

Projekt aplikacji sklepu internetowego z asortymentem cukierniczym.



Autorzy projektu:

Chmiel Maciej

Kamińska Agnieszka

Kotowska Justyna

Marchewka Maciej

Nytko Wojciech

Spis treści

1. Faza przygotowania	3
1.1 Wizja systemu	3
1.2 Technologie.....	3
1.3 Wzorzec architektoniczny	5
2. Faza analityczna	7
2.1 Model przypadków użycia	7
3. Faza projektowania.....	11
3.1 Wstęp.....	11
3.2 Tworzenie aplikacji	11
3.3 Dodawanie Dockera.....	12
3.4 Githubowe akcje.....	13
4. Faza implementacji	16

1. Faza przygotowania

1.1 Wizja systemu

Założenia oraz zależności

Wykonanie systemu wymaga zainstalowanie odpowiedniej infrastruktury. Należy zapewnić możliwość testowania systemu na możliwie największej ilości konfiguracji. Aplikacja powinna być zbudowana w technologii witryny internetowej. Ponadto aplikacja będzie obsługiwana przez wielu użytkowników, dlatego powinna być możliwość stworzenia indywidualnych kont oraz zastosowanie klasycznego logowania, po czym następuje możliwość dokonywania zamówień.

Korzyści klienta zamawiającego system

- Wysoka funkcjonalność systemu, pozwalająca mniej zaawansowanym użytkownikom na uzyskanie interesujących ich informacji,
- Spersonalizowana aplikacja zorientowana na potrzebach klienta,
- Usprawnienie sprzedaży, poprzez nabycie odpowiedniej aplikacji,

1.2 Technologie

ASP.Net

Zbiór technologii opartych na frameworku zaprojektowanym przez firmę Microsoft. Przeznaczony jest do budowy różnorodnych aplikacji internetowych, a także aplikacji typu XML Web Services. ASP.NET rozszerza platformę deweloperską .NET o narzędzia i biblioteki przeznaczone specjalnie do tworzenia aplikacji internetowych.

Strony ASP.NET są uruchamiane przy użyciu serwera, który umożliwia wygenerowanie treści HTML (wraz z CSS), WML lub XML – rozpoznawanych oraz interpretowanych przez przeglądarki internetowe. ASP.NET jest wspierany przez separujący warstwę logiki od warstwy prezentacji, wątkowo-kierowany model programistyczny, co poprawia wydajność działania aplikacji. Logika stron ASP.NET oraz XML Web Services może być tworzona w językach Visual Basic .NET, C# lub w dowolnym innym języku wspierającym technologię Microsoft .NET Framework lub .NET Core.

C#

Wieloparadygmatowy język programowania zaprojektowany w latach 1998–2001 przez zespół pod kierunkiem Andersa Hejlsberga dla firmy Microsoft. Program napisany w tym języku kompilowany jest do języka Common Intermediate Language (CIL), specjalnego kodu pośredniego wykonywanego w środowisku uruchomieniowym takim jak .NET Framework, .NET Core, Mono. Wykonanie skompilowanego programu przez system operacyjny wymaga zainstalowanego środowiska uruchomieniowego lub zawarcia go w dystrybuowanej aplikacji.

Cechy języka:

- Garbage Collector działający automatycznie;
- dostęp do standardowych bibliotek;
- delegaty oraz zarządzanie zdarzeniami (Delegates, Events Management);
- łatwość używania typów generycznych (Generics);
- kompilacja warunkowa;
- wielowątkowość;
- LINQ i wyrażenia lambda (lambda Expressions).

Microsoft SQL Server

System zarządzania bazą danych, wspierany i rozpowszechniany przez korporację Microsoft. Jest to główny produkt bazodanowy tej firmy, który charakteryzuje się tym, iż jako język zapytań używany jest przede wszystkim Transact-SQL, który stanowi rozwinięcie standardu ANSI/ISO.

MS SQL Server jest platformą bazodanową typu klient-serwer. W stosunku do Microsoft Jet, który stosowany jest w programie MS Access, odznacza się lepszą wydajnością, niezawodnością i skalowalnością. Przede wszystkim są tu zaimplementowane wszelkie mechanizmy wpływające na bezpieczeństwo operacji (m.in. procedury wyzwalane).

SQL

Strukturalny oraz deklaratywny język zapytań. Jest to język dziedziczny używany do tworzenia, modyfikowania relacyjnych baz danych oraz do umieszczania i pobierania danych z tych baz. Decyzję o sposobie przechowywania i pobrania danych pozostawia się systemowi zarządzania bazą danych.

Entity Framework

Narzędzie mapowania obiektowo-relacyjnego, tj. ORM. Generuje obiekty biznesowe oraz encje zgodnie z tabelami baz danych. Obiekt biznesowy jest encją (entity) w wielowarstwowej aplikacji, która działa w połączeniu z dostępem do bazy danych oraz warstwą logiki biznesowej służącą do przesyłania danych. Z kolei nasze entity odnosi się do czegoś co jest unikatowe i istnieje oddzielnie, np. tabela bazy danych. Entity Framework pozwala na:

- Wykonywanie podstawowych operacji CRUD (Create, Read, Update, Delete).
- Łatwe zarządzanie relacjami 1 do 1, 1 do wielu oraz wiele do wielu.
- Tworzenie relacji dziedziczenia pomiędzy encjami.

