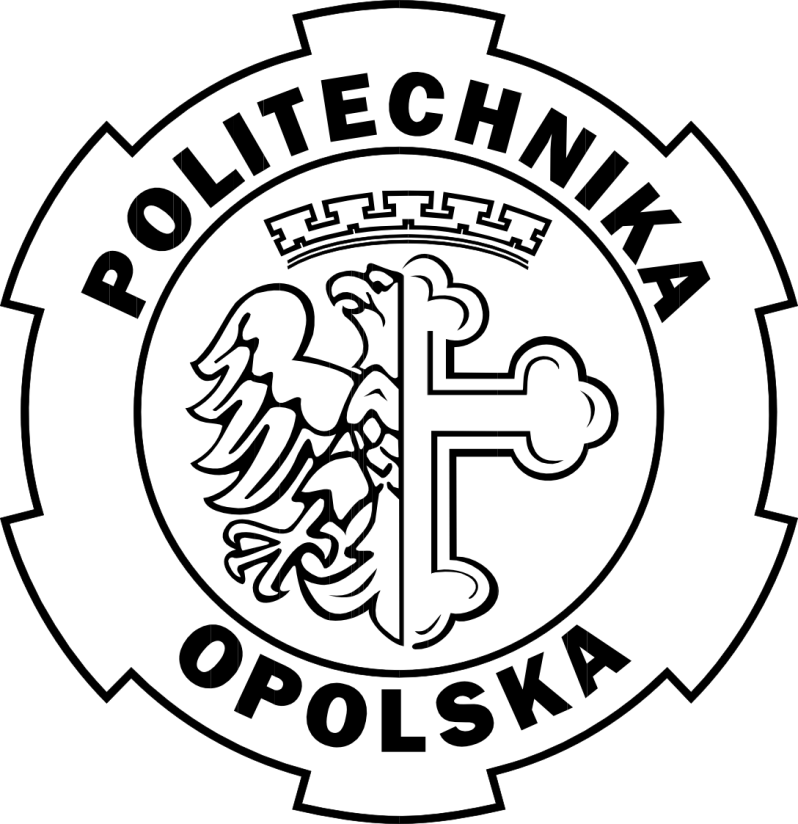
**Politechnika Opolska**

Wydział Elektrotechniki, Automatyki i Informatyki

Instytut Automatyki i Informatyki

Kierunek Informatyka

Studia Dzienne



**Praca dyplomowa inżynierska**

Projekt i implementacja systemu budowania i administrowania stronami internetowymi przez personel nietechniczny – warstwa prezentacji

|  |  |
| --- | --- |
| Promotor: | Autor: |
| dr hab. inż. Rafał Stanisławski, prof. PO | Artur Stelmach |

Opole, 2016

Spistreści

[1. Wprowadzenie 5](#_Toc441334984)

[2. Cel i zakres pracy 6](#_Toc441334985)

[3. Narzędzia wykorzystane w projekcie 8](#_Toc441334986)

[3.1. Serwer 8](#_Toc441334987)

[3.2. Platforma .NET 9](#_Toc441334988)

[3.3. Język C# 9](#_Toc441334989)

[3.4. AngularJS 10](#_Toc441334990)

[3.4.1. Text-angular 10](#_Toc441334991)

[3.4.2. Angular-chart 10](#_Toc441334992)

[3.4.3. Ui-Calendar 11](#_Toc441334993)

[3.5. LINQ 11](#_Toc441334994)

[3.6. Kontrola wersji 11](#_Toc441334995)

[3.7. Producteev 12](#_Toc441334996)

[3.8. RestSharp 12](#_Toc441334997)

[3.9. Wzorce projektowe 13](#_Toc441334998)

[3.9.1. Unit of work 13](#_Toc441334999)

[3.9.2. Wstrzykiwanie zależności 13](#_Toc441335000)

[3.9.3. Repozytorium 14](#_Toc441335001)

[3.9.4. MVC 14](#_Toc441335002)

[3.10. Visual Studio 14](#_Toc441335003)

[3.11. Autofac 15](#_Toc441335004)

[4. Projekt systemu wspólnie z Kamilem Ślusarczykiem 16](#_Toc441335005)

[4.1. Wymagania biznesowe 16](#_Toc441335006)

[4.2. Słownik pojęć 16](#_Toc441335007)

[4.3. Wymagania funkcjonalne 17](#_Toc441335008)

[4.3.1. Specyfikacja wymagań dla panelu administratora 17](#_Toc441335009)

[4.3.2. Specyfikacja wymagań dla panelu klienckiego 18](#_Toc441335010)

[4.3.3. Wymagania niefunkcjonalne 18](#_Toc441335011)

[4.4. Warstwy aplikacji 18](#_Toc441335012)

[4.5. Projekt bazy danych 19](#_Toc441335013)

[4.6. Diagramy sekwencji 22](#_Toc441335014)

[4.6.1. Rejestracja na usługę 22](#_Toc441335015)

[4.6.2. Edycja aktualności 23](#_Toc441335016)

[4.6.3. Usuwanie aktualności 24](#_Toc441335017)

[4.6.4. Wyświetlenie statystyk 24](#_Toc441335018)

[4.6.5. Skorzystanie z formularza kontaktowego 25](#_Toc441335019)

[4.6.6. Przekazanie plików na serwer 26](#_Toc441335020)

[5. Prezentacja systemu 27](#_Toc441335021)

[5.1. Ekran logowania 27](#_Toc441335022)

[5.2. Pulpit administracyjny 28](#_Toc441335023)

[5.3. Moduł aktualności 29](#_Toc441335024)

[5.4. Moduł plików 30](#_Toc441335025)

[5.5. Moduł stron 31](#_Toc441335026)

[5.6. Moduł przycisków menu 32](#_Toc441335027)

[5.7. Moduł newslettera 33](#_Toc441335028)

[5.8. Moduł ustawień globalnymi 34](#_Toc441335029)

[5.9. Moduł usług 34](#_Toc441335030)

[5.10. Moduł dostarczyciela usług 35](#_Toc441335031)

[5.11. Moduł Pop-up 36](#_Toc441335032)

[5.12. Moduł zarejestrowanych usług 36](#_Toc441335033)

[5.13. Moduł Statistics 37](#_Toc441335034)

[5.14. Ekran aplikacji klienckiej 39](#_Toc441335035)

[5.15. Ekran ładowania 40](#_Toc441335036)

[6. Budowa systemu 41](#_Toc441335037)

[6.1. Funkcjonalności w warstwie logiki biznesowej 41](#_Toc441335038)

[6.1.1. Menadżer sesji 41](#_Toc441335039)

[6.1.2. Obsługa ustawień regionalnych 42](#_Toc441335040)

[6.1.3. Generyczne repozytorium 43](#_Toc441335041)

[6.1.4. FluentApi 44](#_Toc441335042)

[6.1.5. Unit of Work 45](#_Toc441335043)

[6.1.6. Logger 46](#_Toc441335044)

[6.1.7. Wstawki 46](#_Toc441335045)

[6.1.7.1. InsetParser 47](#_Toc441335046)

[6.1.7.2. InsetRecognizer 48](#_Toc441335047)

[6.1.7.3. ArgumentValidator 49](#_Toc441335048)

[6.1.7.4. LocalLinkParser 50](#_Toc441335049)

[6.1.8. Timeblocks 50](#_Toc441335050)

[6.1.9. Settings 52](#_Toc441335051)

[6.1.10. Dependency injection 53](#_Toc441335052)

[6.1.11. Cryptography 54](#_Toc441335053)

[6.1.12. FileManager 55](#_Toc441335054)

[6.1.13. BaseController 55](#_Toc441335055)

[6.1.14. Moduł File 57](#_Toc441335056)

[6.2. Funkcjonalności frontowe 58](#_Toc441335057)

[6.2.1. Mechanizm paczkowania 59](#_Toc441335058)

[6.2.2. Moduł aplikacji 60](#_Toc441335059)

[6.2.3. Serwisy 60](#_Toc441335060)

[6.2.4. Walidacja formularzy 61](#_Toc441335061)

[6.2.5. Podstawowe operacje CRUD 62](#_Toc441335062)

[6.2.6. PopUp 69](#_Toc441335063)

[6.2.7. Statystki 69](#_Toc441335064)

[6.2.8. Animacja ładowania 70](#_Toc441335065)

[6.2.9. Edycja tekstu 71](#_Toc441335066)

[6.2.10. Wstawki 72](#_Toc441335067)

[7. Podsumowanie 75](#_Toc441335068)

[8. Bibliografia 76](#_Toc441335069)

# Wprowadzenie

Początek rozwoju technologii webowych datuje się na przełom lat 80 i 90. Były to wówczas aplikacje tworzone przez pasjonatów bez większego praktycznego znaczenia. Z czasem jednak proces ten wyszedł poza wąską sferę twórców i dotarł do mas. Nagle stało się oczywiste, że rozwój informatyki prowadzi do polepszenia organizacji życia, zwiększenia możliwości zarobkowych oraz udsostępnia wiele udogodnień, które ułatwiają codzienne życie. Gwałtowny wzorst dostępności Internetu sprawił, że konieczne jest wprowadzenie osób nietechnicznych do używanych technologii. Można stwierdzić, że postępująca informatyzacja społeczeństwa wręcz wymusza na nas adaptację do nowych technologii.

Poniższa aplikacja ma ułatwić budowanie stron internetowych osobom nietechnicznym. Jest ona wykonana za pomocą technologii firmy Microsoft, z użyciem frameworku .NET i AngularJS.

Omawiana aplikacja została napisana w języku C# w wersji 5.0. Język ten po raz pierwszy został wypuszczony na rynek w 2000 roku. Narzędzie to było wtedy wykorzystywane do budowania dużych systemów informatycznych opartych na systemie operacyjnym Windows. Dzięki ogromnej popularności tego języka, aplikacje budowane w .NET-cie wciąż zyskują na popularności. Wpływ na to ma też obniżenie cen rozwiązań serwerowych, dedykowanych dla tego typu aplikacji.

Do wykonania systemu posłużono się architekturą MVC (Model-View-Controller), która umożliwia rozwój oprogramowania w sposób szybki i skalowalny. Dzięki wykorzystaniu tego wzorca osiągnięto również stosunkowo małą ilość błędów co bezpośrednio wpłynęło na czas, jaki należało poświęcić na rozwój aplikacji. Architektura MVC odziela warstwę prezentacji, która jest widoczna dla użytkownika docelowego, od warstwy logiki i danych. Podział ten umożliwia swobodny rozwój kolejnych modułów bez obaw o błędy, wynikające z powiązań pomiędzy nimi.

# Cel i zakres pracy

Celem pracy jest przeprowadzenie oraz opis procesu projektowania i implementacji systemu, który umożliwi tworzenie i administrowanie stronami internetowymi przez osoby nietechniczne. W poniższej pracy opisano zarówno technologie, jak i wzorce projektowe, użyte do wykonania aplikacji. Wyszczególniono również przewagi wykorzystanych technologii, które doprowadziły do ich wyboru. Opisano szczegółowo moduły i hierarchię warstw, dzięki której możliwe było wykreowanie możliwie najbardziej elastycznego i skalowalnego systemu. Podczas implementacji autor miał na uwadze wytyczne, pochodzące z dokumentacji technicznej firmy Microsoft,która odpowiada za produkty użyte do wykonania programu.

Przedstawiono narzędzia i technologie, użyte do implementacji warstwy wizualnej aplikacji, jak i części logiki biznesowej, aplikacji ServiceCMS. Opis ten będzie dotyczył panelu administracyjnego oraz klienckiego. Najważniejsze narzędzia, które zostaną opisane, to: platforma .NET, AngularJS, Visual Studio. Opisano również wzorce projektowe, które wykorzystano do implementacji funkcjonalności.

