**Zarządzanie projektem informatycznym**

|  |  |  |  |
| --- | --- | --- | --- |
| **Kierunek studiów:** | *Informatyka* | **Rok studiów:** | *III* |
| **Numer grupy:** | *L5* | | |
| **Rok akademicki:** | *2014/2015* | **Semestr:** | *VI* |

**Sprawozdanie z wykonania projektu:**

**System zarządzania treścią dla szkół**

|  |  |  |
| --- | --- | --- |
| **Lp.** | **Nr indeksu** | **Imię i nazwisko** |
| 1. | 86316 | Radosław Szymański |
| 2. | 84135 | Damian Szymański |
| 3. | 84139 | Kamil Ślusarczyk |
| 4. | 84123 | Artur Stelmach |

|  |  |  |
| --- | --- | --- |
| **Termin zajęć:** | | **Prowadzący:** |
| dzień: | *Środa* | dr inż. Anna Zatwarnicka |
| godzina: | *12:50* |

## Główne założenia projektu

### Idea

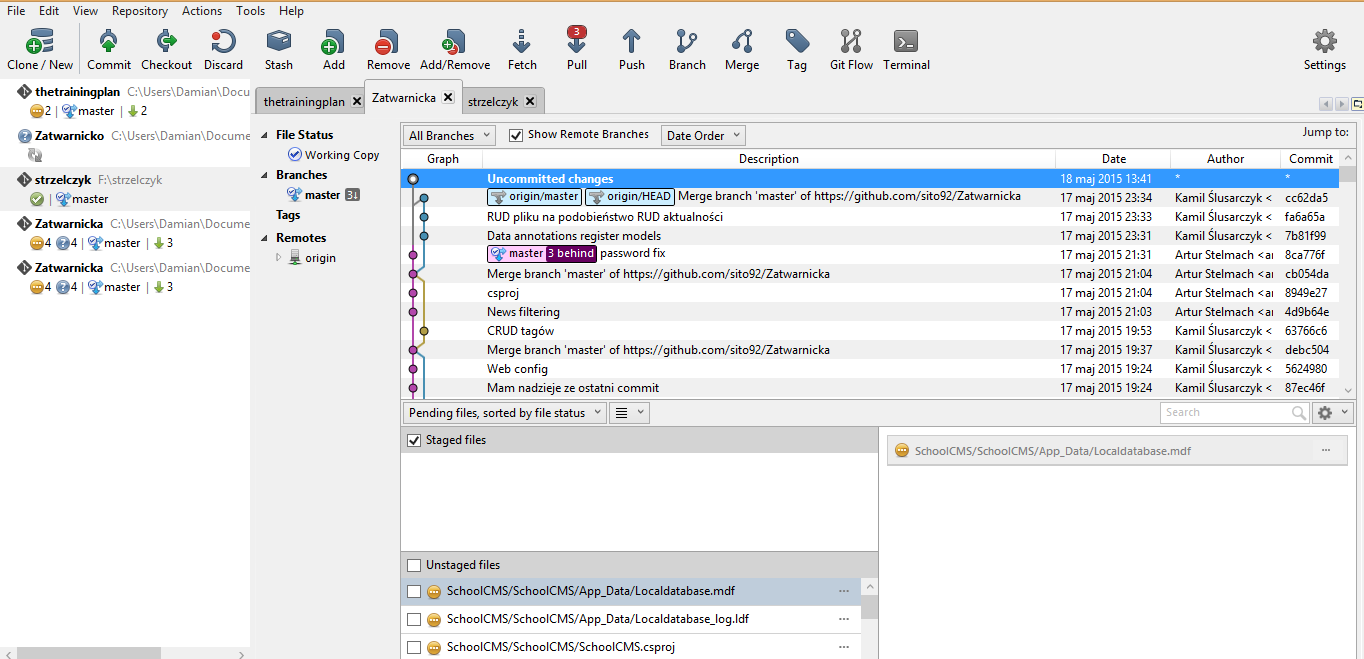
Nasza grupa postanowiła podjąć się wykonania systemu zarządzania treścią przeznaczonego dla szkolnych stron internetowych. Jednym z głównych argumentów, które przesądziły o tej decyzji było to, że niejednokrotnie, jak pokazuje rynek w ostatnich latach, potencjał komercyjny aplikacji zależy od prostoty systemu. Chcieliśmy więc wykonać system, który bez uszczuplania możliwych funkcjonalności będzie maksymalnie łatwy obsłudze, tak by grono potencjalnych odbiorców było jak największe.