Zalety używania tego rozwiązania:

- Cała logika dostępu do bazy danych może być napisana w języku wyższego poziomu.
- Koncepcyjny model może zostać zaprezentowany w lepszy sposób za pomocą relacji pomiędzy encjami.
- Bazowe dane mogą być zastąpione bez większego narzutu ponieważ cała logika biznesowa dostępu do danych jest na wyższym poziomie.

Docker

otwarte oprogramowanie służące do realizacji wirtualizacji na poziomie systemu operacyjnego (tzw. „konteneryzacji”), działające jako „platforma dla programistów i administratorów do tworzenia, wdrażania i uruchamiania aplikacji rozproszonych”. Docker jest określany jako narzędzie, które pozwala umieścić program oraz jego zależności (biblioteki, pliki konfiguracyjne, lokalne bazy danych itp.) w lekkim, przenośnym, wirtualnym kontenerze, który można uruchomić na prawie każdym serwerze z systemem opartym. Kontenery wraz z zawartością działają niezależnie od siebie i nie wiedzą o swoim istnieniu. Mogą się jednak ze sobą komunikować w ramach ściśle zdefiniowanych kanałów wymiany informacji.

Dzięki uruchamianiu na jednym wspólnym systemie operacyjnym, konteneryzacja oprogramowania jest znacznie lżejszym (mniej zasobochłonnym) sposobem wirtualizacji niż pełna wirtualizacja za pomocą wirtualnych systemów operacyjnych.

Git

Git to rozproszony system kontroli wersji, co oznacza, że lokalny klon projektu jest kompletnym repozytorium kontroli wersji. Te w pełni funkcjonalne repozytoria lokalne ułatwiają pracę w trybie offline lub zdalnie. Deweloperzy zatwierdzają swoją pracę lokalnie, a następnie synchronizują kopię repozytorium z kopią na serwerze. Ten paradygmat różni się od scentralizowanej kontroli wersji, w której klienci muszą synchronizować kod z serwerem przed utworzeniem nowych wersji kodu. Elastyczność i popularność usługi Git sprawiają, że jest to doskonały wybór dla dowolnego zespołu. Wielu deweloperów i absolwentów uczelni już wie, jak korzystać z usługi Git. Społeczność użytkowników usługi Git utworzyła zasoby, aby wytrenować deweloperów i popularność usługi Git, co ułatwia uzyskanie pomocy w razie potrzeby. Prawie każde środowisko programistyczne ma obsługę usługi Git i narzędzia wiersza polecenia Git zaimplementowane w każdym głównym systemie operacyjnym.

Bootstrap

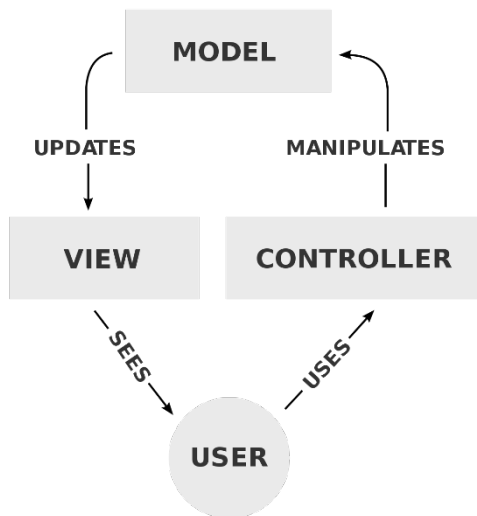
Biblioteka CSS, rozwijana przez programistów Twittera, wydawany na licencji MIT[1]. Zawiera zestaw przydatnych narzędzi ułatwiających tworzenie interfejsu graficznego stron oraz aplikacji internetowych. Bazuje głównie na gotowych rozwiązaniach HTML oraz CSS (kompilowanych z plików Less[2]) i może być stosowany m.in. do stylizacji takich elementów jak teksty, formularze, przyciski, wykresy, nawigacje i innych komponentów wyświetlanych na stronie. Biblioteka korzysta także z języka JavaScript.

[1.3 Wzorzec architektoniczny](#)

Wzorzec MVC

Wzorzec architektury Model-View-Controller (MVC) oddziela aplikację od trzech głównych grup składników: modele, widoki i kontrolery. Ten wzorzec pomaga osiągnąć separację obaw. Korzystając z tego wzorca, żądania użytkowników są kierowane do kontrolera, który jest odpowiedzialny za pracę z modelem w celu wykonywania akcji użytkownika i/lub pobierania

wyników zapytań. Kontroler wybiera widok, który ma być wyświetlany użytkownikowi, i udostępnia je dowolnym wymaganym danym modelu.



To rozdzielenie obowiązków ułatwia skalowanie aplikacji pod względem złożoności, ponieważ łatwiej jest kodować, debugować i testować (model, widok lub kontroler). Trudniej jest zaktualizować, przetestować i debugować kod, który ma zależności rozłożone na co najmniej dwa z tych trzech obszarów. Na przykład logika interfejsu użytkownika zwykle zmienia się częściej niż logika biznesowa. Jeśli kod prezentacji i logika biznesowa są łączone w jednym obiekcie, obiekt zawierający logikę biznesową musi być modyfikowany za każdym razem, gdy interfejs użytkownika zostanie zmieniony. Często wprowadza to błędy i wymaga ponownego testowania logiki biznesowej po każdej minimalnej zmianie interfejsu użytkownika.

