

System obsługi hurtowni elektrycznej

09.06.2021 r.

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
AGH University of Science and Technology

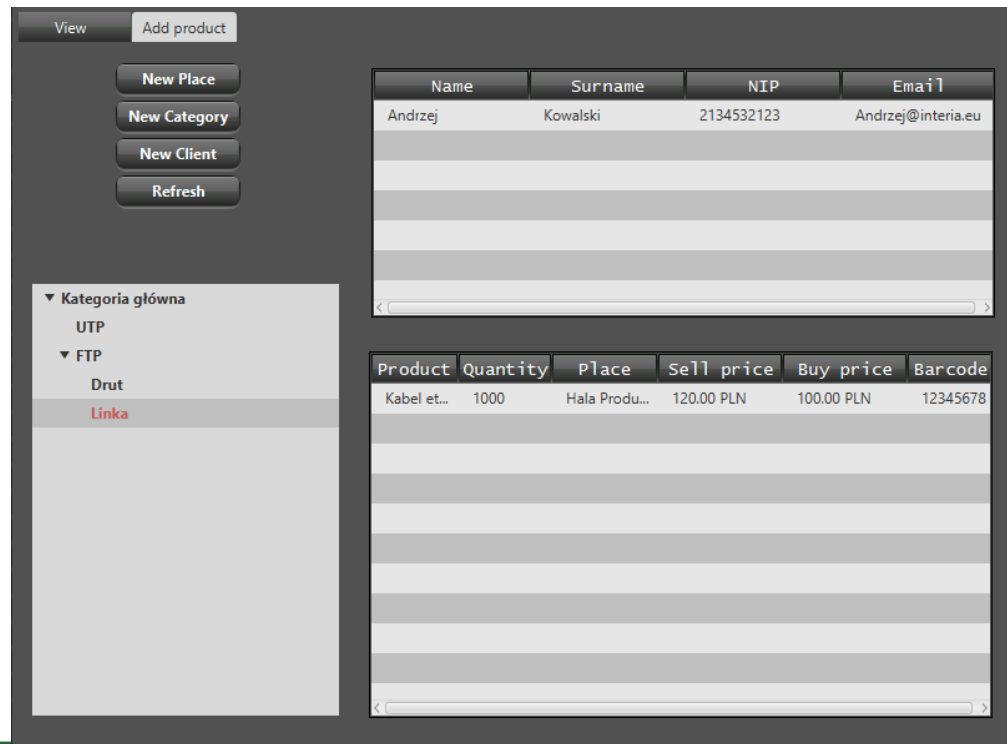
Mateusz Bednarski
Krzysztof Duda



Do czego właściwie służy nasz program?

Program powstał, aby zapewnić funkcjonalności potrzebne w nowoczesnej hurtowni elektrycznej. Pozwala na:

- Zarządzanie produktami w różnych lokalizacjach(kilka filii, magazyn itp.)
- Ustalanie cen produktów indywidualnie dla zamówień
- Obsługa statusów zamówień
- Zarządzanie kontrahentami
- Administracja kategoriami



