

<b>AI1</b>	<b>Dokumentacja projektu</b>
<b>Autor</b>	Justyna Toczek, 125170
<b>Kierunek, rok</b>	Informatyka, II rok, st. stacjonarne (3,5-l)
<b>Temat projektu</b>	<i>Platforma do zamawiania jedzenia do domu</i>

## 1. Wstęp

Tematem projektu jest aplikacja do zamawiania jedzenia do domu online. Projekt ma na celu w sposób prosty i przyjazny przedstawić użytkownikom ofertę strony oraz zachęcić ich do skorzystania z oferowanych usług. Użytkownik może wybrać spośród miast i restauracji oraz zamówić wybrane przez niego produkty i dania. Kolejnym założeniem projektu jest możliwość zarządzania przez administratora złożonymi przez użytkownika zamówieniami oraz ofertą, tj. miastami, restauracjami, produktami.

## 2. Narzędzia i technologie

Aplikacja została stworzona przy pomocy poniżej wymienionych narzędzi i technologii.

### 2.1. Języki programowania:

- PHP
  - Dokumentacja: <https://www.php.net/docs.php>
  - Licencja: <https://www.php.net/license/index.php>
  - Dostęp darmowy
  - Wersja 8.2.12
- JavaScript
  - Dokumentacja: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
  - Dostęp darmowy

### 2.2. Język znaczników:

- HTML
  - Dokumentacja: <https://developer.mozilla.org/en-US/docs/Web/HTML>
  - Dostęp darmowy

### 2.3. Język arkuszy styli:

- CSS
  - Dokumentacja: <https://developer.mozilla.org/en-US/docs/Web/CSS>
  - Dostęp darmowy

### 2.4. Baza danych:

- MySql
  - Dokumentacja: <https://dev.mysql.com/doc/>
  - Licencja: <https://www.mysql.com/about/legal/licensing/oem/>
  - Dostęp darmowy

### 2.5. Biblioteki

- Bootstrap
  - Dokumentacja: <https://getbootstrap.com/>
  - Licencja: <https://getbootstrap.com/docs/5.3/about/license/>
  - Dostęp darmowy
  - Wersja 5.3.3

## 2.6. Frameworki

- Laravel
  - Dokumentacja: <https://laravel.com/docs/11.x/readme>
  - Licencja: <https://github.com/translation/laravel/blob/master/LICENSE>
  - Dostęp darmowy
  - Wersja 11.7.0

## 2.7. Środowiska programistyczne

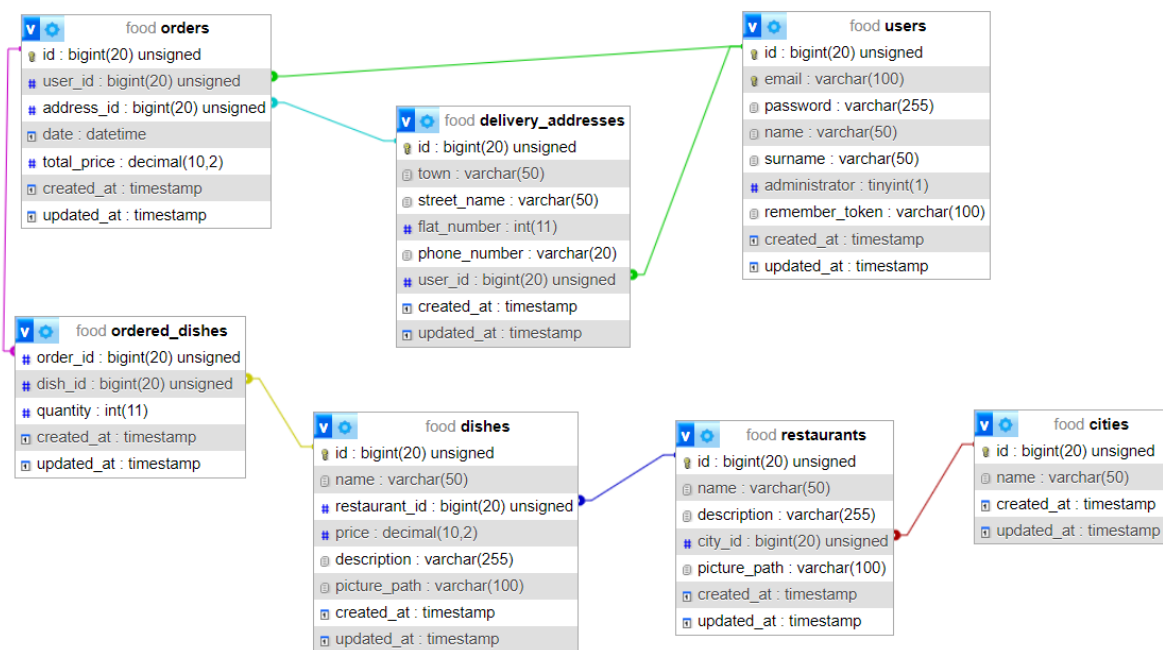
- Visual Studio Code
  - Dokumentacja: <https://code.visualstudio.com/docs>
  - Licencja: <https://code.visualstudio.com/License/>
  - Dostęp darmowy