Zalety MVC:

- Kody są łatwe w utrzymaniu i można je łatwo rozszerzać.
- Komponent modelu MVC można przetestować oddzielnie.
- Komponenty MVC mogą być rozwijane jednocześnie.
- Zmniejsza złożoność, dzieląc aplikację na trzy jednostki. Model, widok i kontroler.
- Obsługuje programowanie sterowane testami (TDD).
- Działa dobrze w przypadku aplikacji sieci Web obsługiwanych przez duże zespoły projektantów i programistów sieci Web.
- Ta architektura pomaga w niezależnym testowaniu komponentów, ponieważ wszystkie klasy i obiekty są od siebie niezależne
- Przyjazny dla optymalizacji pod kątem wyszukiwarek (SEO) .

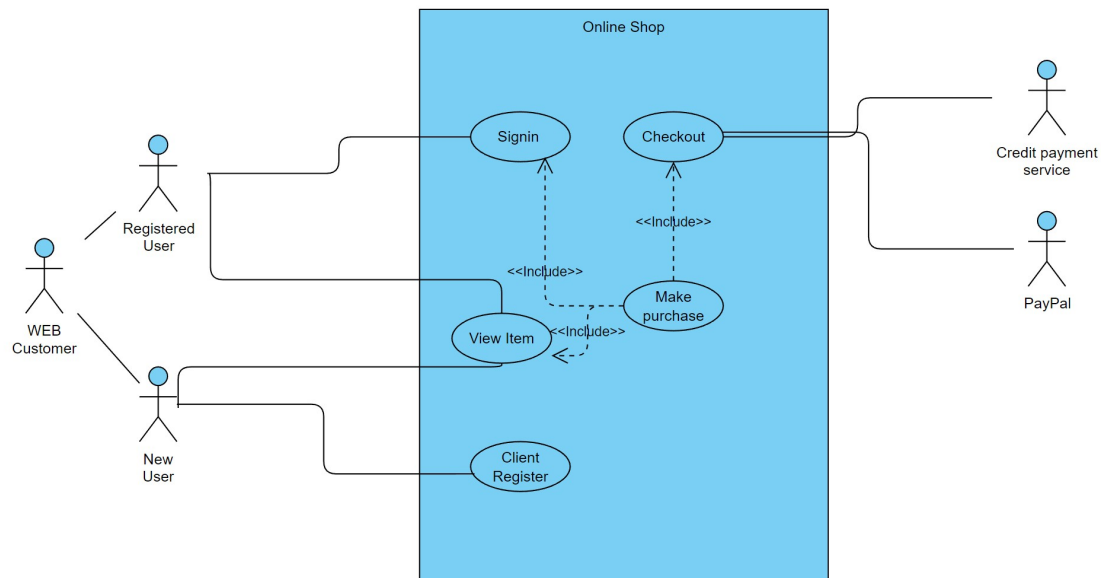
Wady MVC:

- Trudno jest odczytać, zmienić, przetestować i ponownie wykorzystać ten model
- W świetle nieefektywności dostępu do danych.
- Nawigacja po frameworku może być złożona, ponieważ wprowadza nowe warstwy abstrakcji, które wymagają od użytkowników dostosowania się do kryteriów dekompozycji MVC.
- Zwiększona złożoność i nieefektywność danych

