie mamy w bazie dwóch składników o tym samym ingredient_nr)
    const ingredientsExist = await Ingredient.aggregate([
      {
        $match: { _id: { $in: ingredients_ObjId } }
      },
      {
        $group: {
          _id: null,
          matchedIngredientsCount: { $sum: 1 }
        }
      },
      {
        $project: {
          ingredientsExist: { $eq: ["$matchedIngredientsCount", ingredients_ObjId.length] }
        }
      },
      {
        $match: {
          ingredientsExist: true
        }
```

```
      }
    ]);
    if (ingredientsExist.length === 0) {
      throw new Error("At least one of the given ingredients doesn't exist");
    }
    const next_menu_number_query = await Pizza.aggregate([
      {
        $sort: { menu_number: -1 }
      },
      {
        $limit: 1
      }
    ]);
    const next_menu_number = next_menu_number_query.length > 0 ? next_menu_number_query[0].menu_number + 1 : 1;
    await Pizza.create({
      name,
      menu_number: next_menu_number,
      ingredients: ingredients_ObjId,
      price,
      available
    });

    res.status(200).json({
      message: "Pizza saved",
      name,
      ingredients_ObjId,
      price,
      available
    })
  } catch(err) {
    next(err);
  }
});
```

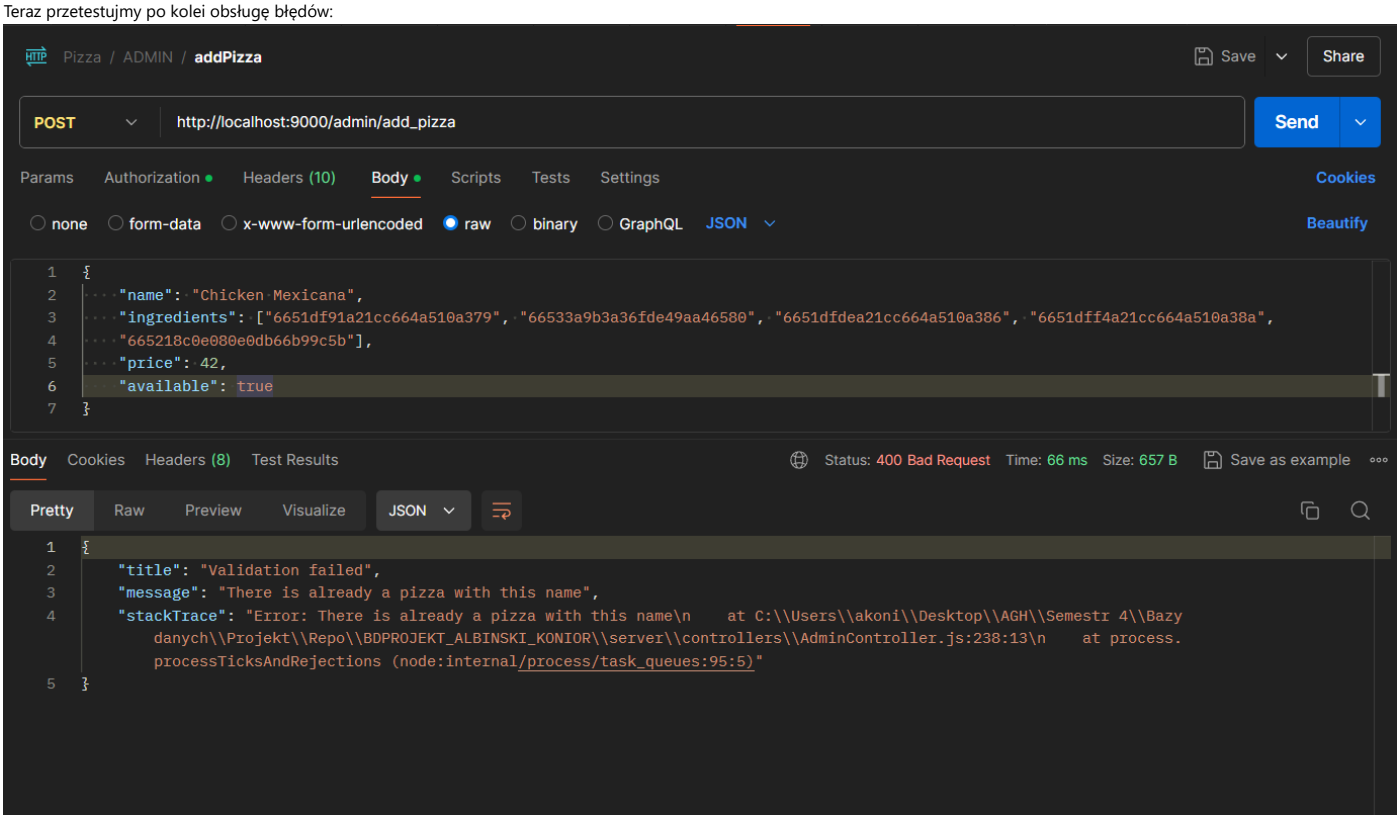


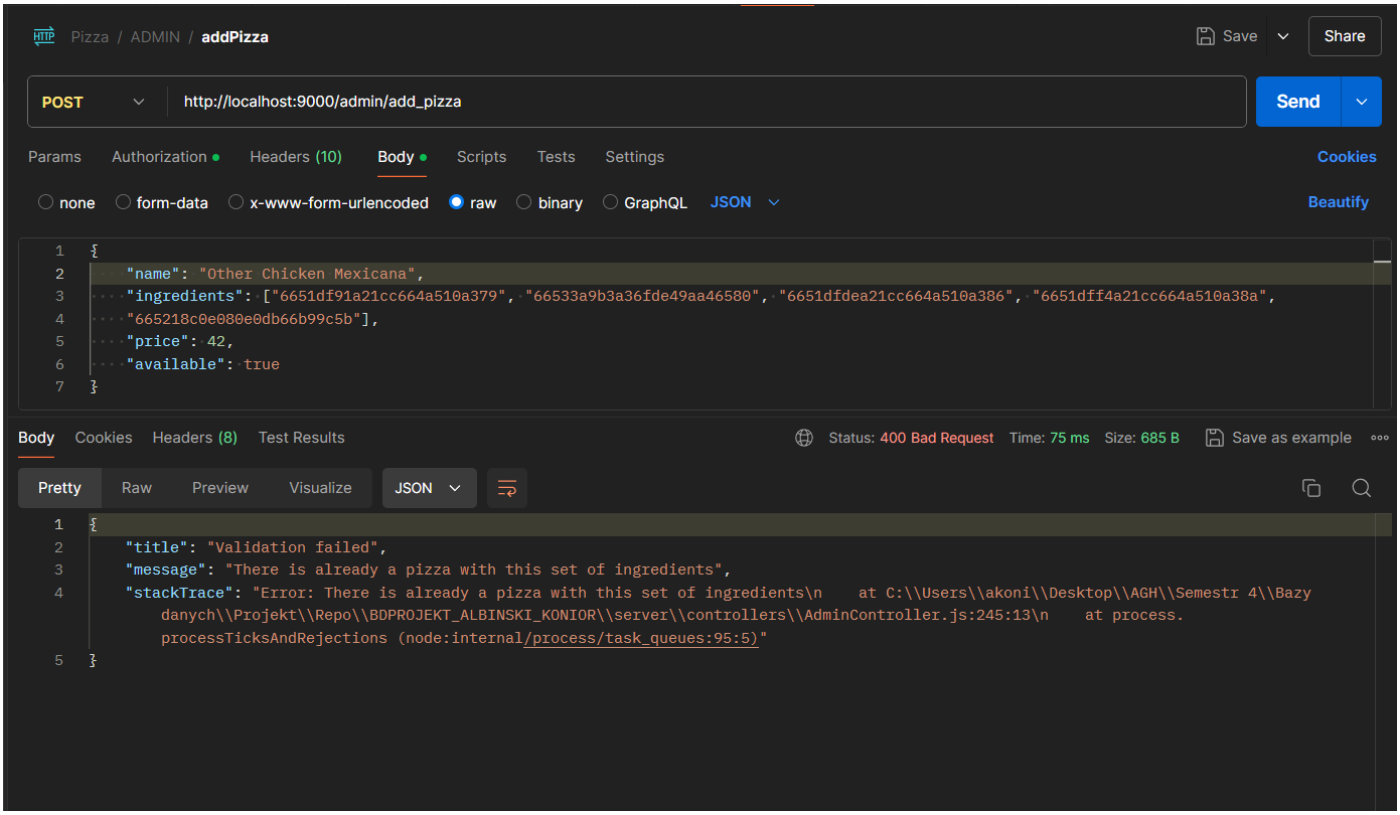

```
name : "Margherita"
menu_number : 1
ingredients : Array (2)
price : 33
available : true
grades : Object
__v : 0
```

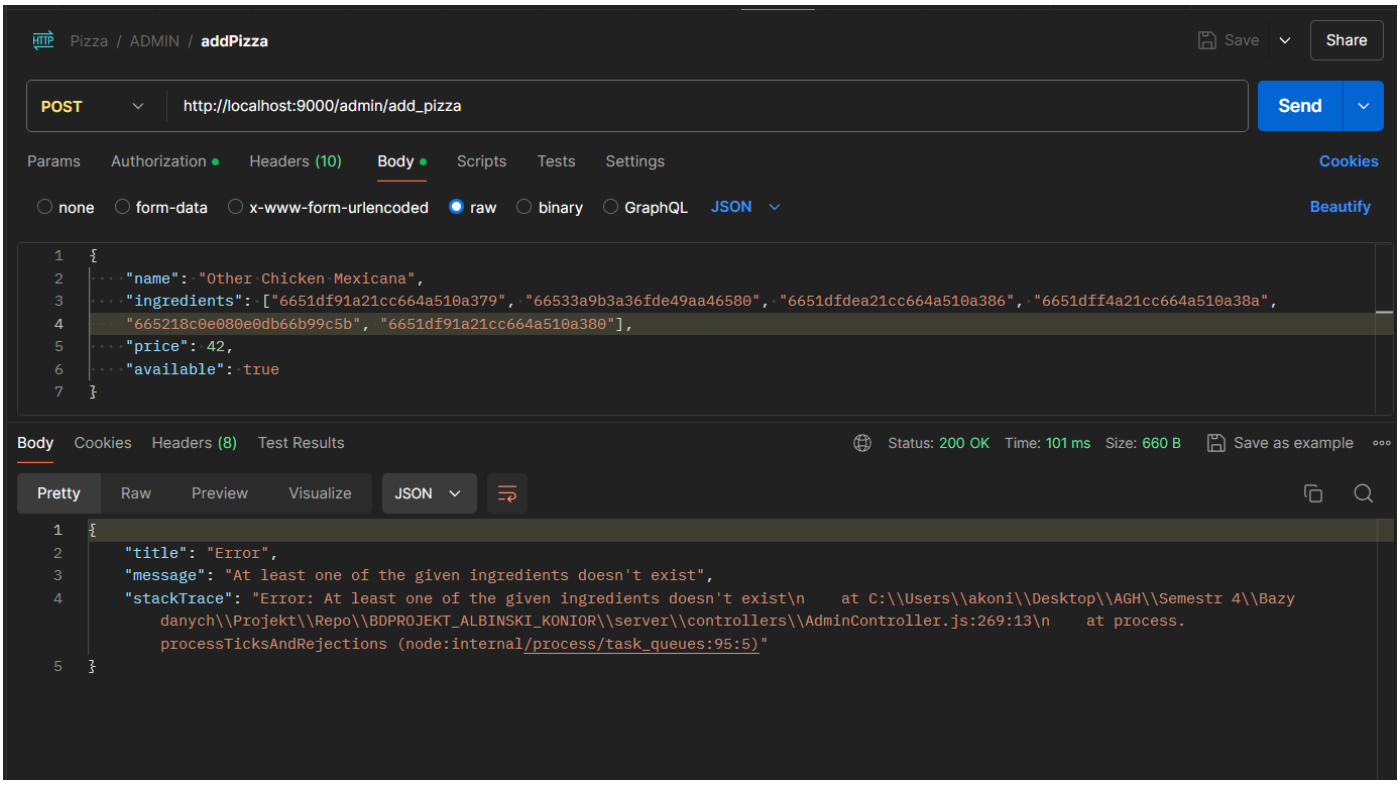
```
_id: ObjectId('665350568029580187e349a5')
name : "Americana"
menu_number : 2
ingredients : Array (3)
price : 38
available : true
grades : Object
__v : 0
```

```
_id: ObjectId('665353038029580187e349b0')
name : "Chicken Mexicana"
menu_number : 3
ingredients : Array (5)
price : 42
available : true
grades : Object
__v : 0
```

Teraz przetestujmy po kolei obsługę błędów:



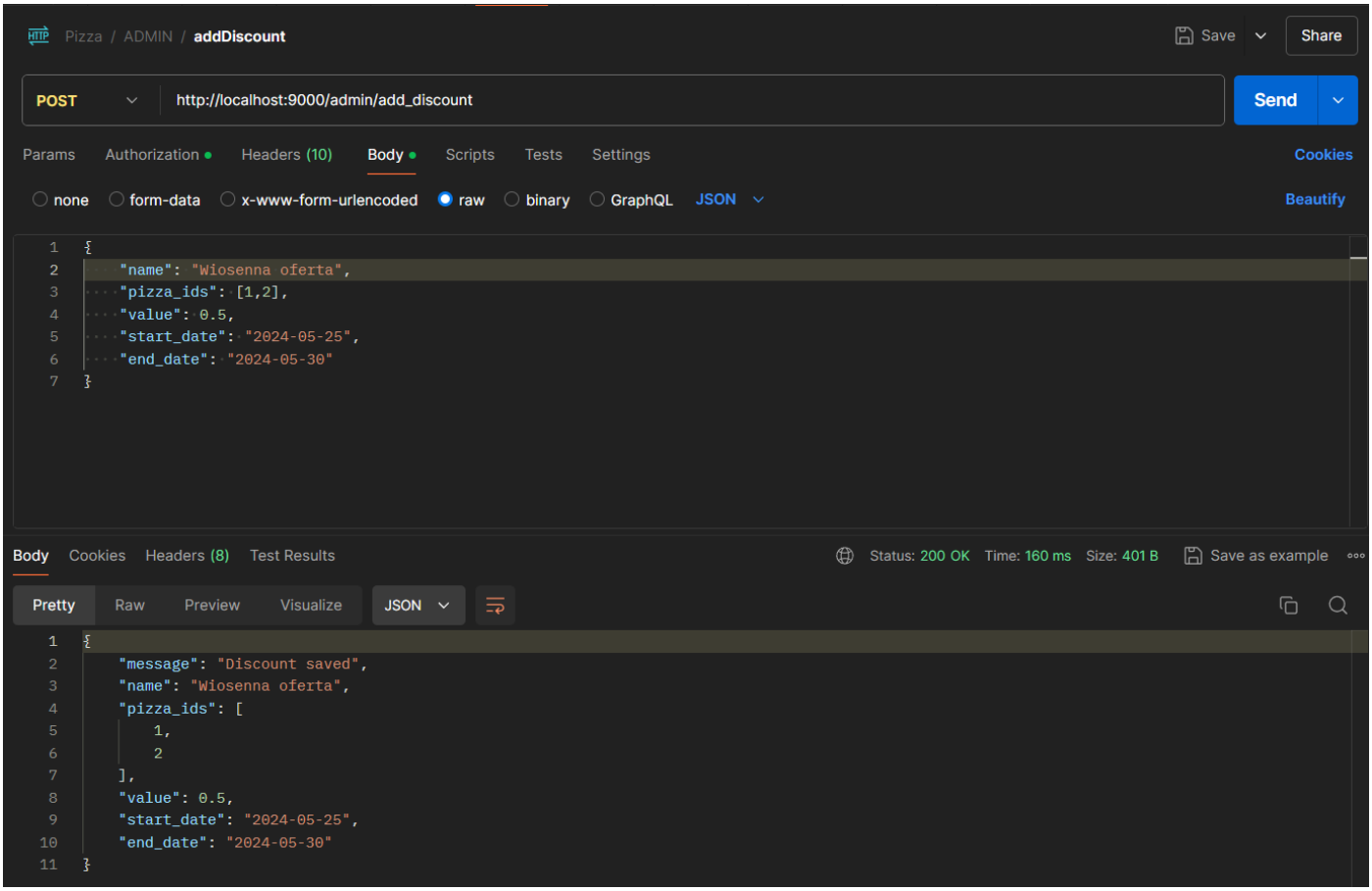




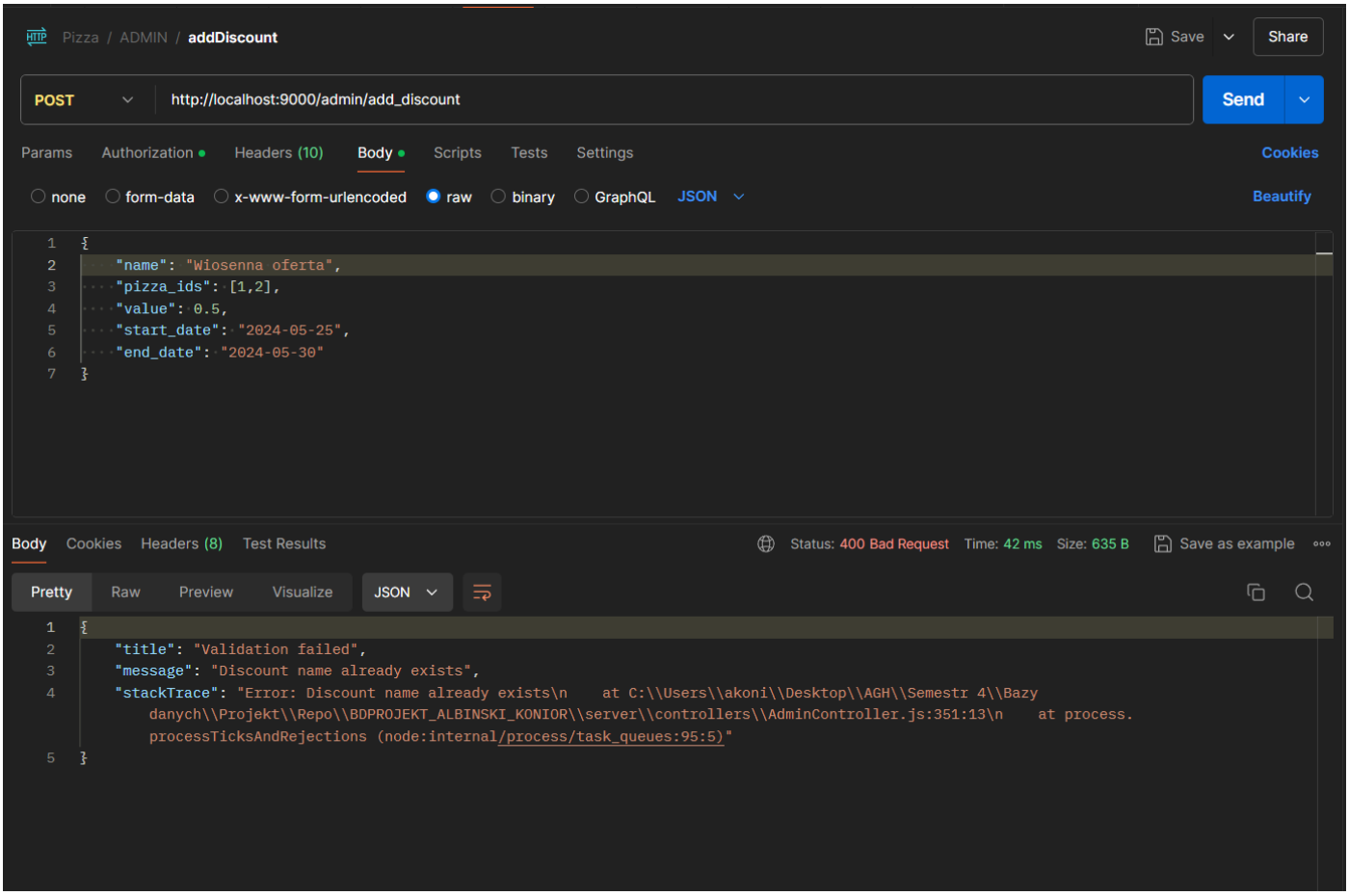
addDiscount (dodanie nowej zniżki do bazy)

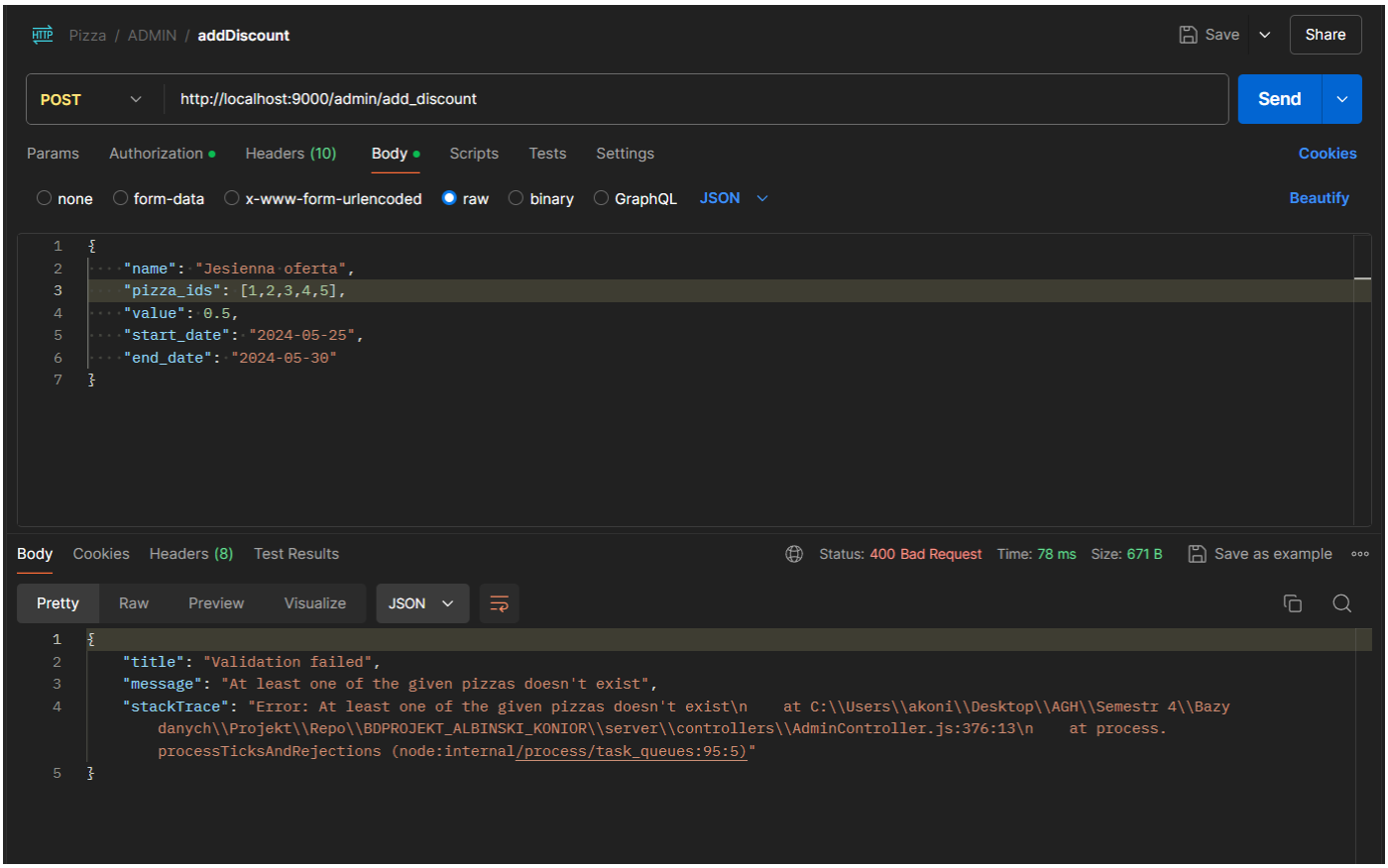
```
const addDiscount = asyncHandler(async (req, res, next) => {
  try {
    const {name, pizza_ids, value, start_date, end_date} = req.body;
    const pizza_ids_ObjId = pizza_ids.map(pizza_id => new ObjectId(pizza_id));
    const existingName = await Discount.findOne({name: name});
    if (existingName) {
      res.status(400);
      throw new Error("Discount name already exists");
    }
    const pizza_idsExist = await Pizza.aggregate([