## 2.8. Narzędzia

- XAMPP
  - Dokumentacja: <https://linetc.readthedocs.io/pl/latest/tools/lamp/xampp.html>
  - Licencja: <https://www.apachefriends.org/pl/about.html>
  - Dostęp darmowy
- Composer
  - Dokumentacja: <https://getcomposer.org/doc/>
  - Licencja: <https://github.com/composer/composer/blob/main/LICENSE>
  - Dostęp darmowy

## 3. Baza danych

### 3.1. Diagram ERD



Zdjęcie 1. Diagram ERD bazy danych

### **3.2. Opis tabel**

Baza danych składa się z siedmiu powiązanych ze sobą tabel. Są to kolejno tabele:

- Users – przechowuje informacje o użytkownikach aplikacji, ich dane logowania oraz informację, czy użytkownik jest administratorem.
- Delivery\_addresses – zawiera dane o adresach dostaw użytkowników. Każdy użytkownik może mieć wiele adresów.
- Orders – tabela przechowująca informacje o złożonych przez użytkownika zamówieniach.
- Ordered\_dishes – tabela łącząca. Zawiera informacje o numerze zamówienia oraz zakupionym przez użytkownika produkcie i jego ilości.
- Dishes – przechowuje dane o ofercie wszystkich dań i produktów.
- Restaurants – przechowuje dane o świadczących swoje usługi restauracjach.
- Cities – tabela zawierająca miasta, w których występują oferujące się restauracje.

### **3.3. Opis powiązań pomiędzy tabelami**

Relacja pomiędzy tabelami „Users” i „Delivery\_address” to relacja jeden do wielu, ponieważ jeden użytkownik może mieć zapisanych wiele adresów dostaw.

Relacja pomiędzy tabelami „Users” i „Orders” to relacja jeden do wielu, ponieważ jeden użytkownik ma możliwość złożenia zamówień wiele razy.

Relacja pomiędzy tabelami „Delivery\_address” i „Orders” to relacja jeden do wielu, ponieważ jeden adres dostawy może odnosić się do wielu zamówień.

Relacja pomiędzy tabelami „Orders” i „Ordered\_dishes” to relacja jeden do wielu, ponieważ jedno zamówienie może składać się z wielu przypisanych w tym zamówieniu produktów.

