AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

PizzaUtopia

Autor:
Kacper Kosal
Adrian Kądziołka

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2025

Spis treści

1. Ogólne określenie wymagań

1.1. Cel programu

Celem aplikacji jest umożliwienie użytkownikom zamawiania pizzy bezpośrednio z poziomu ich smartfonów. Oprócz standardowej funkcji zamawiania, aplikacja ma na celu także zapewnienie dodatkowych funkcji bezpieczeństwa oraz interakcji z użytkownikami poprzez wykorzystanie wbudowanych czujników smartfonów, co sprawi, że korzystanie z niej będzie nie tylko wygodne, ale i bezpieczne.

1.2. Zakres funkcjonalny

Logowanie biometryczne: Aplikacja wykorzysta czujnik linii papilarnych, co umożliwi bezpieczne logowanie oraz autoryzację użytkowników. Dzięki temu, każdy użytkownik będzie mógł szybko i w bezpieczny sposób uzyskać dostęp do swojego konta bez konieczności wpisywania hasła.

Interfejs użytkownika reagujący na warunki oświetleniowe: Aplikacja będzie dynamicznie dostosowywać wygląd interfejsu na podstawie danych z czujnika światła. W zależności od oświetlenia, interfejs może przyjąć jasny lub ciemny motyw, co zapewni optymalne wrażenia podczas korzystania z aplikacji w różnych warunkach oświetleniowych.[1]

Wykorzystanie aparatu: Użytkownik zyska możliwość korzystania z aparatu w celu: - Robienia zdjęć zamówionych produktów, co daje szansę na dokumentację i dzielenie się doświadczeniami ze znajomymi. - Skanowania kodów QR związanych z promocjami, pozwalając na szybkie uzyskiwanie zniżek i innych korzyści. - Używania aparatu do skanowania menu w restauracjach, co ułatwi składanie zamówień bez konieczności fizycznego kontaktu z kartą dań.[2]

Zamawianie pizzy: Kluczową funkcjonalnością aplikacji będzie możliwość efektywnego zamawiania pizzy. Użytkownik będzie mógł przeglądać pełne menu, personalizować swoje zamówienia, np. wybierając różne składniki, oraz śledzić status realizacji zamówienia na bieżąco. Dzięki prostemu i intuicyjnemu interfejsowi, składanie zamówienia będzie szybkie i komfortowe.

1.3. Wymagania niefunkcjonalne

Bezpieczeństwo: Aplikacja musi zapewniać wysoki poziom bezpieczeństwa, zwłaszcza przy korzystaniu z biometrii oraz przetwarzaniu danych użytkowników.

Wszelkie dane osobowe będą przechowywane w sposób zaszyfrowany, a komunikacja z serwerami będzie odbywać się z użyciem bezpiecznych protokołów (np. HTTPS).

Responsywność: Interfejs aplikacji musi być zoptymalizowany pod kątem różnych warunków oświetleniowych i urządzeń mobilnych. Aplikacja powinna być dostosowana do ekranów o różnej wielkości, co zapewni wygodę korzystania zarówno na telefonach, jak i tabletach.

Wydajność: Aplikacja powinna działać płynnie na różnych urządzeniach mobilnych, niezależnie od ich specyfikacji sprzętowej. Niezależnie od obciążenia systemu, użytkownik powinien doświadczać szybkiego działania aplikacji, co przyczyni się do lepszego komfortu użytkowania.

1.4. Grupa docelowa

Aplikacja skierowana jest do szerokiego kręgu użytkowników, w tym osób młodszych, średniego wieku oraz seniorów, którzy zamawiają jedzenie przez smartfony. Docelową grupą odbiorców są osoby poszukujące nowoczesnych, wygodnych i bezpiecznych sposobów na zarządzanie zamówieniami. Aplikacja umożliwi także szybkie personalizowanie zamówień, co z pewnością spotka się z zainteresowaniem ze strony użytkowników ceniących indywidualne podejście do gastronomii. Oprócz tego, celem jest przyciągnięcie entuzjastów nowinek technologicznych, którzy cenią sobie innowacyjne rozwiązania w codziennych czynnościach.

2. Określenie wymagań szczegółowych

2.1. Funkcje aplikacji

2.1.1. Logowanie biometryczne

Opis: Aplikacja wykorzysta czujnik linii papilarnych do logowania użytkowników.
Wymagania funkcjonalne: Użytkownik może zalogować się za pomocą odcisku palca. System przechowuje dane biometryczne lokalnie w urządzeniu, a nie w bazie danych Firebase. Firebase służy do przechowywania danych uwierzytelniających użytkownika (np. e-mail, hasło) w przypadku alternatywnego logowania.

2.1.2. Interfejs dostosowujący się do oświetlenia

Opis: Interfejs użytkownika automatycznie dostosowuje jasność i kolorystykę w zależności od warunków oświetlenia.
Wymagania funkcjonalne: Aplikacja pobiera dane z czujnika światła w urządzeniu. Na podstawie natężenia światła zmienia się motyw aplikacji (ciemny/lekki motyw).

2.1.3. Zamawianie pizzy

Opis: Użytkownik może przeglądać menu, tworzyć zamówienia i dostosowywać je według własnych preferencji.
Wymagania funkcjonalne: Menu pizzy przechowywane w Firebase, z możliwością edytowania przez administratora. Użytkownik może dodawać pizze do koszyka, dostosowywać składniki (np. wybór dodatkowych składników) i składać zamówienie. Firebase przechowuje szczegóły zamówień (produkty, ilość, czas zamówienia).

2.1.4. Użycie aparatu

Opis: Aparat będzie wykorzystywany do robienia zdjęć zamówień lub skanowania kodów QR.
Wymagania funkcjonalne: Użytkownik może zrobić zdjęcie zamówionej pizzy i dodać je do swojej historii zamówień. Użytkownik może skanować kody QR, aby uzyskać rabaty lub promocje.

2.2. Technologie

2.2.1. Platforma Mobilna Android (C#, .NET MAUI)

W ramach aplikacji mobilnej rozwijanej na platformie Android przy użyciu C# [3] i .NET MAUI [4], dostęp do różnych czujników urządzeń jest istotnym elementem wpływającym na funkcjonalność oraz interaktywność aplikacji. W poniższej sekcji przedstawiono trzy kluczowe czujniki: linii papilarnych, światła oraz aparat.

2.2.2. Czujnik linii papilarnych

Czujnik linii papilarnych [5] umożliwia bezpieczną autoryzację użytkowników. Dzięki niemu aplikacja może korzystać z biometrycznego uwierzytelniania, co znacząco podnosi poziom bezpieczeństwa. Wykorzystanie tej technologii pozwala na:

- **Logowanie do aplikacji:** Użytkownicy mogą logować się do aplikacji za pomocą odcisku palca, co eliminuje konieczność pamiętania haseł.
- **Autoryzacja transakcji:** Biometryczne potwierdzenie działań w aplikacji, takich jak składanie zamówień, co zwiększa bezpieczeństwo.

2.2.3. Czujnik światła

Czujnik światła [1] mierzy natężenie oświetlenia w otoczeniu, co pozwala na automatyczne dostosowanie ustawień aplikacji. Jego zastosowania obejmują:

- **Dostosowanie jasności ekranu:** Aplikacja może automatycznie regulować jasność wyświetlacza, co poprawia komfort użytkowania w różnych warunkach oświetleniowych.
- **Zmiana trybu kolorystycznego:** Przełączanie między trybem jasnym a ciemnym, co jest bardziej przyjazne dla oczu w ciemnych warunkach.

2.2.4. Aparat

Aparat [2] urządzenia odgrywa kluczową rolę w interakcji użytkownika z aplikacją. Dzięki niemu możliwe jest:

- **Skanowanie kodów QR:** Użytkownicy mogą szybko uzyskiwać dostęp do zamówień lub produktów przez skanowanie kodów QR, co przyspiesza proces zakupowy.

- **Robienie zdjęć:** Możliwość dodawania zdjęć potraw do zamówienia lub do recenzji, co zwiększa interaktywność aplikacji.

2.3. Wymagania techniczne

2.3.1. MySQL jako baza danych

Opis: MySQL [6] zostanie użyty do przechowywania danych aplikacji.
Wymagania funkcjonalne:

- **Przechowywanie danych użytkowników:** MySQL będzie zarządzać loginami użytkowników, przechowując dane takie jak adres e-mail, hasło (zaszyfrowane za pomocą algorytmu bcrypt lub innego bezpiecznego standardu) oraz inne informacje profilowe.
- **Przechowywanie zamówień:** MySQL będzie przechowywać dane dotyczące zamówień, w tym historię zamówień, aktualne zamówienia oraz szczegóły dotyczące produktów w zamówieniu (np. ilość, cena, czas zamówienia).
- **Przechowywanie danych produktów:** Informacje o produktach, w tym nazwy, opisy, ceny, dostępne promocje oraz opcje dostosowywania składników, będą przechowywane w tabelach MySQL.

2.3.2. Responsywność i wydajność

Opis: Aplikacja musi działać płynnie i być responsywna na różnych urządzeniach.
Wymagania funkcjonalne:

- Aplikacja powinna szybko reagować na dane z czujnika światła i płynnie zmieniać motywy.
- Wydajność bazy danych MySQL musi być zoptymalizowana poprzez odpowiednie struktury tabel, indeksowanie oraz zapytania SQL o niskim koszcie czasowym.

2.3.3. Bezpieczeństwo

Opis: Priorytetem jest zapewnienie wysokiego poziomu bezpieczeństwa, zwłaszcza przy logowaniu biometrycznym i przetwarzaniu danych osobowych.
Wymagania niefunkcjonalne:

- Wszystkie dane użytkowników przechowywane w MySQL muszą być szyfrowane (np. szyfrowanie danych wrażliwych przed zapisaniem do bazy).

- Autoryzacja biometryczna musi być oparta na standardach bezpieczeństwa mobilnego.

- Mechanizmy weryfikacji tożsamości użytkowników muszą być zintegrowane z aplikacją poprzez odpowiednie procedury zapytań SQL i walidację danych po stronie serwera.

2.4. Wymagania dotyczące interfejsu użytkownika

2.4.1. Prosty i intuicyjny interfejs

Opis: Interfejs powinien być prosty w obsłudze, a jednocześnie estetyczny.
Wymagania funkcjonalne: Użytkownik powinien mieć łatwy dostęp do najważniejszych funkcji, takich jak logowanie, menu pizzy, zamówienia. Dynamiczne zmiany wyglądu interfejsu w zależności od oświetlenia zewnętrznego.

2.4.2. Przyjazne środowisko dla użytkownika

Opis: Aplikacja musi być przyjazna i łatwa w obsłudze nawet dla mniej zaawansowanych technologicznie użytkowników.
Wymagania niefunkcjonalne: Interfejs powinien reagować szybko na wszystkie interakcje użytkownika, bez opóźnień. Powinna istnieć funkcja pomocy, która wyjaśni, jak korzystać z poszczególnych funkcji aplikacji.

2.5. Testowanie i stabilność

2.5.1. Testy jednostkowe

Opis: Aplikacja musi przejść serię testów jednostkowych, które zapewnią poprawne działanie jej funkcji.
Wymagania: Testy logowania biometrycznego. Testy obsługi zamówień i komunikacji z Firebase. Testy dynamicznego zmieniania motywu w zależności od warunków oświetleniowych.

2.5.2. Stabilność działania

Opis: Aplikacja musi działać stabilnie na różnych urządzeniach mobilnych, niezależnie od ich specyfikacji.
Wymagania: Aplikacja nie może się zawieszać ani wyłączać w przypadku intensywnego użytkowania. Działanie aplikacji powinno być płynne przy wolniejszych połączeniach internetowych, z danymi Firebase.

