**Dokumentacja**

Link do repozytorium: https://github.com/jozek24/Pzpp-projekt1.git

**1. Zespół:**

Józef Salik:

* Dokumentacja
* Testy w aplikacji MediaReviewer
* Aplikacja „MediaReviewer” folder ‘Model’:
  + Pobieranie danych z bazy
  + Wyjmowanie samego tekstu artykułu z pliku HTML

Konrad Garlacz:

* Aplikacja „MediaReviewer” - View oraz ViewModel

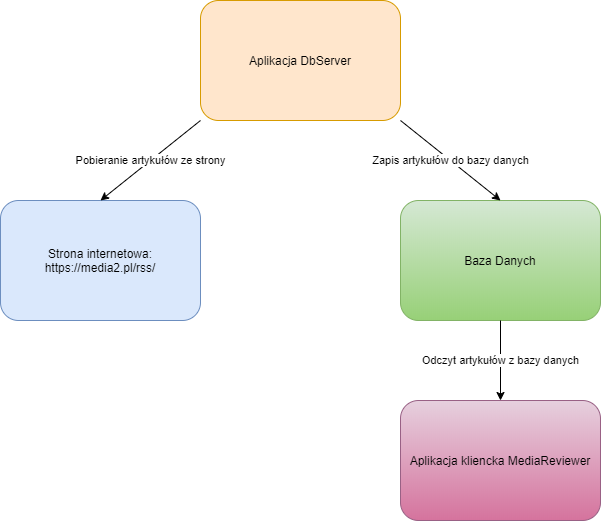
Tomasz Zając:

* Aplikacja „DbServer”:
  + Pobieranie danych XML z internetu
  + Serializacja na JSON oraz odpowiednie klasy
  + Testy

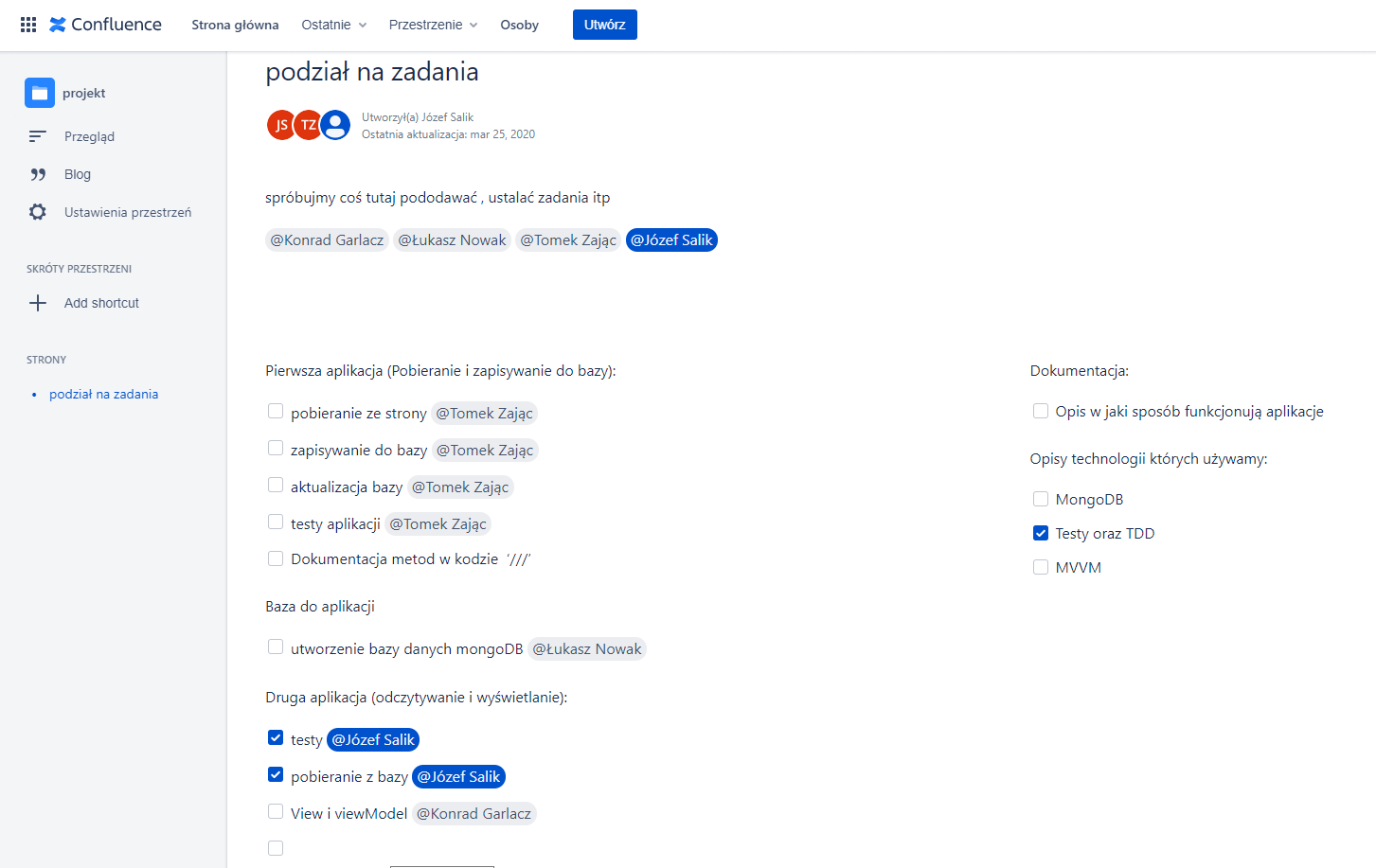
Łukasz Nowak:

* Zapis danych do bazy - Klasy:
  + MongoCRUD – komunikacja z bazą danych (połączenie z bazą, zapis oraz odczyt)
  + Repository - Metody odpowiedzialne za sprawdzeniie oraz dodawanie co 5min kanałów i artykułów, których nie ma jeszcze w bazie

Ogólny schemat komunikacji pomiędzy aplikacjami:



W celu ułatwienia współpracy wykorzystaliśmy narzędzie jakim jest Confluence (*rysunek 1*). Komunikowaliśmy się również przy pomocy narzędzia Microsoft Teams. Mogliśmy tam np. dodawać zadania, bądź opisywać zauważone problemy do rozwiązania.



Rysunek 1 Widok strony Confluence

**Aplikacja DbServer**

Tworząc obiekt klasy Repository w konstruktorze podajemy 2 parametry. Pierwszy to nazwa bazy dancych. Drugi to interwał czasu w sekundach, w jakim aplikacja będzie sprawdzała czy nie zostały dodane jakieś nowe kanały lub artykuły na stronie media2.pl/rss.

*static void Main(string[] args)*

*{*

*Repository repository = new Repository("ChannelsFromMedia2",300);*

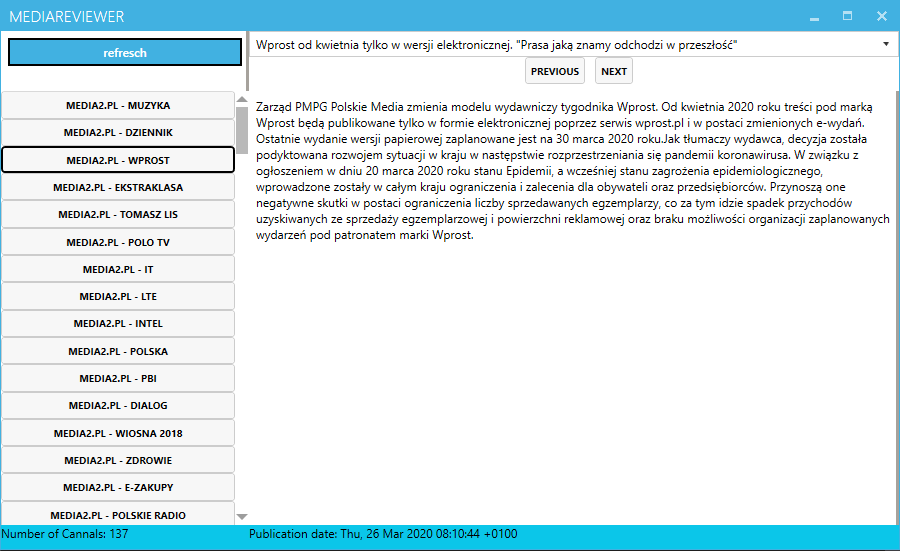
*repository.CheckNewData();*

*Console.ReadLine();*

*}*

**Wygląd aplikacji Media Reviewer**

Domyślny wygląd w aplikacji został zmieniony za pomocą biblioteki mahapps.metro (*rysunek 2*).



Rysunek 2 Wygląd aplikacji Media Reviewer

**2. Technologie:**

Do wyciągnięcia samego tekstu artykułu z całego pliku HTML wykorzystano bibliotekę „Html Agility Pack”.

**Wzorzec architektoniczny MVVM**

Aplikacja Media Reviewer została napisana według wzorca MVVM. Do realizacji tego celu wykorzystano framework Prism.

Dzięki zastosowaniu tego wzorca możemy oddzielić warstwę widoku (View) od modelu biznesowego (Model). Częścią, która jest odpowiedzialna za komunikację między View oraz Model jest ViewModel.

Stworzona aplikacja wykorzystuje bazę danych opartą o MongoDB.

**MongoDB** – jest to nierelacyjny system zarządzania bazą danych. Dane w nim są składowane jako dokumenty zapisane w stylu JSON.

Strukturę jednego dokumentu w bazie przedstawiono na *rysunku 3*.



