Uniwersytet Rzeszowski
Kolegium Nauk Przyrodniczych

Ekosystem dla obsługi zakupów on-line

Proseminarium

Prowadzący: **mgr inż. Marcin Ochab**

Członkowie zespołu (grupa laboratoryjna 3):

- Marcin Wielgos
- Karol Lech
- Przemysław Pleśniarski
- Krzysztof Podkulski

Rok akademicki 2020/2021, semestr zimowy

Spis treści

Opis projektu

Celem projektu jest stworzenie systemu sieci magazynów ze sklepami. Będzie możliwe utworzenie dowolnej ilości instancji magazynów. Użytkownik będzie miał możliwość wyboru produktów z listy (wszystko ze wszystkich magazynów). Podczas zamawiania administrator wybiera, z którego magazynu zostanie złożone zamówienie.

Funkcjonalności

- budowanie dowolnej ilości instancji magazynów
- przechowywanie informacji o ilościach stanów magazynowych
- możliwość dodawania do koszyka produktów pochodzących z różnych magazynów
- sumowanie kosztów za produkty
- autoryzacja użytkownika

Struktura i uruchomienie projektu

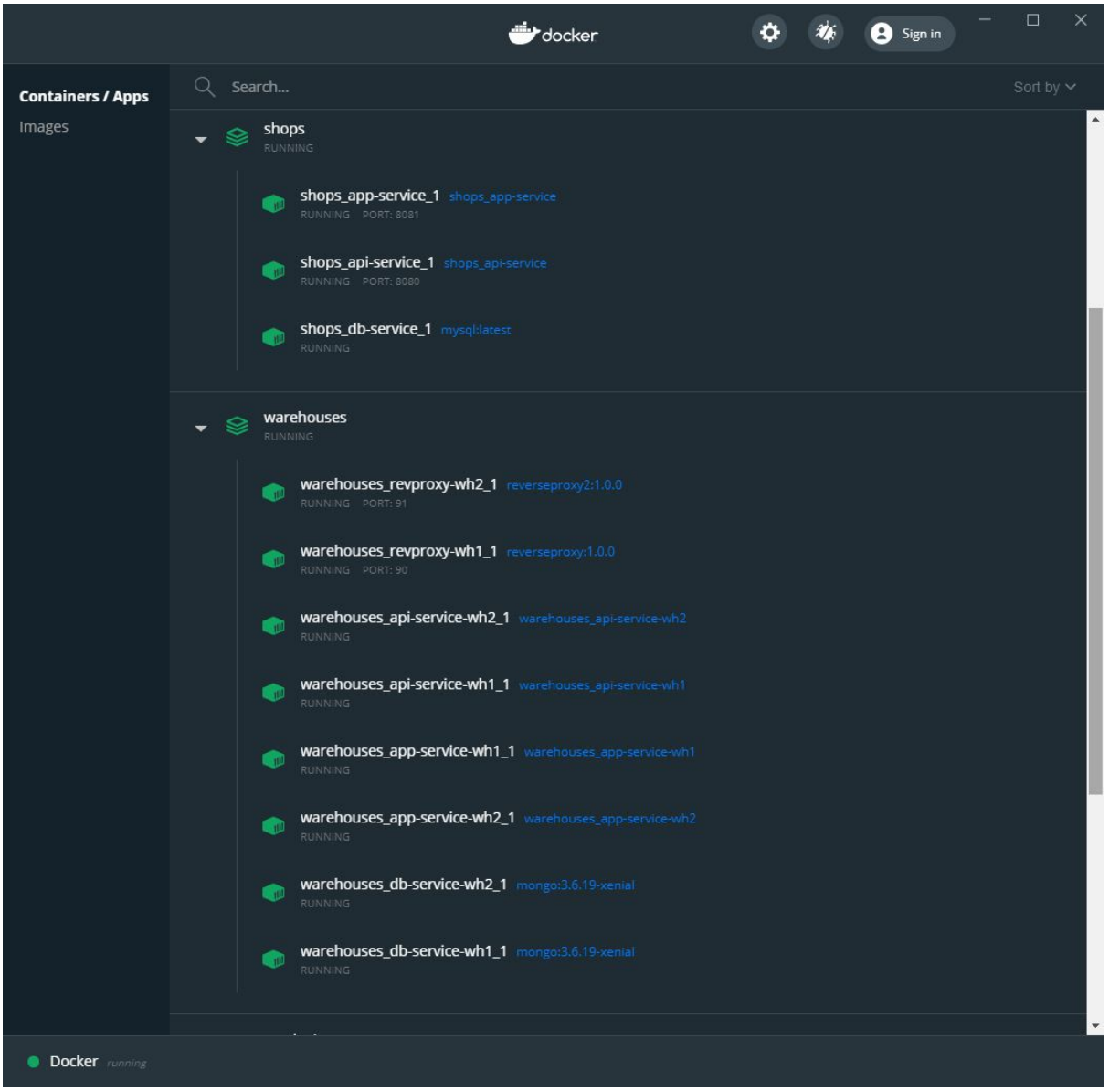
Stabilna wersja aplikacji znajduje się w branchu **main**.
Aby uruchomić każdy mikroserwis przechodzimy do każdego folderu i budujemy kontenery Dockera poprzez *docker-compose up --build*

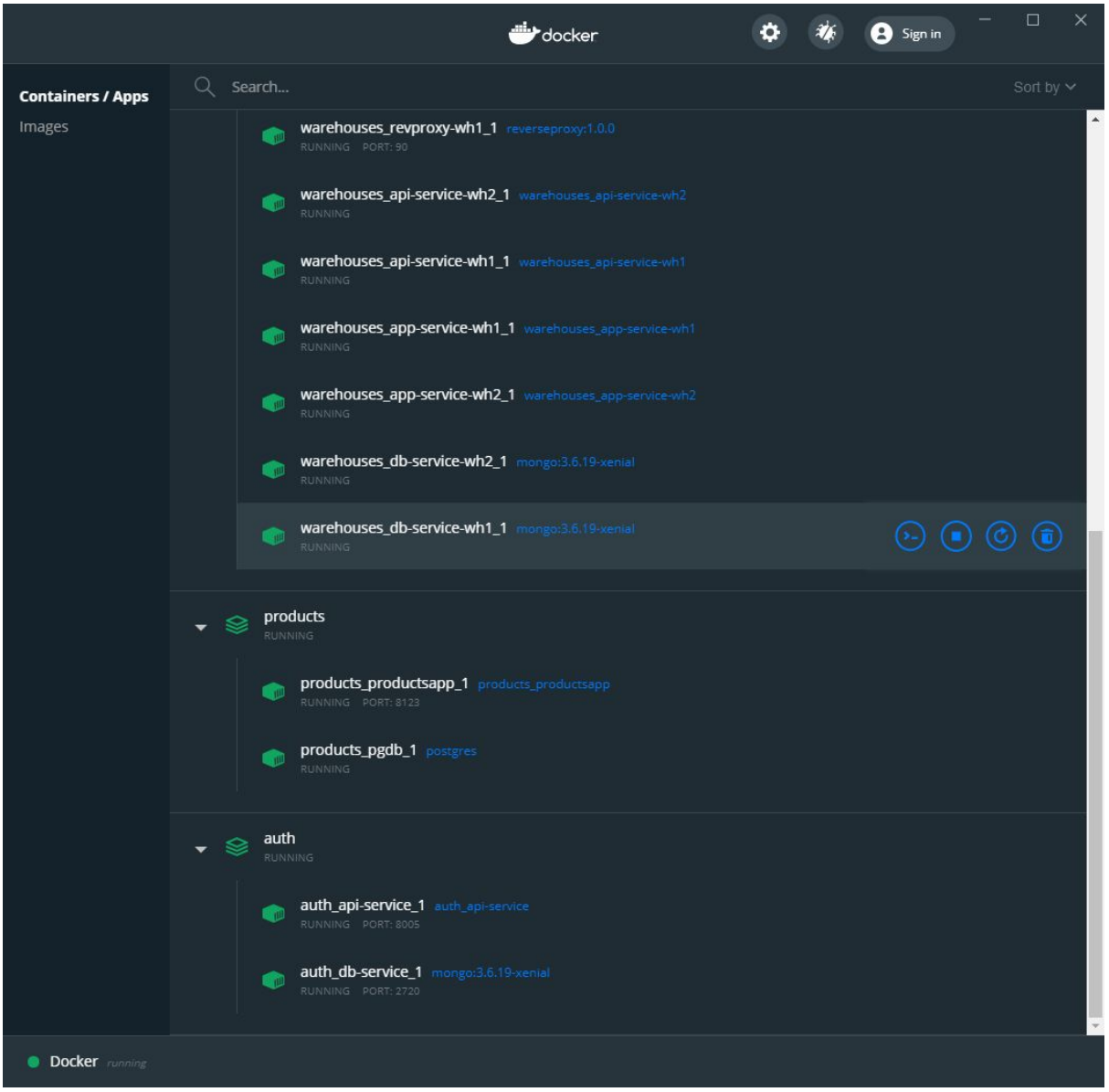
Pierwszy magazyn dostępny jest pod adresem http://localhost:90/
API magazynu http://localhost:90/wh1/api

Dla drugiego magazynu analogicznie
http://localhost:91/
http://localhost:91/wh2/api

Mikroserwis produktów pod adresem http://localhost:90/products

Mikroserwis sklepów http://localhost:8081 z API na porcie 8080


docker
Sign in
Containers / Apps
Images
Search...
Sort by
shops
RUNNING
shops_app-service_1 shops_app-service
RUNNING PORT: 8081
shops_api-service_1 shops_api-service
RUNNING PORT: 8080
shops_db-service_1 mysql:latest
RUNNING
warehouses
RUNNING
warehouses_revproxy-wh2_1 reverseproxy2:1.0.0
RUNNING PORT: 91
warehouses_revproxy-wh1_1 reverseproxy:1.0.0
RUNNING PORT: 90
warehouses_api-service-wh2_1 warehouses_api-service-wh2
RUNNING
warehouses_api-service-wh1_1 warehouses_api-service-wh1
RUNNING
warehouses_app-service-wh1_1 warehouses_app-service-wh1
RUNNING
warehouses_app-service-wh2_1 warehouses_app-service-wh2
RUNNING
warehouses_db-service-wh2_1 mongo:3.6.19-xenial
RUNNING
warehouses_db-service-wh1_1 mongo:3.6.19-xenial
RUNNING
Docker running