2.6. Wymagania dotyczące zarządzania danymi

2.6.1. Przechowywanie danych lokalnych

Opis: Niektóre dane (np. logi z czujników, historia zamówień) mogą być przechowywane lokalnie na urządzeniu.
Wymagania: Użytkownik powinien mieć możliwość przeglądania swojej historii zamówień, która jest synchronizowana z Firebase. Dane lokalne muszą być synchronizowane z bazą danych Firebase, kiedy urządzenie jest online.

2.6.2. Zarządzanie sesją użytkownika

Opis: Aplikacja musi zarządzać sesją użytkownika, aby zapobiec przypadkowemu wylogowaniu.
Wymagania: Sesje muszą być przechowywane bezpiecznie w pamięci urządzenia, z możliwością automatycznego odnowienia sesji. Po wygaśnięciu sesji użytkownik powinien być automatycznie wylogowany, a dalszy dostęp wymaga ponownego uwierzytelnienia.

2.7. Dane wejściowe i obsługa przycisków

2.7.1. Opis elementów interfejsu i przycisków

Przycisk "Zaloguj":
Po kliknięciu otwiera się ekran logowania, gdzie użytkownik może skorzystać z biometrii (czytnik linii papilarnych) lub zalogować się za pomocą e-maila i hasła.
Przycisk "Zamów":
Otwiera menu pizzy z możliwością wyboru pizzy i składników dodatkowych.
Przycisk "Dodaj do koszyka":
Dodaje wybraną pizzę do koszyka. Po kliknięciu wyświetla się liczba produktów w koszyku oraz jego zawartość.

Przycisk "Złóż zamówienie":
Finalizuje zamówienie. Po kliknięciu wyświetla okno podsumowania zamówienia i umożliwia wybór metody płatności.
Przycisk "Aparat":
Uruchamia aparat w celu zrobienia zdjęcia zamówienia lub zeskanowania kodu QR, który daje dostęp do promocji.

2.7.2. Czujniki i ich działanie

Czytnik linii papilarnych:
Służy do bezpiecznego logowania użytkowników. Po przyłożeniu palca do czytnika system uwierzytelnia użytkownika, a po poprawnej weryfikacji przekierowuje go do głównego ekranu aplikacji.
Czujnik światła:
Dynamicznie zmienia jasność i kolorystykę aplikacji (tryb jasny/ciemny) w zależności od otoczenia. Jeśli poziom światła jest niski, aplikacja automatycznie przełącza się w tryb ciemny, a przy jasnym oświetleniu w tryb jasny.
Aparat:
Umożliwia robienie zdjęć zamówionych produktów, które mogą być zapisywane w historii użytkownika. Skanuje kody QR, dodając zniżki do zamówienia.

2.8. Zachowanie aplikacji po kliknięciu elementów

2.8.1. Ekran logowania

Po kliknięciu "Zaloguj":
Użytkownik może wybrać metodę logowania: biometryczne (odcisk palca) lub za pomocą hasła.
Przy logowaniu biometrycznym, po udanym skanie użytkownik jest przekierowany do głównego menu. Jeśli logowanie nie powiedzie się, wyświetlany jest komunikat o błędzie i opcja powtórzenia próby lub wybrania innej metody logowania.

2.8.2. Proces składania zamówienia

Po kliknięciu "Zamów":
Użytkownik widzi menu pizzy z możliwością wyboru i personalizacji zamówienia (rozmiar pizzy, dodatki). Po wybraniu pizzy i kliknięciu "Dodaj do koszyka", produkt trafia do koszyka, a użytkownik widzi jego zawartość.
Po kliknięciu "Złóż zamówienie":

Użytkownik potwierdza zamówienie. Na ekranie pojawia się komunikat o statusie zamówienia ("W trakcie realizacji").

2.8.3. Użycie aparatu

Po kliknięciu przycisku "Aparat":
Użytkownik może zrobić zdjęcie zamówionego produktu lub zeskanować kod QR. Po wykonaniu zdjęcia użytkownik może dodać je do historii swojego zamówienia. Skanowanie kodu QR automatycznie dodaje zniżkę do zamówienia.

2.9. Możliwości dalszego rozwoju oprogramowania

Rozszerzenie obsługi zamówień:
Dodanie możliwości śledzenia kuriera z wykorzystaniem GPS. Wprowadzenie systemu oceny zamówienia przez użytkowników.
Rozbudowa funkcji promocji:
Automatyczne dodawanie zniżek na podstawie analizy poprzednich zamówień (personalizacja promocji). Integracja z systemami lojalnościowymi
. Integracja z płatnościami mobilnymi:
Rozszerzenie o metody płatności.
Multiplatformowość:
Rozszerzenie aplikacji na inne platformy mobilne (iOS).

2.10. Zachowanie aplikacji w niepożądanych sytuacjach

2.10.1. Utrata połączenia z internetem

Aplikacja wyświetla komunikat o braku połączenia internetowego i pozwala użytkownikowi kontynuować zamówienie w trybie offline (np. przeglądanie menu). Zamówienie jest zapisywane lokalnie, a przywrócenie połączenia internetowego automatycznie synchronizuje dane z Firebase.

2.10.2. Nieudana weryfikacja biometryczna

W przypadku niepowodzenia logowania biometrycznego, aplikacja wyświetla komunikat o błędzie i umożliwia próbę ponownego logowania lub użycie alternatywnej metody (e-mail/hasło).

2.10.3. Niski poziom baterii w urządzeniu

Aplikacja zmienia ustawienia wyświetlania (zmniejsza jasność) w celu oszczędzania energii. Użytkownik jest informowany o niskim poziomie baterii i sugeruje się podłączenie ładowarki przed kontynuacją zamówienia.

2.10.4. Błąd podczas składania zamówienia

Jeśli zamówienie nie zostanie poprawnie złożone (np. błąd połączenia z Firebase), aplikacja wyświetla komunikat z możliwością powtórzenia próby lub zapisania zamówienia lokalnie w celu późniejszej synchronizacji.

3. Projektowanie

3.1. Wykorzystane narzędzia

3.1.1. Visual Studio

Visual Studio to wszechstronne zintegrowane środowisko programistyczne, używane głównie do tworzenia aplikacji w wielu językach, w tym C#. Zawiera zaawansowane narzędzia do pisania kodu, debugowania i testowania.[7]

3.1.2. .Net Maui

.NET MAUI to wszechstronne środowisko programistyczne umożliwiające tworzenie aplikacji wieloplatformowych na Androida, iOS, Windows i macOS za pomocą jednego wspólnego kodu w języku C#. Oferuje zaawansowane narzędzia do projektowania interfejsów, pisania kodu, debugowania i testowania, zapewniając jednocześnie dostęp do natywnych funkcji systemów operacyjnych.[8]

3.1.3. Github

GitHub to wszechstronna platforma dla programistów, umożliwiająca hostowanie, zarządzanie oraz współpracę nad projektami opartymi na systemie kontroli wersji Git. Oferuje narzędzia do śledzenia zmian w kodzie, zarządzania zgłoszeniami (issues), przeprowadzania przeglądów kodu (code reviews) oraz automatyzacji procesów za pomocą GitHub Actions. Jest szeroko wykorzystywany w projektach open source i komercyjnych.[9]

3.1.4. Figma

Figma to nowoczesne narzędzie do projektowania interfejsów użytkownika i prototypowania, działające w przeglądarce internetowej. Umożliwia zespołową pracę w czasie rzeczywistym, dzięki czemu projektanci, programiści i interesariusze mogą współpracować nad projektami w jednym miejscu. Figma oferuje funkcje takie jak tworzenie wektorowych elementów graficznych, zarządzanie komponentami oraz integrację z innymi narzędziami, co czyni ją idealnym rozwiązaniem dla projektowania interfejsów aplikacji i stron internetowych.[10]

3.2. Działanie aplikacji

3.2.1. Strona wyboru logowanie, rejestracja

Na stronie logowania użytkownik ma dostęp do dwóch przycisków, które umożliwiają mu wybór między zalogowaniem się a rejestracją w aplikacji. Przyciski te są wyraźnie widoczne i intuicyjne, co pozwala użytkownikowi szybko podjąć odpowiednią akcję w zależności od tego, czy posiada już konto, czy też musi je dopiero utworzyć. Dzięki temu proces rozpoczęcia korzystania z aplikacji jest prosty i wygodny.

Rys. 3.1. Głowna strona logowania

3.2.2. Strona logowania

Na stronie logowania użytkownik ma do dyspozycji dwa wyraźnie widoczne przyciski, które umożliwiają mu dokonanie wyboru pomiędzy zalogowaniem się a rejestracją w aplikacji. Te intuicyjnie zaprojektowane przyciski są łatwe do zidentyfikowania, co pozwala użytkownikowi szybko podjąć decyzję w zależności od tego, czy już posiada konto, czy też chce je dopiero utworzyć.

Taki układ sprawia, że proces rozpoczęcia korzystania z aplikacji jest niezwykle prosty i wygodny. Dzięki temu użytkownik może bez zbędnych trudności rozpocząć swoją przygodę z aplikacją, bez względu na to, na jakim etapie jest – czy jest nowym użytkownikiem, czy wraca do swojego konta. To przyjazne podejście do interfejsu zapewnia pozytywne doświadczenie już od samego początku.

Rys. 3.2. Strona do zalogowania się

3.2.3. Strona rejestracji

Na stronie rejestracji aplikacji "PizzaUtopia"dominuje jasny, pomarańczowy kolor tła, co nadaje jej przyjazny i energetyzujący charakter. Na górze znajduje się logo w formie kawałka pizzy z czarnymi kropkami, co natychmiast kojarzy się z tematyką aplikacji.

Pod logo znajduje się tytuł "Rejestracja", wyraźnie wskazujący cel strony. Formularz rejestracyjny składa się z trzech pól: pierwsze do wprowadzenia hasła, drugie do powtórzenia hasła oraz trzeci przycisk "Zarejestruj się"w intensywnym pomarańczowym kolorze, co czyni go bardzo widocznym i zachęcającym do kliknięcia. Dodatkowo, w polach hasła znajduje się ikona, umożliwiająca użytkownikom ukrycie lub pokazanie wprowadzanego hasła, co zwiększa komfort użytkowania.

Całość jest zaprojektowana w sposób prosty i intuicyjny, co sprawia, że proces rejestracji jest szybki i nieskomplikowany. Użytkownicy od razu wiedzą, jakie informacje muszą wprowadzić, co ułatwia im rozpoczęcie korzystania z aplikacji.