Rysunek 3 Dokument przechowujący artykuły z jednego kanału

Opis komunikacji aplikacji z bazą danych:

Aby aplikacja mogła komunikować się z bazą danych należało pobrać do projektu bibliotekę Mongo .NET driver.

Sposób połączenia z bazą przedstawiono w *listingu 1*. Metoda GetDatabase klasy MongoClient() łączy się z bazą, której nazwę podajemy w pierwszym parametrze. Jeżeli taka baza jeszcze nie istnieje, to zostanie ona utworzona.

*var client = new MongoClient();*

*db = client.GetDatabase(databaseName);*

*Listing 1: Połączenie z bazą*

Dodawanie do bazy listy obiektów wykonuje metoda przedstawiona w *listingu 2*.

*public void InsertRecords<T>(string table, List<T> record)*

*{*

*var collection = db.GetCollection<T>(table);*

*collection.InsertMany(record);*

*}*

*Listing 2: Metoda InsertRecord<T>()*

Pobieranie z bazy listy obiektów dokonuje metoda przedstawiona w *listingu 3*.

*public List<T> LoadRecords<T>(string table)*

*{*

*var collection = db.GetCollection<T>(table);*

*return collection.Find(new BsonDocument()).ToList();*

*}*

*Listing 3: Metoda LoadRecords<T>()*

**Testy oraz podejście TDD**

W projekcje testy zostały napisane z wykorzystaniem frameworka Nunit (w aplikacji MediaReviewer) oraz xUnit (w aplikacji DbServer) oraz podejścia TDD.

Podejście **Test Driven Development (TDD)** – polega na tworzeniu oprogramowania w taki sposób, że tworzenie nowej funkcjonalności zaczynamy od utworzenia testów.

Podejście to składa się z cykli, które składają się z trzech faz:

1. Red

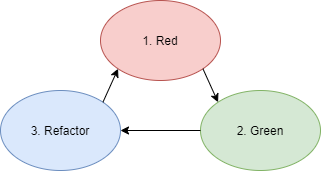
2. Green

3. Refactor

Faza **Red** – Piszemy testy dla jeszcze nie zaimplementowanej metody. W tej fazie należy określić co metoda dokładnie ma robić/zwracać.

Faza **Green** – Implementujemy metody tak aby testy się powiodły.

Faza **Refactor** – Wprowadzamy zmiany w kodzie, aby był czytelniejszy, nie zmieniając jego funkcjonalności. Sprawdzamy czy testy się powiodły.



**Opis użycia ‘dependency injection’ oraz frameworka ‘Moq’ w aplikacji ‘MediaReviewer’**

W celu testowania metod, które korzystają z bazy danych należy stworzyć interface (*listing 4*).

*public interface IArticlesStorage*

*{*

*List<T> LoadRecords<T>(string table);*

*}*

*Listing 4: Interface IArticlesStorage*

Stworzyć klasę komunikującą się z bazą danych. Dziedziczy ona oraz implementuje stworzony wcześniej interface (*listing 5*).

*public class ArticlesStorage : IArticlesStorage*

*{*

*private IMongoDatabase db;*

*public ArticlesStorage(string databaseName)*

*{*

*var client = new MongoClient();*

*db = client.GetDatabase(databaseName);*

*}*

*public List<T> LoadRecords<T>(string table)*

*{*

*var collection = db.GetCollection<T>(table);*

*return collection.Find(new BsonDocument()).ToList();*

*}*

*}*

*Listing 5: Klasia ArticlesStorage*

W osobnej klasie metoda umożliwiająca podmianę klasy implementującej interface IArticlesStorage (*listing 6*).

*public class ArticlesHelper*

*{*

*private readonly IArticlesStorage \_articlesStorage;*

*public ArticlesHelper(string databaseName, IArticlesStorage articlesStorage = null)*

*{*

*\_articlesStorage = articlesStorage ?? new ArticlesStorage(databaseName);*

*}*

*public List<RssChannel> GetChannels()*

*{*

*var rssChannel = \_articlesStorage.LoadRecords<RssChannel>("RssChannel");*

*return rssChannel;*

*}*

*}*

*Listing 6: Klasa ArticlesHelper*

Przy tak przygotowanej konstrukcji klas możemy dla realizacji testów imitować obiekt klasy ArticlesStorage. Można to zrobić korzystając z framework Moq.

Tworzenie takiego obiektu przedstawiono w *listingu 7*.

*\_storage = new Mock<IArticlesStorage>();*

*\_storage.Setup(s => s.LoadRecords<RssChannel>("RssChannel"))*

*.Returns(new List<RssChannel>*

*{new RssChannel()*

*{*

*Articles = new List<Article>()*

*{*

*new Article()*

*{*

*Title = "a"*

*},*

*new Article()*

*{*

*Title = "b"*

*}*

*},*

*}*

*});*

*Listing 3 Sposób towrzenia obiektu Mock*