

**POLITECHNIKA WROCŁAWSKA**  
**WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI**

---

**PROJEKT Z BAZ DANYCH**

**System bazodanowy do obsługi sklepu z grami komputerowymi wykorzystujący PostgreSQL**

AUTOR:

Szymon Kluska

Indeks: 263974

E-mail: 263974@student.pwr.edu.pl

PROWADZĄCY ZAJĘCIA:

Dr inż. Robert Wójcik, K30W04D03

OCENA PRACY:

---

Wrocław, 2023

## Spis treści

<b>1.</b>	<b>WSTĘP.....</b>	<b>5</b>
1.1.	CEL PROJEKTU.....	5
1.2.	ZAKRES PROJEKTU .....	5
<b>2.</b>	<b>ANALIZA WYMAGAŃ.....</b>	<b>6</b>
2.1.	OPIS DZIAŁANIA I SCHEMAT LOGICZNY SYSTEMU .....	6
2.2.	DIAGRAM WYMAGAŃ FUNKCJONALNYCH I NIEFUNKCJONALNYCH .....	6
2.3.	WYMAGANIA FUNKCJONALNE.....	7
2.3.1	<i>Diagram przypadków użycia .....</i>	7
2.3.2	<i>Scenariusze wybranych przypadków użycia.....</i>	8
2.4.	WYMAGANIA NIEFUNKCJONALNE .....	9
2.4.1	<i>Wykorzystywane technologie i narzędzia.....</i>	9
2.4.2	<i>Wymagania dotyczące rozmiaru bazy danych .....</i>	9
2.4.3	<i>Wymagania dotyczące bezpieczeństwa systemu.....</i>	9
2.5.	PRZYJĘTE ZAŁOŻENIA PROJEKTOWE .....	9
<b>3.</b>	<b>PROJEKT SYSTEMU .....</b>	<b>10</b>
3.1.	PROJEKT BAZY DANYCH.....	10
3.1.1	<i>Analiza rzeczywistości i uproszczony model konceptualny .....</i>	10
3.1.2	<i>Model logiczny i normalizacja .....</i>	10
3.1.3	<i>Model fizyczny i ograniczenia integralności danych.....</i>	11
3.1.4	<i>Inne elementy schematu – mechanizmy przetwarzania danych.....</i>	11
3.1.5	<i>Projekt mechanizmów bezpieczeństwa na poziomie bazy danych.....</i>	11
3.2.	PROJEKT APLIKACJI UŻYTKOWNIKA .....	12
3.2.1	<i>Architektura aplikacji.....</i>	14
3.2.2	<i>Interfejs graficzny i struktura menu .....</i>	14
3.2.3	<i>Projekt wybranych funkcji systemu.....</i>	15
3.2.4	<i>Metoda podłączania do bazy danych – integracja z bazą danych.....</i>	15
3.2.5	<i>Projekt zabezpieczeń na poziomie aplikacji.....</i>	16
<b>4.</b>	<b>IMPLEMENTACJA SYSTEMU .....</b>	<b>17</b>
4.1.	REALIZACJA BAZY DANYCH .....	17
4.1.1	<i>Tworzenie tabel i definiowanie ograniczeń.....</i>	17
4.1.2	<i>Implementacja mechanizmów przetwarzania danych.....</i>	19
4.1.3	<i>Implementacja uprawnień i innych zabezpieczeń .....</i>	20
4.2.	REALIZACJA ELEMENTÓW APLIKACJI .....	21
4.2.1	<i>Obsługa menu .....</i>	21
4.2.2	<i>Walidacja i filtracja.....</i>	22
4.2.3	<i>Implementacja interfejsu dostępu do bazy danych .....</i>	23
4.2.4	<i>Implementacja wybranych funkcjonalności systemu .....</i>	26
4.2.5	<i>Implementacja mechanizmów bezpieczeństwa .....</i>	27
<b>5.</b>	<b>TESTOWANIE SYSTEMU .....</b>	<b>29</b>
5.1.	INSTALACJA I KONFIGUROWANIE SYSTEMU .....	29
5.2.	TESTOWANIE OPRACOWANYCH FUNKCJI SYSTEMU.....	29
5.2.1	<i>Testowanie funkcji dodawania produktu.....</i>	29
5.2.2	<i>Testowanie funkcji usuwania produktu .....</i>	30
5.3.	TESTOWANIE MECHANIZMÓW BEZPIECZEŃSTWA .....	30
5.3.1	<i>Test żądania POST bez wcześniejszego logowania.....</i>	30
5.3.2	<i>Test żądania POST po zalogowaniu jako klient .....</i>	31
5.3.3	<i>Test żądania POST po zalogowaniu jako pracownik.....</i>	31
5.4.	WNIOSKI Z TESTÓW .....	32
<b>6.</b>	<b>PODSUMOWANIE.....</b>	<b>33</b>
<b>7.</b>	<b>LITERATURA .....</b>	<b>34</b>

Rysunek 1 Schemat logiczny systemu .....	6
Rysunek 2 Diagram wymagań funkcjonalnych i niefunkcjonalnych.....	6
Rysunek 3 Diagram przypadków użycia.....	7
Rysunek 4 Model konceptualny bazy danych .....	10
Rysunek 5 Model logiczny bazy danych.....	10
Rysunek 6 Model fizyczny bazy danych.....	11
Rysunek 7 Zrzut ekranu strony głównej.....	12
Rysunek 8 Zrzut strony katalogu.....	12
Rysunek 9 Zrzut strony logowania.....	13
Rysunek 10 Zrzut strony rejestracji.....	13
Rysunek 11 Architektura aplikacji .....	14
Rysunek 12 Menu pracownika .....	14
Rysunek 13 Menu klienta.....	14
Rysunek 14 Zrzut ekranu katalogu gier w trybie dziennym.....	14
Rysunek 15 Projekt formularza służącego do rejestracji użytkownika.....	15
Rysunek 16 Projekt strony wyświetlającej dane o danym produkcie .....	15
Rysunek 17 Fragment kodu pliku User.swift .....	20
Rysunek 18 Fragment kodu pliku SessionController.swift .....	20
Rysunek 19 Fragment kodu pliku SessionController.swift .....	20
Rysunek 20 Obsługa menu w pliku +layout.svelte .....	21
Rysunek 21 Fragment pliku User.swift obsługujący walidacje danych .....	22
Rysunek 22 Wykorzystanie walidacji przy tworzeniu konta nowego użytkownika w pliku UserController.swift .....	23
Rysunek 23 Fragment pliku configure.swift .....	23
Rysunek 24 Fragment pliku Produkt.swift .....	24
Rysunek 25 Fragment kodu pliku ProduktyController.swift.....	24
Rysunek 26 Fragment kodu pliku ProduktyController.swift.....	25
Rysunek 27 Przykład działania filtracji w aplikacji webowej.....	25
Rysunek 28 Fragment kodu pliku routes.swift.....	26
Rysunek 29 Fragment kodu pliku SessionController.swift .....	27
Rysunek 30 Fragment kodu pliku configure.swift .....	27
Rysunek 31 Fragment kodu pliku SessionController.swift .....	27
Rysunek 32 Fragment kodu pliku login/+page.svelte .....	28
Rysunek 33 Wyniki testów jednostkowych.....	29
Rysunek 34 Fragment kodu AppTests.swift .....	29
Rysunek 35 Fragment kodu AppTests.swift .....	30
Rysunek 36 Zrzut ekranu wyniku testu POST /addProduct bez zalogowania .....	30
Rysunek 37 Zrzut ekranu wyniku testu POST /addProduct po zalogowaniu jako klient .....	31
Rysunek 38 Zrzut ekranu wyniku testu POST /addProduct po zalogowaniu jako pracownik.	31

Tabela 1 Opis przypadków użycia poszczególnych aktorów .....	8
Tabela 2 Technologie i narzędzia wykorzystane do danego elementu systemu.....	9

# **1. Wstęp**

## **1.1. Cel projektu**

Celem projektu jest implementacja bazy danych, prostego interfejsu użytkownika przeznaczonych do obsługi z poziomu Internetu sklepu z grami oraz serwera, który będzie obsługiwał zapytania do bazy danych wysłane z aplikacji webowej.

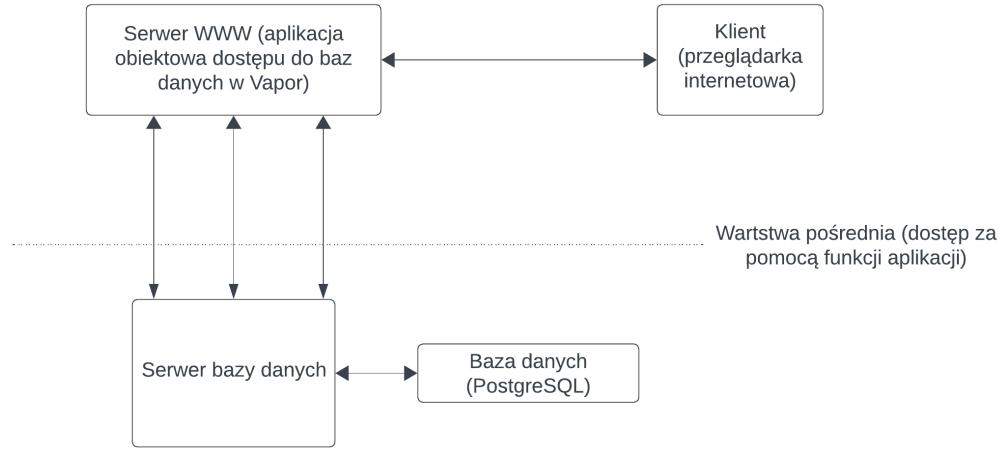
## **1.2. Zakres projektu**

W zakres projektu wchodzą różne etapy projektowania aplikacji. Najpierw jest analiza w której zostają wybrane technologie, które mają spełnić wymagania systemu oraz projektowanie wymagań funkcjonalnych, niefunkcjonalnych i diagramów przypadków użycia. W kolejnym etapie tworzony jest projekt systemu, czyli projekt bazy danych oraz trzech podstawowych modelów przedstawiających strukturę bazy danych oraz projekt aplikacji, która będzie korzystać z tej bazy danych. Następnie jest etap implementacji systemu i pokazania jak poszczególne funkcjonalności zostały zaimplementowane w kodzie. Następnie jest testowanie poprawności działania systemu. Na końcu jest podsumowanie i wnioski wyciągnięte z projektu.