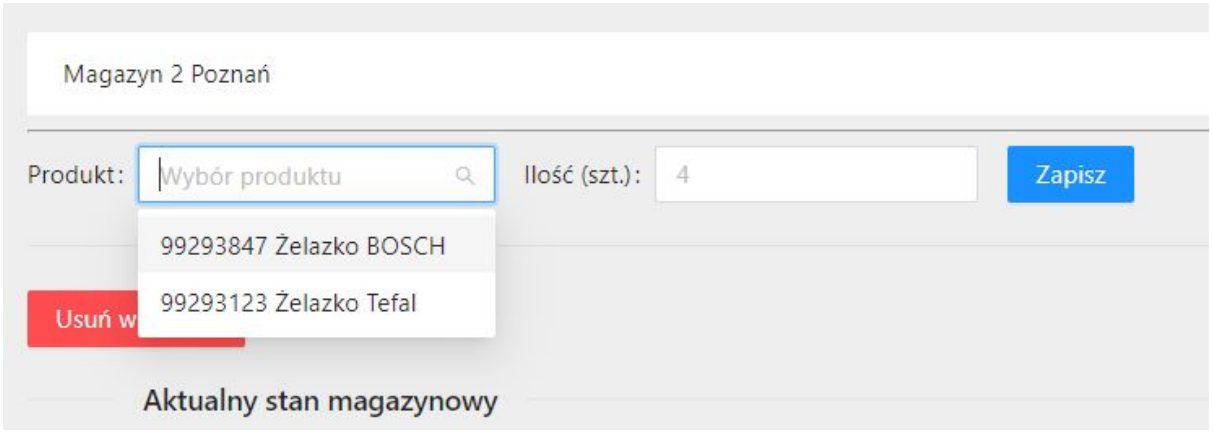


Mikroserwis magazynów

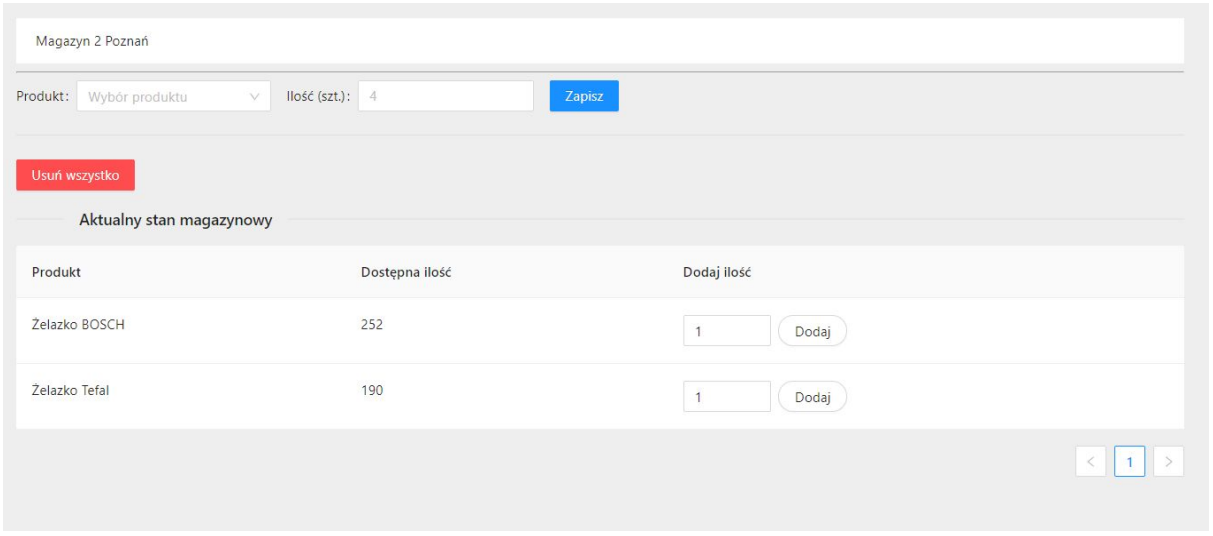
Warstwa frontend: ReactJS
Warstwa backend: ExpressJS
Baza danych: MongoDB
Obsługa ruchu sieciowego: nginx z *reverse-proxy*

Warstwa frontend

Po dodaniu przykładowych produktów w mikroserwisie produktów, jesteśmy w stanie odczytać te informacje w mikroserwisie magazynów i przypisać odpowiedni status dostępności (ilość na magazynie).



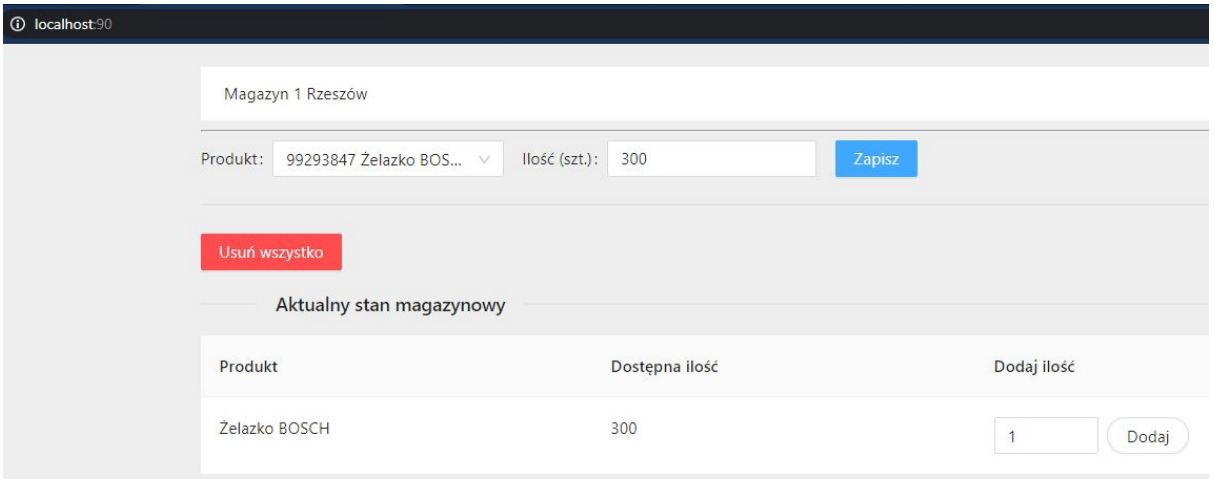
GUI w mikroserwisie magazynów pozwala na zarządzanie stanem magazynowym.

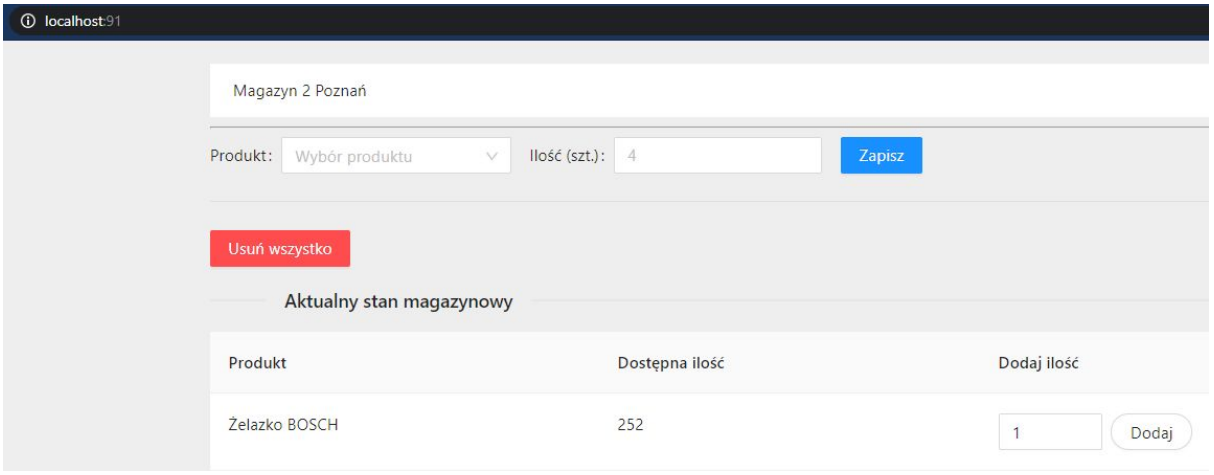


System przewiduje możliwość utworzenia wielu odrębnych instancji magazynów z odrębnym statusem zamówień.

W ramach testu utworzono dwie instancje (dwa magazyny).

