Uniwersytet Rzeszowski

Michał Rembisz

Numer albumu: 092092

Daniel Smolak

Numer albumu: 092098

Informatyka, studia stacjonarne I stopnia, semestr 5

Projekt inżynierski II pt.:

Implementacja aplikacji webowej do translacji formatów opisów bibliograficznych BibTeX <-> RIS z opcjami filtrowania

Rzeszów, 2018 r.

Rozdziały:

1. Cel i zakres projektu.

2. Opis obszaru tematyki projektu.

3. Projekt ontologii OWL ...

4. Specyfikacja projektu aplikacji do generowania i edycji ontologii OWL (w tym uzasadnienie zaproponowanego rozwiązania w kontekście projektu aplikacji).

5. Wykorzystane metody, narzędzia i technologie (w tym uzasadnienie wyboru).

6. Opis implementacji aplikacji (w tym uzasadnienie zastosowanych rozwiązań w kontekście implementacji aplikacji).

7. Podsumowanie (w tym opis możliwych zastosowań).

1. Cel i zakres projektu.

Celem projektu było stworzenie aplikacji webowej umożliwiającej konwersję plików zapisanych w formacie BibTeX na plik w formacie RIS i na odwrót. Program wczytuje plik wskazany przez użytkownika umożliwia mu przefiltrowanie tego pliku według dostępnych kryteriów tzn. według informacji zawartych w tych plikach (umożliwia np. wybranie konkretnych autorów których publikacje chcemy zapisać w pliku wynikowym) i zapisanie przefiltrowanych danych do pliku wynikowego w formacie BibTeX lub RIS.

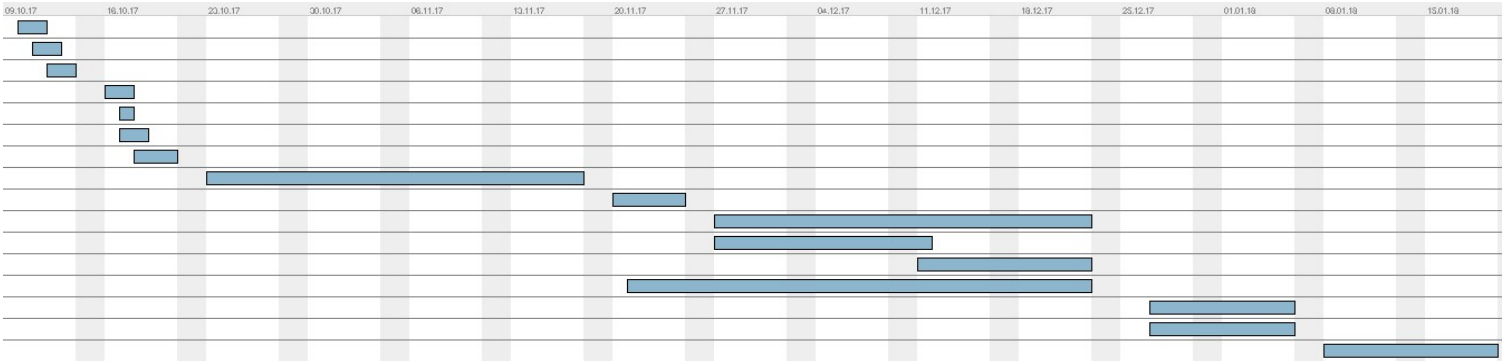
Zakres jaki spełniliśmy w celu stworzenia projektu:

- Stworzenie zarysu i planu projektu,
- Zbudowanie aplikacji webowej pozwalającej na wyświetlanie danych, filtrację i usuwanie danych zawartych w plikach BibTeX/RIS,
- Sporządzenie niezbędnych dokumentów tj. schematy, diagramy,
- Wykonanie dokumentacji projektu.

Cała struktura powstawania projektu oparta była o przygotowany na samym początku wykres Gantta, który niestety musiał zostać kilka razy zmodyfikowany ze względu na inne terminy zakończenia zadań niż przewidywane. Wykres składa

się z następujących po sobie zadań, co pozwoliło na stworzenie projektu w oparciu o wcześniej określone zdania, i łatwe dzielenie się zadaniami w zespole.