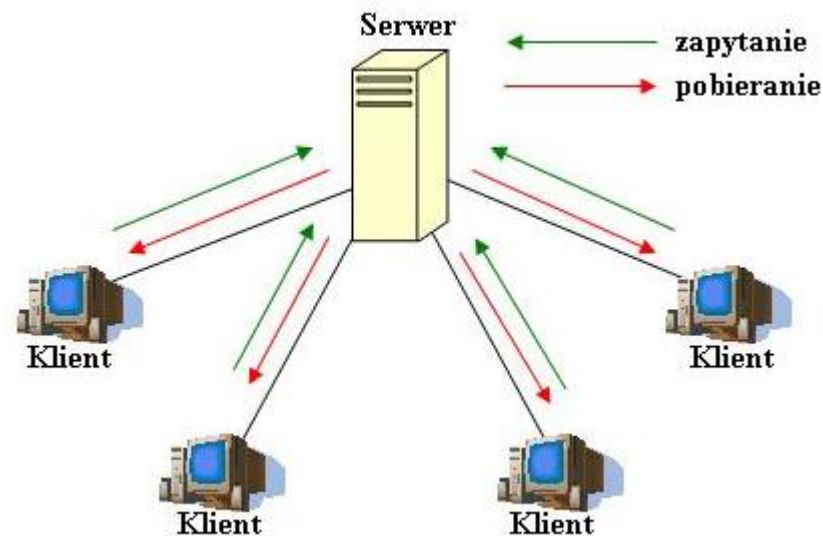
The screenshot displays the AGH program interface. On the left is a sidebar with a category tree under 'Kategoria główna', including 'UTP' and 'FTP' (with sub-item 'Drut' and a red 'Linka' link). The main area has a top bar with 'View' and 'Add product' tabs, and a vertical menu with 'New Place', 'New Category', 'New Client', and 'Refresh' buttons. Two data tables are shown:

Name	Surname	NIP	Email
Andrzej	Kowalski	2134532123	Andrzej@interia.eu

Product	Quantity	Place	Sell price	Buy price	Barcode
Kabel et...	1000	Hala Produ...	120.00 PLN	100.00 PLN	12345678

Jak to wszystko jest zbudowane?

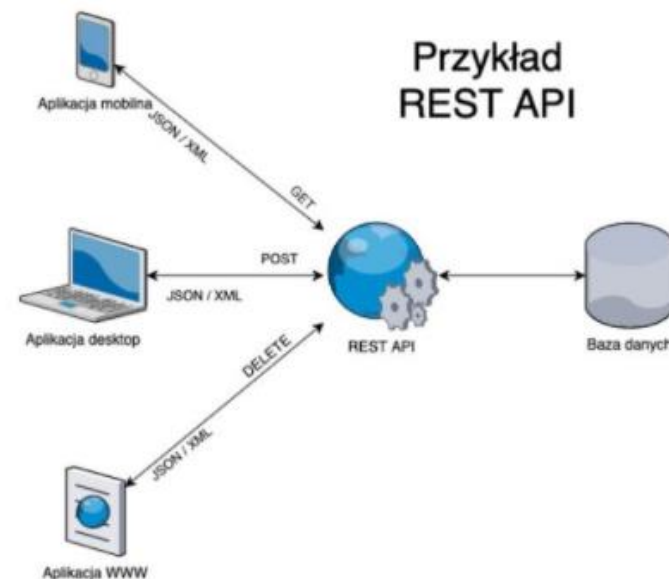
Backend(oprogramowanie serwera) zbudowane jest w oparciu o framework webowy dla Pythona – Flask. Dla każdego modelu występującego w aplikacji(użytkownik, kategoria, zamówienie itd.) stworzony został pełny interfejs CRUD obsługiwany przez zapytania HTTP/S. Frontend oparty został o język Java, przy wykorzystaniu biblioteki JavaFX, pozwalającej na tworzenie profesjonalnie wyglądających i łatwych w zarządzaniu interfejsów graficznych.



Czym jest Rest API?

REST (Representational State Transfer) – styl architektury oprogramowania, w którym wszelkie dane reprezentowane są w z góry określony, ustrukturyzowany sposób. W praktyce często przez obiekt (słownik) JSON

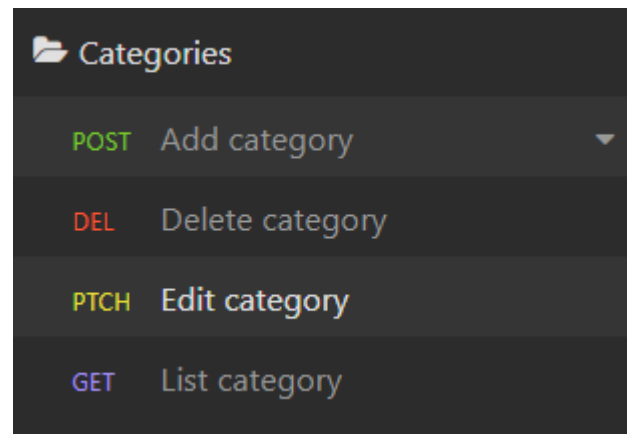
API – Application Programming Interface – zbiór reguł (w ujęciu programistycznym także klas i funkcji) pozwalający na komunikację z programem, usługą lub ich częścią.