## 2. Analiza wymagań

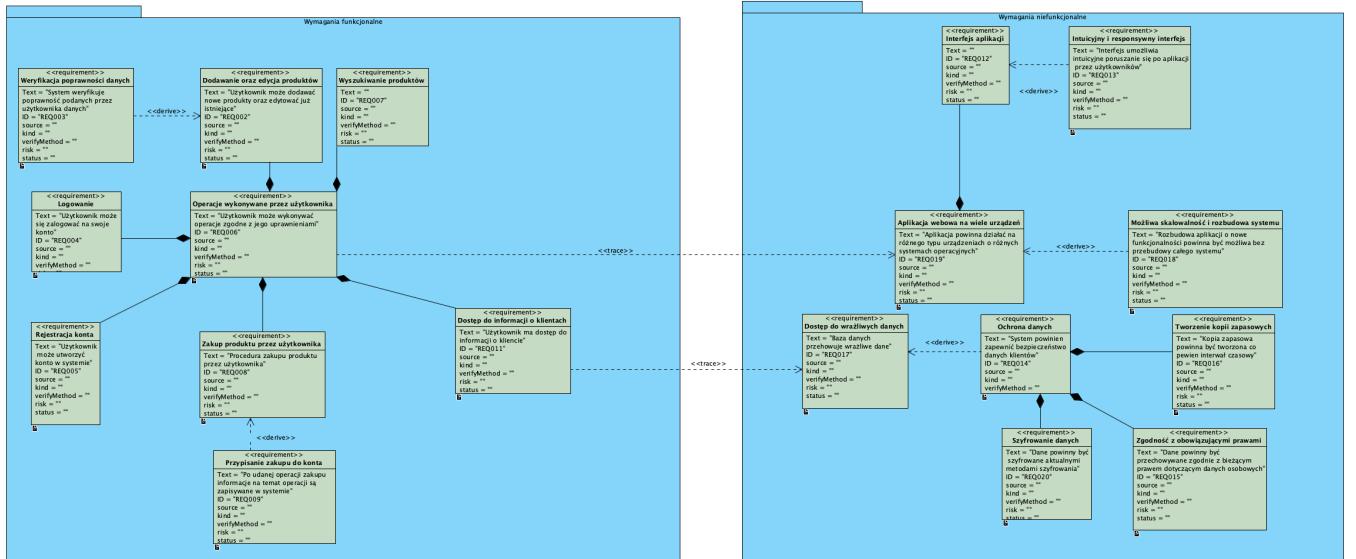
### 2.1. Opis działania i schemat logiczny systemu

Użytkownik będzie miał dostęp do strony sklepu za pomocą aplikacji webowej. Aplikacja webowa będzie wysyłać odpowiednie żądania REST do serwera WWW. Serwer będzie przetwarzanie żądań oraz będzie odpowiadać za wewnętrzne procesy i logikę aplikacji. Serwer będzie połączony z serwerem bazy danych do którego będzie wysyłać zapytania SQL. Serwer bazy danych po wykonaniu zapytań SQL będzie odsyłać do serwera wynik w odpowiednim formacie.



Rysunek 1 Schemat logiczny systemu

### 2.2. Diagram wymagań funkcyjonalnych i niefunkcyjnych

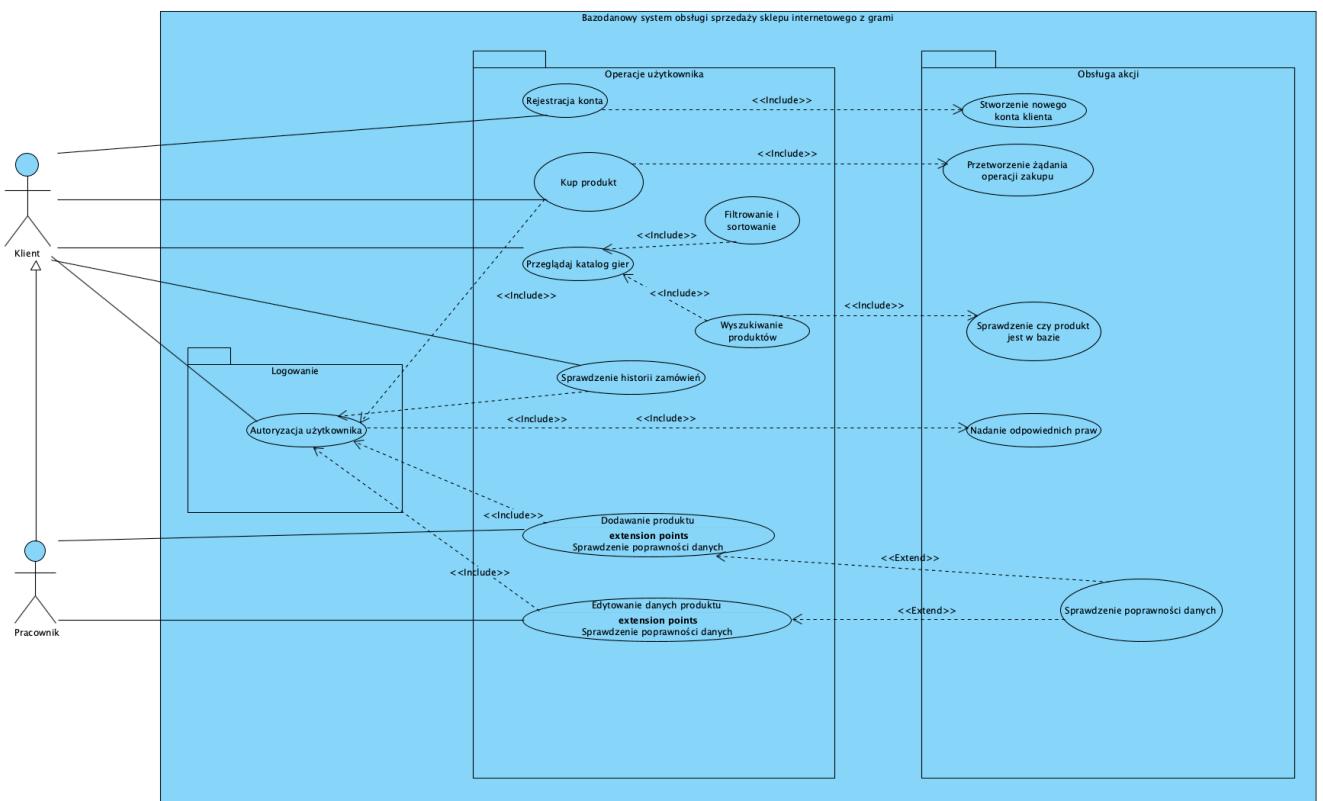


Rysunek 2 Diagram wymagań funkcyjonalnych i niefunkcyjnych

### 2.3. Wymagania funkcjonalne

- Pracownik sklepu ma możliwość wstawiania, usuwania oraz edycji przedmiotów w systemie
- Do systemu można dodać wiele produktów o takich samych danych prócz unikatowego ID
- System umożliwia rejestrację i logowanie.
- System wymaga zalogowania przed zakupem produktu
- Dane produktu są weryfikowane przed dodaniem do bazy czy zostały poprawnie podane
- System umożliwia wyszukiwanie produktów wg wielu kryteriów oraz ich sortowanie
- System umożliwia przejrzenie historii zakupów

#### 2.3.1 Diagram przypadków użycia



Rysunek 3 Diagram przypadków użycia

### 2.3.2 Scenariusze wybranych przypadków użycia

Tabela 1 Opis przypadków użycia poszczególnych aktorów

Aktor	Opis	Przypadki użycia
Klient	Klient może przeglądać katalog produktów, filtrować, sortować oraz wyszukiwać produkty. Może również zarejestrować konto. Po zalogowaniu może dokonać zakupu produktu oraz przeglądać historię swoich zamówień.	<ul style="list-style-type: none"> <li>• PU Rejestracja konta powiązane przez &lt;&lt;include&gt;&gt; z PU Stworzenie nowego konta klienta</li> <li>• PU Kup produkt powiązane przez &lt;&lt;include&gt;&gt; z PU Przetworzenie żądania operacji zakupu oraz PU Autoryzacja użytkownika</li> <li>• PU Przeglądaj katalog gier powiązane przez &lt;&lt;include&gt;&gt; z PU Wyszukiwanie produktów oraz PU Filtrowanie i sortowanie</li> <li>• PU Sprawdzenie historii zamówień</li> </ul>
Pracownik	Pracownik może dodatkowo dodawać oraz edytować produkty.	<ul style="list-style-type: none"> <li>• PU Dodawanie produktu powiązane przez &lt;&lt;include&gt;&gt; z PU Autoryzacja użytkownika oraz powiązane przez &lt;&lt;extend&gt;&gt; z PU Sprawdzenie poprawności danych</li> <li>• PU Edytowanie produktu powiązane przez &lt;&lt;include&gt;&gt; z PU Autoryzacja użytkownika oraz powiązane przez &lt;&lt;extend&gt;&gt; z PU Sprawdzenie poprawności danych</li> </ul>

#### PU Rejestracja konta

##### OPIS

CEL: Stworzenie nowego konta

WS (warunki wstępne): może być wywołany, jeśli użytkownik nie jest obecnie zalogowany

WK (warunki końcowe): podanie nazwy użytkownika, emaila, który nie został przypisany do żadnego konta, hasła, powtórzenie hasła oraz podanie imienia i nazwiska

##### PRZEBIEG:

1. Jeśli dane są poprawnie podane, aplikacja webowa wysyła żądanu do serwera o stworzenie konta, w przeciwnym wypadku użytkownik zostaje powiadomiony o błędnych danych.
2. Konto zostaje założone, użytkownik dostaje powiadomienie o poprawnej rejestracji i może się zalogować.