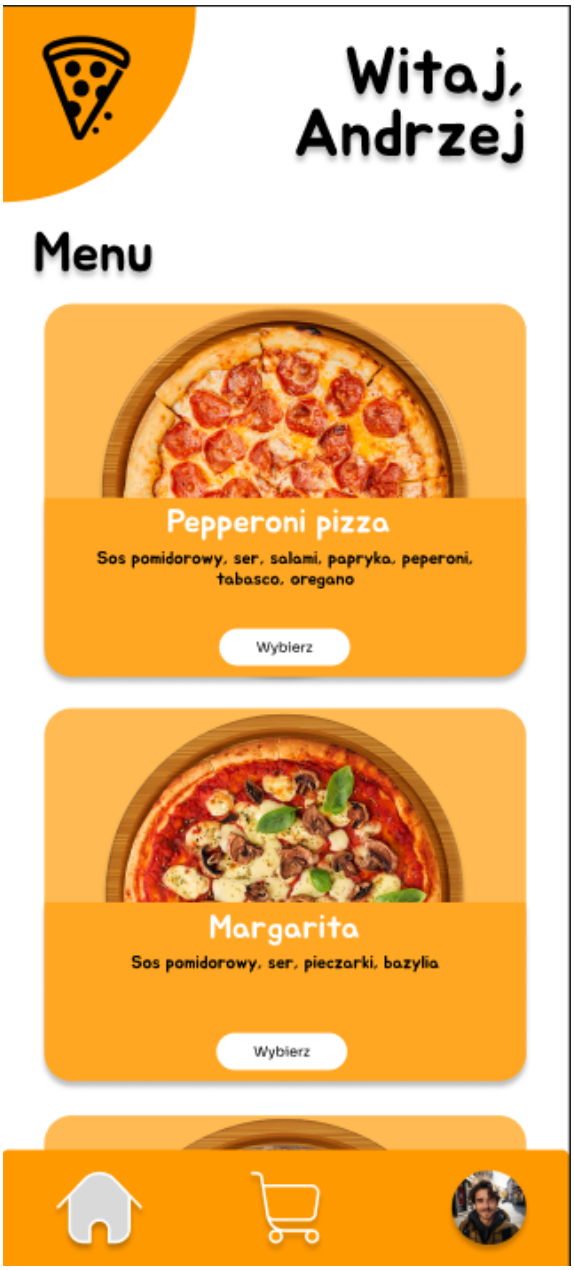
Rys. 3.3. Strona do rejestracji

3.2.4. Strona głowna

Na ekranie głównym aplikacji "PizzaUtopia"witany jest użytkownik o imieniu Andrzej, co nadaje personalny charakter interfejsowi. U góry ekranu znajduje się kolorowy graficzny element w kształcie kawałka pizzy, który od razu przyciąga uwagę i podkreśla tematykę aplikacji.

Pod powitanie umieszczony jest nagłówek "Menu", który jasno wskazuje, co jest głównym celem tej sekcji. Poniżej wyświetlają się dwa popularne rodzaje pizzy. Pierwsza to Pepperoni pizza, z opisem składników: sos pomidorowy, ser, salami, papryka, pepperoni oraz oregano. Druga to Margarita, z prostym i klasycznym składem: sos pomidorowy, ser, pieczarki i bazylia. Obie propozycje są reprezentowane na kolorowych obrazkach umieszczonych w okrągłych ramkach, co nadaje interfejsowi przyjemny wygląd.

Każda pozycja w menu zawiera również przycisk "Wyświetl", który umożliwia użytkownikowi uzyskanie dodatkowych informacji na temat danego produktu. Na dole ekranu znajdują się również ikony reprezentujące różne funkcje, takie jak powrót do ekranu głównego oraz ikonka koszyka, co ułatwia nawigację i korzystanie z aplikacji. Całość jest zaprojektowana w sposób intuicyjny i estetyczny, co zachęca do eksploracji dostępnych opcji.



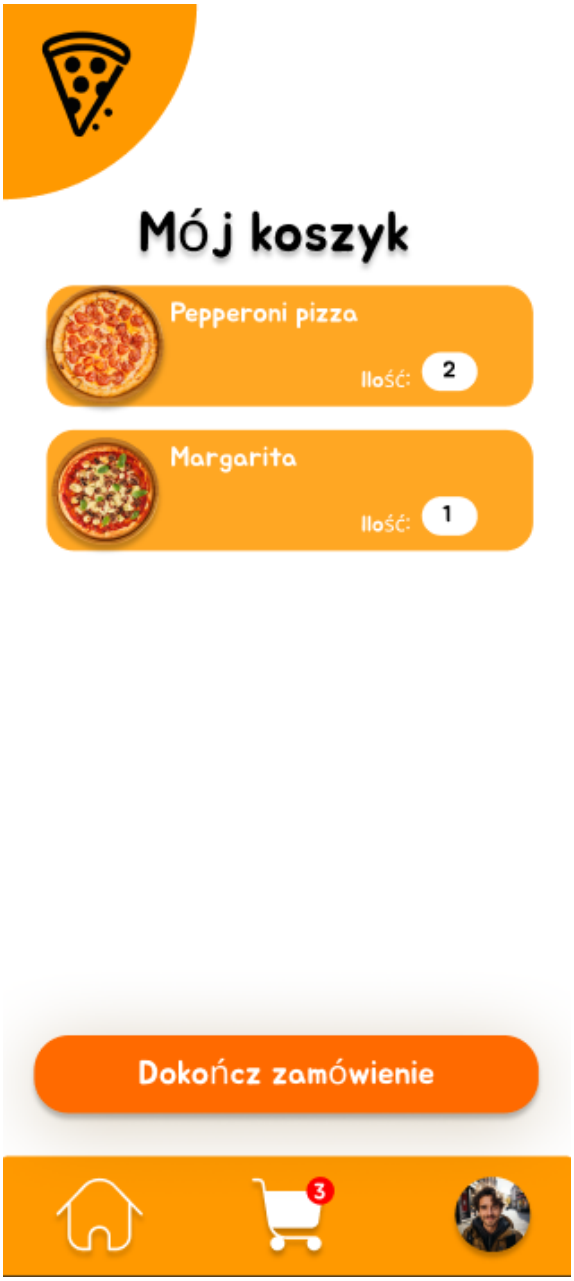
Rys. 3.4. Strona głowna

3.2.5. Koszyk

Na ekranie "Mój koszyk"w aplikacji "PizzaUtopia"użytkownik zyskuje przegląd swojego zamówienia. U góry widoczny jest nagłówek "Mój koszyk", co jasno informuje o celach tej sekcji.

W koszyku znajdują się dwie pozycje – Pepperoni pizza oraz Margarita. Przy każdej pizzy podana jest ilość zamówionych sztuk: dwie Pepperoni i jedna Margarita. Obrazy pizzy są wyraźne i kolorowe, co nadaje przyjemny wygląd.

Na dole ekranu znajduje się efektowny przycisk "Dokończ zamówienie"w intensywnym pomarańczowym kolorze, co zachęca użytkownika do finalizacji zakupu. Obok przycisku widoczne są ikony reprezentujące dodatkowe funkcje, w tym ikona powrotu do ekranu głównego oraz koszyka, co ułatwia nawigację. Całość jest intuicyjna i estetyczna, co sprawia, że przeglądanie i finalizowanie zamówienia jest wygodne i przyjemne.



Rys. 3.5. Koszyk

3.2.6. Profil

Na ekranie profilu użytkownika w aplikacji "PizzaUtopia"widnieje powitanie dla Andrzeja Kucyka, z jego imieniem oraz numerem telefonu umieszczonym tuż pod nazwiskiem. U góry znajduje się także zdjęcie profilowe, które nadaje interfejsowi osobisty charakter.

Zaraz pod danymi użytkownika znajduje się przycisk "Edytuj profil", co umożliwia Andrzejowi aktualizację swoich informacji. Poniżej znajdują się sekcje podzielone na kategorie, zaczynając od "Ustawienia". W tej sekcji użytkownik ma możliwość zmiany hasła, aktywacji biometrii oraz przeglądania widoku swojego profilu.

Kolejna sekcja zatytułowana "O nas"zawiera dodatkowe informacje, takie jak FAQ, polityka prywatności oraz kontakt, co ułatwia użytkownikowi dostęp do istotnych danych.

Na dole ekranu umieszczone są inne opcje, w tym przycisk "Udostępnij", co zachęca do dzielenia się aplikacją. Całość jest estetycznie zaprojektowana w jasnych kolorach, co sprawia, że nawigacja jest intuicyjna i przyjemna.

Rys. 3.6. Profil

3.2.7. Edycja profilu

Na ekranie edycji profilu w aplikacji "PizzaUtopia"użytkownik ma możliwość aktualizacji swoich danych. U góry ekranu znajduje się nagłówek "Edycja profilu", co jasno wskazuje na cel tej sekcji.

Centralnym punktem jest okrągłe zdjęcie profilowe Andrzeja Kucyka, które nadaje osobisty charakter. Pod zdjęciem umieszczone są dwa pola tekstowe: pierwsze z imieniem i nazwiskiem, a drugie z numerem telefonu. W polach znajdują się już wprowadzone dane, co ułatwia ich edytowanie.

Na dole ekranu znajduje się przycisk "Zapisz zmiany"w intensywnym pomarańczowym kolorze, co przyciąga uwagę i zachęca użytkownika do kliknięcia, gdy skończy wprowadzać zmiany.

Dodatkowo, nawigacja jest ułatwiona dzięki ikonkom na dole ekranu, które pozwalają szybko wrócić do ekranu głównego lub do koszyka. Całość jest estetyczna i przejrzysta, co sprawia, że edytowanie profilu jest szybkie i wygodne.

Rys. 3.7. Edycja profilu

3.2.8. Zmiana hasła

Na ekranie zmiany hasła w aplikacji "PizzaUtopia"użytkownik ma możliwość zaktualizowania swojego hasła w prosty i przejrzysty sposób. U góry ekranu znajduje się nagłówek "Zmień hasło", co jasno informuje o celu tej sekcji.

Formularz składa się z trzech pól: pierwsze pole przeznaczone jest na wprowadzenie obecnego hasła, drugie na nowe hasło, a trzecie na potwierdzenie nowego hasła. Każde pole zawiera ikonę, umożliwiającą użytkownikowi ukrycie lub pokazanie wprowadzanych znaków, co zwiększa komfort wpisywania haseł.

Na dole ekranu znajduje się wyraźny przycisk "Zapisz zmiany"w intensywnym pomarańczowym kolorze, co czyni go atrakcyjnym i zachęca do kliknięcia. W lewym górnym rogu znajduje się ikona powrotu do poprzedniego ekranu, a na dole ekranu umieszczone są ikonki umożliwiające szybki dostęp do ekranu głównego oraz koszyka.

Całość jest minimalistycznie zaprojektowana, co sprawia, że proces zmiany hasła jest wygodny i intuicyjny.

Rys. 3.8. Zmiana hasła

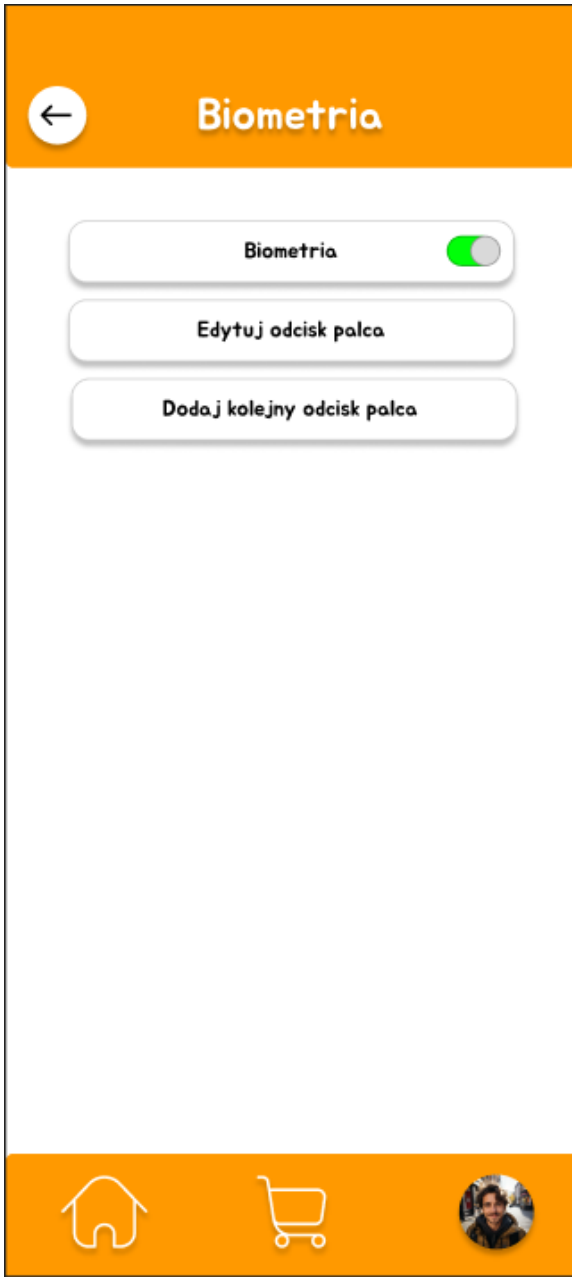
3.2.9. Biometria

Na ekranie ustawień biometrii w aplikacji "PizzaUtopia"użytkownik ma możliwość zarządzania funkcjami zabezpieczeń. U góry ekranu znajduje się wyraźny nagłówek "Biometria", co jasno określa, o jaką sekcję chodzi.

