

AI1	Dokumentacja projektu
Autor	Michał Dyjak, 125114
Kierunek, rok	Informatyka, II rok, st. stacjonarne (3,5-l)
Temat projektu	<i>Aplikacja do zamawiania jedzenia z dostawą online.</i>

Tytuł: Royal repast

SPIS TREŚCI:

Wstęp	5
Cel aplikacji.....	5
Komponenty i funkcjonalności	6
Użyte technologie	7
Języki programowania	7
Frameworki	8
Bazy danych	8
Biblioteki i narzędzia front-endowe.....	8
Narzędzia i środowiska programistyczne	8
Systemy kontroli wersji.....	8
Inne narzędzia	8
Baza danych	9
Tabele	9
Powiązania Między Tabelami	10
GUI	10
Strona Główna	10
Menu Restauracji	11
Koszyk.....	12
Formularz Zamówienia	12

Potwierdzenie Zamówienia	12
Uruchomienie Aplikacji	12
Zainstalowane środowisko PHP	12
Baza Danych.....	12
Środowisko Wykonawcze	12
Instalacja Zależności.....	13
Konfiguracja pliku .env	13
Migracje Bazy Danych	13
Seedowanie Bazy Danych (Opcjonalne).....	13
Uruchomienie Serwera Deweloperskiego	13
Testowanie Aplikacji	13
Skrypty uruchamiające	13
Skrypt start.bat	14
Skrypt start.sh	14
Skrypt DBrefresh.bat i DBrefresh.sh	14
Funkcjonalności aplikacji	15
Logowanie.....	15
Ekran Logowania.....	15
Rejestracja	16
Ekran Rejestracji	16
Weryfikacja Działania	16
Panel CRUD dla administratora	17
Wyświetlanie tabeli użytkowników	17
Formularz edycji użytkownika	18
Formularz usunięcia użytkownika	18
Formularz dodawania nowego użytkownika	19
Aktualizacja użytkownika.....	19
Usuwanie użytkownika	19
Dodawanie nowego użytkownika	19
Przeglądanie ogólnodostępnych zasobów	20
Pobieranie kategorii restauracji.....	20
Sortowanie restauracji	20

Filtrowanie według kategorii	20
Filtrowanie według darmowej dostawy	21
Wyszukiwanie restauracji	21
Paginacja wyników	21
Widok przeglądania restauracji	21
Przegląd pojedynczej restauracji	21
Pobieranie szczegółów posiłku	22
Posiłki z tej samej restauracji	22
Posiłki z tej samej kategorii <code>\$mealsFromSameCategory = \$meal->category->meals()->where('id', '!=', \$meal->id)->inRandomOrder()->take(4)->get();</code>	22
Zarządzanie swoimi danymi przez użytkownika	22
Wyświetlanie profilu użytkownika	22
Edycja profilu użytkownika	23
Aktualizacja profilu użytkownika	23
Usuwanie konta użytkownika	23
Aktualizacja adresu użytkownika	24
Zarządzanie swoimi zasobami przez użytkownika	24
Dodanie posiłku do koszyka	24
Wyświetlenie koszyka	25
Aktualizacja koszyka	25
Usunięcie posiłku z koszyka	26
Checkout	26
Złożenie zamówienia	26
Rejestracja nowego użytkownika	27
Wyświetlanie widoku rejestracji	27
Obsługa żądania rejestracji	27
Dodatki	28
Przesuwanie się posiłków na stronie głównej	28
Funkcja <code>randomSpeed</code>	28
Funkcja <code>autoScroll</code>	28
Event <code>DOMContentLoaded</code>	29

Darmowa dostawa przy określonej cenie zamówienia	30
Konfiguracja progu darmowej dostawy	30
Funkcja checkout w CartController	30
Pobranie zawartości koszyka:	31
Inicjalizacja całkowitego kosztu zamówienia:	31
Iteracja przez elementy koszyka:	31
Sprawdzenie progu darmowej dostawy:	32
Dodanie kosztu dostawy do całkowitej wartości zamówienia:	32
Zwrócenie widoku z danymi o zamówieniu:	32
Wyświetlanie aktualnej ilości elementów w koszyku	33
Aktualizacja Metody viewCart w CartController	33
Aktualizacja AppServiceProvider	34

Wstęp

Aplikacja, nad którą pracujemy, to zaawansowany system do przeglądania, zarządzania i zamawiania jedzenia z lokalnych restauracji. Jest to platforma internetowa, która łączy użytkowników z różnorodnymi restauracjami, oferując bogaty wybór dań, łatwy dostęp do menu, oraz wygodne opcje zamawiania jedzenia online.

Główne cechy aplikacji obejmują:

- Interaktywne przeglądanie restauracji: Użytkownicy mogą przeglądać dostępne restauracje, zobaczyć ich oferty, a także sortować i filtrować wyniki według różnych kryteriów, takich jak kategorie kuchni, dostępność darmowej dostawy, czy oceny klientów.
- Szczegółowe strony restauracji: Każda restauracja ma dedykowaną stronę, na której prezentowane są szczegółowe informacje, w tym pełne menu, opisy potraw, zdjęcia dań, oraz recenzje klientów.
- Zarządzanie kontem użytkownika: Użytkownicy mają możliwość tworzenia kont, logowania się, edytowania swoich profili oraz przeglądania historii zamówień.
- Administrowanie: Aplikacja zawiera również panel administracyjny, który pozwala zarządcom restauracji i administratorom aplikacji na dodawanie nowych restauracji, zarządzanie menu, monitorowanie zamówień i analizowanie danych dotyczących użytkownika aplikacji.

Aplikacja jest zaprojektowana z myślą o łatwości użycia i responsywności, dzięki czemu użytkownicy mogą korzystać z niej na różnych urządzeniach, takich jak komputery stacjonarne, laptopy, tablety i smartfony. Głównym celem aplikacji jest uproszczenie procesu zamawiania jedzenia online, zwiększenie dostępności lokalnych restauracji dla szerokiej bazy klientów oraz zapewnienie restauratorom narzędzi do skutecznego zarządzania swoim biznesem w środowisku cyfrowym.

Cel aplikacji

Aplikacja ma za zadanie usprawnić i zautomatyzować proces zamawiania jedzenia online, łącząc użytkowników z lokalnymi restauracjami. Główne cele aplikacji obejmują:

a. Ułatwienie przeglądania restauracji i menu

- Katalog restauracji: Aplikacja prezentuje użytkownikom obszerny katalog lokalnych restauracji. Użytkownicy mogą przeglądać restauracje według różnych kryteriów, takich jak typ kuchni, lokalizacja, oceny klientów czy promocje.
- Szczegółowe strony restauracji: Każda restauracja ma własną stronę z pełnym menu, zdjęciami potraw, opisami dań, godzinami otwarcia oraz informacjami o dostawie.
- Filtry i sortowanie: Użytkownicy mogą korzystać z różnych filtrów i opcji sortowania, aby szybko znaleźć restauracje, które spełniają ich oczekiwania, np. restauracje oferujące darmową dostawę lub specjalizujące się w wegańskich potrawach.

b. Ułatwienie procesu zamawiania jedzenia

- Koszyk zakupowy: Użytkownicy mogą dodawać wybrane dania do koszyka, modyfikować zamówienie, dodawać uwagi do potraw i ostatecznie złożyć zamówienie.
- Szybkie i bezpieczne płatności: Aplikacja wspiera różne metody płatności, w tym karty kredytowe/debetowe, systemy płatności online oraz opcję płatności przy odbiorze.
- Śledzenie zamówień: Użytkownicy mogą śledzić status swoich zamówień w czasie rzeczywistym, od momentu złożenia zamówienia, przez przygotowanie, aż po dostawę.

c. Personalizacja doświadczenia użytkownika

- Konta użytkowników: Użytkownicy mogą tworzyć konta, logować się, edytować swoje profile i przeglądać historię zamówień.
- Rekomendacje: Aplikacja analizuje preferencje i wcześniejsze zamówienia użytkowników, aby proponować im spersonalizowane rekomendacje.
- Recenzje i oceny: Użytkownicy mogą oceniać restauracje i zamówione potrawy, dzielić się swoimi opiniami oraz przeglądać recenzje innych klientów.

d. Ułatwienie zarządzania dla restauratorów

- Panel administracyjny: Restauratorzy mają dostęp do panelu administracyjnego, gdzie mogą zarządzać swoimi restauracjami, dodawać nowe potrawy do menu, ustawiać ceny, promować specjalne oferty i monitorować zamówienia.
- Analiza danych: Aplikacja oferuje narzędzia analityczne, które pomagają restauratorom zrozumieć trendy sprzedaży, preferencje klientów oraz efektywność różnych promocji i ofert specjalnych.
- Komunikacja z klientami: Restauratorzy mogą komunikować się z klientami za pośrednictwem aplikacji, odpowiadać na recenzje i uwagi, a także informować o nowych daniach i promocjach.

e. Zwiększenie dostępności lokalnych restauracji

- Promowanie restauracji: Aplikacja pomaga lokalnym restauracjom dotrzeć do większej liczby klientów, zarówno nowych, jak i stałych, poprzez widoczność w katalogu oraz specjalne funkcje promocyjne.
- Wspieranie lokalnej gospodarki: Ułatwiając dostęp do lokalnych restauracji i zachęcając użytkowników do zamawiania jedzenia online, aplikacja wspiera lokalną gospodarkę i pomaga restauracjom rozwijać się w środowisku cyfrowym.
- Głównym zadaniem aplikacji jest więc nie tylko ułatwienie procesu zamawiania jedzenia dla użytkowników, ale także wsparcie lokalnych restauracji w zarządzaniu ich działalnością i zwiększeniu ich widoczności oraz dostępności w cyfrowym świecie.