- Magazyn 1 Rzeszów
- Magazyn 2 Poznań

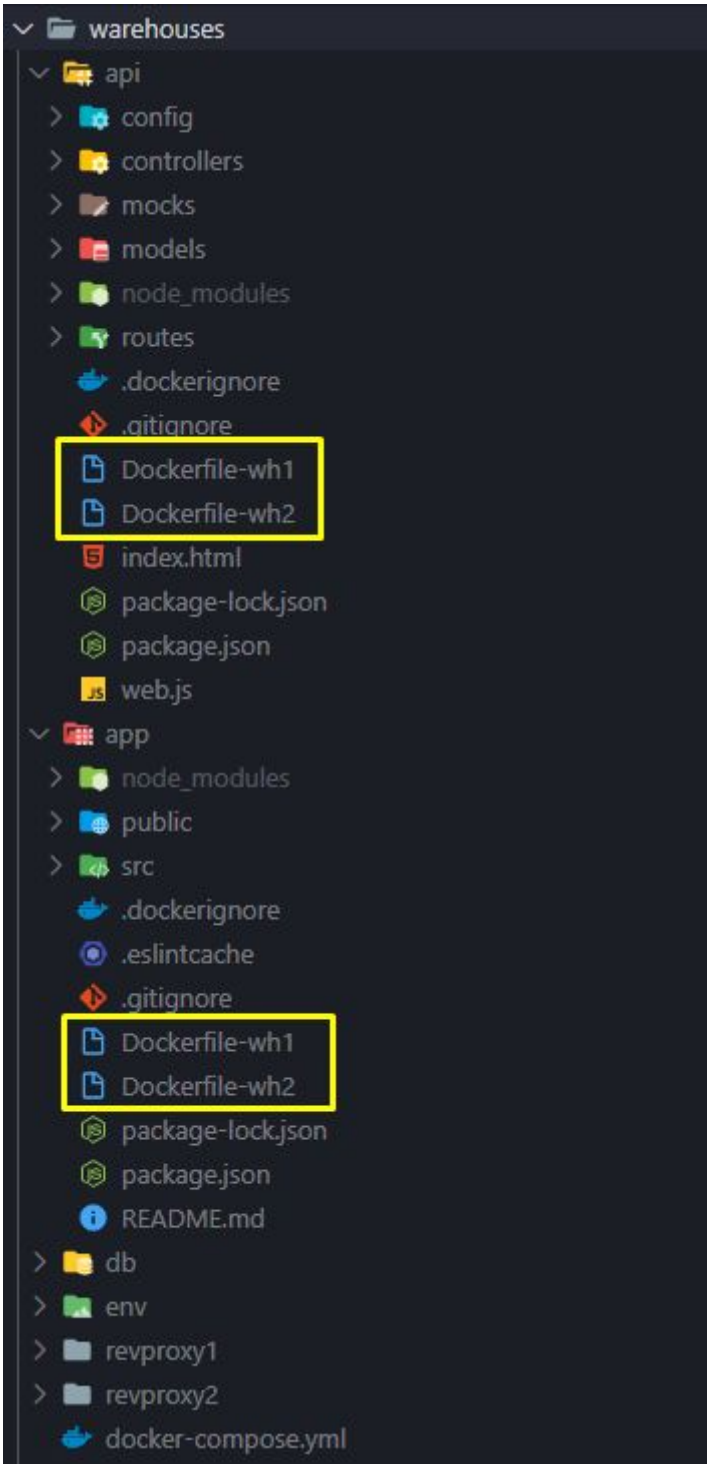




W magazynie możemy przechowywać dowolne produkty, których informacje są przechowywane w mikroserwisie produktów, jednak ich ilości magazynowe są odrębne dla każdej instancji magazynu.

Uruchomienie nowej instancji magazynu

Uruchomienie nowej instancji wiąże się z wykonaniem kilku kroków:



1. Ustawienie odpowiednich plików Dockerfile dla /api i /app pamiętając o zmiennych środowiskowych dla portu i nazwy:

```
ENV REACT_APP_PORT=90
ENV REACT_APP_WHKEY=wh1
ENV REACT_APP_WHNAME="Magazyn 1 Rzeszów"
```

2. Ustawienie odpowiedniej konfiguracji kontenera nginx dla ruchu sieciowego z odpowiednim portem (revproxy).

```
server {
    listen 90;

    location /wh1/api/ {
      proxy_pass         http://apiservice:8005/;
      proxy_redirect     off;
      proxy_set_header   Host $host;
      proxy_set_header   X-Real-IP $remote_addr;
                  proxy_set_header         X-Forwarded-For
$proxy_add_x_forwarded_for;
      proxy_set_header   X-Forwarded-Host $server_name;
    }
```

Oraz w pliku Dockerfile dla nginx

```
# dockerfile
FROM nginx:1.16.0-alpine
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 90
CMD ["nginx", "-g", "daemon off;"]
```

Zapytania

Sklep internetowy może pobrać aktualną dostępność produktów przez zapytanie GET `http://localhost:91/wh2/api/stock/`

Przykładowy wynik:


```
{
    "stock": [
        {
            "_id": "60256e85435d12e212732365",
            "productId": "1",
            "__v": 0,
            "availableQuantity": 300,
            "createdAt": "2021-02-11T17:51:01.887Z",
            "updatedAt": "2021-02-11T17:51:01.887Z"
        },
        {
            "_id": "60256e8b435d12e21273236b",
            "productId": "2",
            "__v": 0,
```

```
            "availableQuantity": 200,
            "createdAt": "2021-02-11T17:51:07.150Z",
            "updatedAt": "2021-02-11T17:51:07.150Z"
        }
    ]
}
```

Sklep może także złożyć dyspozycję zmniejszenia stanu magazynowego w przypadku zakupu poprzez zapytanie POST `http://localhost:91/wh2/api/stock/make` z przykładowym body:

```
{
    "products": [
        {
            "id": 1,
            "quantity": 50
        },
        {
            "id": 2,
            "quantity": 10
        }
    ]
}
```

Wysłanie zapytania spowoduje zmniejszenie ilości stanu magazynowego produktu o ID 1 o 50 sztuk, a produktu z ID 2 o 10 sztuk.

Mikroserwis produktów

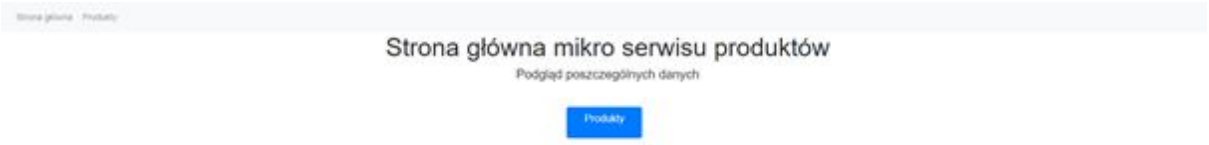
Opis mikroserwisu produktów

Aplikacja została napisana za pomocą:
FrontEnd - engine view = pug
BackEnd - Node.js, express.js
Baza Danych: PostgreSQL

Aplikacja przedstawia mikro serwis produktów. Został napisany w node.js, z wykorzystaniem express.js oraz silnika PUG, który umożliwia tworzenie stron internetowych bez ingerencji zewnętrznych frameworków takich jak REACT bądź ANGULAR.

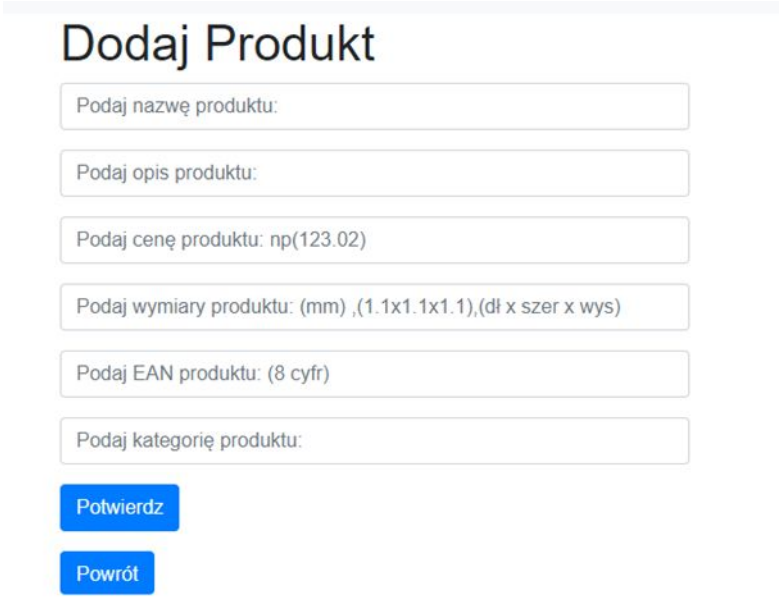
Aplikacji mikro serwisu produktów

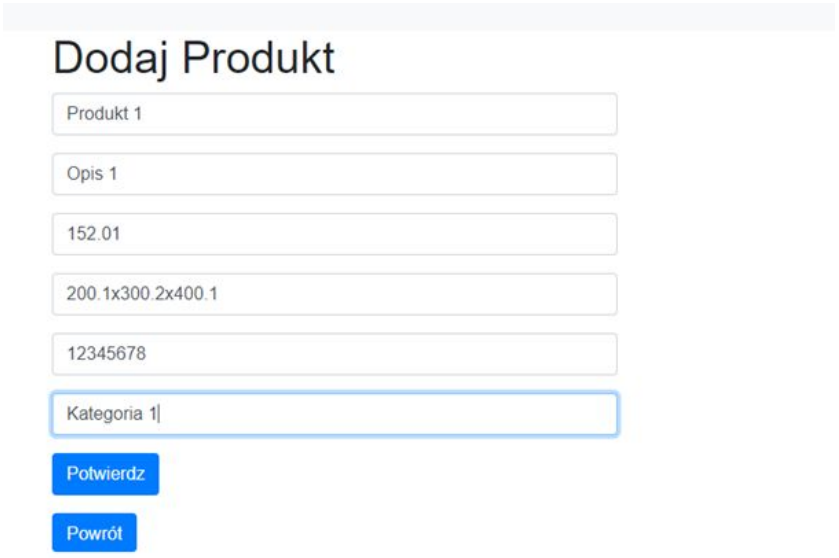
Na głównej stronie znajduje się informacja co to za serwis oraz odnośnik do listy produktów.



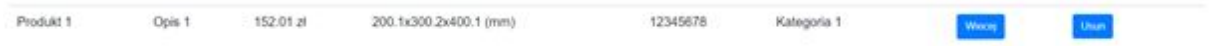
Lista produktów przedstawia wszystkie najważniejsze informacje dotyczące produktów, również jest możliwość stworzenia nowego produktu za pomocą przycisku dodaj.

Dzięki odpowiedniemu formularzu jest możliwe dodanie produktu. Jest aktywna walidacja, gdzie produkt nie zostanie wprowadzony do listy, jeżeli nie zostały spełnione wszystkie warunki walidacji jak odpowiednio zapisana cena np. (123.00) bądź rozmiar. Wszystkie pola muszą zostać wypełnione.