## PU Zakup produktu

### OPIS

CEL: Zakup produktu

WS (warunki wstępne): może być wywołany, jeśli użytkownik jest obecnie zalogowany

WK (warunki końcowe): przypisanie do konta nowego zamówienia

### PRZEBIEG:

1. Jeśli użytkownik jest zalogowany, zostaje wysłane żądanie do serwera o zakup.
2. Jeśli produkt jest na stanie użytkownik może go kupić.
3. Produkt po zakupie jest przypisany do konta.

## 2.4. Wymagania niefunkcjonalne

- Aplikacja webowa zapewni możliwość korzystania z niej z różnego rodzaju urządzeń
- Aplikacja webowa zapewni szybkość ładowania strony i działania serwisu
- Interfejs użytkownika jest responsywny i intuicyjny w obsłudze dla użytkownika
- System chroni dane przed dostępem do nich osób nieuprawnionych poprzez autoryzacje oraz szyfrowanie danych.
- Użytkownik z uprawnieniami pracownika ma dostęp do danych klientów.
- System jest zgodny z obowiązującymi przepisami ochrony danych.
- System ma możliwość rozbudowy i skalowalności

### 2.4.1 Wykorzystywane technologie i narzędzia

Tabela 2 Technologie i narzędzia wykorzystane do danego elementu systemu

	Technologia	Narzędzia
Klient	HTML, CSS, JavaScript + framework Svelte + Skeleton UI	WebStorm
Serwer WWW	Swift + framework Vapor	XCode
Serwer bazy danych	PostgreSQL	DataGrip

### 2.4.2 Wymagania dotyczące rozmiaru bazy danych

Brak wymagań dotyczących rozmiaru bazy danych.

### 2.4.3 Wymagania dotyczące bezpieczeństwa systemu

Przesyłane dane pomiędzy klientem, a serwerem powinny być szyfrowane. Aby zapewnić bezpieczeństwo strona internetowa musi mieć certyfikat SSL, a połączenie z nią powinno odbywać się przez protokół HTTPS. Hasła użytkowników powinny być szyfrowane według aktualnych standardów.

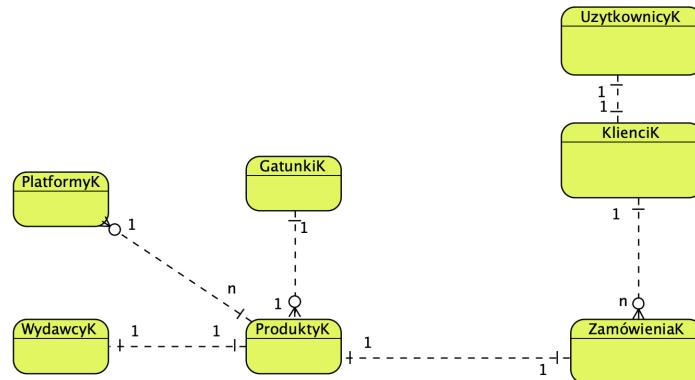
## 2.5. Przyjęte założenia projektowe

Funkcjonalności między klientem, a serwerem zostały zrealizowane tak aby wykorzystać jak najwięcej możliwości bazy danych i zapytań do niej. Niektóre funkcjonalności zostaną uproszczone jak np. brak certyfikatu SSL.

### 3. Projekt systemu

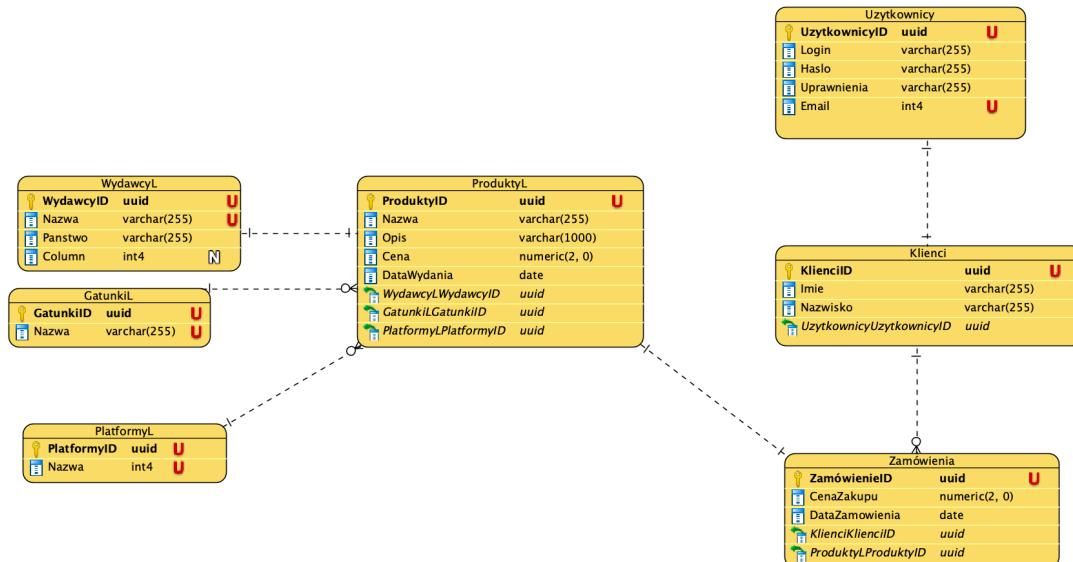
#### 3.1. Projekt bazy danych

##### 3.1.1 Analiza rzeczywistości i uproszczony model konceptualny



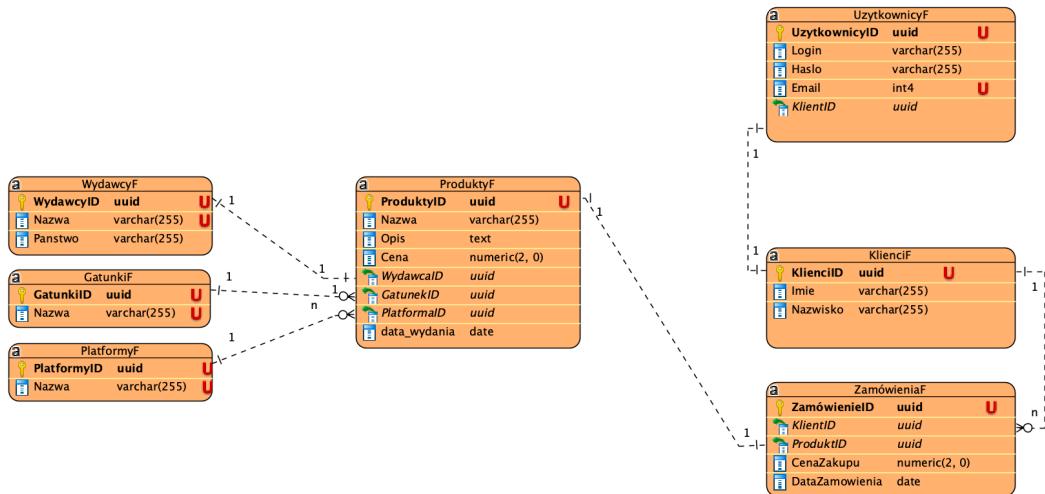
Rysunek 4 Model konceptualny bazy danych

##### 3.1.2 Model logiczny i normalizacja



Rysunek 5 Model logiczny bazy danych

### 3.1.3 Model fizyczny i ograniczenia integralności danych



Rysunek 6 Model fizyczny bazy danych

### 3.1.4 Inne elementy schematu – mechanizmy przetwarzania danych

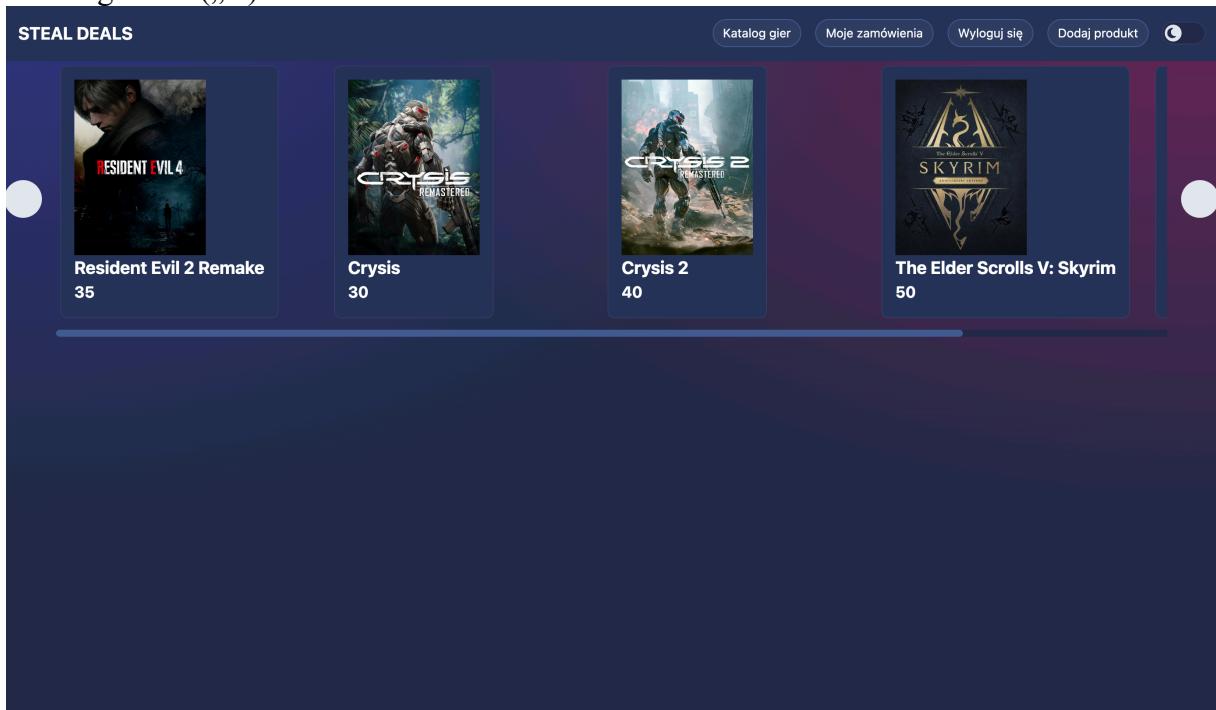
Funkcje dodajKlienta i dodajPracownika w PL/pgSQL służąca do założenia nowego konta o uprawnieniach klienta lub pracownika.