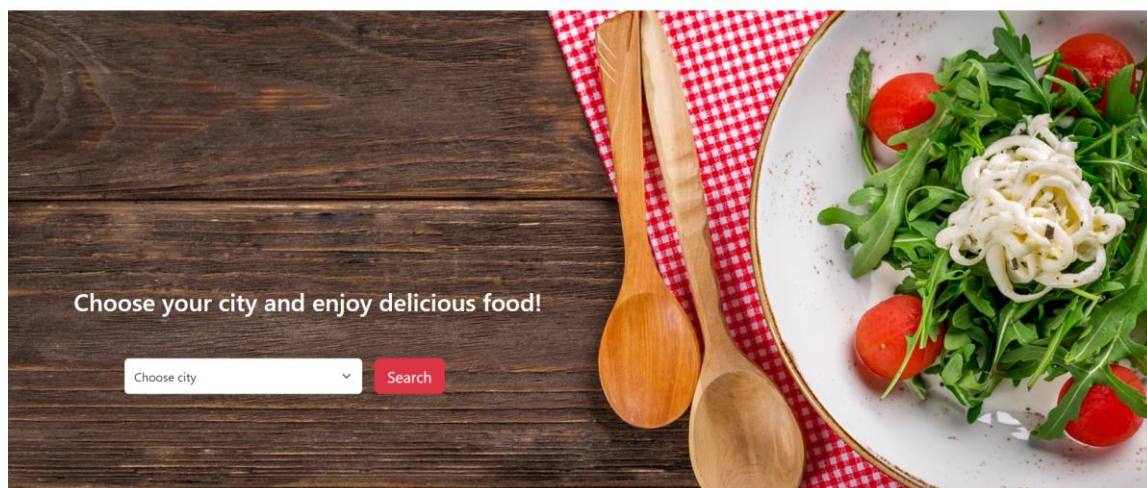
Relacja pomiędzy tabelami „Dishes” i „Ordered\_dishes” to relacja jeden do wielu, ponieważ jeden produkt/danie może być wiele razy zamówione.

Relacja pomiędzy tabelami „Restaurants” i „Dishes” to relacja jeden do wielu, ponieważ jedna restauracja może posiadać wiele dań i produktów.

Relacja pomiędzy tabelami „Cities” i „Restaurants” to relacja jeden do wielu, ponieważ w jednym mieście może występować wiele restauracji.

## **4. GUI**

### **4.1. Widok strony głównej**



Zdjęcie 2. Strona główna aplikacji

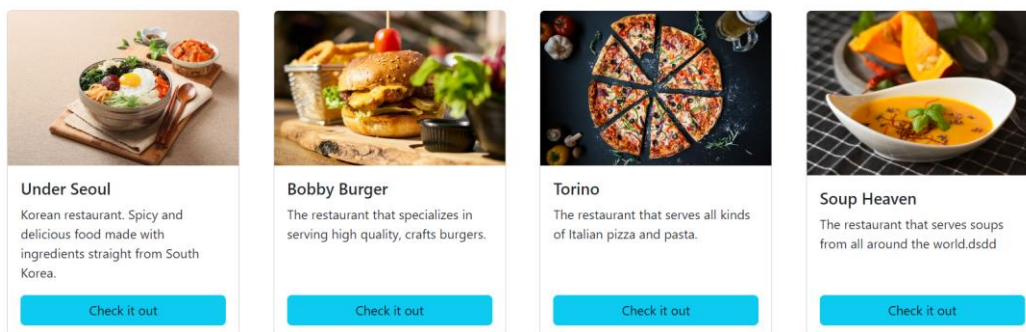
Strona główna aplikacji zrobiona jest w sposób przejrzysty dla użytkownika. Użytkownik może wybrać z menu dropdown miasto, z którego będzie przeglądał oferty restauracji.

W pasku nawigacyjnym przycisk „Home” przekierowuje właśnie do ww. strony głównej. Ikona wózka na zakupy przekierowuje do koszyka zalogowanego użytkownika.

Przyciski z prawej strony paska nawigacyjnego służą zalogowaniu lub rejestracji użytkowników.

#### 4.2. Widok wyboru restauracji

What would you like to eat in Rzeszow?



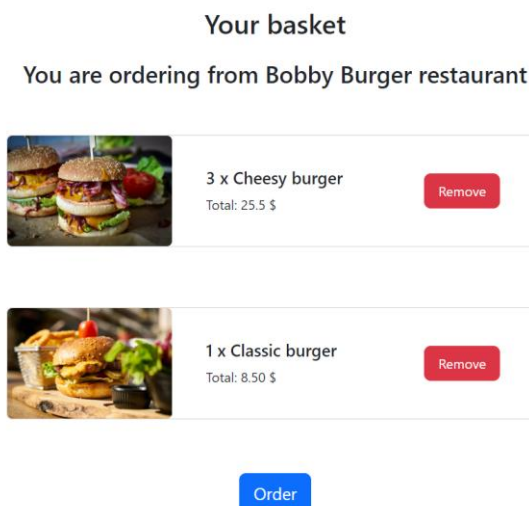
Zdjęcie 3. Widok wyboru restauracji

Po wybraniu miasta na stronie głównej, aplikacja przekierowuje użytkownika do widoku z wyborem restauracji. Każdą restaurację reprezentuje zdjęcie, jej nazwa oraz opis.

