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

1. Menadżer sesji



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 dzięku generycznepu typowi T.

1. Obsługa ustawień regionalnych



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

1. 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 mechaniźmie refleksji.

1. 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.

1. UnitofWork



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

1. 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.

1. CRUD Logic

Ze względu na duże podobieństwo operacji CRUD dla wszystkich encji, poniżej przedstawiono przykładową implementację dla encji User. Każda metoda opatrzona jest w blok try-catch, który gwarantuje odpowiednie zabezpieczenia tzw. information-flow. W przypadku nieprzewidzianego zachowania wyjątek zostanie zalogowany do pliku. Zostaje zwrócona również wiadomość o statusie operacji.



Metody o nazwie GetById oraz GetAll zawierają kod który realizuje pobieranie encji, bądź zbioru encji według określonych zasad. W przypadku braku encji o określonym kluczu głównym zwracany jest obiekt pusty. Podobnie w przypadku metody GetAll – tutaj zwracany jest lista, która może być pusta. Błędy są przechwytywane przez klasę Logger.

Operacja Insert umieszcza obiekt w bazie danych. Generyczne repozytorium przy udziale Entity Framework rzutuje obiekt języka C# na bazodanową krotkę. Umieszczenie w bazie danych następuje, gdy obiekt spełnia założenia, które zostały zdefiniowane przy jego tworzeniu - dba o to generyczne repozytorium.

Aktualizacja obiektu w bazie danych następuje, gdy obiekt spełnia założenia, które zostały zdefiniowane przy jego tworzeniu. Ponownie – operacja ta jest wykonywana przez instancję repozytorium generycznego.

Operacja usunięcia krotki z bazy danych jest zbieżna z metodami Insert oraz Update. Jeśli w bazie danych istnieje obiekt o przekazanej wartości klucza głównego generyczne repozytorium usuwa odpowiednią krotkę.

1. File



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.



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

1. Inset

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



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



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



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

LocalLinkParser



Każdy typ wstawki ma zaimplementowańą 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.

News

Update jest insertem



W celu impelementacji historii aktualności oraz stron napisano metodę powyższą metodę Update. Wbrew nazwie, metoda ta przeprowada proces umieszczenia w bazie danych nowego rekordu, który wskazuje na swojego poprzednika. Dzięki temu system pamięta wszelkie zmiany, które dokonano na encjach News oraz Page. Zależności bazodanowe zostały stworzone w postaci klucza obcego o nazwie RestoreNewsId. Metoda przypisuje wszystkie własności encji starszej do jej nowej wersji, łącznie z kategoriami aktualności.



Metoda Update dla encji Page działa podobnie jak ta, zaimplementowana w punkcie… dla encji News. Tutaj również następuje umieszczenie nowego rekordu, wraz z przekazaniem do niego wszystkich własności obiektu pochodnego. W zaktualizowanej wersji strony przypisano również pliki, obecne w poprzedniej wersji.

Aplikacji ServiceCMS w dużej mierze opiera się na procesie rejestracji usług. Proces ten wymagał przeciążenia metody o nazwie GetAllServicesWithMatchingCriteria. Było to niezbędne do uporządkowania przepływu informacji w systemie.



Metoda GetAllServicesWithMatchingCriteria, która otrzymuje jako argument datę, zwraca listę usług, zarejestrowanych na określony czas.

Metoda GetAllServicesWithMatchingCriteria, która otrzymuje jako argument dostarczyciela, zwraca listę usług, wykonywanych przez określoną osobę, bądź na określonym stanowisku.

Metoda GetAllServicesWithMatchingCriteria, która otrzymuje jako argument dostarczyciela oraz datę, zwraca listę usług. Jej zawartość to część wspólna usług, wykonywanych przez określoną osobę, bądź na określonym stanowisku, w określonym czasie.

Timeblocks



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.

Metoda GetServiceTypesTimeBlocks tworzy z typów usług podobne bloki czasu, jak opisane w punkcie…. 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.



Metoda CheckAvailability sprawdza, czy dany typ usługi może być zarejestrowany na daną datę. Metoda wykorzystuje opisane wcześniej metody do sprawdzenia czy bloki czasowe się na siebie nie nakładają . Jeśli taka sytuacja zaistnieje, dany typ usługi jest oznaczany jako niemożliwy do wykonania.

Settings



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.



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.

Statistics

Moduł statystyk jest szerzej opisany w rozdziale … Poniżej zostanie zaprezentowany kod, odpowiadający za realizację wymienionych tam funcjonalności. Moduł ten opiera się na filtrach - obiektach, zawartych w ASP.NET MVC. Umożliwiają one wywołanie zbioru metod podczas wykonywania żądań HTTP na określonych akcjach w określonych kontrolerach.



Klasa StatisticsFilter przechwytuje dane użytkownika, który korzysta z aplikacji klienckiej. Dane, które są przechowywane w obiekcie StatisticsInformationModel to adres IP, nazwa kontrolera i akcji, z jakiej korzystał, oraz data wykonania żądania.



Metoda AddEntry otrzymuje dane, przechwycone przez filtr statystyczny i umieszcza je w bazie, w celu późniejszego opracowania ich przez metody warstwy logiki modułu statystyk.

Klasa statyczna DistinctByHelper umożliwia wybranie rekordów bez potwarzania klucza, który określimy jako jeden z atrybutów metody. Jest to rozwiązanie wyjątkowo skalowalne, oparte na mechaniźmie generyczności. Użycie słowa kluczowego yield zapewnia poprawną zawartość zwracanej kolekcji.



