

**Instytut Informatyki  
Kolegium Nauk Przyrodniczych  
Uniwersytet Rzeszowski**

**Dokumentacja techniczna projektu:**

***Interactive Restaurant***

**Wykonał:**

**Maksymilian Ziembicki**

**Rzeszów 2023**

1. **Temat projektu**

Tematem projektu jest aplikacja służąca do zarządzania zasobami restauracji oraz udostepniania szczegółowych informacji na temat restauracji.

1. **Cel projektu**

Celem projektu jest usprawnienie działania restauracji, sprawniejsze zarządzanie oraz dostarczenie wielu szczegółowych danych na temat restauracji a w szczególności aspektu finansowego

1. **Aktualne funkcjonalności aplikacji**

* Navbar, który sprawnie nawiguje nas po strukturze strony internetowej
* Panel magazynu, który przechowuje informacje o aktualnych produktach spożywczych umieszczonych w magazynie restauracji
* Panel składników, który służy nam do tworzenia nowych dań
* Filtracja składników po ich kategorii
* Panel zamówień, w którym możemy tworzyć zamówienia
* Obliczanie wartości zamówienia na podstawie dań w „koszyku”
* Możliwość zarządzania „koszykiem”
* Panel zamówień, który wyświetla szczegółowe dane na temat zamówień

1. **Technologie**

JavaScript, Node.JS, HTML, CSS, Express.JS, MongoDB, Mongoose, EJS, Nodemon

**5. Interesariusze aplikacji**

Personel restauracji.

1. **Harmonogram realizacji projektu**

**Faza 1: planowanie**

1. **Rozmowa z opiekunem proseminarium**

* Ustalenie tematu oraz funkcjonalności projektu

1. **Analiza wymagań i celów**

* Zdefiniowanie funkcji i cech aplikacji
* Określenie docelowej grupy użytkowników

1. **Tworzenie specyfikacji**

* Dokładne opisanie funkcji i interfejsu użytkownika.
* Określenie technologii, które będą używane.

1. **Ustalenie harmonogramu**

* Określenie dat kluczowych etapów projektu.

**Faza 2: Implementacja**

1. **Nauka języka**
   * Research języków programowania
   * Wybranie odpowiedniego języka
   * Realizacja kursów
   * Ćwiczenia we własnym zakresie
2. **Budowa podstawowej struktury aplikacji**

* Stworzenie struktury plików
* Instalacja odpowiednich pakietów
* Tworzenie szkieletu HTML
* Tworzenie struktury bazy danych
* Tworzenie arkuszy stylów
* Zastosowanie architektury MVC

1. **Navbar**
   * Utworzenie panelu nawigacyjnego
   * Dodanie funkcji responsywności
2. **Panel składników**
   * Utworzenie podstrony do zarządzania składnikami
   * Integracja z bazą danych
   * Implementacja funkcjonalności panelu składników
3. **Panel magazynu**
   * Utworzenie panelu magazynu
   * Połączenie panelu magazynu z modelem składników
   * Utworzenie podstron dokładnych statystyk
4. **Panel dań**
   * Utworzenie panelu tworzenia dań
   * Interakcja z bazą danych
   * Utworzenie funkcjonalności koszyka
5. **Panel zamówień**

* Utworzenie panelu zamówień

**Faza 3: Testowanie**

1. **Testy jednostkowe**
   * Testowanie poszczególnych komponentów aplikacji.
2. **Testy integracyjne**
   * Sprawdzenie współpracy między różnymi częściami aplikacji.

**Faza 4: Korekty i doskonalenie**

1. **Optymalizacja**
   * Poprawa wydajności i optymalizacja kodu.
   1. **Wykres Gantta**
2. **Projekt GUI**

* Strona startowa

Obraz zawierający zrzut ekranu, tekst

Opis wygenerowany automatycznie

* Panel Magazynu

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

* Panel skladników

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

* Panel kreowania dania

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

* Panel zamówień

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

1. **Aktualna baza danych**
   1. **Cala baza danych**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie**

* 1. **Dania**

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

* 1. **Skladniki**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie**

* 1. **SkladnikSzablon**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie**

* 1. **Zamowienia**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie**

* 1. **Uzytkownicy**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie**

* 1. **Opis**

MongoDB to popularna, nierelacyjna baza danych typu NoSQL, która jest oparta na modelu dokumentowym. Poniżej opisuję podstawowe koncepcje i sposób działania bazy MongoDB:

**Dokumenty:**

W MongoDB dane przechowywane są w formie dokumentów BSON (Binary JSON), które są zbliżone do formatu JSON.