Nr	Zadanie:	Data Rozpoczęcia	Czas trwania	Data Zakończenia
Zad 1	Stworzenie zarysu projektu, podział obowiązków, wybór technologii.	10 paź	2	11 paź
Zad 2	Stworzenie wykresu Gannta	11 paź	2	12 paź
Zad 3	Zaprojektowanie serwerowej części aplikacji	12 paź	2	14 paź
Zad 4	Zaprojektowanie Graficznego Interfejsu Użytkownika	16 paź	2	17 paź
Zad 5	Stworzenie diagramu przypadków użycia	17 paź	1	17 paź
Zad 6	Wprowadzenie danych do testowych plików	17 paź	2	18 paź
Zad 7	Zaimplementowanie klas w projekcie	18 paź	3	20 paź
Zad 8	Zaimplementowanie odczytu z pliku BibTeX/RIS do programu	23 paź	20	17 lis
Zad 9	Zaimplementowanie zapisu danych do pliku BibTeX/RIS	20 lis	5	24 lis
Zad 10	Zaimplementowanie filtracji danych	27 lis	20	22 gru
Zad 11	Zaimplementowanie w pełni dynamicznej tabeli	21 lis	11	11 gru
Zad 12	Zaimplementowanie usuwania danych	11 gru	10	22 gru
Zad 13	Stworzenie pełnej dokumentacji i podręcznika użytkownika	21 lis	24	22 gru
Zad 14	Zaimplementowanie opcji pobrania dokumentacji i podręcznika	27 gru	8	5 sty
Zad 15	Wprowadzenie pełnych danych testowych do plików i testowanie	27 gru	8	5 sty