2. Faza analityczna

2.1 Model przypadków użycia

Diagramy przypadków użycia



Model klas

Diagram klas – logowanie

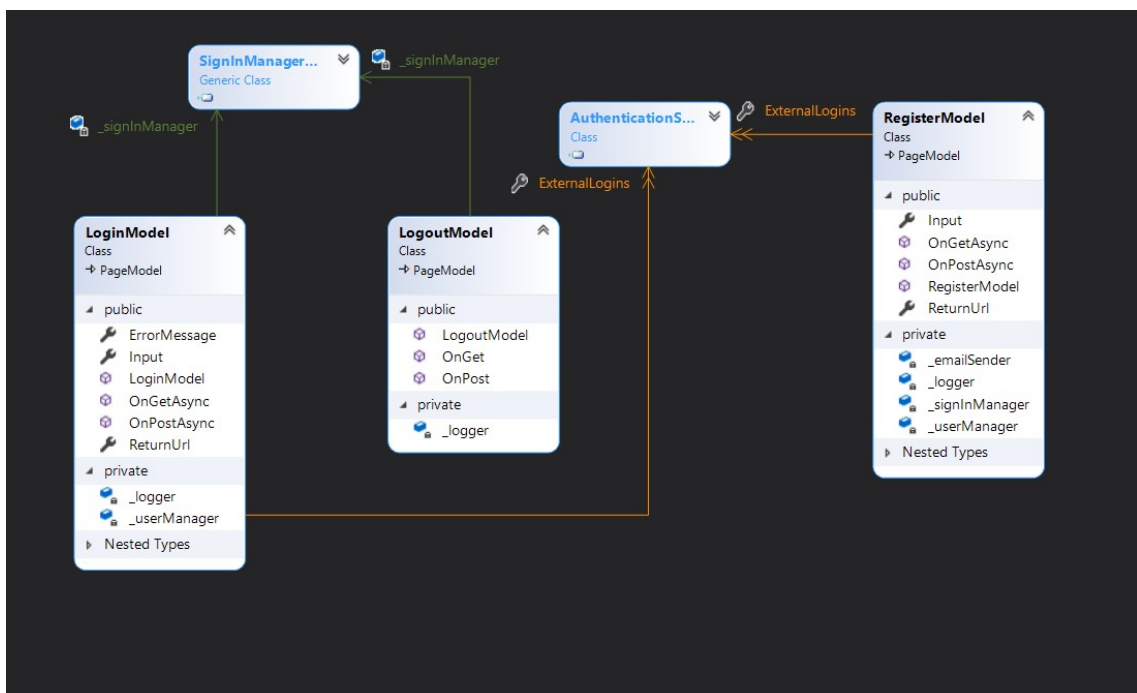


Diagram klas – koszyk sklepu

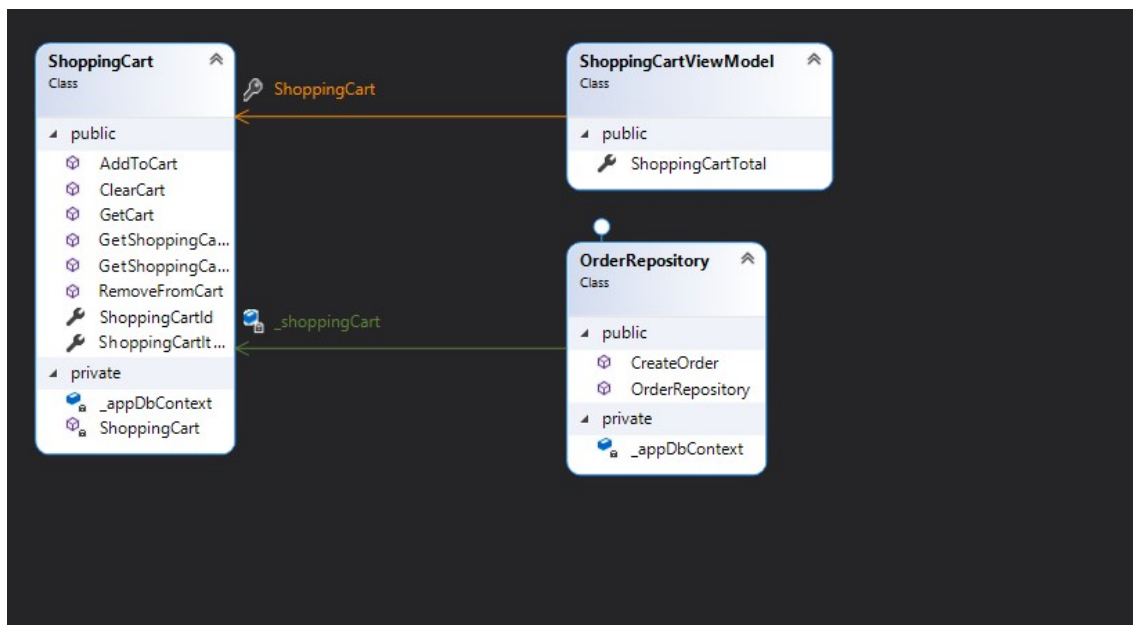
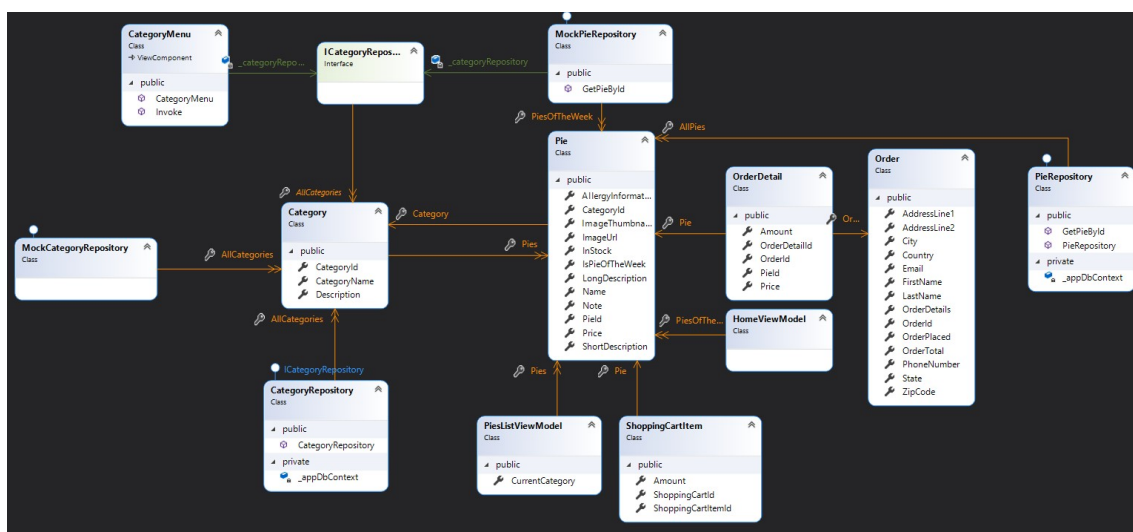


