godlo-kontur

POLITECHNIKA ŚLĄSKA W GLIWICACH

WYDZIAŁ ELEKTRYCZNY

Katedra Energoelektroniki, Napędu Elektrycznego i Robotyki

**PROJEKT INŻYNIERSKI**

**Autoryzacja witryny internetowej w oparciu o komponent framework’u Joomla!**

**Website authorization with Joomla! framework component**

Student: **Krzysztof Janusz CHADYNKA**  
Nr albumu: 210370

Studia: Stacjonarne I stopnia  
Kierunek: Informatyka

Specjalność: Informatyka w systemach elektrycznych

Prowadzący: dr inż. Maciej SAJKOWSKI

Recenzent: dr inż. Michał JELEŃ

sPIS TREŚCI

[1. Wstęp 4](#_Toc409906275)

[1.1 Cel projektu 4](#_Toc409906276)

[2. Zakres projektu 5](#_Toc409906277)

[2.1 Analiza przedmiotu projektu 5](#_Toc409906278)

[3. Założenia projektu 6](#_Toc409906279)

[3.1 Model MVC 6](#_Toc409906280)

[3.2 Wykorzystywane środowisko 7](#_Toc409906281)

[3.3 Wykorzystywany system bazy danych 7](#_Toc409906282)

[4. Dobór rozwiązań 8](#_Toc409906283)

[5. Część projektowa 9](#_Toc409906284)

[5.1 Algorytm procesu autoryzacji 9](#_Toc409906285)

[5.2 Struktura bazy danych 10](#_Toc409906286)

[6. Implementacja 13](#_Toc409906287)

[6.1 Struktura plików i katalogów 13](#_Toc409906288)

[6.2 Ogólne informacje na temat kodu źródłowego 14](#_Toc409906289)

[6.3 Część administracyjna komponentu 16](#_Toc409906290)

[6.4 Część frontowa komponentu 21](#_Toc409906291)

[6.5 Proces autoryzacji 21](#_Toc409906292)

[7. Dokumentacja użytkownika 25](#_Toc409906293)

[7.1 Wymagania sprzętowe 25](#_Toc409906294)

[7.2 Instalacja 25](#_Toc409906295)

[7.3 Zarządzanie aplikacją 28](#_Toc409906296)

[7.4 Część frontowa witryny 37](#_Toc409906297)

[8. Podsumowanie i wnioski 38](#_Toc409906298)

[8.1 Podsumowanie 38](#_Toc409906299)

[8.2 Wnioski 38](#_Toc409906300)

[9. Literatura Błąd! Nie zdefiniowano zakładki.](#_Toc409906301)

[10. Ikonografia 40](#_Toc409906302)

# Wstęp

Na początku tego podrozdziału warto wspomnieć czym właściwie jest autoryzacja. Autoryzacja polega na dodawaniu pewnych uprawnień użytkownikowi, zezwalających na dostęp do określonych zasobów oraz ograniczenie dostępu do innych.

## Cel projektu

Celem projektu jest zaimplementowanie funkcjonalności pozwalającej administratorowi witryny internetowej na łatwe i przejrzyste zarządzanie rolami użytkowników, a co za tym idzie, przydzielanie im odpowiednich uprawnień.

Mechanizm oparty zostanie na jednym z wielu dostępnych na rynku framework’ów  –  Joomla!. Jest to jeden z najstarszych szkieletów aplikacji internetowych opartym na modelu MVC.

Motywacją do podjęcia takiej tematyki była wykonywana praca zawodowa, opierająca się właśnie na tworzeniu stron internetowych i rozszerzeń dla nich.

Końcowym efektem realizacji projektu powinien być komponent pozwalający na dodawanie nowych, nazwanych przez administratora ról oraz przypisywanie ich odpowiednim użytkownikom. Dla każdej roli będą przypisane kategorie artykułów, do których dana grupa użytkowników będzie miała dostęp. Na podstawie przypisanej roli, przy każdorazowej próbie wyświetlenia artykułu przez danego użytkownika, następować będzie proces autoryzacji, który zweryfikuje uprawnienia nadane użytkownikowi, a co za tym idzie zezwoli lub zabroni dostępu do treści.

# Zakres projektu

W tej części dokumentu wyjaśniono na czym polega projekt, co stanowi jego istotę oraz przedstawiono istniejące rozwiązania alternatywne.

## Analiza przedmiotu projektu

Na początku tego podrozdziału warto wspomnieć czym właściwie jest autoryzacja. Autoryzacja polega na dodawaniu pewnych uprawnień użytkownikowi, zezwalających na dostęp do określonych zasobów oraz ograniczenie dostępu do innych.

Jak już zostało wspomniane w ramach opisywania przedmiotu projektu w  rozdziale pierwszym, aplikacja, której procesu implementacji dotyczy niniejsza praca, zostanie oparta na framework’u Joomla. Będzie stanowić jej rozszerzenie – a konkretnie komponent.

Wybór tematyki i framework’u sugerowany był zapotrzebowaniem na tego typu narzędzie. Twórcy technologii Joomla! przewidują co prawda funkcjonalność typów użytkowników, jednak dotyczą one jedynie uprawnień administrowania witryny od strony zaplecza.

Podczas analizy przedmiotu projektu natrafiono na rozwiązanie, które także spełnia wymagania autoryzacji artykułów, jednak jest to tylko część jego funkcjonalności. Wadą tego rozwiązania jest także fakt, iż jest ono płatne.

Istnieje także rozwiązanie pozwalające na umieszczenie załączników pobieranych na podstawie wykupionej wcześniej licencji.

Zakres projektu obejmuje prosty interfejs służący do obsługi autoryzacji wyłącznie artykułów.

Problem, będący istotą projektu, polega na zaimplementowaniu rozwiązania, które w łatwy i przystępny dla administratora systemu sposób, pozwoli na bezpośrednie przypisanie ról, na podstawie których dany użytkownik otrzyma dostęp do kategorii artykułów.

Założeniem tej pracy jest możliwe optymalna, ze względu na zasoby, implementacja kodu źródłowego, a także przystępność i prostota użytkowania gotowego komponentu dla użytkowników nie mających większej wiedzy o zagadnieniach informatycznych.

# Założenia projektu [1]

W tym rozdziale zostały opisane założenia jakie zostały przyjęte podczas etapów: projektowania i implementacji komponentu, będącego obiektem niniejszego projektu.

## Wykorzystywany język programowania

Implementacja projektu zostanie oparta o język programowania PHP, gdyż w takim języku obsługiwane są skryptu framework’u Joomla!. Wykorzystywany język programowania

## Model MVC [2]

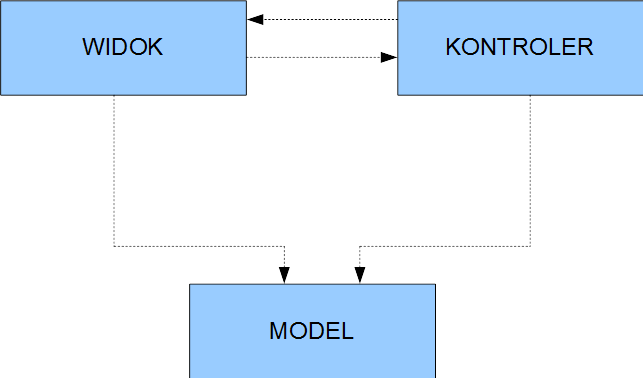
Tworzony komponent, jak i cały framework Joomla!, oparty jest na modelu MVC (ang. Model, View, Controller). MVC jest wzorcem projektowym powstałym w latach 70. XX wieku, którego architektura opiera się na oddzieleniu od siebie trzech podstawowych elementów aplikacji:

* modelu danych,
* widoku,
* kontrolera.

**Model** – jest funkcjonalnie niezależny od pozostałych dwóch elementów, przechowuje dane oraz rejestruje zależne od siebie widoki i kontrolery, powiadamiając o zmianie danych.

**Widok** – jest odpowiedzialny za sposób prezentacji danych. W niniejszym projekcie do tego celu używa się plików HTML oraz XML. Ogólnie zadania widoku można określić następująco: inicjalizacja powiązanego z nim kontrolera, wyświetlanie informacji dla użytkownika, pobieranie danych z modelu.

**Kontroler** – implementuje logikę sterowania. Do jego zadań można zaliczyć: pobieranie wejść użytkowników jako zdarzeń oraz reakcję na działania użytkownika wywołującą akcje zawarte w modelu oraz powodującą zmiany w widoku.



Rysunek 1 Schemat wzorca MVC

## Wykorzystywane środowisko

Dla celów tworzenia projektu używane będzie oprogramowanie pozwalające na utworzenie lokalnego serwera - *WebServ*, a także środowisko programistyczne *NetBeans*, w którym dokonana zostanie implementacja całego kodu projektu.

## Wykorzystywany system bazy danych

Jako systemu bazy danych używany będzie system MySQL, co także narzuca wykorzystywany framework.

# Dobór rozwiązań [3]

W poprzednich rozdziałach założono, iż projekt będzie stanowił implementację komponentu. W dalszej części tego rozdziału nastąpi wyjaśnienie dokonanego wyboru.

Joomla! obsługuje trzy podstawowe typy rozszerzeń:

* moduły,
* dodatki (wtyczki),
* komponenty.

**Moduł** jest najprostszym typem rozszerzenia. Stanowi niewielką funkcjonalność służącą najczęściej do reprezentacji danych (np. moduł reklamy) lub prostego pobierania danych – w tym wypadku najczęściej są częścią składową komponentu (przykładowo moduł logowania). Moduły stanowią także reprezentacje menu na stronie – aby wyświetlić dodatkowe menu należy najpierw przypisać je do modułu.

**Dodatek (wtyczka)** stanowi pewną funkcjonalność wykonującą operację na gotowych elementach strony, świadczą usługi na rzecz komponentów i modułów. Wtyczki są również najbardziej wyspecjalizowanymi rozszerzeniami.

**Komponent** jest najbardziej złożonym rozszerzeniem. Komponenty stanowią rodzaj niewielkich aplikacji, składających się z części administracyjnej i prezentacyjnej. Działanie całej witryny uruchomionej na silniku Joomla!, opiera się na funkcjonowaniu różnych komponentów. Przykładem tego jest komponent służący do rejestracji użytkowników (*com\_registration*).