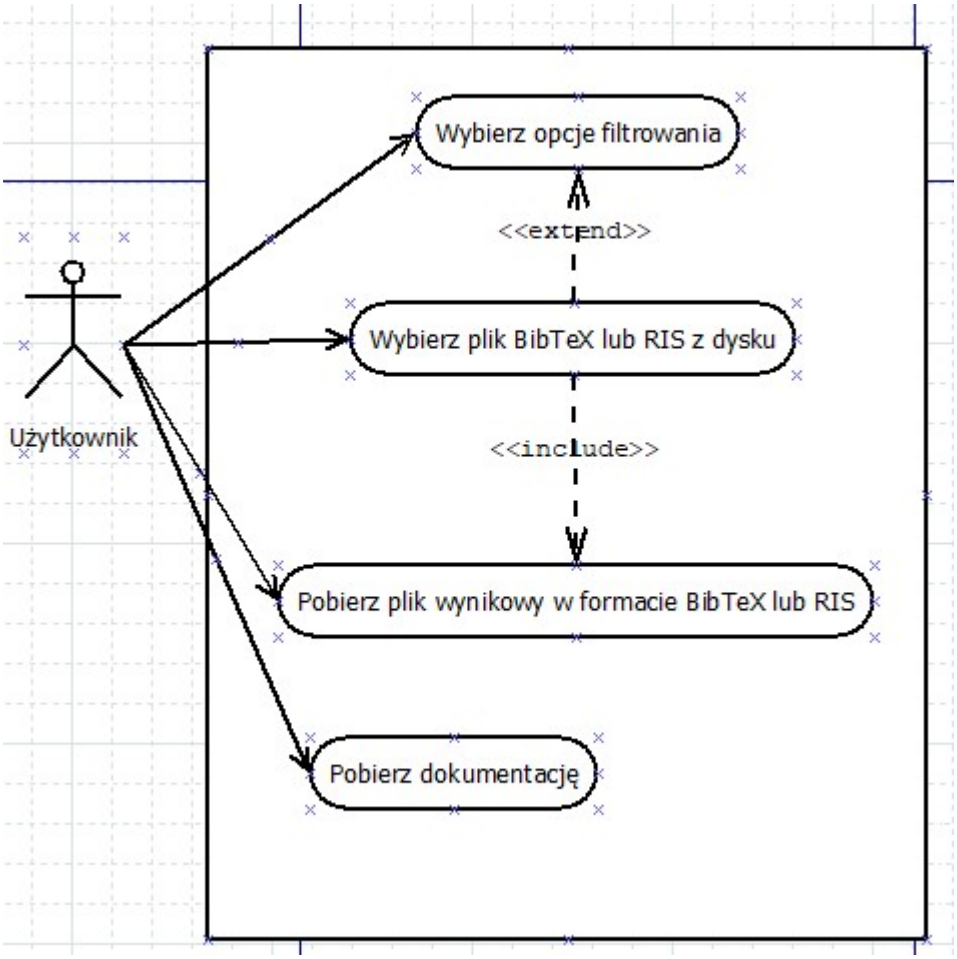


2. Plan aplikacji.

W początkowym planie aplikacji przyjęliśmy następujące założenia
Aplikacja powinna:

- umożliwiać użytkownikowi wybranie pliku z dysku
- rozpoznać rozszerzenie wybranego pliku i uruchomić odpowiednie dla niego metody
- jeśli plik posiada obsługiwane rozszerzenie (.bib lub .ris) wczytać jego zawartości która powinna być zapisana zgodnie ze standardem dla obsługiwanych formatów do odpowiedniej listy

- przeprowadzić translacje danych do stworzonej przez nas klasy zawierającej wszystkie wspólne pola formatów BibTeX i RIS
- wyświetlić listę obiektów powyższej klasy w postaci w pełni dynamicznej tabelki – umożliwiającej przesuwanie na boki gdy nie mieści się na ekranie, automatycznie dopasowującej liczbę wierszy do ilości danych i liczbę kolumn oraz ich nagłówki do danych zawartych w plikach o ile te dane mogą być zapisane do drugiego formatu
- na podstawie tych danych wygenerować odpowiednie opcje filtrowania
- po wybraniu opcji przefiltrować dane i wyświetlić je w tabelce
- umożliwiać odznaczenie opcji filtrowania i pokazać wszystkie dane
- umożliwiać trwałe usunięcie żądanych rekordów bez opcji ich przywrócenie (chyba że zostanie ponownie wczytany plik początkowy)
- umożliwiać zapisanie danych do pliku BibTeX lub RIS i pobranie go poprzez przeglądarkę
- umożliwiać pobranie dokumentacji i podręcznika użytkownika



3. Wykorzystane technologie.

ASP .NET MVC 5– Framework .NET technologia ASP .NET i rodzaj aplikacji MVC w wersji 5. Wybraliśmy tą technologię, ponieważ w prosty sposób można stworzyć w niej aplikację webową zgodną z najnowszymi standardami projektowania aplikacji (np. RESTfull API, Single Page Application) i nie wymaga to dodatkowych narzędzi, technologii, frameworków, ale można ich użyć. Używaliśmy starszej wersji, ponieważ wersje 5 i 6 są w kompatybilne i nie wymagają żadnej konfiguracji więc standardowo tworząc projekt używa się wersji 5 natomiast można w dowolnym momencie użyć elementów z wersji 6, których my nie potrzebowaliśmy (wersja 6 jest raczej niedużą wersją i nie wprowadza zbyt wielu zmian do najczęściej używanych zmian w wersji 6 należą nowe znaczniki obsługujące zapytania do bazy danych). W porównaniu do innych technologii i języków tego typu kod programu jest krótszy i prostszy do zrozumienia (od np. frameworka Spring z Javy) i znacznie bardziej czytelny (szczególnie jeśli popatrzymy na np. PHP nie licząc frameworka Laravel który jest oparty na technologii ASP .NET) oraz co jest dla nas ważne materiały, informacje i wsparcie są bardzo łatwo dostępne (oficjalne strony Microsoftu i wspierane przez nich blogi, fora), a my mamy doświadczenie w tej technologii.

C# – język programowania. Wybraliśmy ten język, ponieważ najlepiej się nam w nim pracuje oraz mamy z nim największe doświadczenie.