### 3.1.5 Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

Aby uzyskać dostęp do bazy danych trzeba zalogować się jako root. Tabele *klienci* i *uzytkownicy* są rozdzielone, aby dostęp do wrażliwych danych był jak najrzadziej używany.

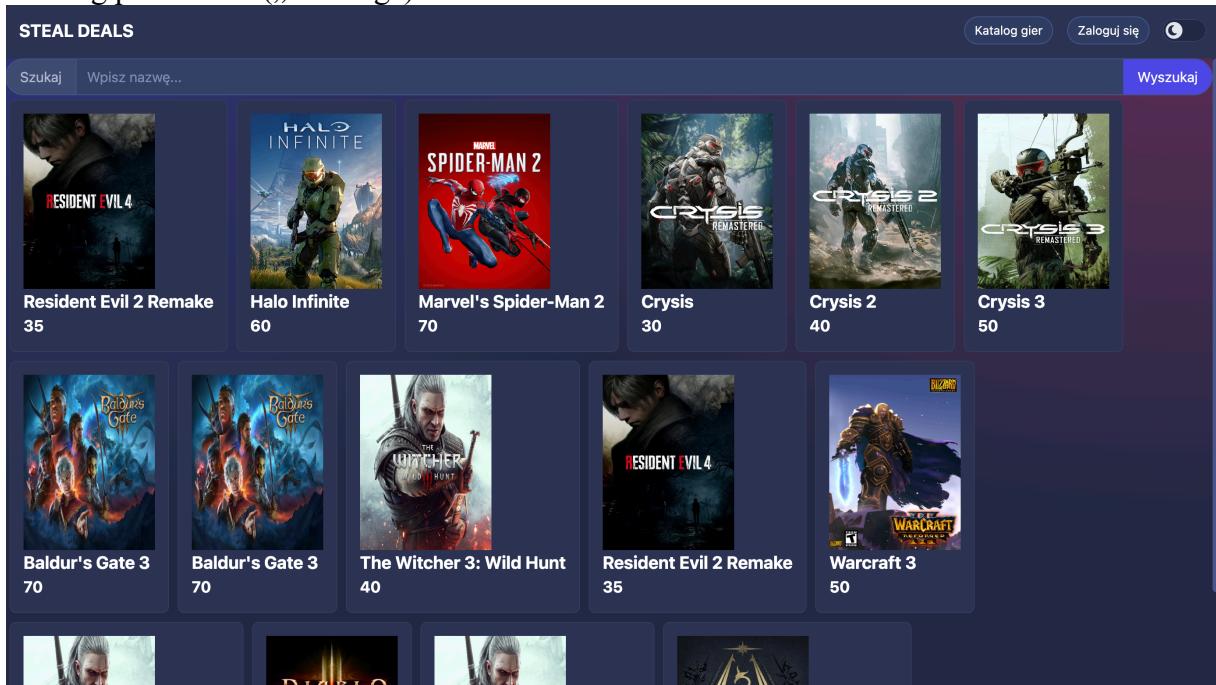
### 3.2. Projekt aplikacji użytkownika

- Strona główna („/”)



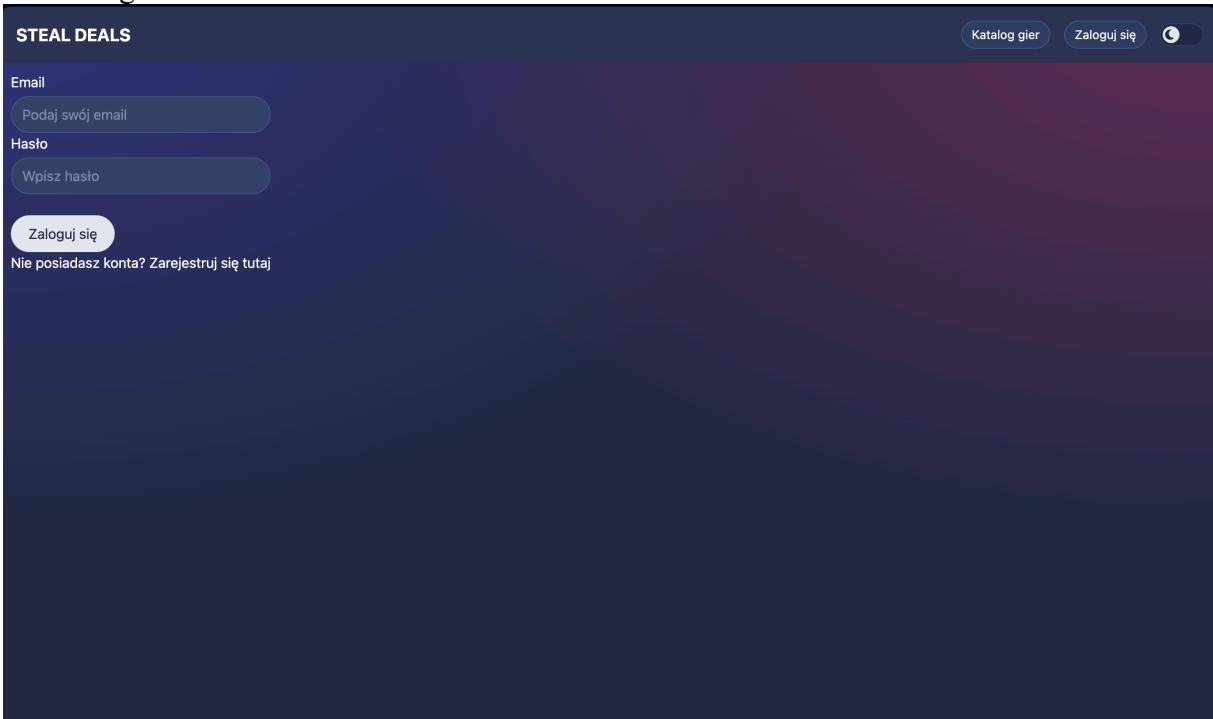
Rysunek 7 Zrzut ekranu strony głównej

- Katalog produktów („,/katalog”)



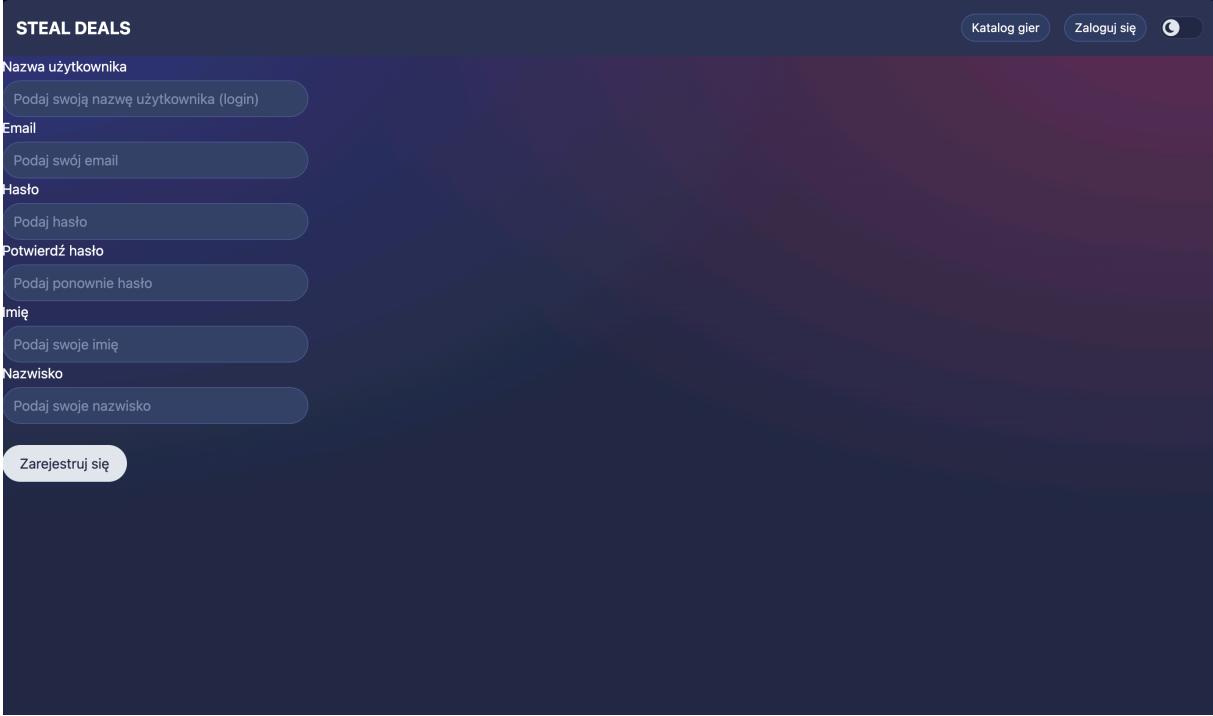
Rysunek 8 Zrzut strony katalogu

- Strona logowania



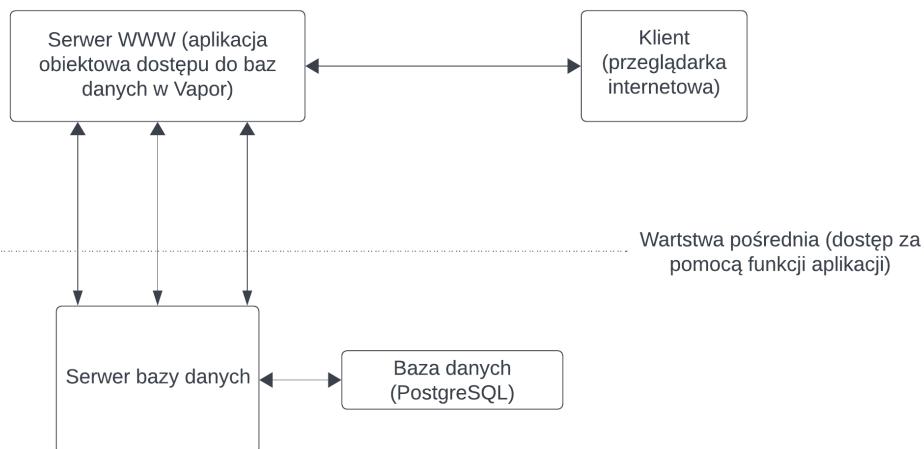
Rysunek 9 Zrzut strony logowania

- Strona rejestracji



Rysunek 10 Zrzut strony rejestracji

### 3.2.1 Architektura aplikacji



Rysunek 11 Architektura aplikacji

### 3.2.2 Interfejs graficzny i struktura menu

Interfejs graficzny aplikacji został stworzony przy pomocy framework'u Svelte [2] oraz biblioteki SkeletonUI [8].