### 4.3. Widok koszyka

Home 

Ala, log out



Zdjęcie 4. Widok koszyka

Widok koszyka jest zapewniony tylko dla zalogowanych użytkowników. Zalogowani użytkownicy po przejściu do widoku koszyka mogą zobaczyć dodane przez nich dania, ilości oraz należną cenę.

## 5. Uruchomienie aplikacji

Aby uruchomić aplikację należy wykonać poniższe kroki.

1. Zainstaluj XAMPP.
2. W XAMPP uruchom usługę Apache oraz MySQL.
3. Uruchom plik projektu „start.bat” (utworzy on bazę danych, zainstaluje composer, utworzy storage link)
4. W terminalu, w katalogu projektu wprowadź komendy „php artisan migrate” oraz „php artisan db:seed”.
5. W terminalu w katalogu projektu wprowadź komendę „php artisan serve” oraz w przeglądarce przejdź pod adres <http://127.0.0.1:8000/>

## 6. Funkcjonalności aplikacji

### 6.1. Rejestracja

### Don't you have an account? Register here!

Name

Surname

Email

Password


Repeat password


Zdjęcie 5. Formularz rejestracji


### Don't you have an account? Register here!

Name

Surname

Email  
   
Email is not correct!

Password  
   
Password must be at least 8 characters long

Repeat password  
   
Passwords do not match!

Zdjęcie 6. Formularz rejestracji z wprowadzeniem błędnych danych

W formularzu rejestracji użytkownik wprowadza imię, nazwisko, email oraz hasło, wraz z jego powtórzeniem. Występuje walidacja po stronie frontendu i backendu: po wprowadzeniu błędnych danych, formularz nie zostanie przesłany, a użytkownik zostanie poinformowany odpowiednimi komunikatami które pola formularza należy poprawić.

```

public function register(RegisterRequest $request)
{
    $user = User::create([
        'name' => $request->name,
        'surname' => $request->surname,
        'email' => $request->email,
        'password' => Hash::make($request->password),
        'administrator' => false,
    ]);

    Auth::login($user);
    return redirect()->route('index');
}

```

Zdjęcie 7. Funkcja odpowiedzialna za rejestrację

```

public function rules(): array
{
    return [
        'name' => 'required|string|max:50',
        'surname' => 'required|string|max:50',
        'email' => 'required|string|email|max:100|unique:users',
        'password' => 'required|string|min:8|confirmed'
    ];
}

```

Zdjęcie 8. Rules w pliku RegisterRequest.php

```

<div class="form-group mb-2">
    <label for="name" class="form-label">Name</label>
    <input id="name" name="name" type="text" class="form-control @if ($errors->first('name')) is-invalid @endif" value="{{ old('name') }}">
    <div class="invalid-feedback">Name is required!</div>
</div>
<div class="form-group mb-2">
    <label for="surname" class="form-label">Surname</label>
    <input id="surname" name="surname" type="text" class="form-control @if ($errors->first('surname')) is-invalid @endif" value="{{ old('surname') }}">
    <div class="invalid-feedback">Surname is required!</div>
</div>
<div class="form-group mb-2">
    <label for="email" class="form-label">Email</label>
    <input id="email" name="email" type="text" class="form-control @if ($errors->first('email')) is-invalid @endif" value="{{ old('email') }}">
    <div class="invalid-feedback">Email is not correct!</div>
</div>
<div class="form-group mb-2">
    <label for="password" class="form-label">Password</label>
    <input id="password" name="password" type="password" class="form-control @if ($errors->first('password')) is-invalid @endif">
    <div class="invalid-feedback">Password must be at least 8 characters long</div>
</div>
<div class="form-group mb-2">
    <label for="password_confirmation" class="form-label">Repeat password</label>
    <input id="password_confirmation" name="password_confirmation" type="password" class="form-control @if ($errors->first('password')) is-invalid @endif">
    <div class="invalid-feedback">Passwords do not match!</div>
</div>

```

Zdjęcia 9 i 10. Walidacja po stronie frontendu, formularz w widoku


## 6.2. Logowanie

Przykładowe konta na które można się zalogować w aplikacji:

- Konto administratora:
  - Email: jan@wp.pl
  - Hasło: aaa



- Konto zwykłego użytkownika:
  - Email: ala@wp.pl
  - Hasło: aaa

Home 

Log in Register


Do you have an account already?  
Log in!