### Podział ról i organizacja pracy

Team managerem z uwagi na cechy charakteru został Artur Stelmach. Jego zastępcą został mianowany Kamil Ślusarczyk.

Przy podziale ról i zadań kierowaliśmy się jedynie równomiernym podziałem pracy. Powodem tego jest podobny profil zainteresowań każdego z nas, który jest skierowany na programowanie w środowisku .Net.

Dla łatwiejszego zarządzania projektem skorzystaliśmy z rozproszonego systemu kontroli wersji Git, a do jego obsługi aplikacji SourceTree. Oprócz tego poszczególne zadania („taski”) rozpisywaliśmy w internetowej aplikacji Producteev, która systematyzowała naszą prace. Wybór tych narzędzi był podyktowany wcześniejszymi doświadczeniami pracy w zespołowych projektach informatycznych. Stworzyliśmy także diagram Gantta, który z uwagi na dużą rozdzielczość **został załączony w załączniku nr 1 do sprawozdania**.



Rys.1. SourceTree

### Wybór narzędzi

Z uwagi na wcześniejsze doświadczenia i zainteresowania aplikację postanowiliśmy napisać w środowisku .Net. Wybraliśmy platformę aplikacyjną do budowy aplikacji internetowych opartych na wzorcu Model-View-Controller (MVC) ASP.NET MVC. Jest to środowisko i model, który umożliwia szybką pracę, jednocześnie nie zamykający możliwości programiście.

Użyliśmy również Enity Framework, który jest narzędziem typu ORM (Object Relational Mapping), pozwalającym odwzorować relacyjną bazę danych za pomocą architektury obiektowej. Skorzystaliśmy również z wzorca Dependency Injection poprzez jego implementację - Ninject.

# Wymagania funkcjonalne

Poniżej opisujemy wymagania funkcjonalne, których wykonanie założyliśmy.

1. Dodawanie wzorców stron do menu
2. Logowanie i wylogowywanie administratora i redaktorów.
3. Przypomnienie hasła dla administratora i redaktorów.
4. Tworzenie kont redaktorów przez administratorów.
5. CRUD aktualności dla administratora.
6. CRUD aktualności dla redaktorów.
7. Możliwość budowania menu z podstronami

Jest to funkcjonalność, która ma umożliwić użytkownikom systemu budowanie wielopoziomowych, zagnieżdżonych struktur tworzonej w naszym systemie CMS strony internetowej.

1. Dobieranie szablonów graficznych
2. Archiwum aktualności
3. Filtrowanie treści (tagi)
4. Możliwość dołączania załączników do podstron
5. E-dziennik
6. Możliwość dodawania materiałów dydaktycznych dla konkretnych klas i zajęć

# 3. Wymagania niefunkcjonalne

Poniżej wymieniamy wymagania niefunkcjonalne:

* Możliwość wyświetlania strony na najpopularniejszych przeglądarkach: Chrome, Mozzila Firefox, Opera i Internet Explorer
* Zapewnienie ochrony kont użytkowników serwisu poprzez autentykację za pomocą hasła
* Umożliwienie personelowi nie technicznemu tworzenie własnych wersji serwisu
* Możliwość przechowywania wielu różnych plików: pdf, jpg ,itp.

# 4. Szacowanie kosztów

Szacowanie kosztów składa się z następujących punktów:

• koszt sprzętu będącego częścią tworzonego systemu

• koszt zakupu narzędzi

• koszt wyjazdów i szkoleń

• nakład pracy

Pierwsze 3 punkty nie dotyczą naszego projektu. Koszt sprzętu, dlatego iż zarówno programiści jak i klienci będą wykorzystywać wcześniej zakupiony sprzęt. Koszt zakupu narzędzi również nas nie dotyczy, ponieważ środowisko programistyczne i całe potrzebne oprogramowanie jest już w naszym posiadaniu. Jedynym kosztem wyjazdu jest koszt wyjazdu na szkolenie klienta, jednak jest to wartość pomijalna(czas które poświęcimy na to dodamy podczas szacowanie nakładu pracy).