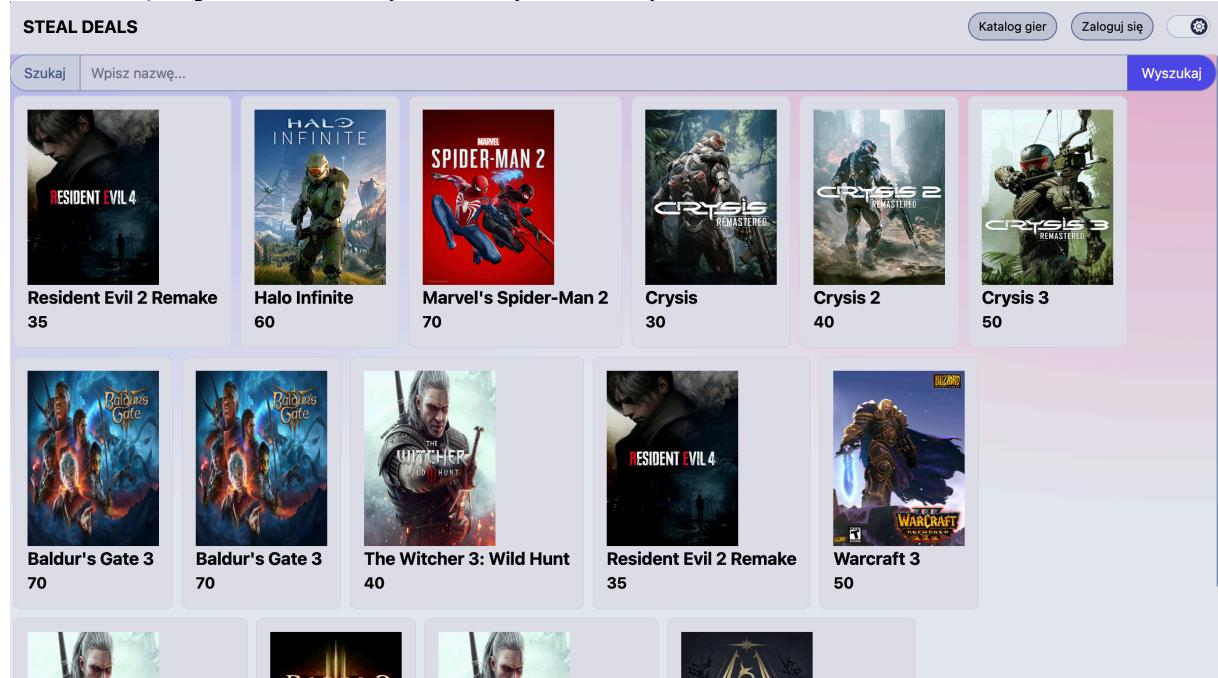


Rysunek 12 Menu pracownika



Rysunek 13 Menu klienta

Kliknięcie w element menu odsyła do odpowiednich podstron. Powrót do strony głównej następuje po kliknięciu w logo. Po kliknięciu w przycisk typu „toggle” z ikoną księżyca UI zmienia się odpowiednio w tryb dzienny lub nocny.



Rysunek 14 Zrzut ekranu katalogu gier w trybie dziennym

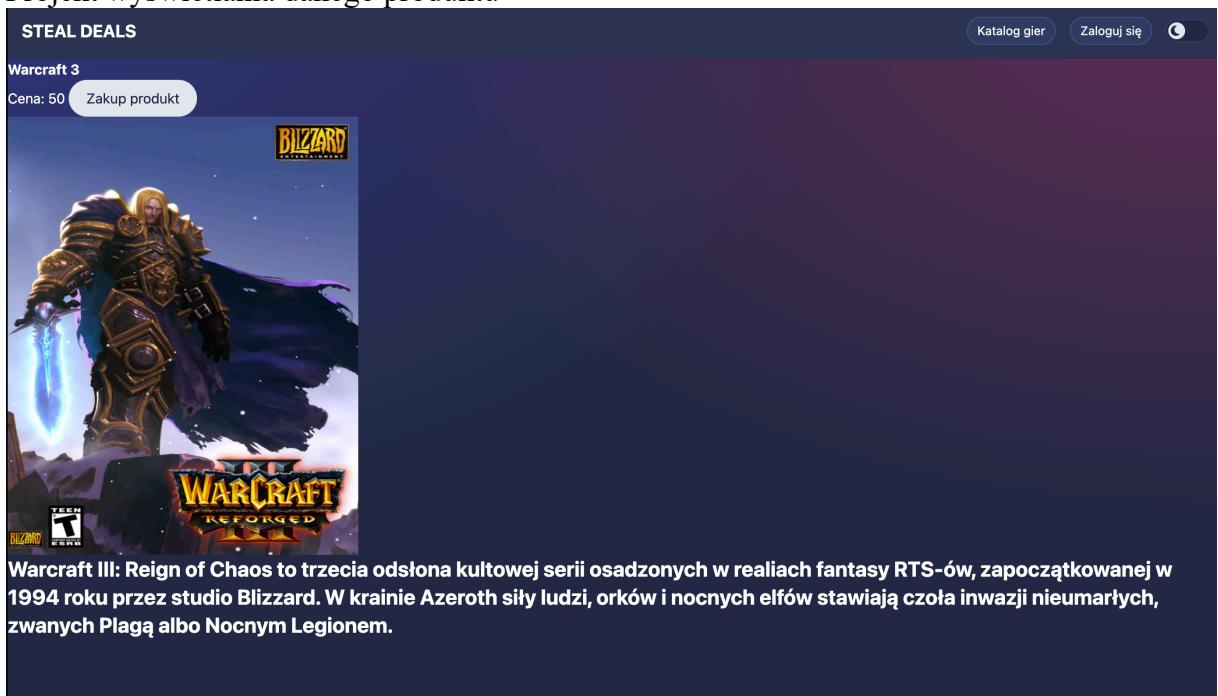
### 3.2.3 Projekt wybranych funkcji systemu

- Projekt rejestracji

The screenshot shows a dark-themed user registration form. At the top, it says 'STEAL DEALS'. Below that are fields for 'Nazwa użytkownika' (Username), 'Email', 'Hasło' (Password), 'Potwierdź hasło' (Confirm Password), 'Imię' (Name), and 'Nazwisko' (Surname). Each field has a placeholder text inside. At the bottom is a large blue button labeled 'Zarejestruj się' (Register).

Rysunek 15 Projekt formularza służącego do rejestracji użytkownika

- Projekt wyświetlania danego produktu



Rysunek 16 Projekt strony wyświetlającej dane o danym produkcie

### 3.2.4 Metoda podłączania do bazy danych – integracja z bazą danych

Integracja serwera z bazą danych przebiega przy pomocy frameworku ORM Fluent [6]. Fluent służy jako interfejs pomiędzy serwerem a bazą danych. Fluent ułatwia połączenie do bazy danych oraz wysyłania zapytań SQL.

### **3.2.5 Projekt zabezpieczeń na poziomie aplikacji**

Ścieżki (ang. *routes*), które według wymagań wymagają autoryzacji są zabezpieczone po stronie serwera i żeby wysłać żądanie do nich wymagane jest ustanowienie sesji. Autoryzacja przebiega przy użyciu metody *Basic Authentication* [7]. Serwer po otrzymaniu poprawnych danych logowania użytkownika ustanawia między klientem sesje oraz zapamiętuje po stronie serwera uprawnienia użytkownika oraz jego dane. Następnie odsyła odpowiedź do aplikacji klienta z danymi zapisanymi po stronie klienta w plikach *cookies* aby aplikacja klienta wiedziała, które elementy interfejsu wyświetlić. Jest to czysto estetyczne, ponieważ użytkownik nawet po ręcznej zmianie tych plików i próbie dostania się do stron wymagających innych uprawnień nie będzie w stanie dokonać zmian, bo ostateczna autoryzacja dzieje się po stronie serwera.

## 4. Implementacja systemu

### 4.1. Realizacja bazy danych

#### 4.1.1 Tworzenie tabel i definiowanie ograniczeń

```
create table gatunki
```

```
(
```

```
    gatunkiid uuid not null  
        primary key,  
    nazwa varchar(255) not null  
        unique
```

```
);
```

```
create table klienci
```

```
(
```

```
    klienciid uuid not null  
        primary key,  
    imie varchar(255) not null,  
    nazwisko varchar(255) not null
```

```
);
```

```
create table platformy
```

```
(
```

```
    platformyid uuid not null  
        primary key,  
    nazwa varchar(255) not null  
        unique
```

```
);
```

```
create table uzytkownicy
```

```
(
```

```
    uzytkownicyid uuid not null  
        primary key,  
    login varchar(255) not null,  
    haslo varchar(255) not null,  
    email varchar not null  
        unique,  
    klientid uuid not null  
        constraint fk_uzytkownicy_klienci  
            references klienci,  
    uprawnienia varchar not null
```

```
);
```

```
create table wydawcy
```

```
(
```

```
    wydawcyid uuid not null  
        primary key,  
    nazwa varchar(255) not null  
        unique,  
    panstwo varchar(255) not null
```

```
);
```

```
create table produkty
(
    produktyid uuid not null
        primary key,
    nazwa varchar(255) not null,
    opis text not null,
    cena numeric(10,2),
    wydawcaid uuid not null
        constraint fk_produkty_wydawcy
            references wydawcy,
    gatunekid uuid not null
        constraint fk_produkty_gatunki
            references gatunki,
    platformaid uuid not null
        constraint fk_produkty_platformy
            references platformy,
    data_wydania date,
    ilosc integer default 0 not null
        constraint produkty_ilosc_check
            check (ilosc >= 0)
);
```

```
create table zamowienia
(
    zamowienied id uuid not null
        constraint zamowienia_pkey
            primary key,
    klientid uuid not null
        constraint fk_zamowienia_klienci
            references klienci,
    produktid uuid not null
        constraint fk_zamowienia_produkty
            references produkty,
    cenazakupu numeric(10,2) not null,
    datazamowienia date not null
);
```