Email

Hasło

© Ordering food – 2024


Zdjęcie 11. Formularz logowania

Home 

Log in Register

The provided credentials do not match our records.

Do you have an account already?  
Log in!

Email  
 

Wrong email!

Hasło

© Ordering food – 2024

Zdjęcie 12. Formularz logowania z wprowadzeniem błędnych danych

W formularzu logowania użytkownik wprowadza email oraz hasło. Walidacja zrobiona jest po stronie frontu oraz backendu. Po wprowadzeniu błędnych danych oraz po kliknięciu w przycisk „submit” strona przeładuje się, a użytkownik zostanie poinformowany o nieprawidłowościach.

```

public function authenticate(Request $request)
{
    $credentials = $request->validate([
        'email' => ['required', 'email'],
        'password' => ['required'],
    ]);

    if (Auth::attempt($credentials)) {
        $request->session()->regenerate();
        return redirect()->route('index');
    }

    return back()->withErrors([
        'email' => 'The provided credentials do not match our records.',
    ])->onlyInput('email');
}

```

Zdjęcie 13. Funkcja odpowiedzialna za logowanie

```

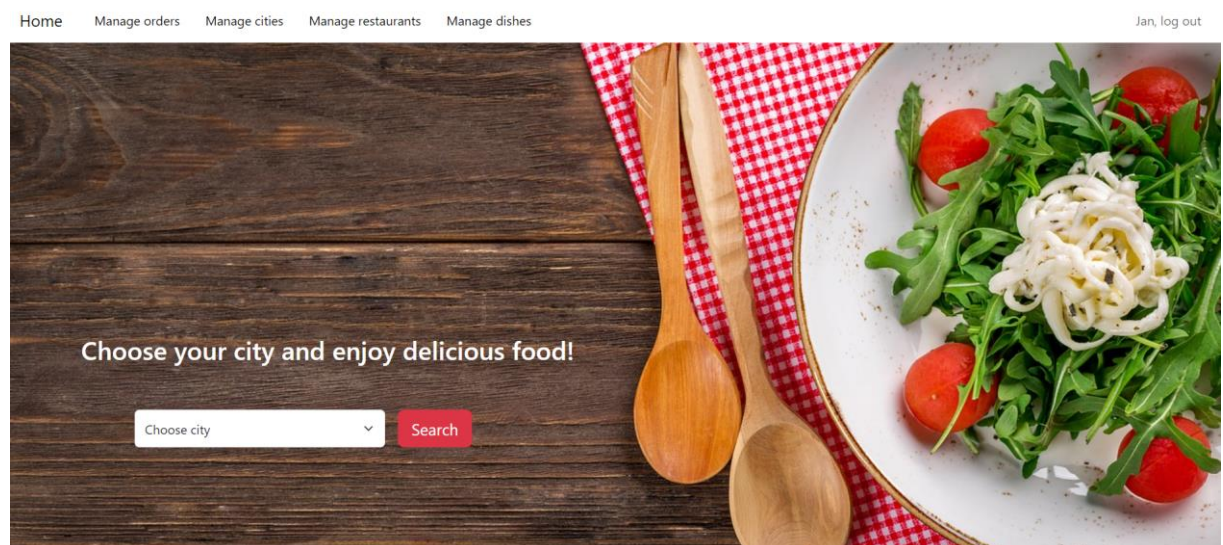
<form method="POST" action="{{ route('login.authenticate') }}" class="needs-validation" novalidate>
    @csrf
    <div class="form-group mb-2">
        <label for="email" class="form-label">Email</label>
        <input id="email" name="email" type="text" class="form-control @if ($errors->first('email')) is-invalid @endif value="{{ old('email') }}">
        <div class="invalid-feedback">Wrong email</div>
    </div>
    <div class="form-group mb-2">
        <label for="password" class="form-label">Hasło</label>
        <input id="password" name="password" type="password" class="form-control @if ($errors->first('password')) is-invalid @endif>
        <div class="invalid-feedback">Wrong password!</div>
    </div>
    <div class="text-center mt-5 mb-4">
        <input class="btn btn-primary btn-lg" type="submit" value="Submit">
    </div>
</form>