Komunikacja HTTP

Aplikacja, komunikuje się z serwerem za pomocą zapytań HTTP/S, wykorzystując do tego metody takie jak:

- GET
- POST
- DELETE
- PATCH



```
> POST /products/ HTTP/2
> Host: hurtownia.mbednarski.pl
> user-agent: insomnia/2021.3.0
> content-type: application/json
> accept: */*
> content-length: 183

| {
|   "name" : "Kabel ethernet",
|   "category_id" : 1,
|   "description" : "nowy produkt",
|   "sell_price" : 20000,
|   "buy_price" : 10000,
|   "tax_id" : 1,
|   "unit_id" : 1,
|   "barcode" : "125673"
| }

* We are completely uploaded and fine

< HTTP/2 200
< server: nginx
< date: Sun, 06 Jun 2021 14:33:08 GMT
< content-type: application/json
< content-length: 24
< vary: Accept-Encoding
< x-powered-by: Phusion Passenger
< status: 200 OK

* Received 24 B chunk
* Connection #1 to host hurtownia.mbednarski.pl left intact
```

Frontend vs Backend

```
public void modifyProduct(String name, String cat_id, String description, int sell_price,
                        int buy_price, String tax_id, String unit_id, String barcode, String id){

    String json = null;
    URL url = null;

    AddProductJson addProductJson= new AddProductJson(name,cat_id,
        description,sell_price,buy_price,tax_id,unit_id,barcode);
    addProductJson.setId(id);
    try {
        url = new URL( spec: "http://hurtownia.mbednarski.pl/products/");
    } catch (MalformedURLException e) {
        System.out.println(e.toString());
    }

    ObjectMapper objectMapper = new ObjectMapper();

    try {
        json = objectMapper.writeValueAsString(addProductJson);
    } catch (IOException e) {
        e.printStackTrace();
    }

    System.out.println(json);
    patchMetod(json,url);
}
```

```
public void patchMetod(String json,URL url){

    HttpClient httpClient = HttpClient.newBuilder()
        .version(HttpClient.Version.HTTP_1_1)
        .build();

    HttpRequest.BodyPublisher jsonPayload = HttpRequest.BodyPublishers.ofString(json);
    HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create(String.valueOf(url)))
        .method( method: "PATCH", jsonPayload)
        .header( name: "Content-Type", value: "application/json")
        .build();

    HttpResponse<String> response = null;
    try {
        response= httpClient.send(request,HttpResponse.BodyHandlers.ofString());
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    System.out.println(response);
}
```

```
@app.route('/products', methods = ["POST","GET","DELETE","PATCH"])
def products():
    app.logger.info(request)
    if request.method == 'POST':
        if request.is_json:
            data = request.get_json()
            try:
                name = data['name']
                category_id = data['category_id']
                description = data['description']
                sell_price = data['sell_price']
                buy_price = data['buy_price']
                tax_id = data['tax_id']
                unit_id = data['unit_id']
                barcode = data['barcode']

            except:
                return error("Invalid JSON")
            record = Product(name, category_id,description,sell_price,tax_id,buy_price,unit_id,barcode)
            db.session.add(record)
            try:
                db.session.commit()
            except:
                return error("DB error when adding product")
            places = Place.query.all()
            for place in places:
                stock = Stock(record.id, place.id)
                db.session.add(stock)
            try:
                db.session.commit()
            except:
                return error("DB error when adding stocks")
            return success(id=record.id)
```