Komponenty i funkcjonalności

Aplikacja składa się z wielu komponentów i funkcjonalności, które razem tworzą spójny i zintegrowany system do zamawiania jedzenia online. Poniżej znajduje się szczegółowy opis głównych komponentów i funkcji:

a. Interfejs użytkownika (UI)

- Strona główna: Zawiera przegląd popularnych restauracji, promocji i nowości. Użytkownicy mogą szybko znaleźć interesujące ich oferty.
- Menu nawigacyjne: Umożliwia łatwy dostęp do różnych sekcji aplikacji, takich jak przegląd restauracji, koszyk, profil użytkownika, itp.
- Strony restauracji: Szczegółowe strony z informacjami o restauracjach, ich menu, recenzjami i ocenami klientów.
- Strony potraw: Szczegółowe opisy potraw, zdjęcia, składniki i możliwość dodania do koszyka.
- Koszyk: Przegląd wybranych potraw, możliwość edytowania zamówienia, dodawania uwag i finalizacji zamówienia.

b. Konta użytkowników i personalizacja

- Rejestracja i logowanie: Formularze rejestracji i logowania, możliwość odzyskiwania hasła.
- Profil użytkownika: Edycja danych osobowych, przegląd historii zamówień, preferencje użytkownika.

c. Przeglądanie i wyszukiwanie restauracji

- Lista restauracji: Przegląd dostępnych restauracji z możliwością filtrowania według różnych kryteriów (typ kuchni, oceny, lokalizacja).
- Sortowanie i filtry: Umożliwiają sortowanie restauracji według popularności, ocen, ceny itp. oraz filtrowanie wyników według specyficznych potrzeb (np. wegańskie, darmowa dostawa).

d. Zamawianie jedzenia

- Dodawanie do koszyka: Możliwość dodania wybranych potraw do koszyka, edytowania ilości i usuwania pozycji.
- Finalizacja zamówienia: Przegląd zamówienia, dodawanie uwag, wybór metody płatności, potwierdzenie zamówienia.
- Płatności: Integracja z różnymi systemami płatności online oraz opcja płatności przy odbiorze.

e. Panel administracyjny dla restauratorów

- Zarządzanie restauracją: Możliwość edycji danych restauracji, dodawania nowych potraw do menu, ustawiania cen, promocji i ofert specjalnych.
- Monitorowanie zamówień: Przegląd zamówień, statusów realizacji, historii zamówień.
- Analiza danych: Narzędzia analityczne do monitorowania trendów sprzedaży, preferencji klientów, skuteczności promocji.

f. Techniczne komponenty

- Bezpieczeństwo: Zabezpieczenie danych użytkowników, bezpieczne transakcje płatnicze, ochrona przed atakami.
- Wydajność: Optymalizacja działania aplikacji, szybkie ładowanie stron, skalowalność.

Każdy z tych komponentów i funkcji został zaprojektowany z myślą o użytkownikach, aby uczynić proces zamawiania jedzenia jak najbardziej intuicyjnym, szybkim i przyjemnym, jednocześnie wspierając restauratorów w efektywnym zarządzaniu ich działalnością.

Użyte technologie

Aplikacja do zamawiania jedzenia online wykorzystuje szeroki zakres nowoczesnych technologii i narzędzi, które razem tworzą solidne, wydajne i skalowalne rozwiązanie. Poniżej znajduje się lista głównych technologii używanych w projekcie, wraz z linkami do ich dokumentacji:

Języki programowania

- PHP: Główny język używany do tworzenia back-endu aplikacji.
- JavaScript: Używany głównie po stronie front-endu do interaktywnych elementów i logiki klienta.

Frameworki

- Laravel: PHP framework do budowy back-endu aplikacji, który oferuje bogaty zestaw narzędzi i funkcji.
- Vue.js: JavaScript framework używany do budowy dynamicznego i responsywnego interfejsu użytkownika.

Bazy danych

- MySQL: Relacyjna baza danych, w której przechowywane są wszystkie dane aplikacji, w tym użytkownicy, restauracje, zamówienia i recenzje.

Biblioteki i narzędzia front-endowe

- Bootstrap: Framework CSS używany do tworzenia responsywnych i estetycznych interfejsów użytkownika.
- Axios: Biblioteka JavaScript do wykonywania zapytań HTTP, używana do komunikacji z API.
- Sass: Preprocesor CSS, który pozwala na bardziej zorganizowane i czytelne pisanie stylów.

Narzędzia i środowiska programistyczne

- PHP Storm: Popularne środowisko programistyczne używane do pisania kodu.
- Composer: Menedżer pakietów dla PHP, używany do zarządzania zależnościami projektu.
- npm: Menedżer pakietów dla Node.js, używany do zarządzania zależnościami front-endu.

Systemy kontroli wersji

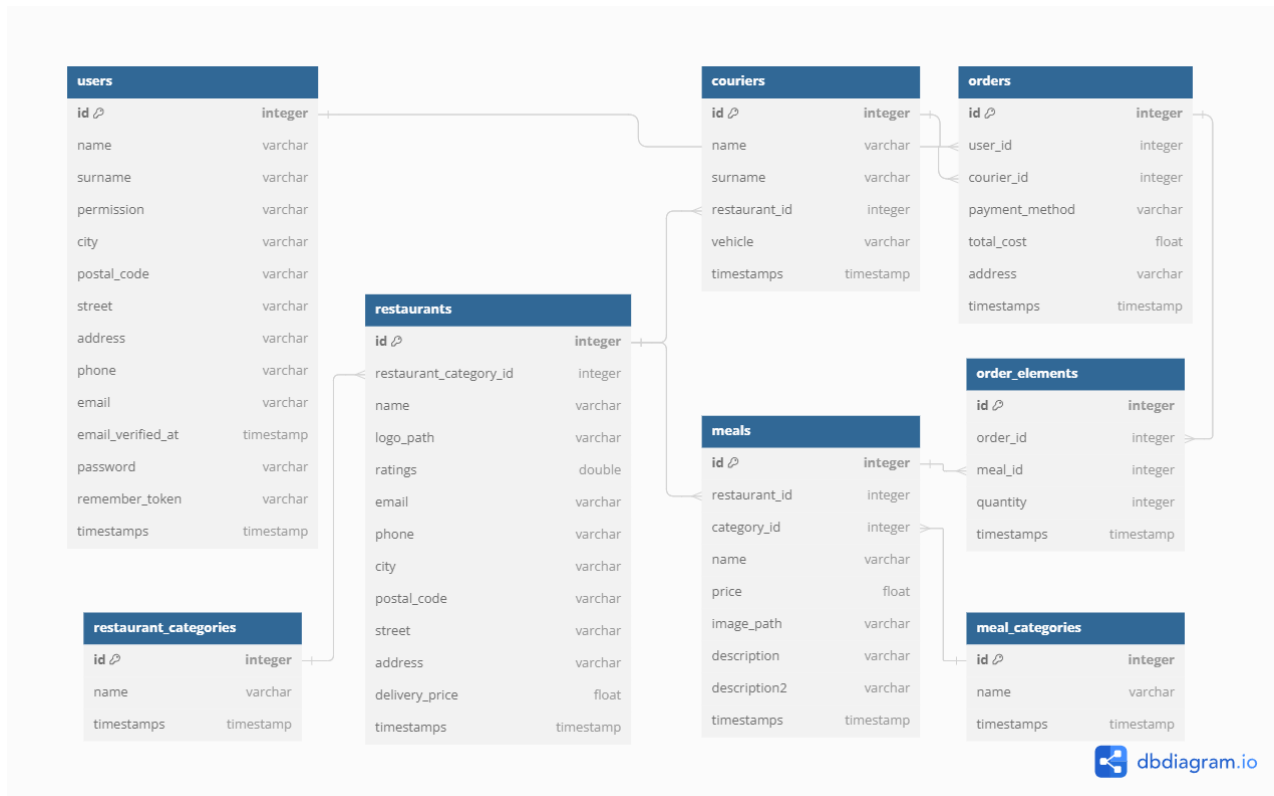
- Git: System kontroli wersji używany do zarządzania kodem źródłowym.
- GitHub: Platforma do hostowania repozytoriów Git, umożliwiająca współpracę nad kodem.
github.com/Dyjak/RoyalRepast

Inne narzędzia

- Vite: Narzędzie do budowy projektów front-endowych, które oferuje szybsze budowanie i natychmiastowy HMR (Hot Module Replacement).
- Tailwind CSS: Narzędzie do tworzenia niestandardowych stylów CSS bez konieczności pisania własnych klas.

Każda z tych technologii została wybrana ze względu na swoje mocne strony i przydatność w projekcie, co razem tworzy potężną i nowoczesną aplikację do zamawiania jedzenia online.