```

Zdjęcie 14. Walidacja po stronie frontentu, formularz w widoku

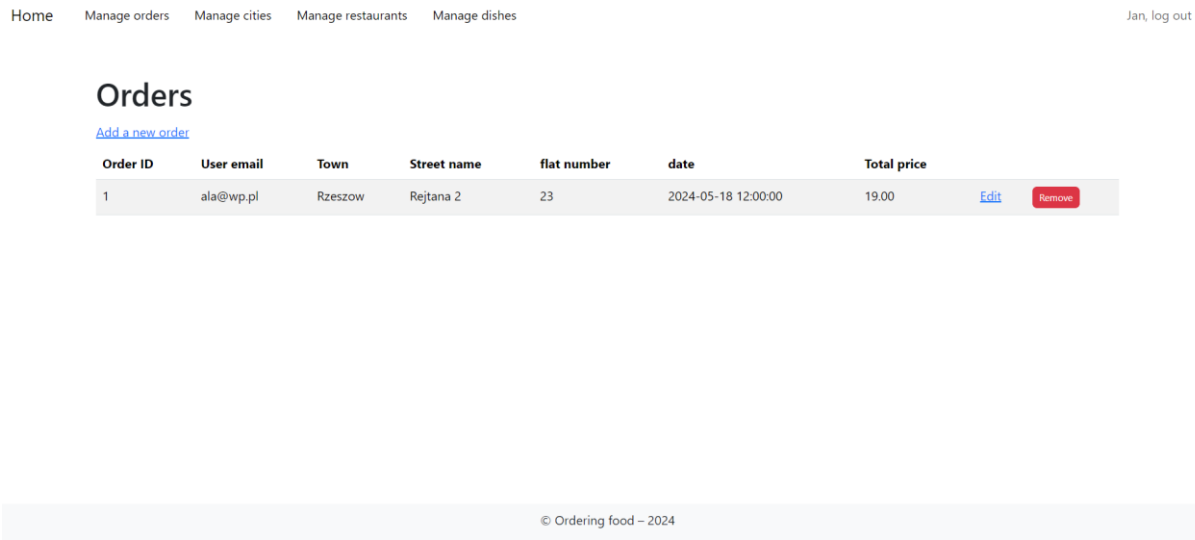
### 6.3. CRUD przeprowadzany przez administratora

Administrator w pasku nawigacyjnym ma możliwość wyboru zarządzania: zamówieniami, miastami, restauracjami oraz daniami.



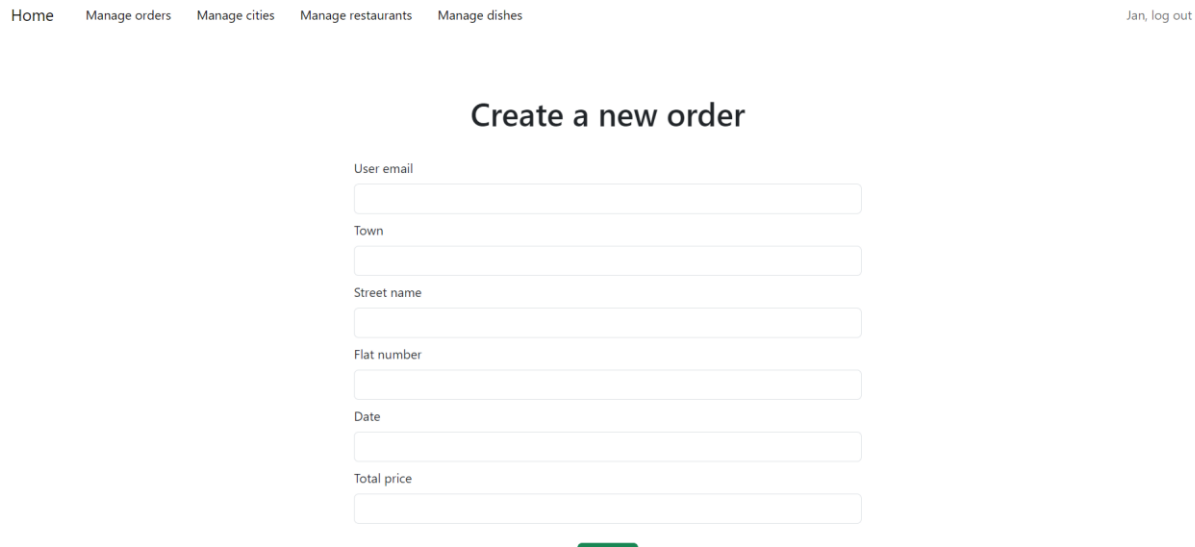
Zdjęcie 15. Widok aplikacji dla administratora