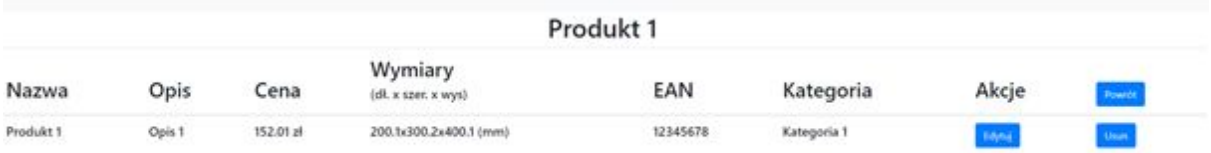


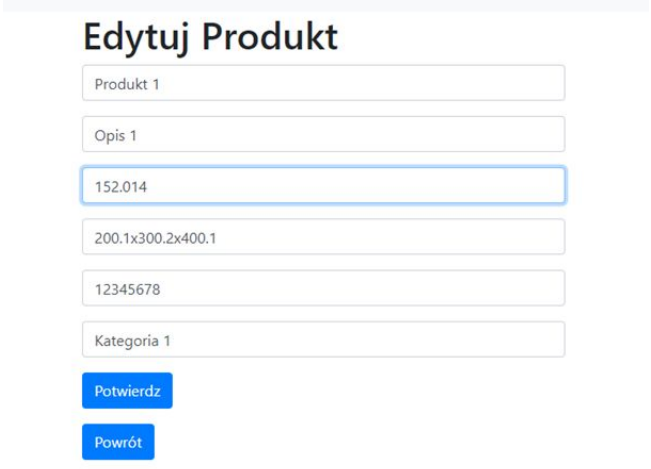


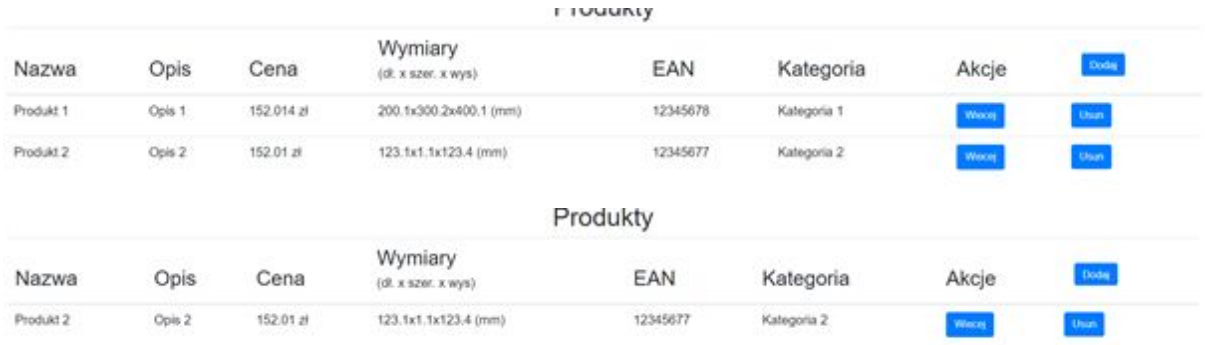
Po zatwierdzeniu, produkt zostanie wypisany na liście produktów. Jest możliwe usunięcie produktu za pomocą przycisku usuń, który działa w technologii AJAX, bez potrzeby przeładowania strony produkt zostanie usunięty. Przycisk więcej przenosi nas do podstrony o produkcie.



Dokładny opis produktu oraz możliwość edytowania produktu oraz usunięcie produktu. Przy zmianie danych produktów również występuje walidacja, która musi zostać spełniona.



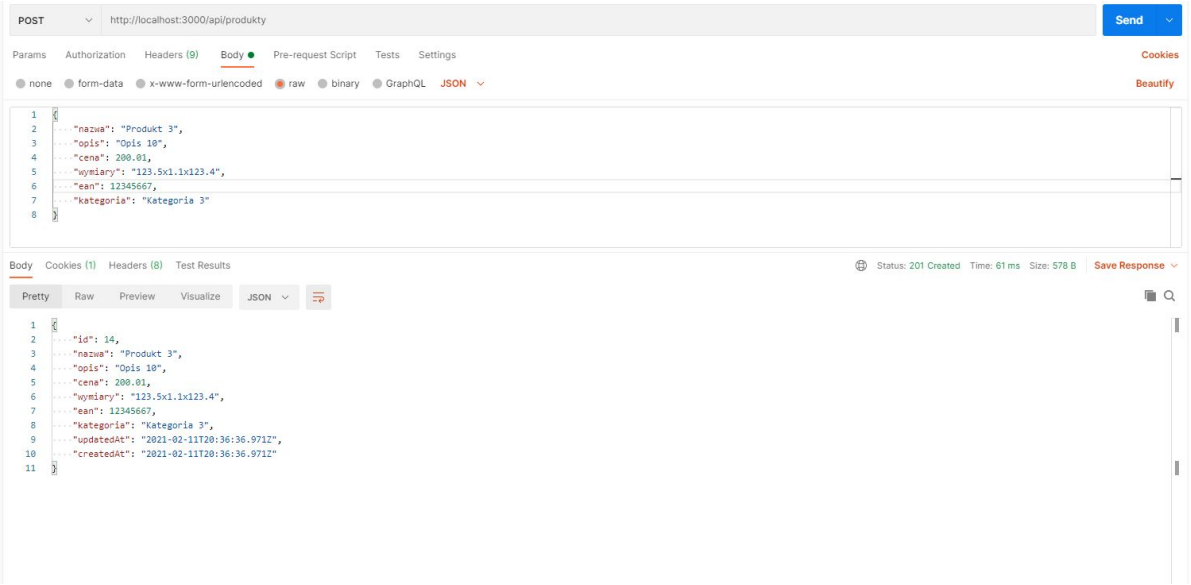




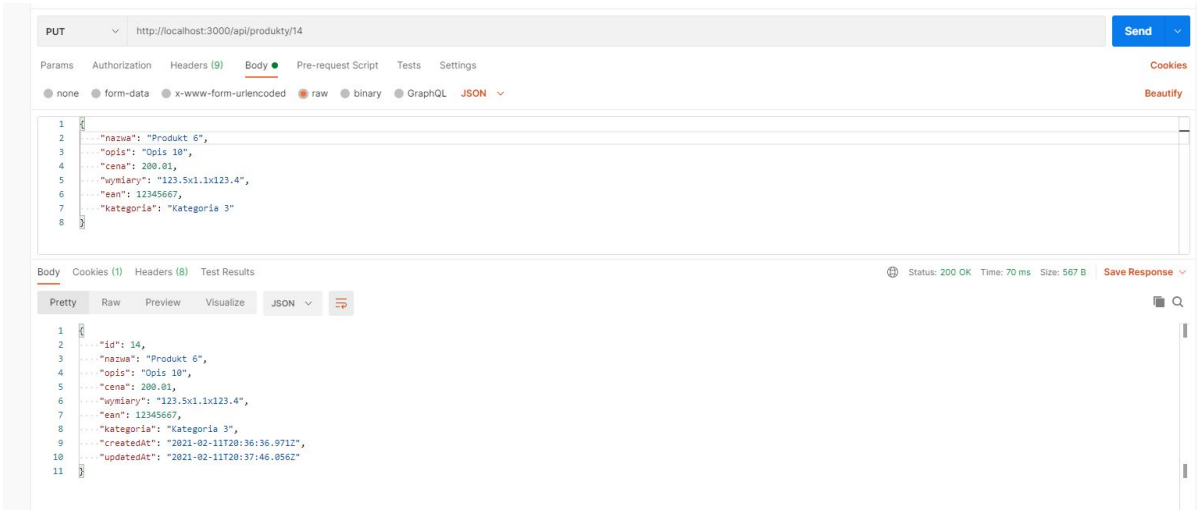
API mikro serwisu produktów

Do komunikacji między mikro serwisami jest wymagane api, które zostało przygotowane w pełnej konfiguracji.