Diagram klas – Użytkowanie i zamawianie



Lista klas

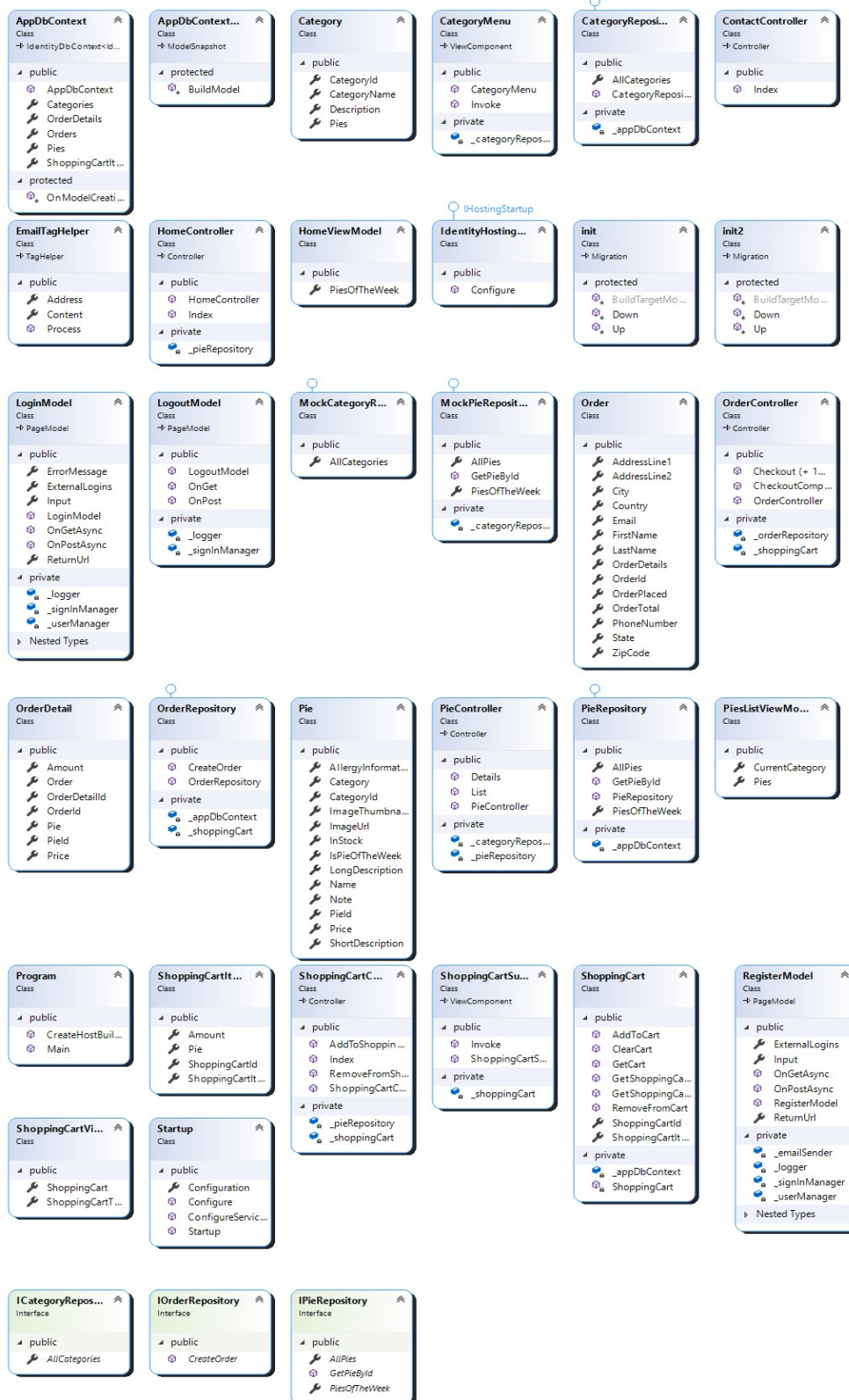
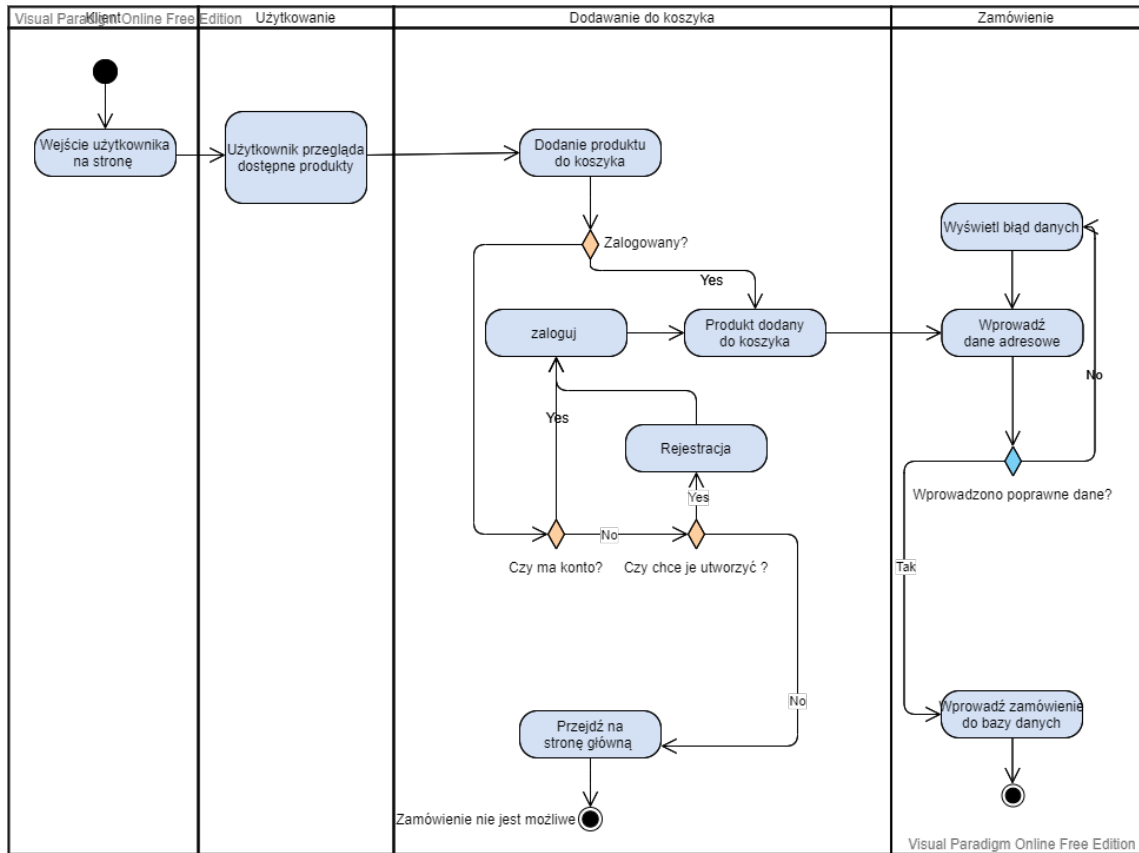