### 6.3.1. Zarządzanie zamówieniami



Zdjęcie 16. Panel zarządzania zamówieniami

Administrator, w panelu zarządzania zamówieniami ma możliwość odczytu dotychczasowych zamówień, ich edycji, usunięcia oraz dodania nowych zamówień.



Zdjęcie 17. Panel dodawania nowego zamówienia

Występuje walidacja po stronie frontendu oraz backendu. Jeśli jakieś pole w formularzu będzie miało niepoprawny format lub wprowadzony adres dostawy/adres użytkownika nie będzie istnieć, użytkownik zostanie o tym poinformowany, a formularz nie prześle się.

```

public function storeFromAdmin(UpdateOrderRequest $request)
{
    $validated = $request->validated();

    $user = User::where('email', $validated['email'])->first();
    if (!$user) {
        return redirect()->back()->withErrors(['email' => 'User not found'])->withInput();
    }

    $address = Delivery_address::where('town', $validated['town'])
        ->where('street_name', $validated['street_name'])
        ->where('flat_number', $validated['flat_number'])
        ->first();
    if (!$address) {
        return redirect()->back()->withErrors(['address' => 'Address not found'])->withInput();
    }

    $order = new Order();
    $order->user_id = $user->id;
    $order->address_id = $address->id;
    $order->date = $validated['date'];
    $order->total_price = $validated['total_price'];
    $order->save();

    return redirect()->route('manage.orders')->with('success', 'Order created successfully.');
```

Zdjęcie 18. Funkcja odpowiedzialna za dodanie nowego zamówienia

```

<form method="POST" action="{{ route('order.store') }}" class="needs-validation" novalidate>
    @csrf
    <div class="form-group mb-2">
        <label for="email" class="form-label">User email</label>
        <input id="email" name="email" type="text" class="form-control @if ($errors->first('email')) is-invalid @endif value="{{ old('email') }}">
        <div class="invalid-feedback">Wrong user email!</div>
    </div>
    <div class="form-group mb-2">
        <label for="town" class="form-label">Town</label>
        <input id="town" name="town" type="text" class="form-control @if ($errors->first('town')) is-invalid @endif value="{{ old('town') }}">
        <div class="invalid-feedback">Wrong town name!</div>
    </div>
    <div class="form-group mb-2">
        <label for="street_name" class="form-label">Street name</label>
        <input id="street_name" name="street_name" type="text" class="form-control @if ($errors->first('street_name')) is-invalid @endif value="{{ old('street_name') }}">
        <div class="invalid-feedback">Wrong street name!</div>
    </div>
    <div class="form-group mb-2">
        <label for="flat_number" class="form-label">Flat number</label>
        <input id="flat_number" name="flat_number" type="text" class="form-control @if ($errors->first('flat_number')) is-invalid @endif value="{{ old('flat_number') }}">
        <div class="invalid-feedback">Wrong flat number!</div>
    </div>
    <div class="form-group mb-2">
        <label for="date" class="form-label">Date</label>
        <input id="date" name="date" type="text" class="form-control @if ($errors->first('date')) is-invalid @endif value="{{ old('date') }}">
        <div class="invalid-feedback">Wrong date!</div>
    </div>
    <div class="form-group mb-2">
        <label for="total_price" class="form-label">Total price</label>
        <input id="total_price" name="total_price" type="text" class="form-control @if ($errors->first('total_price')) is-invalid @endif value="{{ old('total_price') }}">
        <div class="invalid-feedback">Wrong price!</div>
    </div>
    <div class="text-center mt-4 mb-4">
        <input class="btn btn-success" type="submit" value="Submit">
    </div>
</form>
```