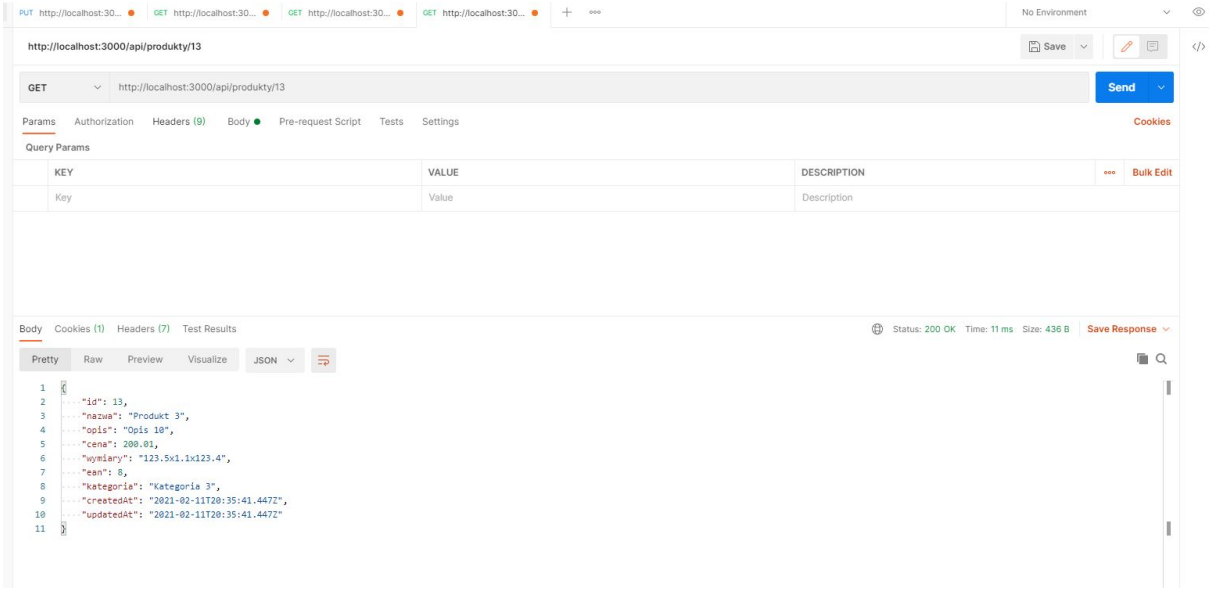
Dodawanie produktu



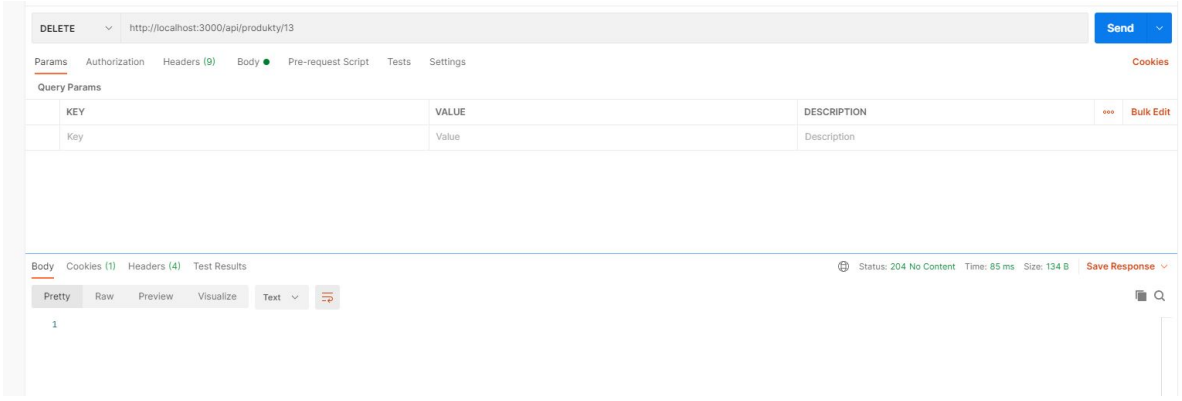
Edycja produktu



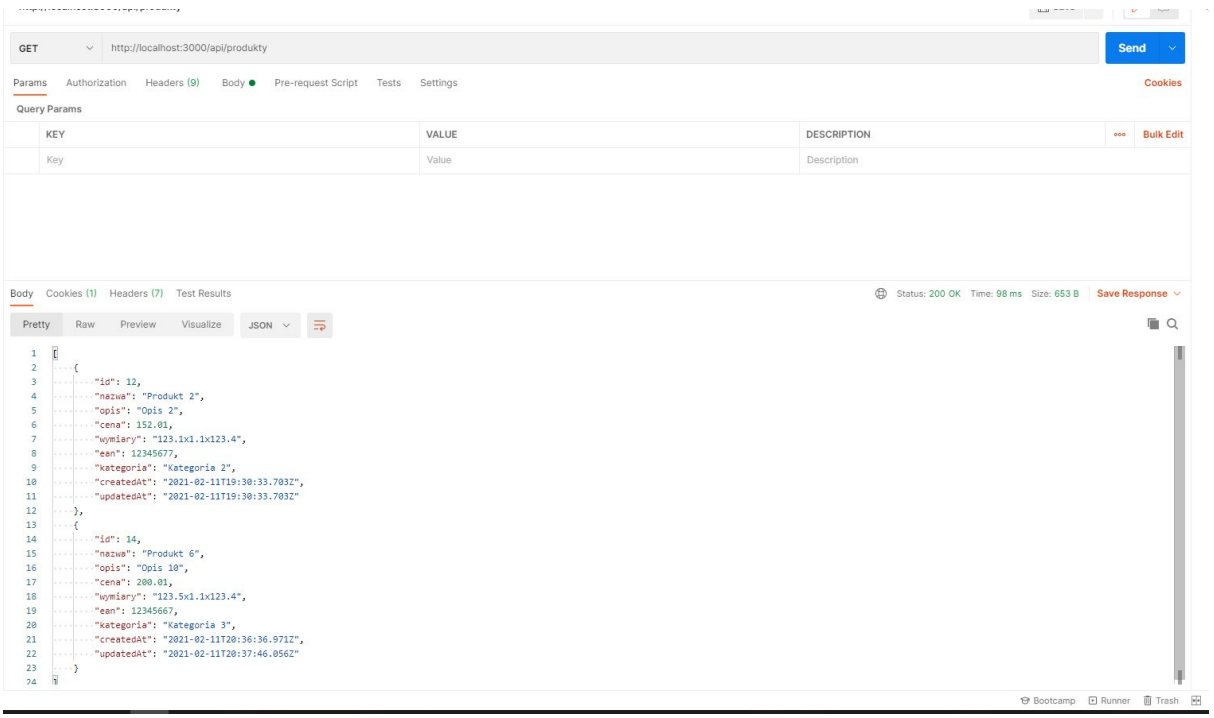
Jeden produkt



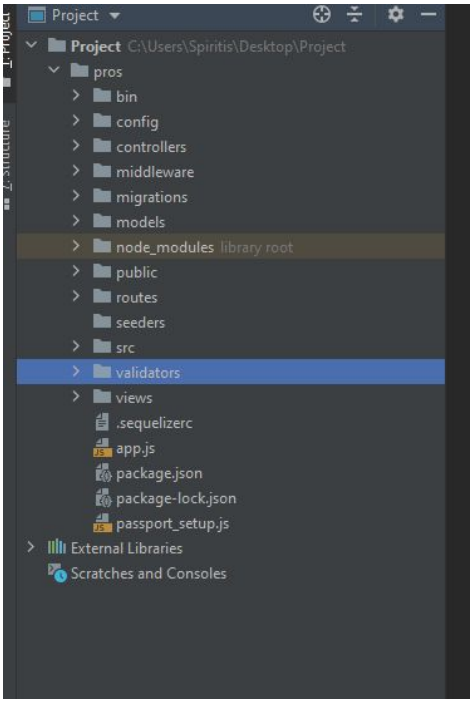
Usuwanie produktu

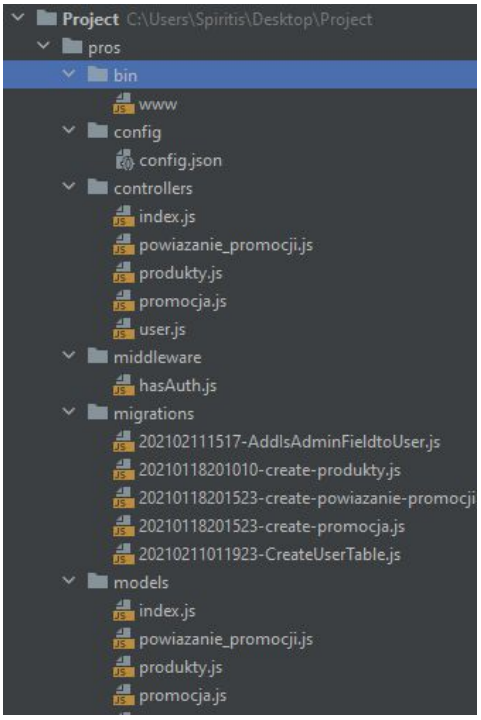


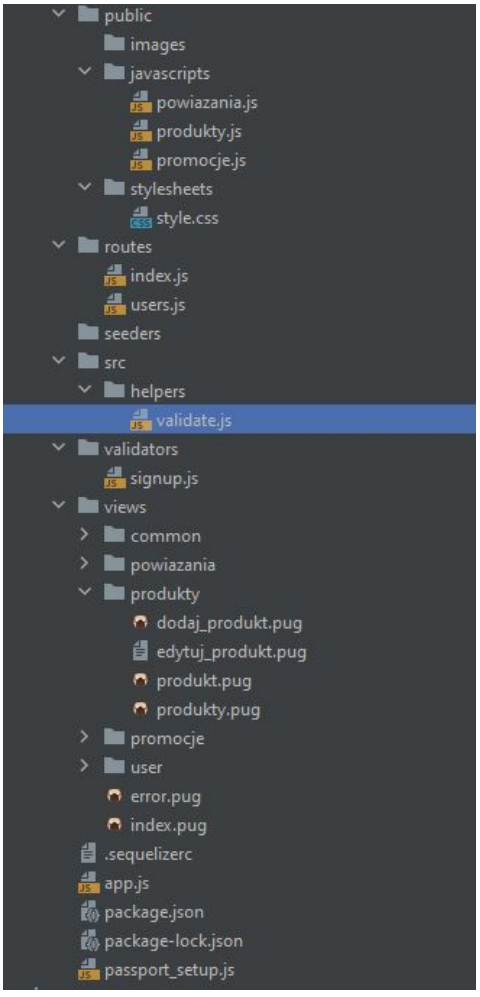
Lista wszystkich produktów

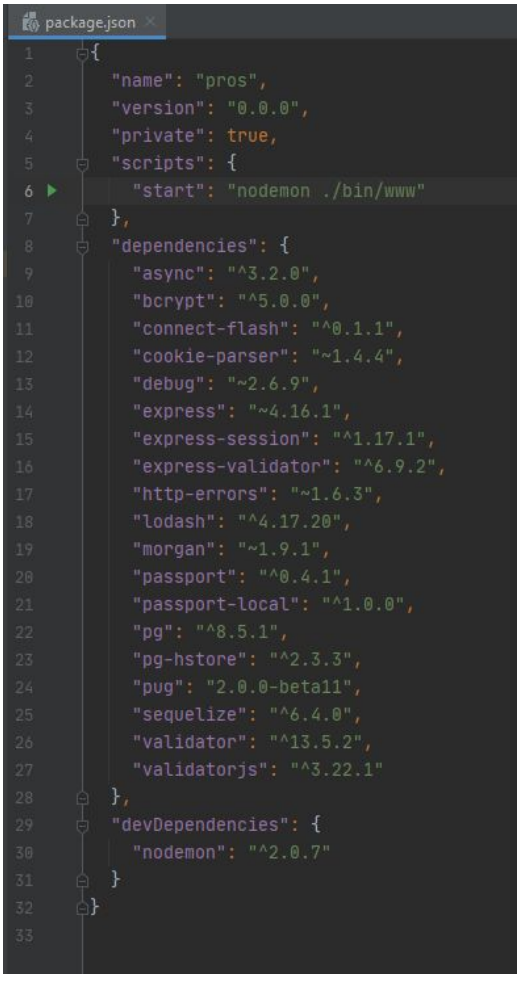


Struktura plików mikro serwisu









Przedstawienie poszczególnych kodów źródłowych odpowiadających za działanie mikro serwisu

Usuwanie produktu przy pomocy technologii AJAX

```
function usunProdukt(id) {
    $.ajax({
        url: '/products/produkty/' + id + '/delete-json',
        contentType: 'application/json; charset=utf-8',
        dataType: 'json',
        data: JSON.stringify( value: {id}),
        type: 'POST',
        success: ((res) => {
            // Replace follow button with unfollow.
            console.log("Result: ", res)
            $("#"+id).remove();
        }),
        error: ((error) => {
            console.log("Error:", error);
        })
    });
}
```

Przykładowy plik PUG do przedstawiania stron internetowych

```
doctype html
html(lang='en')
    include ../common/head.pug
    +head("Dodaj Produkt")
    body
        header
            include ../common/navbar.pug
            +navbar()
        .container
            .row.align-items-center
                .col-md-6.order-md-1.text-center.text-md-left.pr-md-5
                    h1 Dodaj Produkt
                    form(action="/products/produkt", method="POST")
                        .input-group.mb-3
                            input(name="nazwa", type="text", placeholder="Podaj nazwę produktu: ").form-control
                        .input-group.mb-3
                            input(name="opis", type="text", placeholder="Podaj opis produktu: ").form-control
                        .input-group.mb-3
                            input(name="cena", type="text", placeholder="Podaj cenę produktu: np(123.02)").form-control
                        .input-group.mb-3
                            input(name="wymiary", type="text", placeholder="Podaj wymiary produktu: (mm) ,(1.1x1.1x1.1),(dł x szer x wys)").form-control
                        .input-group.mb-3
                            input(name="ean", type="text", placeholder="Podaj EAN produktu: (8 cyfr)").form-control
                        .input-group.mb-3
                            input(name="kategoria", type="text", placeholder="Podaj kategorię produktu: ").form-control
                        .input-group.mb-3
                            button(type="submit").btn.btn-primary Potwierdz

                    a(href="/products/produkty").check3.btn.btn-primary Powrót
        include ../common/footer.pug
        +footer()
```