Diagram czynności



3. Faza projektowania

3.1 Wstęp

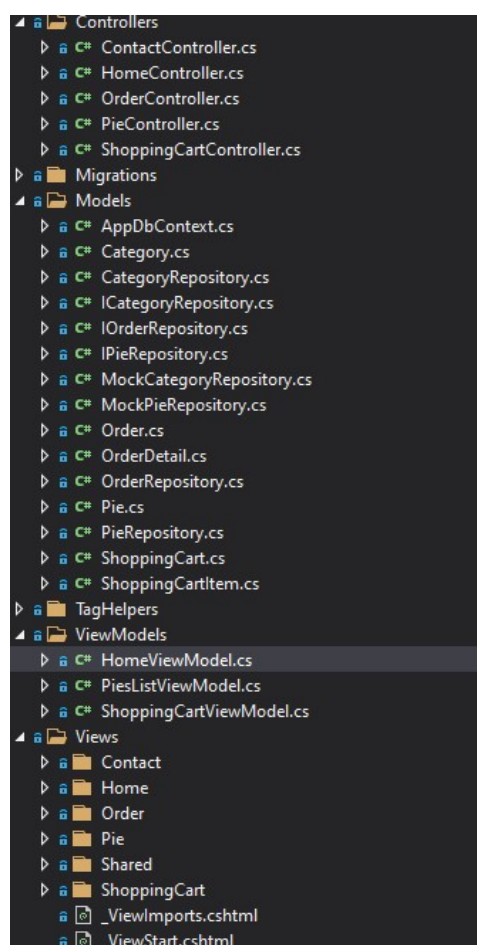
Wojciech Nytko stworzył projekt na GitHubie

https://github.com/NytWo/Projekt_System_Informatyczne.git, wraz ze szkieletem aplikacji o wzorcu architektonicznym MVC od której każda z osób miała zacząć swoją pracę. Powstał również schemat blokowy witryny. Dodatkowo zespół skonfigurował GitHuba i Visual Studio w celu stworzenie wspólnego środowiska do pisania kodu. Do projektu została doinstalowana biblioteka Bootstrap oraz JQuery umożliwiające szybkie tworzenie widoków.

3.2 Tworzenie aplikacji

Budowanie aplikacji rozpoczęliśmy od zaplanowania schematu bazy danych i relacji między encjami. W trakcie pisania kodu schemat ten rozrastał się. Baza danych jest tworzona poprzez migrację wykonywaną przez Entity Framework.

Kolejnym krokiem było stworzenie kontrolerów odpowiadających za komunikację użytkownika z danymi w bazie. W Kontrolerach zawarta jest logika zarządzająca działaniem aplikacji i przepływem danych oraz ich wyświetlaniu w widoku aplikacji.



Aplikacja była tworzona poprzez dodawanie kolejnych funkcjonalności i zakładkę w witrynie sklepu internetowego.

3.3 Dodawanie Dockera

Doker został skonfigurowany poprzez dodanie do projektu dwóch plików, który pierwszy odpowiada za stworzenie obrazu projektu i jest nim Dockerfile.

```
FROM mcr.microsoft.com/dotnet/aspnet:3.1 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:3.1 AS build
WORKDIR /src
COPY ["BethanysPieShop.csproj", "."]
RUN dotnet restore "./BethanysPieShop.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "BethanysPieShop.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "BethanysPieShop.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "BethanysPieShop.dll"]
```



Drugim plikiem jest Docker-compose, odpowiadający za utworzenie dwóch kontenerów i stworzenie łącza między nimi. W pierwszej kolejności stawiany jest obraz z serwerem SQL, a następnie obraz aplikacji stworzony poprzez wywołanie pliku dockerfile. Aplikacja zaraz po odpaleniu tworzy migrację do serwera bazy danych.

```
version: "3.9" # optional since v1.27.0

services:
  sql.data:
    image: "mcr.microsoft.com/mssql/server:2022-latest"
    ports: # not actually needed, because the two services are on the same network
      - "1433:1433"
    environment:
      - ACCEPT_EULA=y
      - SA_PASSWORD=1qaz@WSX
  web_api:
    image: web_api
    ports:
      - "5000:80"
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      - sql.data
```

3.4 Githubowe akcje

Ostatnią czynnością związaną z wykonaniem zadania było stworzenie githubowych akcji.