Javascript – najpopularniejszy język wykonywany w po stronie przeglądarki. Nie używaliśmy go w widoczny sposób ale wykorzystują go używane przez nas frameworki i technologie.

Silnik Razor - to silnik renderujący wprowadzony w MVC3, pozwalający na bardzo łatwe oddzielenie kodu HTML od kodu aplikacji. Posiada prostą składnię. W porównaniu do starszych silników renderujących, aby uzyskać taki sam efekt, wymaga napisania mniejszej ilości kodu. Aby wyświetlić wartość zmiennej, wystarczy postawić przed nią znak „@”, analogicznie postępujemy w przypadku pętli oraz innych elementów nienależących do składni języka HTML, czy JavaScript. Dodatkowo RAZOR obsługuje klamry, które są bardzo pomocne, gdy mamy więcej niż jedną linię kodu wymagającego użycia RAZORa. Używany przez nas w wersji 3.23 ostatniej stabilnej wersji silnik ten został zastąpiony przez silnik Razor 2.0 dla frameworka .NET CORE którym Microsoft stara się zastąpić .NET Framework aktualnie przynajmniej dla aplikacji webowych i konsolowych, nie

używaliśmy nowszego frameworka ze względu na to że jeszcze nie pokrywa wszystkich elementów .NET frameworka (aktualnie 75%) a my nie potrzebowaliśmy aplikacji wieloplatformowej.

AngularJS - otwarty framework oparty na języku JavaScript, wspierany i firmowany przez Google, wspomagający tworzenie i rozwój aplikacji internetowych na pojedynczej stronie. Przez nas używany do obsługi dynamicznej tabelki niestety silnik Razor wymaga ponownego renderowania strony do zmiany wyświetlanych danych a my chcieliśmy aby dane w tabelce zmieniały się gdy tylko zmiana nastąpi.

Twitter Bootstrap - framework CSS, rozwijany przez programistów Twittera, wydawany na licencji MIT. Zawiera zestaw przydatnych narzędzi ułatwiających tworzenie interfejsu graficznego stron oraz aplikacji internetowych. Bazuje głównie na gotowych rozwiązaniach HTML oraz CSS (kompilowanych z plików Less) i może być stosowany m.in. do stylizacji takich elementów jak teksty, formularze, przyciski, wykresy, nawigacje i innych komponentów wyświetlanych na stronie. Używany przez nas w wersji 4.0.0-beta.

NuGet – jest rozszerzeniem do Visual Studio ułatwiającym zarządzanie referencjami do bibliotek (tzw. system zarządzania pakietami). Pozwolił nam w prosty sposób dodawać potrzebne frameworki i biblioteki do projektu tak dodaliśmy np. Bootstrapa i Angulara

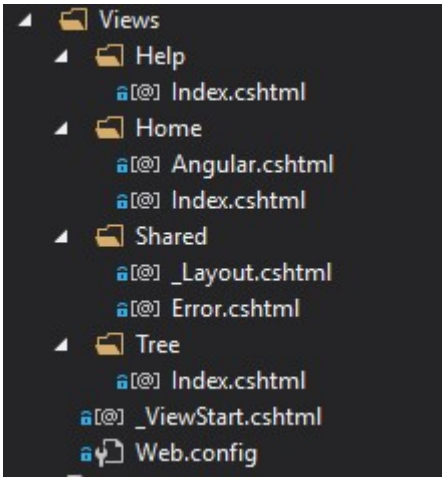
Visual Studio 2017 - środowisko programistyczne pozwalające na implementacje aplikacji w między innymi języku C# wybraliśmy najnowszą wersję pomimo, że wystarczyłaby starsza wersja z 2015 roku, ponieważ jest ona już dopracowana i możne w pełni zastąpić starszą wersję, a jest bardziej przyszłościowa i oferuje umożliwia tworzenie aplikacji wykorzystujących nowe standardy i technologie których my nie używaliśmy w tym projekcie (cross-platform), ale zdążyliśmy się już przyzwyczaić do tej wersji i w przyszłości planujemy budować aplikacje między platformowe. Natomiast wybór padł akurat na Visual Studio ponieważ tworzy go akurat firma opowiadająca za język C# oraz ze względu na to że jest prawdopodobnie najpopularniejszym IDE do używanych przez nas technologii.