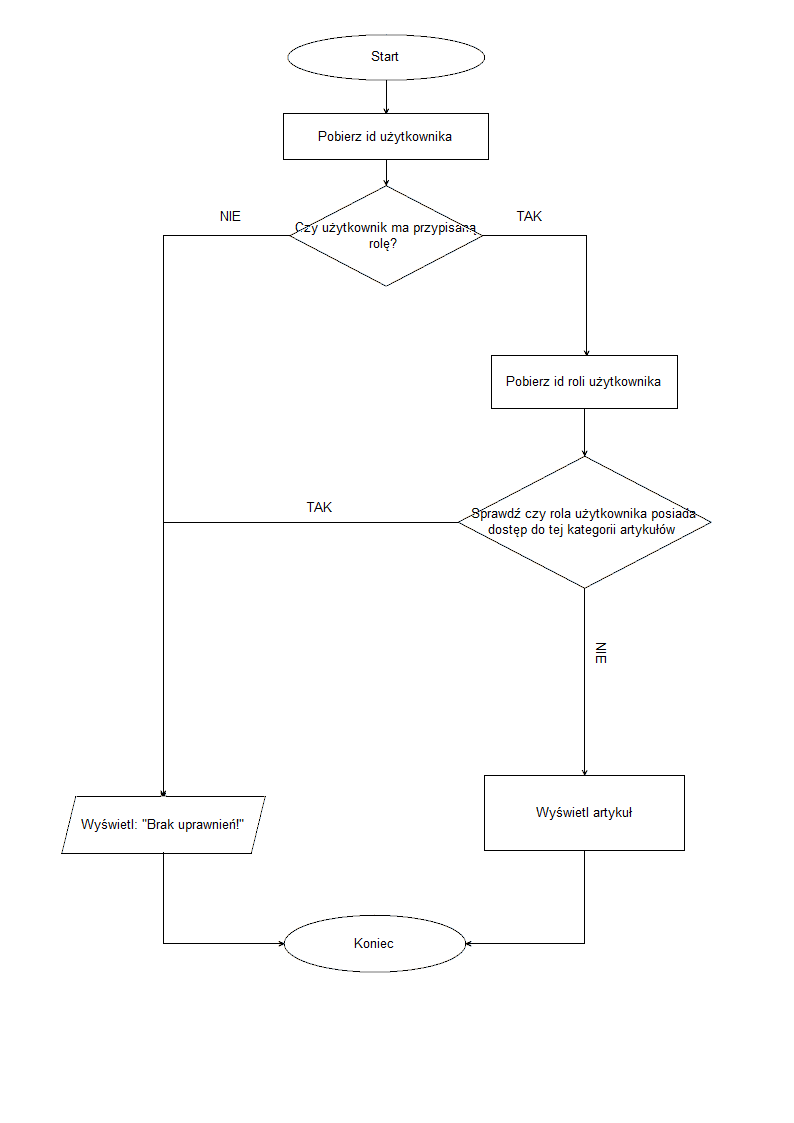
Aplikacja opracowana w ramach projektu będzie stanowić dodatkową funkcjonalność, składającą się z części administracyjnej i frontowej, co powoduje wybór komponentu, jako typu rozszerzenia na którym zostanie oparta.

# Część projektowa

W rozdziale przedstawiono algorytm procesu autoryzacji oraz strukturę bazy danych. Aplikacja opracowana w ramach projektu będzie stanowić dodatkową funkcjonalność, składającą się z części administracyjnej i frontowej, co powoduje wybór komponentu, jako typu rozszerzenia na którym zostanie oparta.

## Algorytm procesu autoryzacji

Na rysunku 2 zaprezentowano schemat blokowy procesu autoryzacji.



Rysunek 2 Schemat blokowy procesu autoryzacji

Jak wynika z Rysunku 2, proces autoryzacji odbywa się następująco:

1. Na początku zostaje pobrane ID użytkownika obecnie zalogowanego w  witrynie.
2. Na podstawie pobranego ID przeszukiwana jest odpowiednia tabela w bazie danych w celu sprawdzenia, czy zalogowanemu użytkownikowi przypisana jest jakaś rola.
3. Jeżeli wynik powyższego warunku okazał się negatywny, proces zostaje zakończony wyświetleniem komunikatu o nieudanej autoryzacji. W przeciwnym przypadku pobierane jest z tej samej tabeli ID roli przypisanej do użytkownika.
4. Następnie przeszukiwana jest tabela przechowującą informacje o kategoriach dostępnych dla danej roli w celu sprawdzenia, czy dla zalogowanego użytkownika możliwe jest przeglądanie artykułów z bieżącej kategorii.
5. Jeżeli wynik drugiego warunku okazał się pozytywny zwracany jest artykuł a proces ulega zakończeniu. W przeciwnym przypadku wyświetlany jest komunikat o nieudanej autoryzacji i również proces zostaje zakończony.

## Struktura bazy danych

Framework Joomla! dysponuje już gotową bazą danych, na której oparte jest działanie witryny internetowej. Do każdej nazwy tabeli dodawany jest, ustawiony przez administratora lub programistę, prefix tabel (w przypadku wykorzystywanej w danym projekcie witryny jest to przedrostek joo\_).Ingerencja w bazę danych będzie ograniczała się jedynie do utworzenia czterech dodatkowych tabel:

• ROLES\_INFO(ID, HEADER, DESCRIPTION, IMAGE, USER\_INFO, ROLE\_INFO, CATEGORIES\_INFO),

• ROLES\_USER (USER\_ID, ROLE\_ID),

• ROLES\_ROLE (ROLE\_ID, NAME),

• ROLES\_CATEGORIES (ROLE\_ID, CATEGORY\_ID),

gdzie znaczenie poszczególnych tabel jest następujące:

1. **ROLES\_INFO (a właściwie JOO\_ROLES\_INFO)**

Jest to tabela przechowująca dane, które zostaną wyświetlone w części frontowej komponentu, zawierającej informacje na temat użytkownika, przypisanej mu roli, oraz powiązanych z tą rolą kategorii.

Informacje w tabeli zapisane zostały w postaci aliasów, które podczas ładowania na stronę zostaną automatycznie przetłumaczone (proces tłumaczenia został opisany w  rozdziale implementacyjnym).

Znaczenie poszczególnych aliasów jest następujące:

• HEADER – nagłówek strony;

• DESCRIPTION – krótki opis funkcji, jaką spełnia komponent;

• USER\_INFO – tekst poprzedzający dane identyfikujące użytkownika;

• ROLE\_INFO – tekst poprzedzający rolę użytkownika;

• CATEGORIES\_INFO – tekst poprzedzający listę kategorii dostępnych dla przypisanej użytkownikowi roli.

Tabela już przy instalacji wypełniana jest przykładowymi aliasami:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **HEADER** | **DESCRIPTION** | **IMAGE** | **USER\_INFO** | **ROLE\_INFO** | **CATEGORIES\_INFO** |
| COM\_  ROLES\_  INFO\_ HEADER | COM\_ ROLES\_ INFO\_ DESC | /components/  com\_roles/img/  avatar.png | COM\_  ROLES\_  INFO\_  USERINFO | COM\_  ROLES\_  INFO\_  ROLE\_  INFO | COM\_  ROLES\_  INFO\_  CAT\_  INFO |

Tabela 1 Struktura początkowa tabeli JOO\_ROLES\_INFO

1. **ROLES\_USER ( a właściwie JOO\_ROLES\_USER)**

Jest to tabela przechowująca ID identyfikujące użytkownika (pobrane z tabeli JOO\_USERS) oraz identyfikator roli mu przypisanej.

1. **ROLES\_ROLE ( a właściwie JOO\_ROLES\_ROLE)**

Jest to tabela przechowująca dostępne role użytkowników wraz z ich identyfikatorami. Przy instalacji tworzone są od razu trzy role użytkowników („student”, „pracownik” i „vip”), tak więc początkowa struktura tej tabeli przedstawia się następująco:

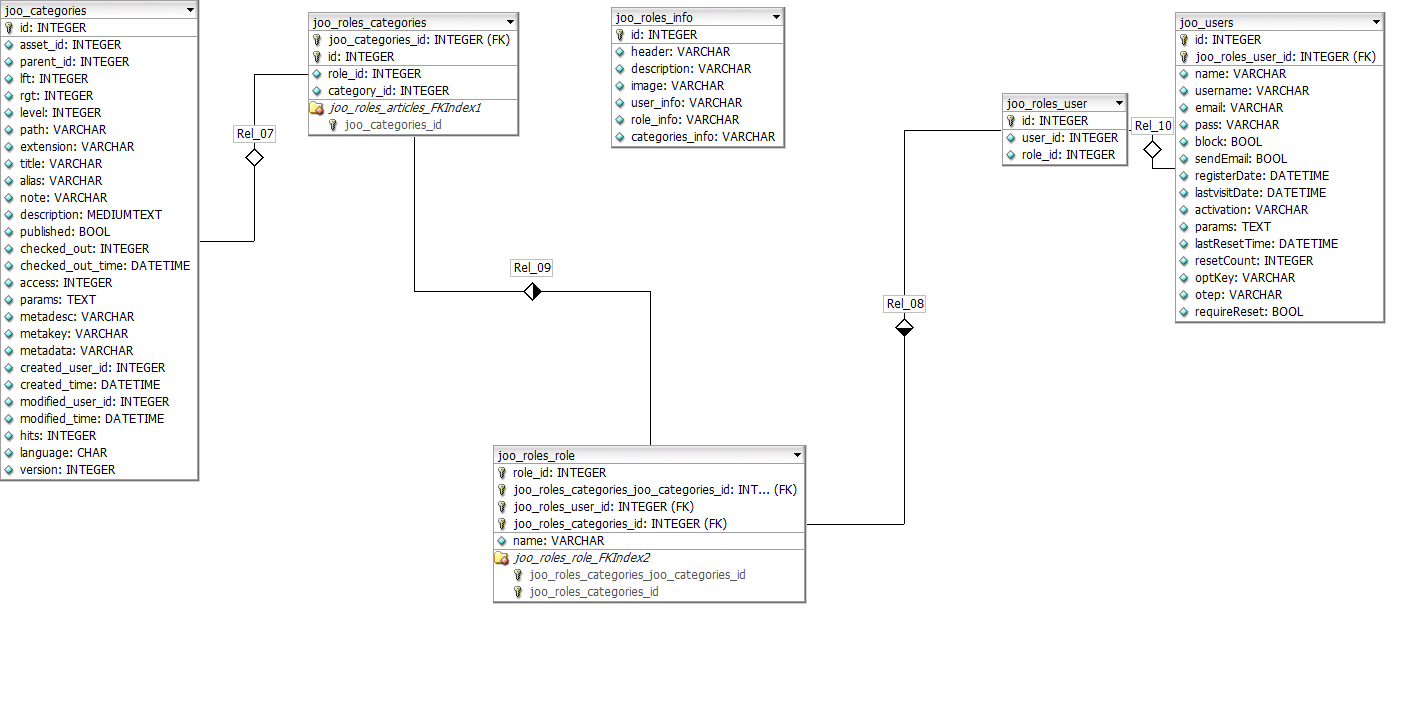
|  |  |
| --- | --- |
| **ROLE\_ID** | **NAME** |
| 1 | student |
| 2 | pracownik |
| 3 | vip |