Frontend vs Backend

Aplikacja => Serwer

```
{  
  "name" : "Kabel ethernet",  
  "category_id" : 1,  
  "description" : "nowy produkt",  
  "sell_price" : 20000,  
  "buy_price" : 10000,  
  "tax_id" : 1,  
  "unit_id" : 1,  
  "barcode" : "125673"  
}
```

Serwer => Aplikacja

```
{  
  "id": 3,  
  "message": "OK"  
}
```

JSON, a obiekty Java

Obiekt \Rightarrow JSON

```
AddPlace addPlace= new AddPlace(name,description);
ObjectMapper objectMapper = new ObjectMapper();

try {
    json = objectMapper.writeValueAsString(addPlace);
} catch (IOException e) {
    e.printStackTrace();
}
```

```
package converters;

public class AddPlace {

    private String name;
    private String description;

    public AddPlace(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getDescription() { return description; }

    public void setDescription(String description) { this.description = description; }
}
```

JSON \Rightarrow Obiekt

```
URL url = null;
try {
    url = new URL( spec: "http://hurtownia.mbednarski.pl/places");
} catch (MalformedURLException e) {
    System.out.println(e.toString());
}

String result = new DbConnection().getString(url) ;

final ObjectMapper objectMapper = new ObjectMapper();
List<Place> langList = null;
try {
    langList = objectMapper.readValue(result, new TypeReference<List<Place>>(){});
} catch (IOException e) {
    e.printStackTrace();
}

return langList;
```


Komunikacja z serwerem

```
public void patchMetod(String json, URL url){

    HttpClient httpClient = HttpClient.newBuilder()
        .version(HttpClient.Version.HTTP_1_1)
        .build();

    HttpRequest.BodyPublisher jsonPayload = HttpRequest.BodyPublishers.ofString(json);
    HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create(String.valueOf(url)))
        .method( method: "PATCH", jsonPayload)
        .header( name: "Content-Type", value: "application/json")
        .build();

    HttpResponse<String> response = null;
    try {
        response= httpClient.send(request, HttpResponse.BodyHandlers.ofString());
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println(response);
}
```

Kontrola najwyższą formą zaufania

Projektując interfejs API działający przez HTTP/S i ogólnodostępny (nie wyizolowany od świata dostępny tylko przez VPN), musimy zwrócić uwagę na zagrożenia występujące w globalnej sieci. API może być narażone na ataki typu DDoS lub inne nastawione typowo na pozyskanie danych np. przez wykorzystanie podatności serwisu na tzw. wstrzykiwanie SQL (SQL-injection). W tej chwili większość otwartych i darmowych API i tak wykorzystuje mechanizmy autoryzacyjne, ponieważ pozwalają one na rejestrowanie aktywności pojedynczych użytkowników i ewentualne wykrywanie anomalii. Dzięki zastosowaniu autoryzacji przez Bearer Token, można bardzo skutecznie wykrywać ataki DDoS dzięki Web Application Firewall i odcinać niechciany ruch, jeszcze przed tym jak trafi na serwer mający dostęp do danych.

Bezpieczeństwo już na Frontendzie

```
try {
    selPrice = Integer.parseInt(sellPriceTextField.getText());
    sellPriceError.setText("");
} catch (NumberFormatException e) {
    sellPriceError.setText("Invalid value.");
    numberError = true;
}
for (ProductView pr :
    productListView) {
    if(barcodeTextField.getText().equals(pr.getBarcode())){
        errorBarcode.setText("Bad barcode");
        return;
    }else{
        errorBarcode.setText("");
    }
}
```

Name	<input type="text"/>	
Barcode	<input type="text"/>	
Buy Price	<input type="text" value="dwa"/>	Invalid value.
Sell Price	<input type="text" value="pięć"/>	Invalid value.
Quantity	<input type="text" value="jeden"/>	Invalid value.