4. Opis implementacji i diagram klas

Z powodu wielkości naszego diagramu klas przytaczane będą jego fragmenty

w trakcie opisu implementacji natomiast całość dostępna jest w głównym folderze projektu w plikach DiagramKlas.cd oraz DiagramKlas.jpg.

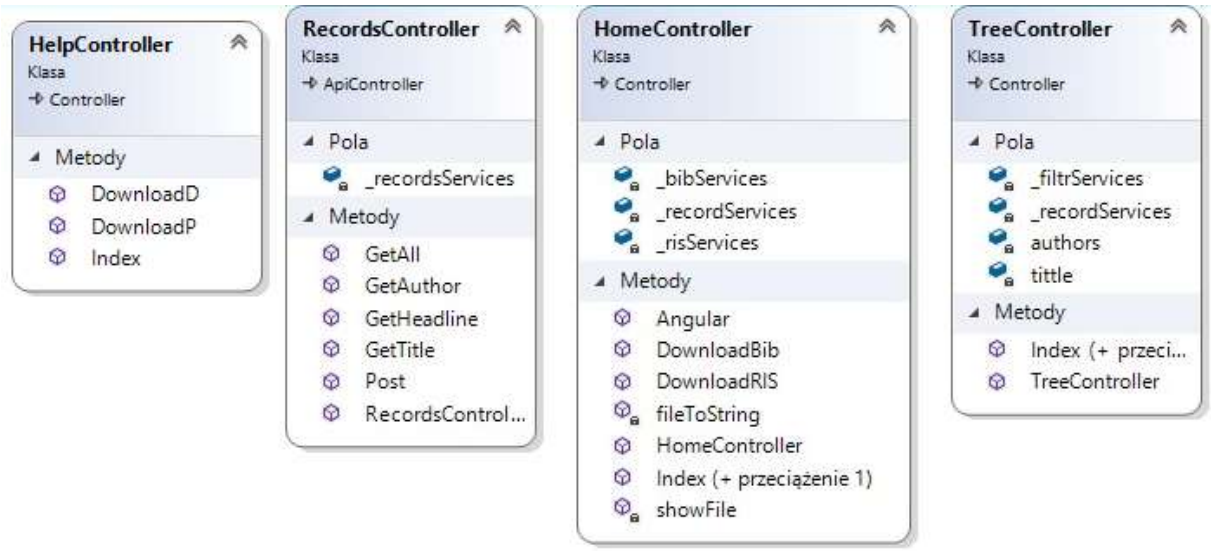
Projekt został wykonany zgodnie z architekturą MVC (Model-Widok-Kontroler)
Widoki:



Widoki czyli część aplikacji która jest wyświetlana użytkownikowi i przez którą odbywa się komunikacja z użytkownikiem są zebrane w folderze Views. Są to pliki z rozszerzeniem cshtml jest to rozszerzenie dla silnika Razor w którym można używać kodu html w połączeniu z elementami języka C# z czego następnie jest generowany jest kod zrozumiały dla przeglądarki. Widoki mogą być dowolną stroną internetową napisaną w dowolnych językach zrozumiałych dla przeglądarki w przypadku aplikacji internetowych. Zdecydowaliśmy się użyć domyślnego i zalecanego sposobu tzn. stworzenie widoku jako tzw. „single page application” za pomocą silnika Razor ze względu na to, że jest to sposób prosty i przyjemny dla programistów którzy nie przepadają za tworzeniem stron internetowych przy użyciu kodu html, a oprócz tego single page application zapewnia wydajność aplikacji (wystarczy załadować tylko interesujący nas fragment strony gdy użytkownik chce go zobaczyć) i przejrzystość kodu oraz znacznie zmniejsza jego ilość (nie trzeba pisać kilka razy tego samego).
Widoki są podzielone na 4 foldery ze względu a to jaki kontroler ich używa jedynie pliki z folderu Shared są używane zawsze.
Po włączeniu aplikacji ładowany jest plik _ViewStart w którym informujemy który plik zawiera szablon naszej strony u nas jest to plik _Layout.cshtml
Plik Error.cshtml jest widokiem pokazywanym jeżeli w aplikacji wystąpił błąd.
Web.config to plik z konfiguracją silnika Razor.