W centralnej części ekranu znajduje się przełącznik, który umożliwia użytkownikowi włączenie lub wyłączenie funkcji biometrycznych. Obok przełącznika widnieje napis "Biometria", co ułatwia identyfikację opcji.

Poniżej znajdują się dwa przyciski: "Edytuj odcisk palca"oraz "Dodaj kolejny odcisk palca". Te opcje pozwalają użytkownikowi na zarządzanie zarejestrowanymi odciskami palców w prosty sposób.

Na dole ekranu umieszczone są klasyczne ikony nawigacyjne: przycisk powrotu do ekranu głównego oraz ikona koszyka, co zapewnia łatwy dostęp do innych funkcji aplikacji. Całość jest estetyczna i minimalistyczna, co wpływa na wygodę użytkowania.



Rys. 3.9. Biometria

3.2.10. Wyglad

Na ekranie ustawień wyglądu w aplikacji "PizzaUtopia"użytkownik może dostosować interfejs do swoich preferencji. U góry znajduje się nagłówek "Wygląd", co jasno określa tematykę tej sekcji.

W centralnej części ekranu znajdują się dwa przełączniki. Pierwszy dotyczy "Automatycznego motywu", który użytkownik może włączyć lub wyłączyć, co pozwala na dostosowanie aplikacji do aktualnych warunków oświetleniowych. Drugi przełącznik umożliwia włączenie lub wyłączenie "Ciemnego motywu", co może być bardziej komfortowe dla oczu w słabym świetle.

Każdy przełącznik jest dobrze oznaczony, a ich funkcjonalność jest intuicyjna i łatwa do zrozumienia. Na dole ekranu znajdują się klasyczne ikony nawigacyjne: przycisk powrotu do ekranu głównego oraz ikona koszyka, co ułatwia poruszanie się po aplikacji.

Całość interfejsu jest estetyczna, z jasnymi kolorami i przejrzystym układem, co sprawia, że zmiana ustawień jest szybka i przyjemna.

Rys. 3.10. Wyglad

3.2.11. Finalizacja zamowienia

Na ekranie "Moje zamówienie"w aplikacji "PizzaUtopia"użytkownik widzi podsumowanie swojego zamówienia. U góry ekranu znajduje się nagłówek "Moje zamówienie", co jasno informuje o zawartości sekcji.

Pod nagłówkiem widnieją szczegóły zamówienia: użytkownik zamówił dwie pizze Pepperoni oraz jedną pizzę Margheritę. Informacje te są wyraźne i łatwe do odczytania.

Na dole ekranu znajdują się dwa kluczowe przyciski. Pierwszy z nich, "Wróć do koszyka", umożliwia użytkownikowi powrót do przeglądania i edytowania zamówienia. Drugi przycisk, "Dokończ zamówienie", jest wyróżniony na pomarańczowo, co zwraca na siebie uwagę i zachęca do finalizacji zakupu.

W dolnej części ekranu są również ikony nawigacyjne, w tym ikona koszyka oraz profil użytkownika. Całość jest estetycznie zaprojektowana, z jasnymi kolorami i prostym układem, co sprawia, że korzystanie z aplikacji jest intuicyjne i komfortowe.

Rys. 3.11. Finalizacja zamowienia

4. Implementacja

4.1. Zarządzanie nawigacją i stylami

Na listingu 1 przedstawiono fragment kodu odpowiedzialny za nawigację użytkownika w przypadku, gdy nie jest zalogowany. Jeśli użytkownik nie jest zalogowany, zostanie automatycznie przekierowany do menu. Dodatkowo załadowane są zewnętrzne style i fonty.

```
@inject NavigationManager NavigationManager
@using PizzaUtopia.Components.Elements.StartScreen

<link href="https://fonts.googleapis.com/css?family=Jua" rel="
    stylesheet">
<link rel="stylesheet" href="Profile.css">

@if (!_isLogedIn)
{
    NavigationManager.NavigateTo("/menu");
}

@code {
  private readonly bool _isLogedIn = false;
}
```

Listing 1. Kod zarządzania nawigacją i stylami

4.2. Formularz logowania

Na listingu 2 przedstawiono kod formularza logowania, który umożliwia użytkownikowi wprowadzenie adresu e-mail i hasła oraz zalogowanie się za pomocą przycisku. Formularz korzysta z pól wejściowych i przycisku zdefiniowanych w aplikacji.

```
<form id="loginForm">
    <!-- Formularz logowania -->
    <!-- Pole do wpisania emaila -->
    <Input InputType="Enums.InputType.Email" placeholder="Wprowad
    e-mail" />

    <!-- Pole do wpisania hasła z możliwością pokazania/ukrycia
    -->
    <div class="password-container">
        <Input InputType="Enums.InputType.Password" placeholder="
   Wprowad hasło" />
```

```
    </div>

    <!-- Przycisk logowania -->
    <Button OnClick="OnMenuClick">Zaloguj si Ź</Button>
</form>
```

Listing 2. Kod formularza logowania

4.3. Formularz zmiany hasła

Na listingu 3 przedstawiono implementację formularza zmiany hasła. Formularz umożliwia użytkownikowi wprowadzenie aktualnego hasła, nowego hasła, oraz potwierdzenie nowego hasła. Dodatkowo zawiera walidację danych wejściowych oraz obsługę zdarzenia przesłania formularza.

```
<EditForm Model="@changePasswordModel" OnValidSubmit="HandleSubmit"
   >
    <DataAnnotationsValidator />
    <ValidationSummary />

    <!-- Pole wprowadzania aktualnego has Ća -->
    <div class="password-container">
        <InputText @bind-Value="changePasswordModel.CurrentPassword
   " Type="password" placeholder="Wprowad  aktualne has Ćo" />
        <ValidationMessage For="@(() => changePasswordModel.
   CurrentPassword)" />
    </div>

    <!-- Pole wprowadzania nowego has Ća -->
    <div class="password-container">
        <InputText @bind-Value="changePasswordModel.NewPassword"
   Type="password" placeholder="Wprowad  nowe has Ćo" />
        <ValidationMessage For="@(() => changePasswordModel.
   NewPassword)" />
    </div>

    <!-- Pole potwierdzania nowego has Ća -->
    <div class="password-container">
        <InputText @bind-Value="changePasswordModel.ConfirmPassword
   " Type="password" placeholder="Potwierd  nowe has Ćo" />
        <ValidationMessage For="@(() => changePasswordModel.
   ConfirmPassword)" />
    </div>

```

```
23      <!-- Przycisk zmiany hasła -->
24      <button class="change-password-btn" type="submit">Zmie  hasło
        </button>
25  </EditForm>
```

Listing 3. Kod formularza zmiany hasła

4.4. Górna sekcja powitania

Na listingu 4 przedstawiono kod górnej sekcji powitania, zawierającej logo oraz przywitanie użytkownika.

```
1  <!-- Górna sekcja powitania -->
2  <div class="header">
3      <div class="logo">
4          <img src="Images/logopizzy.png" alt="Logo Pizzy" />
5      </div>
6      <div class="welcome">
7          <h1>Witaj, Andrzej</h1>
8      </div>
9  </div>
```

Listing 4. Kod górnej sekcji powitania

4.5. Karta pizzy

Na listingu 5 przedstawiono kod karty pizzy, która wyświetla zdjęcie pizzy, jej nazwę, skład oraz przycisk umożliwiający jej wybór.

```
1  <!-- Pizza 1 -->
2  <div class="pizza-card">
3      <img src="Images/pepperoni.png" alt="Pepperoni Pizza" />
4      <div class="pizza-details">
5          <h3>Pepperoni pizza</h3>
6          <p>Sos pomidorowy, ser, salami, papryka, peperoni, tabasco,
       oregano</p>
7          <Button>Wybierz</Button>
8      </div>
9  </div>
```

Listing 5. Kod karty pizzy

4.6. Wyświetlanie zawartości koszyka

Na listingu 6 przedstawiono kod odpowiedzialny za wyświetlanie zawartości koszyka użytkownika, w tym obsługę przypadku pustego koszyka.

```
@if (CartItems.Count == 0)
{
    <p>Tw j koszyk jest pusty.</p>
}
else
{
    @foreach (var item in CartItems)
    {
        <div class="cart-item">
            <img src="@item.ImageUrl" alt="@item.Name" />
            <div class="cart-item-content">
                <div class="cart-item-title">@item.Name</div>
                <div class="cart-item-quantity">Ilo Ż : @item.
   Quantity</div>
            </div>
        </div>
    }
}
```

Listing 6. Kod wyświetlania zawartości koszyka

4.7. Menu ustawień

Na listingu 7 przedstawiono kod sekcji ustawień, która zawiera różne opcje umożliwiające użytkownikowi zarządzanie kontem, wyglądem aplikacji oraz dostęp do informacji o aplikacji.

```
<div class="container">
    <div class="section-title">Ustawienia</div>
    <div class="menu-item" @onclick="RedirectToChangePassword">
   Zmie  has Ćo</div>
    <div class="menu-item" @onclick="RedirectToBiometry">Biometria<
   /div>
    <div class="menu-item" @onclick="RedirectToAppearance">Wygl d<
   /div>

    <div class="section-title">O nas</div>
    <div class="menu-item" @onclick="RedirectToFAQ">FAQ</div>
    <div class="menu-item" @onclick="RedirectToPrivacyPolicy">
   Polityka prywatno Żci</div>
```

```
    <div class="menu-item" @onclick="RedirectToContact">Kontakt</div>

    <div class="section-title">Inne</div>
    <div class="menu-item" @onclick="RedirectToShare">Udostępnij</div>
</div>
```

Listing 7. Kod menu ustawień

4.8. Przełącznik biometrii

Na listingu 8 przedstawiono kod przełącznika biometrii, który pozwala użytkownikowi włączyć lub wyłączyć funkcję biometryczną.

```
<!-- Przełącznik biometrii -->
<div class="toggle-wrapper">
    <div class="toggle-container">
        <label for="biometryToggle" class="toggle-label">Biometria</label>
        <ToggleSwitch OnToggle="HandleToggle" IsChecked="@isBiometryEnabled"></ToggleSwitch>
    </div>
</div>
```

Listing 8. Kod przełącznika biometrii

4.9. Integracja z czujnikiem światła

Na listingu 9 przedstawiono kod klasy `MainPage`, która wykorzystuje usługę `ILightSensorService` do reagowania na zmiany poziomu światła. Kod umożliwia dynamiczną zmianę motywu aplikacji w zależności od warunków oświetleniowych.

```
public partial class MainPage : ContentPage
{
    private ILightSensorService _lightSensorService;
    public MainPage()
    {
        InitializeComponent();
        _lightSensorService = DependencyService.Get<ILightSensorService>();
    }
    protected override void OnAppearing()
    {
```

```
11          base.OnAppearing();
12          if (_lightSensorService != null)
13          {
14              _lightSensorService.LightSensorChanged +=
    OnLightSensorChanged;
15              _lightSensorService.StartListening();
16          }
17      }
18      protected override void OnDisappearing()
19      {
20          base.OnDisappearing();
21          if (_lightSensorService != null)
22          {
23              _lightSensorService.LightSensorChanged -=
    OnLightSensorChanged;
24              _lightSensorService.StopListening();
25          }
26      }
27      private void OnLightSensorChanged(object sender, float
    lightLevel)
28      {
29          if (lightLevel < 10)
30          {
31              Application.Current.UserAppTheme = AppTheme.Dark; //
    Zmiana na ciemny motyw
32          }
33          else
34          {
35              Application.Current.UserAppTheme = AppTheme.Light; //
    Zmiana na jasny motyw
36          }
37      }
38  }
```

Listing 9. Integracja z czujnikiem światła

Opis:

- `ILightSensorService`: interfejs usługi odpowiedzialnej za dostarczanie informacji o zmianach natężenia światła.
- `OnAppearing()` i `OnDisappearing()`: metody cyklu życia strony, odpowiedzialne za rejestrację i wyrejestrowanie zdarzeń związanych z czujnikiem światła.
- `OnLightSensorChanged`: metoda wywoływana przy każdej zmianie natężenia

światła, zmieniająca motyw aplikacji na jasny lub ciemny w zależności od wartości `lightLevel`.