      {
        $match: { _id: { $in: pizza_ids_ObjId } }
      },
      {
        $group: {
```

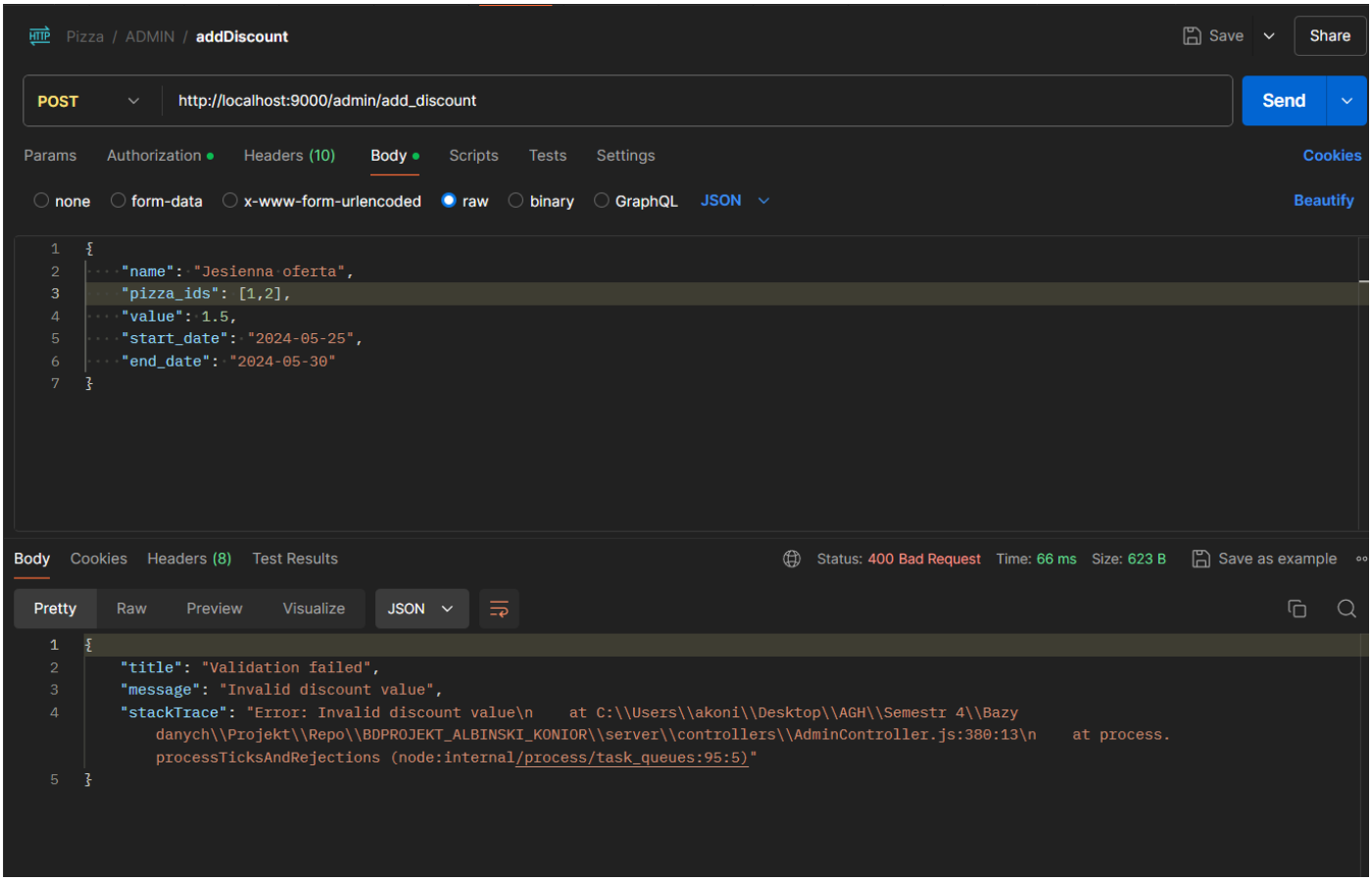
```
          _id: null,
          matchedPizzasCount: { $sum: 1 }
        }
      },
      {
        $project: {
          pizzasExist: { $eq: ["$matchedPizzasCount", pizza_ids_ObjId.length] }
        }
      },
      {
        $match: {
          pizzasExist: true
        }
      }
    ]);
    if (pizza_idsExist.length === 0) {
      res.status(400);
      throw new Error("At least one of the given pizzas doesn't exist");
    }
    if (value > 1 || value < 0) {
      res.status(400);
      throw new Error("Invalid discount value");
    }
    const start_date_DATE = new Date(start_date).toISOString();
    const end_date_DATE = new Date(end_date).toISOString();
    if (end_date_DATE < start_date_DATE) {
      res.status(400);
      throw new Error("Invalid dates");
    }
    await Discount.create({
      name,
      pizza_ids: pizza_ids_ObjId,
      value,
      start_date: start_date_DATE,
      end_date: end_date_DATE
    });
    res.status(200).json({
      message: "Discount saved",
      name,
      pizza_ids: pizza_ids_ObjId,
      value,
      start_date,
      end_date