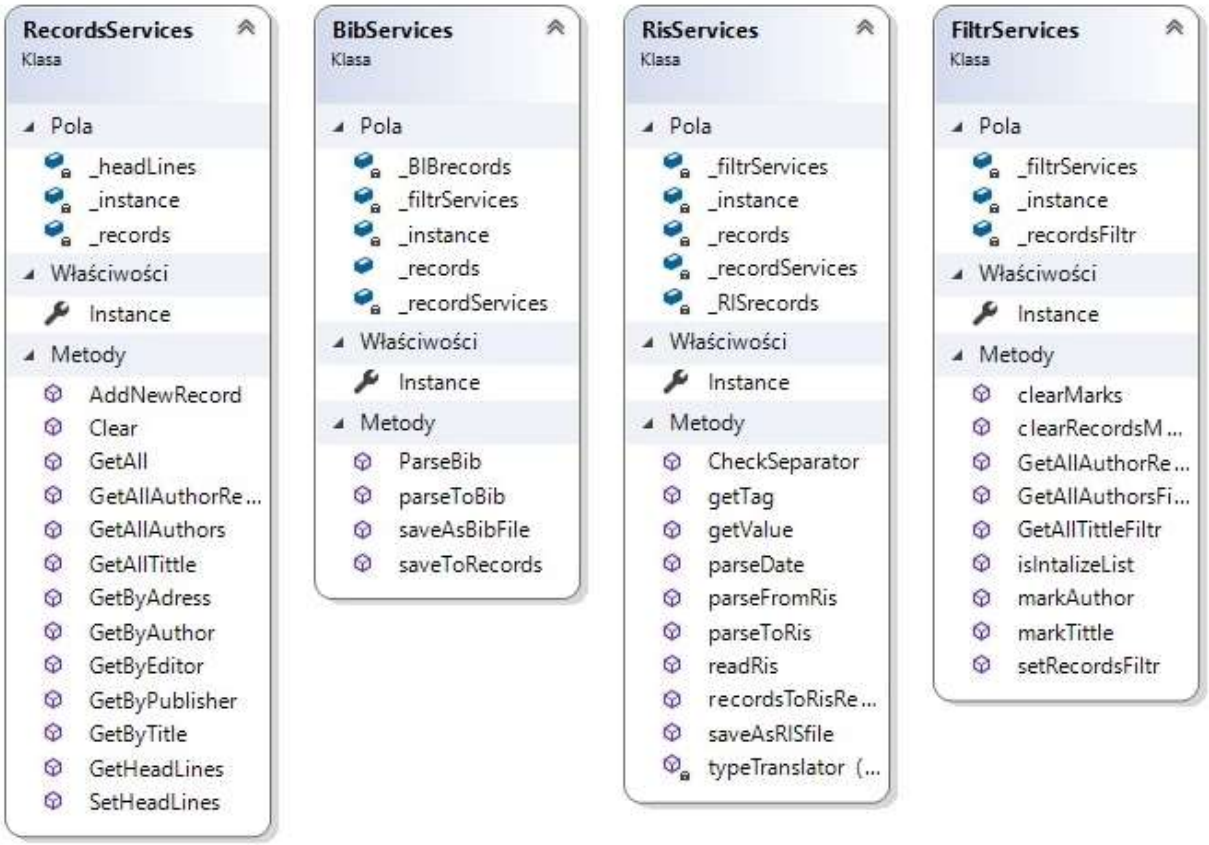
Wszystkie nie opisane wyżej pliki a znajdujące się w tym folderze to widoki wywoływane przez poszczególne kontrolery widoków.