- `DependencyService.Get<ILightSensorService>()`: mechanizm wstrzykiwania zależności, umożliwiający uzyskanie instancji usługi `ILightSensorService`.

Kod dynamicznie dostosowuje interfejs użytkownika aplikacji do warunków oświetleniowych, co zwiększa wygodę korzystania z aplikacji w różnych środowiskach.

4.10. Metoda TakePhoto

Metoda `TakePhoto` przedsawiona na listningu 10 umożliwia użytkownikowi zrobienie zdjęcia za pomocą wbudowanego aparatu w urządzeniu i zapisanie go w pamięci lokalnej. Wykorzystuje bibliotekę `MediaPicker` do interakcji z kamerą.

```
private async Task TakePhoto()
{
    try
    {
        var photo = await MediaPicker.CapturePhotoAsync();
        if (photo != null)
        {
            var stream = await photo.OpenReadAsync();
            profilePicture = await SavePhotoAsync(stream);
            StateHasChanged(); // Od Żwie  komponent po zapisaniu
   zdj Źcia
        }
    }
    catch (Exception ex)
    {
        // Obs Ćuga  b Ć Źd  w
    }
}
```

Listing 10. Metoda `TakePhoto`

Opis:

- Zmienna `photo`: przechowuje uchwyt do zdjęcia przechwyconego za pomocą kamery.

- `MediaPicker.CapturePhotoAsync()`: metoda do uruchomienia aparatu i przechwycenia zdjęcia.

- `SavePhotoAsync(Stream stream)`: zapisuje przekazany strumień danych obrazu w lokalnym katalogu aplikacji.
- `StateHasChanged()`: powoduje odświeżenie komponentu po zapisaniu zdjęcia.

```
.container {
    width: 100%;
    max-width: 400px;
    text-align: center;
    padding: 20px;
    border-radius: 10px;
}

/* Logo and Title */
.logo-container {
    margin-bottom: 50px;
}

.logo {
    width: 250px;
    height: 250px;
    margin-bottom: 10px;
}
```

Listing 11. Kod CSS dla stylizacji formularza logowania z logo

W listingu 11 przedstawiono kod CSS odpowiedzialny za stylizację formularza logowania, szczególnie w odniesieniu do wyświetlania logo. Kluczowe fragmenty kodu to:

- **Container:** Sekcja `.container` ustawia szerokość formularza na 100% z maksymalną szerokością 400px, wyśrodkowując zawartość i nadając jej zaokrąglone rogi. Dodatkowo wprowadza przestrzeń wewnętrzną (padding) dla estetyki.
- **Logo i tytuł:** W `.logo-container` dodano margines na dole, aby oddzielić logo od innych elementów formularza. Logo w `.logo` ma wymiary 250px na 250px i jest odpowiednio wyśrodkowane w formularzu, z marginesem u dołu.

Kod ten ma na celu estetyczne i funkcjonalne wyświetlenie logo oraz dostosowanie formularza do różnych rozmiarów ekranów.

4.11. Strona "Dziękujemy za zamówienie"w Blazor

W poniższym listingu przedstawiono kod strony dziękczynnej w aplikacji Blazor. Strona ta informuje użytkownika o pomyślnym złożeniu zamówienia oraz umożliwia

powrót do menu:

```
@page "/order"
@inject NavigationManager NavigationManager

<link href="https://fonts.googleapis.com/css?family=Jua" rel="
    stylesheet">
<link rel="stylesheet" href="Profile.css">

<div class="thank-you-container">
    <h1>Dzi Źkujemy za zam wienie!</h1>
    <p>Twoje zam wienie zosta Ćo pomy Żlnie z Ćo one. Wkr tce je
    przygotujemy.</p>
    <button class="menu-button" @onclick="OnMenuClick">Powr t do
    menu</button>
</div>

@code {
    /// <summary>
    /// Navigates back to the menu page.
    /// </summary>
    private void OnMenuClick()
    {
        NavigationManager.NavigateTo("/menu");
    }
}
```

Listing 12. Kod strony "Dziękujemy za zamówienie"w Blazor

Kod ten umożliwia użytkownikowi powrót do menu za pomocą przycisku, którego kliknięcie wywołuje metodę `OnMenuClick`, przekierowując użytkownika na stronę menu aplikacji.

4.12. Metoda SavePhotoAsync

Metoda `SavePhotoAsync` przedsawiona na listningu 13 zajmuje się zapisaniem zdjęcia w lokalnej pamięci urządzenia. Używa ścieżki katalogu aplikacji (`AppDataDirectory`) do przechowywania plików.

```
private async Task<string> SavePhotoAsync(Stream stream)
{
    var filePath = Path.Combine(FileSystem.AppDataDirectory, "
    profile.jpg");
    using (var fileStream = new FileStream(filePath, FileMode.
    Create, FileAccess.Write))
    {
```

```
        await stream.CopyToAsync(fileStream);
    }
    return filePath;
}
```

Listing 13. Metoda `SavePhotoAsync`

Opis:

- Argument `stream`: strumień danych obrazu przechwyconego z aparatu.
- `FileSystem.AppDataDirectory`: zapewnia ścieżkę do katalogu danych aplikacji, gdzie zapisane zostanie zdjęcie.
- `FileStream`: używany do tworzenia pliku i zapisywania danych obrazu.
- Zwraca ścieżkę do zapisanego pliku, co umożliwia późniejsze wykorzystanie tego zdjęcia w aplikacji.

4.13. Przyciski zarządzania odciskami palców

Na listingu 14 przedstawiono kod przycisków do edytowania oraz dodawania odcisków palców.

```
<div class="button-container">
    <button class="action-button" @onclick="EditFingerprint">Edytuj
    odcisk palca</button>
    <button class="action-button" @onclick="AddFingerprint">Dodaj
   kolejny odcisk palca</button>
</div>
```

Listing 14. Kod przycisków zarządzania odciskami palców

4.14. Zarządzanie nawigacją i stanem biometrii

Na listingu 15 przedstawiono kod odpowiedzialny za zarządzanie stanem biometrii oraz nawigację w aplikacji.

```
    // Zmienna do zarz dzania stanem prze Ć cznika biometrii
    private bool isBiometryEnabled = false;

    private void NavigateBack()
    {
        Navigation.NavigateTo("/profile");
    }
```

```
8
9       private void RedirectToMenu() => Navigation.NavigateTo("/menu")
        ;
10      private void RedirectToCart() => Navigation.NavigateTo("/cart")
        ;
11      private void RedirectToProfile() => Navigation.NavigateTo("/
        profile");
```

Listing 15. Kod zarządzania stanem biometrii i nawigacją

4.15. Nagłówek z przyciskiem powrotu i tytułem

Na listingu 16 przedstawiono kod nagłówka zawierającego przycisk powrotu oraz tytuł strony.

```
1  <!-- Nag Ć wek z przyciskiem powrotu i tytu Ćem -->
2  <div class="header">
3      <button class="back-button" @onclick="NavigateBack">
4          <img src="Images/back-arrow.png" alt="Powr t" />
5      </button>
6      <h2 class="page-title">Wygl d</h2>
7  </div>
```

Listing 16. Kod nagłówka z przyciskiem powrotu i tytułem

```
1  .container {
2      width: 100%;
3      max-width: 400px;
4      text-align: center;
5      padding: 20px;
6      border-radius: 10px;
7  }
8
9  /* Logo and Title */
10 .logo-container {
11     margin-bottom: 50px;
12 }
13
14 .logo {
15     width: 200px;
16     height: auto;
17     margin-bottom: 10px;
18 }
19
20 /* Password visibility button */
```

```
.password-container {
    position: relative;
}

.show-password {
    position: absolute;
    right: 10px;
    top: 50%;
    transform: translateY(-50%);
    background: none;
    border: none;
    cursor: pointer;
}

.show-password img {
    width: 25px;
    height: auto;
    align-items: center;
}
```

Listing 17. Kod CSS dla stylizacji formularza logowania

W listingu 17 przedstawiono kod CSS, który jest odpowiedzialny za stylizację formularza logowania. Obejmuje on następujące sekcje:

- **Container:** Sekcja `.container` ustala szerokość formularza na 100% z maksymalną szerokością 400px, wyśrodkowując zawartość oraz nadając jej zaokrąglone rogi. Dodatkowo wprowadza odstępy wewnętrzne (padding).
- **Logo i tytuł:** W `.logo-container` ustawiane jest marginesowanie na dole, aby oddzielić logo od reszty formularza. Logo ma szerokość 200px, a jego wysokość jest automatycznie dopasowywana. Dodatkowo stosowane są marginesy, by nadać estetyczny wygląd.
- **Widoczność hasła:** Dla przycisku do pokazania/ukrycia hasła wprowadzono `.password-container`, gdzie przycisk umieszczany jest w prawym górnym rogu pola hasła, a jego pozycjonowanie jest precyzyjnie ustawione za pomocą `absolute` i `transform`. Obrazek przycisku jest skalowany do 25px.

4.16. Ustawienia motywu

Na listingu 18 przedstawiono kod elementów umożliwiających użytkownikowi wybór między automatycznym motywem oraz ciemnym motywem.

```
<div class="settings-wrapper">
    <div class="setting-item">
        <label for="autoThemeToggle" class="setting-label">
    Automatyczny motyw</label>
        <input type="checkbox" id="autoThemeToggle" class="toggle-
    switch" @bind="isAutoThemeEnabled" />
    </div>

    <div class="setting-item">
        <label for="darkThemeToggle" class="setting-label">Ciemny
    motyw</label>
        <input type="checkbox" id="darkThemeToggle" class="toggle-
    switch" @bind="isDarkThemeEnabled" />
    </div>
</div>
```

Listing 18. Kod ustawień motywu

Na listingu 19 przedstawiono fragment kodu CSS, który stylizuje przycisk, nadając mu odpowiedni wygląd i interaktywność. Stylizacja obejmuje szerokość, tło, kolor tekstu, padding, rozmiar czcionki, granice, zaokrąglenie rogów oraz efekty podczas najechania na przycisk.

```
button {
    width: 100%;
    background-color: #e57a00;
    color: white;
    padding: 12px;
    font-size: 18px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
    margin-top: 10px;
}

button:hover {
    background-color: #e57a00;
}
```

Listing 19. Stylizacja przycisku

4.17. Metody nawigacyjne motywu

Na listingu 22 przedstawiono kod metod odpowiedzialnych za nawigację między stronami w aplikacji.

```
1     private void NavigateBack() => Navigation.NavigateTo("/profile
      ");
2     private void RedirectToMenu() => Navigation.NavigateTo("/menu")
      ;
3     private void RedirectToCart() => Navigation.NavigateTo("/cart")
      ;
4     private void RedirectToProfile() => Navigation.NavigateTo("/
      profile");
```