Definiowanie ścieżek do obsługi stron WWW

```
router.get( path: '/products/produkty',produktyKontroller.list2);
router.get('/products/produkt',produktyKontroller.pre_add2);
router.post('/products/produkt',produktyKontroller.add2);
router.get( path: '/products/produkt/:id', produktyKontroller.getById2);
router.get( path: '/products/produkt/:id/edit', produktyKontroller.pre_update2);
router.post('/products/produkt/:id/edit', produktyKontroller.update2);
router.post( path: '/products/produkt/:id/delete', produktyKontroller.delete2);
router.post( path: '/products/produkty/:id/delete-json', produktyKontroller.usun_produkt_json);
```

Przykładowe fragmenty kodu kontrolerów

```
list(req, res) {
    return Produkty
        .findAll( options: {
        }) Promise<Model[]>
        .then((produkties :Model[] ) => res.status(200).send(produkties)) Promise<Model[]>
        .catch((error) => { res.status(400).send(error); });
},
getById(req, res) {
    return Produkty
        .findByPk(req.params.id, options: {
        }) Promise<Model>
        .then((produkties :Model ) => {
            if (!produkties) {
                return res.status(404).send({
                    message: 'Produkt Not Found',
                });
            }
            return res.status(200).send(produkties);
        }) Promise<Model>
        .catch((error) => {
            console.log(error);
            res.status(400).send(error);
        });
},
```

```
update(req, res) {
    const validationRule={
        "nazwa": "required|string",
        "opis": "required|string",
        "cena": "required:regex:/^[0-9]{1,10}[.,]{1}[0-9]{1,2}$/",
        "wymiary": "required|regex:/^([0-9]{1,10}[.,]{1}[0-9]{1,2}[x,X]{1}[0-9]{1,10}[.,]{1}[0-9]{1,2}[x,X]{1}[0-9]{1,10}[.,]{1}[0-9]{1,2})$/",
        "ean": "required|min:8",
        "kategoria": "required"
    }
    validator(req.body,validationRule, customMessages: {}, callback: (err,status)=>{
        if (!status) {
            res.status(412)
                .send({
                    success: false,
                    message: 'Validation failed',
                    data: err
                });
        } else {
            return Produkty
                .findByPk(req.params.id, options: {}) Promise<Model>
                .then(produkties => {
                    if (!produkties) {
                        return res.status(404).send({
                            message: 'coo? Not Found',
                        });
                    }
                    return produkties
                        .update({
                            nazwa: req.body.nazwa || produkties.nazwa,
                            opis: req.body.opis || produkties.opis,
                            cena: req.body.cena || produkties.cena,
                            wymiary: req.body.wymiary || produkties.wymiary,
                            ean: req.body.ean || produkties.ean,
                            kategoria: req.body.kategoria || produkties.kategoria,
                        })
                        .then(() => res.status(200).send(produkties))
                        .catch((error) => res.status(400).send(error));
                }) Promise<Model>
                .catch((error) => res.status(400).send(error));
        }
    });
},
```

```
    add2(req, res) {
        const validationRule={
            "nazwa": "required|string",
            "opis": "required|string",
            "cena": "required:regex:/^[0-9]{1,10}[.,]{1}[0-9]{1,2}$/",
            "wymiary": "required|regex:/^([0-9]{1,10}[.,]{1}[0-9]{1,2}[x,X]{1}[0-9]{1,10}[.,]{1}[0-9]{1,2}[x,X]{1}[0-9]{1,10}[.,]{1}[0-9]{1,2})$/",
            "ean": "required|size:8",
            "kategoria": "required",
        }
        validator(req.body,validationRule, customMessages: {}, callback: (err,status)=>{
            if (!status) {
                res.render('produkty/dodaj_produkt');
                //res.render('produkty/dodaj_produkt');
            } else {
                return Produkty
                    .create( values: {
                        nazwa: req.body.nazwa,
                        opis: req.body.opis,
                        cena: req.body.cena,
                        wymiary: req.body.wymiary,
                        ean: req.body.ean,
                        kategoria: req.body.kategoria,
                    }) Promise<Model>
                    //.then((produkties) => res.redirect('./produkty'))
                    .then((produkties :Model ) => res.redirect('/products/produkty')) Promise<Model>
                    .catch((error) => res.status(400).send(error));
            }
        });
```

```
delete2(req, res) {
    return Produkty
        .findByPk(req.params.id) Promise<Model>
        .then(produkties => {
            if (!produkties) {
                return res.status(400).send({
                    message: 'Product Not Found',
                });
            }
            return produkties
                .destroy() Promise<void>
                //.then(() =>  res.redirect('./produkty'))
                .then(() =>  res.redirect('/products/produkty')) Promise<void>
                //.catch((error) => res.redirect('./produkt/' + req.params.id));
                .catch((error) => res.redirect('/products/produkt/' + req.params.id));
        }) Promise<Model>
        //.catch((error) => res.redirect('./produkt/' + req.params.id));
        .catch((error) => res.redirect('/products/produkt/' + req.params.id));
},
usun_produkt_json(req, res, next) {
    return Produkty.destroy( options: {
        where: {
            id: req.params.id
        }
    }).then(result => {
        res.send({ msg: "Success" });
    })
},
```

Tworzenie migracji i modelu odpowiedzialnych za komunikację z bazą danych

```
'use strict';
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable( tableName: 'Produkties', attributes: {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      nazwa: {
        type: Sequelize.STRING
      },
      opis: {
        type: Sequelize.STRING
      },
      cena: {
        type: Sequelize.FLOAT
      },
      wymiary: {
        type: Sequelize.STRING
      },
      ean: {
        type: Sequelize.INTEGER(8)
      },
      kategoria: {
        type: Sequelize.STRING
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE
      }
    });
  },
  down: async (queryInterface, Sequelize) => {
    await queryInterface.dropTable( tableName: 'Produkties');
  }
};
```

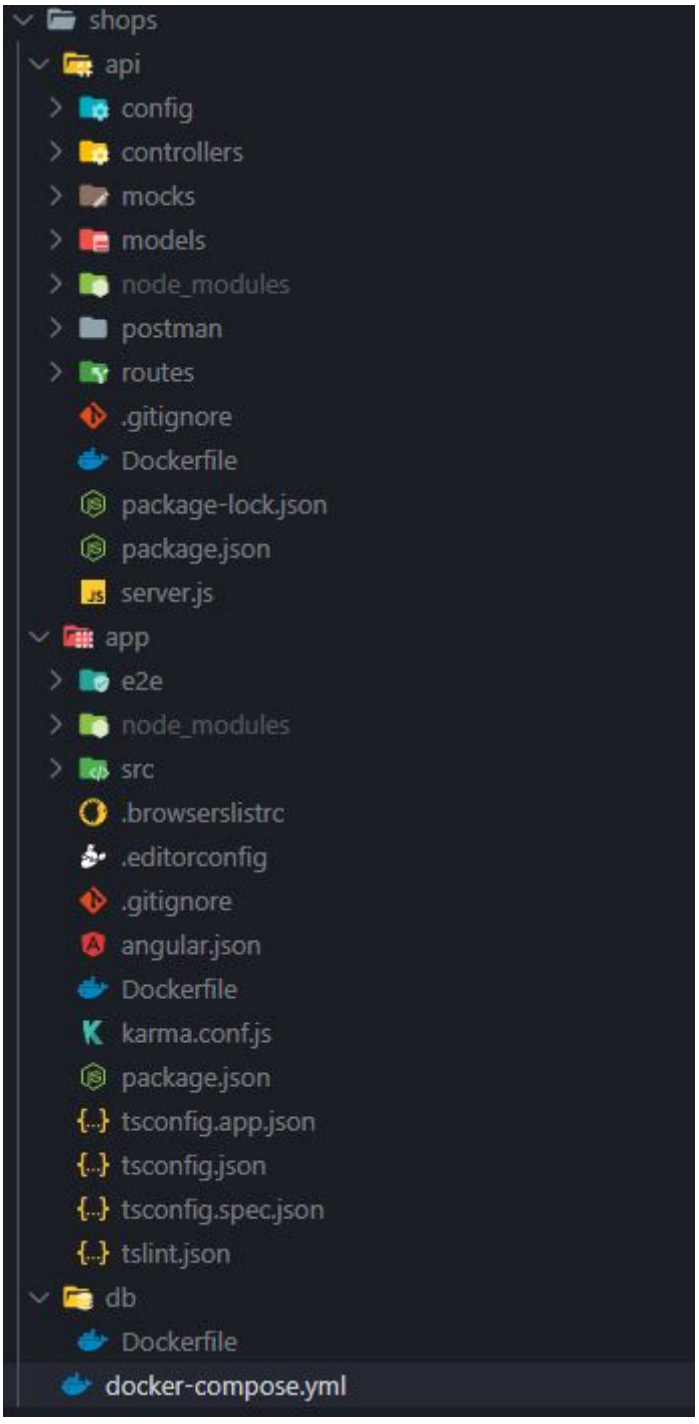
```
'use strict';
const {
  Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class Produkty extends Model {
    static associate(models) {
      Produkty.hasMany(models.Powiazanie_promocji, options: {
        foreignKey: 'id_produktu',
        as: 'produkty_powiazania',
      });
    }
  };
  Produkty.init( attributes: {
    nazwa: DataTypes.STRING,
    opis: DataTypes.STRING,
    cena: DataTypes.FLOAT,
    wymiary: DataTypes.STRING,
    ean: DataTypes.INTEGER,
    kategoria: DataTypes.STRING
  }, options: {
    sequelize,
    modelName: 'Produkty',
  });
  return Produkty;
};
```

