Dokumentacja systemu

FastFood

~ IO 2025

Aleksander Kwaśny
Szymon Łabędziewski
Karol Starzyk
Kacper Plutowski
Wojciech Marchewka
Paweł Dyjak

Spis treści

1. Wprowadzenie do systemu – cel, kontekst użycia 2
2. Główne funkcjonalności z podziałem na role użytkowników 2
2.1 Klient 2
2.2 Pracownik kuchni 2
2.3 Administrator 3
3. Opis architektury aplikacji 3
3.1 Frontend 3
3.2 Backend (Supabase) 3
3.3 Komunikacja i organizacja plików 3
4. Diagramy UML: przypadki użycia, diagram klas, diagram aktywności 3
4.1 Diagram przypadków użycia (Use Case) 3
4.2 Diagram klas 4
4.3 Ogólny diagram aktywności (Activity) 4
5. Struktura bazy danych 4
5.1 Tabele i relacje 4
5.2 Triggery 5
5.3 Typy ENUM 5
6. Opis interfejsów użytkownika 5
6.1 Panel klienta (*client.html*) 5
6.2 Panel kuchni (*kitchen.html*) 5

6.3 Panel administratora (*admin.html*) 5
6.4 Strona główna (*index.html*) 5
7. Kryteria akceptacji i scenariusze testowe 6
7.1 Kryteria akceptacji 6
7.2 Scenariusze testowe 6
8. Testy automatyczne – opis testów Selenium 6

1. Wprowadzenie do systemu – cel, kontekst użycia

System *FastFood* to internetowa aplikacja do składania zamówień z restauracji typu fast-food. Umożliwia klientom przeglądanie menu, dodawanie produktów do koszyka i składanie zamówień przez przeglądarkę WWW. Złożone zamówienie jest następnie automatycznie przekazywane do systemu restauracji (panelu kuchni), co usprawnia obsługę i skraca czas realizacji. System posiada trzy główne role użytkowników – Klienta, Pracownika kuchni i Administratora – dzięki czemu każdy może wykonywać właściwe dla siebie zadania. Aplikacja może być wykorzystywana w restauracjach fast-food do zdalnego przyjmowania i realizacji zamówień online.

2. Główne funkcjonalności z podziałem na role użytkowników

2.1 Klient

- **Przeglądanie menu:** Klient widzi listę dostępnych produktów podzielonych na kategorie (np. przekąski, dania główne, napoje). Przy każdym produkcie wyświetlana jest nazwa, opis i cena (zazwyczaj w walucie lokalnej).
- **Dodawanie do koszyka:** Klient wybiera produkty i podaje ilość zamawianych sztuk. Interfejs umożliwia łatwe dodanie produktów do wirtualnego koszyka, przy czym ceny i kategorie są dobrze widoczne.
- **Finalizacja zamówienia:** Po wybraniu produktów klient zatwierdza zamówienie. Po zatwierdzeniu następuje potwierdzenie złożenia zamówienia. Cały proces jest analogiczny do standardowego zamawiania online, gdzie zamówienie „trafia" natychmiast do restauracji.

2.2 Pracownik kuchni

- **Odbiór zamówień:** Panel kuchni (kitchen.html) wyświetla w czasie rzeczywistym listę nowych zamówień złożonych przez klientów. System prezentuje je jako kolejkę z informacją o produktach. Gdy zamówienie jest składane w punkcie sprzedaży, natychmiast pojawia się ono na ekranie kuchni, co pozwala personelowi natychmiast je podjąć.
- **Zarządzanie statusem zamówienia:** Pracownik kuchni może aktualizować status każdego zamówienia – np. oznaczać je jako „w przygotowaniu" lub „gotowe do odbioru". Dzięki temu system wspiera płynne zarządzanie realizacją. Schematy typu *Kitchen Display System (KDS)*

ułatwiają organizację pracy – zamówienia są przejrzyste i zorganizowane, co poprawia wydajność i minimalizuje pomyłki.

2.3 Administrator

- **Zarządzanie asortymentem:** Administrator ma pełny dostęp do edycji treści menu. Poprzez panel administracyjny (admin.html) może dodawać nowe produkty, zmieniać ich opis, cenę lub usuwać pozycje z menu. Pozwala to na bieżąco aktualizować ofertę restauracji. Administrator może również zarządzać kategoriami produktów, co ułatwia utrzymanie porządku w menu.