Listing 20. Metody nawigacyjne motywu

4.18. Dolna nawigacja

Na listingu 21 przedstawiono implementację dolnej nawigacji w aplikacji. Składa się ona z trzech przycisków, które odpowiadają za przekierowanie użytkownika do różnych sekcji: strony głównej, koszyka oraz profilu.

```
1  <div class="bottom-nav">
2      <button class="nav-button" @onclick="OnMenuClick">
3          <img src="Images/home-icon.png" alt="Home" />
4      </button>
5      <button class="nav-button" @onclick="OnCartClick">
6          <img src="Images/cart-icon.png" alt="Cart" />
7      </button>
8      <button class="nav-button" @onclick="OnProfileClick">
9          <img src="Images/profile-icon.png" alt="Profile" />
10     </button>
11 </div>
```

Listing 21. Kod dolnej nawigacji

4.19. Metody dolnej nawigacji

Na listingu 22 przedstawiono metody odpowiedzialne za nawigację po aplikacji. Każda z metod przekierowuje użytkownika na odpowiednią stronę: menu, koszyk lub profil.

```
1  private void OnMenuClick()
2  {
3      NavigationManager.NavigateTo("/menu");
```

```
}

private void OnCartClick()
{
    NavigationManager.NavigateTo("/cart");
}

private void OnProfileClick()
{
    NavigationManager.NavigateTo("/profile");
}
```

Listing 22. Metody dolnej nawigacji

Na listingu 23 przedstawiono fragment pliku konfiguracyjnego projektu .NET MAUI, który określa ustawienia dotyczące platformy, debugowania oraz plików w projekcie. Ten plik jest używany przez środowisko MSBuild do kompilacji i debugowania aplikacji na różnych platformach, takich jak Android i Windows.

```

<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="Current" xmlns="http://schemas.microsoft.com
    /developer/msbuild/2003"> <PropertyGroup>
<IsFirstTimeProjectOpen>False</IsFirstTimeProjectOpen> <
    ActiveDebugFramework>net9.0-windows10.0.19041.0</
    ActiveDebugFramework> <ActiveDebugProfile>Windows Machine</
    ActiveDebugProfile> <SelectedPlatformGroup>Emulator</
    SelectedPlatformGroup> <DefaultDevice>pixel_7_-_api_35</
    DefaultDevice> </PropertyGroup> <PropertyGroup Condition="'$(
    Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net9.0-
    android|AnyCPU'">
<DebuggerFlavor>ProjectDebugger</DebuggerFlavor>
</PropertyGroup>
<ItemGroup> <None Update="App.xaml">
<SubType>Designer</SubType> </None>
<None Update="MainPage.xaml">
<SubType>Designer</SubType> </None>
<None Update="Platforms\Windows\App.xaml">
<SubType>Designer</SubType> </None> <None Update="Platforms\Windows
    \Package.appxmanifest">
<SubType>Designer
</SubType>
</None>
</ItemGroup>
```

```
</Project>
```

Listing 23. Plik konfiguracyjny projektu .NET MAUI

4.20. Konfiguracja kontekstu bazy danych w aplikacji

Poniższy kod przedstawia klasę `PizzaUtopiaDbContext`, która konfiguruje połączenie z bazą danych MySQL w aplikacji. Używa ona `DbContext` z Entity Framework do zarządzania danymi użytkowników.

```
namespace PizzaUtopia.Backend.Context;

public class PizzaUtopiaDbContext(IConfiguration configuration) : DbContext
{
    public DbSet<User> Users { get; set; } = null!;

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseMySql(configuration.GetValue<string>("MySQLConnectionString")
            ?? throw new ArgumentNullException("MySQLConnectionString", "Connection string nie mo e by  pusty"),
            ServerVersion.AutoDetect(configuration.GetValue<string>("MySQLConnectionString")
            ?? throw new ArgumentNullException("MySQLConnectionString")));
    }
}
```

Listing 24. Konfiguracja kontekstu bazy danych PizzaUtopiaDbContext

W tym kodzie:

- `DbContext` jest używane do tworzenia klasy kontekstu bazy danych, która zawiera zbiór tabel `DbSet<User>` do przechowywania danych użytkowników.

- Metoda `OnConfiguring` konfiguruje połączenie z bazą danych MySQL, korzystając z łańcucha połączenia pobranego z konfiguracji aplikacji. Jeśli łańcuch połączenia jest pusty, generowany jest wyjątek.

4.21. Klasa LoginDataDto

Poniższy kod przedstawia klasę `LoginDataDto`, która reprezentuje dane logowania użytkownika, zawierające adres e-mail oraz hasło. Klasa ta jest używana do przesyłania danych logowania w aplikacji.

```
namespace PizzaUtopia.Backend.Dto
{
    public class LoginDataDto
    {
        public string? Email { get; set; }
        public string? Password { get; set; }
    }
}
```

Listing 25. Klasa LoginDataDto

W tym kodzie:

- Klasa `LoginDataDto` zawiera dwie właściwości: `Email` i `Password`, które mogą być używane do przechowywania danych logowania użytkownika.
- Obie właściwości są typu `string?`, co oznacza, że mogą przyjąć wartość `null`, umożliwiając ich opcjonalne wypełnianie.

4.22. Klasa LoginEndPoint

Poniższy kod przedstawia klasę `LoginEndPoint`, która implementuje logikę logowania użytkownika. Klasa ta korzysta z `PizzaUtopiaDbContext` do weryfikacji danych logowania użytkownika.

```
public class LoginEndPoint(PizzaUtopiaDbContext dbContext)
{
    public async Task<string> Login(LoginDataDto loginData)
    {
        var user = await dbContext.Users.FirstOrDefaultAsync(u => u
    .Email == loginData.Email && u.PasswordHash == loginData.
    Password);

        if (user == null)
        {
            return "Niepoprawne dane logowania";
        }
        else
        {
```

```
            return "Zalogowano";
        }
    }
}
```

Listing 26. Klasa LoginEndPoint

W tym kodzie:

- Klasa `LoginEndPoint` zawiera metodę `Login`, która przyjmuje dane logowania użytkownika (`LoginDataDto`) i sprawdza je w bazie danych przy użyciu `dbContext`.
- Metoda `Login` wykorzystuje `FirstOrDefaultAsync` do wyszukania użytkownika w bazie danych na podstawie e-maila i hasła.
- Jeśli użytkownik zostanie znaleziony, zwróci komunikat `"Zalogowano"`, w przeciwnym razie zwróci `"Niepoprawne dane logowania"`.

4.23. Migracja DodanieNCIUzytkownika

Poniższy kod przedstawia migrację `DodanieNCIUzytkownika`, która tworzy tabelę `Users` w bazie danych MySQL, zawierającą kolumny dla `Id`, `Email` oraz `PasswordHash`.

```
namespace PizzaUtopia.Backend.Migrations
{
    /// <inheritdoc />
    public partial class DodanieNCIUzytkownika : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder
    migrationBuilder)
        {
            migrationBuilder.AlterDatabase()
                .Annotation("MySql:CharSet", "utf8mb4");

            migrationBuilder.CreateTable(
                name: "Users",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable:
    false)
                        .Annotation("MySql:ValueGenerationStrategy
    ", MySqlValueGenerationStrategy.IdentityColumn),
```

```
18                      Email = table.Column<string>(type: "longtext",
    nullable: false)
19                          .Annotation("MySql:CharSet", "utf8mb4"),
20                      PasswordHash = table.Column<string>(type: "
    longtext", nullable: false)
21                          .Annotation("MySql:CharSet", "utf8mb4")
22                  },
23                  constraints: table =>
24                  {
25                      table.PrimaryKey("PK_Users", x => x.Id);
26                  })
27                  .Annotation("MySql:CharSet", "utf8mb4");
28          }
29
30          /// <inheritdoc />
31          protected override void Down(MigrationBuilder
    migrationBuilder)
32          {
33              migrationBuilder.DropTable(
34                  name: "Users");
35          }
36      }
37  }
```

Listing 27. Migracja DodanieNCIUzytkownika

W tym kodzie:

- Migracja `DodanieNCIUzytkownika` tworzy tabelę `Users`, która zawiera trzy kolumny: `Id`, `Email` oraz `PasswordHash`.
- W metodzie `Up` użyto `CreateTable`, aby stworzyć tabelę w bazie danych MySQL z odpowiednimi typami danych i ustawieniami dla kolumn.
- W metodzie `Down` tabela `Users` jest usuwana, co umożliwia cofnięcie migracji.
- Zastosowano adnotację `MySql:CharSet` dla ustawienia zestawu znaków na `utf8mb4`, co zapewnia wsparcie dla znaków Unicode.

4.24. Klasa User

Poniższy kod przedstawia klasę `User`, która reprezentuje użytkownika w aplikacji `PizzaUtopia`. Zawiera ona podstawowe informacje o użytkowniku, takie jak identyfikator, adres e-mail oraz hasło w postaci zaszyfrowanej wartości.

```
public class User
{
    public int Id { get; set; }
    public string Email { get; set; } = null!;
    public string PasswordHash { get; set; } = null!;
}
```

Listing 28. Klasa User

W tym kodzie:

- Klasa `User` zawiera trzy właściwości: `Id`, `Email` i `PasswordHash`.
- `Id` jest kluczem głównym użytkownika, a jego wartość jest automatycznie generowana przez bazę danych.
- `Email` to adres e-mail użytkownika, który jest wymagany, a `PasswordHash` zawiera zaszyfrowane hasło użytkownika.
- Właściwości `Email` i `PasswordHash` mają przypisane wartości domyślne `null!`, co oznacza, że nie mogą być `null`, ale są inicjowane w konstruktorze.

4.25. Model Snapshot dla PizzaUtopiaDbContext

Poniższy kod przedstawia klasę `PizzaUtopiaDbContextModelSnapshot`, która jest używana do generowania modelu bazy danych na podstawie migracji w Entity Framework. Zawiera definicję tabeli `Users` oraz konfigurację jej kolumn i klucza głównego.

```
namespace PizzaUtopia.Backend.Migrations
{
    [DbContext(typeof(PizzaUtopiaDbContext))]
    partial class PizzaUtopiaDbContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder
   modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "9.0.0")
                .HasAnnotation("Relational:MaxIdentifierLength",
   64);

            MySqlModelBuilderExtensions.AutoIncrementColumns(
   modelBuilder);
```

```
14
15              modelBuilder.Entity("PizzaUtopia.Backend.Models.User",
    b =>
16                  {
17                      b.Property<int>("Id")
18                          .ValueGeneratedOnAdd()
19                          .HasColumnType("int");
20
21                      MySqlPropertyBuilderExtensions.
    UseMySqlIdentityColumn(b.Property<int>("Id"));
22
23                      b.Property<string>("Email")
24                          .IsRequired()
25                          .HasColumnType("longtext");
26
27                      b.Property<string>("PasswordHash")
28                          .IsRequired()
29                          .HasColumnType("longtext");
30
31                      b.HasKey("Id");
32
33                      b.ToTable("Users");
34                  });
35 #pragma warning restore 612, 618
36          }
37      }
38 }
```

Listing 29. Model Snapshot dla PizzaUtopiaDbContext

W tym kodzie:

- Klasa `PizzaUtopiaDbContextModelSnapshot` jest odpowiedzialna za generowanie snapshotu modelu bazy danych, który jest wykorzystywany przez Entity Framework w procesie migracji.
- Metoda `BuildModel` konfiguruje tabelę `Users`, określając kolumny `Id`, `Email` oraz `PasswordHash`.
- Dla kolumny `Id` ustawiono automatyczne inkrementowanie wartości, a dla `Email` i `PasswordHash` określono typ `longtext`.
- Klucz główny tabeli `Users` jest ustawiony na kolumnę `Id`.
- Zastosowano adnotacje MySQL, takie jak `MySql:CharSet`, oraz konfigurację dla automatycznego inkrementowania kolumn.