Dokumenty składają się z pól i wartości, a każdy dokument jest unikalny w obrębie kolekcji (analogii do tabel w relacyjnych bazach danych).

**Kolekcje:**

Kolekcje w MongoDB są zbiorem dokumentów. Są to odpowiedniki tabel w relacyjnych bazach danych, ale w MongoDB nie mają określonej struktury.

Kolekcje umożliwiają grupowanie dokumentów na podstawie wspólnego kontekstu.

**Bazy danych:**

Baza danych w MongoDB to zbiór kolekcji. Każda baza danych jest niezależna i może zawierać wiele kolekcji o różnych strukturach.

**ID dokumentów:**

Każdy dokument w kolekcji ma unikalny identyfikator, który domyślnie jest polem o nazwie "\_id".

Można używać różnych typów danych jako identyfikator, ale najczęściej stosowanym jest ObjectI

* 1. **Zastosowanie poszczególnych tabel**

**Dishes:**

Dokument dishes zawiera nastepujace informacje:

* \_id: id dania
* Name: nazwa dania
* Price: cena dania
* ingredientTemplates: informacje o składnikach z których zrobione jest danie, ceny skladnikow oraz ich ilość

**Ingredient-templates:**

Dokument ingredient-template ma na celu przechowywanie danych, które następnie pomoga w organizacji magazynu oraz tworzenia nowych dań. Oto jej skład:

* \_id: id szablony składnika
* Name: nazwa szablonu składnika (nazwa składnika)
* expirationDate: data waznosci
* category: kategoria

**Ingredients:**

Dokument zawiera informacje o składnikach, które sa aktualnie przechowywane w magazynie

* \_id: id składniku
* Name: nazwa składniku
* Price: cena składniku
* Weight: waga składniku
* expirationDate: data waznosci
* category: kategoria

**Orders:**

Dokument zawiera informacje o zamówieniach:

* \_id: id zamówienia
* Dishes: kolekcja, która zawiera szczegolowe dane o daniach które wchodzą w skład zamówienia
* Price: cena zamówienia

**Users:**

Zawiera informacje o użytkowniku

* Name: nazwa uzytkownika
* Email: email uzytkownika
* DishCart: koszyk, który pozwala na tworzenie zamówienia
* IngredientCart: koszyk, który pozwala na tworzenie dania

1. **Architektura aplikacji**

Aplikacja oparta jest na architekturze MVC, która jest popularnym wzorcem projektowym wykorzystywanym do strukturyzacji kodu źródłowego. MVC dzieli aplikację na trzy główne komponenty: Model, View i Controller, co ułatwia zarządzanie logiką biznesową, warstwą prezentacji i kontrolą przepływu danych.

**Model** odpowiada za zarządzanie danymi i logiką biznesową aplikacji. W moim przypadku:

Model danych: Przechowuje dane, zarządza dostępem do bazy danych (jeśli jest używana), i definiuje logikę biznesową. Wszelkie zmiany danych są odzwierciedlane w Modelu.

Interakcje z bazą danych: Odpowiada za pobieranie, aktualizację, dodawanie i usuwanie danych w bazie.

Definiuje strukturę komórki w dokumencie

W mojej aplikacji posiadam 5 modeli: dish, ingredient, ingredientTemplate, order oraz user które odpowiadają posczegolnym dokumentom w bazie danych.

Przykład modelu:

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

W warstwie **View** znajdują się elementy związane z prezentacją danych użytkownikowi. W naszym przypadku:

Szablony HTML/CSS/JavaScript: Odpowiadają za prezentację danych użytkownikowi.

Wyświetlanie danych: View otrzymuje dane z Modelu i prezentuje je w zrozumiały sposób dla użytkownika.