Koncepcja projektu zakładała stworzenie aplikacji przyjaznej użytkownikowi nietechnicznemu. Wymagało to wiedzy zarówno odnośnie technologii programistycznych, używanych do implementacji logiki, jak i znajomości narzędzi, które umożliwiają estetyczną prezentację zawartości. Aby osiągnąć zamierzony cel, zdecydowano, że praca zostanie podzielona pomiędzy dwie osoby – Artura Stelmacha i Kamila Ślusarczyka. Poniższa praca opisuje część warstwy prezentacji. Kompletny opis warstwy logiki biznesowej zostanie przedstawiony w pracy pod tytułem *Projekt i implementacja systemu administrowania stronami internetowymi – część logiki biznesowej*, napisanej przez Kamil Ślusarczyka.

Praca jest podzielona na 2 części. W pierwszej części opisano założenia projektu oraz narzędzia, które zostały wykorzystane do jego wdrożenia. W części tej zostaną opisane również wzorce programistyczne i fakty, które implikowały wybór określonych rozwiązań technologicznych.

Druga część jest częścią praktyczną, która przedstawia szczegóły wdrożeniowe i opisuje podejście do problemów, jakie wypracowano w trakcie fazy implementacji funkcjonalności. Opis będzie uformowany zgodnie z modułową budową aplikacji i przedstawi część kodu źródłowego oraz schematy graficzne projektu.

W podsumowaniu opisano różnice pomiędzy założeniami, które określono podczas projektowania aplikacji, a rezultatem, który osiągnięto. Przeanalizowano również to, czy wybrane narzędzia i technologie spełniły zadania, których treść sformułowano w fazie projektowania systemu. Zawarto również analizę różnic, pomiędzy wybranymi narzędziami a ich możliwymi zamiennikami.

Zakres pracy obejmuje 7 rozdziałów:

1. Wprowadzenie – wprowadzenie do problemu, których zdecydowano opisać w poniższej pracy.

2. Cel i zakres pracy – przedstawienie celów, które zrealizowane w fazie implementacji. Opisano również zakres przeprowadzonych prac.

3.Narzędzia wykorzystane w projekcie –opis narzędzi i technologii, których użyto do implementacji wymagań funkcjonalnych systemu.

4. Projekt systemu – przedstawienie przypadków użycia oraz wymagań funkcjonalnych i niefunkcjonalnych

5. Prezentacja systemu – opis oraz wizualizacja graficznego interfejsu użytkownika i ogólnych zasad działania systemu.

6. Budowa systemu – opis konkretnych zadań programistycznych oraz przedstawienie sposobu ich rozwiązania za pomocą języka C# oraz JavaScript

7. Podsumowanie – porównanie przyjętych założeń z uzyskanymi wynikami. Zwrócono uwagę na różnice pomiędzy założeniami i wynikami oraz analiza uzyskanego systemu przez pryzmat realiów rynkowych.

# Narzędzia wykorzystane w projekcie

Obecnie istnieje mnóstwo różnorodnych technologii i narzędzi, które umożliwiają dogodny sposób rozwoju aplikacji internetowych. Dokonanie wyboru pomiędzy nimi jest zazwyczaj uzależnione od relacji koniecznych nakładów do zysku, jakie może wygenerować takie rozwiązanie. Z powodu rosnącej popularności treści zamieszczonych w Internecie, autorzy oprogramowania muszą poświęcić dużo czasu na opracowanie interaktywnego i estetycznego systemu.

Do wykonania opisywanego projektu wykorzystano technologie firmy Microsoft. Technologie te jeszcze do niedawna były wykorzystywane głównie do rozwoju dużych aplikacji biznesowych. Obecnie jednak wraz z obniżeniem cen rozwiązań serwerowych jak i darmowemu udostępnieniu środowiska pracy Visual Studio przez Microsoft następuje wzrost popularności tej technologii.

W celu szczegółowego opisu warstwy prezentacji przedstawiono technologie związane z językiem JavaScript. Umożliwiły one stworzenie estetycznej i interaktywnej warstwy wizualnej w stosunkowo krótkim czasie, spełniając jednocześnie kryteria dotyczące bezawaryjności i skalowalności.

## Serwer

Przyjęto, że z system będzie rozwijany lokalnie, na komputerach obu autorów. Aby umożliwić uruchomienie aplikacji konieczne było uruchomienie instancji serwera. Posłużono się do tego programem IIS Express wbudowanym w Visual Studio. Jest to zbiór usług internetowych, który podczas prac programistycznych pełnił rolę serwera HTTP . Z racji korzystania z platformy .NET niemożliwe było wykorzystanie innego rozwiązania. Można jednak stwierdzić, że IIS Express spełnił swoje zadanie i w żaden sposób nie spowalniał ani nie utrudniał prac programistyczych.

## Platforma .NET

Do wykonania aplikacji skorzystano z platformy .NET. Jest to bogata w funkcje platforma programistyczna, opracowana przez firmę Microsoft. Obejmuje ona środowisko uruchomieniowe oraz biblioteki klas, które są odpowiedzialne za podstawowe funkcjonalności aplikacji. Platforma ta umożliwia programowanie w wielu językach – na potrzeby realizacji projektu przyjęto język C#. Platforma .NET zarządza takimi częściami systemu, jak: kod źródłowy aplikacji, sposób zarządzania pamięcią, obsługa wyjątków oraz zabezpieczenia. Możlwe jest wykorzystanie wyżej opisanej platformy zarówno do aplikacji internetowych, jak i desktopowych. W opisywanym przypadku wybrano środowisko sieciowe, które jest zarządzane przez serwer internetowy IIS (więcej w rozdziale….). W skład platformy .NET wchodzą zarówno kompilatory języków wysokiego poziomu (C#, VB, C++), jak i kompilatory JIT. Nie są one jednak tematem poniższej pracy i nie będą dokładnie opisywane. Kompilator języka C# odpowiada za konwersję do języka pośredniego CIL (Common Intermediate Language) który następnie jest tłumaczony przez kompilator JIT do kodu maszynowego.

## Język C#

Do wykonania aplikacji zastosowano język C#. Jest to silnie typowany, obiektowy język programowania rozwijany przez firmę Microsoft od 2000 roku. Jest jednym z języków, zaprojektowanych na CLR. Umożliwia on budowę aplikacji z zachowaniem wydzielenia warstw abstrakcyjnych, co jest zgodne z paradygmatem programowania obiektowego. Dzięki zastosowaniu wzorców projektowcyh (patrz punkt 3.9.) możliwa była implementacja funkcjonalności zgodnie z zasadami SOLID. Wzorce umożliwiły również eliminację twardych zależności (patrz rozdział ….) co znacznie zwiększyło skalowalność kodu.

Na etapie projektu założono, że aplikacja będzie realizowana w metodologii code-first. Polega ona na implementacji klas typu POCO języka C# i konwersji ich na tabele w bazie danych. W celu przeprowadzenia tej konwersji zastosowano Enity Framework, który jest narzędziem typu ORM (Object Relational Mapping). Jest to najpopularniejszy mapper dla platformy .NET dostępny na rynku. Alternatywą była biblioteka Nhibernate, która umożliwia bardzo podobny zbiór funkcjonalności, co EF. Wybór w tym przypadku był podyktowany świetnym wsparciem technologii EF, w porównaniu do Nhibernate oraz wcześniejszym doświadczeniem tej technologii

## AngularJS

AngularJS jest biblioteką programistyczną, rozpowszechnianą na licencji MIT przez firmę Google. Jest to biblioteka języka JavaScript, przeznaczona do rozwoju aplikacji internetowych. Dziękie zastosowaniu opisywanego narzędzia możliwe jest oddzielenie zmiany struktury dokumenty (DOM) od logiki aplikacji. Poprawne wykorzystanie AngularJS umożliwia oddzielenie warstwy klienckiej od warstwy serwerowej. Od strony architektury, AngularJS implementuje wzorzec MVC (Model-View-Controller) i zaleca utrzymywanie słabych zależności pomiędzy warstwami logiki, prezentacji i danych. Dzięki czytelności kodu w porównaniu do jQuery uzyskano warstwę łatwą do testowania, skalowalną i odporną na awarie.

## Text-angular

Text-angular jest darmowym edytorem tekstowym, który realizuje proces edycji tekstu zgodnie z zasadą WYSIWYG (ang. What you see is what you get). Umożliwia on edycję zawartości aktualności, stron oraz pop-up’a za pomocą tagów HTML. Text-angular sprawia, że możliwe jest również zaawansowana edycja tekstu i jego formy – możliwa jest między innymi justowanie, wybór formy wcięć oraz stworzenia listy punktorów. Text-angular jest rozpowszechniany na licencji open-source. Został on wybrany z powodu dużej dokumentacji.

## Angular-chart

Angular-chart jest darmową biblioteką programistyczną, rozpowszechnianą na zasadach licencji BSD. Umożliwia ona stworzenie estetycznych i responsywnych wykresów, które w aplikacji ServiceCMS były wykorzystane w module Statistics. Angular-chart znacznie obniżył nakład pracy, który należało ponieść w celu implementacji funkcjonalności tego modułu. Dzięki zachowaniu konwencji rozwoju dobrego oprogramowania, zachowanej przez autorów tej biblioteki praca z nią nie nastręczała żadnych problemów.

## Ui-Calendar

Ui-calendar to darmowa biblioteka oparta o AngularJS dzięki której wyświetlanie wszelkiego rodzaju kalendarzy jest dużo prostsze i szybsze. Została użyta do obsługi rejestracji usług. Została wybrana ze względu na licencję MIT która umożliwia darmowe wykorzystywanie i ze względu na udostępnione duże i elastyczne API.

## LINQ

System ServiceCMS w dużej mierze opiera się na zwinnym zarządzaniu operacjami CRUD (Create-Read-Update-Delete) na zbiorach encji biznesowych. Z tego powodu konieczne było wypracowanie procedury postępowania z operacjami na kolekcjach. Należało mieć na uwadzę zarówno czystość, prostotę kodu jak i kwestie wydajności oraz bezawaryjności. Naturalnym wyborem dla platformy .NET jest rozszerzenie LINQ (Language Integrated Query). Dostarcza ono bogaty zbiór metod do przeprowadzenia wszelkich operacji na zbiorach obiektów. Dzięki mechanizmowi Lazy Loading (ang. Opóźnione wywołanie) wbudowanemu w LINQ rozwiązanie to spełnia również kryteria wydajnościowe. LINQ zostało uznane za rozwiązanie bezkonkurencyjne i nie rozważano innych rozszerzeń, które mogłyby zostać użyte podczas implementacji ServiceCMS.

## Kontrola wersji

Koncepcja systemu zakładała realizację w dwuosobowym zespole programistycznym. Z tego powodu konieczna była organizacja projektu za pomocą systemu kontroli wersji. Do przechowywania plików źródłowych wykorzystano serwis Github. Jest to najbardziej popularna platforma, umożliwiająca wersjonowanie za pomocą systemu kontroli wersji Git. Do obługi repozytorium wykorzystano bezpłatny klient Source Tree. Umożliwia on zarówno zatwierdzanie zmian, jak i ewentualny powrót do wcześniejszych faz projektu w przypadku napotkania błędów. Rozważano również wykorzystanie innych klientów przeznaczonych do wersjonowania. Wśród nich były między innymi – TFS i Tortoise. Z uwagi jednak na podobieństwo tych rozwiązań wybrano darmowe i nieskomplikowane narzędzie, które jednocześnie realizowało wszystkie założenia projektowe.

## Producteev

W trakcie tworzenia aplikacji konieczny był szczegółowy podział pracy pomiędzy osoby, realizujące projekt. Na etapie projektu wyszczególniono obszary, za które odpowiedzialni będą poszczególni autorzy. Formalna organizacja została przeprowadzona za pomocą aplikacji Producteev. Służy ona do prrzydzielania zadań osobm, biorącym udział w projekcie. Zadanie te posiadają statusy (ang. States) oraz możliwe pod-zadania (ang. Subtask). Te podstawowe funkcjonalności były w zupełności wystarczające do organizacji zadań programistycznych – każda bardziej rozbudowana aplikacja kreowała by jedynie dodatkowe obowiązki, związane z jej obsługą.