3. Opis architektury aplikacji

3.1 Frontend

Interfejs użytkownika zbudowano jako stronę statyczną HTML/JavaScript. Główna strona (**index.html**) pełni rolę kontenera lub punktu wyjścia, z którego linkuje się do trzech paneli funkcyjnych: klienta, kuchni i administracji. Każdy panel (client.html, kitchen.html, admin.html) jest osobnym dokumentem HTML zawierającym formy i skrypty potrzebne do realizacji funkcji. Frontend komunikuje się bezpośrednio z backendem (usługą Supabase) za pomocą biblioteki **supabase-js** lub wywołań REST do API. W efekcie nie ma konieczności pisania własnego serwera – przeglądarka korzysta z bezserwerowej bazy danych.

3.2 Backend (Supabase)

Backendem systemu jest platforma Supabase, która opiera się na relacyjnej bazie danych PostgreSQL jako *rdzeniu systemu*. Supabase automatycznie generuje nad bazą danych pełne RESTful API (dzięki technologii **PostgREST**). Oznacza to, że każda tabela w bazie jest od razu dostępna do operacji CRUD poprzez standardowe końcówki https://<project>.supabase.co/rest/v1/. Dzięki temu frontend może m.in. pobierać listę produktów, zapisywać nowe zamówienia czy aktualizować statusy bez pisania dodatkowego kodu serwera. Backend zapewnia także autoryzację użytkowników (GoTrue) oraz mechanizmy czasu rzeczywistego (Realtime), co umożliwia np. automatyczne odświeżanie widoków.

3.3 Komunikacja i organizacja plików

Komunikacja między frontem a bazą danych odbywa się za pośrednictwem API Supabase – aplikacja wysyła zapytania HTTP/REST lub używa SDK supabase-js, a otrzymane odpowiedzi przetwarza na stronie klienta. Struktura katalogów projektu jest prosta: pliki HTML reprezentują poszczególne panele, a dodatkowe pliki JavaScript/CSS są dołączane w razie potrzeby. Konfiguracja środowiska Supabase (łączenie do konkretnej bazy, klucze API) znajduje się w plikach JS frontendu. Całość działa w architekturze bezserwerowej (*serverless*) – nie ma tu tradycyjnego backendu w Node.js czy PHP, co eliminuje konieczność obsługi serwera aplikacji.

4. Diagramy UML: przypadki użycia, diagram klas, diagram aktywności

4.1 Diagram przypadków użycia (Use Case)

Diagram *przypadków użycia* ilustruje interakcje użytkowników z systemem. Pokazuje aktorów (np. Klient, Pracownik kuchni, Administrator) oraz ich główne czynności. Na przykład Klient może wybrać produkty i złożyć zamówienie, Pracownik kuchni – obsłużyć zamówienie i zmienić jego status, a

Administrator – zarządzać produktami. Diagram pozwala zrozumieć „kto co robi” w systemie i pokazuje zakres funkcji oraz relacje między nimi.

4.2 Diagram klas

Diagram klas przedstawia strukturę obiektową systemu. Pokazuje klasy (np. *Produkt*, *Zamówienie*, *PozycjaZamówienia*, *Użytkownik*) wraz z atrybutami (pola danych) i metodami. Rysunek odzwierciedla zależności między klasami, takie jak dziedziczenie czy powiązania jeden-do-wielu. Na przykład klasa *Zamówienie* może mieć listę obiektów *PozycjaZamówienia*, z każdym *PozycjaZamówienia* powiązanym z jednym *Produktem*. Diagram klas „dostarcza jasnego widoku architektury systemu”, co pomaga programistom zrozumieć, jak zorganizowane są dane i jakie są między nimi zależności.

4.3 Ogólny diagram aktywności (Activity)

Diagram aktywności to rodzaj schematu blokowego przedstawiający przebieg procesu w systemie. W kontekście systemu FastFood może on pokazywać np. kolejność kroków przy realizacji zamówienia: od zalogowania się/przejścia do menu, poprzez wybór produktów, przez składanie zamówienia, aż po przygotowanie i odbiór posiłku. Diagram ten modeluje “flow” działań – pokazuje, jakie czynności są wykonywane i w jakim porządku. Tego typu diagram pomaga przedstawić logikę działania systemu w sposób czytelny dla zespołu i interesariuszy, ułatwiając zrozumienie procesów biznesowych.