    })
  } catch(err) {
    next(err);
  }
});
```

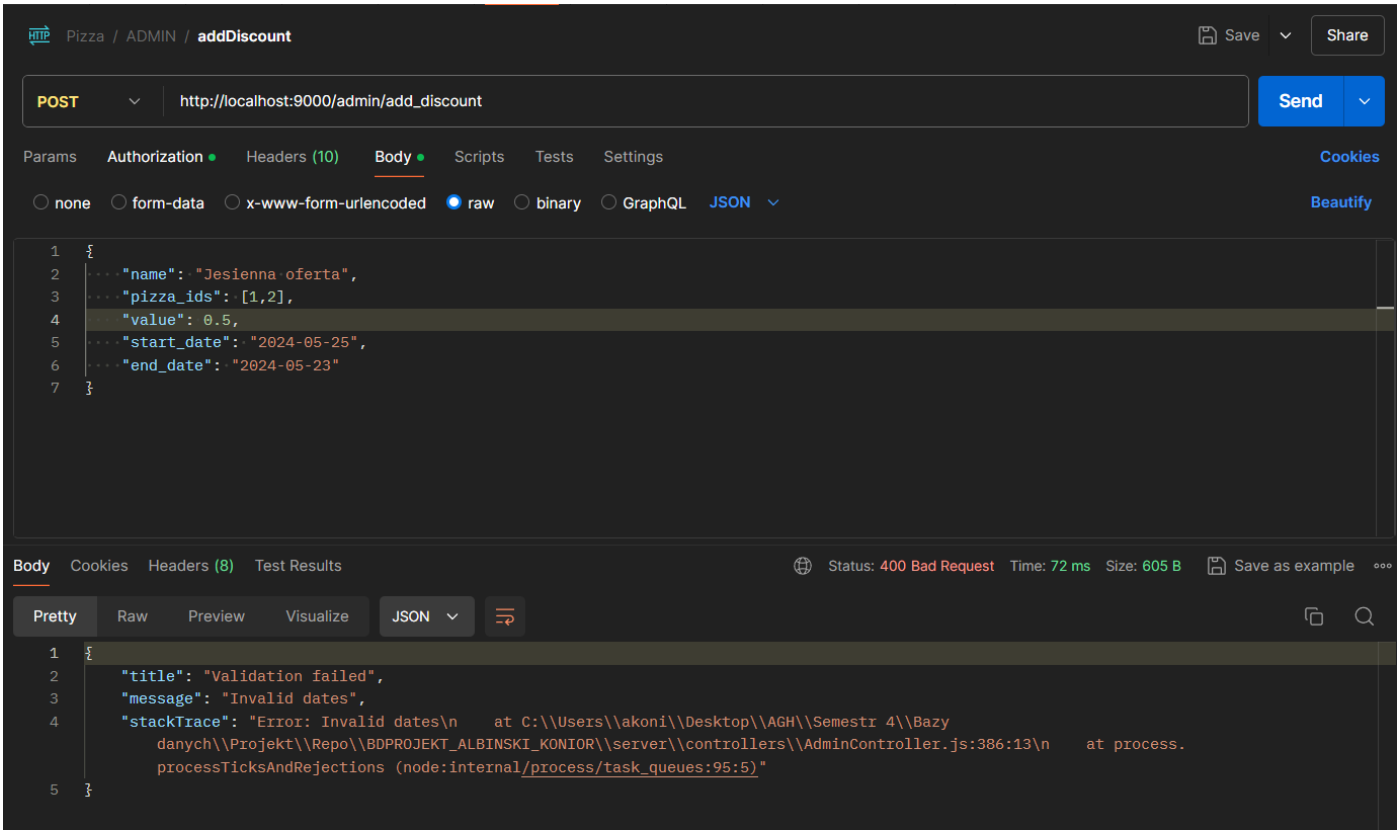


Teraz przetestujmy obłsugę błędów:


Pizza / ADMIN / addDiscount
Save
Share
POST
http://localhost:9000/admin/add_discount
Send
Params
Authorization
Headers (10)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"name": "Wiosenna oferta",
"pizza_ids": [1,2],
"value": 0.5,
"start_date": "2024-05-25",
"end_date": "2024-05-30"
}
Body
Cookies
Headers (8)
Test Results
Status: 400 Bad Request
Time: 42 ms
Size: 635 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"title": "Validation failed",
"message": "Discount name already exists",
"stackTrace": "Error: Discount name already exists\n at C:\\Users\\akoni\\Desktop\\AGH\\Semestr 4\\Bazy danych\\Projekt\\Repo\\BDPROJEKT_ALBINSKI_KONIOR\\server\\controllers\\AdminController.js:351:13\n at process.processTicksAndRejections (node:internal/process/task_queues:95:5)"
}


Pizza / ADMIN / addDiscount
Save
Share
POST
http://localhost:9000/admin/add_discount
Send
Params
Authorization
Headers (10)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"name": "Jesienna oferta",
"pizza_ids": [1,2,3,4,5],
"value": 0.5,
"start_date": "2024-05-25",
"end_date": "2024-05-30"
}
Body
Cookies