## RestSharp

RestSharp jest biblioteką przeznaczoną dla platformy .NET, która ułatwia przeprowadzanie żądań o architekturze REST do zewnętrznych usług. Dzięki wykorzystaniu tej biblioteki proces przeprowadzenia żądań http oraz deserializacji otrzymanych wyników jest znacznie prostszy. W aplikacji ServiceCMS konieczne było przechwycenie wyników w formacie JSON, co było możliwe dzięki wbudowanemu w RestSharpa serializatorowi JSON. Jest to biblioteka posiadająca bogatą dokumentację i wsparcie online. Powyższe cechy jak i mnogość funkcji z jednoczesnym zachowaniem prostoty obsługi zadecydowały o wybraniu tego narzędzia.

## Wzorce projektowe

Poniżej zostały opisane wzorce projektowe użyte w prezentowanym systemie. Każdy z nich jest rekomendowany przez firmę Microsoft w celu uzyskania skalowalnego i łatwego w utrzymaniu kodu.

## Unit of work

Unit of work jest wzorcem, który ułatwia korzystanie z repozytorium. Dzięki implementacji tego podejścia możliwy jest dostęp do wszystkich repozytoriów z poziomu jednej klasy. Dodatkowym atutem jest fakt, że dzięki wzorcowi Unit of work łatwo zaimplementować obsługę transakcji bazodanowych. Z racji konieczności wielokrotnego przeprowadzania operacji CRUD w bazie danych było to spore ułatwienie i zabezpieczenie przed niespójnością danych.

## Wstrzykiwanie zależności

Wstrzykiwanie zależności jest wzorcem architektonicznym, którego założenia polegają na usuwaniu bezpośrednich zależności pomiędzy komponentami na rzecz wykorzystania powiązań miękkich czyli połączeń za pomocą interfejsów a nie konkretnych implementacji. Z technicznego punktu widzenia wstrzykiwanie zależności jest przpadkiem szczególnym realizacji paradygmatu IoC który polega na odwróceniu odpowiedzialności za kontrolę wybranych operacji. Z praktycznego punktu widzenia zastosowanie tego wzorca umożliwiło ograniczenie koniecznej refaktoryzacji kodu po wprowadzeniu poprawek. Dzięki ograniczeniu twardych zależności nie była konieczna zmiana implementacji logiki po zmianie klasy zależnej. W takim wypadku wystarczyło wstrzyknięcie innej implementacji.

Wzorzec wstrzykiwania zależności silnie preferuje zewnętrzne tworzenie połączeń (zależności) pomiędzy komponentami oprogramowania (np. asocjacje w klasach), nad zlecaniem tworzenia zależności im samym. Jest to wzorzec, w którym odpowiedzialność za tworzenie obiektów i łączenie jest przeniesiona z obiektów do fabryki (np. kontenera IoC).

## Repozytorium

Repozytorium jest warstwą aplikacji, która izoluje obiektową warstwę dostępu do danych. Działa ono na poziomie jednej klasy modelu .Służy ono do uniezależnienia warstwy bazodanowej, która umożliwia dostęp do danych, od implementacji logiki biznesowej. W poniższej aplikacji wykorzystano repozytorium generyczne, które znacznie zmniejszło nakład pracy, potrzebny do implementacji dostępu do danych.

## MVC

Wzorzec MVC jest wzorcem architektonicznym, służącym do organizacji struktury systemów informatycznych. Został on zaprojektowany w 1979 przez programistów, pracujących nad językiem Smalltalk. Zakłada on podział aplikacji na trzy warstwy:

-warstwa Modelu (reprezentacja logiki biznesowej aplikacji).

-warstwa widoku (opisuje sposób prezentacji systemu użytkownikowi)

-warstwa kontrolera (przyjmuje dane z warstwy prezentacji i wywołuje metody zaimplementowane w modułach).

Warstwy te są ściśle zależne pomiędzy sobą. W trakcie implementacji funkcjonalności trzymano się zasad SOLID, co spowodowało marginalizację roli kontrolerów, na rzecz izolacji logii biznesowej w osobnych modułach. Wzorzec MVC zastosowano w celu organizacji procesu implementacji i spełnił on całkowicie oczekiwania i założenia, które sformułowano w fazie projektu.

## Visual Studio

Do wykonania projektu konieczne było korzystanie z środowiska programistycznego. Oczywistym wyborem dla języka C# i platformy .NET jest zintegrowane środowisko programistyczne Visual Studio. W poniższym projkecie korzystano z wersji Community, wydanej w 2013 roku. Wybór wersji był podyktowany postanowieniami licencyjnymi firmy Microsoft. Visual studio w znacznym stopniu przyczyniło się do ułatwienia pracy z kodem. Posiada on wbudowany debugger, który był wielokrotnie wykorzystywany do śledzenia występujących błędów. Usługa Nuget umożliwiała bezproblemowe instalowanie i korzystanie z dodatkowych bibliotek (RESTSharp, Ninject). Z racji charakterystyki wykonywanego projektu i systemu operacyjnego, na którym pracowano, niemożliwe było wykorzystanie jakiegokolwiek innego rozwiązania. Można jednak stwierdzić, że jest to narzędzie całkowicie spełniające swoje zadania i umożliwiające ekspresowy rozwój oprogramowania.

## Autofac

Autofac jest kontenerem wstrzykiwania zależności (IoC Container), opartym na licencji open-source. W realizowanym projekcie został wykorzystany do implementacji wzorca projektowego IoC. Biblioteka ta jest nastawiona na ograniczenie zbędnych funkcji, dzięki czemu jej obsługa nie sprawia problemów. Dzięki bogatej i wyczerpującej dokumentacji próg wejścia i nauki tego narzędzia był relatywnie niski. Rozważano również użycie biblioteki Ninject, która w procesie implementacji nie różniła by się znacznie od wykorzystanej. Na korzyść Autofac-a przemawiała jednak dużo bogatsza dokumentacja i zbiór gotowych rozwiązań, dostępnych w sieci.

# Projekt systemu wspólnie z Kamilem Ślusarczykiem

Zgodnie z założeniami, przyjętymi na etapie projektu, zdecydowano o wspólnej realizacji aplikacji. Service CMS został zrealizowany wspólnie przez Kamila Ślusarczyka oraz Artura Stelmacha. Poniższy rozdział jest wspólny dla obu prac – zarówno powyższej, jak i pracy Kamil Ślusarczyk pt. Projekt i implementacja systemu budowania i administrowania stronami internetowymi przez personel nietechniczny – część logiki biznesowej. Podczas implementacji ściśle trzymano się modułowego podziału systemu, który był podstawą do podziału pomiędzy autorów.

## 

## Wymagania biznesowe

ServiceCMS to aplikacja internetowa, dedykowana dla wszelakich firm usługowych. Aplikacja ma za zadanie wspomaganie w tworzeniu strony internetowej firmy. Najważniejszą funkcjonalnością aplikacji jest zautomatyzowanie rejestracji klientów na zdefiniowane wcześniej usługi. Ponadto aplikacja umożliwia zarządzanie treściami, które są reprezentowane przez strony i aktualności. Umożliwia również zarządzanie plikami i udostępnia możliwość przeglądania statystyk odwiedzin panelu klienckiego. Oprogramowanie może obsługiwać osoba nietechniczna z podstawową znajomością obsługi komputera.

## 

## Słownik pojęć

**Administrator** – osoba wyznaczona do zarządzania treścią, która jest wyświetlana w panelu klienckim.

**Klient** – osoba odwiedzająca panel kliencki

**Panel administracyjny** – strona internetowa umożliwiająca administratorowi zarządzanie treścią

**Panel kliencki** – strona internetowa która jest wynikiem ustawień ustalonych w panelu administracyjnym

**Aktualność** – prosty typ treści, wyświetlany na głównej stronie panelu klienckiego

**Strona** – typ treści który może zawierać wiele elementów takich jak ilustracje i załączniki w postaci plików. Do strony może prowadzić link w postaci przycisku z menu głównego.

**Dostawca usługi** – byt związany z typem usługi. Może to być osoba, stanowisko lub narzędzie wykonujące usługę.

**Typ usługi** – opis usługi wykonywanej w firmie. Każdy typ może posiadać wiele faz usługi.

**Popup** – wiadomość wyświetlana klientowi przy pierwszej wizycie w panelu klienckim.

**Odbiorca newslettera** – adres email na który może być wysłany newsletter

## 

## Wymagania funkcjonalne

Poniżej przedstawiono wymagania funkcjonalne zarówno panelu administratora jak i panelu klienckiego:

### Specyfikacja wymagań dla panelu administratora

* Dodawanie, usuwanie i edytowanie aktualności
* Dodawanie, usuwanie i edytowanie kategorii aktualności
* Dodawanie, usuwanie i edytowanie artykułów/stron
* Dodawanie, usuwanie i edytowanie usług zarejestrowanych przez klienta
* Dodawanie, usuwanie i edytowanie aktualności
* Dodawanie, usuwanie i edytowanie dostawców usług
* Dodawanie, usuwanie i edytowanie typów usług
* Możliwość zarządzania przyciskami menu
* Dodawanie, usuwanie i edytowanie odbiorców newslettera
* Możliwość kompozycji i wysyłania newslettera
* Dodawanie, usuwanie plików
* Dodawanie, usuwanie i edytowanie popupów
* Możliwość tylko jednego aktywnego popupu
* Możliwość przeglądania statystyk odwiedzin dotyczących panelu klienckiego
* Możliwość edytowania ustawień
* Możliwość zmiany języka na polski lub angielski

### Specyfikacja wymagań dla panelu klienckiego

* Wyświetlanie aktualności
* Wyświetlanie stron
* Nawigowanie za pomocą menu
* Rejestracja na usługę
* Wyświetlanie popupu przy pierwszej wizycie
* Zbieranie informacji o odwiedzinach

### Wymagania niefunkcjonalne

* Aplikacja powinna mieć spójny i przejrzysty interfejs użytkownika
* Panel administracyjny powinien być w języku polskim jak i angielskim
* Zarówno panel kliencki jak i panel administracyjny powinien być poprawnie wyświetlany na rozdzielczościach nie mniejszych niż 1600x900
* Aplikacja powinna działać na najnowszych wersjach przeglądarek Chrome, Firefox i Opera

## 

## Warstwy aplikacji

Aplikacja została podzielona na 4 warstwy : rdzenia, modułów, logiki biznesowej i prezentacji. Dzielenie aplikacji miało na celu uporządkowanie struktury aplikacji i zmniejszenie redundancji kodu. Poniżej przedstawiono każdą z warstw:

1. **Rdzeń** – warstwa odpowiadająca za połączenie i obsługę bazy danych, obsługę i logowanie wyjątków oraz zawierająca klasy wykorzystywane w całej aplikacji.
2. **Logika biznesowa** – warstwa zawierająca implementacje algorytmów logiki biznesowej takiej jak zapisywanie, edytowanie i usuwanie modeli wykorzystywanych w aplikacji.
3. **Moduły** – warstwa zawierająca wymienialne rozwiązania niezależne od reszty aplikacji np. obsługa zapisu i usuwania plików, łączenie z zewnętrznym serwisem lub obsługa kryptografii haseł.
4. **Prezentacja** – warstwa odpowiedzialna za interakcje z użytkownikiem aplikacji, zawiera wszystkie widoki wyświetlane użytkownikowi oraz akcje obsługujące działania użytkownik

## 

## Projekt bazy danych

Aplikacja wykorzystuje relacyjną bazę danych SQL wygenerowaną przy pomocy Entity Framework zgodnie z metodologią CodeFirst. Każda z tabel bazy posiada klucz główny który zapewnia możliwość łączenia ze sobą dwóch lub więcej tabel. Baza danych została znormalizowana do 3 postaci normalnej, aby zapobiec rozrastaniu i nadmiarowości danych. W celu przyspieszenia przeszukiwania tabel zostały nałożone indeksy na każdy klucz główny.