Klasa statyczna BetweenDatesValidationHelper umożliwia zbudowanie obiektu Expression w celu umieszczenia go w wyrażeniu lambda. Zapytanie to odpowiada następnie za filtrowanie wyników, pobranych z bazy danych. Rozwiązanie to było koniecznie, aby umożliwić administratorowi określenie ram czasowych, z jakich mają być pokazane statystyki. Powyższa metoda umożliwia implementację czterech ścieżek, jakie można wykonać podczas korzystania z modułu statystyk.

Metoda GetUsersPerCountry odwołuje się do zewnętrznego serwisu za pomocą biblioteki RestSharp. Zwracany jest słownik, który zawiera nazwę kraju jako klucz, oraz liczbę użytkowników, którzy przypadają na ten kraj, jako wartość. Metoda GetUsersForEveryMonth zwraca ten sam obiekt, co metoda GetUsersPerCountry. Ustawienia regionalne są zaimplementowane w sposób twardy.



Powyżej zaprezentowano metody odpowiadające za sformatowanie wyników dla modułu statystyk. Dzięki nim, administrator może korzystać z estetycznych i jasnych danych. Ponownie, jak w przypadku reszty metod z modułu statystyk, wykorzystano słowniki danych.

DIRegister

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



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

Cryptograhpy



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

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, któryw 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



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.

Moduł MailManagement służy do obsługi newslettera oraz formularza kontaktowego, umieszczonego na stronie klienckiej. ServiceCMS posiada możliwość zdefiniowania danych uwierzytelniających dla e-maila właściciela zakładu usługowego. Dzięki temu, moduł ten jest możliwie najbardziej bezobsługowy oraz elastyczny.

Metoda SendMail umożliwia wysłanie maila do zbioru adresatów, bądź – w wersji przeciążonej – do jednego adresata. W ciele metody następuje definiowanie treści oraz tematu wiadomości. Wywoływana jest też metoda, konfigurująca klienta SMTP (ang..), który opisany jest poniżej. Cała operacja jest opatrzona blokiem try-catch, na wypadek, gdyby napotkano błędy związane z połączeniem internetowym.

W ciele metody ConfigureSmtpClient następuje konfiguracja klienta SMTP, który jest użyty do wysłania maila za pomocą modułu Newsletter oraz RegistratedServices. Konfiguracja klienta jest zależna od ustawień, pobranych z modułu Settings.

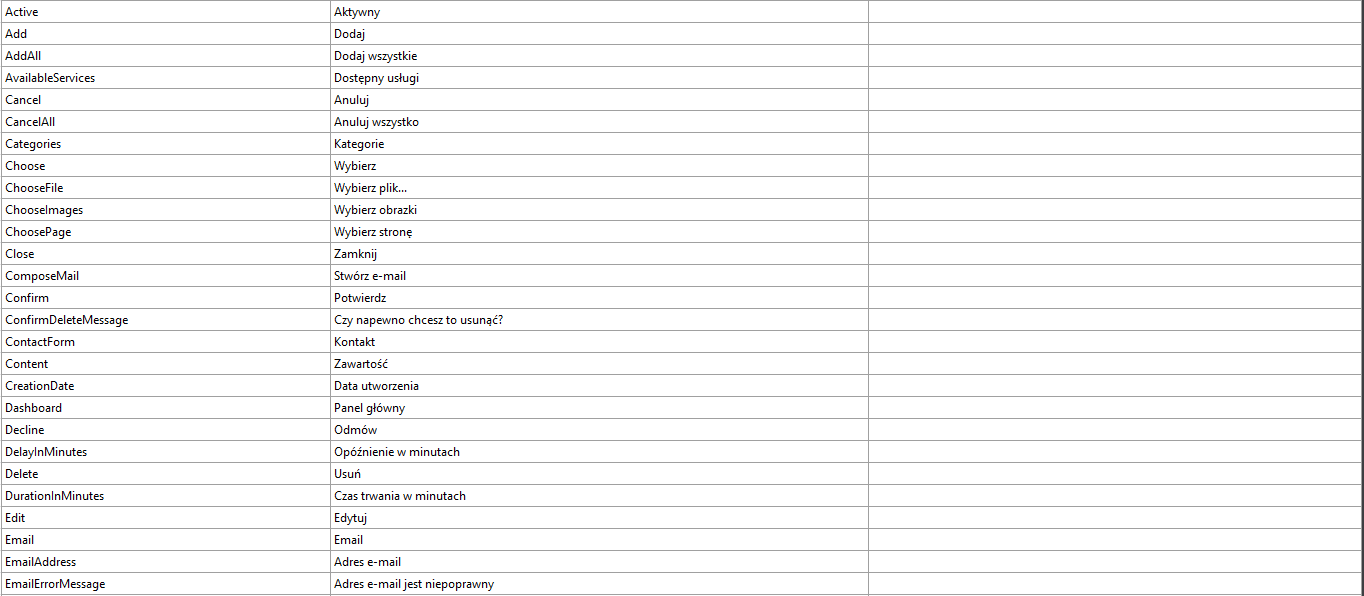
RestSharp



Metoda statyczna GetIpInfo, korzystając z biblioteki RestSharp, pobiera kraj dla podanego w atrybucie metody adresu IP. Metoda jest zabezpieczone przed błędami z połączeniem internetowym za pomocą bloku try-catch, przechwytującego zarówno wyjątek połączenia, jak i błąd serializacji wyniku.



Clasa BaseController jest dziedziczona przez każdy kontroler, zarówno w aplikacji adminstracyjnej, 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.



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.