Tabela 2Struktura początkowa tabeli JOO\_ROLES\_ROLE

1. **ROLES\_ARTICLES ( a właściwie JOO\_ROLES\_ARTICLES)**

Opisywana tabela zawiera połączone w pary identyfikatory roli wraz z przypisanymi do nich identyfikatorami kategorii artykułów (pobranych z tabeli JOO\_CATEGORIES), na dostęp do których zezwala dana rol

Ostatecznie, struktura wykorzystywanych tabel bazy danych (wraz z odpowiednimi związkami), przedstawiono na rysunku 3.



Rysunek 3 Schemat bazy danych

Na rysunku 3 widać, iż tabele JOO\_CATEGORIES oraz JOO\_USERS są tabelami predefiniowanymi framework’u. Na ich podstawie pobierane są informacje o kategoriach artykułów oraz o użytkownikach.

# Implementacja [1] [4]

W rozdziale w skróconej formie przedstawiono strukturę plików i katalogów utworzonego komponentu. Dla pojedynczych plików podano pełną nazwę wraz z rozszerzeniem.

## Struktura plików i katalogów

* + - **Główny katalog komponentu (*admin*)**
      * ***controllers***
        + kontrolery używane w zapleczu administracyjnym
      * ***css***
        + *style.css* – arkusz stylów dla zaplecza administracyjnego
      * ***language***
        + pliki językowe służące do zarządzania tłumaczeniami treści w panelu administracyjnym
      * ***models***
        + modele używane w zapleczu administracyjnym
      * ***sql***
        + pliki obsługujące zapytania do bazy danych wywoływane podczas instalacji, deinstalacji, czy aktualizacji komponentu
      * ***tables***
        + klasy opisujące połączenie z tabelami w bazie danych
      * ***views***
        + struktura katalogów i plików opisująca widoki; każdy widok posiada własną podstrukturę (dla przykładu widok *roles)*

***tmpl***

pliki reprezentujące widoki

*view.html.php*

* + - * *controller.php* – główny kontroler zaplecza administracyjnego
      * *roles.php ­*– główny plik zarządzający zapleczem administracyjnym
    - **Część frontowa komponentu (*site)***
      * ***css***
        + *style.css* – arkusz stylów dla części frontowej komponentu
      * ***img***
        + grafiki wykorzystywane w części frontowej komponentu
      * ***language***
        + pliki językowe służące do zarządzania tłumaczeniami treści w części frontowej komponentu
      * ***models***
        + modele używane w części frontowej komponentu
      * ***views***
        + struktura katalogów i plików opisująca widoki; każdy widok posiada własną podstrukturę (analogicznie do zaplecza administracyjnego)
      * *controller.php ­*– główny kontroler części
      * *roles.php* – główny plik zarządzający częścią frontową komponentu
    - *roles.xml* – plik konfiguracyjny całego komponentu

## Ogólne informacje na temat kodu źródłowego

Każdy plik z kodem źródłowym (niezależnie czy jest to widok, model, kontroler, czy też klasa pomocnicza) rozpoczyna się nagłówkiem zabraniającym dostępu do niego osobom niepowołanym.

Listing 1 Ograniczenie dostępu do pliku

|  |
| --- |
| *defined('\_JEXEC') or die('Restricted access');* |

Głównym plikiem używanym podczas procesu instalacji jest plik *roles.xml*, który został przedstawiony na listingu 2.

Listing 2 plik roles.xml

|  |
| --- |
| <extension type="component" version="3.2" method="upgrade">  <name>Roles</name>  {…}  <install>  <sql><file driver="mysql" charset="utf8">sql/install.mysql.utf8.sql</file></sql>  </install>  <uninstall>  <sql><file driver="mysql" charset="utf8">sql/unistall.mysql.utf8.sql</file></sql>  </uninstall>  <update>  <schemas><schemapath type="mysql">sql/updates/mysql/</schemapath></schemas>  </update>  <files folder="site">  { …}  <filename>controller.php</filename>  <folder>views</folder>  {…}  </files>  <administration>  <menu link="option=com\_roles">COM\_ROLES\_MENU</menu>  <submenu>  <menu link="option=com\_roles&amp;view=users">COM\_ROLES\_SUBMENU\_USERS</menu>  {…}  </submenu>  <files folder="admin">  <filename>index.html</filename>  {…}  <folder>sql</folder>  {…}  </files>  </administration>  </extension> |

Zaprezentowany na listingu 2 plik *roles.xml* przechowuje informacje na temat komponentu, takie jak dane identyfikujące (nazwa, autor), strukturę katalogów, budowę menu oraz wskazuje, których plików sql należy użyć odpowiednio do instalacji, aktualizacji czy deinstalacji komponentu.

Jak zostało wspomniane w dokumentacji użytkownika (rozdział 7), w komponencie został wdrożony system tłumaczeń, zarówno dla części administracyjnej, jak i frontowej.

Implementacja mechanizmu tłumaczenia od strony technicznej polegała na utworzeniu, w obydwu częściach komponentu, dodatkowego katalogu *language* i umieszczeniu w nim odpowiednich plików tekstowych (*.ini*) oraz dodanie wpisu w pliku roles.xml, informującego o nowym katalogu.

Przykładowy plik przedstawiony jest na listingu 3.

Listing 3 plik site/language/en-GB/en-GB.com\_roles.ini

|  |
| --- |
| COM\_ROLES\_INFO\_HEADER="Your permission"  COM\_ROLES\_INFO\_DESC="Here is a place for beautifull description"  COM\_ROLES\_INFO\_USERINFO="Your username" |

## Część administracyjna komponentu

Część administracyjna komponentu odpowiada za pobieranie i edycję danych przetwarzanych podczas działania rozszerzenia.

Pierwszą, bardzo istotną kwestią za jaką odpowiada ta część komponentu jest instalacja. To właśnie w tym miejscy w kodzie deklarowane są początkowe zapytania do bazy danych (tworzenie nowych tabel i wypełnienie ich przykładowymi danymi). Z tą czynnością powiązane są dwie inne – deinstalacja i aktualizacja komponentu –  korzystająca także ze stworzonych do tego celu zapytań SQL.

Kolejną istotną, a właściwie kluczową, funkcją części administracyjnej jest obsługa zarządzania danymi wykorzystywanymi w komponencie. Każdorazowa zmiana danych, dodanie lub usunięcie rekordu obsługiwana jest przez współpracę kontrolera, modelu oraz widoku.

Przy tworzeniu każdego z elementów architektury MVC (dla pojedynczej funkcjonalności) należy pamiętać o pewnych założeniach przyjętych przez framework Joomla!. Niezastosowanie się do wytycznych powoduje błędy uniemożliwiające prawidłową pracę aplikacji.

Ograniczenia stosowane są do odpowiedniej struktury katalogów oraz nazewnictwa plików i klas w nich zawartych.

Dla przykładu rozpatrzone zostanie utworzenie zaplecza dla obsługi ról (tworzenia nowych, edycji i usuwania istniejących).

Na początku w katalogu *views* tworzony jest podkatalog o nazwie widoku, który zostanie dodany – w tym przypadku jest to nazwa *roles*. Wewnątrz tego katalogu tworzony jest plik *view.html.php*, w którym znajduje się klasa odpowiedzialna za obsługę widoku. Następnie, wewnątrz tego katalogu tworzony jest podkatalog *tmpl*, w którym dodaje się pliki wyświetlające dane.

Aby wyświetlić rekordy z bazy danych należy, w katalogu *models*, dodać plik *roles.php*, wewnątrz którego znajduje się klasa odpowiedzialnego za to modelu.

Wymienione czynności są wystarczające do wyświetlenia listy dostępnych ról.

Natomiast by dokonywać edycji, usuwania lub dodawania pojedynczej roli należy w  katalogu *controllers* dodać dwa pliki: *roles.php* oraz *role.php*, które przechowują klasy: kontrolera listy ról i kontrolera pojedynczej roli.

W katalogu *models* dodaje się plik *role.php*, w którym tworzona jest klasa modelu pojedynczej roli. Dodatkowo tworzy się podkatalog *forms*, a nim plik *role.xml*, w którym zadeklarowane są pola formularza do edycji.

Ostatnią rzeczą jaką należy zrobić, aby dane do edycji zostały pobrane poprawnie, jest utworzenie w katalogu *tables* pliku *role.php*, który zawiera klasę będącą reprezentacją tabeli w bazie danych.

Aby ostatecznie wyświetlić formularz edycji istniejącej lub dodania nowej roli, należy w katalogu *views/role* utworzyć dodatkowy podkatalog *tmpl*, a w nim plik *edit.php*, który będzie wyświetlał formularz odpowiednio edycji lub dodania roli.