Headers (8)
Test Results
Status: 400 Bad Request
Time: 78 ms
Size: 671 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"title": "Validation failed",
"message": "At least one of the given pizzas doesn't exist",
"stackTrace": "Error: At least one of the given pizzas doesn't exist\n at C:\\Users\\akoni\\Desktop\\AGH\\Semestr 4\\Bazy danych\\Projekt\\Repo\\BDPROJEKT_ALBINSKI_KONIOR\\server\\controllers\\AdminController.js:376:13\n at process.processTicksAndRejections (node:internal/process/task_queues:95:5)"
}



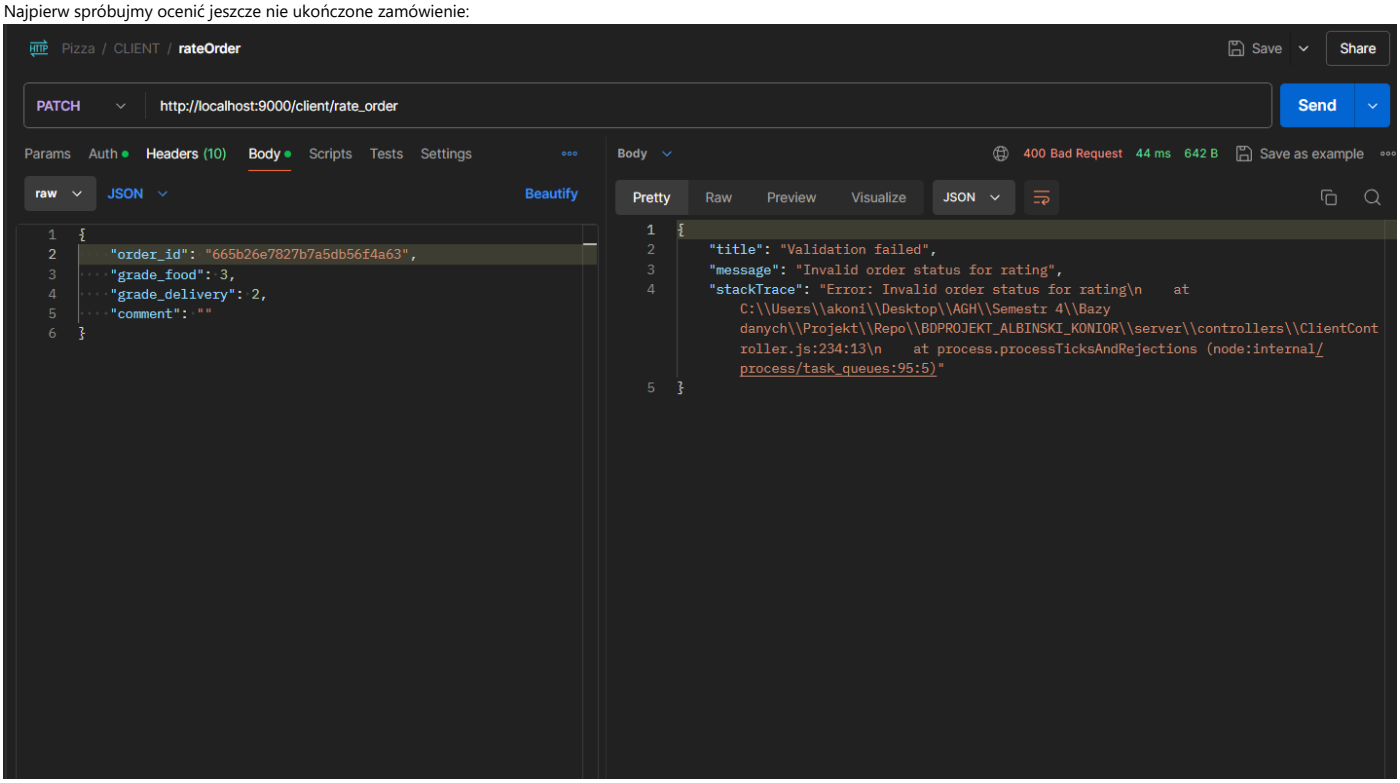


rateOrder

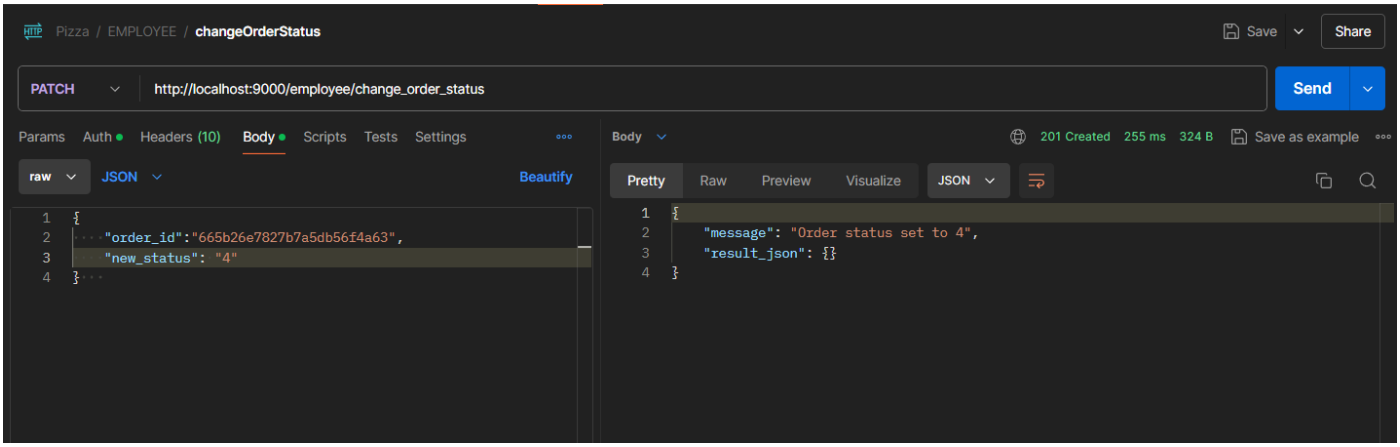
```
const rateOrder = asyncHandler(async (req, res, next) => {
  try {
    const {email, id, role} = req.user;
    let { order_id, grade_food, grade_delivery, comment } = req.body;
    if (!grade_food) {
      res.status(400);
      throw new Error("Please fill in all fields");
    }
    const order = await Order.findOne({_id: order_id});
    if (order.to_deliver && !grade_delivery) {
```

```
        res.status(400);
        throw new Error("Please fill in all fields");
      }
      if (!order) {
        res.status(400);
        throw new Error("Order doesn't exist");
      }
      if (!order.client_id.equals(new ObjectId(id))) {