Poniżej przedstawiono najważniejsze tabele i ich relacje:

* **ServiceTypes –** tabela odwzorowująca typ usługi. Oprócz klucza głównego „Id” posiada kolumnę „Name” przechowującą nazwę danego typu usługi. Jest połączona relacją jeden do wielu z tabelą **ServicePhases** (typ usługi ma wiele faz).
* **ServicePhases –** tabela przedstawiająca fazy usług. Posiada kolumnę „Order” które przechowuje informację o kolejności wykonywania. Tabela przechowuje również informacje o czasie trwania danej fazy (kolumna „DurationInMinutes”) oraz czasie opóźnienia (kolumna „DelayInMinutes”). Posiada również klucz obcy tabeli ServiceType, umożliwiający relację typu „jeden do wielu”.
* **ServiceProviders –** przechowuje informacje o dostawcach usług. Jest połączona z tabelą **ServiceTypes** relacją wiele do wielu (dostawca usługi może wykonywać wiele usług a usługi mogą być wykonywane przez wielu dostawców) przy pomocy dodatkowej tabeli **ServiceProviderServiceTypes.**
* **RegistratedServices –** odwzorowuje usługi zarejestrowane przez klientów. Połączona jest z tabelą **ServiceTypes** relacją jeden do wielu (typ usługi może być w wielu zarejestrowanych usługach) oraz z tabelą **ServiceProviders** również relacją jeden do wielu. Oprócz tego przechowuje podstawowe informacje o kliencie który zarejestrował się na usługę.
* **Pages –** przedstawia strony edytowane przez administratora. Przy pomocy tabeli **PageFiles** tworzy relację wiele do wielu z tabelą **Files.** Kolumna „RestorePageId” odpowiada za połączenie tej tabeli z samą sobą. Dzięki temu baza danych umożliwia przechowywania historii modyfikacji danych stron.
* **MenuButtons –** tabela przedstawiająca przyciski menu. Łączy się sama ze sobą dzięki czemu uzyskujemy strukturę drzewiastą która doskonale odwzorowuje menu. Oprócz tego posiada połączenie jeden do wielu z tabelą **Pages** co pozwala na nawigowanie pomiędzy stronami dzięki stworzonemu menu.
* **News –** tabela podobna do tabeli **Pages**. Posiada połączenie wewnętrzne, co umożliwia przechowywanie historii. Oprócz tego za pośrednictwem tabeli **NewsNewsCategories** realizowana jest relacja wiele do wielu z tabelą **NewsCategories** która pozwala na filtrowanie aktualności za pomocą kategorii.
* **Settings** – tabela przechowująca informacje o ustawieniach aplikacji. Posiada pole „Value” które może przechowywać dowolną wartość możliwą do zapisania oraz pole InputType definiujące typ kontrolki który należy wyświetlić do edytowania danego ustawienia.

Na rys. 4.1 został przedstawiony kompletny schemat bazy danych aplikacji:

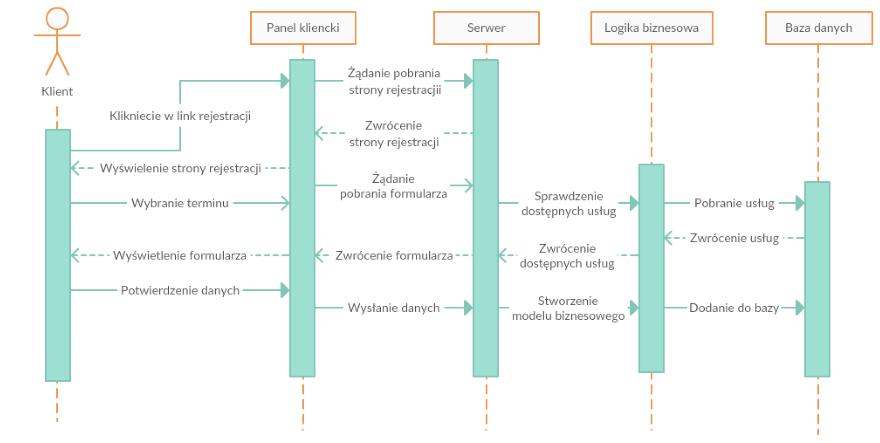
Rys 4.1. Schemat bazy danych aplikacji

## 

## Diagramy sekwencji

Poniżej przedstawiono diagramy sekwencji najważniejszych akcji podejmowanych przez użytkownika.

### Rejestracja na usługę

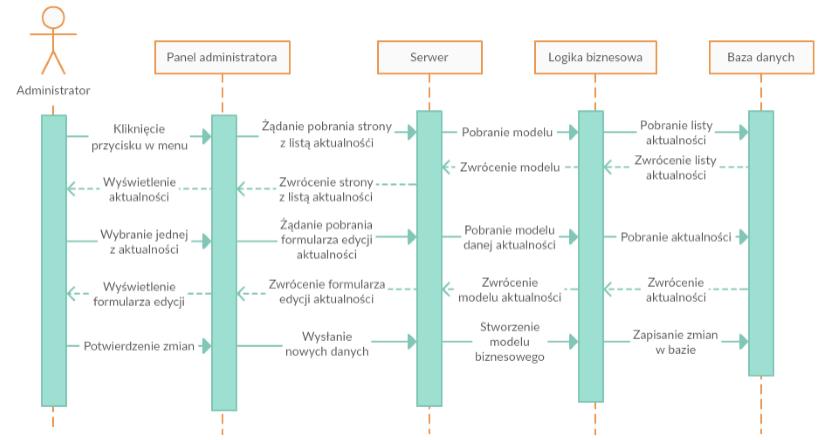


Rys 4.2. Diagram sekwencji rejestracji na usługę

Diagram przedstawia podstawową funkcjonalność aplikacji, jaką jest rejestracja klienta na

usługę. Użytkownik poprzez kliknięcie w odpowiedni link sygnalizuję chęć rejestracji. Wyświetlana jest strona z kalendarzem i dostępnymi terminami usług. Użytkownik wybiera odpowiadający mu termin. Następnie wysyłane jest żądanie pobrania formularza z dostępnymi w tym terminie usługami. Kolejnym krokiem jest potwierdzenie przez użytkownika danych i zarejestrowanie usługi w bazie danych.

### Edycja aktualności

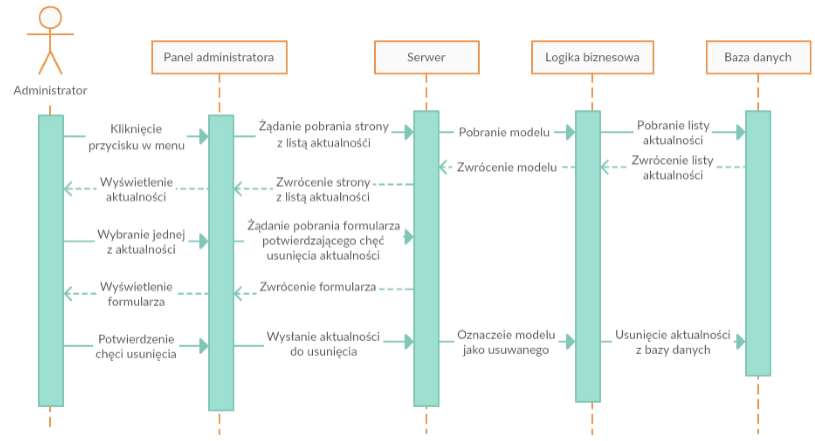


Rys 4.3. Diagram sekwencji edycji aktualności

Ze względu na podobieństwo wszystkich funkcjonalności edycji i dodawania, aktualności, stron itd. ograniczono się do przedstawienia jednego przykładu w postaci edycji aktualności.

Użytkownik poprzez menu wybiera typ danych który chce edytować. Wyświetlana jest strona z listą wybranych wcześniej encji. Użytkownik wybiera jedną z encji którą chce edytować i wyświetlany jest mu formularz edycji z danymi. Następnie użytkownik potwierdza dokonanie zmian co skutkuje zapisaniem danych w bazie.

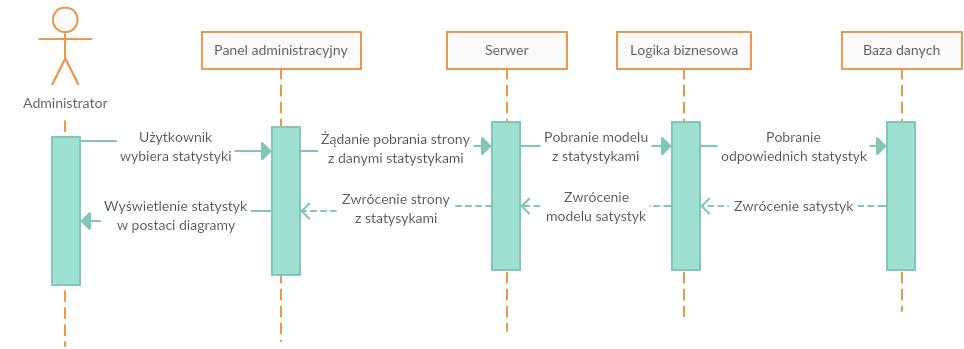
### Usuwanie aktualności



Rys 4.4. Diagram sekwencji usuwania aktualności

Usuwanie każdego typu danych jest realizowane w podobny sposób. Tak jak w przypadku edycji czy usuwania użytkownik wybiera typ danych do usunięcia. Następnie wyświetlana jest strona z listą encji. Użytkownik wybiera encję którą chce usunąć, wyświetlany jest mu formularz który pozwala na potwierdzenie chęci usunięcia. Po potwierdzeniu wysyłane jest żądanie usunięcie aktualności, która jest usuwana z bazy danych.

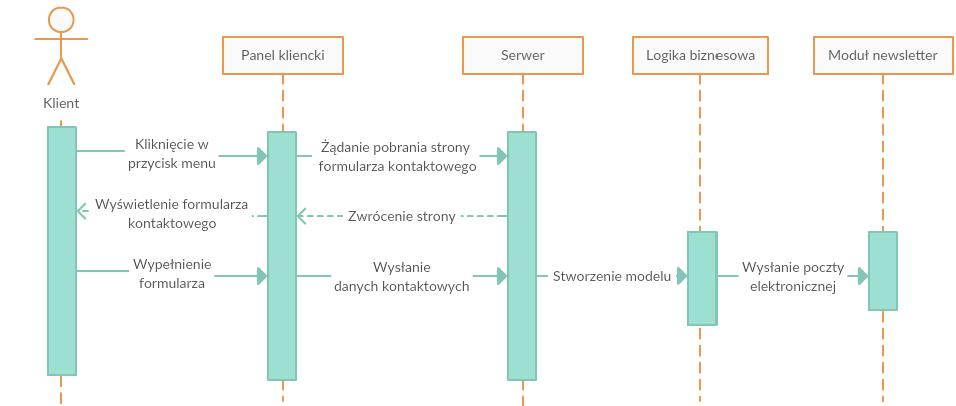
### Wyświetlenie statystyk



Rys 4.5. Diagram sekwencji wyświetlania statystyk

Administrator sygnalizuje chęć przeglądania statystyk odwiedzin, poprzez wybranie w menu odpowiedniego przycisku. Wysyłane jest żądanie pobrania strony z statystykami, następnie serwer pobiera odpowiednie dane z bazy danych i wyświetlana jest strona z danymi przedstawionymi w postaci odpowiedniego diagramu.