W tabeli 4 zaprezentowano zasady dotyczące nazywania klas.

|  |  |
| --- | --- |
| **Element** | **Struktura (przykład)** |
| **model** | NazwaKomponentu*Model*NazwaModelu (*RolesModelRole*) |
| **kontroler** | NazwaKomponentu*Controller*NazwaKontrolera (*RolesControllerRole*) |
| **klasa widoku** | NazwaKomponentu*View*NazwaWidoku (*RolesViewRole*) |
| **Klasa tabeli** | NazwaKomponentu*Table*NazwaTabeli (*RolesTableRole*) |

Tabela 3 Zasady dotyczące nazywania klas - panel administracyjny

W dalszej części tego podpunktu zaprezentowano wybrane fragmenty kodu źródłowego.

Listing 4 plik install.mysql.utf8.sql

|  |
| --- |
| DROP TABLE IF EXISTS `#\_\_roles\_info`; DROP TABLE IF EXISTS `#\_\_roles\_user`;  DROP TABLE IF EXISTS `#\_\_roles\_role`; DROP TABLE IF EXISTS `#\_\_roles\_categories`;  CREATE TABLE `#\_\_roles\_info` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `header` varchar(128) NOT NULL, `description` varchar(512) NOT NULL,  `image` varchar(128), `user\_info` varchar(128),  `role\_info` varchar(128), `categories\_info` varchar(128),  PRIMARY KEY (`id`)  ) ENGINE=InnoDB;  CREATE TABLE `#\_\_roles\_user` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `user\_id` int(11) NOT NULL, `role\_id` int(11) NOT NULL, PRIMARY KEY (`id`)  ) ENGINE=InnoDB;  CREATE TABLE `#\_\_roles\_role` (  `role\_id` int(11) NOT NULL AUTO\_INCREMENT,  `name` varchar(128) NOT NULL,  PRIMARY KEY (`role\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  CREATE TABLE `#\_\_roles\_categories` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `role\_id` int(11) NOT NULL, `category\_id` int(11) NOT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  INSERT INTO `#\_\_roles\_role` (`name`) VALUES ('student'), ('employee'), ('vip');  INSERT INTO `#\_\_roles\_categories` (`role\_id`, `category\_id`) VALUES (1,2), (2,2), (3,2);  INSERT INTO `#\_\_roles\_info`  (`header`, `description`, `image` , `user\_info`, `role\_info`, `categories\_info`)  VALUES  ('COM\_ROLES\_INFO\_HEADER','COM\_ROLES\_INFO\_DESC', '/components/com\_roles/img/avatar.png' ,'COM\_ROLES\_INFO\_USERINFO',  'COM\_ROLES\_INFO\_ROLE\_INFO', 'COM\_ROLES\_INFO\_CAT\_INFO'); |

Na listingu 4 przedstawiono zestaw zapytań do bazy danych wykonywanych w  trakcie instalacji komponentu.

Listing 5 klasa RolesControllerCategory

|  |
| --- |
| class RolesControllerCategories extends JControllerAdmin  {  public function getModel($name = 'Category', $prefix = 'RolesModel')  {  return parent::getModel($name, $prefix, array('ignore\_request' => true));  }  } |

Na listingu 5 zamieszczono definicję klasy stanowiącej kontroler listy kategorii.

Listing 6 klasa RolesModelCategories

|  |
| --- |
| class RolesModelCategories extends JModelList  {  protected $categories;    protected function getListQuery()  {  $db = JFactory::getDbo();  $query = $db->getQuery(true);    $query->select('rc.id, rc.role\_id AS role\_id, '  . 'rc.category\_id AS category\_id, '  . 'r.name AS role\_name, c.title AS category\_name')  ->from('#\_\_roles\_categories rc')  ->leftJoin('#\_\_roles\_role r '  . 'ON r.role\_id=rc.role\_id')  ->leftJoin('#\_\_categories c '  . 'ON c.id=rc.category\_id');    return $query;  }  } |

Listing 6 prezentuje klasę będącą reprezentacją modelu służącego do pobierania z  bazy danych informacji niezbędnych do operacji na kategoriach.

Listing 7 klasa RolesViewCategories

|  |
| --- |
| class RolesViewCategories extends JViewLegacy  {  private function setData()  {  $this->items = $this->get('Items');  $this->pagination = $this->get('Pagination');  }    private function displayErrors($errors)  {  if (count($errors))  {  JError::raiseError(500, implode('<br />', $errors));  return false;  }  }    protected function addToolBar()  {  JToolBarHelper::title(JText::\_('COM\_ROLES\_MANAGER\_CATEGORIES'));  JToolBarHelper::deleteList('', 'categories.delete');  JToolBarHelper::editList('category.edit');  JToolBarHelper::addNew('category.add');  }  public function display($tpl = null)  {  $this->setData();  $this->displayErrors($this->get('Errors'));  $this->addToolBar();    parent::display($tpl);  }  } |

Na listingu 7 zamieszczono klasę *RolesViewCategories* odpowiedzialną za wyświetlanie listy kategorii.

W obrębie opisywanej klasy zaimplementowano cztery następujące metody:

* + - setData() – pobiera dane z modelu RolesModelCategories i przypisuje je do zmiennych obsługiwanych w widoku;
    - displayErrors() – odpowiada za obsługę błędów;
    - display() – pobiera dane i przekazuje do widoku, wyświetla pasek narzędzi, wyświetla istniejące błędy;
    - addToolBar() – dodaje pasek narzędzi do widoku;

Listing 8 klasa RolesModelCategory

|  |
| --- |
| class RolesModelCategory extends JModelAdmin  {  public function getTable($type = 'RolesCategories', $prefix = 'RolesTable', $config = array())  {  return JTable::getInstance($type, $prefix, $config);  }  public function getForm($data = array(), $loadData = true)  {  $form = $this->loadForm('com\_roles.category', 'category',  array('control' => 'jform', 'load\_data' => $loadData));    if (empty($form))  return false;    return $form;  }    protected function loadFormData()  {  $data = JFactory::getApplication()->getUserState('com\_roles.edit.category.data', array());    if (empty($data))  $data = $this->getItem();    return $data;  }  } |

Listing 8 zawiera klasę *RolesModelCategory* odpowiedzialną za przekazanie odpowiednich danych do formularza edycji kategorii. Kod tej klasy pobiera elementy z  bazy danych, przygotowuje formularz edycyjny i przypisuje mu odpowiednie pola (zdeklarowane w pliku *xml*).

## Część frontowa komponentu

Część frontowa odpowiedzialna jest za wyświetlanie danych ustawionych w panelu administracyjnym.

## Proces autoryzacji

Cały proces autoryzacji odbywa się w pliku */administrator/components /com\_content/controller.php*, który na potrzeby projektu został zmodyfikowany. Na listingach 9 - 12 zamieszczono zmienione lub dodane na potrzeby autoryzacji fragmenty kodu.

Listing 9 funkcja getCurrentCategoryID

|  |
| --- |
| private function getCurrentCategoryID($art\_id)  {  $db = JFactory::getDbo();  $query = $db->getQuery(true);  $query->select('catid')->from('#\_\_content')->where('id='.$art\_id);  $db->setQuery($query);  return $db->loadResult();  } |

Funkcja *getCurrentCategoryID,* zaprezentowana na listingu 9, odpowiada za  pobranie ID bieżącej kategorii z bazy danych na podstawie, przekazanego jako argumentu funkcji, ID artykułu.

Listing 10 funkcja GetCurrentRoleID

|  |
| --- |
| private function getCurrentRoleID($user\_id)  {  $db = JFactory::getDbo();  $query = $db->getQuery(true);  $query->select('role\_id')->from('#\_\_roles\_user')->where('user\_id='.$user\_id);  $db->setQuery($query);  return $db->loadResult();  } |

Funkcja przedstawiona na listingu 10 odpowiada za pobranie z bazy danych ID roli przypisanej użytkownikowi na podstawie jego ID, przekazanego jako argument funkcji.

Listing 11 funkcja checkIfCategoryExists

|  |
| --- |
| private function checkIfCategoryExists($role\_id, $cat\_id)  {  $db = JFactory::getDbo();  $query = $db->getQuery(true);  $query->select('category\_id')->from('#\_\_roles\_categories')->where('role\_id='.$role\_id);  $db->setQuery($query);  $results = $db->loadObjectList();  foreach ($results as $result)  {  if ($result->category\_id == $cat\_id)  {  return true;  break;  }  }  return false;  } |

Zamieszczona na listingu 11 funkcja *checkIfCategoryExists* odpowiada za konkretny proces autoryzacji. Jako argumenty pobierane są: ID roli użytkownika oraz ID kategorii. Następnie, z tabeli *joo\_roles\_categories*, pobierane są te wpisy, które mają ID roli równe przekazanemu. W kolejnym kroku w pętli wykonywane jest sprawdzanie, czy w którymś z pobranych wierszy znajduje się przekazane ID kategorii. Jeżeli tak, funkcja zwraca wartość *true*.

Ostateczne sprawdzenie dostępu użytkownika do danej kategorii przedstawiono na listingu 12.

Listing 12 Weryfikacja autoryzacji

|  |
| --- |
| if ($role\_id != 0 && $cat\_id != 0)  {  if ($this->checkIfCategoryExists($role\_id, $cat\_id))  parent::display($cachable, $safeurlparams);  else  JFactory::getApplication()-> enqueueMessage(JText::\_('COM\_ROLES\_ACCESS\_DENIED\_MESSAGE'), 'Error');  } |

Na początku następuje sprawdzenie czy ID roli użytkownika oraz ID kategorii zostały określone. Jeżeli tak, to wywoływana jest funkcja *checkIfCategoryExists* i,  na  podstawie zwróconej przez nią wartości, udostępnia się lub blokuje daną kategorię.

# Dokumentacja użytkownika [1]

Niniejszy rozdział dotyczy obsługi komponentu przez użytkownika. Opisane zostało jakimi narzędziami należy dysponować, aby móc korzystać z komponentu utworzonego w ramach projektu. Przedstawiono także proces instalacji oraz wytyczne dotyczące poruszania się po aplikacji.