#### 4.1.2 Implementacja mechanizmów przetwarzania danych

- Funkcja dodajKlienta

```
CREATE OR REPLACE FUNCTION dodajKlienta(  
    imie_klienta VARCHAR,  
    nazwisko_klienta VARCHAR,  
    login_uzytkownika VARCHAR,  
    haslo_uzytkownika VARCHAR,  
    email_uzytkownika VARCHAR  
)  
RETURNS VOID AS $$  
DECLARE  
    uuid_klienta UUID := gen_random_uuid();  
    uuid_uzytkownika UUID := gen_random_uuid();  
BEGIN  
    INSERT INTO Klienci (KlienciID, Imie, Nazwisko)  
    VALUES (uuid_klienta, imie_klienta, nazwisko_klienta);  
  
    INSERT INTO Uzytkownicy (UzytkownicyID, Login, Haslo, Email, KlientID,  
    uprawnienia)  
    VALUES (uuid_uzytkownika, login_uzytkownika, haslo_uzytkownika,  
    email_uzytkownika, uuid_klienta, 'klient');  
END $$ LANGUAGE plpgsql;
```

- Funkcja dodajPracownika

```
CREATE OR REPLACE FUNCTION dodajPracownika(  
    imie_pracownika VARCHAR,  
    nazwisko_pracownika VARCHAR,  
    login_uzytkownika VARCHAR,  
    haslo_uzytkownika VARCHAR,  
    email_uzytkownika VARCHAR  
)  
RETURNS VOID AS $$  
DECLARE  
    uuid_klienta UUID := gen_random_uuid();  
    uuid_uzytkownika UUID := gen_random_uuid();  
BEGIN  
    INSERT INTO Klienci (KlienciID, Imie, Nazwisko)  
    VALUES (uuid_klienta, imie_pracownika, nazwisko_pracownika);  
  
    INSERT INTO Uzytkownicy (UzytkownicyID, Login, Haslo, Email, KlientID,  
    uprawnienia)  
    VALUES (uuid_uzytkownika, login_uzytkownika, haslo_uzytkownika,  
    email_uzytkownika, uuid_klienta, 'pracownik');  
END $$ LANGUAGE plpgsql;
```

#### 4.1.3 Implementacja uprawnień i innych zabezpieczeń

- Implementacja zabezpieczeń po stronie serwera

Struktura User służy do modelowania danych odwzorowując tabele *Uzytkownicy* w bazie danych. Aby służyła do weryfikowania danych logowania należy rozszerzyć ją o protokół *ModelAuthenticable*.

```
extension User: ModelAuthenticatable {
    static let usernameKey = \User.$email
    static let passwordHashKey = \User.$haslo
    func verify(password: String) throws -> Bool {
        try Bcrypt.verify(password, created: self.haslo)
    }
}
```

Rysunek 17 Fragment kodu pliku User.swift

W SessionController zostały zdefiniowane ścieżki (ang. *routes*), które wymagają autoryzacji oraz obsługa żądania POST do „/login”, które wykonuje próbę zalogowania. Jeśli zostaną podane błędne dane logowania to do aplikacji klienta zostaje zwrócony błąd. Po udanym zalogowaniu zostaje ustanowiona sesja oraz dane użytkownika zostają zapisane w plikach cookies sesji.

```
func boot(routes: Vapor.RoutesBuilder) throws {
    let passwordProtected = routes.grouped(User.authenticator())
    passwordProtected.post("login") { req -> [String:String] in
        let user = try req.auth.require(User.self)

        req.session.data["userId"] = user.id?.uuidString
        req.session.data["clientId"] = user.klientId.uuidString
        req.session.data["uprawnienia"] = user.uprawnienia
        let responseData: [String:String] = [
            "id": user.id?.uuidString ?? "",
            "uprawnienia": user.uprawnienia
        ]
        return responseData
    }

    passwordProtected.post("logout") { req -> HTTPStatus in
        req.session.data["userId"] = ""
        req.session.data["clientId"] = ""
        req.session.data["uprawnienia"] = ""
        return .ok
    }

    passwordProtected.group("user", "orders") { pass in
        pass.get(use: getOrders)
        pass.post(use: makeOrder)
    }
}
```

Rysunek 18 Fragment kodu pliku SessionController.swift

Przykład sprawdzenia autoryzacji oraz uprawnień.

```
passwordProtected.post("addProduct") { req -> HTTPStatus in

    if (req.session.data["uprawnienia"] == "pracownik") {
        let produkt = try req.content.decode(Produkt.self)
        try await produkt.save(on: req.db)
        return .ok
    } else {
        return .badGateway
    }
}
```

Rysunek 19 Fragment kodu pliku SessionController.swift

## 4.2. Realizacja elementów aplikacji

### 4.2.1 Obsługa menu

```
<svalte:fragment slot="lead">
    <a href="/">
        <strong class="text-xl uppercase">Steal Deals</strong>
    </a>
</svalte:fragment>
<svalte:fragment slot="trail">
    <a
        class="btn btn-sm variant-ghost-surface"
        href="/katalog"
        rel="noreferrer"
    >
        Katalog gier
    </a>

    {#if isLoggedIn}
        <a
            class="btn btn-sm variant-ghost-surface"
            href="/zamowienia"
            rel="noreferrer"
        >
            Moje zamówienia
        </a>

        <a
            class="btn btn-sm variant-ghost-surface"
            href="/"
            rel="noreferrer"
            on:click = {wylogujSie}
        >
            Wyloguj się
        </a>
    {:else}
        <a
            class="btn btn-sm variant-ghost-surface"
            href="/login"
            rel="noreferrer"
        >
            Zaloguj się
        </a>
   {/if}

    {#if getPermissions() === 'pracownik'}
        <a
            class="btn btn-sm variant-ghost-surface"
            href="/dodajProdukt"
            rel="noreferrer"
        >
            Dodaj produkt
        </a>
   {/if}

    <LightSwitch>

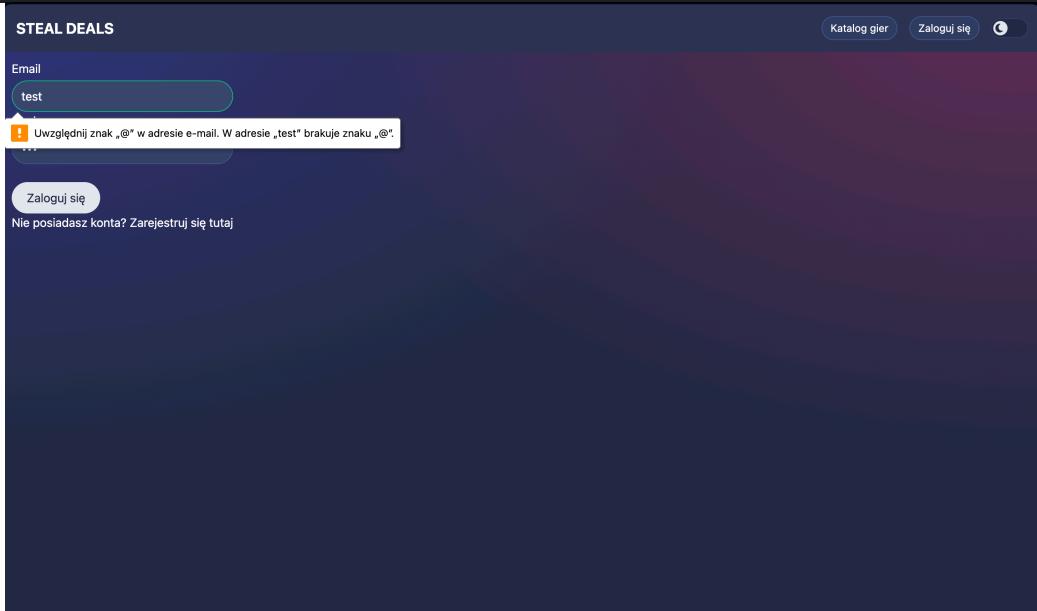
    </LightSwitch>
```

Rysunek 20 Obsługa menu w pliku +layout.svelte

#### 4.2.2 Walidacja i filtracja

WALIDACJA z poziomu aplikacji klienta opiera się na sprawdzeniu poprawności wprowadzanych danych zgodnie z typem pola *input* w formularzu.

```
<input bind:value={email} class="input" type="email" placeholder="Podaj swój email"/>
```



WALIDACJA z poziomu serwera jest bezpieczniejsza i gwarantuje nieprzepuszczenie nieprawidłowych danych.

```
extension User.Create: Validatable {
    static func validations(_ validations: inout Validations) {
        validations.add("login", as: String.self, is: !.empty)
        validations.add("email", as: String.self, is: .email)
        validations.add("haslo", as: String.self, is: .count(3...))
    }
}
```

Rysunek 21 Fragment pliku User.swift obsługujący walidacje danych

```

    func create(req: Request) async throws -> HTTPStatus {
        let klient = try req.content.decode(Klient.self)

        try User.Create.validate(content: req)
        let create = try req.content.decode(User.Create.self)
        guard create.haslo == create.potwierdzHaslo else {
            throw Abort(.badRequest, reason: "Podane hasła się nie zgadzają")
        }

        let user = try User(
            login: create.login,
            haslo: Bcrypt.hash(create.haslo),
            email: create.email
        )

        if let postgres = req.db as? PostgresDatabase {
            = postgres
            .simpleQuery("SELECT dodajklienta('\"(klient.imie)", '\"(klient.nazwisko)",
            '\\"(user.login)', '\"(user.haslo)', '\"(user.email)'")"
        } else {
            return .badRequest
        }
        return .ok
    }
}

```

Rysunek 22 Wykorzystanie walidacji przy tworzeniu konta nowego użytkownika w pliku UserController.swift

#### 4.2.3 Implementacja interfejsu dostępu do bazy danych

Dostęp do bazy danych przy użyciu Fluent:

- Zdefiniowanie konfiguracji dostępu do serwera bazy danych

```

app.databases.use(DatabaseConfigurationFactory.postgres(configuration: .init(
    hostname: Environment.get("DATABASE_HOST") ?? "localhost",
    port: Environment.get("DATABASE_PORT").flatMap(Int.init(_)) ??
        SQLPostgresConfiguration.ianaPortNumber,
    username: Environment.get("DATABASE_USERNAME") ?? "postgres",
    password: Environment.get("DATABASE_PASSWORD") ?? "bazydanych",
    database: Environment.get("DATABASE_NAME") ?? "postgres",
    tls: .prefer(try .init(configuration: .clientDefault)))
), as: .pgsql)

app.migrations.add(ProduktyMigration())

```

Rysunek 23 Fragment pliku configure.swift

- Zdefiniowanie modelu danych (rozszerzenie protokołu Model), który odwzorowuje daną tabelę

```

import Fluent
import Vapor

final class Produkt: Model, Content {

    static let schema = "produkty"

    @ID(custom: "produktyid")
    var id: UUID?

    @Field(key: "nazwa")
    var nazwa: String

    @Field(key: "opis")
    var opis: String

    @Field(key: "cena")
    var cena: Decimal

    @Field(key: "wydawcaid")
    var wydawcaid: UUID

    @Field(key: "gatunekid")
    var gatunekid: UUID

    @Field(key: "platformaid")
    var platformaid: UUID

    @Field(key: "data_wydania")
    var datawydanina: Date

    @Field(key: "ilosc")
    var ilosc: Int

    init() {}
    init(id: UUID? = nil, nazwa: String, opis: String, cena: Decimal, wydawcaid: UUID, gatunekid: UUID, platformaid: UUID, datawydanina: Date, ilosc: Int) {
        self.id = id
        self.nazwa = nazwa
        self.opis = opis
        self.cena = cena
        self.wydawcaid = wydawcaid
        self.gatunekid = gatunekid
        self.platformaid = platformaid
        self.datawydanina = datawydanina
        self.ilosc = ilosc
    }
}

```

Rysunek 24 Fragment pliku *Produkt.swift*

- Przykład użycia (zapytanie do bazy danych zwracające produkt o podanym id)

```

func show(req: Request) async throws -> Produkt {
    guard let produkt = try await Produkt.find(req.parameters.get("id"), on: req.db) else {
        throw Abort(.notFound)
    }
    return produkt
}

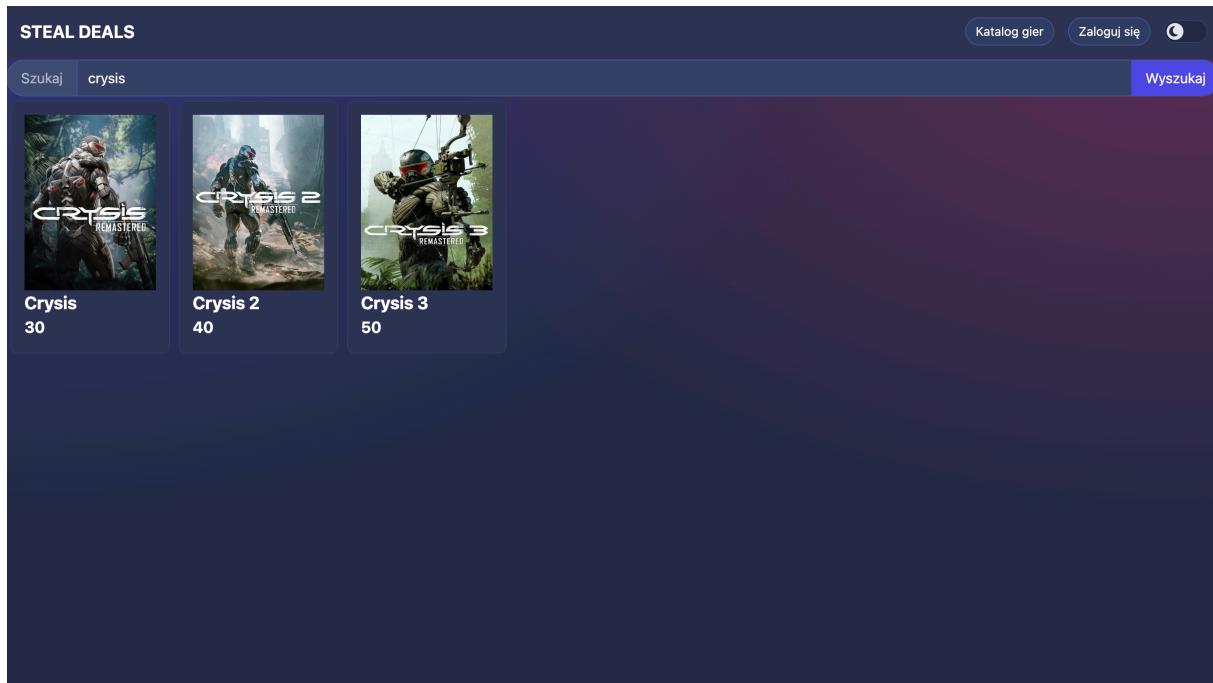
```

Rysunek 25 Fragment kodu pliku *ProduktyController.swift*

- Filtracja produktów według nazwy

```
func index(req: Request) async throws -> [Produkt] {
    do {
        if let search: String = try req.query.get(at: "szukane") {
            print(search)
            let produkty = try await Produkt.query(on: req.db)
                .filter(\$.nazwa, .custom("ILIKE"), "%\((search)%")
                .all()
            return produkty
        } else {
            return try await Produkt.query(on: req.db).all()
        }
    } catch {
        throw Abort(.badRequest, reason: "Error: \(error)")
    }
}
```

Rysunek 26 Fragment kodu pliku ProduktController.swift



Rysunek 27 Przykład działania filtracji w aplikacji webowej

#### 4.2.4 Implementacja wybranych funkcjonalności systemu

- Implementacja składania zamówienia po stronie serwera

```
func makeOrder(req: Request) async throws -> HTTPStatus {
    guard let clientIdString = req.session.data["clientId"],
          let clientId = UUID(clientIdString) else {
        throw Abort(.unauthorized)
    }

    struct ProductIdOnly: Content {
        let id: String
    }

    let productIdOnly = try req.content.decode(ProductIdOnly.self)
    guard let productId = UUID(productIdOnly.id) else {
        throw Abort(.badRequest)
    }

    guard let product = try await Produkt.query(on: req.db)
        .filter(\.$id == productId)
        .first() else {
        throw Abort(.notFound)
    }

    let cena = product.cena

    let newOrder = Zamowienie(klientId: clientId,
                             productid: productId,
                             cenazakupu: cena,
                             datazamowienia: Date())

    do {
        try await newOrder.save(on: req.db)
        return .ok
    } catch {
        throw Abort(.internalServerError, reason: "Błąd: \(error)")
    }
}
```

- Implementacja obsługi wysyłania obrazów do aplikacji klienta

```
app.get("images", ":imageName") { req -> EventLoopFuture<Response> in
    guard var imageName = req.parameters.get("imageName") else {
        throw Abort(.badRequest)
    }

    imageName = imageName.replacingOccurrences(of: ":", with: "-")
    let imagePath = app.directory.publicDirectory + "images/\(imageName).png"

    guard let imageData = FileManager.default.contents(atPath: imagePath) else {
        throw Abort(.notFound)
    }

    let response = Response()
    response.headers.contentType = ".png"
    response.body = .init(data: imageData)

    return req.eventLoop.makeSucceededFuture(response)
}
```

Rysunek 28 Fragment kodu pliku routes.swift

#### 4.2.5 Implementacja mechanizmów bezpieczeństwa

- System logowania

```
let passwordProtected = routes.grouped(User.authenticator())
passwordProtected.post("login") { req -> [String:String] in
    let user = try req.auth.require(User.self)

    req.session.data["userId"] = user.id?.uuidString
    req.session.data["clientId"] = user.klientId.uuidString
    req.session.data["uprawnienia"] = user.uprawnienia
    let responseData: [String:String] = [
        "id": user.id?.uuidString ?? "",
        "uprawnienia": user.uprawnienia
    ]
    return responseData
}
```

Rysunek 29 Fragment kodu pliku SessionController.swift

- Zasady CORS

```
let corsConfiguration = CORSMiddleware.Configuration(
    allowedOrigin: .custom("http://localhost:5173"),
    allowedMethods: [.GET, .POST, .PUT, .OPTIONS, .DELETE, .PATCH],
    allowedHeaders: [.accept, .authorization, .contentType, .origin, .xRequestedWith, .userAgent, .accessControlAllowOrigin],
    allowCredentials: true
)
let cors = CORSMiddleware(configuration: corsConfiguration)
app.middleware.use(cors, at: .beginning)
app.middleware.use(app.sessions.middleware)
app.sessions.use(.memory)
```

Rysunek 30 Fragment kodu pliku configure.swift

- Szyfrowanie hasła

```
let user = try User(
    login: create.login,
    haslo: Bcrypt.hash(create.haslo),
    email: create.email
)
```

Rysunek 31 Fragment kodu pliku SessionController.swift

- Wysyłania żądania z aplikacji klienta do serwera

```
async function postDataWithAuth(username: string, password: string) {
  let url = 'http://localhost:8080/Login'
  const headers = new Headers();
  headers.append('Authorization', `Basic ${btoa(`${username}:${password}`)})';

  const requestOptions = {
    method: 'POST',
    credentials: 'include',
    headers,
    body: JSON.stringify('')
  };

  try {
    const response = await fetch(url, requestOptions);
    if (!response.ok) {
      throw new Error('Odpowiedź nie ok');
    }
    return await response.json();
  } catch (error) {
    console.error('Error:', error);
    throw error;
  }
}
```