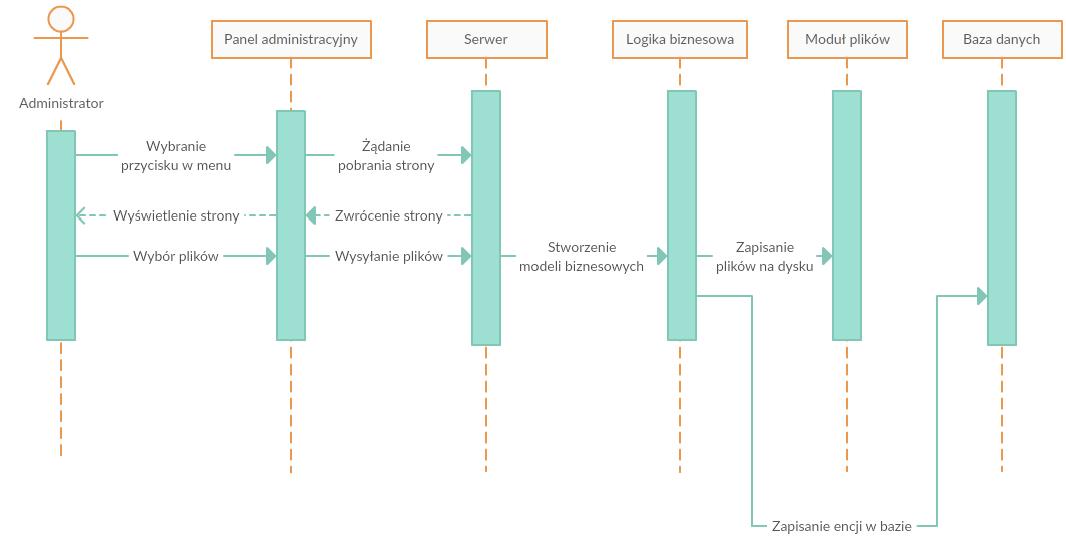
### Skorzystanie z formularza kontaktowego



Rys 4.6. Diagram sekwencji korzystania z formularza kontaktowego

Klient wybiera w menu odpowiedni przycisk. Wyświetlana jest mu strona z formularzem kontaktowym. Następnie klient wypełnia formularz danymi i potwierdza wysyłanie wiadomości. System przekazuje dane do modułu newslettera który tworzy i wysyła wiadomość poprzez serwer SMTP.

### Przekazanie plików na serwer



Rys 4.7. Diagram sekwencji przekazywania plików na serwer

Administrator wybiera odpowiedni przycisk w menu. Następnie wybiera pliki które chce przekazać na serwer i potwierdza ten wybór. Pliki są zapisywane na dysku oraz tworzone są encje opisujące te pliki do zapisania w bazie danych .

# Prezentacja systemu

Poniżej przedstawiono najważniejsze rysunki prezentujące ServiceCMS w działaniu.

## Ekran logowania

Rys 5.1 Przedstawia ekran logowania do panelu administracyjnego.



Rys 5.1. Ekran logowania do panelu administracyjnego

Logowanie jest realizowany w standardowy sposób, przyjęty w aplikacjach internetowych. Udostępniono możliwość zapamiętania nazwy użytkownika i hasła.

## Pulpit administracyjny

Poniżej zaprezentowano ekran główny aplikacji administracyjnej. Jest to ekran, wyświetlany po zalogowaniu się do systemu przez administratora. W trakcie wykonywania zwrócono szczególną uwagę na to, aby ekran ten był przyjazny dla użytkowika i estetyczny. Pomogło w tym użycie technologii Bootstrap.



Rys 5.2. Ekran główny panelu administracyjnego

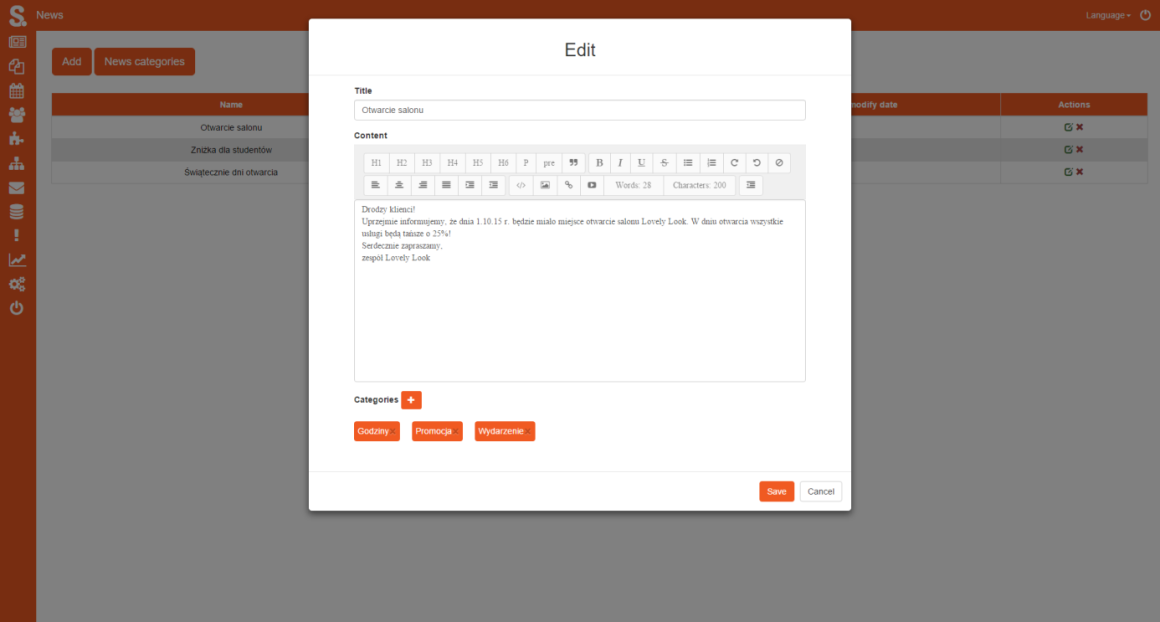
Poniżej zaprezentowane menu, które jest widoczne po wywołaniu akcji *hover*. Menu jest umieszczone po lewej stronie pulpitu administracyjnego. Ikony mają za zadanie intuicyjną prezentację funkcjonalności, do których prowadzą.



Rys 5.3. Lewe menu aplikacji

## Moduł aktualności

Poniżej zaprezentowano widok edycji encji News (ang. Aktualność). W wybranym oknie możliwa jest zmiana tytułu, zawartości oraz dodanie kategorii aktualności (widoczne u dołu). Okno edycji posiada panel edycyjny, ułatwiający pracę z tekstem.



Rys 5.4. Ekran edycji aktualności

## Moduł plików

Rys 5.5 prezentuje widok zarządzania plikami. Aplikacja ServiceCMS umożliwia umieszczanie na serwerze plików o dowolnych rozszerzeniach. Należy zaznaczyć, że ich wielkość jest domyślnie ograniczona do 50 mB i można ją zwiększyć do maksymalnie 2 GB. Poniższa ilustracja przedstawia zakończony proces umieszczania plików na serwerze.



Rys 5.5. Ekran zarządzania plików

## Moduł stron

Poniżej zaprezentowano okno edycji strony. Jest ono podobne do okna edycji aktualności (ang. News) - tu również umożliwiono edycję nazwy oraz zawartości za pomocą narzędzi edycyjnych. Różnicą jest możliwość załączania plików do strony. Muszą one być wcześniej umieszczone na serwerze, za pomocą modułu Files.



Rys 5.6. Ekran edycji strony

## Moduł przycisków menu

Poniżej zaprezentowano widok edycji encji MenuButton. Odpowiada ona za przyciski menu po stronie klienckiej. Ułożenie w hierarchii jest odwzorowano jako struktura drzewiasta menu. Stopnie zagnieżdżenia można zmieniać zgodnie z koncepcją Drag n Drop (ang. Przeciągnij i upuść). Elementy można poddać tradycyjnym operacjom CRUD – można je dodawać, edytować i usuwać. Możliwe jest również zmiana kolejności i stopnia zagnieżdzenia. Menu jest połączone z wybranymi stronami w relacji jeden do wielu.



Rys 5.7. Ekran edycji strony

Poniżej zaprezentowano strukturę drzewiastą, odwzorowaną w aplikacji klienckiej.



Rys 5.8. Ekran edycji strony

## Moduł newslettera

Poniżej przedstawiono ekran zarządzania newsletterem. Umieszczono ilustracje, które przedstawiają zarządzanie listą odbiorców i widok edycji komponowanej wiadomości, w którym oprócz standardowych akcji możliwe jest również przypisanie do listy adresatów wybranych, bądź wszystkich odbiorców z bazy danych.



Rys 5.9. Ekran zarządzania adresatami newsletterów



Rys 5.10. Ekran kompozycji newslettera

## Moduł ustawień globalnymi

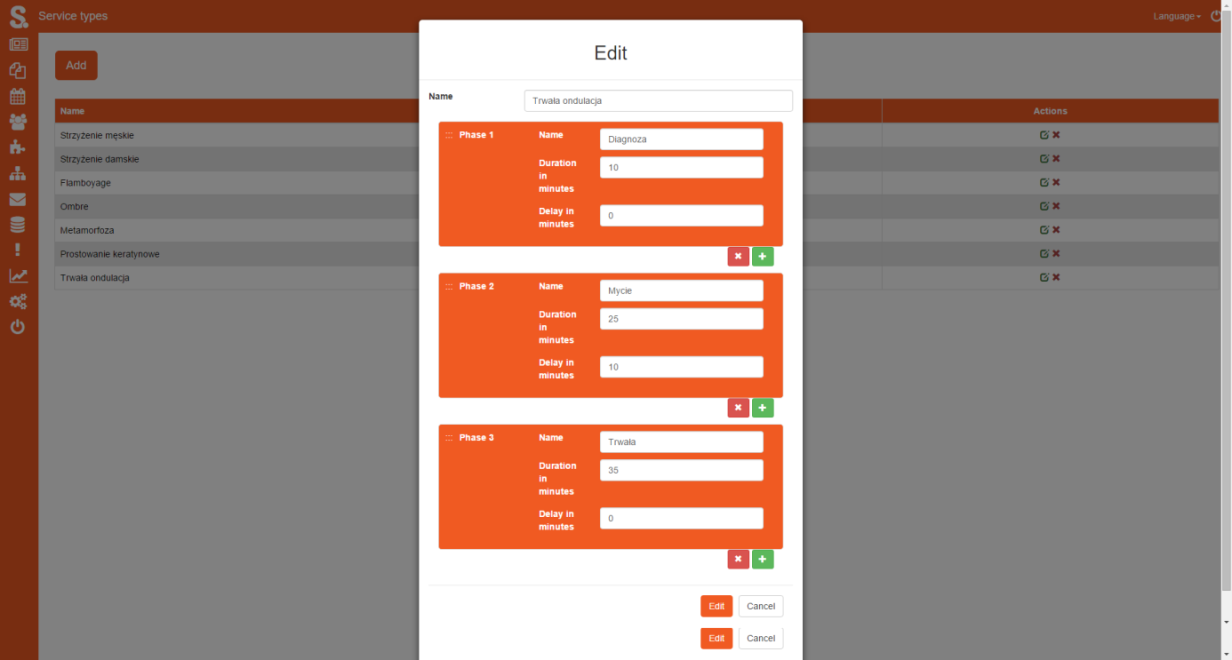
Poniżej przedstawiono widok modułu Settings (ang. Ustawienia). Moduł ten umożliwia zmianę ustawień globalnych dla aplikacji ServiceCMS. Możliwa jest zmiana nazwy firmy, dane autentykacyjne konta e-mail oraz ustawienia ilości aktualności na pojedynczej stronie. Dostępna jest też edycja stanu funkcjonalności formularza kontaktowego, kalendarza usług oraz pop-up – w tym przypadku można wybrać stan na włączony, bądź wyłączony.



Rys 5.11. Ekran sterowania ustawieniami

## Moduł usług

Na Rys 5.12. przedstawiano moduł, służący do zarządzania dostępnymi usługami. Każda ze stworzonych usług może zawierać wiele faz. Z kolei każda faza posiada nazwę, czas trwania oraz opóxnienie. Dzięki pomysłowej implementacji dostępnych usług, możliwe jest zarezerwowanie usługi, podczas gdy inny klient jest w trakcie fazy z opóźnieniem. Fazy, podobnie jak przyciski menu, są zrealizowane zgodnie z podejściem Drag n’ Drop.