Nakład pracy został przez nas oszacowany poprzez analogię. Zdecydowaliśmy się na tę metodę szacowaniu czasu pracy, ponieważ jest to kolejny projekt, który wykonujemy w tym samym składzie(lub prawie tym samym składzie). Szacowany czas wyniósł: 68 dni. Szczegółową rozpiskę poszczególnych zadań zamieszczamy w diagramie Gantt’a, który umieściliśmy w osobnym załączniku.

Średnio każdego dnia będziemy pracować 1 godzinę nad projektem. Mnożąc to przez ilość osób w projekcie i ilość dni, otrzymujemy 272 roboczogodziny.

# 5. Diagram baz danych

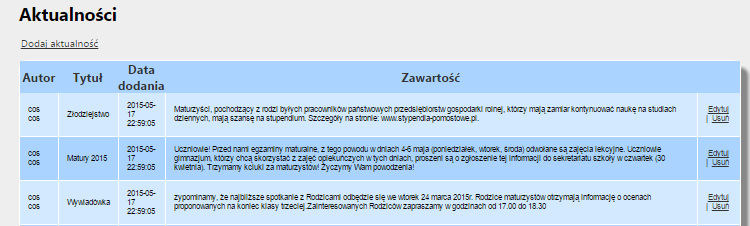
Zanim mogliśmy rozpocząć pracę stricte implementacyjną stworzyliśmy diagram relacyjnej baz danych. Skorzystaliśmy tutaj z program MS Office Visio. Z uwagi na dużą rozdzielczość **został on załączony w załączniku nr 2 do sprawozdania**.

# 6. Zaimplementowane funkcjonalności

***Omawianie wszystkich funkcjonalności systemu zaczniemy od panelu administratora.***

### 6.1. Aktualności

Administrator ma możliwość dodawania, edycji i usuwania dowolnych aktualności.



Rys.2. Zarządzanie aktualnościami

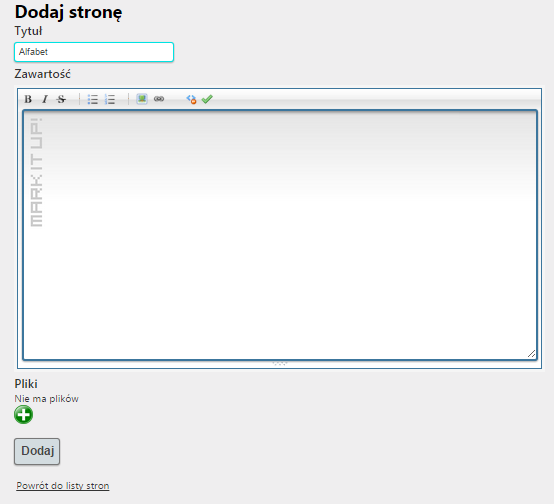


|  |
| --- |
| [Authorize]  [HttpPost]  public ActionResult Add(NewsEdit model,IEnumerable<int> filesToAdd, IEnumerable<int> filesToRemove)  {  if (model.SelectedTags==null || !model.SelectedTags.Any())  {  ModelState.AddModelError(string.Empty,"News musi mieć chociaż jeden tag");  model.Tags = new SelectList(context.Tags, "Id", "Name");  PopulateFiles();  }  if (ModelState.IsValid)  {  var author = context.Users.FirstOrDefault(x => x.Username == WebSecurity.CurrentUserName);  model.News.AuthorId = author.Id;  model.News.Date = DateTime.Now;  model.News.Tags = context.Tags.Where(x => model.SelectedTags.Contains(x.Id)).ToList();  model.News.ManageFiles(filesToRemove, filesToAdd,context);  context.InformationSources.Add(model.News);  context.SaveChanges();  return RedirectToAction("List", "News"); } |