W mojej aplikacji widoku podzielone sa na katalogi:

Admin: który zawiera widoku podstron do zarządzania zasobami

Includes: który zawiera fragmenty kodu powtarzające się na podstronach np. footer i navbar

Restaurant: który zawiera widoki dostępne dla gości aplikacji webowej

**Kontroler** zarządza przepływem danych pomiędzy Modelem a Widokiem. W naszym przypadku:

Obsługa żądań użytkownika: Kontroler reaguje na żądania użytkownika (np. HTTP Request w aplikacjach webowych).

Przetwarzanie danych: Kontroler przetwarza dane wejściowe od użytkownika, komunikuje się z Modelem w celu uzyskania/zmiany danych i aktualizuje Widok.

1. **Plan rozbudowy aplikacji**

**Zamówienia i sprzedaż:**

Dzienny/tygodniowy/miesięczny obrót.

Najpopularniejsze dania

Średnia wartość zamówienia.

Trendy sprzedaży w czasie.

Predykcje sprzedażowe

Modyfikowanie menu

**Magazyn i zaopatrzenie:**

Poziom zapasów.

Najczęściej zamawiane produkty.

Historia dostaw i koszty zaopatrzenia.

Obliczanie strat związanych z marnowaniem jedzenia

**Personel:**

Wydajność pracowników.

Grafik pracy.

**Finanse:**

Rentowność danego dania lub kategorii produktów.

Analiza marży.

Platnosci cykliczne

Obliczanie oplacalnosci biznesu

Predykcje finansowe

**Zarządzanie rezerwacjami:**

Ilość rezerwacji online i offline.

Czas oczekiwania na stolik.

Historia rezerwacji klientów.

**Połączenie pracy menadżera, kucharza oraz kelnera za pomocą interfejsu, który przetwarza zamówienia**

1. **Opis funkcjonalności, które będą wprowadzane w przyszlosci**

**obliczanie ceny dania:**

skladnik w magazynie: 5kg pomidorow, cena 200zl

skladnik w jednym daniu: pomidor 100g

cena skladnika w daniu: 5000g = 200zl, 100g = 4zl

cena dania += cena pomidorow

**miesieczne cele sprzedazy:**

albo importuje sie z symulacji albo uzytkownik je ustala z gory

cel: sprzedanie 200 w miesiac

tablica z liczbami dni pracy (bedzie mozna je edytowac) w panelu admina

zalozmy z w maju jest 20 dni pracy: 200/20 = 10

sprzedane danie=200-1, pod koniec dnia pracowania restauracji 200-sprzedane,np 200-4=196

nastepnego dnia: 196/19 = 10.3 = 11, zawsze zaokraglam w gore

panel celow sprzedazy na kazdy dzien miesiąca

**panel dania:**

request: id\_dania:

findById(id\_dania)

populate(id\_dania.skladniki) -> bede mial skladniki gdzie pokazuja sie ich ceny

marza = cena dania - sumaSkladnikow

response: marza

**obliczanie przychodu:**

request: obszarCzasowy (np rok, kwartal itp. wybieram po prostu jaki przedzial czasowy)

request: danie

orders.find(order.items.danie == danie && obszarCzasowy)

suma = 0

for each order of orders

suma += marza (cena dania - cena skladnikow)

suma = tyle ile restauracja zarobi na tym daniu w okreslonym przedziale czasu

response: suma[]

**tworzenie wykresu:**

bedzie mozna ogladac ile sie chce ale dla uproszczenia podam przyklad na rok

request: okres czasowy

okresCzasu = [12], 12->length

orders.find(order.items.danie == danie && obszarCzasowy)

for each order in orders

suma += marza (cena dania - cena skladnikow)

if order.date = 01\* (styczen)

okresCzasu[1] = suma

response: okresCzasu[]

**zmiana ilosc porcji oraz symulacja:**

cel: jestem w statystykach dania i chce zmienic wartosc skladnikow zeby pobierac wieksza marze

jak to dziala:

wchodze w danie, skladniki zmieniam czyms w rodzaju suwaka procenty skladnikow

np kazdego skladnika jest o 10% mniej przez co cena produkcji spada o X%

biore aktualny miesiac np z luty i z requesta jaki czas ma pokazywac symulacja, np 3 miesiace

pseudokod:

request: id\_dania, okresCzasu, procentZmniejszeniaPorcji

findById(id\_dania)

populate(id\_dania.skladniki)

for each skladnik in skladniki

skladnik.cena -= 10%

danieTemplate.save() -> template bo sluzy tylko do symulacji, nie zmieniam rekordu prawdziwego dania w bazie

nastepne:

pobieram sobie dane dania ktoro zmienilem

pobieram dane ile to danie zostalo sprzedawane w 3 nastepnych miesiacach czyli marcu kwietniu i maju