## Wymagania sprzętowe

Aby móc korzystać z komponentu, serwer, na którym będzie on zainstalowany, musi spełniać następujące wymagania:

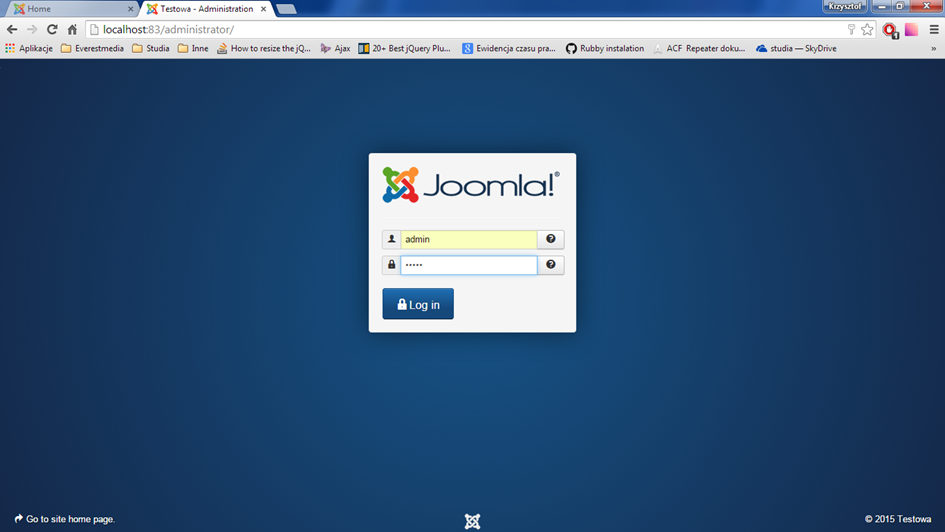
* + - obsługa PHP w wersji 5.3.10 lub wyższej;
    - Obsługa bazy danych w standardzie MySQL 5.1 (lub nowsza);
    - Na serwerze musi być zainstalowana witryna internetowa oparta na framework’u Joomla! wersji 3.1 lub wyższej (Proces instalacji szczegółowo opisany jest na stronie producenta – www.wiki.joomla.pl).

## Instalacja

Po zainstalowaniu framework’u na serwerze przechodzi się do instalacji komponentu.

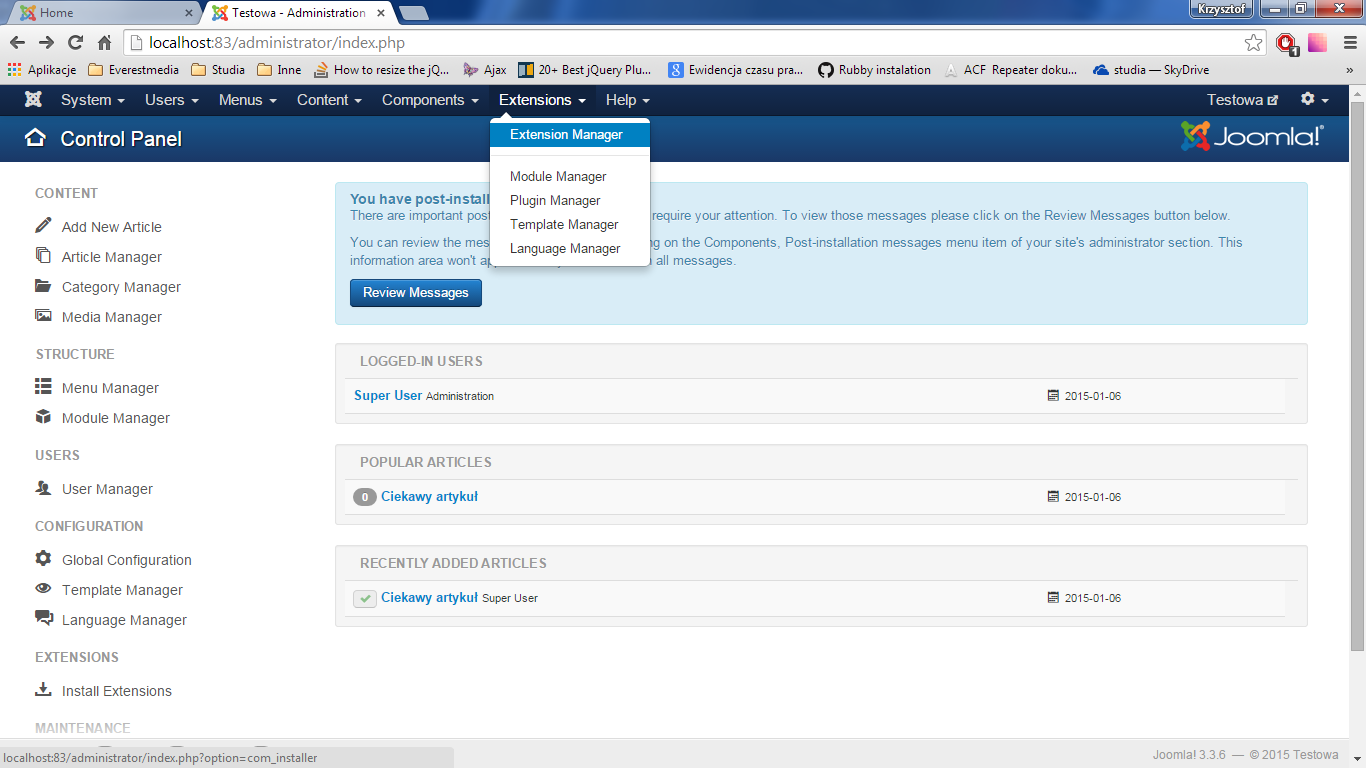
W tym celu należy:

1. zalogować się do panelu administracyjnego, korzystając z danych podanych podczas instalacji Joomla!. Widok ekranu w trakcie logowania przedstawiono na rysunku 4.



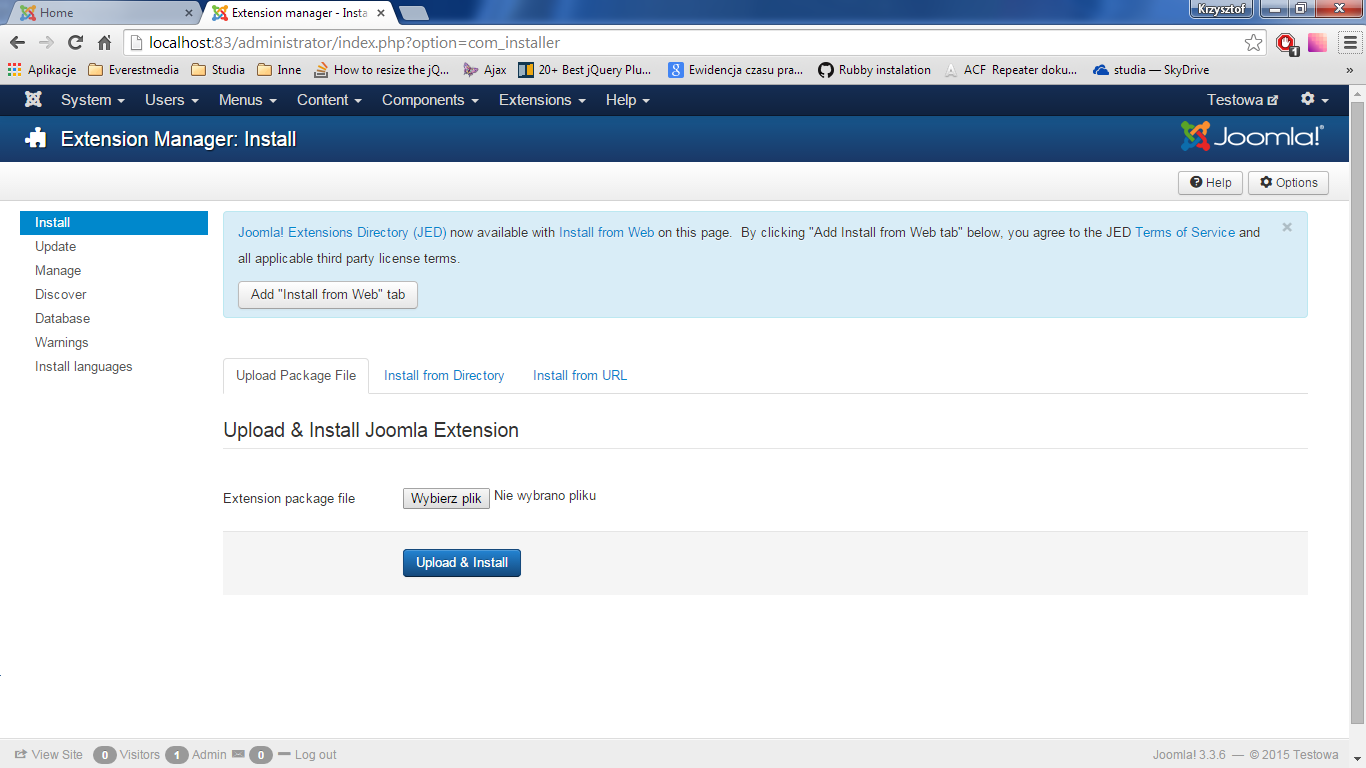
Rysunek 4 Instalacja komponentu - logowanie do panelu administracyjnego

1. Wybrać z górnego menu pozycję *Extensions -> Extension Manager*. Widok głównej strony panelu administracyjnego zaprezentowano na rysunku 5.