Zdjęcia 19. i 20. Walidacja po stronie frontendu, formularz w widoku

## Edit the order

User email	<input type="text" value="ala@wp.pl"/>
Town	<input type="text" value="Rzeszow"/>
Street name	<input type="text" value="Rejtana 2"/>
Flat number	<input type="text" value="23"/>
Date	<input type="text" value="2024-05-18 12:00:00"/>
Total price	<input type="text" value="19.00"/>

Zdjęcie 21. Panel edycji istniejącego zamówienia

Walidacja w panelu edycji jest zrobiona podobnie jak w panelu dodawania nowego zamówienia.

```
public function update(UpdateOrderRequest $request, $id)
{
    $order = Order::findOrFail($id);
    $validated = $request->validated();

    $user = User::where('email', $validated['email'])->first();
    if (!$user) {
        return redirect()->back()->withErrors(['email' => 'User not found'])->withInput();
    }

    $address = Delivery_address::where('town', $validated['town'])
        ->where('street_name', $validated['street_name'])
        ->where('flat_number', $validated['flat_number'])
        ->first();
    if (!$address) {
        return redirect()->back()->withErrors(['address' => 'Address not found'])->withInput();
    }

    $order->user_id = $user->id;
    $order->address_id = $address->id;
    $order->date = $validated['date'];
    $order->total_price = $validated['total_price'];
    $order->save();

    return redirect()->route('manage.orders')->with('success', 'Order updated successfully.');
```

Zdjęcie 22. Funkcja odpowiedzialna za edycję istniejącego zamówienia.

```

<form method="POST" action="{{ route('order.update', $order->id) }}" class="needs-validation novalidate"
@csrf
@method('PUT')
<div class="form-group mb-2">
  <label for="email" class="form-label">User email</label>
  <input id="email" name="email" type="text" class="form-control @if ($errors->first('email')) is-invalid @endif" value="{{ $order->users->email }}">
  <div class="invalid-feedback">Wrong user email!</div>
</div>
<div class="form-group mb-2">
  <label for="town" class="form-label">Town</label>
  <input id="town" name="town" type="text" class="form-control @if ($errors->first('town')) is-invalid @endif" value="{{ $order->delivery_addresses->town }}">
  <div class="invalid-feedback">Wrong town!</div>
</div>
<div class="form-group mb-2">
  <label for="street_name" class="form-label">Street name</label>
  <input id="street_name" name="street_name" type="text" class="form-control @if ($errors->first('street_name')) is-invalid @endif" value="{{ $order->delivery_addresses->street_name }}">
  <div class="invalid-feedback">Wrong street name!</div>
</div>
<div class="form-group mb-2">
  <label for="flat_number" class="form-label">Flat number</label>
  <input id="flat_number" name="flat_number" type="text" class="form-control @if ($errors->first('flat_number')) is-invalid @endif" value="{{ $order->delivery_addresses->flat_number }}">
  <div class="invalid-feedback">Wrong flat number!</div>
</div>
<div class="form-group mb-2">
  <label for="date" class="form-label">Date</label>
  <input id="date" name="date" type="text" class="form-control @if ($errors->first('date')) is-invalid @endif" value="{{ $order->date }}">
  <div class="invalid-feedback">Wrong date!</div>
</div>
<div class="form-group mb-2">
  <label for="total_price" class="form-label">Total price</label>
  <input id="total_price" name="total_price" type="text" class="form-control @if ($errors->first('total_price')) is-invalid @endif" value="{{ $order->total_price }}">
  <div class="invalid-feedback">Wrong price!</div>
</div>
<div class="text-center mt-4 mb-4">
  <input class="btn btn-success" type="submit" value="Submit">
</div>
</form>

```

Zdjęcie 23. Walidacja po stronie frontendu, formularz w widoku

Zarządzanie miastami, restauracjami oraz daniami zostało zrealizowane w sposób analogiczny do zarządzania zamówieniami.