Kontrolery:



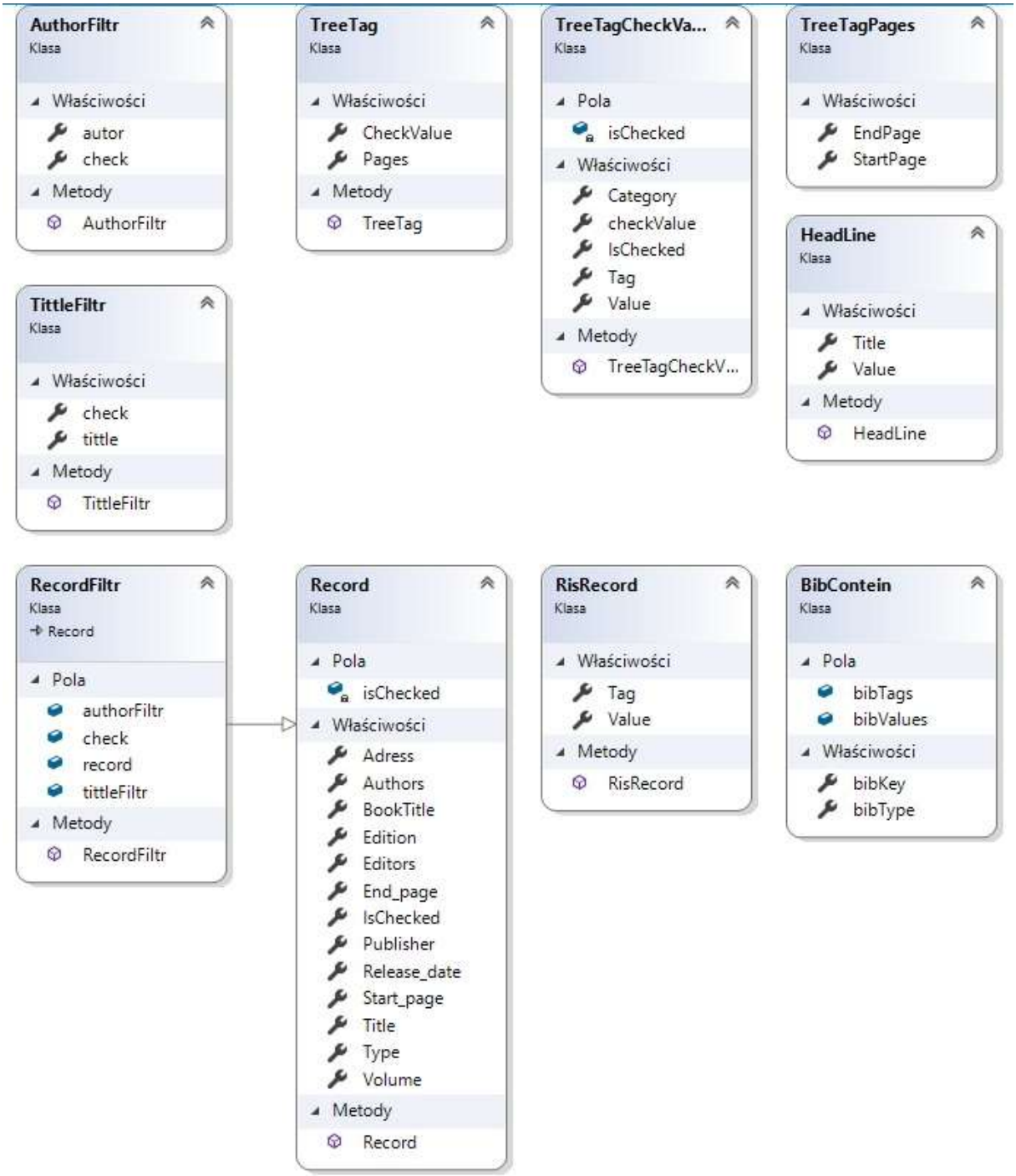
Kontroler jest to część aplikacji która komunikuje się z widokiem w ASP .NET są dwa typy kontrolerów te które dziedziczą po klasie „Controller” są to kontrolery typowe dla ASP .NET MVC (w naszej aplikacji to kontrolery home, tree i help) i różnią się od drugiego typu tym, że w odpowiedzi na zapytanie http mogą zwrócić nie tylko dane (np. w postaci JSON lub XML) ale też widok. Drugi typ kontrolerów dziedziczy po klasie „ApiController” są to kontrolery uniwersalne i mogą być używane nie tylko w ASP .NET MVC ale też przy ASP .NET WebApi te kontrolery nie mogą zwracać widoków a jedynie kody odpowiedzi http i dane.