### Listing. 1. Metoda Add(dodawanie stron)

### 6.2. Strony

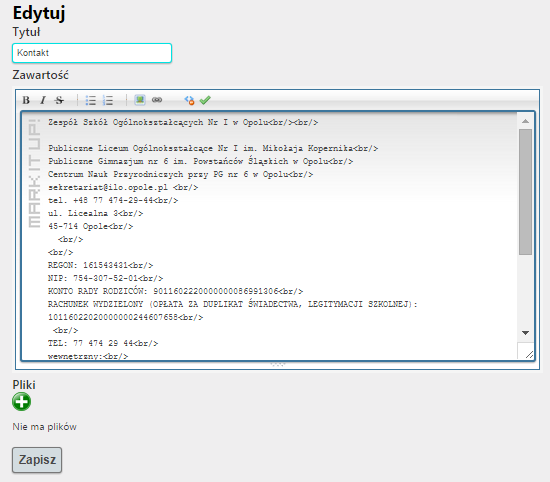
Administrator może dodawać, edytować i usuwać strony. Strony można podpinać pod wybrane menu.



Rys.4. Dodawanie stron

|  |
| --- |
| public ActionResult Create(Page page, IEnumerable<int> filesToAdd, IEnumerable<int> filesToRemove)  {    if (ModelState.IsValid)  {  var author = context.Users.FirstOrDefault(x => x.Username == WebSecurity.CurrentUserName);  page.AuthorId = author.Id;  page.Date = DateTime.Now;  page.ManageFiles(filesToRemove, filesToAdd, context);  context.InformationSources.Add(page);  context.SaveChanges();  return RedirectToAction("List");  }  return View(page);  } |

Listing.2. Tworzenie strony



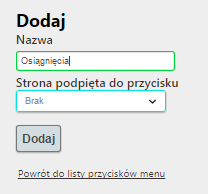
Rys.5. Edytowanie stron

|  |
| --- |
| public ActionResult Edit(Page page, IEnumerable<int> filesToAdd, IEnumerable<int> filesToRemove)  {  if (!ModelState.IsValid)  {  return View(page);  }  var prePage = context.InformationSources.OfType<Page>().FirstOrDefault(x => x.Id == page.Id);  if (prePage == null)  {  return HttpNotFound();  }  prePage.ManageFiles(filesToRemove, filesToAdd,context);  prePage.Content = page.Content;  prePage.Title = page.Title;  context.SaveChanges();  return RedirectToAction("List");  } |

Listing.3. Edytowanie strony

### 6.3. Tworzenie menu

Jest to jedna z głównych funkcjonalności systemu. Pozwala ona w łatwy sposób budować rozgałęziona strukturę strony. Procedura tworzenia menu została pokazana na poniższych rysunkach, zawartych w punkcie 2.6.



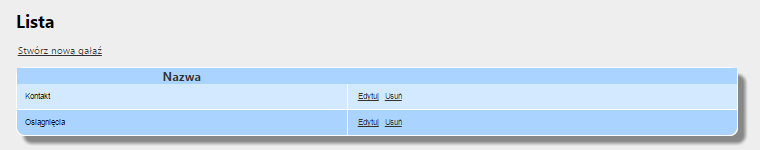
Rys.6. Dodawanie menu

Nowo utworzone menu pojawia się w prawym górnym rogu.



Rys.7. Nowo utworzone menu

Następnie możemy przejść do edycji utworzonego menu poprzez listę menu:



Rys.8. Lista menu

Następnie możemy dodać przycisk menu:



Rys.9. Dodanie przycisku menu

Stworzyliśmy rozgałęzioną strukturę. W prawym górnym menu mamy rozwijalne menu, które możemy dowolnie edytować, budując wielopoziomowe zagnieżdżenia. Dzięki temu nawet osoba bez informatycznego doświadczenia może stworzyć stronę, która w pełni wyczerpie jego oczekiwania.