4.26. Konfiguracja aplikacji w pliku Program.cs

Poniższy kod przedstawia konfigurację aplikacji `PizzaUtopia`, w tym ustawienia serwera Kestrel, dodanie usług do kontenera DI oraz konfigurację OpenAPI i bazy danych.

```
var builder = WebApplication.CreateBuilder(args);

// Ustawienia URL i portu
builder.WebHost.ConfigureKestrel(options =>
{
    options.ListenAnyIP(5000); // Nasłuchiwanie na porcie 5000 na wszystkich interfejsach
    options.ListenAnyIP(5001, listenOptions =>
    {
        listenOptions.UseHttps(); // Jeżli chcesz obsługiwa HTTPS na porcie 5001
    });
});

// Dodaj usługi do kontenera
builder.Services.AddOpenApi();
builder.Services.AddDbContext<PizzaUtopiaDbContext>();

var app = builder.Build();

// Konfiguracja potoku da HTTP
if (app.Environment.IsDevelopment())
{
    app.MapOpenApi();
}
```

Listing 30. Konfiguracja aplikacji w Program.cs

W tym kodzie:

- Tworzony jest obiekt `builder`, który umożliwia konfigurację aplikacji.
- Ustawienia serwera Kestrel pozwalają na nasłuchiwanie na porcie `5000` na wszystkich interfejsach sieciowych oraz na porcie `5001` z obsługą HTTPS.
- Do kontenera DI dodawane są usługi: `AddOpenApi`, która umożliwia generowanie dokumentacji API, oraz `AddDbContext`, która konfiguruje dostęp do bazy danych `PizzaUtopiaDbContext`.

- Jeśli aplikacja jest uruchomiona w środowisku deweloperskim, mapa OpenAPI jest dodawana do potoku żądań.

4.27. Konfiguracja endpointów w aplikacji

Poniższy kod przedstawia konfigurację endpointów w aplikacji `PizzaUtopia` przy użyciu minimalnej konfiguracji API w .NET. Zawiera definicje dla endpointów do logowania, rejestracji oraz domyślnego endpointu.

```
app.MapGet("/", () => "Backend serwisu PizzaUtopia");

app.MapPost("/auth/login", async (PizzaUtopiaDbContext dbContext,
    LoginDataDto loginData) =>
{
    var loginEndPoint = new LoginEndPoint(dbContext);

    var result = await loginEndPoint.Login(loginData);

    if (result == "Niepoprawne dane logowania")
    {
        return Results.BadRequest(result);
    }
    else
    {
        return Results.Ok(result);
    }
});

app.MapPost("/register", (PizzaUtopiaDbContext dbContext, User user
    ) =>
{
    dbContext.Users.Add(user);
    dbContext.SaveChanges();
    return Results.Created($"/user/{user.Id}", user);
});

app.Run();
```

Listing 31. Konfiguracja endpointów w aplikacji

W tym kodzie:

- Endpoint `GET /` zwraca prostą wiadomość tekstową `"Backend serwisu PizzaUtopia"`.
- Endpoint `POST /auth/login` obsługuje proces logowania. Sprawdza dane logowania użytkownika, a jeśli są poprawne, zwraca wynik `Ok`, w przeciwnym

przypadku `BadRequest` z komunikatem błędu.

- Endpoint `POST /register` umożliwia rejestrację użytkownika, dodając dane użytkownika do bazy i zapisując je w bazie danych, a następnie zwraca odpowiedź `Created` z identyfikatorem nowego użytkownika.
- `app.Run()` uruchamia aplikację.

5. Testowanie

W tej sekcji szczegółowo opisano proces testowania aplikacji, obejmujący testy funkcjonalne, integracyjne oraz akceptacyjne. Celem testowania było upewnienie się, że aplikacja działa zgodnie z założeniami projektowymi, nie generuje błędów, a jej funkcjonalności spełniają wymagania użytkowników. Proces testowania obejmował również identyfikację ewentualnych problemów oraz ich rozwiązanie przed wydaniem wersji produkcyjnej.

5.1. Testy funkcjonalne

Testy funkcjonalne miały na celu sprawdzenie, czy wszystkie kluczowe funkcje aplikacji działają poprawnie. W trakcie testów funkcjonalnych przeanalizowane zostały następujące przypadki testowe:

- **Rejestracja nowego użytkownika:** Testy obejmowały proces tworzenia konta użytkownika. Weryfikowane były sytuacje, w których użytkownik wprowadza dane poprawne, niepełne, lub błędne (np. niepoprawny format adresu e-mail). Sprawdzono także odpowiednie komunikaty o błędach w przypadku niewłaściwych danych.
- **Logowanie:** Testy obejmowały zarówno udane logowanie, jak i scenariusze błędnego logowania (np. nieprawidłowe hasło lub adres e-mail). Sprawdzono również integrację aplikacji z systemem biometrycznym (odcisk palca lub rozpoznawanie twarzy).
- **Składanie zamówienia:** Testowano proces dodawania produktów do koszyka, edycji zamówienia oraz finalizowania zakupu. Sprawdzono poprawność danych zamówienia, w tym cen i produktów, a także integrację z systemem płatności.
- **Edycja profilu:** Przetestowano możliwość zmiany danych osobowych, takich jak numer telefonu, adres e-mail oraz zdjęcie profilowe. Sprawdzono również prawidłowe zapisanie zmian w bazie danych.

5.2. Testy integracyjne

Testy integracyjne miały na celu sprawdzenie współdziałania różnych modułów aplikacji, aby upewnić się, że całość systemu działa poprawnie. Kluczowe scenariusze obejmowały:

- **Integracja systemu logowania z MySQL:** Sprawdzono, czy dane użytkowników są poprawnie przesyłane do bazy danych MySQL oraz czy aplikacja odpowiednio reaguje na nieudane próby logowania.

- **Przesyłanie danych zamówienia do bazy danych:** Przetestowano proces przesyłania danych zamówienia z aplikacji do bazy danych MySQL. Testowano również poprawność zapisanych danych, takich jak produkty, liczba sztuk, adres dostawy, itp.

- **Synchronizacja ustawień biometrycznych z urządzeniem użytkownika:** Przetestowano proces synchronizacji danych biometrycznych użytkownika z urządzeniem (np. z czytnikiem linii papilarnych), aby zapewnić prawidłowe logowanie i autoryzację.

5.3. Testy akceptacyjne

Testy akceptacyjne zostały przeprowadzone z udziałem grupy użytkowników testowych, którzy oceniali aplikację z perspektywy doświadczeń użytkownika. Testy akceptacyjne obejmowały następujące aspekty:

- **Łatwość obsługi i intuicyjność interfejsu:** Testowano interfejs użytkownika pod kątem jego intuicyjności, łatwości nawigacji oraz dostępności wszystkich funkcji. Zbierano opinie testerów na temat ewentualnych trudności w obsłudze aplikacji.

- **Zgłaszanie błędów:** Testerzy zgłaszali wszelkie napotkane błędy, takie jak problemy z ładowaniem stron, błędne komunikaty o błędach czy brak reakcji aplikacji na działania użytkownika.

- **Ocena szybkości działania:** Sprawdzono czas reakcji aplikacji na różne akcje użytkownika, takie jak przejście między ekranami, dodawanie produktów do koszyka oraz finalizacja zamówienia.

5.4. Podsumowanie wyników testów

Testy wykazały, że aplikacja działa zgodnie z założeniami projektowymi, a wszystkie zgłoszone błędy zostały naprawione przed wydaniem wersji produkcyjnej. Poniżej przedstawiono podsumowanie wyników testów funkcjonalnych, integracyjnych i akceptacyjnych w formie tabel:

Test	Opis	Wynik
Rejestracja użytkownika	Tworzenie konta, testy błędów	Pomyślnie, komunikaty działają
Logowanie	Testowanie poprawnych i błędnych danych	Pomyślnie, błąd przy błędnych danych
Składanie zamówienia	Dodawanie produktów, finalizacja	Pomyślnie, dane przesyłane
Edycja profilu	Zmiana danych użytkownika	Pomyślnie, dane zapisane
Integracja MySQL	Logowanie z MySQL	Pomyślnie, synchronizacja działa
Przesyłanie danych zamówienia	Przesyłanie danych do bazy	Pomyślnie, zapisano dane
Synchronizacja biometrii	Synchronizacja biometrii	Pomyślnie, dane synchronizowane

Tab. 5.1. Wyniki testów funkcjonalnych i integracyjnych

Tabela **??** pokazuje, że wszystkie kluczowe funkcjonalności aplikacji zostały pomyślnie przetestowane, a wyniki testów są zgodne z założeniami. Wszystkie błędy zgłoszone podczas testów zostały naprawione przed wydaniem aplikacji.

Aspekt	**Opis**	**Ocena**
Łatwość obsługi	Intuicyjność interfejsu	4.8/5
Zgłaszanie błędów	Liczba zgłoszonych błędów	2 błędy
Szybkość działania	Czas reakcji aplikacji na różne akcje	4.5/5

Tab. 5.2. Podsumowanie wyników testów akceptacyjnych

Tabela **??** przedstawia ocenę aplikacji przez testerów. Zgłoszone błędy były nieznaczne i zostały szybko naprawione. Testerzy ocenili aplikację jako intuicyjną i szybką w działaniu.

Proces testowania potwierdził wysoką jakość aplikacji i jej gotowość do wdrożenia. Wszystkie zgłoszone problemy zostały rozwiązane, a aplikacja spełnia oczekiwania użytkowników pod względem funkcjonalności, wydajności i łatwości obsługi.

6. Podręcznik użytkownika

W tej sekcji przedstawimy szczegółowe instrukcje dotyczące korzystania z aplikacji do zamawiania jedzenia. Każdy krok będzie ilustrowany zrzutami ekranu.

6.1. Ekran powitalny

Po uruchomieniu aplikacji użytkownik zobaczy ekran powitalny (Rysunek 6.1). W górnej części znajduje się logo „PizzaUtopia", a poniżej dwie opcje: - **Zaloguj się**: kliknij ten przycisk, jeśli jesteś już zarejestrowanym użytkownikiem. - **Zarejestruj się**: wybierz tę opcję, jeśli chcesz utworzyć nowe konto.

Użytkownik jest zachęcany do wyboru jednej z opcji, co pozwala na dalsze etapy korzystania z aplikacji.

6.2. Rejestracja

Jeśli wybierzesz „Zarejestruj się", przejdziesz do ekranu rejestracji (Rysunek: 6.2). Wprowadź wymagane dane: - **Hasło**: Stwórz silne hasło, które będzie zabezpieczało Twoje konto. - **Powtórz hasło**: Upewnij się, że hasło jest identyczne w obu polach, aby uniknąć błędów.

Po wpisaniu danych, należy kliknąć przycisk **Zarejestruj się**, aby utworzyć nowe konto. W przypadku błędów, aplikacja wskaże nieprawidłowe informacje.

Zachowaj swoje hasło w bezpiecznym miejscu, aby móc się później zalogować.

6.3. Logowanie

Dla użytkowników, którzy już mają konto, ekran logowania pozwala na szybki dostęp (Rysunek:6.3). Należy wprowadzić: - **Email**: Wprowadź adres e-mail, którym się zarejestrowałeś. - **Hasło**: Wprowadź hasło używane podczas rejestracji.

Klikając przycisk **Zaloguj się**, użytkownik uzyskuje dostęp do swojego konta. W przypadku zapomnienia hasła, należy skorzystać z opcji „Zapomniałeś hasła?", aby otrzymać instrukcje resetowania.

6.4. Moje zamówienie

Po zalogowaniu, użytkownik przechodzi do ekranu zamówienia (Rysunek:6.4). Wyświetlane są wszystkie zamówione pozycje, na przykład: - 2 x Pizza Pepperoni - 1 x

Rys. 6.1. Ekran powitalny aplikacji

Rys. 6.2. Ekran rejestracji w aplikacji

Rys. 6.3. Ekran logowania w aplikacji

Pizza Margarita

Na dole ekranu znajdują się przyciski nawigacyjne: - **Wróć do koszyka**: pozwala powrócić do sekcji, gdzie można zmodyfikować zamówienie. - **Dokonaj zamówienia**: przekształca zamówienie w transakcję.