Baza danych



Tabele

- Tabela meals przechowuje informacje o posiłkach oferowanych przez restauracje.
- Tabela restaurants przechowuje informacje o restauracjach.
- Tabela meal_categories przechowuje kategorie posiłków.
- Tabela restaurant_categories przechowuje kategorie restauracji.
- Tabela users przechowuje informacje o użytkownikach aplikacji.
- Tabela orders przechowuje informacje o zamówieniach składanych przez użytkowników.
- Tabela order_elements przechowuje informacje o poszczególnych elementach zamówienia. Jest to tabela transakcyjna, z której rekordów składają się poszczególne rekordy tabeli orders.
- Tabela couriers przechowuje informacje o kurierach realizujących dostawy.

Powiązania Między Tabelami

- `meals.restaurant_id` → `restaurants.id`: Posiłek należy do jednej restauracji.
- `meals.category_id` → `meal_categories.id`: Posiłek należy do jednej kategorii posiłków.
- `restaurants.restaurant_category_id` → `restaurant_categories.id`: Restauracja należy do jednej kategorii restauracji.
- `couriers.restaurant_id` → `restaurants.id`: Kurier jest przypisany do jednej restauracji.
- `orders.user_id` → `users.id`: Zamówienie jest składane przez jednego użytkownika.
- `orders.courier_id` → `couriers.id`: Zamówienie jest realizowane przez jednego kuriera.
- `order_elements.order_id` → `orders.id`: Element zamówienia należy do jednego zamówienia.
- `order_elements.meal_id` → `meals.id`: Element zamówienia odnosi się do jednego posiłku.

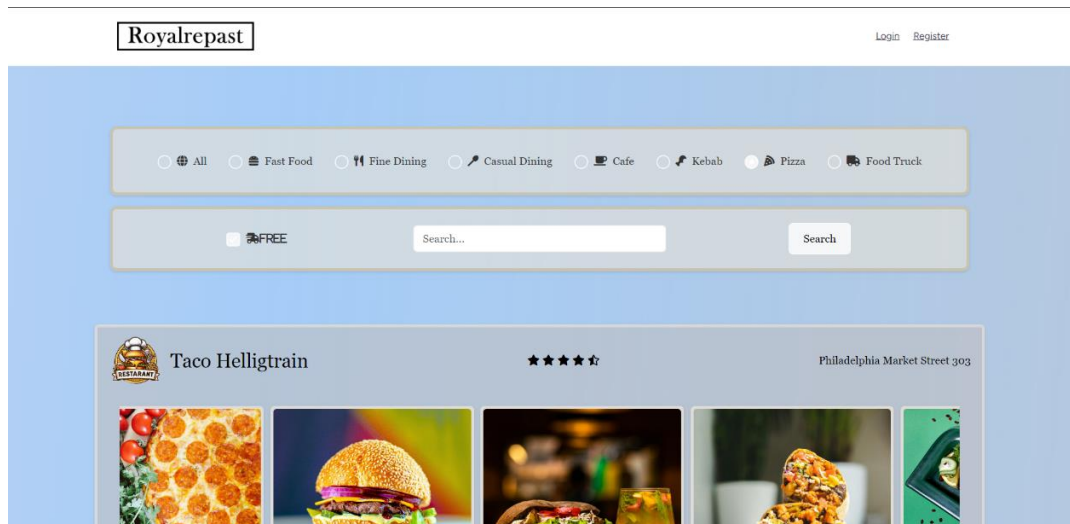
Te powiązania definiują integralność danych i logikę biznesową aplikacji, zapewniając, że każde zamówienie, posiłek, kurier i użytkownik są poprawnie powiązane i reprezentowane w systemie.

GUI

Interfejs użytkownika został zaprojektowany w sposób, który umożliwia użytkownikom łatwe przeglądanie restauracji, wybieranie posiłków oraz składanie zamówień w prosty i intuicyjny sposób. Oto krótki opis głównych elementów GUI:

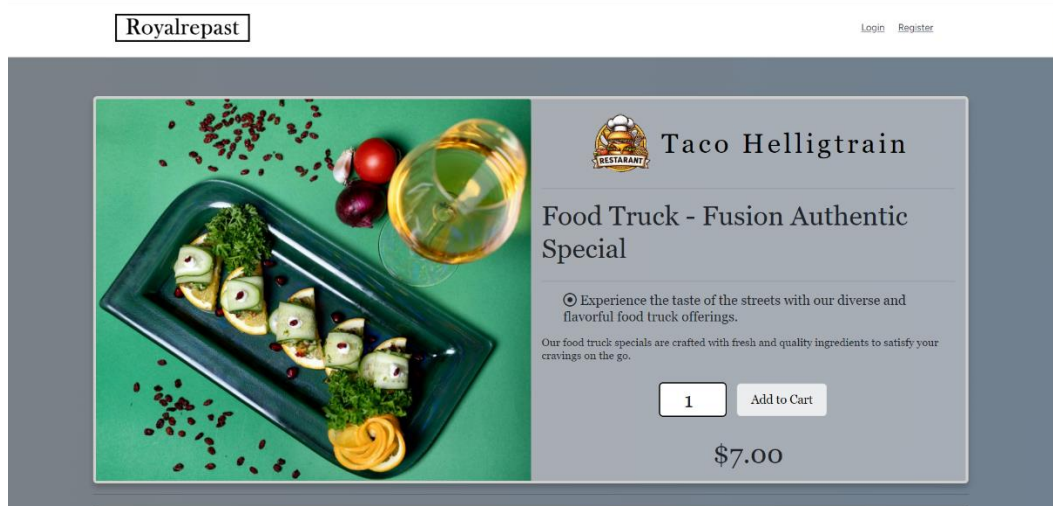
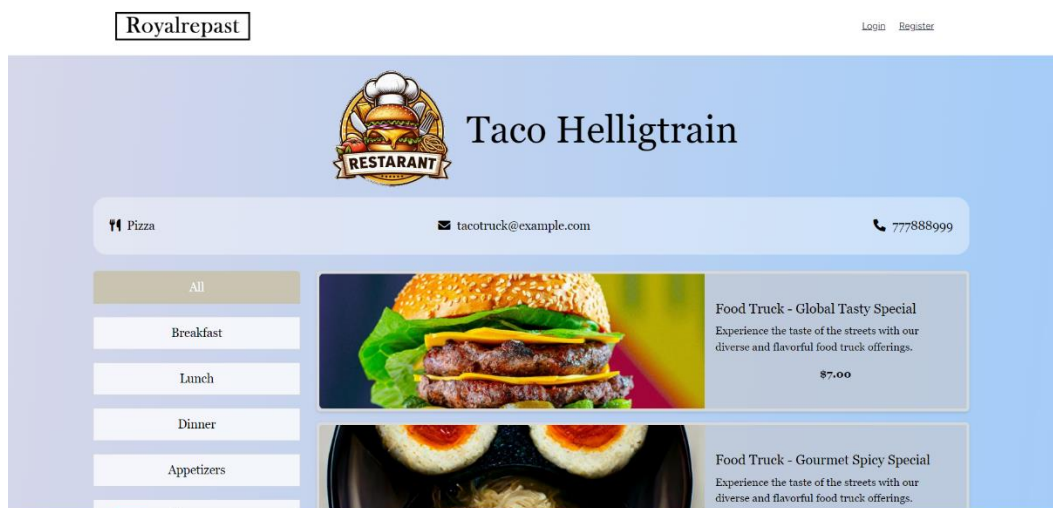
Strona Główna

- Na stronie głównej użytkownicy mogą przeglądać dostępne restauracje oraz ich menu.
- Restauracje są wyświetlane w formie kart, zawierających informacje takie jak nazwa, logo, ocena, adres i dostępność dostawy.
- Użytkownicy zarówno Ci zalogowani, jak i goście mogą przewijać restauracje w poziomie, aby łatwo przeglądać ofertę.



Menu Restauracji

- Po wybraniu restauracji użytkownicy mogą przeglądać jej menu.
- Posiłki są również wyświetlane w formie kart, zawierających zdjęcie, nazwę, opis i cenę.
- Użytkownicy mogą dodawać posiłki do koszyka za pomocą prostego interfejsu.



Koszyk

- Koszyk jest dostępny na każdej stronie aplikacji, umożliwiając użytkownikom monitorowanie ich zamówień.
- Zawiera podsumowanie zamówienia, w tym ilość i cena każdego posiłku oraz łączny koszt.
- Użytkownicy mogą edytować zawartość koszyka, zmieniając ilość lub usuwając posiłki.

Formularz Zamówienia

- Po złożeniu zamówienia użytkownicy są kierowani do formularza zamówienia, gdzie mogą wprowadzić szczegóły dostawy i wybrać metodę płatności.
- Formularz jest prosty i przejrzysty, z polami do wprowadzenia adresu dostawy, wyboru czasu dostawy oraz wyboru metody płatności.

Potwierdzenie Zamówienia

- Po złożeniu zamówienia użytkownicy otrzymują potwierdzenie, które zawiera podsumowanie zamówienia oraz szacowany czas dostawy.
- Potwierdzenie zawiera również informacje kontaktowe, w przypadku potrzeby skontaktowania się z restauracją lub kurierem.

Interfejs użytkownika został zaprojektowany z myślą o prostocie i wygodzie użytkowania, zapewniając jednocześnie wszystkie niezbędne funkcjonalności do składania zamówień online w sposób efektywny i przyjemny.