Rysunek 32 Fragment kodu pliku login/+page.svelte

## 5. Testowanie systemu

### 5.1. Instalacja i konfigurowanie systemu

- Instalacja aplikacji webowej

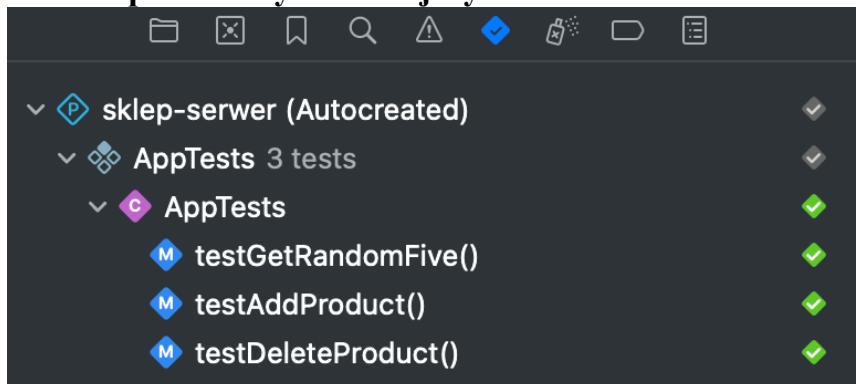
Należy pobrać z repozytorium projekt <https://github.com/SzymonSergiusz/sklep-aplikacja> i następnie w folderze sklep-aplikacja z poziomu terminala za pomocą narzędzia npm:

- npm install
- npm run dev --port 5173

- Instalacja serwera

Należy pobrać z repozytorium projekt <https://github.com/SzymonSergiusz/sklep-serwer> i następnie uruchomić w środowisku Xcode lub zgodnie z instrukcją wystawić serwer w Docker <https://docs.vapor.codes/deploy/docker/>

### 5.2. Testowanie opracowanych funkcji systemu



Rysunek 33 Wyniki testów jednostkowych

#### 5.2.1 Testowanie funkcji dodawania produktu

```
func testAddProduct() async throws {
    let app = Application(.testing)
    defer { app.shutdown() }
    try await configure(app)
    let ile = try await Produkt.query(on: app.db).count()

    let produkt = Produkt(nazwa: "Test", opis: "opis testu", cena: 10, wydawcaid:
        UUID("48ACAA00-9F76-45E2-8A83-FCE3F442D54F")!, gatunekid:
        UUID("B09746DF-132E-4DBE-901A-94E90A246D73")!, platformaid:
        UUID("51E25813-0CCE-4705-8D3D-DD5FFB8874D3")!, datawydania: Date.now, ilosc: 1)
    try await produkt.save(on: app.db)

    let ilePlusJeden = try await Produkt.query(on: app.db).count()
    XCTAssertEqual(ile+1, ilePlusJeden)
}
```

Rysunek 34 Fragment kodu AppTests.swift

### **5.2.2 Testowanie funkcji usuwania produktu**

```
func testDeleteProduct() async throws {
    let app = Application(.testing)
    defer { app.shutdown() }
    try await configure(app)
    let ile = try await Produkt.query(on: app.db).count()
    let produkt = Produkt(nazwa: "Test", opis: "opis testu", cena: 10, wydawcaid:
        UUID("48ACAA0-9F76-45E2-8A83-FCE3F442D54F")!, gatunekid:
        UUID("B09746DF-132E-4DBE-901A-94E90A246D73")!, platformaid:
        UUID("51E25813-0CCE-4705-8D3D-DD5FFB8874D3")!, datawydania: Date.now, ilosc: 1)
    try await Produkt.query(on: app.db)
        .filter(\.$nazwa == produkt.nazwa)
        .delete()

    let ileMinusJeden = try await Produkt.query(on: app.db).count()
    XCTAssertEqual(ile, ileMinusJeden+1)
}
```

Rysunek 35 Fragment kodu AppTests.swift

### **5.3. Testowanie mechanizmów bezpieczeństwa**

Testowanie mechanizmów bezpieczeństwa odbyło się przy użyciu narzędzia Postman [9]. Testowano żądania na ścieżki, które wymagają poprawnego zalogowania.

### **5.3.1 Test żądania POST bez wcześniejszego logowania**

Żądanie zostało odrzucone (502 Bad Gateway).

HTTP test-serwer-sklep / test sesji / **dodajProdukt**

Save

POST localhost:8080/addProduct **Send**

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON**

1 {  
2 ... "wydawcaid": "48ACAC0-9F76-45E2-8A83-FCE3F442D54F",  
3 ... "opis": "lorem ipsum dolores",  
4 ... "platformaid": "51E25813-0CCE-4705-803D-DD5FFB8874D3",  
5 ... "nazwa": "TEST",  
6 ... "gatunekid": "B09746DF-132E-40BE-901A-94E90A246D73",  
7 ... "cena": 100000,  
8 ... "ilosc": 20,  
9 ... "datawydania": "2019-01-25T00:00:00Z"  
10 }  
11

Body Cookies (1) Headers (4) Test Results

Status: 502 Bad Gateway Time: 46 ms Size: 241 B

Pretty Raw Preview Visualize Text

1

Rysunek 36 Zrzut ekranu wyniku testu POST /addProduct bez zalogowania

### 5.3.2 Test żądania POST po zalogowaniu jako klient

Żądanie zostało odrzucone (502 Bad Gateway).

The screenshot shows a Postman test environment. The URL is `http://test-serwer-sklep/test sesji/dodajProdukt`. The method is `POST` to `localhost:8080/addProduct`. The `Body` tab is selected, containing the following JSON payload:

```
1 {
2     "wydawcaid": "48ACAA0-9F76-45E2-8A83-FCE3F442D54F",
3     "opis": "lorem ipsum dolores",
4     "platformaid": "51E25813-0CCE-4705-8D3D-DD5FFB8874D3",
5     "nazwa": "TEST",
6     "gatunekid": "B09746DF-132E-4DBE-901A-94E90A246D73",
7     "cena": 100000,
8     "ilosc": 20,
9     "datawydania": "2019-01-25T00:00:00Z"
10 }
11
```

The `Cookies` tab shows a single cookie: `vapor-session` with value `o7HWMHYFd...`. The `Test Results` tab indicates a failure with status `502 Bad Gateway`, time `11 ms`, and size `241 B`.

Rysunek 37 Zrzut ekranu wyniku testu POST /addProduct po zalogowaniu jako klient

### 5.3.3 Test żądania POST po zalogowaniu jako pracownik

Żądanie zostało wykonane (200 OK).

The screenshot shows a Postman test environment. The URL is `http://test-serwer-sklep/test sesji/dodajProdukt`. The method is `POST` to `localhost:8080/addProduct`. The `Body` tab is selected, containing the same JSON payload as in the previous screenshot.

The `Headers` tab shows the following response headers:

Key	Value
content-length	0
set-cookie	vapor-session=o7HWMHYFd...; Expires=...
connection	keep-alive
date	Thu, 01 Feb 2024 16:04:32 GMT

The `Test Results` tab indicates success with status `200 OK`, time `48 ms`, and size `232 B`.

Rysunek 38 Zrzut ekranu wyniku testu POST /addProduct po zalogowaniu jako pracownik

#### **5.4. Wnioski z testów**

Testy wykonane dla wybranych funkcjonalności są podstawowe i wynika z nich, że działają poprawnie. Dodawanie i usuwanie z poziomu serwera do bazy danych działa poprawnie, a wykonywanie żądań wymagających uprawnień jest odpowiednio zabezpieczone.

Gdyby skupić więcej uwagi na testy można byłoby postarać się szczegółowo pokryć testami jednostkowymi większość funkcji systemowych oraz sprawdzić poprawność działania każdego możliwego żądania.

## 6. Podsumowanie

Założenia początkowe projektu zostały spełnione, a główne funkcjonalności przedstawiające różne zapytania do bazy danych zostały zaimplementowane. Użytkownik z poziomu aplikacji jest w stanie stworzyć konto, zalogować się na to konto, przeglądać oraz wyszukiwać produkty, dokonać zakupu i przejrzeć swoją historię zamówień.

Aplikacja webowa zapisuje pliki cookies co pozwala na ukrycie funkcji przed użytkownikiem o nieodpowiednich uprawnieniach (użytkownik „klient” po zalogowaniu nie widzi podstrony dodawania produktu, którą widzi zalogowany „pracownik”). Nie są to wystarczające zabezpieczenia, więc serwer po swojej stronie również sprawdza uprawnienia w plikach sesji. Do zabezpieczonych ścieżek bez odpowiednich uprawnień nie ma dostępu. Hasła są szyfrowane. Logowanie przebiega metodą *Basic Authentication*, która dodatkowo zabezpiecza komunikacje między aplikacją klienta, a serwerem.

Aby usprawnić system niektóre dane są przetwarzane po stronie bazy danych przy użyciu PL/pgSQL.

Walidacja danych przy tworzeniu konta odbywa się po stronie aplikacji klienta, a następnie po stronie serwera, aby zapewnić poprawność dodawanych danych do bazy danych.

Technologie zastosowane w projekcie zaskoczyły mnie z jaką łatwością można wykonać aplikację o tylu funkcjonalnościach. Framework Svelte [2], który służył do napisania aplikacji webowej, jest przejrzysty i prostszy od wcześniej używanych przeze mnie innych frameworków JavaScript. Tworzenie serwera w Vapor okazało się łatwe dzięki minimalizmowi tej technologii, bibliotece ORM Fluent oraz przejrzystej dokumentacji.

Tworzenie bazy danych w PostgreSQL było satysfakcjonujące i ta technologia będzie moim pierwszym wyborem przy tworzeniu kolejnych aplikacji w przyszłości.

Narzędzia z których korzystałem spełniły moje wymagania. Program do obsługi bazy danych DataGrip bardzo ułatwił mi pracę przy implementacji modelu fizycznego oraz późniejsze przeglądanie danych. Program Postman ułatwił mi testowanie wysyłania żądań http.

Nie mam zastrzeżeń do technologii oraz narzędzi, które wybrałem do wykonania projektu i ponownie wybrałbym je przy pisaniu aplikacji o podobnych zastosowaniach.

## 7. Literatura

- [1] <https://vapor.codes>
- [2] <https://svelte.dev>
- [3] <https://kit.svelte.dev>
- [4] <https://www.postgresql.org/docs/current/index.html>
- [5] <https://www.w3schools.com/postgresql/index.php>
- [6] <https://docs.vapor.codes/fluent/overview/>
- [7] [https://en.wikipedia.org/wiki/Basic\\_access\\_authentication](https://en.wikipedia.org/wiki/Basic_access_authentication)
- [8] <https://www.skeleton.dev>
- [9] <https://www.postman.com>