5. Struktura bazy danych

5.1 Tabele i relacje

W bazie danych projektu FastFood można wyróżnić kilka kluczowych tabel. Przykładowo:

- **users** (użytkownicy) – przechowuje dane użytkowników systemu (np. klienci lub pracownicy). Kolumny mogą obejmować: UserID (klucz główny), Username, Password, Email, PhoneNumber, DeliveryAddress itp. (wg koncepcji standardowej tabeli użytkowników).
- **products** (produkty) – zawiera informacje o daniach i artykułach dostępnych w menu. Typowe kolumny to ProductID, Name, Description, Price, CategoryID (obcy klucz do tabeli kategorii).
- **orders** (zamówienia) – rejestruje złożone zamówienia. Może mieć kolumny: OrderID (PK), UserID (FK – który klient złożył), OrderDate, DeliveryAddress, TotalAmount, OrderStatus (status, np. *nowe*, *w przygotowaniu*, *gotowe*).
- **order_items** lub **order_details** (pozycje zamówienia) – przechowuje szczegóły każdej pozycji zamówienia. Kolumny: OrderDetailID (PK), OrderID (FK), ProductID (FK), Quantity, Price. Każde zamówienie może mieć wiele pozycji – odwzorowane relacją jeden-do-wielu z tabelą *orders*.
- (Opcjonalnie) **categories** (kategorie) – lista kategorii produktów (np. napoje, przekąski). Umożliwia grupowanie produktów.
- (Opcjonalnie) **payments** – informacje o płatnościach za zamówienia (metoda, data płatności).

Relacje między tabelami są typowe dla e-commerce: jeden użytkownik może mieć wiele zamówień, a jedno zamówienie może zawierać wiele pozycji. Tabela *orders* powiązana jest z *order_items*, a *order_items* z *products*. Tabela *users* powiązana jest z zamówieniami i ewentualnymi innymi tabelami (np. *payments*, *reviews*). Struktura oparta jest na kluczach obcych, zapewniając spójność danych.

5.2 Triggery

W bazie danych można definiować triggery – procedury wywoływane automatycznie w reakcji na zmiany w tabeli. Na przykład można utworzyć *trigger* typu *BEFORE INSERT* lub *AFTER UPDATE*, który wykona dodatkowe operacje (np. aktualizację sumy zamówienia) za każdym razem, gdy wstawiane lub modyfikowane są dane. Polecenie SQL CREATE TRIGGER pozwala zdefiniować taki trigger. Treść triggery wykonuje określoną funkcję przy wskazanych operacjach na tabeli (INSERT, UPDATE, DELETE). W praktyce w projekcie FastFood może to służyć np. do automatycznej aktualizacji daty modyfikacji rekordu czy sprawdzania ograniczeń biznesowych.

5.3 Typy ENUM

W projekcie można wykorzystać typy wyliczeniowe (enum) dla pól o skończonym zbiorze wartości. Przykładowo kolumna OrderStatus w tabeli *orders* może mieć typ enum z wartościami takimi jak NOWE, W_PRZYGOTOWANIU, GOTOWE. Typ enum w PostgreSQL to statyczny, uporządkowany zestaw zdefiniowanych wartości, podobny do enum w językach programowania. Dzięki temu statusy są łatwe do walidacji (brak możliwości wpisania nieistniejącej wartości) i uporządkowane. Tworzy się je poleceniem CREATE TYPE nazwa AS ENUM ('wartosc1', 'wartosc2', ...), a następnie używa jako typu kolumny.

6. Opis interfejsów użytkownika

6.1 Panel klienta (*client.html*)

Panel klienta to strona, na której użytkownik końcowy przegląda menu i składa zamówienie. Zazwyczaj wyświetla listę produktów (dania fast-food) z podziałem na kategorie. Każda pozycja zawiera nazwę, zdjęcie (opcjonalnie) i cenę – tak aby klient mógł łatwo ocenić ofertę. Przy każdym produkcie dostępne są pola wyboru ilości i przycisk dodania do koszyka. Klient widzi aktualną zawartość koszyka i może przejść do formularza finalizacji zamówienia. Interfejs klienta jest czytelny i umożliwia łatwe wyszukiwanie i filtrowanie produktów, dzięki czemu szybkie złożenie zamówienia jest intuicyjne.

6.2 Panel kuchni (*kitchen.html*)

Panel kuchni przedstawia bieżące zamówienia do realizacji – pokazuje listę zamówień w kolejności wpływu. Interfejs ten składa się z kafelków zawierających numer zamówienia, zamówione pozycje i statusu. Pracownik kuchni korzysta z tego panelu, aby widzieć, co ma przygotować. Po zmianie statusu (np. „w przygotowaniu", „gotowe") dane te zostają zapisane w bazie. Panel kuchni automatycznie odświeża listę, sortuje zamówienia najpierw po statusie, a następnie po czasie złożenia, oraz dla każdego zamówienia wyświetla „Szacowany czas" realizacji wyliczany jedynie na podstawie bieżącego statusu.