Rys 5.12. Ekran edytowania usług

## Moduł dostarczyciela usług

Poniżej przedstawiono widok edycyjny dla encji Service Provider. W panelu tym umożliwiono zmianę nazwy oraz zakres usług, możliwy do wykonania przez określonego dostarczyciela usług. Usługi, które są dodawane do dostarczycieli definiuje się w module Service Types. Jest to odwzorowane w aplikacji klienckiej jako zakres usług, które może wybrać klient – do wybranego dostarczyciela usług (np. pracownik lub stanowisko pracownicze) można się zapisać na wybrany zbiór usług.



Rys 5.13. Ekran edytowania dostarczyciela usług

## Moduł Pop-up

Poniżej przedstawiono moduł Pop-up. Umożliwia on zamieszczenie jednorazowo wyświetlanej informacji w aplikacji klienckiej. Jego edycji jest zbieżna z edycją aktualności i stron. Widok edycji umożliwia edycję tytułu oraz zawartości. Moduł Pop-up umożliwia również uaktywnienie encji, która będzie aktywna, za pomocą ekskluzywnego checkbox-a.



Rys 5.14. Ekran edytowania dostarczyciela usług

## Moduł zarejestrowanych usług

Poniżej przedstawiono moduł Registrated Services. W module tym widoczna jest lista usług, które zostały zarejestrowane przez klientów. Przedstawione są szczegóły usługi – czas utworzenia rejestracji, typ wybranej usługi, wybrany dostarczyciel oraz dane klienta, który dokonał rezerwacji. Możliwy jest także podgląd danych klienta, po wywołaniu akcji *hover* na ikonę klienta. Moduł umożliwia również odwołanie usługi.



Rys 5.15. Ekran zarejestrowanych usług

Ważna funkcjonalnością w opisywanym module jest potwierdzenie, które jest wysyłane do klienta po wywołaniu akcji rejestracji na usługę. Jest to spersonalizowana wiadomość e-mail, wysyłana w fomie widocznej poniżej.



Rys 5.16. Ekran przykładowego maila

## Moduł Statistics

Poniżej przedstawiono moduł Statistics. Przedstawia on statystyki odwiedzin łącznie z rodzajem aktywności, wykonanych przez klientów. Jest to moduł istotny, ponieważ przedstawia konwersję odwiedzających stronę na klientów, dokonujących rejestrację usług. Dzięki temu administrator może dopasowywać zakres i formę przedstawianej oferty, aby w jak największym stopniu odpowiedziała ona na wymagania klientów. Moduł statystyk pozwala określić ilość odwiedzin, odwiedzone strony i kraj pochodzenia żądania HTTP w zależności od wybranych ram czasowych (wybrane dni, miesiące, lata).

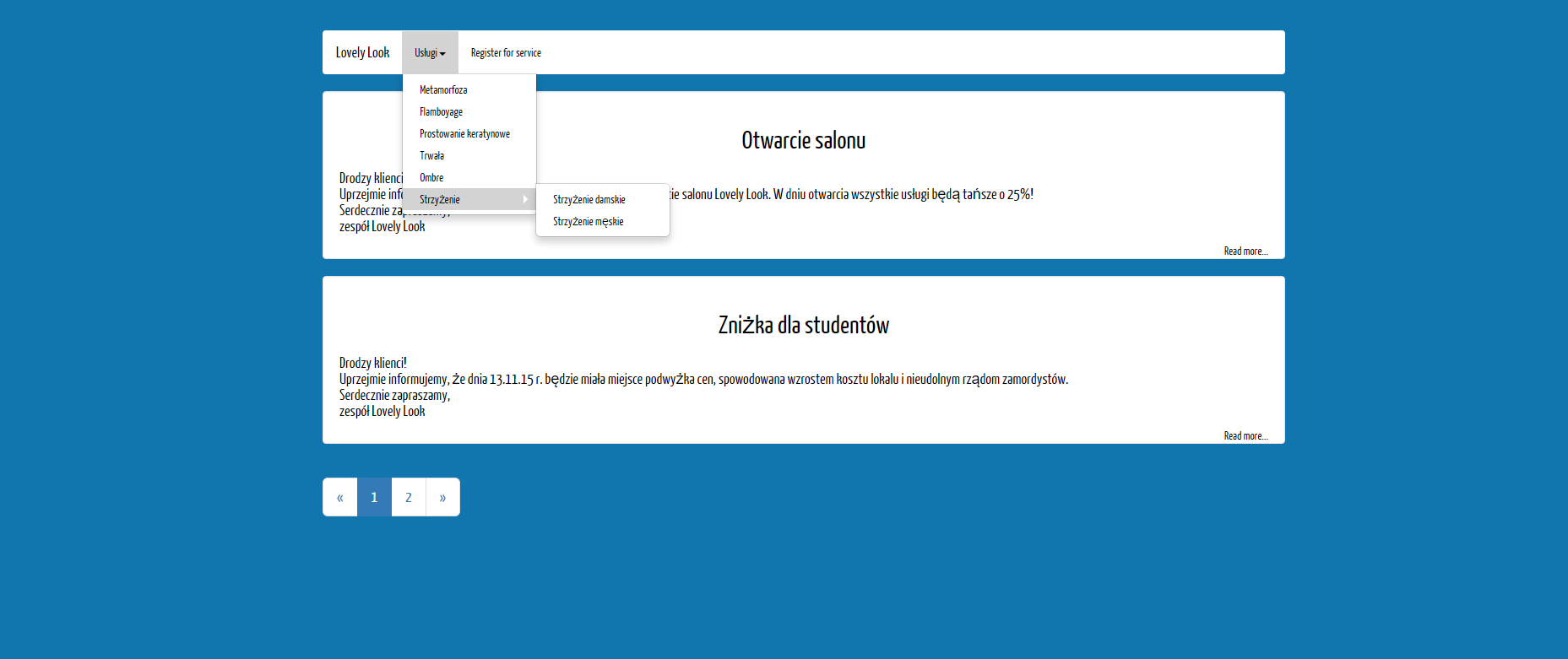


Rys 5.17. Ekran statystyk w postaci diagramu kołowego



Rys 5.18. Ekran statystyk w postaci diagramu kolumnowego

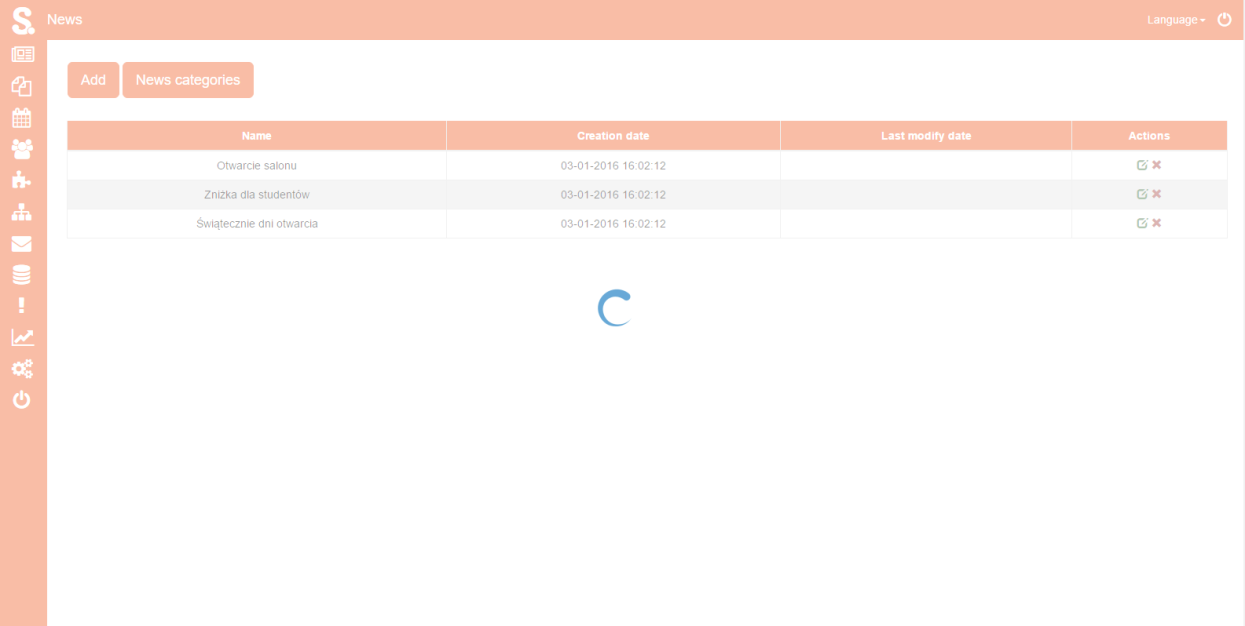
## Ekran aplikacji klienckiej



Rys 5.19. Ekran statystyk w postaci diagramu kolumnowego

Powyżej przedstawiono główny ekran aplikacji klienckiej. Ukazane jest rozwijalne menu, które zostało zdefiniowane za pomocą modułu MenuButtons. Struktura drzewiasta jest wynikiem operacji, przeprowadzonych przez administratora w panelu administracyjnym. Przycisku menu prowadzą do stron, które zostały do nich przypisane przez administratora. Przedstawione jest również stronicowanie aktualności, które jest zgodne z ustawieniami globalnymi zdefiniowanymi w modul Settings.

## Ekran ładowania



Rys 5.20. Ekran ładowania danych

Powyżej zaprezentowano ekran ładowania, który zostaje przedstawiony administratorowi w momencie, gdy ładowanie danych z bazy w zauważalny sposób wpływa na wykonywanie czynności administracyjnych. Proces pobrania danych wielokrotnie wymaga około 0.5-1 sekundy na wykonanie. Powyższy ekran jest wyświetlany aby upewnić administratora, że aplikacja nie napotkała błędów i poprawnie przeprowadza wymagane operacje.

# Budowa systemu

Poniżej zostaną zaprezentowane najważniejsze fragmenty kodu źródłowego. Rozdział został podzielony na 2 części. Pierwsza z nich opisuje fragmenty kodu wykorzystywane w warstwie logiki biznesowej. Druga zaś opisuje fragmenty kodu zaimplementowanego w prezentacji.

## Funkcjonalności w warstwie logiki biznesowej

## Menadżer sesji

Rys 6.1. Klasa SessionManager

Klasa SessionManager obsługuję podstawowe operacje, wykonywane na sesji. Pozwala na wstawienie do sesji obiektu każdego typu. Generyczna metoda Get<T> pozwala na wyciągnięcie i jednoczesne rzutowanie obiektu na dany typ dzięki generycznemu typowi T.

## Obsługa ustawień regionalnych

Rys 6.2. Klasa CultureHelper

CultureHelper odpowiada za bezpieczną obsługę ustawień regionalnych. Dzięki zaimplementowanym metodom możliwe jest bezpieczne pobieranie informacji o aktualnych i zaimplementowanych ustawieniach regionalnych.

## Generyczne repozytorium



Rys 6.3. Generyczne repozytorium

Generyczne repozytorium odpowiada za podstawowe operacje na encjach. Dzięki generycznemu typowi TEntity działa na każdym typie obiektów. Wszystkie metody działają na generycznym obiekcie typu DbSet dostarczanym przez EF. Rozwiązanie to jest wysoce elastyczne i skalowalne – opiera się ono częściowo na mechanizmie refleksji.

## FluentApi

Rys 6.4. FluentApi

Przy podejściu CodeFirst problemem jest zarządzanie relacjami pomiędzy obiektami. W aplikacji zostało to rozwiązane za pomocą FluentApi. Każda z relacji została w odpowiedni sposób opisana za pomocą metod obiektu DbModeBuilder. Podejście to jest alternatywą dla tradycyjnego, zarządzającego zależnością pomiędzy obiektami za pomocą Data Annotations. Powyższe rozwiązanie jest jednak bardziej elastyczne. Dodatkowym atutem jest też fakt, że wszystkie relacje są dostępne w jednym miejscu.