Rys.10. Utworzenie rozgałęzionej struktury

|  |
| --- |
| [HttpPost]  public ActionResult Add(MenuButtonPage button)  {  if (ModelState.IsValid)  {  if (button.SelectedPage==0)  {  button.MenuButton.InformationSourceId = null;  }  else  {  button.MenuButton.InformationSourceId = button.SelectedPage;  }  button.MenuButton.Id = 0;    context.MenuButtons.Add(button.MenuButton);  context.SaveChanges();  if (button.MenuButton.IsRootButton)  {  return RedirectToAction("List");  }  else  {  return RedirectToAction("Branch", new { menuButtonId = GetParrentButton(button.MenuButton).Id});  }    }  return View(button);  } |

Listing.4. Dodawanie menu

### 6.4. Zarządzanie redaktorami

Administrator ma możliwość tworzenia, edycji i usuwania redaktorów(funkcjonalności z unktu widzenia redaktora omówimy w dalszej części)



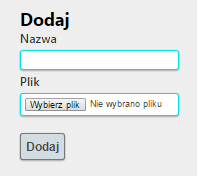
Rys.11. Dodawanie redaktorów

|  |
| --- |
| [HttpPost]  [AllowAnonymous]  [ValidateAntiForgeryToken]  public ActionResult Register(RegisterModel model)  {  if (ModelState.IsValid && model.UserName != " ")  {  // Attempt to register the user  try  {  WebSecurity.CreateUserAndAccount(model.UserName, model.Password, new { Discriminator = "CopyWriter", Name = model.Name, Surname = model.Surname, Email = model.Email });  Roles.AddUserToRole(model.UserName, "CopyWriter");  return RedirectToAction("Index", "Home");  }  catch (MembershipCreateUserException e)  {  ModelState.AddModelError("", LoginMessagesHelper.ErrorCodeToString(e.StatusCode));  }  }  // If we got this far, something failed, redisplay form  return View(model);  } |

### Listing.5.Rejestrowanie/tworzenie redaktora

### 6.5. Zarządzanie plikami

W systemie istnieje możliwość wgrywania plików na serwer. Pliki można później wykorzystywać w różnych funkcjonalnościach (np. podczas tworzenia aktualności). Takie rozwiązanie jest zaimplementowane we wszystkich wiodących Systemach Zarządzania Treścią.



Rys.12. Dodawanie plików



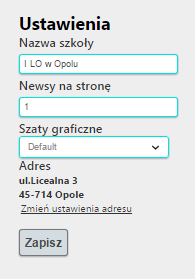
Rys.13. Lista plików

|  |
| --- |
| [Authorize]  public ActionResult List()  {  IQueryable<File> files;  if (Roles.IsUserInRole("Administrator"))  {  files = context.Files;  }  else  {  var userId = WebSecurity.GetUserId(User.Identity.Name);  files = context.Files.Where(x => x.AuthorId == userId);  }  return View(files); } |

### Listing.6. Wyświetlanie zamieszczonych plików

### 6.6. Ustawienia

Tutaj możemy zmienić najbardziej podstawowe informacje o szkole, szatę graficzną oraz ilość wiadomości, pojawiających się na stronę.



Rys.14. Ustawienia

***Teraz omówimy funkcjonalności systemu z punktu widzenia redaktora.***

### 6.7. Aktualności



Rys.15. Dodawanie aktualności

Każdy redaktor ma możliwość usuwania i edycji **jedynie swoich aktualności.**



Rys.16. Lista aktualności

### 6.8. Pliki

Każdy redaktor ma możliwość dodawania, edycji i usuwania swoich plików na serwerze.



Rys.17. Dodawanie plików

Dodatkowymi funkcjonalnościami miało być wyszukiwanie po tagach i pobieranie plików z aktualności. Te funkcjonalności zostały zaprezentowane w punktach 6.10 i 6.11.

### 6.9. Tagi-wyszukiwanie aktualności

System posiada możliwość otagowywania aktualności. Jest to wykorzystane przy wyszukiwaniu aktualności.



Rys.18. Dodawanie tagów



Rys.19. Wyszukiwanie aktualności



Rys.20. Znalezione aktualności

|  |
| --- |
| [HttpPost]  [ValidateAntiForgeryToken]  public ActionResult Create(Tag tag)  {  if (ModelState.IsValid)  {  context.Tags.Add(tag);  context.SaveChanges();  return RedirectToAction("List", "Tags");  }  return View(tag);  } |

Listing.7.Tworzenie tagów.