Uruchomienie Aplikacji

Aby uruchomić aplikację, wymagane jest spełnienie następujących warunków:

Zainstalowane środowisko PHP

Upewnij się, że masz zainstalowane środowisko PHP wraz z menedżerem pakietów Composer.

Baza Danych

Skonfiguruj lokalne środowisko baz danych, takie jak MySQL, PostgreSQL lub SQLite.

Utwórz pustą bazę danych i zapisz dane dostępowe (nazwę bazy danych, użytkownika i hasło).

Środowisko Wykonawcze

W projekcie korzystam z frameworka Laravel, więc upewnij się, że masz zainstalowane środowisko wykonywalne dla Laravel.

Instalacja Zależności

Przejdź do katalogu projektu za pomocą terminala.

Uruchom komendę `composer install`, aby zainstalować wszystkie zależności PHP.

Konfiguracja pliku .env

Skopiuj plik `.env.example` i nazwij go `.env`.

W pliku `.env` ustaw parametry bazy danych (`DB_CONNECTION`, `DB_HOST`, `DB_PORT`, `DB_DATABASE`, `DB_USERNAME`, `DB_PASSWORD`) oraz inne niezbędne ustawienia, takie jak klucz aplikacji (`APP_KEY`).

Migracje Bazy Danych

Wykonaj migracje bazy danych, uruchamiając komendę `php artisan migrate`.

Migracje utworzą tabele w bazie danych na podstawie określonych schematów.

Seedowanie Bazy Danych (Opcjonalne)

Jeśli chcesz wypełnić bazę danych przykładowymi danymi, możesz wykonać seedowanie za pomocą komendy `php artisan db:seed`.

Upewnij się, że stworzyłeś odpowiednie seedery dla swoich modeli.

Uruchomienie Serwera Deweloperskiego

Po zakończeniu konfiguracji, uruchom lokalny serwer deweloperski za pomocą komendy `php artisan serve`.

Aplikacja będzie dostępna pod adresem `http://localhost:8000`.

Testowanie Aplikacji

Po uruchomieniu aplikacji, przetestuj jej funkcjonalności, sprawdzając czy wszystkie funkcje działają poprawnie.

Upewnij się, że użytkownicy mogą przeglądać restauracje, dodawać produkty do koszyka, składać zamówienia i śledzić status dostawy.

Skrypty uruchamiające

Aby ułatwić konfigurację i inicjalizację bazy danych dla aplikacji, przygotowane zostały dwa skrypty: `start.bat` dla systemu Windows i `setup.sh` dla systemów Unix (Linux, macOS). Skrypty te tworzą bazę danych, kopiują plik konfiguracyjny, instalują zależności, migrują bazę danych, uruchamiają skrypt seedujący ją danymi testowymi w odpowiedniej kolejności (ze względu na powiązania między tabelami w bazie), a następnie uruchamiany jest lokalny serwer deweloperski.

Skrypt start.bat

```
%systemDrive%\xampp\mysql\bin\mysql -uroot -e "CREATE DATABASE IF
NOT EXISTS royal_repast;
"

if %errorlevel% neq 0 msg %username% "Nie udało się utworzyć bazy
danych." && exit /b %errorlevel%
php -r "copy('.env.example', '.env');
"

call composer install
call DBrefresh.bat
call php artisan key:generate

call php artisan storage:link
code .
```

Skrypt start.sh

```
#!/bin/bash
mysql -uroot -e "CREATE DATABASE IF NOT EXISTS royal_repast;
"

if [ $? -ne 0 ];
then
    echo "Nie udało się utworzyć bazy danych."
    exit 1
fi
php -r "copy('.env.example', '.env');
"

composer install
./DBrefresh.sh
php artisan key:generate
php artisan storage:link
php artisan serve &
code .
```

Skrypt DBrefresh.bat i DBrefresh.sh

Wykonują migracje i zasiewa bazę danych w odpowiedniej kolejności:

```
call php artisan migrate:fresh
call php artisan db:seed --class="UserSeeder"
call php artisan db:seed --class="RestaurantCategorySeeder"
call php artisan db:seed --class="MealCategorySeeder"
call php artisan db:seed --class="RestaurantSeeder"
call php artisan db:seed --class="MealSeeder"
```

```
call php artisan db:seed --class="CourierSeeder"  
call php artisan db:seed --class="OrderSeeder"  
call php artisan db:seed --class="OrderElementSeeder"
```

Seedy są wykonywane w określonej kolejności ze względu na powiązania w tabelach. Na przykład, MealSeeder musi być uruchomiony po RestaurantSeeder, ponieważ posiłki są powiązane z restauracjami. Analogicznie, OrderElementSeeder musi być uruchomiony po OrderSeeder, ponieważ elementy zamówienia są powiązane z zamówieniami.

W przypadku skryptów .sh ważne jest, aby pamiętać o odpowiednich uprawnieniach dla nich:

```
chmod +x setup.sh  
chmod +x DBrefresh.sh
```

Funkcjonalności aplikacji

Logowanie

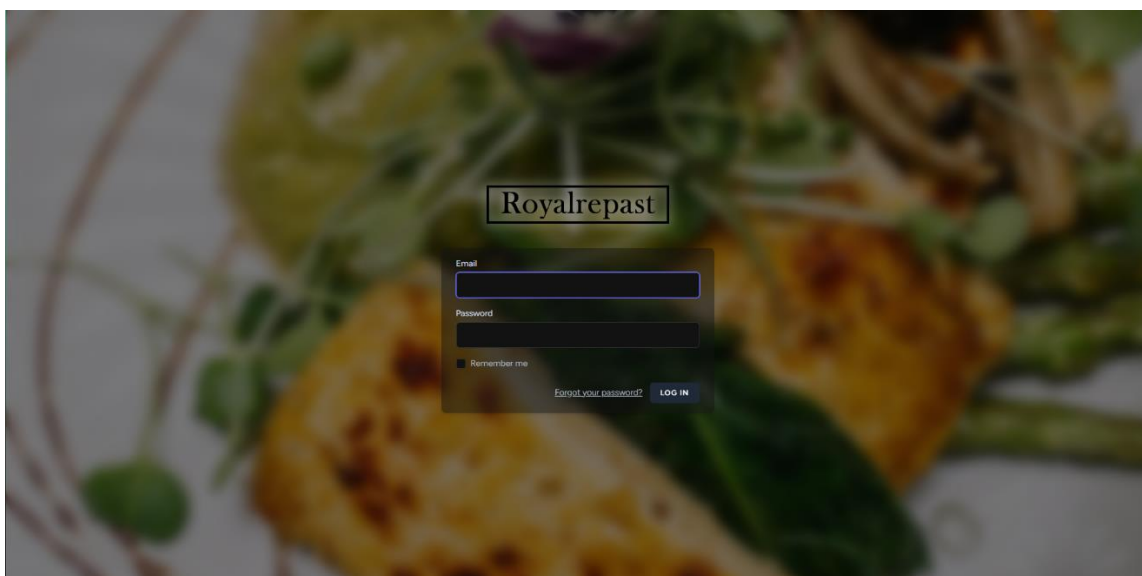
Ekran Logowania

- Użytkownik zostaje poproszony o podanie swojego adresu e-mail oraz hasła.
- Po wypełnieniu formularza i kliknięciu przycisku "Zaloguj", dane są weryfikowane.
- W przypadku poprawnych danych logowania, użytkownik zostaje przekierowany do swojego panelu kontrolnego.
- W przeciwnym razie wyświetla się komunikat o błędnych danych logowania.