Ważne jest, aby dokładnie sprawdzić zamówienie przed dokonaniem zakupu.

6.5. Edycja profilu

Aby edytować profil, użytkownik musi przejść do sekcji edycji (Rysunek:6.5). Można tam wprowadzić: - **Imię i nazwisko**: Upewnij się, że dane są aktualne. - **Numer telefonu**: Niezbędne do potwierdzenia zamówień i kontaktu.

Przygotuj się do wprowadzenia zmian i użyj ikony aparatu, aby zmienić zdjęcie profilowe. Możesz również dodać dodatkowe dane, jeśli aplikacja to umożliwia.

Po dokonaniu wszelkich zmian, kliknij przycisk "Zapisz zmiany", aby zapisać swoje ustawienia.

6.6. Ustawienia

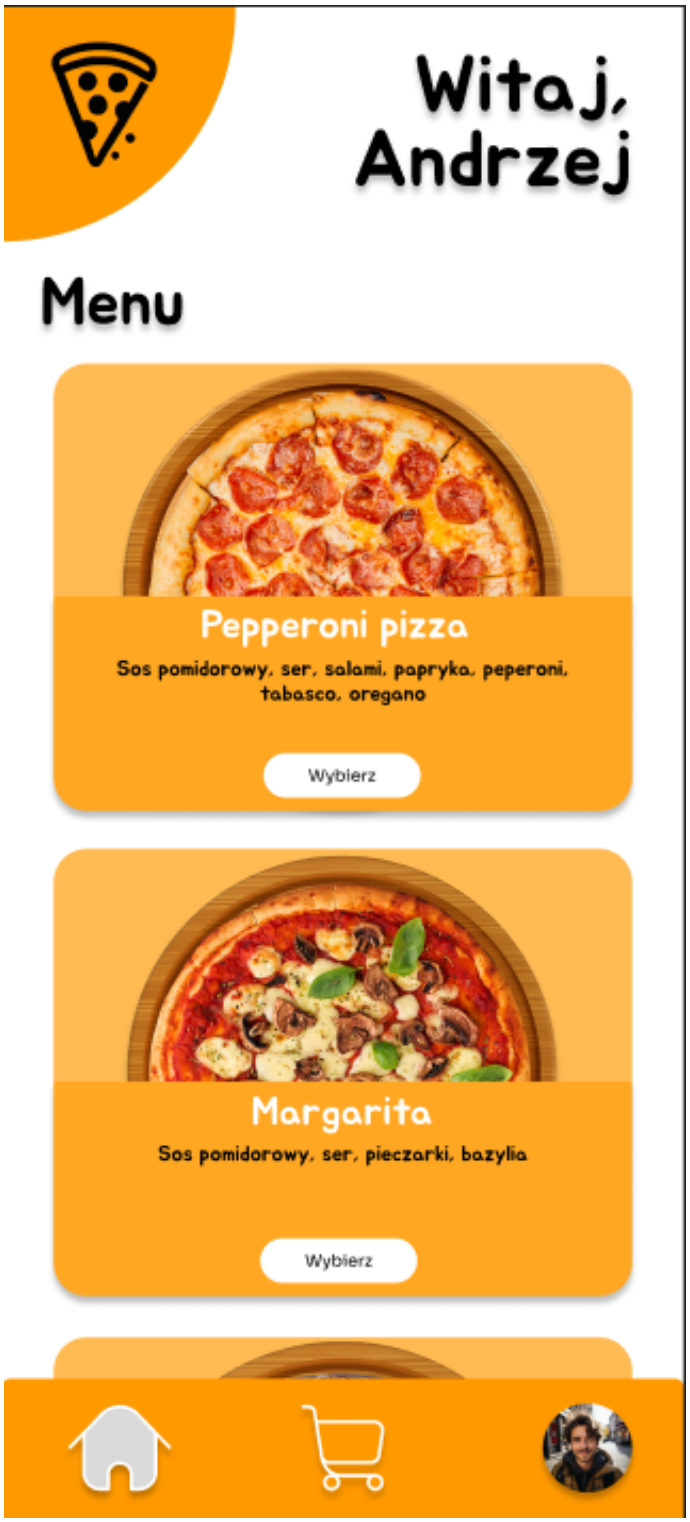
Ekran ustawień zawiera opcje do (Rysunek:6.6): - **Zmiany hasła**: Jeśli czujesz, że Twoje hasło mogło zostać ujawnione, natychmiast je zmień. - **Konfiguracji biometrii**: Opcjonalne ustawienia logowania za pomocą odcisku palca lub rozpoznawania twarzy. - **Ustawienia wyglądu**: Możesz dostosować interfejs aplikacji do swoich preferencji.

Każda sekcja ustawień pozwala na indywidualne dostosowanie aplikacji.

6.7. Zmiana hasła

Aby zmienić hasło, postępuj zgodnie z poniższymi krokami (Rysunek:6.7): 1. **Wprowadź obecne hasło**: System weryfikuje Twoją tożsamość. 2. **Wprowadź nowe hasło**: Upewnij się, że jest wystarczająco silne. 3. **Potwierdź nowe hasło**: Dla bezpieczeństwa wpisz je ponownie.

Po dokonaniu wszelkich zmian, kliknij przycisk "Zapisz zmiany", który finalizuje proces zmiany hasła.



Rys. 6.4. Ekran z podsumowaniem mojego zamówienia

Rys. 6.5. Ekran edycji profilu

Rys. 6.6. Ekran ustawień konta

Rys. 6.7. Ekran zmiany hasła

6.8. Biometria

Sekcja biometrii pozwala użytkownikom włączyć lub edytować ustawienia biometryczne (Rysunek:6.8): - **Przełącznik**: Pozwala włączyć lub wyłączyć funkcję biometrii. - **Dodaj kolejny odcisk palca**: Umożliwia dodanie więcej niż jednego odcisku dla większej wygody.

Upewnij się, że każdy odcisk jest dobrze zarejestrowany.

6.9. Wygląd

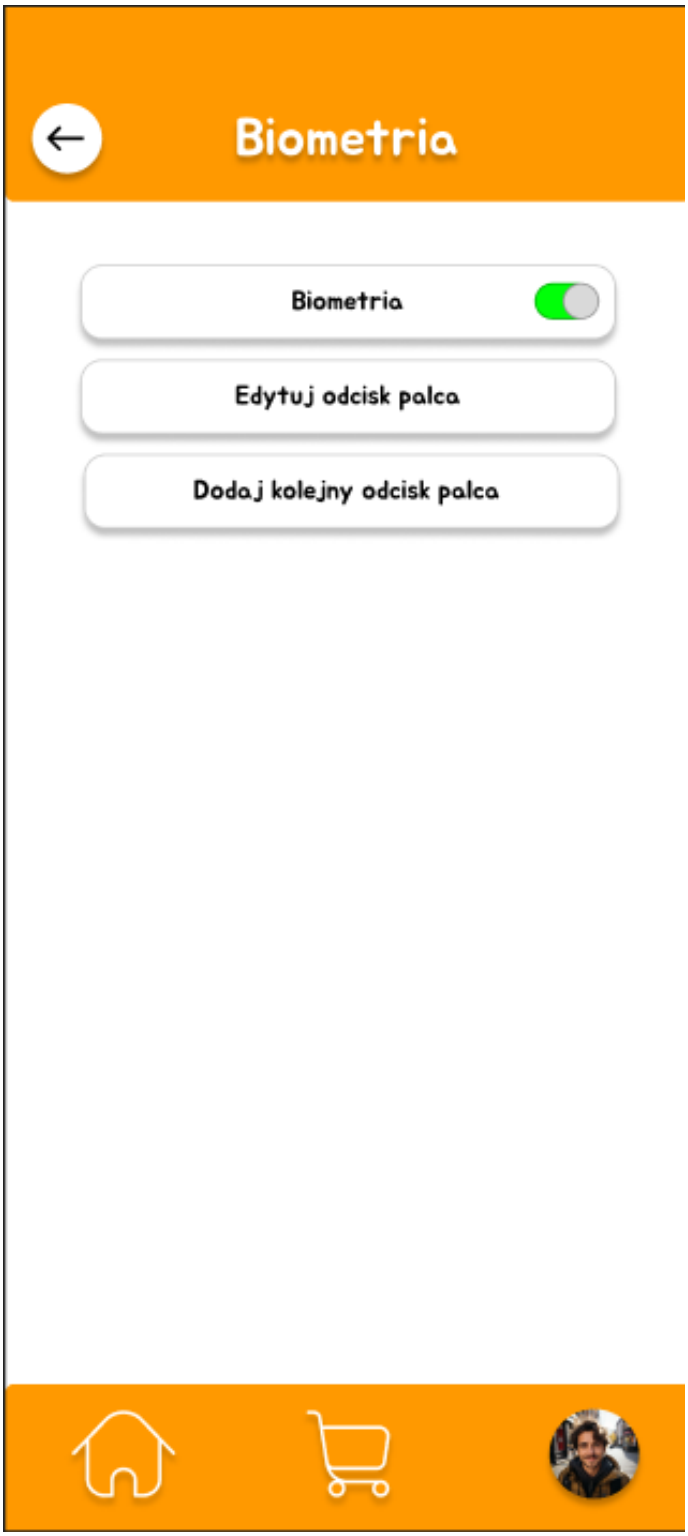
W sekcji wyglądu możesz dostosować interfejs aplikacji (Rysunek:6.9): - **Automatyczny motyw**: Aplikacja dostosowuje się do ustawień urządzenia. - **Ciemny motyw**: Idealny do użycia w warunkach słabego oświetlenia, aby zminimalizować zmęczenie oczu.

Dokonaj wyboru, który najbardziej Ci odpowiada.

6.10. Podsumowanie

Przestrzegając powyższych kroków, możesz łatwo zarządzać swoim kontem, zmieniać ustawienia i składać zamówienia w aplikacji. W razie dodatkowych pytań, skorzystaj z sekcji FAQ lub skontaktuj się z obsługą klienta.

Zastosowanie tych informacji pozwoli Ci na pełne wykorzystanie możliwości aplikacji, co ułatwi codzienne zamawianie jedzenia.



Rys. 6.8. Ekran ustawień biometrii

Rys. 6.9. Ekran ustawień wyglądu

Bibliografia

[1] *Światło*. URL: https://pl.wikipedia.org/wiki/Czujnik_fotoelektryczny.

[2] *Aparat*. URL: https://pl.wikipedia.org/wiki/Aparat_fotograficzny.

[3] *C*. URL: https://pl.wikipedia.org/wiki/C_Sharp.

[4] *.NET MAUI*. URL: https://learn.microsoft.com/pl-pl/dotnet/maui/what-is-maui?view=net-maui-9.0.

[5] *Biometryka*. URL: https://pl.wikipedia.org/wiki/Zabezpieczenie_biometryczne.

[6] *MySQL*. URL: https://pl.wikipedia.org/wiki/MySQL.

[7] *Microsoft Visual Studio*. URL: https://pl.wikipedia.org/wiki/Microsoft_Visual_Studio.

[8] *.NET MAUI*. URL: https://learn.microsoft.com/pl-pl/dotnet/maui/what-is-maui?view=net-maui-9.0 (term. wiz. 08. 10. 2024).

[9] *Github*. URL: https://pl.wikipedia.org/wiki/GitHub (term. wiz. 08. 10. 2024).

[10] *Figma*. URL: https://en.wikipedia.org/wiki/Figma (term. wiz. 10. 10. 2024).

Spis rysunków

Spis tabel

Spis listingów