Serwisy:



W celu zapewnienia przejrzystości kodu oraz możliwości używania metod w kilku kontrolerach bez duplikowania ich biorąc przykład z projektów komercyjnych stworzyliśmy serwisy i przenieśliśmy do nich większość logiki standardowo zawartej w kontrolerach i nazwaliśmy tak żebyśmy łatwo mogli zidentyfikować co robią (np. RisServices odpowiada za odczytanie danych z pliku ris i zapisanie ich w postaci listy obiektów klasy Record i w drugą stronę).

Modele:



Model jest to klasa która odpowiada za zamodelowanie rzeczywistości i zapewnienie odpowiedniego formatu danych. To właśnie na obiektach klas typu model operuje cała aplikacja.
Klasa Record to klasa w dla pojedynczego rekordu w tabelce którą wyświetlamy użytkownikowi.
Klasa RisRecord to klasa dla jednego elementu danych w pliku RIS (kończącego się znacznikiem ER)

Klasa BibContein to klasa dla danych z pliku BibTex zawiera dwie listy w których przechowuje znaczniki i ich wartości.
Klasa HeadLine to klasa dla nagłówków tabeli wyświetlanej użytkownikowi.
Klasy zawierające w nazwie słowo „Filtr" odpowiadają za filtrowanie danych.
Klasy zawierające w nazwie słowo „Tag" odpowiadają za prezentację opcji filtrowania.

Jak działa aplikacja:

W skrócie po wybraniu pliku do translacji i kliknięciu przycisku od Uploadu plik wysyłany jest do HomeController gdzie rozpoznawane jest jego rozszerzenie (akceptowane są tylko pliki z rozszerzeniem .bib lub .ris) a następnie plik zostaje odczytany przez odpowiedni serwis (RisServices lub BibServies). Dane zawarte w pliku zapisywane są do obiektów klasy Record które zostają dodane do odpowiedniej listy. Główny widok oraz widok w zakładce Podgląd zawierają identyczne tabelki w których te dane z listy się wyświetlają są one uaktualniane dynamicznie dzięki aplikacji napisanej w AngularJS kontrolującej tabelkę łączy się ona z RecordControllerem z którego odczytuje dane po każdej zmianie. Po wybraniu opcji filtrowania lista ta jest ograniczana przez co pokazywane są tylko odpowiednie a wszystkie dane zachowywane są w liście tymczasowej. Natomiast wyszukiwanie i sortowanie działa tylko na tej wyświetlanej tabelce za pomocą AngularJS nie ograniczają one listy. Zdecydowaliśmy się na różne sposoby filtracji danych ze względu na to, że skoro i tak używaliśmy AngularaJS to chcieliśmy pokazać chociaż troszkę jego możliwości, ale obydwaj specjalizujemy się raczej w technologii .NET i programowania po stronie klienta używamy raczej jako dodatków to chcieliśmy pokazać że tego typu rzeczy można robić też bez Angulara w dodatku to rozwiązanie jest pewniejsze ponieważ jest znacznie bardziej odporne na próby modyfikacji wyświetlanej zawartości przez użytkownika końcowego (za pomocą konsoli w przeglądarce).

5. Podsumowanie

Program został stworzony w sposób stawiający na prostotę i wygodę użytkowania, najprawdopodobniej nie jest to program pozbawiony błędów, ale uważamy że spełnia swoje zadanie i nie powinien sprawiać problemów użytkownikom. Wybrane przeze nas technologie i sposoby rozwiązywania problemów miały jak najbardziej ułatwić pracę programistą dzięki czemu

program mógł zostać napisany w sposób prosty i dający się zrozumieć nawet przez początkujących programistów.