Przykładowy login i hasło

Login: jane@example.com

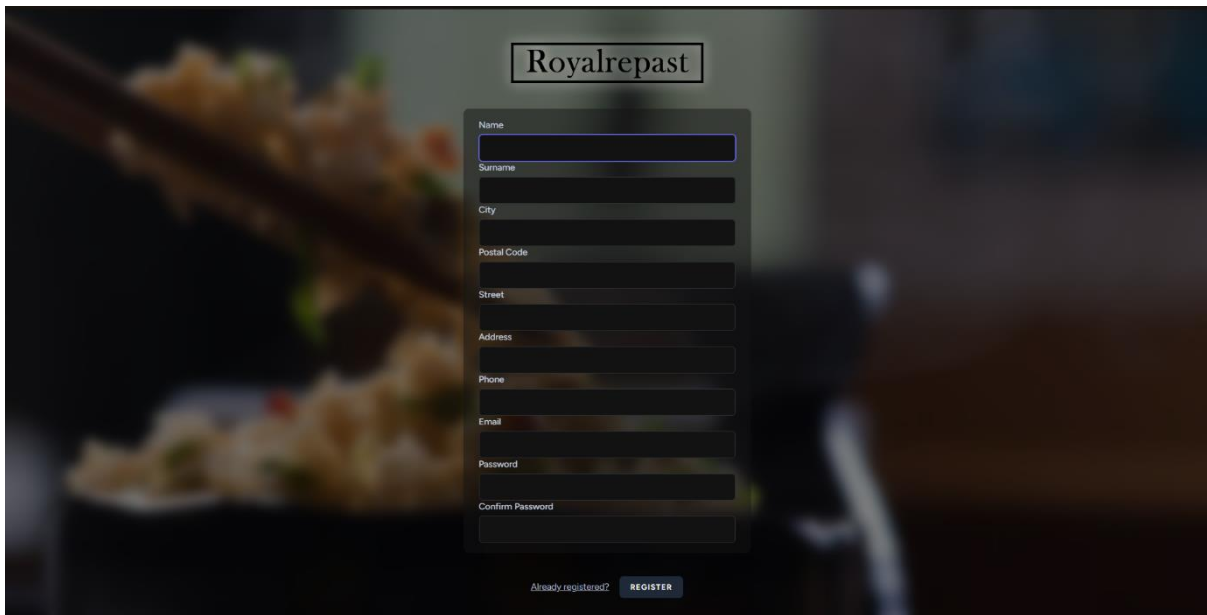
Hasło: password



Rejestracja

Ekran Rejestracji

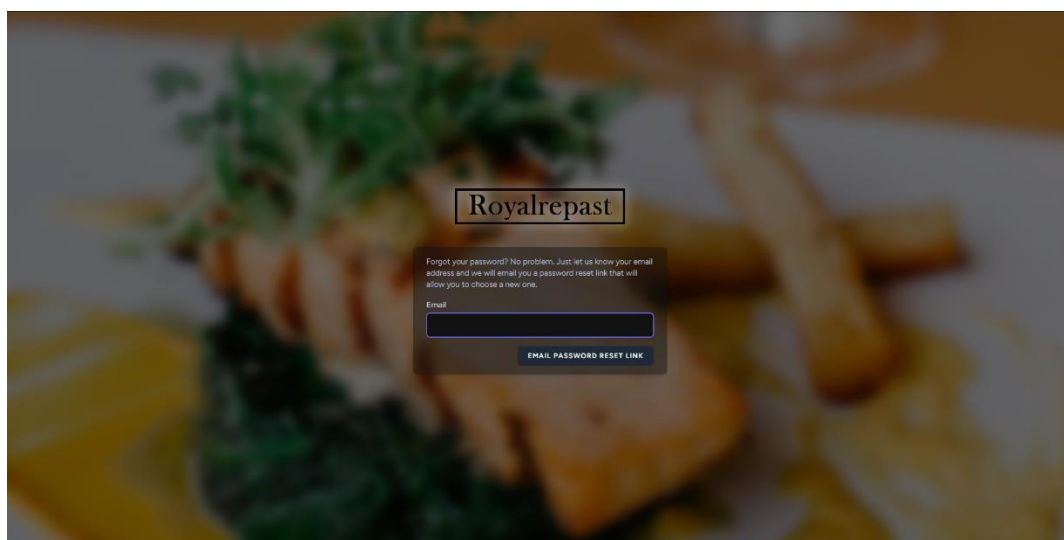
- Nowi użytkownicy mają możliwość utworzenia konta, wypełniając formularz rejestracyjny.
- Formularz zawiera pola takie jak imię, nazwisko, adres e-mail, adres zamieszkania, hasło itp.
- Po wypełnieniu formularza i zatwierdzeniu, nowe konto jest tworzone.

The image shows a registration form for 'Royalrepast' overlaid on a blurred background of food. The form is titled 'Royalrepast' in a box at the top. Below the title, there are input fields for: Name, Surname, City, Postal Code, Street, Address, Phone, Email, Password, and Confirm Password. At the bottom of the form, there is a link 'Already registered?' and a 'REGISTER' button.

Weryfikacja Działania

Po zalogowaniu lub zarejestrowaniu się, użytkownik może przetestować inne funkcjonalności aplikacji, takie jak przeglądanie restauracji, dodawanie produktów do koszyka, składanie zamówień itp.

W razie potrzeby użytkownik może również skorzystać z funkcji zapomnianego hasła, aby zresetować swoje hasło i odzyskać dostęp do konta.

The image shows a 'Forgot your password?' form for 'Royalrepast' overlaid on a blurred background of food. The form is titled 'Royalrepast' in a box at the top. Below the title, there is a message: 'Forgot your password? No problem. Just let us know your email address and we will email you a password reset link that will allow you to choose a new one.' Below this message is an input field for 'Email'. At the bottom of the form, there is a button labeled 'EMAIL PASSWORD RESET LINK'.

Panel CRUD dla administratora

Do każdej tabeli został zaimplementowany panel CRUD do zarządzania danymi przez administratora. Poniżej omówione zostały komponenty panelu CRUD na przykładzie tabeli users i kontrolera AdminUsersController:

Wyświetlanie tabeli użytkowników

```
public function index(Request $request)
{
    $search = $request->input('search');
    $column = $request->input('column', 'name');
    $usersQuery = User::query();
    if ($search) { $usersQuery->where($column, 'like', "%$search%"); }
    $users = $usersQuery->where('id', '<>', Auth::id())->get();
    return view('admin.users.table-users', compact('users'));
}
```

- Cel: Wyświetla listę wszystkich użytkowników (poza zalogowanym adminem).
- Funkcje:
- Filtracja: Możliwość filtrowania użytkowników na podstawie kolumny (domyślnie name) i wartości wyszukiwania.
- Widok: admin.users.table-users - widok, który wyświetla tabelę użytkowników.

←

Admin

Users Table

Name

▼

Search...

Search

ID	Name	Surname	Email	Permission	City	Postal Code	Street	Address	Phone		
2	John	Doe	john@example.com	user	New York	10001	Main Street	123	123456789	Edit	Delete
3	Jane	Doe	jane@example.com	user	Los Angeles	90001	Broadway	456	987654321	Edit	Delete
4	Michael	Smith	michael@example.com	user	Chicago	60601	Michigan Avenue	789	111222333	Edit	Delete
5	Emily	Johnson	emily@example.com	user	Houston	77001	Main Street	101	444555666	Edit	Delete
6	Christopher	Brown	chris@example.com	user	Philadelphia	19101	Market Street	202	777888999	Edit	Delete
7	Jessica	Williams	jessica@example.com	user	Phoenix	85001	Central Avenue	303	333222111	Edit	Delete
8	Daniel	Jones	daniel@example.com	user	San Antonio	78201	Alamo Street	404	666777888	Edit	Delete

Add New User

Formularz edycji użytkownika

```
public function updateUser($id)
```

```
{ $user = User::find($id);  
  return view('admin.users.user-edit', compact('user'))
```

- Cel: Wyświetla formularz edycji dla wybranego użytkownika.
- Widok: admin.users.user-edit - widok, który zawiera formularz edycji użytkownika.

Formularz usunięcia użytkownika

```
public function userDestroy($id)
```

```
{ $user = User::find($id);  
  return view('admin.users.user-delete', compact('user'));  
}  
}
```

- Cel: Wyświetla formularz usunięcia dla wybranego użytkownika.
- Widok: admin.users.user-delete - widok, który zawiera formularz usunięcia użytkownika.

Formularz dodawania nowego użytkownika

```
public function userCreate()
{
    return view('auth.register');
}
```

- Cel: Wyświetla formularz rejestracji dla nowego użytkownika.
- Widok: auth.register - widok, który zawiera formularz rejestracji.

Aktualizacja użytkownika

```
public function update(Request $request, $id) {
    $request->validate([
        'name' => 'required|string|max:255',
        'surname' => 'required|string|max:255',
        'email' => 'required|email|max:255',
        'city' => 'required|string|max:255',
        'postal_code' => ['required', 'string', 'max:255', 'regex:/^\d{2}-\d{3}$/'],
        'street' => 'required|string|max:255',
        'address' => 'required|string|max:255',
        'phone' => ['required', 'string', 'max:255', 'regex:/^\+\d{2} \d{9}$/'],
    ]);
    $user = User::findOrFail($id);
    $user->update($request->all());
    return redirect()->route('admin.users')->with('success', 'User updated successfully');
}
```

- Walidacja: Sprawdza, czy wszystkie wymagane pola są poprawnie wypełnione.
- Aktualizacja: Aktualizuje dane użytkownika na podstawie przekazanych wartości.
- Przekierowanie: Przekierowuje z komunikatem sukcesu.

Usuwanie użytkownika

```
public function destroy($id) {
    $user = User::findOrFail($id);
    if ($user->orders()->exists()) {
        return redirect()->route('admin.users')->with('error', 'Couldn\'t remove this user, because there are referral records.');
```

```
    }
    $user->delete();
    return redirect()->route('admin.users')->with('success', 'User has been successfully removed.');
```

```
}
```

- Sprawdzenie: Sprawdza, czy użytkownik ma powiązane zamówienia. Jeśli tak, nie usuwa użytkownika.
- Usunięcie: Usuwa użytkownika, jeśli nie ma powiązanych zamówień.
- Przekierowanie: Przekierowuje z odpowiednim komunikatem (sukces lub błąd).

Dodawanie nowego użytkownika

```
public function store(Request $request) {
    $request->validate([
        'name' => 'required|string|max:255',
        'surname' => 'required|string|max:255',
        'email' => 'required|email|max:255',
        'password' => 'required|string|max:255',
        'permission' => 'required|string|max:255',
        'city' => 'required|string|max:255',
        'postal_code' => ['required', 'string', 'max:255', 'regex:/^\d{2}-\d{3}$/'],
        'street' => 'required|string|max:255',
        'address' => 'required|string|max:255',
        'phone' => ['required', 'string', 'max:255', 'regex:/^\+\d{2} \d{9}$/'],
    ]);
```

```

User::create([ 'name' => $request->name, 'surname' => $request->surname,
'email' => $request->email, 'permission' => $request->permission, 'city' =>
$request->city, 'postal_code' => $request->postal_code, 'street' => $request-
>street, 'address' => $request->address, 'phone' => $request->phone, 'password'
=>
Hash::make($request->password),
]);
return redirect()->route('admin.users')->with('success', 'User added
successfully.');
```

- Walidacja: Sprawdza, czy wszystkie wymagane pola są poprawnie wypełnione.
- Tworzenie: Tworzy nowego użytkownika z podanymi danymi.
- Przekierowanie: Przekierowuje z komunikatem sukcesu.

