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 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 możemy bezpiecznie pobierać informacje 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.

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 DbModelilder.

1. UnitofWork



Implementacja wzorca projektowego UnitOfWork. Implementacja ta ma zapewnić spójność danych, odwzorować transakcję bazodanową. Dzięki klasie unitofwork mamy w jednym miejscu repozytoria wszystkich encji.

1. Logger



Klasa logger odpowiada za bezpieczne logowanie przechwyconych wyjątków. Klasa zapisuje wyjątki w ścieżce podanej w stałej LOG\_PATH.

1. CRUD Logic

Ze względu na duże podobieństwo operacji CRUD dla wszystkich encji poniżej przedstawiono przykładowe dla encji User. Każda metoda opatrzona jest w blok try Catach który gwarantuje bezpieczeństwo. W przypadku nieprzewidzianego zachowania zostanie zalogowany wyjątek przy pomocy klasy Logger oraz zostaie zwrócona wiadomośc o nie powodzeniu operacji









1. File



Oprócz podstawowych operacji CRUD klasa FileSerivce posiadę metodę UploadWithInsert. Odpowiada ona za dodanie encji do bazy z jednoczesnym zapisem pliku na dysku.



Metoda GetFilePath odpowiada za unikalne nadanie nazwy plikom – dzięki Giud.NewGuid().

Dzięki temu upewniamy się 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 całość. Posiada metodę która za pomocą wyrażenia regularnego zamienia Tag wstawki na odpowiadający mu kod.

InsetRecognizer



Klasa InsetRecognizer odpowiada za walidację znalezionych wstawek. Klasa waliduje czy dana wstawka ma poprawną składnię czy posiada wszystkie wymagane argumenty i czy te argumenty mają poprawne wartości.

ArgumentValidator



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 się udało to znaczy że argument jest prawidłowy

LocalLinkParser

Każdy typ wstawki ma zaimplementowańą klasę typu Parser. Ze względu na podobieństwo implementacji poniżej przedstawiono klasę LocalLinkParser mającą na celu wygenerowanie linku do lokalnej strony



News

Update jest insertem



Page

Update



Services







Timeblocks



Metoda tworząca z zarejestrowanych usług bloki czasu typu <początek,koniec>. Dla każdej fazy obliczany jest czas trwania czyli czas początku usługi + czas poprzednich faz + opóźnienie poprzednich faz.

Metoda tworząca z typów usług takie same bloki czasu jak wyżej z tą różnicą że punktem wyjścia jest wybrana data przez użytkownika. Czyli dla każdej fazy każdego typu usługi tworzony jest bkol czasowy gdzie do daty wybranej przez użytkownika dodawany jest czas poprzednich faz i opóźnień tych faz.



Metoda sprawdzająca 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 tak jest to dany typ usługi jest oznaczany jako nie możliwy.

Settings



Metoda rzutująca wartość ustawienia odczytanego z bazy danych w postaci ciągu znaków na typ pasujący do do pola InputType.Dla bezpieczeństwa jeśli się nie uda rzutować wartości to ta wartość pozostaje ciągiem znaków.



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

Statistics











DIRegister



Klasa odpowiedzialna za rejestrację komponentów aplikacji w kontenerze IOC. Klasa rejestruje konkretne implementacja do wstrzyknięcie jeśli żądany jest dany interfejs.

Cryptograhpy



Moduł kryptografi odpowiedzialny jest za hashowanie? Haseł użytkowników. Dobrym podejściem w projektowaniu takich aplikacji jest nie przechowywanie haseł w prostej postaci

Klasa PAssworManager posiada 2 metody które są odpowiedzialne za obsługę zahashowanych haseł. Metoda GeneratePasswordHash generuje na podstawie podanego hasła jego zakodowaną postać. Do samego hasła dodawana jest również losowa „sól” w postaci ciągów znaków która znacznie zwiększa bezpieczeństwo. Metoda IsPasswordMatch porównuje czy podane hasło zgadza się z zakodowanym hasłem. W systemie po zakodowaniu hasła nie ma możliwości go odczytania w zwykłej postaci.

FileManager



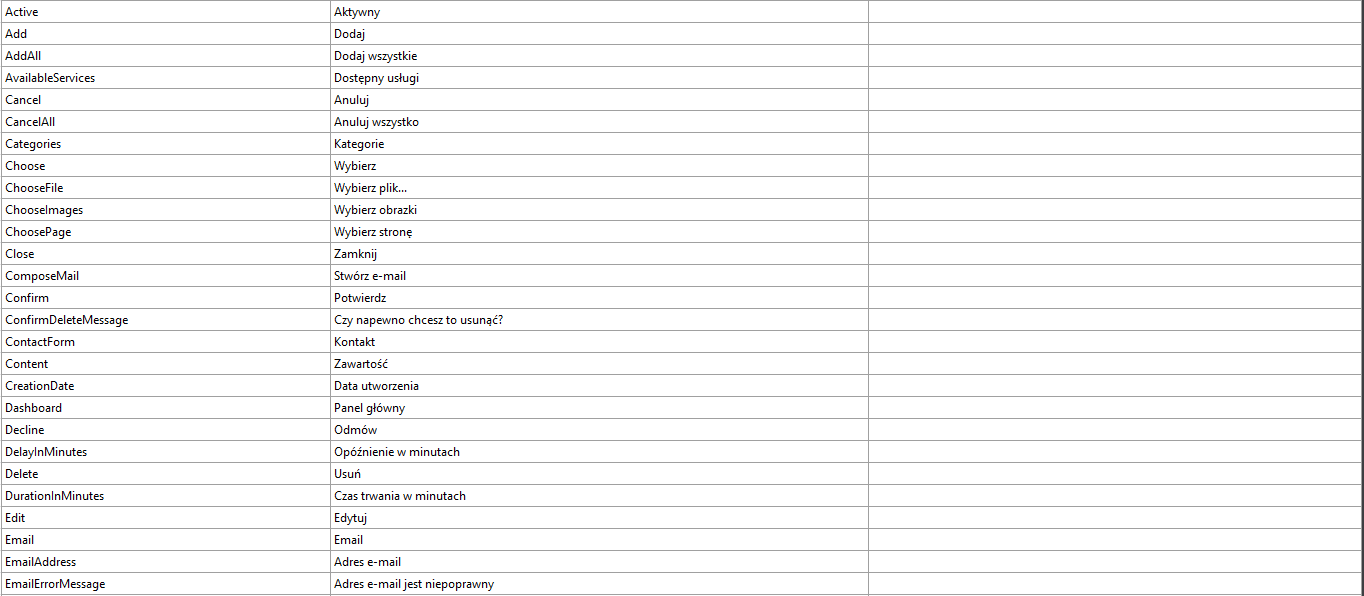
Klasa FileManager posiada zbiór metod które pozwalają w bezpieczny i wygodny sposób działać na systemie plików

MailManagment





Resources



RestSharp