        res.status(400);
        throw new Error("Order is not yours");
      }
      if (order.status !== '3.2' && order.status !== '4') {
        res.status(400);
        throw new Error("Invalid order status for rating");
      }
      if (!order.to_deliver) {
        grade_delivery = null;
      }
      const order_id_ObjId = new ObjectId(order_id);
      await Order.updateOne({ _id: order_id_ObjId }, {$set: {grade:
          {
            grade_food,
            grade_delivery,
            comment
          }}});
      res.status(200).json({
        message: "Order has been rated",
        grade_food,
        grade_delivery,
        comment
      });
    } catch (err) {
      next(err);
    }
  });
```

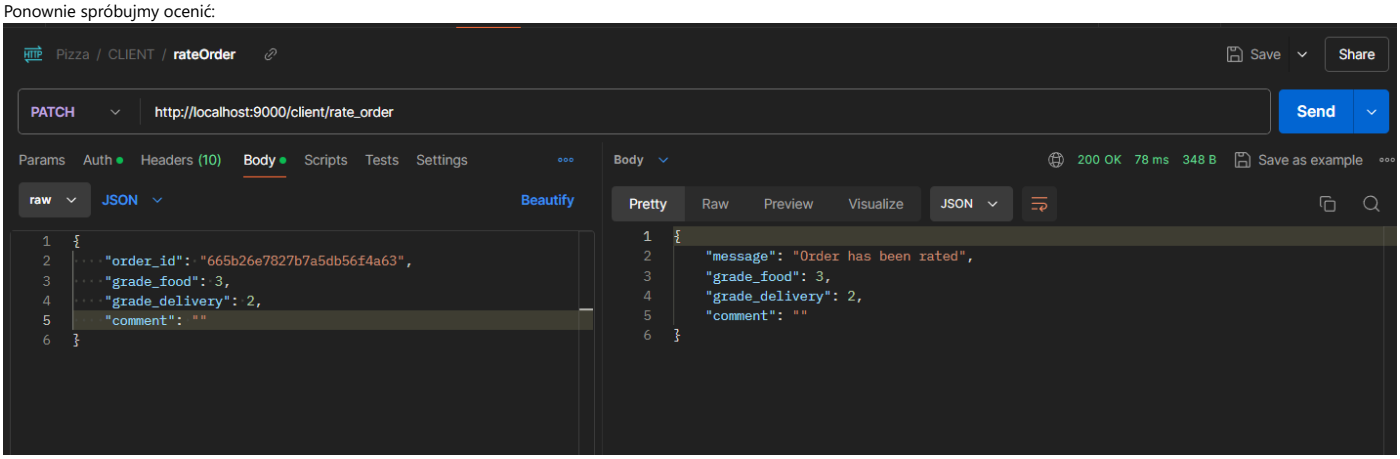
Najpierw spróbujmy ocenić jeszcze nie ukończone zamówienie:



Teraz zaktualizujmy status tego zamówienia:

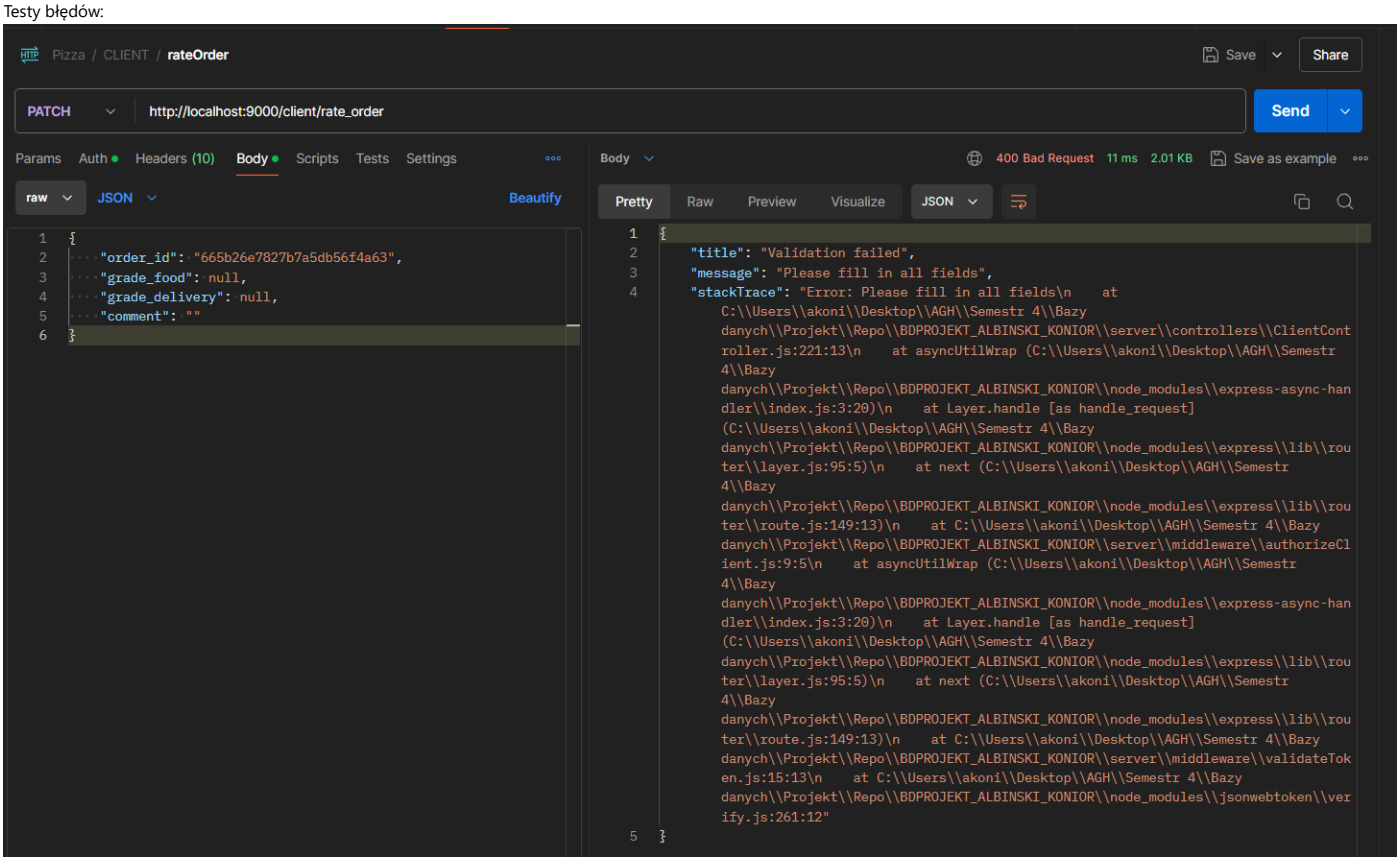


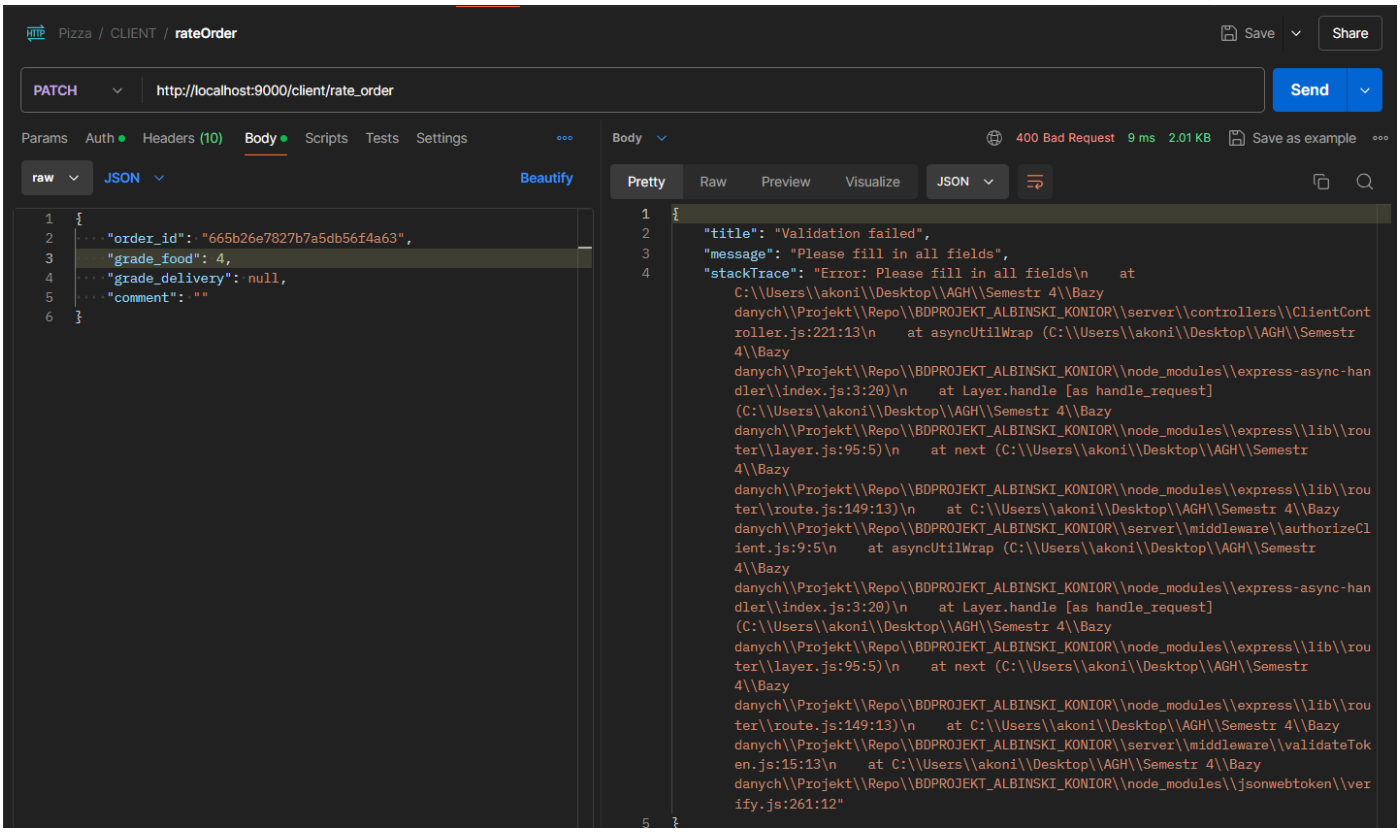
Ponownie spróbujmy ocenić:

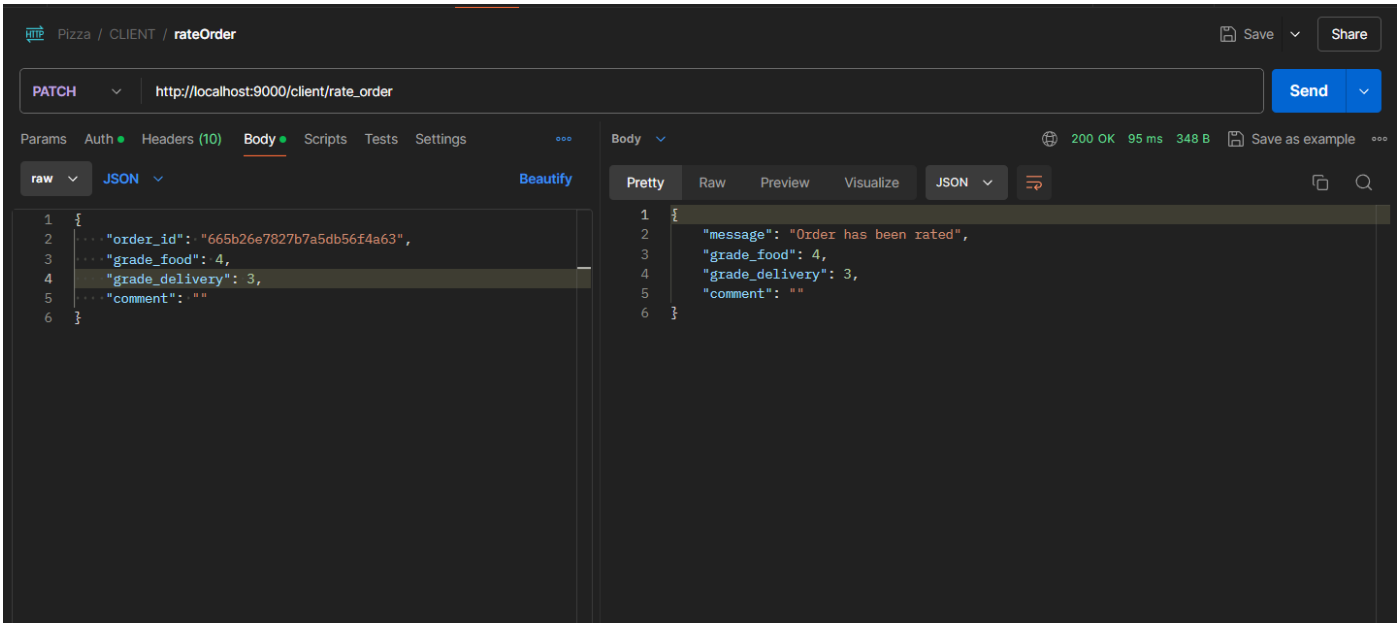



```
_id: ObjectId('665b26e7827b7a5db56f4a63')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('665b06e7b17d5f1918a48aea')
▸ pizzas : Array (2)
▸ client_address : Object
order_notes : ""