6.3 Panel administratora (*admin.html*)

Panel administracyjny służy do zarządzania zawartością menu i innymi ustawieniami systemu. Zawiera formularz dodawania nowego produktu – pola takie jak nazwa, opis, cena, kategoria, a także przyciski do zapisu. Administrator widzi też listę już istniejących produktów z opcjami edycji lub usunięcia. Po dodaniu produktu następuje natychmiastowa aktualizacja bazy i pojawienie się nowego produktu w panelu klienta. Interfejs ten jest prosty: formularze i tabele pozwalające wprowadzać oraz modyfikować dane systemu (np. zarządzanie kategoriami czy ewentualnie kontami).

6.4 Strona główna (*index.html*)

Strona główna pełni rolę punktu startowego – może zawierać wybór roli lub linki do paneli. W niektórych realizacjach może być to zwykła strona informacyjna z przyciskami „Panel klienta", „Panel

kuchni", „Panel admina", które przekierowują do odpowiednich stron HTML. Może także zawierać wspólne elementy nawigacyjne (np. menu główne aplikacji). W każdym razie index.html integruje pozostałe panele w jedną aplikację i służy jako kontener dla całości.

7. Kryteria akceptacji i scenariusze testowe

7.1 Kryteria akceptacji

Kryteria akceptacji to specyficzne warunki, które muszą zostać spełnione, aby dana funkcjonalność uznana została za poprawnie zaimplementowaną. Innymi słowy, określają one, jakie wymagania musi realizować system, by zostać zaakceptowany przez użytkownika. Przykładowe kryteria dla projektu FastFood mogą obejmować: *klient może dodać produkt do koszyka i złożyć zamówienie*, *zamówienie klienta pojawia się w panelu kuchni*, *administrator może dodać nowy produkt, który natychmiast staje się widoczny w menu*. Każde kryterium jest zwięzłe, mierzalne i związane z oczekiwanym zachowaniem systemu.

7.2 Scenariusze testowe

Scenariusze testowe to opisy przebiegu testów weryfikujących spełnienie wymagań (kryteriów akceptacji). Każdy scenariusz definiuje kroki do wykonania oraz oczekiwane rezultaty. Przykładowo: *Scenariusz „Złożenie zamówienia przez klienta"*: klient wybiera pozycje, składa zamówienie, a następnie tester sprawdza, czy na liście zamówień w panelu kuchni pojawiło się nowe zamówienie z poprawnymi pozycjami i danymi klienta*. Inny scenariusz: *„Dodanie produktu przez administratora"* – test automatycznie loguje się jako admin, dodaje produkt, po czym sprawdza, że produkt jest widoczny w menu klienta oraz w bazie danych. Scenariusze testowe są zazwyczaj zdefiniowane w formie tabeli lub w dokumencie tekstowym (jak plik Word), a następnie realizowane ręcznie lub automatycznie.

8. Testy automatyczne – opis testów Selenium

W projekcie FastFood zaimplementowano testy automatyczne z użyciem **Selenium WebDriver** (biblioteki do automatyzacji przeglądarki). Selenium „automatyzuje przeglądarki" w celu testowania aplikacji webowych. Przykładowe testy to:

- **AdminAddProductTest:** Test ten otwiera w przeglądarce panel admina, wypełnia formularz dodawania produktu (nazwa, cena, kategoria) i zatwierdza dodanie. Następnie sprawdza, czy produkt pojawia się na liście produktów (np. przez wyszukaniem jego nazwy). Weryfikuje także, czy dane nowego produktu zostały zapisane w bazie. Dzięki temu test sprawdza poprawność działania funkcji dodawania produktów oraz integracji frontendu z backendem.
- **KitchenOrderTest:** Ten test symuluje pełny przebieg zamówienia od strony klienta: uruchamia panel klienta, wybiera kilka produktów i składa zamówienie. Następnie przechodzi do panelu kuchni i sprawdza, czy pojawiło się nowe zamówienie o oczekiwanych parametrach (np. właściwe ID, lista pozycji, status „nowe"). Test ten weryfikuje, czy przepływ danych działa prawidłowo – tzn. czy zamówienie przesyłane jest z panelu klienta do kuchni oraz czy interfejsy reagują poprawnie.
- Testy tworzone są w Javie (np. przy użyciu JUnit i Selenium) i mają charakter end-to-end. Ich celem jest automatyczne wykonywanie najważniejszych scenariuszy użytkownika, co przyspiesza wykrywanie regresji i błędów. Dzięki Selenium możliwe jest testowanie aplikacji dokładnie tak, jak użytkownik – sterując przeglądarką, klikając przyciski i wypełniając formularze.