Przeglądanie ogólnodostępnych zasobów

Aplikacja umożliwia użytkownikom przeglądanie dostępnych zasobów, takich jak restauracje i posiłki, z możliwością filtrowania wyników według różnych kryteriów. Poniżej znajduje się demonstracja i opis działania filtrowania zasobów na podstawie kodu z kontrolera RestaurantController i MealController.

Pobieranie kategorii restauracji

```
$categories = Restaurant_category::all();
```

Pobiera wszystkie dostępne kategorie restauracji.

Sortowanie restauracji

```

$sort_by = $request->input('sort_by');
if ($sort_by) { $restaurants->orderBy($sort_by);
}
```

Sortowanie restauracji według wybranej kolumny (np. nazwy, ocen).

Filtrowanie według kategorii

```

$selected_category = $request->input('category', 'all');
if ($selected_category && $selected_category !== 'all') { $restaurants-
>whereHas('category', function($q) use ($selected_category) { $q->where('name',
$selected_category);
});
}
```

Filtrowanie restauracji na podstawie wybranej kategorii.

Filtrowanie według darmowej dostawy

```
$free_delivery = $request->input('free_delivery');
if ($free_delivery) { $restaurants->where('delivery_price', 0.0);
}
```

Filtrowanie restauracji oferujących darmową dostawę.

Wyszukiwanie restauracji

```
$search = $request->input('search');
if ($search) { $restaurants->where('name', 'like', '%' . $search . '%');
}
```

Wyszukiwanie restauracji na podstawie nazwy.

Paginacja wyników

```
$restaurants = $restaurants->paginate(5);
```

Paginacja wyników, aby wyświetlać 5 restauracji na stronie.

Widok przeglądania restauracji

Widok main-panel.restaurants wyświetla listę restauracji wraz z opcjami filtrowania i sortowania.

```
return view('main-panel.restaurants', compact('categories', 'restaurants',
'sort_by', 'selected_category', 'free_delivery', 'search'));
```

Przegląd pojedynczej restauracji

Metoda show w kontrolerze RestaurantController umożliwia przeglądanie szczegółów wybranej restauracji oraz jej posiłków.

```
public function show($restaurant_id) { $restaurant =
Restaurant::findOrFail($restaurant_id);
$meals = Meal::where('restaurant_id', $restaurant_id)->get();
$categories = Meal_category::all();
$nonEmptyCategories = $categories->filter(function($category) use ($meals) {
return $meals->where('category_id', $category->id)->isNotEmpty();
});
return view('main-panel/restaurant-show', compact('restaurant',
'nonEmptyCategories', 'meals'));
}
```

Wyświetlanie szczegółów restauracji i jej posiłków, filtrowanie posiłków według kategorii, z uwzględnieniem, że jeśli w danej kategorii nie występuje żaden posiłek, to nie jest wyświetlana.

Pobieranie szczegółów posiłku

Metoda show pobiera szczegóły posiłku na podstawie przekazanego modelu Meal.

```
public function show(Meal $meal) { $mealsFromSameRestaurant = $meal->restaurant->meals()->where('id', '!=', $meal->id)->inRandomOrder()->take(4)->get();
    $mealsFromSameCategory = $meal->category->meals()->where('id', '!=', $meal->id)->inRandomOrder()->take(4)->get();
    return view('main-panel.meals-show', compact('meal', 'mealsFromSameRestaurant', 'mealsFromSameCategory'));
}
```

Posiłki z tej samej restauracji

```
$mealsFromSameRestaurant = $meal->restaurant->meals()->where('id', '!=', $meal->id)->inRandomOrder()->take(4)->get();
```

Pobiera 4 losowe posiłki z tej samej restauracji, z wyłączeniem obecnie wyświetlanego posiłku.

Posiłki z tej samej kategorii

```
$mealsFromSameCategory = $meal->category->meals()->where('id', '!=', $meal->id)->inRandomOrder()->take(4)->get();
```

Pobiera 4 losowe posiłki z tej samej kategorii, z wyłączeniem obecnie wyświetlanego posiłku.

Zarządzanie swoimi danymi przez użytkownika

Kontroler ProfileController umożliwia użytkownikom zarządzanie swoim profilem, zamówieniami oraz adresami. Poniżej znajdują się szczegóły dotyczące tych funkcjonalności:

Wyświetlanie profilu użytkownika

Metoda show wyświetla profil użytkownika oraz jego zamówienia.

Profile

Edit Profile

My Orders

Order ID	Date	Payment Method	Total Price	Details
11	30 May 2024	Card	\$21.00	View Details

Edycja profilu użytkownika

Metoda edit umożliwia wyświetlenie formularza do edycji profilu użytkownika.

Profile Information
Update your account's profile information and email address.

Name

Email

SAVE

Update Password
Ensure your account is using a long, random password to stay secure.

Current Password

New Password

Confirm Password

SAVE

Aktualizacja profilu użytkownika

Metoda update aktualizuje informacje w profilu użytkownika na podstawie danych z formularza. Weryfikuje zmiany w adresie e-mail i ustawia pole email_verified_at na null, jeśli adres e-mail został zmieniony.

Usuwanie konta użytkownika

Metoda destroy usuwa konto użytkownika po weryfikacji hasła, wylogowuje użytkownika i unieważnia sesję.

Delete Account
Once your account is deleted, all of its resources and data will be permanently deleted. Before deleting your account, please download any data or information that you wish to retain.

DELETE ACCOUNT

Aktualizacja adresu użytkownika

Metoda `updateAddress` umożliwia użytkownikowi aktualizację adresu.

Zarządzanie swoimi zasobami przez użytkownika

Zarządzanie koszykiem produktów do zamówienia umożliwia kontroler `CartController`:

Dodanie posiłku do koszyka

```
public function add(Request $request, Meal $meal) { $request->validate([
    'quantity' => 'required|integer|min:1' ]);
    $cart = session()->get('cart', []);
    $restaurantId = $meal->restaurant_id;
    if (isset($cart[$restaurantId][$meal->id])) { $cart[$restaurantId][$meal-
    >id]['quantity'] += $request->quantity;
    } else { $cart[$restaurantId][$meal->id] = [ 'name' => $meal->name,
    'quantity' => $request->quantity, 'price' => $meal->price, 'image' => $meal-
    >image_path, 'restaurant_name' => $meal->restaurant->name ];
    } session()->put('cart', $cart);
    return redirect()->back()->with('success', 'Meal added to cart!');
}
```

- Walidacja: Sprawdza, czy ilość jest podana i jest co najmniej 1.
- Dodanie do koszyka: Jeśli posiłek już istnieje w koszyku, aktualizuje ilość. W przeciwnym razie dodaje nowy posiłek do koszyka.
- Zapis do sesji: Aktualizuje koszyk w sesji użytkownika.
- Powrót do strony: Przekierowuje z komunikatem o powodzeniu.

Wyświetlenie koszyka

```
public function viewCart() { $cart = session()->get('cart', []);
    $restaurantDetails = [];
    $totalCost = 0;
    foreach ($cart as $restaurantId => $meals) { $restaurant =
    \App\Models\Restaurant::find($restaurantId);
    $restaurantDeliveryCost = $restaurant->delivery_price;
    $restaurantDetails[$restaurantId] = [ 'name' => $restaurant->name,
    'delivery_price' => $restaurantDeliveryCost, 'logo_path' => $restaurant-
    >logo_path, ];
    $restaurantTotal = 0;
    foreach ($meals as $meal) { $restaurantTotal += $meal['price'] *
    $meal['quantity'];
    } if ($restaurantTotal >= $this->freeDeliveryThreshold) { $restaurantDeliveryCost
    = 0;
    } $totalCost += $restaurantTotal + $restaurantDeliveryCost;
    } $cartCount = array_sum(array_map(fn($meals) => array_sum(array_column($meals,
    'quantity')), $cart));
    return view('main-panel.cart.view', compact('cart', 'cartCount',
    'restaurantDetails', 'totalCost'));
}
```

- Pobranie koszyka: Pobiera koszyk z sesji.
- Obliczenie kosztów: Oblicza całkowity koszt zamówienia oraz koszty dostawy, uwzględniając darmową dostawę, jeśli próg kosztów został przekroczony.
- Wyświetlenie koszyka: Przekazuje dane do widoku koszyka.

Aktualizacja koszyka

```
public function updateCart(Request $request, $mealId) { $request->validate([
    'quantity' => 'required|integer|min:1' ]]);
    $cart = session()->get('cart', []);
    foreach ($cart as $restaurantId => $meals) { if (isset($meals[$mealId])) {
    $cart[$restaurantId][$mealId]['quantity'] = $request->quantity;
    session()->put('cart', $cart);
    return redirect()->back()->with('success', 'Cart updated successfully!');
    } } return redirect()->back()->with('error', 'Product not found in cart!');
}
```

- Walidacja: Sprawdza, czy ilość jest podana i jest co najmniej 1.
- Aktualizacja ilości: Aktualizuje ilość posiłku w koszyku.
- Powrót do strony: Przekierowuje z komunikatem o powodzeniu lub błędzie.