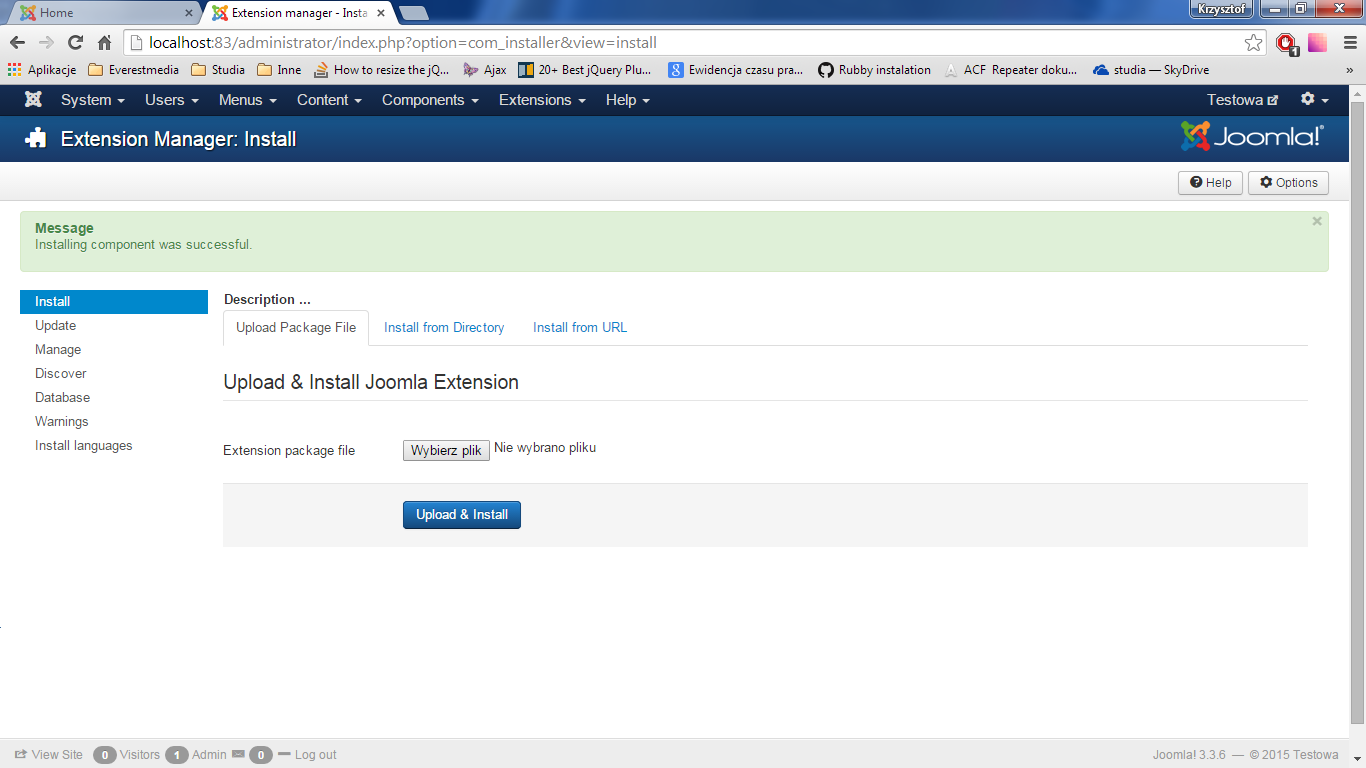
Rysunek 5 Instalacja komponentu - przejście do odpowiedniej zakładki

1. Wybrać plik z dysku (komponent com\_roles w formacie .zip), po czym kliknąć przycisk Upload & Install. Widok ekranu w trakcie wyboru pliku zamieszczono na rysunku 6.



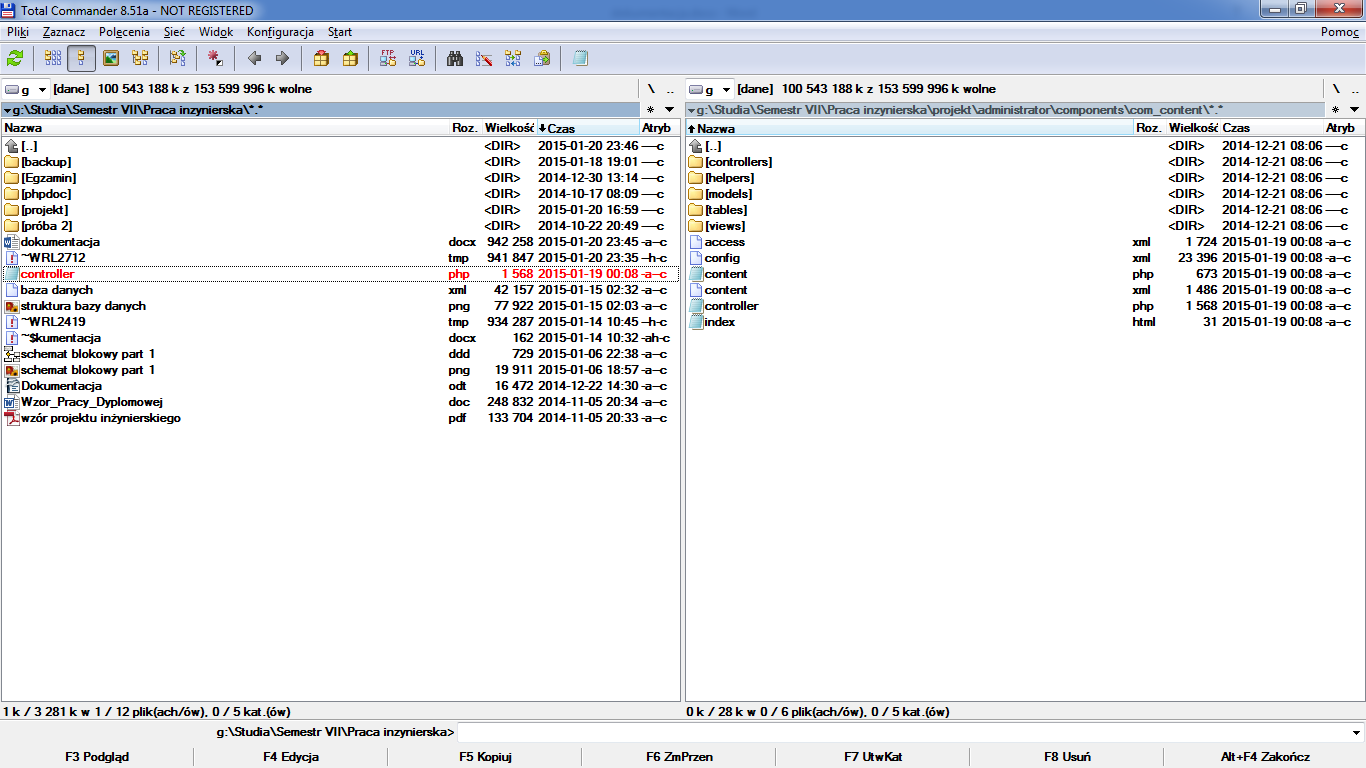
Rysunek 6 Instalacja komponentu - wybór pliku

1. Jeżeli podczas instalacji nie wystąpił żaden błąd, w górnym obszarze strony pojawi się komunikat informujący o zakończonej sukcesem instalacji komponentu, co zostało przedstawiono na rysunku 7.



Rysunek 7 Instalacja komponentu - informacja o pozytywnym wyniku instalacji

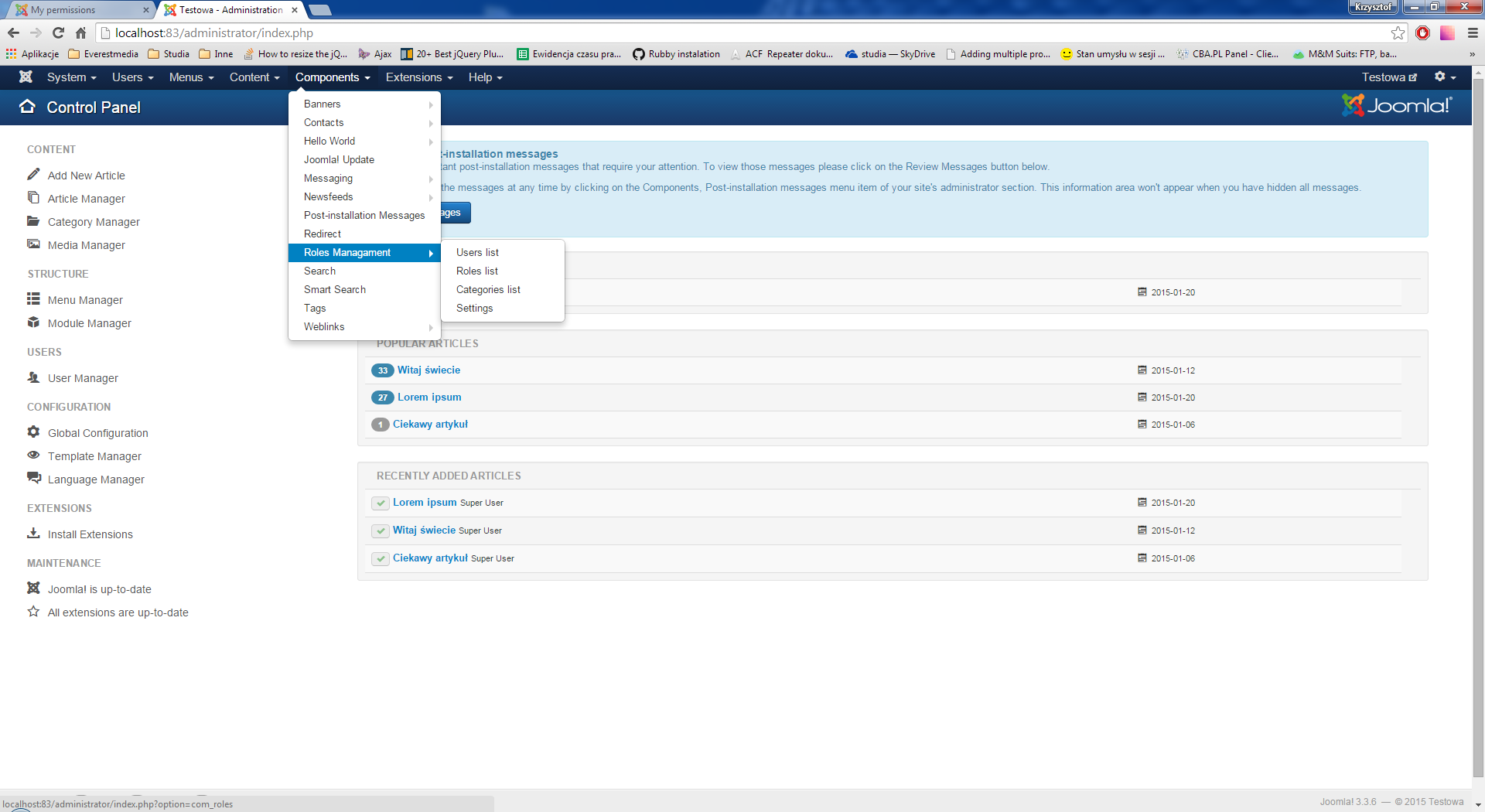
1. Po zakończonej instalacji należy, za pomocą protokołu narzędzia obsługującego protokół FTP (np. Total Commander), przesłać na zdalny (lub lokalny) serwer plik *controller.php* (plik ten znajduje się także na płycie cd dołączonej do projektu) do katalogu *administrator/components/com\_content*. Spowoduje to nadpisanie istniejącego już w tej lokalizacji pliku o identycznej nazwie. Czynność tą wykonuje się, ponieważ w pliku *controller.php* zostały zaimplementowane dodatkowe funkcjonalności, niezbędne do prawidłowego działania instalowanego komponentu. Na rysunku 8 zaprezentowano widok ekranu w programie Total Commander w trakcie przesyłania pliku *controller.php* na serwer.