### 6.10. Pobieranie plików z newsów

Dodaliśmy również możliwość pobierania plików z aktualności:



Rys.21. Pobieranie załączonych plików

|  |
| --- |
| [AllowAnonymous]  public ActionResult Download(int id)  {  var file = context.Files.FirstOrDefault(x => x.Id == id);  if (file==null)  {  return HttpNotFound();  }  byte[] fileBytes = System.IO.File.ReadAllBytes(Server.MapPath(file.FilePath));  return File(fileBytes, System.Net.Mime.MediaTypeNames.Application.Octet, file.FileName + file.Extension);  } |

Listing.8.Pobieranie plików

### 6.11.Walidacja danych

W systemie zadbaliśmy o walidację danych. Każde błędne wypełnienie pól kończy się komunikatem ze strony systemu. Poniżej zaimplementowana walidacja na przykładzie procesu dodawania redaktora do systemu.



Rys.22. Walidacja danych

|  |
| --- |
| [Required(ErrorMessage = "Krótki opis jest wymagany")]  [Display(Name = "Krótki opis")]  public string ShortContent { |

Listing.9. Przykład Data Anmotations odpowiedzialnych za wyświetlanie komunikatów o błędnych danych

***Panel klienta składa się jedynie z aktualności(również wyszukiwanie aktualności) oraz zbudowanego przez admina menu wraz z podstronami.***

# 7. Testowanie i poprawki systemu

W trakcie procesu rozwoju aplikacji istotną fazą jest faza testów. W przypadku naszego programu testowanie było podzielone na dwie fazy – zewnętrzne oraz wewnętrzne testy.

Wewnętrzne testy były przeprowadzane metodą testów użytkownika – podczas rozwoju aplikacji programiści na bieżąco sprawdzali jakość i bezawaryjność swoich rozwiązań wcielając się chwilowo w potencjalnego użytkownika.

Zewnętrzne testy zostały przeprowadzone przez osoby z zewnątrz projekty, które starały się wyłapać jak najwięcej niedociągnięć i błędów m.in. za pomocą :

- SQL Injection,

- testowania walidacji wprowadzanych danych przez aplikacje,

- testowania stopnia realizacji wymagań funkcjonalnych przez autorów,

- testowania procesów autoryzacji w aplikacji.

Poniżej zamieszczono tabelę, która przedstawia błędy znalezione podczas fazy rozwoju oraz testów.

Pole „Zewnętrzny/wewnętrzny” informuje czy błąd został znaleziony przez nas samych(wewnętrzny) czy przez inną grupę(zewnętrzny).

|  |  |  |  |
| --- | --- | --- | --- |
| Błąd | Zewnętrzny/  wewnętrzny | Rozwiązany(tak/nie) | Komentarz |
| CopyWriter nie może pisać artkułów, nie może wyświetlić listy artykułów, nie może edytować swoich artykułów, nie może usuwać swoich artykułów. | Wewnętrzny | tak | CopyWriter może wykonywać wymienione zadania. Nie może jedynie edytować i usuwać artykułów innych redaktorów. |
| Brak walidacji kodu pocztowego w adresie szkoły. | Wewnętrzny | tak |  |
| Brak walidacji email przy rejestracji użytkownika. | Wewnętrzny | tak |  |
| CopyWriter nie ma uprawnień do dodawania plików do bazy danych. | Wewnętrzny | tak | CopyWriter może dodawać pliki do bazy. Nie może jedynie usuwać plików innych redaktorów. |
| Zamienione typy inputów password i email przy rejestracji użytkownika. Użytkownik zamiast Nazwy ma Teraz Imię. | Wewnętrzny | tak |  |
| Wyświetlanie "Szczegóły" zamiast tytułu Newsa w Widoku Details i brak informacji o autorze i dacie dodania. | Wewnętrzny | tak |  |
| Wyświetlanie newsów w kolejności od najstarszego do najnowszego. | Zewnętrzny | tak | Aktualności są teraz wyświetlane w kolejności od najnowszego do najstarszego. |
| Częściowa utrata informacji w formularzu dodawania aktualności w przypadku błędnego wypełnienia tego formularza. | Zewnętrzny | nie | Nienaprawienie błędu jest spowodowane tym, że korzystaliśmy z zewnętrznych bibliotek, których już nie edytowaliśmy. |
| Brak wyświetlania informacji o braku uprawnień przy próbie wykonywania działań wymagających odpowiednich uprawnień |  | tak |  |