Mikroserwis sklepów

Warstwa frontend: AngularJS
Warstwa backend: ExpressJS
Baza danych: MySQL

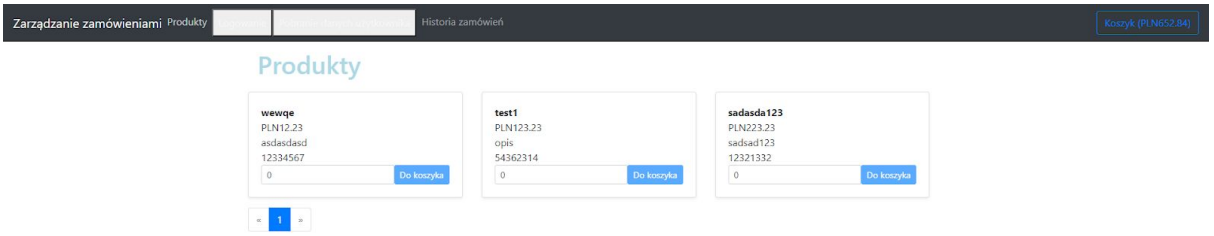
Opis

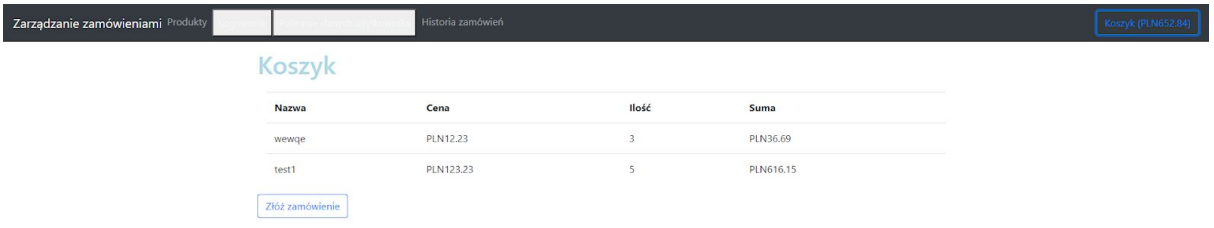
Mikroserwis pobiera dane o produktach i wyświetla je po stronie sklepów. Poprzez GUI możemy dodać produkty do koszyka i złożyć zamówienie.



Mikroserwis sklepów korzysta ze swojej osobnej kolekcji kontenerów z osobną bazą danych.

Warstwa frontend





Po dodaniu produktów do koszyka obliczana jest ich wartość zależnie od posiadanej w koszyku ilości.

Zawartość pliku docker-compose.yaml:

```
version:
"3"
```

```
services:

 db-service:

   image: mysql:latest

   environment:

            MYSQL_ROOT_PASSWORD:
password

     MYSQL_DATABASE: shop

   # ports:

   #   - "3306:3306"

   networks:

     - sh1-network

   volumes:

     - db-data:/data/db



 api-service:
```

```
    build: "./api"

    volumes:

      - ./api:/usr/src/api

                                  -
/usr/src/api/node_modules

    ports:

      - "8080:8080"

    networks:

      - sh1-network

      - wspolna

    depends_on:

      - db-service




  app-service:

    build: "./app"

    volumes:
```

```
      - ./app:/usr/src/app

                                  -
/usr/src/app/node_modules

    ports:

      - "8081:8081"

    networks:

      sh1-network:

      wspolna:

                    ipv4_address:
"10.1.0.200"




networks:

 sh1-network:

    driver: bridge

 wspolna:

    name: wspolna

    ipam:
```

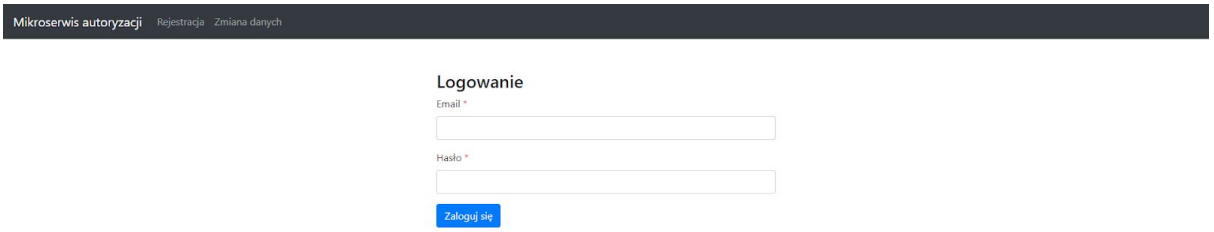
```
    driver: default

    config:

      - subnet: 10.1.0.0/24

volumes:

  db-data:

    driver: local
```

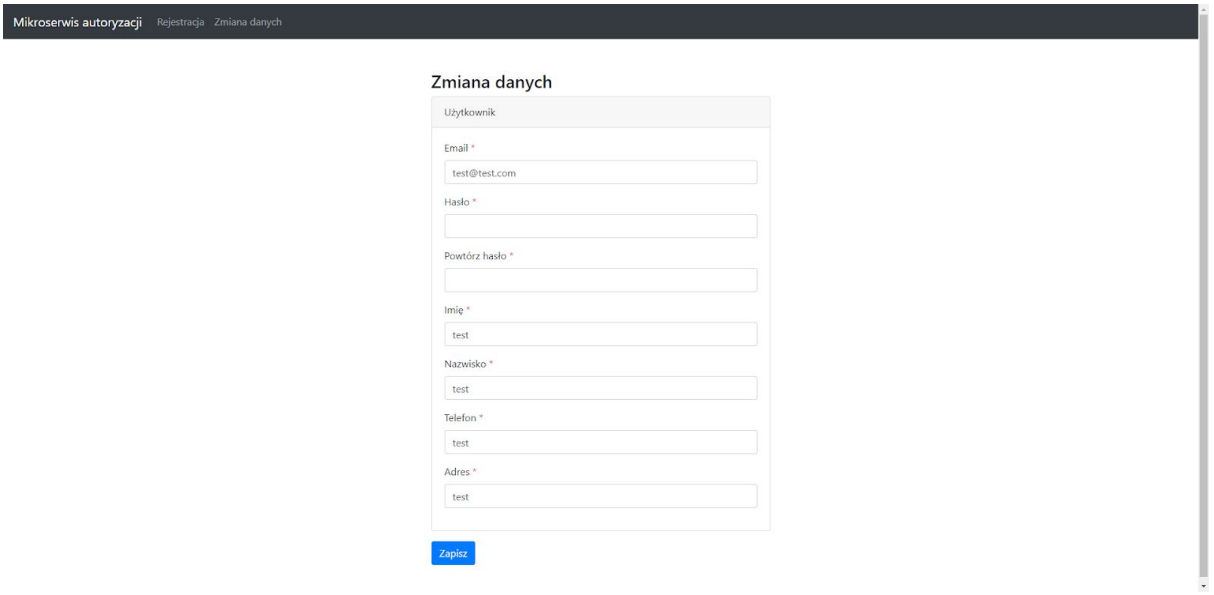
Mikroserwis autoryzacji

API: ExpressJS
Baza danych: MongoDB

Opis

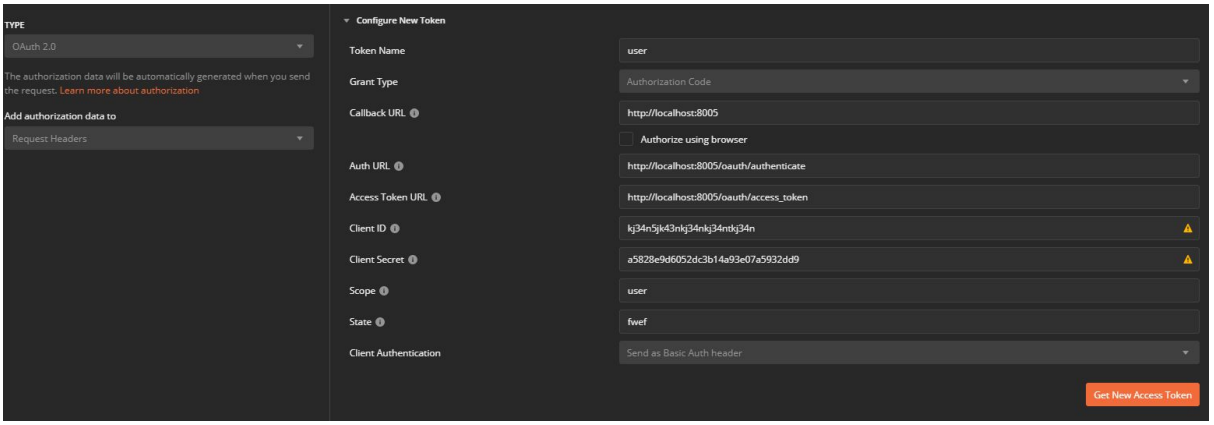
Mikroserwis autoryzacji pozwala użytkownikom na rejestrację, logowanie i zmianę swoich danych. Na tej podstawie można złożyć zamówienie i przypisać je do konkretnego użytkownika.

