19.06.2020

Laboratorium Programowania Komputerów

Temat projektu: Klient Wordpressa

Język projektu: C#

Paweł Seweryn

grupa 4

semestr 4

1. Temat

Założeniem projektu było stworzenie programu umożliwiającego osobom nieobeznanym łatwą i szybką publikację treści na stronie internetowej opartej na systemie Wordpress. Użytkownik może uruchomić aplikację z poziomu pulpitu i w prosty sposób przejrzeć oraz zmodyfikować treści z serwisu YouTube, które mają zostać opublikowane, oraz te, które już są na stronie.

2. Analiza

System zarządzania treścią jakim jest Wordpress oparty jest na relacyjnej bazie danych MySQL, w której zawarte są dane i relacje wszystkich wpisów. Do korzystania z niej został wykorzystany framework MySql, natomiast integrację z serwisem YouTube zapewnił framework YouTube API w wersji trzeciej. Jednym z pomysłów było wykorzystanie frameworku Entity, jednak z uwagi na charakterystykę projektu został on zaniechany.

3. Projektowanie

Wybierając odpowiedni interfejs graficzny postawiono na Windows Forms umożliwiającey natywny dostęp do elementów interfejsu graficznego systemu Windows. Struktura programu zbliżona jest do wzorca MVP (model-view-presenter).

Opis poszczególnych klas oraz ich zastosowanie znajduje się w poniższej tabeli.

YoutubeEntry	Zawiera podstawowe informacje o poście takie jak numer, data publikacji, tytuł, opis i miniatura.
BlogEntry	Dziedziczy po YoutubeEntry, zawiera listę tagów, id wpisu oraz dane niezbędne do umieszczenia w tabeli postów na serwerze; implementuje interfejs IComparable.
Tag	Zawiera podstawowe informacje tagu: nazwę, id rodzica oraz własne id; dziedziczy po TreeNode.

Klasy pełniące rolę kontenerów i manipulatorów danych

EntryPresenter	Jest zdefiniowaną kontrolką użytkownika (dziedziczy po UserControl); odpowiedzialna za wyświetlanie wpisu oraz informacji czy jest opublikowany.
EntryListControl	Zawiera listę obiektów klasy EntryPresenter i również dziedziczy po UserControl; wyświetla wpisy z listy w „płynnym” panelu.

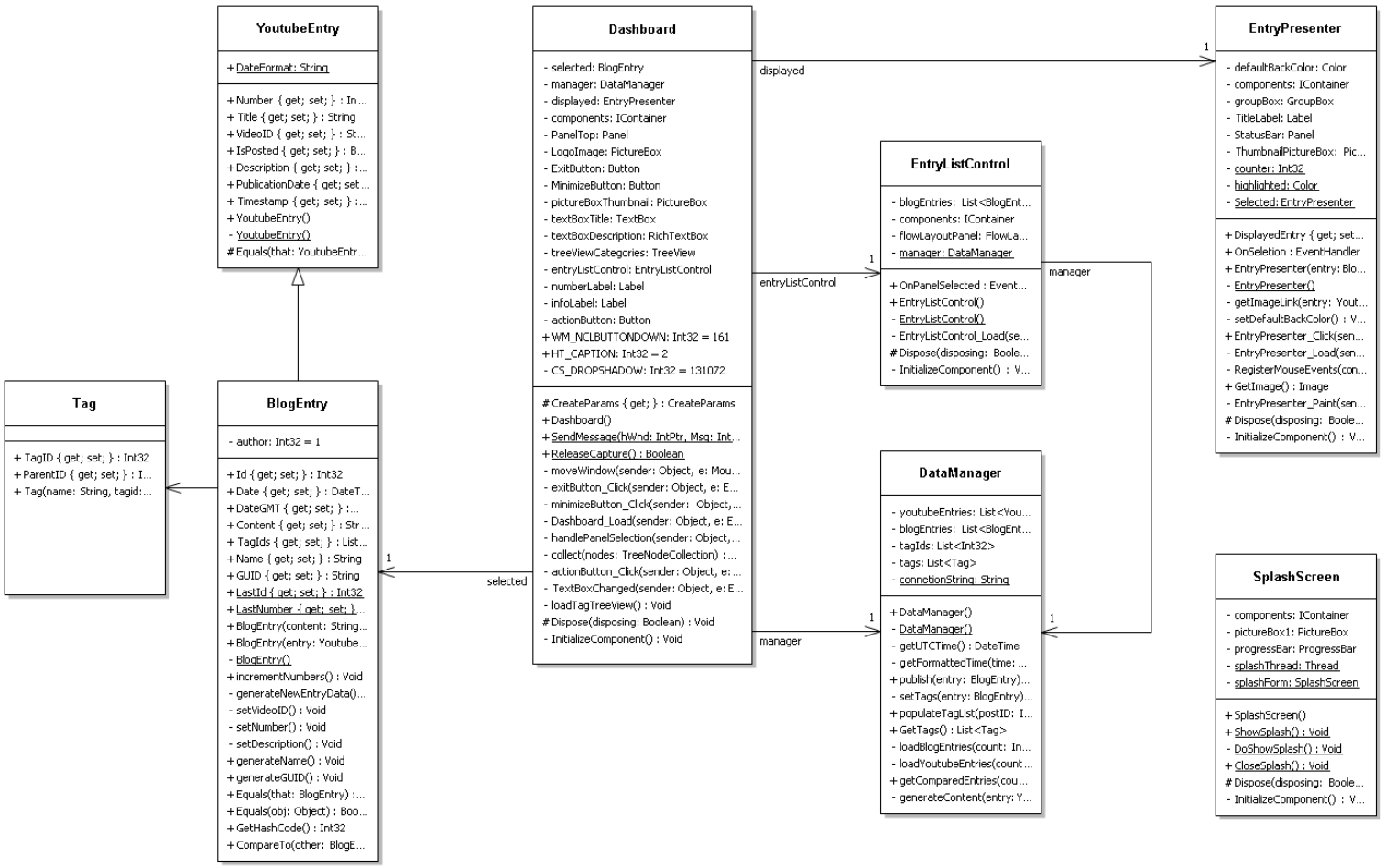
Te klasy są odpowiedzialne za wyświetlanie danych