 Deploy.yml	Update Deploy.yml
 build-and-test.yml	Update build-and-test.yml

Zespół stworzył dwie akcje, pierwsza odpowiada za tworzenie nowego obrazu aplikacji i umieszczanie go na platformie DockerHub.

```
1  name: .NET
2
3
4  on:
5    push:
6      branches: [ "main" ]
7    pull_request:
8      branches: [ "main" ]
9
10 jobs:
11   build:
12
13     runs-on: ubuntu-latest
14
15     steps:
16     - uses: actions/checkout@v3
17       name: Setup .NET
18       uses: actions/setup-dotnet@v3
19       with:
20         dotnet-version: 6.0.x
21     - name: Restore dependencies
22       run: dotnet restore
23
24     - name: Build
25       run: dotnet build --no-restore
26
27     - uses: actions/checkout@v2
28       name: docker login
29       run: |
30         docker login -u ${ secrets.DOCKER_USERNAME }} -p ${ secrets.DOCKER_PASSWORD }}
31     - name: Get current date
32       id: date
33       run: echo "::set-output name=date::$(date +%Y-%m-%d--%M-%S)"
34     - name: Build the Docker image
35       run: docker build . --file Dockerfile --tag samau11/ae:${ steps.date.outputs.date }}
36     - name: Docker Push
37       run: docker push samau11/ae:${ steps.date.outputs.date }}
```

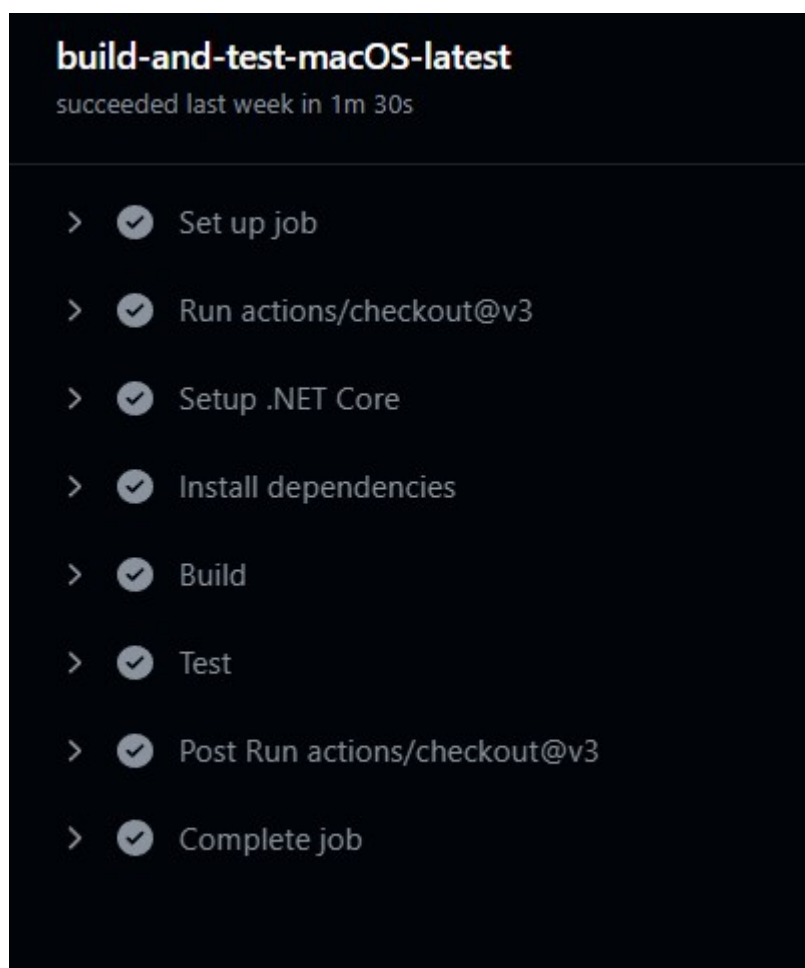
build

succeeded last week in 1m 25s

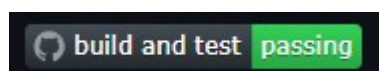
- >  Set up job
- >  Run actions/checkout@v3
- >  Setup .NET
- >  Restore dependencies
- >  Build
- >  Run actions/checkout@v2
- >  docker login
- >  Get current date
- >  Build the Docker image
- >  Docker Push
- >  Post Run actions/checkout@v2
- >  Post Run actions/checkout@v3
- >  Complete job

Druga akcja buduje aplikację i testuje ją w trzech środowiskach (Ubuntu, Windows, MacOS).

```
1  name: build and test
2  on:
3    push:
4      branches: [ "main" ]
5    pull_request:
6      branches: [ "main" ]
7    paths:
8      - '**.cs'
9      - '**.csproj'
10
11  env:
12    DOTNET_VERSION: '6.0.401'
13
14  jobs:
15    build-and-test:
16
17      name: build-and-test-${{matrix.os}}
18      runs-on: ${ matrix.os }
19      strategy:
20        matrix:
21          os: [ubuntu-latest, windows-latest, macOS-latest]
22
23      steps:
24        - uses: actions/checkout@v3
25        - name: Setup .NET Core
26          uses: actions/setup-dotnet@v3
27          with:
28            dotnet-version: ${{ env.DOTNET_VERSION }}
29
30        - name: Install dependencies
31          run: dotnet restore
32
33        - name: Build
34          run: dotnet build --configuration Release --no-restore
35
36        - name: Test
37          run: dotnet test --no-restore --verbosity normal
```



Wynik zwracany jest do pliku README.md



Obydwie akcje są wywoływane poprzez dodanie commita w projekcie.