Rysunek 8 Przesłanie pliku controller.php na serwer

## Zarządzanie aplikacją

Po udanej instalacji komponentu, w menu głównym, w zakładce Components, pojawi się dodatkowa pozycja Roles Management, co zostało udokumentowane na rysunku 9.



Rysunek 9 Położenie komponentu w menu głównym - panel administracyjny

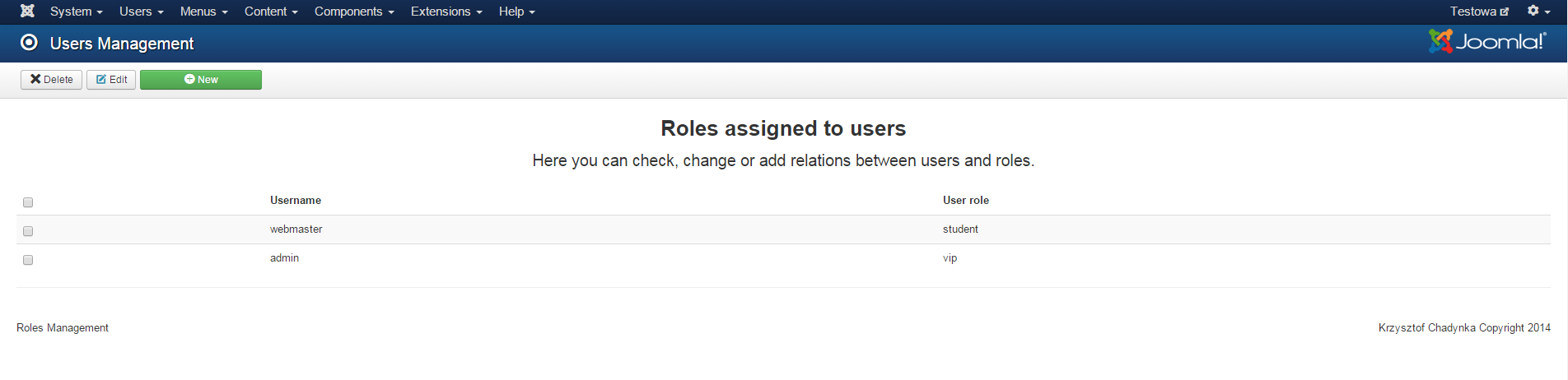
Jak wynika z rysunku, w menu głównym framework Joomla! dostępne są cztery elementy menu podrzędnego, z których każdy odnosi się do podstrony odpowiedzialnej za fragment funkcjonalności komponentu. Zestawienie tych funkcjonalności zostało przedstawione w tabeli 3.

|  |  |
| --- | --- |
| **Pozycja w menu (Podstrona)** | **Funkcjonalność** |
| Users list (Users management) | Wyświetla listę użytkowników do których zostały przyporządkowane strony. Umożliwia dodawanie nowych powiązań oraz usuwanie i edycję już istniejących. |
| Roles list (Roles management) | Wyświetla listę dostępnych ról, które można przypisać do zarejestrowanych użytkowników. Umożliwia dodawanie nowych pozycji oraz usuwanie i edycję już istniejących. |
| Categories list (Categories assigned to roles) | Wyświetla powiązania pomiędzy rolami, a kategoriami artykułów, czyli informacje o tym do których kategorii mają dostęp użytkownicy z odpowiednią rolą. Umożliwia dodawanie nowych powiązań oraz usuwanie i edycję już istniejących. |
| Settings (Frontend settings) | Wyświetla ustawienia informacji wyświetlanych na stronie w widoku informacji o użytkowniku. Umożliwia edycję ustawień. |

Tabela 4 Zestawienie podstron i ich funkcjonalności - panel administracyjny

W dalszej części podpunktu została opisana obsługa poszczególnych podstron zestawionych w tabeli 3.

1. ***Users management***

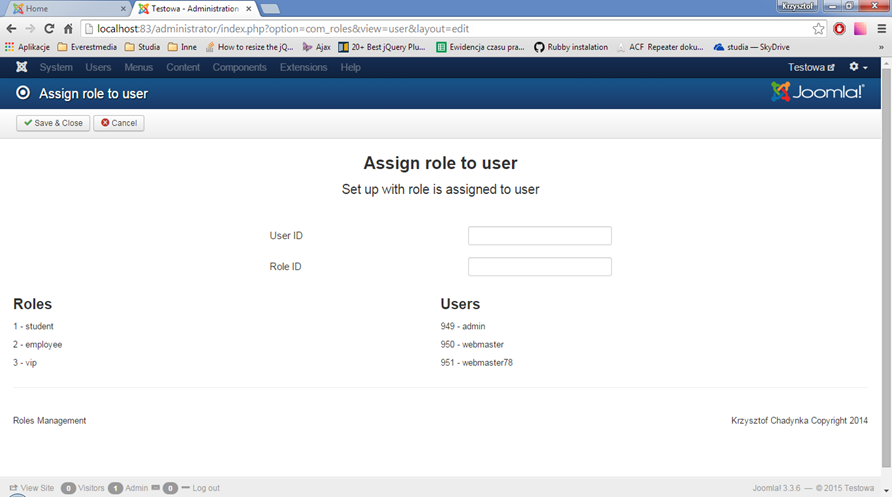
****

Rysunek 10 Główny widok podstrony Users management - panel administracyjny

Zamieszczony na rysunku 10 zrzut ekranu prezentuje główny widok podstrony, czyli listę powiązań użytkowników z rolami.

W owym widoku dostępne są trzy akcje:

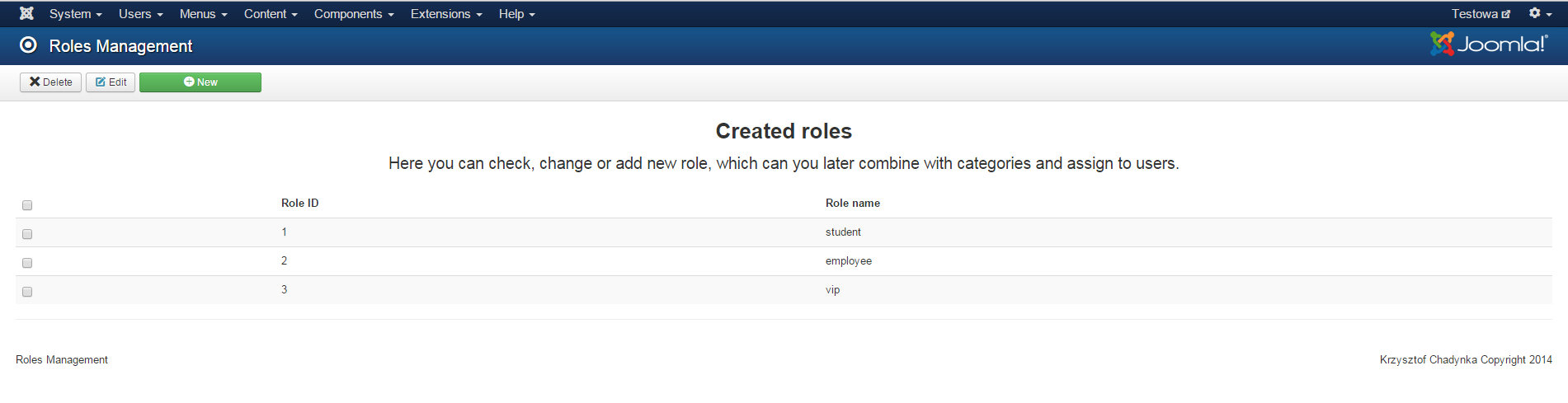
* 1. Usuwanie (należy zaznaczyć element, a następnie wybrać opcję *Delete*);
  2. Edycja (należy zaznaczyć element, a następnie wybrać opcję *Edit*);
  3. Dodawanie (należy wybrać opcję *New*)



Rysunek 11 Dodawanie nowej roli

Szczegółowy widok ekranu w trakcie dodawania nowej roli przedstawiono na rysunku 11.

1. ***Roles list***

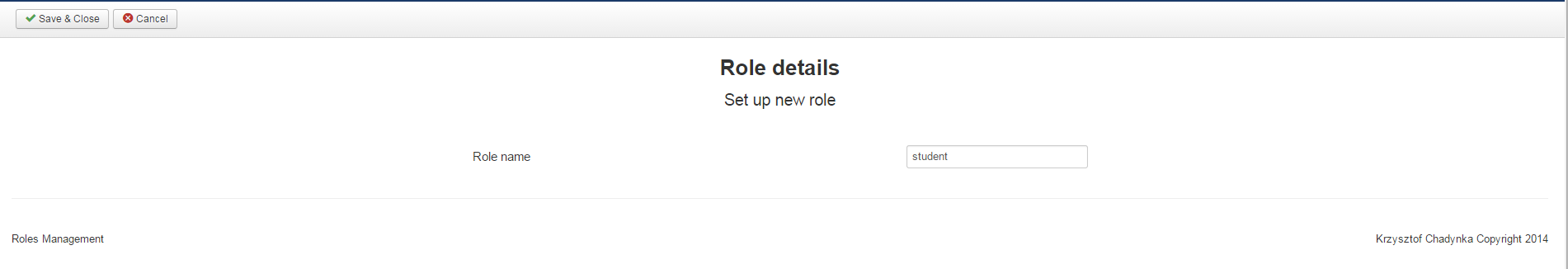
****

Rysunek 12 Główny widok podstrony Roles Management

Jak zostało udokumentowane na rysunku 12, główny widok prezentuje listę dostępnych ról.