Description	<div></div>	<button>Add info</button>
-------------	-------------	---------------------------

Product Name	<input type="text"/>
Category	<input type="text"/>
Sell Price	<input type="text"/>
Buy Price	<input type="text"/>
Tax	<input type="text"/>
Quantity	<input type="text"/>
Place	<input type="text"/>
Barcode	<input type="text"/>
Unit	<input type="text"/>
Description	<input type="text"/>

Create product

Lack of name!!!

Weryfikacja danych po stronie serwera

```
@app.route('/cartitems', methods = ["POST", "GET", "DELETE", "PATCH"])
def cartitems():
    app.logger.info(request)
    if request.method == 'POST':
        if request.is_json:
            data = request.get_json()
            try:
                product = data["product"]
                stock = data["stock"]
                quantity = data["quantity"]
                price = data["price"]
                order = data["order"]
            except Exception as e:
                print(e)
                return error("Invalid JSON")
            if Product.query.filter_by(id=product).count() < 1:
                return error("Invalid product")
            if Order.query.filter_by(id=order).count() < 1:
                return error("Invalid order")
            stock = Stock.query.filter_by(id=stock).first()
            if stock is None:
                return error("Invalid stock")
            if stock.quantity < quantity:
                return error("Not enough items in stock")
            stock.quantity -= quantity
            record = CartItem(order, product, quantity, price)
            db.session.add(record)
            try:
                db.session.commit()
            except Exception as e:
                print(e)
                return error("DB error when adding cart item")
            return success(id=record.id)
```

```
{
  "product":1,
  "stock":1,
  "quantity":1,
  "price":10,
  "order":"d"
}
```

```
{
  "id":1,
  "quantity":"pięć",
  "stock":1
}
```

```
{
  "id":333,
  "stock":1
}
```

```
{
  "product":1,
  "stock":1,
  "quantity":1,
  "price":"d",
  "order":1
}
```

Co mówi frontendowiec do backendowca?

Komunikacja frontend-backend powinna być możliwie bezpieczna, najlepiej gdy jest szyfrowana np. połączenia HTTPS i zabezpieczona (o tym na następnym slajdzie). Poza tym zawsze powinniśmy kierować się zasadą ograniczonego zaufania i wszelkie mechanizmy walidacji formularzy i danych przeprowadzać na frontendzie (aby dać natychmiastową informację użytkownikowi) i na backendzie (w przypadku uzyskania dostępu do API spoza aplikacji klienckiej). Stosowanie tych wytycznych pozwoli także na wykrycie błędów w komunikacji backend-frontend na wczesnym etapie tworzenia kodu.

Najpopularniejsze metody zabezpieczania interfejsów API HTTP/S

Basic auth – autoryzacja otwartym tekstem w zapytaniu w formie username:password, bezpieczna pod warunkiem stosowania HTTPS

OAuth2 – stosowanie zewnętrznego serwera uwierzytelniającego:

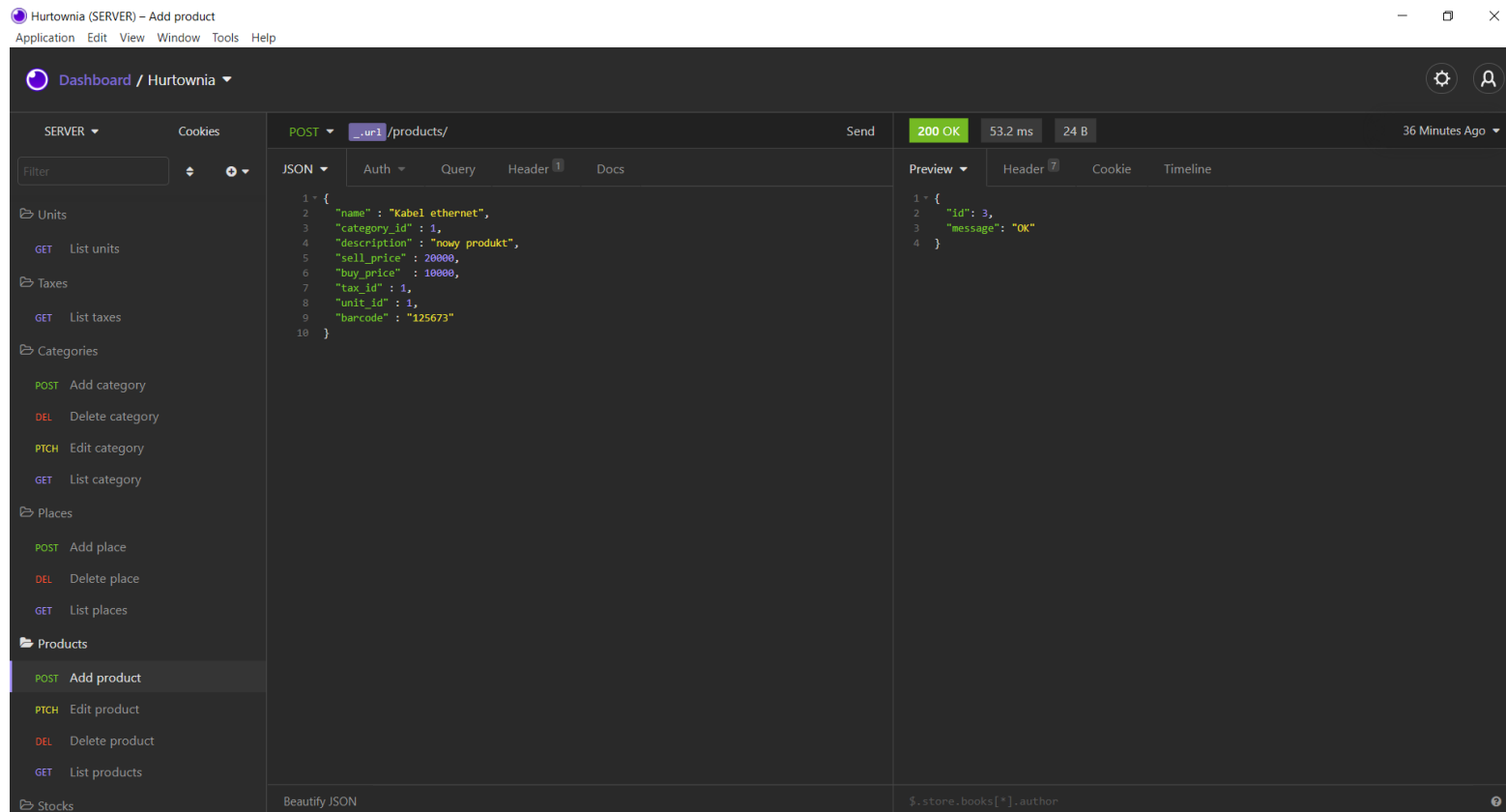
1. klient zwraca się do niego serwera uwierzytelniającego
2. serwer uwierzytelniający przyznaje mu indywidualny token
3. klient przekazuje token serwerowi aplikacji
4. serwer aplikacji weryfikuje token, przez zapytanie do serwera uwierzytelniającego
5. klientowi przyznawany jest dostęp

JWT (JSON web token) – uwierzytelnianie za pomocą tokenu:

1. klient przekazuje przy każdym zapytaniu swój indywidualny token serwerowi
2. serwer weryfikuje token i przyznaje dostęp do zasobów klientowi

Insomnia – API testing tool

Darmowe środowisko do testowania interfejsów API, oferuje wsparcie dla wszystkich metod HTTP, modyfikacje nagłóweków, automatyzację testów, obsługę wielu środowisk równocześnie, obsługę zmiennych



Dziękujemy za uwagę