## Unit of Work

 Rys 6.5. Klasa Unit of Work

Implementacja wzorca projektowego UnitOfWork. Ma ona zapewnić spójność danych, odwzorowując transakcję bazodanową. Dzięki klasie UnitOfWork możliwe było zebranie wszystkich repozytoriów w jednym pliku.

## Logger

Rys 6.6. Klasa Logger

Klasa Logger odpowiada za bezpieczne logowanie przechwyconych wyjątków. Klasa zapisuje wyjątki w ścieżce podanej w stałej LOG\_PATH. Umożliwia to śledzenie błędów przez programistów bez narażania użytkowników na problemy, związane z działaniem aplikacji. Błędy te, są zapisem wygenerowanym przez debugger programu Visual Studio.

## Wstawki

Poniżej została przedstawiona funkcjonalność wstawek. Funkcjonalność ta polega na dodawaniu do treści strony specjalnych znaczników które przy odczytywaniu z bazy danych są zamieniane na konkretny kod html.

### InsetParser

 Rys 6.7. Klasa InsetParser

Klasa Parser jest klasą spinającą funkcjonalność wstawek w spójną całość. Posiada metodę, która za pomocą wyrażenia regularnego zamienia wszystkie tagi wstawek na odpowiadające im kody HTML.

### InsetRecognizer

Rys 6.8. Klasa InsetRecognizer

Klasa InsetRecognizer odpowiada za walidację znalezionych wstawek. Dzięki niej możliwa jest walidacja, czy dana wstawka ma poprawną składnię, czy posiada wszystkie wymagane argumenty i czy te argumenty mają poprawne wartości.

### ArgumentValidator

Rys 6.9. Klasa ArgumentValidator

Klasa ArgumentValidator odpowiada za sprawdzanie, czy dany argument posiada prawidłową wartość. Dla każdego argumentu znany jest typ danych na który podejmowana jest próba konwersji wartości. Jeśli proces ten się udał, znaczy to, że argument jest prawidłowy.

### LocalLinkParser

Rys 6.10. Klasa LocalLinkParser

Każdy typ wstawki ma zaimplementowaną klasę typu Parser. Ze względu na podobieństwo implementacji tego typu klas przedstawiono przykład w postaci klasy LocalLinkParser mającą na celu wygenerowanie linku do lokalnej strony.

## Timeblocks

Rys 6.11. Metoda GetRegistratedServicesTimeBlocks

Metoda GetRegistratedTimeBlocks tworzy z zarejestrowanych usług bloki czasowe, o typie określonym jako pary początek-koniec. Dla każdej fazy obliczany jest czas trwania, który jest sumą czasu początku usługi, czasu poprzednich faz oraz opóźnienia poprzednich faz.

Rys 6.12. Metoda GetServiceTypesTimeBlocks

Metoda GetServiceTypesTimeBlocks tworzy z typów usług podobne bloki czasu, jak opisane wyżej. Różnicą jest to, że punktem wyjścia jest data, wybrana przez użytkownika. W efekcie takiego podejścia dla każdej fazy każdego typu usługi tworzony jest blok czasowy, gdzie do daty wybranej przez użytkownika dodawany jest czas poprzednich faz oraz opóźnień tych faz.

## Settings



Rys 6.13. Metoda GetBoxedType

Metoda GetPoxedType rzutuje wartość ustawienia odczytanego z bazy danych w postaci ciągu znaków na typ, pasujący do pola InputType. W celu zabezpieczenia rozwiązania w przypadku niepowodzenia rzutowania wartości, wartość ta pozostaje ciągiem znaków.



Rys 6.14. Metoda GetAll

Metoda zwracająca ustawienie aplikacji, wraz z nazwą ustawienia, wartością i typem kontrolki użytej do edycji. Wartości ustawień są rzutowane na odpowiedni typ dzięki użyciu wyżej wymienionej metody i opakowywane w podstawowy typ w platformie .NET – object.

## Dependency injection

Klasa DIRegister odpowiada za organizację procesu wstrzykiwania zależności. Dzięki zebraniu wszystkich wywołań biblioteki Autofac w jednym miejscu otrzymano skalowalny i elastyczny proces.

 Rys 6.15. Metoda RegisterLogic

Metoda RegisterLogic jest odpowiedzialna za rejestrację komponentów aplikacji w kontenerze IOC. Metoda rejestruje konkretne implementacja do wstrzyknięcie, gdy żądany jest dany interfejs.

## Cryptography

Moduł kryptografi odpowiedzialny jest za kodowanie haseł użytkowników. Dobrym podejściem w projektowaniu takich aplikacji jest przechowywanie haseł w postaci zaszyfrowanej.

 Rys 6.16. Metoda GeneratePasswordHash

Klasa PasswordManager posiada 2 metody, które są odpowiedzialne za obsługę zakodowanych haseł. Metoda GeneratePasswordHash generuje na podstawie podanego hasła jego zakodowaną postać. Do samego hasła dodawana jest również losowa „sól” w postaci ciągów znaków, który w znaczny sposób zwiększa bezpieczeństwo. Metoda IsPasswordMatch porównuje czy podane hasło zgadza się z zakodowanym hasłem. W systemie po zakodowaniu hasła nie ma możliwości odczytania go w zwykłej postaci. Podejście to jest zgodne z podejściem, stosowanym w nowoczesnych aplikacjach.

## FileManager

 Rys 6.16. Klasa FileManager

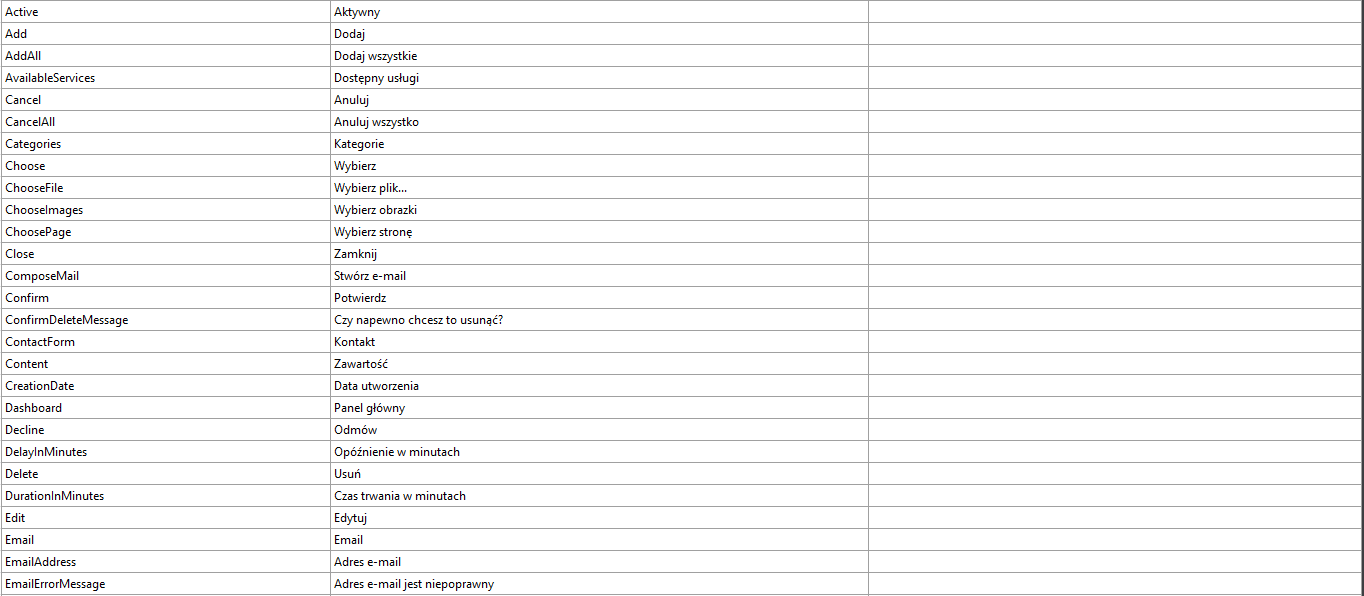
Klasa FileManager posiada zbiór metod, które pozwalają w bezpieczny i wygodny sposób działać na systemie plików. Dzięki przeciążeniu metody SaveFile możliwa jest łatwa obsługa operacji zapisu plików na serwerze. Pozostałe metody są wykorzystywane do usuwania plików z serwera, określenia, czy plik istnieje, oraz pobrania go przez użytkownika aplikacji klienckiej.

## BaseController



Rys 6.17. BaseController

Clasa BaseController jest dziedziczona przez każdy kontroler, zarówno w aplikacji administracyjnej, jak i klienckiej. Zawiera ona logikę, która odpowiada za ustawienie języka panelu administracyjnego. Ustawienie jest pobierane z ciasteczka (ang. Cookie) i przypisywane do wątku, w którym obsługiwana jest reszta żądań. Ustawienia te są zmieniane poprzez kontroler LanguageController.



Rys 6.18. Plik Resources

Powyżej przedstawiono plik Resources.pl-PL.resx, dzięki któremu jest możliwa implementacja dwujęzyczności. Pliki ten został stworzony w dwóch wersjach polskiej (Resources.pl-PL.resx) i angielskiej (Resources.resx). Dzięki wbudowanemu mechanizmowi w momencie odczytywania wartości z tych plików platforma .NET odczytuje ustawienia lokalne z bieżącego wątku i na ich podstawie ustala z którego pliku ma korzystać.

## Moduł File

Rys 6.19. Metoda UploadWithInsert

Oprócz podstawowych operacji CRUD klasa FileSerivce posiadę metodę UploadWithInsert. Odpowiada ona za dodanie encji do bazy z jednoczesnym zapisem pliku na dysku serwera. Pliki umieszczone na serwerze można identyfikować po nazwie oraz ścieżce.

Rys 6.20. Metoda GetFilePath

Metoda GetFilePath odpowiada za nadanie unikalnej nazwy plikom – dzięki wywołaniu metody NewGuid() plik otrzymuje globalnie unikalny identyfikator. Daje to pewność, że żaden plik nie zostanie nadpisany.

## Funkcjonalności frontowe

Wszystkie poniższe funkcjonalności warstwy prezentacji zostały zaimplementowane za pomocą frameworka AngularJS (patrz 3.4). Każda z funkcjonalności została podzielona na 4 warstwy:

1. **Kontroler –** warstwa odpowiedzialna za logikę działania aplikacji oraz za komunikowanie się z warstwą serwisu. Instancje poszczególnych serwisów są wstrzykiwane za pomocą wbudowanego mechanizmu Dependency Injection.
2. **Serwis –** warstwa odpowiedzialna za komunikowanie się z serwerem ASP .NET MVC. Każdy z serwisów również wykorzystuje wbudowany mechanizm DI w celu korzystanie z innych serwisów lub wbudowanych w framework gotowych rozwiązań.
3. **Widok –** warstwa odpowiedzialna za prawidłowe wyświetlanie informacji użytkownikowi oraz interakcję z tymże użytkownikiem. Z założenia widok nie ma bezpośredniego połączenia z serwisem. Odpowiednia komunikacja jest realizowana poprzez kontroler.
4. **Dyrektywy –** Narzędzia które z założenia mają być samodzielnymi modułami. Mogą jedynie komunikować się z warstwą serwisu w celu pobierania lub wysyłania odpowiednich danych.

W celu lepszej organizacji i skalowalności kodu każda z warstw została umieszczona w osobnym folderze, a każdy obiekt swojego typu w osobnym pliku z rozszerzeniem .js.