Dashboard	Tutaj tworzone jest okno główne programu, organizowany dynamicznie layout kontrolek, obsługiwane wydarzenia takie jak kliknięcia paneli i przycisków; dziedziczy po Form.
SplashScreen	Klasa dziedziczy po Form; wyświetla ekran ładowania podczas gdy wątki w tle pobierają dane z bazy i ładują okno główne.

W tych klasach tworzone są okna, obsługiwane wątki oraz akcje

DataManager	Odpowiada za połączenie z bazą danych Wordpressa i Youtube'a; ładuje i wysyła listy wpisów oraz tagi.
Program	Punkt rozpoczęcia głównego wątku programu, obsługa skalowania aplikacji.

4. Diagram klas



5. Kluczowe metody

```
public BlogEntry(YoutubeEntry entry)
{
      foreach (PropertyInfo prop in entry.GetType().GetProperties())
                GetType().GetProperty(prop.Name).SetValue(this,
prop.GetValue(entry, null), null);
//...
}
```

Konstruktor klasy BlogEntry przyjmujący jako argument typ YoutubeEntry. Wykorzystane tutaj zostało odbicie, aby z obiektu podstawowego dynamicznie utworzyć obiekt pochodny.

```
public void generateName()
{
      string decomposed = Title.Normalize(NormalizationForm.FormD);
      char[] filtered = decomposed
          .Where(c => char.GetUnicodeCategory(c) !=
UnicodeCategory.NonSpacingMark)
          .ToArray();
      string input = new string(filtered);
      Regex rgx = new Regex("[^a-zA-Z0-9 -]");
      Name = rgx.Replace(input, "").Replace(' ', '-').ToLower();
}
```

Metoda klasy BlogEntry, generuje nazwę posta filtrując tytuł w taki sposób, aby można było z niego utworzyć adres URL.

```
private IEnumerable<TreeNode> collect(TreeNodeCollection nodes)
{
    foreach (TreeNode node in nodes)
    {
        yield return node;

        foreach (var child in collect(node.Nodes))
        yield return child;
    }
}
```

Funkcja iterująca w klasie Dashboard, umożliwia przeiterowanie po każdym elemencie podanego w argumencie drzewa.

```
private void handlePanelSelection(object sender, EventArgs e)
{
       Control control = sender as Control;
       while (control.GetType() != typeof(EntryPresenter))
             control = control.Parent;
       if (control.GetType() == typeof(EntryPresenter))
       {
             displayed = control as EntryPresenter;
             selected = displayed.DisplayedEntry;
       }
}
```