Usunięcie posiłku z koszyka

```
public function removeCart($mealId) { $cart = session()->get('cart', []);
    foreach ($cart as $restaurantId => $meals) { if (isset($meals[$mealId])) {
unset($cart[$restaurantId][$mealId]);
        if (empty($cart[$restaurantId])) { unset($cart[$restaurantId]);
        } session()->put('cart', $cart);
        return redirect()->back()->with('success', 'Product removed from cart
successfully!');
    } } return redirect()->back()->with('error', 'Product not found in cart!');
}
```

- Usunięcie posiłku: Usuwa posiłek z koszyka. Jeśli restauracja nie ma już posiłków w koszyku, usuwa również restaurację z koszyka.
- Powrót do strony: Przekierowuje z komunikatem o powodzeniu lub błędzie.

Checkout

```
public function checkout() { $cart = session()->get('cart', []);
    $totalCost = 0;
    foreach ($cart as $restaurantId => $meals) { $restaurant =
\App\Models\Restaurant::find($restaurantId);
    $restaurantDeliveryCost = $restaurant->delivery_price;
    $restaurantTotal = 0;
    foreach ($meals as $mealId => $details) { $itemTotal = $details['price'] *
    $details['quantity'];
    $restaurantTotal += $itemTotal;
    } if ($restaurantTotal >= $this->freeDeliveryThreshold) {
$restaurantDeliveryCost = 0;
    } $restaurantTotal += $restaurantDeliveryCost;
    $totalCost += $restaurantTotal;
    } $user = Auth::user();
    return view('main-panel.cart.checkout', compact('cart', 'totalCost',
    'user'));
}
```

- Pobranie koszyka: Pobiera koszyk z sesji.
- Obliczenie kosztów: Oblicza całkowity koszt zamówienia oraz koszty dostawy, uwzględniając darmową dostawę, jeśli próg kosztów został przekroczony.
- Wyświetlenie widoku: Przekazuje dane do widoku kasy.

Złożenie zamówienia

```
public function placeOrder(Request $request) { $request->validate([
'payment_method' => 'required|string', ]);
    $cart = session()->get('cart', []);
    if (empty($cart)) { return redirect()->back()->with('error', 'Your cart is
empty');
    } foreach ($cart as $restaurantId => $meals) { $restaurant =
\App\Models\Restaurant::find($restaurantId);
```

```

$restaurantDeliveryCost = $restaurant->delivery_price;
$restaurantTotal = 0;
foreach ($meals as $mealId => $details) { $itemTotal = $details['price'] *
$details['quantity'];
$restaurantTotal += $itemTotal;
} if ($restaurantTotal >= $this->freeDeliveryThreshold) { $restaurantDeliveryCost
= 0;
} $totalCost = $restaurantTotal + $restaurantDeliveryCost;
$fullOrderAddress = $request->city." ".$request->street." ".$request->
postal_code." ".$request->address;
if ($totalCost > 0) { $order = Order::create([ 'user_id' => Auth::id(),
'courier_id' => 1, 'payment_method' => $request->payment_method, 'address' =>
$fullOrderAddress, 'total_cost' => $totalCost, ]);
foreach ($meals as $mealId => $details) { Order_element::create([ 'order_id' =>
$order->id, 'meal_id' => $mealId, 'quantity' => $details['quantity'] ]);
} } session()->forget('cart');
return redirect()->route('restaurants.index')->with('success', 'Order placed
successfully!');
}

```

- Walidacja: Sprawdza, czy metoda płatności jest podana.
- Pobranie koszyka: Pobiera koszyk z sesji.
- Tworzenie zamówienia: Dla każdej restauracji w koszyku tworzy zamówienie, oblicza całkowity koszt i zapisuje szczegóły zamówienia do bazy danych.
- Czyszczenie koszyka: Usuwa koszyk z sesji.
- Powrót do strony: Przekierowuje z komunikatem o powodzeniu.

Rejestracja nowego użytkownika

Tworzenie nowego profilu jest zarządzane poprzez kontroler RegisteredUserController:

Wyświetlanie widoku rejestracji

```

public function create(): View { return view('auth.register');
}

```

- Cel: Wyświetla formularz rejestracji dla nowych użytkowników.
- Widok: auth.register - jest to widok, który zawiera formularz rejestracji.

Obsługa żądania rejestracji

```

public function store(Request $request): RedirectResponse { $request->validate([
'name' => ['required', 'string', 'max:255'], 'surname' => ['required', 'string',
'max:255'], 'city' => ['required', 'string', 'max:255'], 'postal_code' =>
['required', 'string', 'regex:/^\d{2}-\d{3}$/'], 'street' => ['required',
'string', 'max:255'], 'address' => ['required', 'string', 'max:255'], 'phone' =>
['required', 'string', 'regex:/^\+\d{2} \d{3} \d{3} \d{3}$/'], 'email' =>
['required', 'string', 'lowercase', 'email', 'max:255', 'unique:'.User::class],
'password' => ['required', 'confirmed', Rules\Password::defaults()], ]);
}

```

```

$user = User::create([ 'name' => $request->name, 'surname' => $request->surname,
'permission' => 'user', 'city' => $request->city, 'postal_code' => $request->postal_code, 'street' => $request->street, 'address' => $request->address,
'phone' => $request->phone, 'email' => $request->email, 'password' =>
Hash::make($request->password), []);
event(new Registered($user));
Auth::login($user);
return redirect(route('dashboard'));
}

```

- Walidacja: Sprawdza, czy wszystkie wymagane pola są wypełnione prawidłowo:
 - name, surname, city, street, address - wymagane, maksymalnie 255 znaków.
 - postal_code - wymagany, zgodny z formatem XX-XXX.
 - phone - wymagany, zgodny z formatem +XX XXX XXX XXX.
 - email - wymagany, musi być unikalny w tabeli users.
 - password - wymagane, musi spełniać domyślne reguły zabezpieczeń, potwierdzenie hasła (confirmed).
- Tworzenie użytkownika: Jeśli walidacja przejdzie pomyślnie, tworzy nowego użytkownika z podanymi danymi.
- Zdarzenie rejestracji: Wysyła zdarzenie Registered dla nowego użytkownika.
- Automatyczne logowanie: Loguje nowo zarejestrowanego użytkownika.
- Przekierowanie: Przekierowuje użytkownika na stronę główną aplikacji.

Dodatki

Przesuwanie się poślinków na stronie głównej

Funkcja randomSpeed

```

function randomSpeed() {
    return Math.floor(Math.random() * 30) + 30;
}

```

- Cel: Generuje losową prędkość przewijania w przedziale od 30 do 59 milisekund.
- Szczegóły: Math.random() generuje liczbę zmiennoprzecinkową z przedziału [0, 1). Mnożenie przez 30 daje przedział [0, 30). Dodanie 30 przesuwa ten przedział do [30, 60).

Funkcja autoScroll

```

function autoScroll(element) {
    const container = element.closest('.horizontal-scroll-container');
    const containerScrollWidth = container.scrollWidth;
    const containerWidth = container.offsetWidth;

```

```

if (containerScrollWidth > containerWidth) {
  let scrollPosition = 0;
  setInterval(() => {
    if (scrollPosition <= containerScrollWidth - containerWidth) {
      scrollPosition += 1;
      container.scrollLeft = scrollPosition;
    } else {
      scrollPosition = 0;
      container.scrollLeft = 0;
    }
  }, randomSpeed());
}
}

```

- Cel: Automatycznie przewija zawartość poziomego kontenera, jeśli jego szerokość przewijania (scrollWidth) jest większa niż szerokość widoczna (offsetWidth).
 - container: Znajduje najbliższy element nadrzędny z klasą .horizontal-scroll-container.
 - containerScrollWidth: Całkowita szerokość przewijania kontenera.
 - containerWidth: Widoczna szerokość kontenera.
 - Sprawdza, czy szerokość przewijania jest większa od widocznej szerokości.
 - Ustawia scrollPosition na 0.
 - Używa setInterval do regularnego przewijania kontenera o 1 piksel, dopóki nie osiągnie końca. Po osiągnięciu końca resetuje pozycję przewijania do 0.
 - randomSpeed(): Używa losowej prędkości przewijania z przedziału 30-59 ms, aby uzyskać bardziej naturalny efekt.

Event DOMContentLoaded

```

document.addEventListener('DOMContentLoaded', () => {
  const scrollContainers = document.querySelectorAll('.horizontal-scroll-container');
  scrollContainers.forEach(container => {
    autoScroll(container);
  });
});

```

- Cel: Uruchamia funkcję autoScroll na wszystkich elementach z klasą .horizontal-scroll-container po załadowaniu całego dokumentu.
 - Słucha zdarzenia DOMContentLoaded, które jest wywoływane, gdy cały dokument HTML zostanie w pełni załadowany i przetworzony.
 - Znajduje wszystkie elementy z klasą .horizontal-scroll-container.
 - Dla każdego znalezionej elementu wywołuje funkcję autoScroll.