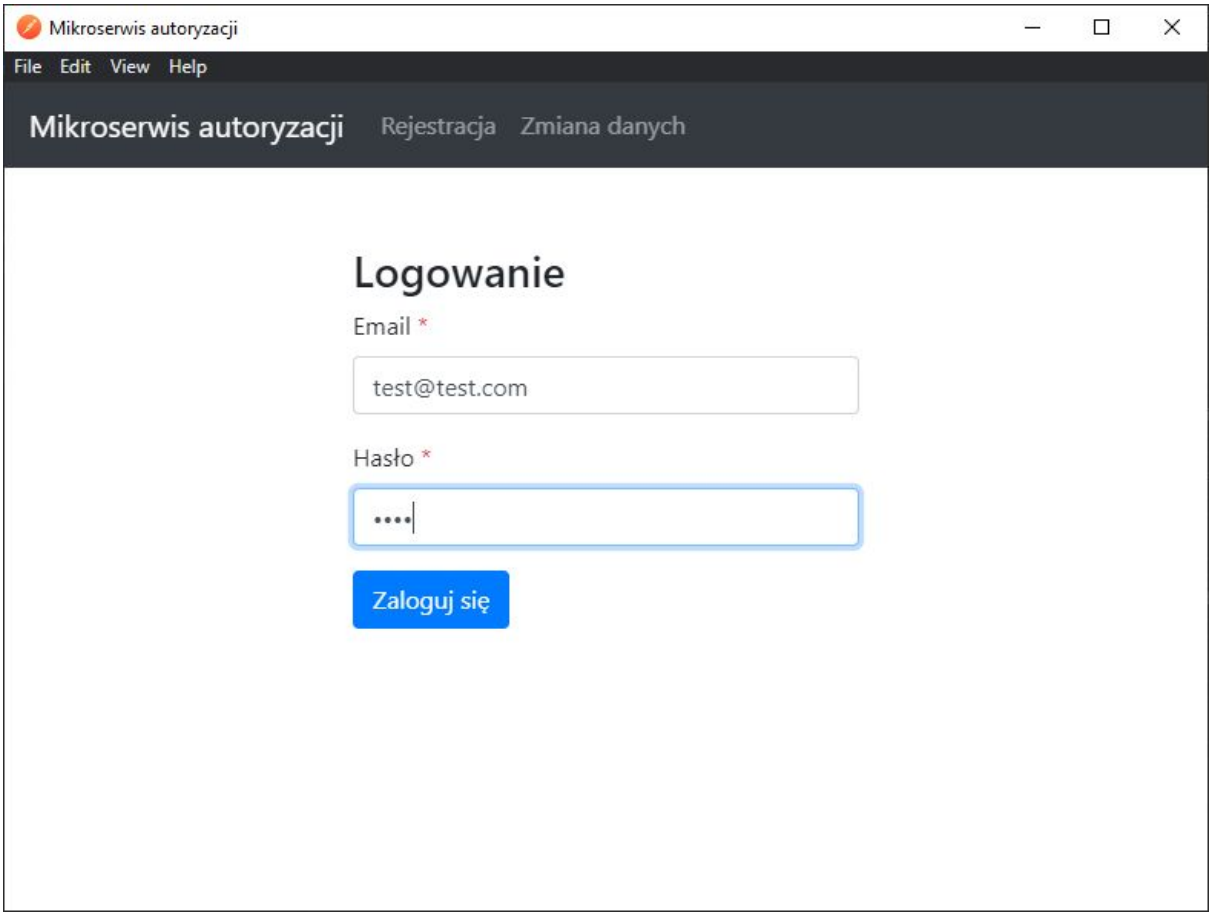


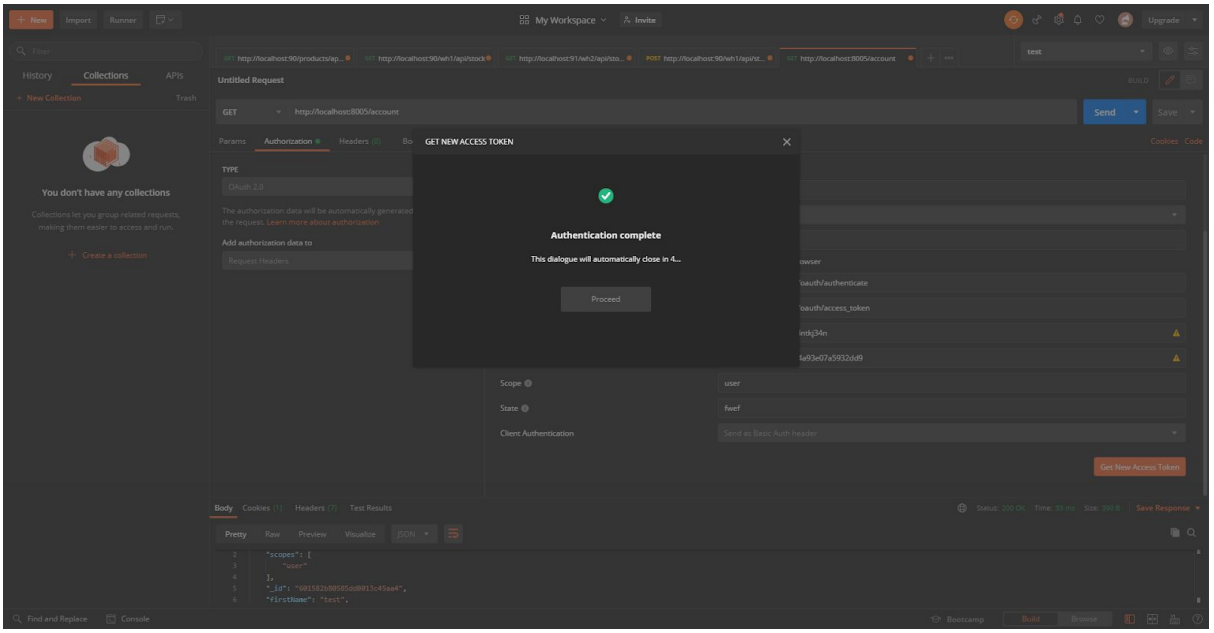


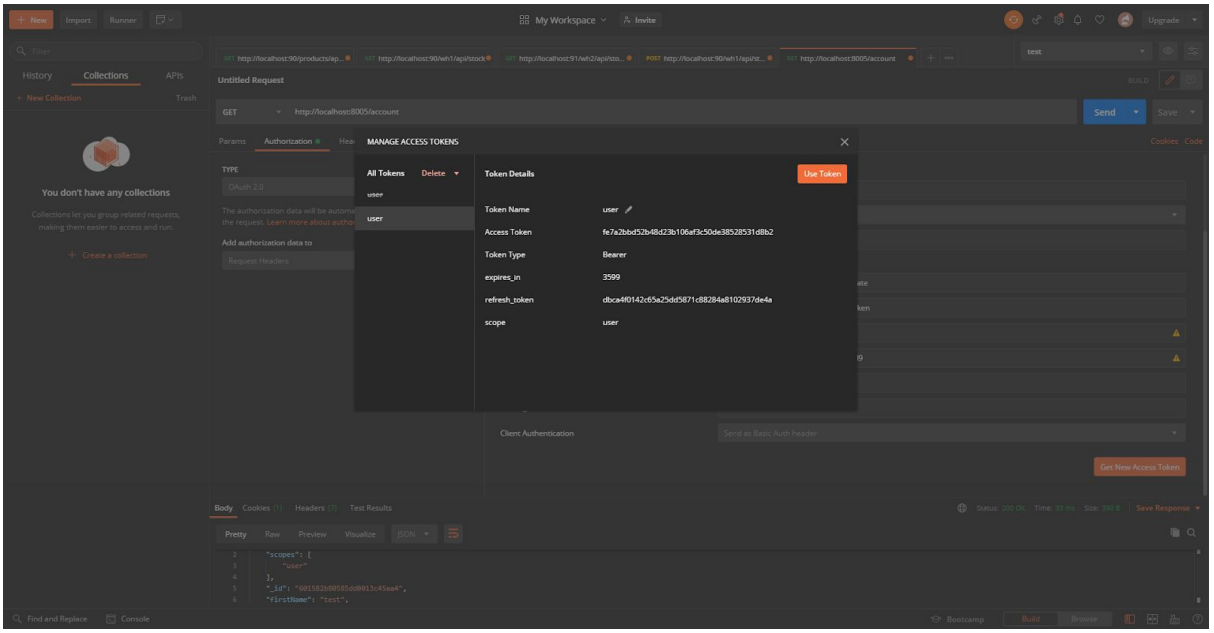
Mikroserwis posiada swoją odrębną kolekcję kontenerów z odrębną bazą danych.

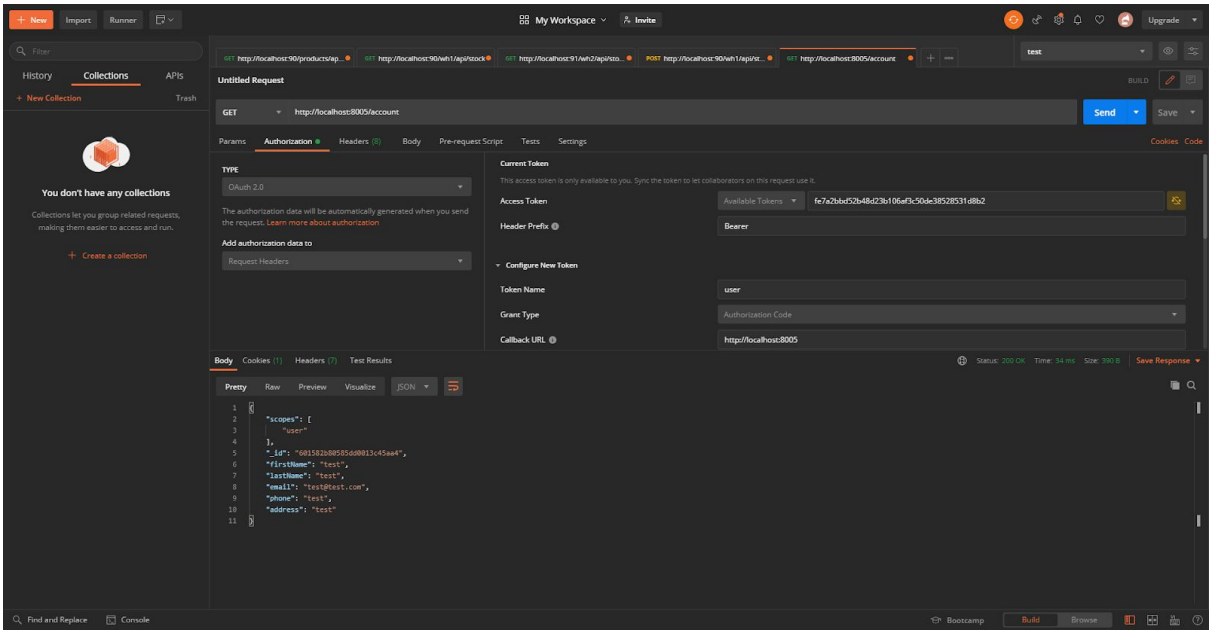
Prezentacja zapytań






GET NEW ACCESS TOKEN
Authentication complete
This dialogue will automatically close in 4...
Proceed
Untitled Request
Send
Save
Get New Access Token





Zawartość pliku docker-compose.yaml:

```
version:
"3"

services:
 api-service:
   build: "./api"
   volumes:
```

```
      - ./api:/usr/src/api
                                  -
/usr/src/api/node_modules
    ports:
      - "8005:8005"
    networks:
      auth-network:
      wspolna:
                  ipv4_address:
"10.1.0.212"
    depends_on:
      - db-service


 db-service:
    image: mongo:3.6.19-xenial
    ports:
      - "2720:27017"
    networks:
      - auth-network
    volumes:
      - db-data:/data/db


networks:
 auth-network:
    driver: bridge
 wspolna:
    name: wspolna
    ipam:
      driver: default
      config:
        - subnet: 10.1.0.0/24
volumes:
 db-data:
    driver: local
```