Fragment metody z klasy Dashboard; obsługuje wybrany wpis poprzez wyszukanie rodzica klikniętej kontrolki.

```
private void loadTagTreeView()
{
    List<Tag> tags = manager.GetTags();

    foreach (var t1 in tags)
    {
        if (t1.ParentID == 0)
        {
            treeViewCategories.Nodes.Add(t1);
            foreach (var t2 in tags)
            {
                if (t2.ParentID == t1.TagID)
                {
                    t1.Nodes.Add(t2);
                    t1.Expand();
                }
            }
        }
    }
}
```

Metoda z klasy Dashboard, ładuje drzewo tagów poprzez porównania id.

```
public List<BlogEntry> getComparedEntries(int count)
{
    loadBlogEntries(count);
    loadYoutubeEntries(count);
    HashSet<BlogEntry> entrySet = new HashSet<BlogEntry>();
    foreach(var entry in blogEntries)
    {
        entrySet.Add(entry);
    }
    foreach(var entry in youtubeEntries)
    {
        entrySet.Add(new BlogEntry(entry));
    }
    var entryList = entrySet.OrderBy(e => e.Date).ToList();
    foreach(var e in entryList)
    {
        if (!e.IsPosted)
        {
            e.incrementNumbers();
            e.Content = generateContent(e);
        }
    }
    entryList.Sort();
    return entryList;
}
```

Jedna z najważniejszych metod w klasie DataManager. Zwraca listę porównanych wpisów. Wykorzystany HashSet eliminuje duplikaty, jednak jego użycie wymagało przeciążenie funkcji hashującej. Kolejność wpisów zachowana jest dzięki uporządkowaniu list po dacie. Uzupełniając zestaw danych generowana jest zawartość przekonwertowanego wcześniej wpisu YoutubeEntry oraz ustawiane są jego id i numer metodą `incrementNumbers()`, która ma dostęp do statycznych pól w klasie BlogEntry przechowujących informacje o numerach najnowszego postu z bazy danych.

```
private void setTags(BlogEntry entry)
{
    using (var connection = new MySqlConnection(connetionString))
    using (var command = connection.CreateCommand())
    {
        connection.Open();
        foreach(int tag in entry.TagIds)
        {
            command.CommandText = $"INSERT INTO cb_term_relationships (object_id,
term_taxonomy_id, term_order) VALUES ({entry.Id}, {tag}, 0)";
            command.ExecuteNonQuery();
        }
    }
}
```

Metoda z klasy DataManager obsługuje wstawienie do tabeli danych dotyczących tagów.

6. Interfejs graficzny

Po uruchomieniu programu wyświetla się ekran oczekiwania, który znika po załadowaniu ekranu głównego. Użytkownik wybiera film z listy, zaznacza kategorie i po kliknięciu przycisku *Opublikuj* pasek obok miniatury zmienia kolor na zielony, a post jest opublikowany. Brak zaznaczenia jakiejkolwiek kategorii skutkuje wyświetleniem odpowiedniego komunikatu.

7. Testowanie

Projekt podczas powstawania był testowany na bieżąco. Zdarzało się czasem, że jakaś funkcja nie działała poprawnie – wówczas podczas sesji debugowania ustawiany był punkt przerwania w newralgicznym miejscu, a do konsoli wypisywane były potencjalnie winne dane. Czasem niewłaściwie napisana funkcja powodowała wyrzucenie nieobsługiwanego wyjątku co w konsekwencji przerywało działanie programu, a wtedy środowisko zwracało ślad stosu i kod błędu, którego opis definiował przyczynę przerwania.

8. Przemyślenia końcowe

Realizacja projektu pochłonęła znacznie więcej czasu i energii niż początkowo zakładałem. Mając doświadczenie w Javie podszedłem do realizacji z myślą, ze semantyka języka C# – którego nie znałem wcześniej – jest podobna, więc nie powinno być problemów. Nic bardziej mylnego. Sporym wyzwaniem było dla mnie zaprojektowanie klas – początkowo podstawową klasą bazową miała być abstrakcyjna klasa Entry, a dziedziczyć po niej miały BlogEntry i YoutubeEntry. Nie miało to sensu, bo Entry od YoutubeEntry niewiele się różniły. To dało mi do myślenia jak istotne jest dokładne przeanalizowanie problemu i zaprojektowanie klas od podstaw. Część pracy niestety nie została wykorzystana – chcąc zabezpieczyć wrażliwe dane (np. connection strings) i aplikację przed niepowołanym dostępem zaprojektowałem okno logowania i klasę szyfrującą. Pomysł niestety spalił na panewce. Mimo wszystko jestem zadowolony z projektu i mam zamiar go dalej rozwijać.