POROWNYWANIE LAT

i zamieniam danie z danieTemplate

nastepnie oblicza mi ile zarobie za 3 nastepne miesiace na podstawie nowych cen

np. w 3 miesiace mialem 1000 zamowien no to 1000\*nowaCenaDania

**Porównywanie lat**

styczen 100

luty 120

marzec 90

kwiecien 105

120/100 \* 100 = 120 - wzroslo o 20%

90/120 \* 100 = 75 - zmalalo 0 25%

105/90 \* 100 = 116,6 - wzroslo o 16,6%

wejscie [100, 120, 90, 105] -> liczba zamowien dania na miesiac

wyniki [0, 120, 75, 116,6] -> tabela zmian procentowych

let zamowienia[24]

request: przedzial czasowy

orders.find(2 poprzednie lata)

for zamowienia.length

zamowienia[i] = order.liczbaZamowienZPosczegolnegoMiesiaca

//i tak dla 12 miesiacach rocznie i w sumie 24 bo 2 lata

let roznice[24]

for zamowienia.length

roznice[i] = zamowienia[i+1]/zamowienia[i]\*100

let poprzednieLata[12]

for zamowienia

poprzednieLata[i] = (zamowienie[i]+zamowienie[i+12])/2

//teraz biore miesiace z aktualnego roki

let tenRok[12]

orders.find(rok = aktualny)

for tenRok.length

tenRok[i] = orders[i].liczbaZamowienZPosczegolnychMiesiecy

let roznica

for tenRok.length

roznica = tenRok[i] - poprzednieLata[i]

let faktycznaRoznica = roznice/ilosc miesiecy jakie uplynely w tym roku

i teraz do roznic w dwoch ostatnich latach dodajemy faktyczna roznice i mamy symulacje aktualnego roku

for poprzednieLata.length

poprzednieLata[i] += faktycznaRoznica

Przykład:

Biore roznice miesięcy z dwóch poprzednich lat

[0, 110, 90, 103,4]

[0, 120, 80, 116,6] +

---------------------

[0. 230, 170, 220]

i dziele przez 2

[0, 115, 85, 110] -> i mam zblizone roznice miedzy miesiącami

teraz biore miesiace z aktualnego roku, tak samo obliczam ich przyrost

[0, 120, 80, 120]

aktualny rok jest odpowiednio lepszy o 5%, -5%, 10%

sumuje do siebie rozniceb = 10%

dziele to na 3 miesiace zeby uzyskac ile srednio wyniki roznia sie od poprzednich lat, 10/3=3.3%

patrzac na to ze w aktualnym roku wyniki sa lepsze o 2.5% to uwzgledniam to w predykcjach

wiec zamiast odpowiedni 115, 85 i 110 bedzie 118,3, 88,3 i 113,3 - to pozwala uzyskac dokladniejszy wynik

**Zmiana ceny dania**

chce obnizyc cene dania o 1zl i zobaczyc jaki to bedzie mialo wplyw na popularnosc dania

w przeszlosci bylo juz takie cos, w lutym a teraz jest lipiec

biore 3 miesiace po lutym oraz luty

luty 100

marzec 110

kwiecien 115

maj 125

obliczam jakie sa przyrosty miesieczne

110/100 \* 100 = 110

115/110 \* 100 = 104.54

125/115 \* 100 = 108.63

wyniki[0, 110, 104.54, 108.63]

nastepnie biore wyniki roznicy miesiecy w skali 2 poprzednich lat od zmiany ceny i ich srednia

np: [0, 103, 101, 104]

wiec wychodzi na to ze zamiast wzrostow na poziomie 10%, 4.5%, 8.6% sa zmiany w wysokosci 7%, 1.5% oraz 4.63% i sa to wyniki bardziej wiarygodne