4. Faza implementacji

Wymagania

Do uruchomienia programu wymagane jest pobranie projektu z platformy GitHub oraz zainstalowany Docker.

Instrukcja

Folder z projektem odpalamy w terminalu a następnie wpisujemy komendę „docker compose up”, która zacznie pobierać, tworzyć i instalować obrazy projektu. Po skończonym procesie w terminalu pojawią się informacje o postawionych obrazach na serwerze lokalnym. Obrazy pobrane i stworzone w tym procesie możemy zobaczyć w aplikacji Docker Desktop. Uruchomione Kontenery również będą widoczne w tym programie w zakładce Containers. Aplikację możemy otworzyć poprzez wpisanie adresu localhost:5000 w pasu przeglądarki,

natomiast baza danych jest dostępna w Microsoft SQL Server Management Studio pod adresem localhost,1433.

```
PS C:\Users\admin\source\repos\Projekt_System_Informatyczne> docker compose up
[+] Running 1/5
 - web_api Warning                                     1.8s
 - sql.data Pulling                                    17.0s
   - 342d87d17479 Pull complete                        6.6s
   - 112c1458d0bd Downloading [=====>]...          16.4s
   - 04016b3a8e25 Download complete                  16.4s
```

```
Windows PowerShell
0023-01-25 09:56:41.60 spid68      [5]. Feature Status: PVS: 0. CTR: 0. ConcurrentPFSUpdate: 1. ConcurrentGAMUpdate: 1.
ConcurrentSGAMUpdate: 1, CleanupUnderUserTransaction: 0. TranLevelPVS: 0
0023-01-25 09:56:41.61 spid68      Starting up database 'PieShopDB'.
0023-01-25 09:56:41.61 spid68      RemoveStaleDbEntries: Cleanup of stale DB entries called for database ID: [5]
0023-01-25 09:56:41.61 spid68      RemoveStaleDbEntries: Cleanup of stale DB entries skipped because master db is not me
Memory optimized. DbId: 5.
0023-01-25 09:56:41.63 spid68      Parallel redo is started for database 'PieShopDB' with worker pool size [8].
0023-01-25 09:56:41.64 spid68      Parallel redo is shutdown for database 'PieShopDB' with worker pool size [8].
0023-01-25 09:56:41.71 spid68      Setting database option READ_COMMITTED_SNAPSHOT to ON for database 'PieShopDB'.
RecoveryUnit::Shutdown. IsOnline: 0023-01-25 09:56:41.75 spid68      RBPEX::NotifyFileShutdown: Called for database ID:
[5], file id [0]
0023-01-25 09:56:41.77 spid68      [5]. Feature Status: PVS: 0. CTR: 0. ConcurrentPFSUpdate: 1. ConcurrentGAMUpdate: 1.
ConcurrentSGAMUpdate: 1, CleanupUnderUserTransaction: 0. TranLevelPVS: 0
0023-01-25 09:56:41.77 spid68      Starting up database 'PieShopDB'.
0023-01-25 09:56:41.77 spid68      RemoveStaleDbEntries: Cleanup of stale DB entries called for database ID: [5]
0023-01-25 09:56:41.77 spid68      RemoveStaleDbEntries: Cleanup of stale DB entries skipped because master db is not me
Memory optimized. DbId: 5.
0023-01-25 09:56:41.79 spid68      Parallel redo is started for database 'PieShopDB' with worker pool size [8].
0023-01-25 09:56:41.82 spid78s    Parallel redo is shutdown for database 'PieShopDB' with worker pool size [8].
projekt_system_informatyczne-web_api-1 | info: Microsoft.Hosting.Lifetime[0]
projekt_system_informatyczne-web_api-1 | Now listening on: http://[::]:80
projekt_system_informatyczne-web_api-1 | info: Microsoft.Hosting.Lifetime[0]
projekt_system_informatyczne-web_api-1 | Application started. Press Ctrl+C to shut down.
projekt_system_informatyczne-web_api-1 | info: Microsoft.Hosting.Lifetime[0]
projekt_system_informatyczne-web_api-1 | Hosting environment: Production
projekt_system_informatyczne-web_api-1 | info: Microsoft.Hosting.Lifetime[0]
projekt_system_informatyczne-web_api-1 | Content root path: /app
projekt_system_informatyczne-web_api-1 | warn: Microsoft.AspNetCore.HttpsPolicy.HttpsRedirectionMiddleware[3]
projekt_system_informatyczne-web_api-1 | Failed to determine the https port for redirect.
```

Images

Volumes

Dev Environments BETA

Extensions BETA

Extensions are disabled

Images Give feedback

An image is a read-only template with instructions for creating a Docker container. [Learn more](#)

LOCAL

REMOTE REPOSITORIES

Search

<input type="checkbox"/>	NAME	TAG	STATUS	CREATED	SIZE	ACTIONS
<input type="checkbox"/>	web_api 707a736e2af6	latest	In use	less than a minute a	248.35 MB	<div></div> <div></div> <div></div>
<input type="checkbox"/>	mcr.microsoft.com/mssql/server 9e28798be691	2022-latest	In use	3 months ago	1.6 GB	<div></div> <div></div> <div></div>