Skrypt ten dodaje automatyczne przewijanie dla wszystkich kontenerów poziomych na stronie, które mają klasę .horizontal-scroll-container. Przewijanie odbywa się z losową prędkością, co sprawia, że wygląd jest bardziej dynamiczny i naturalny. Skrypt uruchamia przewijanie po załadowaniu całego dokumentu HTML.

Darmowa dostawa przy określonej cenie zamówienia

Funkcjonalność darmowej dostawy jest wdrażana w procesie realizacji zamówienia (checkout) w aplikacji. Zamówienia przekraczające określoną wartość kwalifikują się do darmowej dostawy. Próg darmowej dostawy jest zdefiniowany w pliku konfiguracyjnym aplikacji.

Konfiguracja progu darmowej dostawy

Najpierw musimy ustawić próg darmowej dostawy w pliku konfiguracyjnym config/app.php.

```
<?php
return [
    // inne ustawienia...

    'free_delivery_threshold' => 49, // Próg darmowej dostawy w jednostkach
    walutowych
];
```

Próg darmowej dostawy jest ustawiony na 49. Oznacza to, że zamówienia o wartości co najmniej 49 jednostek walutowych będą kwalifikować się do darmowej dostawy.

Funkcja checkout w CartController

Poniżej znajduje się zmodyfikowana funkcja checkout, która uwzględnia darmową dostawę:

```
public function checkout()
{
    $cart = session()->get('cart', []);
    $totalCost = 0;

    foreach ($cart as $restaurantId => $meals) {
```

```

$restaurant = \App\Models\Restaurant::find($restaurantId);
$restaurantDeliveryCost = $restaurant->delivery_price;

$restaurantTotal = 0;
foreach ($meals as $mealId => $details) {
    $itemTotal = $details['price'] * $details['quantity'];
    $restaurantTotal += $itemTotal;
}

// darmowa dostawa, jeśli wartość zamówienia przekracza próg
if ($restaurantTotal >= $this->freeDeliveryThreshold) {
    $restaurantDeliveryCost = 0;
}

$restaurantTotal += $restaurantDeliveryCost;
$totalCost += $restaurantTotal;
}

$user = Auth::user();

return view('main-panel.cart.checkout', compact('cart', 'totalCost', 'user'));
}

```

Pobranie zawartości koszyka:

```
$cart = session()->get('cart', []);
```

Pobiera zawartość koszyka z sesji. Jeśli koszyk jest pusty, zwraca pustą tablicę.

Inicjalizacja całkowitego kosztu zamówienia:

```
$totalCost = 0;
```

Iteracja przez elementy koszyka:

```

foreach ($cart as $restaurantId => $meals) {
    $restaurant = \App\Models\Restaurant::find($restaurantId);
    $restaurantDeliveryCost = $restaurant->delivery_price;
    $restaurantTotal = 0;
    foreach ($meals as $mealId => $details) {
        $itemTotal = $details['price'] * $details['quantity'];
        $restaurantTotal += $itemTotal;
    }
}

```

Iteruje przez każdy element koszyka, grupując je według identyfikatora restauracji. Dla każdej restauracji oblicza całkowity koszt posiłków (\$restaurantTotal).

Sprawdzenie progu darmowej dostawy:

```
if ($restaurantTotal >= $this->freeDeliveryThreshold) {
    $restaurantDeliveryCost = 0;
}
```

Jeśli wartość zamówienia dla danej restauracji przekracza próg darmowej dostawy, koszt dostawy jest ustawiany na 0.

Dodanie kosztu dostawy do całkowitej wartości zamówienia:

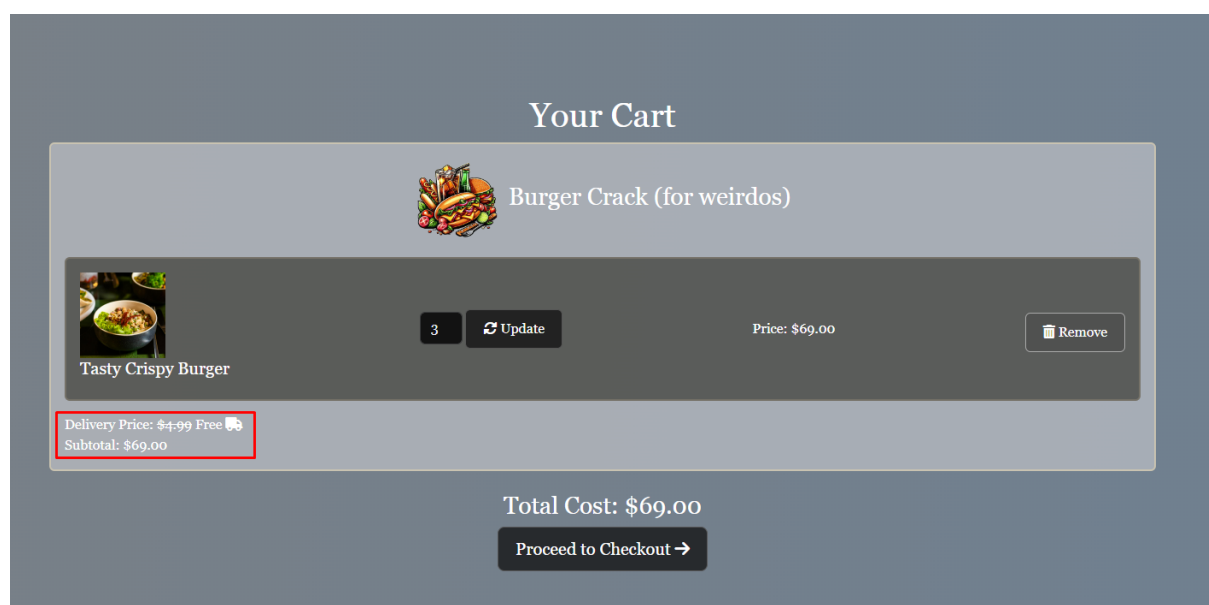
```
$restaurantTotal += $restaurantDeliveryCost;
$totalCost += $restaurantTotal;
```

Zwrócenie widoku z danymi o zamówieniu:

```
$user = Auth::user();
return view('main-panel.cart.checkout', compact('cart', 'totalCost', 'user'));
```

Pobiera aktualnie zalogowanego użytkownika i zwraca widok checkout z danymi o koszyku, całkowitym koszcie i użytkowniku.

Kod powyżej ilustruje, jak wdrożono darmową dostawę dla zamówień przekraczających określoną wartość. Próg darmowej dostawy jest konfigurowany w pliku config/app.php. Podczas realizacji zamówienia, jeśli całkowity koszt zamówienia dla danej restauracji przekracza ten próg, koszt dostawy jest ustawiany na 0. Całkowity koszt zamówienia, w tym koszt dostawy, jest następnie przekazywany do widoku checkout.



Wyświetlanie aktualnej ilości elementów w koszyku

Aktualizacja Metody viewCart w CartController

W celu wyświetlenia łącznej ilości elementów znajdujących się w koszyku, zaktualizujemy metodę viewCart w kontrolerze CartController.

```
public function viewCart()
{
    $cart = session()->get('cart', []);
    $restaurantDetails = [];
    $totalCost = 0;
    $restaurantDeliveryCost = 0;
    foreach ($cart as $restaurantId => $meals) {
        $restaurant = \App\Models\Restaurant::find($restaurantId);
        $restaurantDeliveryCost = $restaurant->delivery_price;
        $restaurantDetails[$restaurantId] = [
            'name' => $restaurant->name,
            'delivery_price' => $restaurantDeliveryCost,
            'logo_path' => $restaurant->logo_path,
        ];
        $restaurantTotal = 0;
        foreach ($meals as $meal) {
            $restaurantTotal += $meal['price'] * $meal['quantity'];
        }
        if ($restaurantTotal >= $this->freeDeliveryThreshold) {
            $restaurantDeliveryCost = 0;
        }
        $totalCost += $restaurantTotal + $restaurantDeliveryCost;
    }
    $cartCount = array_sum(array_map(fn($meals) => array_sum(array_column($meals, 'quantity')), $cart));
    return view('main-panel.cart.view', compact('cart', 'cartCount', 'restaurantDetails', 'totalCost', 'restaurantDeliveryCost'));
}
```

Aktualizacja AppServiceProvider

Aby łączna ilość elementów w koszyku była dostępna we wszystkich widokach, użyjemy View Composer w AppServiceProvider.

```
class AppServiceProvider extends ServiceProvider
{
    public function boot(): void
    {
        View::composer('*', function ($view) {
            $cart = session('cart', []);
            $cartCount = 0;

            foreach ($cart as $restaurantId => $meals) {
                foreach ($meals as $meal) {
                    $cartCount += $meal['quantity'];
                }
            }
            $view->with('cartCount', $cartCount);
        });
    }
}
```

- Aktualizacja viewCart: Metoda viewCart w CartController oblicza teraz cartCount, czyli łączną ilość elementów w koszyku.
- View Composer: W AppServiceProvider dodaliśmy View Composer, który oblicza cartCount na podstawie sesji i udostępnia tę wartość wszystkim widokom.
- Wyświetlanie w widokach: Zmienna cartCount jest teraz dostępna we wszystkich widokach i może być użyta do wyświetlenia liczby elementów w koszyku.