## Mechanizm paczkowania

W celu poprawnego działania każdej aplikacji napisanej w JavaScript należy do strony HTML dodać link wskazujący na plik z rozszerzeniem .js. Platforma .NET w łatwy i przyjemny sposób ułatwia to zadanie. Dzięki klasie BundleConfig jesteśmy w stanie paczkować wiele plików js jaki css. Zwiększa to wydajność ładowania strony jak i minimalizuje wielkość plików ładowanych na stronie. Na poniższej ilustracji przedstawiono kilka przykładowych paczek plików js

 Rys 6.21. Przykładowe paczki plików js

Paczki dołączane są do widoku poprzez użycie metody Scripts.Render Co widać na poniższej ilustracji.

Rys 6.22. Przykład użycia paczki w widoku

## Moduł aplikacji

 Rys 6.23. Główny moduł aplikacji AngularJS

Cała warstwa prezentacji została zawarta w specjalnym obiekcie dostarczanym w frameworku AngularJS o nazwie app. Obiekt odpowiada za działanie całej aplikacji, Abu korzystać z kontrolerów AngularJS należy je umieścić wewnątrz elementu HTML który jest opatrzony specjalną dyrektywą ng-app co pokazano na poniższej ilustracji.

 Rys 6.24. Użycie głównego obiektu w widoku

## Serwisy

Każdy serwis w aplikacji komunikujący się z serwerem ASP .NET MVC jest zaimplementowany w podobny sposób. Jest to obiekt zawierający zbiór metod które wysyłają i odbierają informacje z serwera. Każda z takich metod korzysta z wbudowanego w framework serwisu $http który odpowiedzialny jest za samą komunikację z serwerem. Metody te korzystają również z obiektu $q który wspomaga asynchroniczne operacje.

Rys 6.25. Przykładowy serwis

## Walidacja formularzy

Walidacja formularzy odbywa się za pomocą wbudowanego mechanizmu AngularJS. W elemencie HTML form umieszczane są kontrolki z nadaną nazwą. W ten sposób w obiekcie formu udostępniana jest lista obiektów z błedami walidacyjnymi. Jeśli w danym obiekcie pojawi się jakikolwiek błąd wyświetlana jest dla niego wiadomoś. Gdy jakikolwiek z elementów jest nie poprawny to cały formularz również staje się nie poprawny i za pomocą dyrektywy ng-disabled blokowany jest przycisk do potwierdzenia danych. Mechanizm ten przedstawia poniższa ilustracja:

Rys 6.26. Walidacja formularzy w widoku

## Podstawowe operacje CRUD

Każda z encji dostępnych w aplikacji ServiceCMS posiada swój kontroler który udostępnia podstawowe metody obsługujące operacje typu CRUD. Ze względu na duże podobieństwo tych operacji wybrano jeden kontroler NewsController który posłuży za przykład. Akcje są wywoływane za pomocą przycisków i specjalnej dyrektywy ng-click.

 Rys 6.27. Przyciski wywołujące akcje CRUD

* 1. Odświeżanie

Podstawową funkcjonalnością związaną z każdą encją jest wyświetlanie tychże encji.

W każdym kontrolerze odpowiedzialna jest za to metoda refresh() która za pomocą serwisu pobiera aktualne encje.

Rys 6.28. Przykładowe odświeżanie encji

Metoda NewsService.getNewsestNewses() (jak każda inna metoda serwisu) zwraca tak zwany obiekt – obietnicę dzięki opisanemu wcześniej obiektowi $q. Obiekt ten posiada metodę then() która jako argumenty przyjmuje kolejno : metodę kiedy żądanie się powiedzie i metodę kiedy żądanie się nie powiedzie. Metody tego typu przypisują obiekt zwrócony z serwisu do lokalnego obiektu dostępnego w ramach aktualnego kontrolera, a następnie dzięki specjalnemu, wbudowanemu mechanizmowi wiązania są wyświetlane w widoku za pomocą dyrektyw ng-repeat i ng-bind.

 Rys 6.29. Przykładowe wyświetlanie encji w widoku

* 1. Dodawanie

Operacja dodawania encji jest obsługiwana za pomocą metody add(). Metoda ta korzysta z obiektu $modal w celu otwarcia nowego okna i przekazuje kontrole do podanego kontrolera oraz za pomocą metody then() dostępnej w obiekcie $modal.result ustawia zachowanie po zamknięciu okna, w tym przypadku jest to dświeżenie listy danych encji.

 Rys 6.30. Przykładowa akcja dodawania encji

Kontroler nowego okna posiada 2 podstawowe metody send() i cancel(). Metoda send() wysyła poprzez serwis dane wprowadzone przez użytkownika i zamyka okno. Natomiast metoda cancel() jest odpowiedzialna za zamknięcie okna bez wysyłania danych.



Rys 6.31. Przykładowy kontroler encji

Przykładowy widok dodawania przedstawia poniższa ilustracja:



Rys 6.32. Przykładowy widok dodawania encji

* 1. Edytowanie

Edytowanie encji jest realizowane za pomocą metody edit(). Metoda ta działa w analogiczny sposób co metoda add() z tą różnicą że nie tworzy nowego obiektu a przekazuje (poprzez obiekt resolve) do nowego okna obiekt wybrany przez użytkownika.

 Rys 6.33. Przykładowa akcja edytowania

Kontroler edycji encji działa w podobny sposób co kontroler dodawania z tymże wywołuje inną akcję serwisu.

 Rys 6.34. Przykładowy kontroler edytowania

Przykładowy widok edycji:

 Rys 6.35. Przykładowy widok edytowania

* 1. Usuwanie

Za usuwanie danej encji odpowiada metoda delete(). Otwiera ona nowe okno potwierdzające chęć usunięcia, do którego przekazywana (tak jak w przypadku edycji poprzez obiekt resolve) jest encja wybrana przez użytkownika. Tak jak w przypadku akcji dodawania i edytowania po zwróceniu kontroli przez nowe okno lista encji jest odświeżana.

 Rys 6.36. Przykładowa akcja usuwania

Kontroler usuwanie encji posiada 2 metody confirm() która potwierdza usuwanie i poprzez serwis wysyła żądanie do serwera oraz metodę decline() która zamyka okno bez podejmowania jakiejkolwiek akcji.

 Rys 6.37. Przykładowy kontroler usuwania

Widok okna potwierdzania chęci usunięcia:

 Rys 6.38. Przykładowy widok potwierdzania usuwania

## PopUp

Kontroler Popup oprócz podstawowych operacji CRUD posiada metodę activatePopUp która jest odpowiedzialna za realizację założenia zgodnie z którym tylko jeden popup może być aktywny.

 Rys 6.39. Metoda activePopUp

## Statystki

Moduł statystyk ze względu na swoją specyfikę jest zorganizowany w odmienny sposób. Użytkownik wybierając (za pomocą menu) statystyki do wyświetlenia wywołuje odpowiednią akcję która podmienia kod HTML. Wszystkie akcje kontrolera StatisticsController za pomocą serwisu pobierają odpowiednie strony w postaci kodu HTML i przypisują je do zmiennej pageHtml

 Rys 6.40. Kontroler modułu statystyk

Zmienna pageHtml w widoku jest wykorzystywana przez dyrektywę bind-html-compile aby podmieniać i jednocześnie kompilować wewnętrzny kod HTML danego elementu

 Rys 6.40. Podmiana kodu HTML w widoku

Każda ze stron zawierająca statystki korzysta z zewnętrznego modułu angular-charts (patrz 3.4.2) służącego to rysowania diagramów przy pomocy Tagu <canvas>. Przykładowa strona statystyk:

 Rys 6.41. Wyświetlanie statystyk

## Animacja ładowania

Animacja ładowania danych jest realizowana za pomocą tzw. interceptora . Przechwytuje on każde żądanie realizowane za pomocą obiektu $http i odpowiednio pokazuje lub chowa animację ładowania.

 Rys 6.42. HttpInterceptor

Sama animacja jest realizowana z pomocą dyrektywy LoaderDirective zaprezentowanej poniżej:

 Rys 6.43. Dyrektywa loader

## Edycja tekstu

Edycja tekstu została zrealizowana za pomocą zewnętrznego modułu text-angular (patrz 3.4.1). Aby używać wyżej wymienionego modułu po zaimportowaniu go wystarczy opatrzyć odpowiednią kontrolkę atrybutem text-angular.

Rys 6.43. Przykład użycia text-angular

## Wstawki

Wstawki ze względu na to iż mogą zostać dodawane tylko w tekście zostały dodane do modułu text-angular za pomocą udostępnianego poprzez niego API.

 Rys 6.44. Rozszerzenie narzędzia text-angular

Sam system wstawek w warstwie prezentacji został zaimplementowany za pomocą kontrolera i tzw pickerów. Kontroler kieruje przepływem informacji pomiędzy konkretnymi pickerami a serwisem który waliduje wstawkę. Każdy z pickerów posiada swój widok z formularzem który pozwala wygenerować odpowiednią wstawkę.

InsetController posiada metody pozwalające na zmianę typu wstawek (poprzez podmianę formularz )oraz zamianę konkretnych danych otrzymanych z pickera na odpowiadający im Tag wstawki.

Rys 6.45. Kontroler dodawania wstawki

Każdy z pickerów został zaimplementowany w podobny sposób. Jest to dyrektywa która na odpowiednie zdarzenie (np. kliknięcie w kontrolkę) otwiera nowe okno które umożliwia wybranie odpowiednich elementów do wstawki np. zdjęcia czy strona. Następnie po potwierdzeniu dane są zwracane do kotrolera wstawek.

 Rys 6.46. Przykładowy picker wstawki

# Podsumowanie

Na etapie projektowania aplikacji założono, że system ServiceCMS zostanie zaprojektowany do obsługi stron internetowych ze szczególnym uwzględnieniem personelu nietechnicznego. Przyjęto, że strony będą należeć głównie do firm usługowych, gdzie pracownicy mają bezpośredni wpływ na wykonywana dla klienta usługę. Na potrzeby zaprezentowania tego aspektu aplikacji wykorzystano salon fryzjerski. Przed wdrożeniem systemu opracowano zarys wymagań funkcjonalnych i niefunkcjonalnych z pomocą eksperta dziedzinowego. Umożliwiło to dokładniejsze określenie scenariusz przypadków użycie, na jakie może natrafić użytkownik.

Implementacja funkcjonalności systemu ServiceCMS nie sprawiła większych problemów. Dzięki poprawnie zaprojektowanej architekturze oraz wdrożeniu wzorców projektowych znacząco obniżono nakład pracy, którzy był konieczny do implementacji oraz przetestowania zaprojektowanych rozwiązań.

Po wykonaniu systemu uznano, że narzędzia wykorzystane podczas implementacji spełniły swoje zadanie. Dzięki szerokiemu zastosowaniu biblioteki AngularJS możliwa była realizacja założenia, mówiącego o personelu nietechnicznym. Bogaty zbiór dodatkowych rozszerzeń, takich jak Text-angular, czy Angular-charts sprawił, że aplikacja ServiceCMS jest estetyczna i łatwa w obsłudze dla osoby nie mającej na co dzień związku z administrowaniem stron internetowych.

# Bibliografia

[1] Beck Kent, TDD. Sztuka tworzenia dobrego kodu, Helion, Gliwice 2014.

[2] Freeman Adam, ASP.NET MVC 5. Zaawansowane programowanie, Helion, Gliwice 2015.

[3] Freeman Adam, AngularJS. Profesjonalne techniki, Helion,2015.

[4] Fryźlewicz [**Zbigniew**](http://helion.pl/autorzy/zbigniew-fryzlewicz,zbigniewfryzlewicz.htm) ,**Bukowska** [**Ewa**](http://helion.pl/autorzy/ewa-bukowska,ewabukowska.htm) ,NikończukDaniel ASP.NET MVC 4. Programowanie aplikacji webowych, Helion, Gliwice 2013.

[5] Martin Robert C., Czysty kod. Podręcznik dobrego programisty, Helion, Gliwice 2009.

[6] Metsker Steven John, C#. Wzorceprojektowe, Helion, Gliwice 2005.