order_date : 2024-03-22T00:00:00.000+00:00
status : "4"
to_deliver : true
▸ total_price : Object
discount_id : null
createdAt : 2024-06-01T13:49:27.628+00:00
updatedAt : 2024-06-01T13:58:11.506+00:00
__v : 0
deliverer_id : ObjectId('665b070cb17d5f1918a48aef')
▾ grade : Object
    grade_food : 3
    grade_delivery : 2
    comment : ""
    _id : ObjectId('665b28f3827b7a5db56f4a92')
```

Testy błędów:



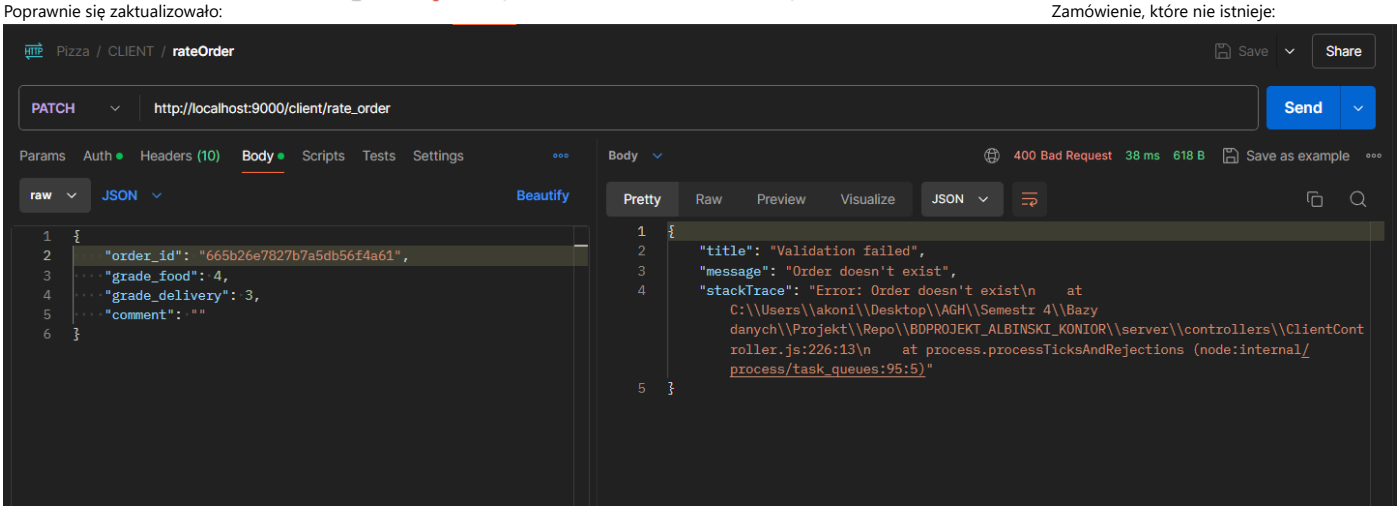




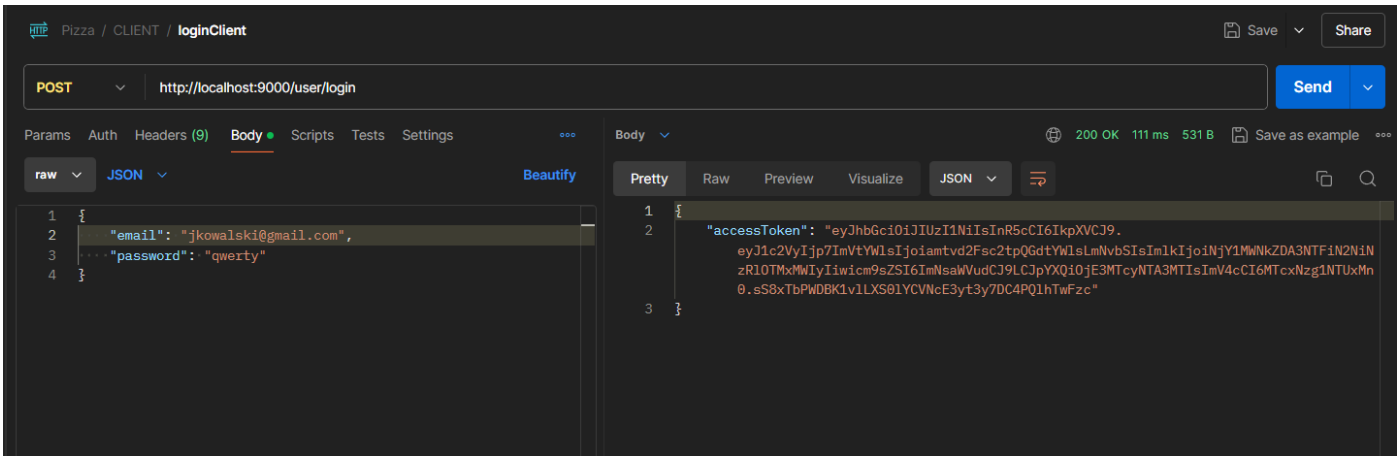

```
_id: ObjectId('665b26e7827b7a5db56f4a63')
client_id : ObjectId('6651d6cc5f48cfd8ded52387')
employee_id : ObjectId('665b06e7b17d5f1918a48aea')
▸ pizzas : Array (2)
▸ client_address : Object
order_notes : ""
order_date : 2024-03-22T00:00:00.000+00:00
status : "4"
to_deliver : true
▸ total_price : Object
discount_id : null
createdAt : 2024-06-01T13:49:27.628+00:00
updatedAt : 2024-06-01T14:02:11.407+00:00
__v : 0
deliverer_id : ObjectId('665b070cb17d5f1918a48aef')
▾ grade : Object
    grade_food : 4
    grade_delivery : 3
    comment : ""
    _id : ObjectId('665b29e3827b7a5db56f4a97')
```

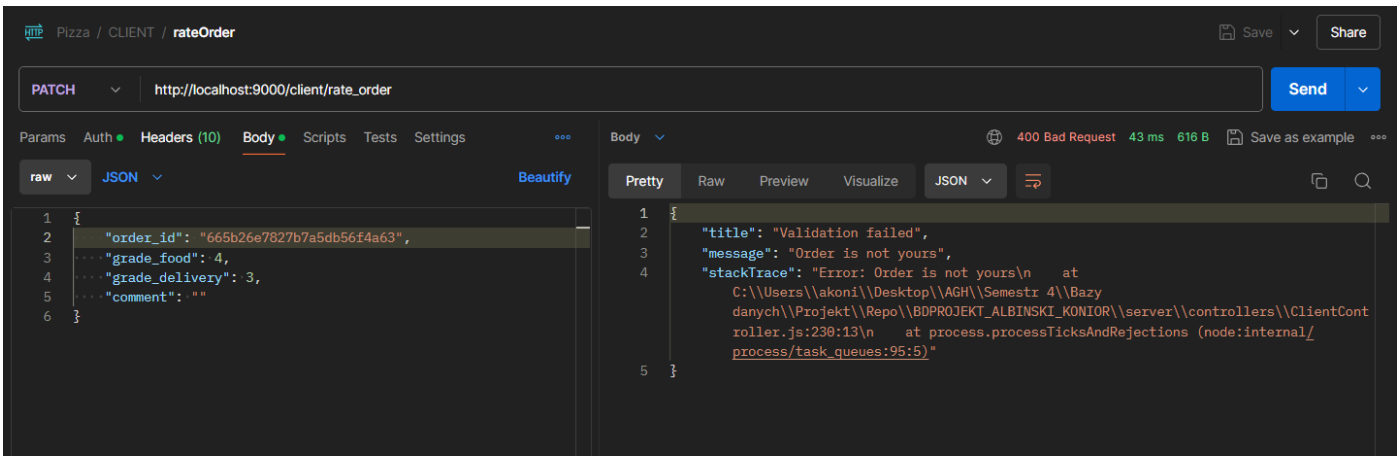
Poprawnie się zaktualizowało:

Zamówienie, które nie istnieje:



I spróbujmy się jeszcze zalogować jako inny użytkownik:


Pizza / CLIENT / loginClient
Save
Share
POST
http://localhost:9000/user/login
Send
Params Auth Headers (9) Body Scripts Tests Settings
raw JSON Beautify
{
"email": "jkowalski@gmail.com",
"password": "qwerty"
}
Body
200 OK 111 ms 531 B
Save as example
Pretty Raw Preview Visualize JSON
{
"accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ1c2VyIjp7ImVtYWlsIjoiamtvd2Fsc2tpQGdtYWlsLmNvbSIsImlkIjoiNjY1MWNkZDA3NTFiN2NiN
zRlOTMxMWIyIiwicm9sZSI6ImNsaWVudCJ9LCJpYXQiOjE3MTcyNTA3MTIsImV4cCI6MTcxNzg1NTUxMn
0.sS8xTbPWDBK1vlLXS0lYCVNcE3yt3y7DC4PQlhTwFzc"
}


Pizza / CLIENT / rateOrder
Save
Share
PATCH
http://localhost:9000/client/rate_order
Send
Params Auth Headers (10) Body Scripts Tests Settings
raw JSON Beautify
{
"order_id": "665b26e7827b7a5db56f4a63",
"grade_food": 4,
"grade_delivery": 3,
"comment": ""
}
Body
400 Bad Request 43 ms 616 B
Save as example
Pretty Raw Preview Visualize JSON
{
"title": "Validation failed",
"message": "Order is not yours",
"stackTrace": "Error: Order is not yours\n at
C:\\Users\\akoni\\Desktop\\AGH\\Semestr 4\\Bazy
danych\\Projekt\\Repo\\BDPROJEKT_ALBINSKI_KONIOR\\server\\controllers\\ClientCont
roller.js:230:13\n at process.processTicksAndRejections (node:internal/
process/task_queues:95:5)"
}