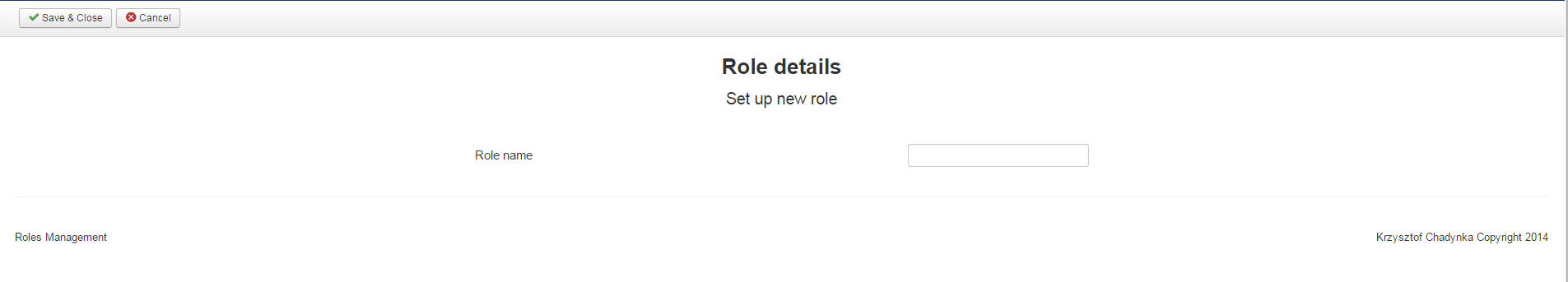
Dostępne są trzy akcje:

* 1. Usuwanie (należy zaznaczyć element, a następnie wybrać opcję *Delete*);
  2. Edycja (należy zaznaczyć element, a następnie wybrać opcję *Edit*);



Rysunek 13 Edycja istniejących roli - panel administracyjny

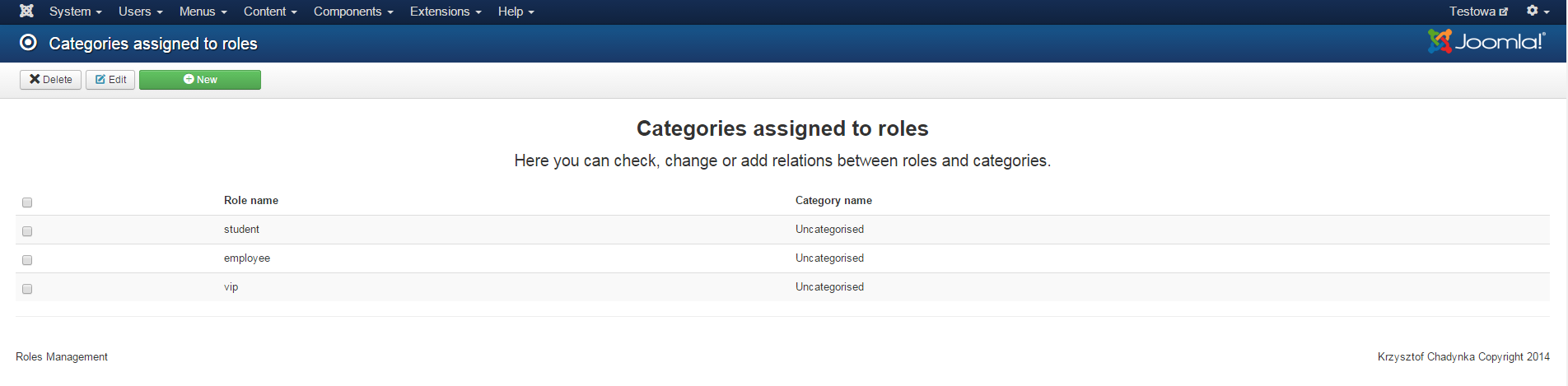
* 1. Dodawanie nowej roli (należy wybrać opcję New)



Rysunek 14 Dodawanie nowej roli - panel administracyjny

Na rysunkach 13 i 14 przedstawiono widok ekranu w trakcie procesu edycji i dodawania nowej roli.

1. ***Categories assigned to roles***

****

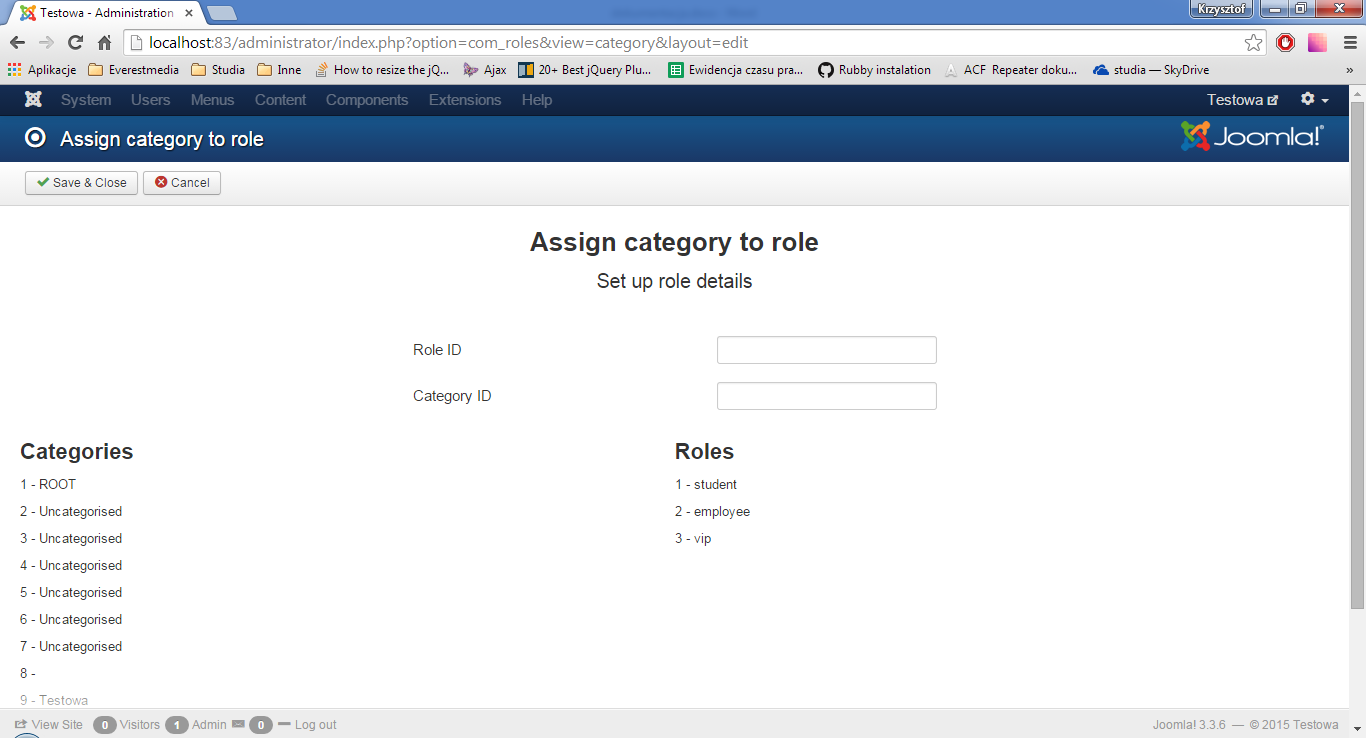
Rysunek 15 Główny widok Categories assigned to roles - panel administracyjny

Rysunek 15 obrazuje główny widok prezentujący listę dostępnych powiązań pomiędzy rolami a kategoriami.

Dostępne są trzy akcje:

* 1. Usuwanie (należy zaznaczyć element, a następnie wybrać opcję *Delete*)
  2. Edycja (należy zaznaczyć element, a następnie wybrać opcję *Edit*)
  3. Dodawanie nowego powiązania pomiędzy rolą, a kategorią (Należy wybrać opcję *New)*

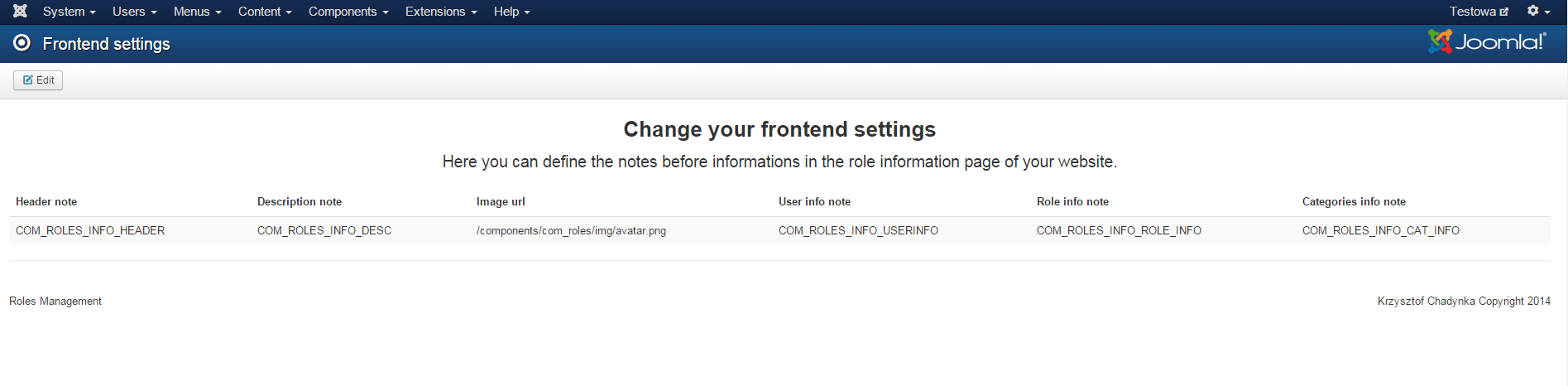
Rysunek 16 ilustruje proces dodawania nowego powiązania między rolą a  kategorią.



Rysunek 16 Dodawanie powiązania pomiędzy rolą, a kategorią - panel administracyjny

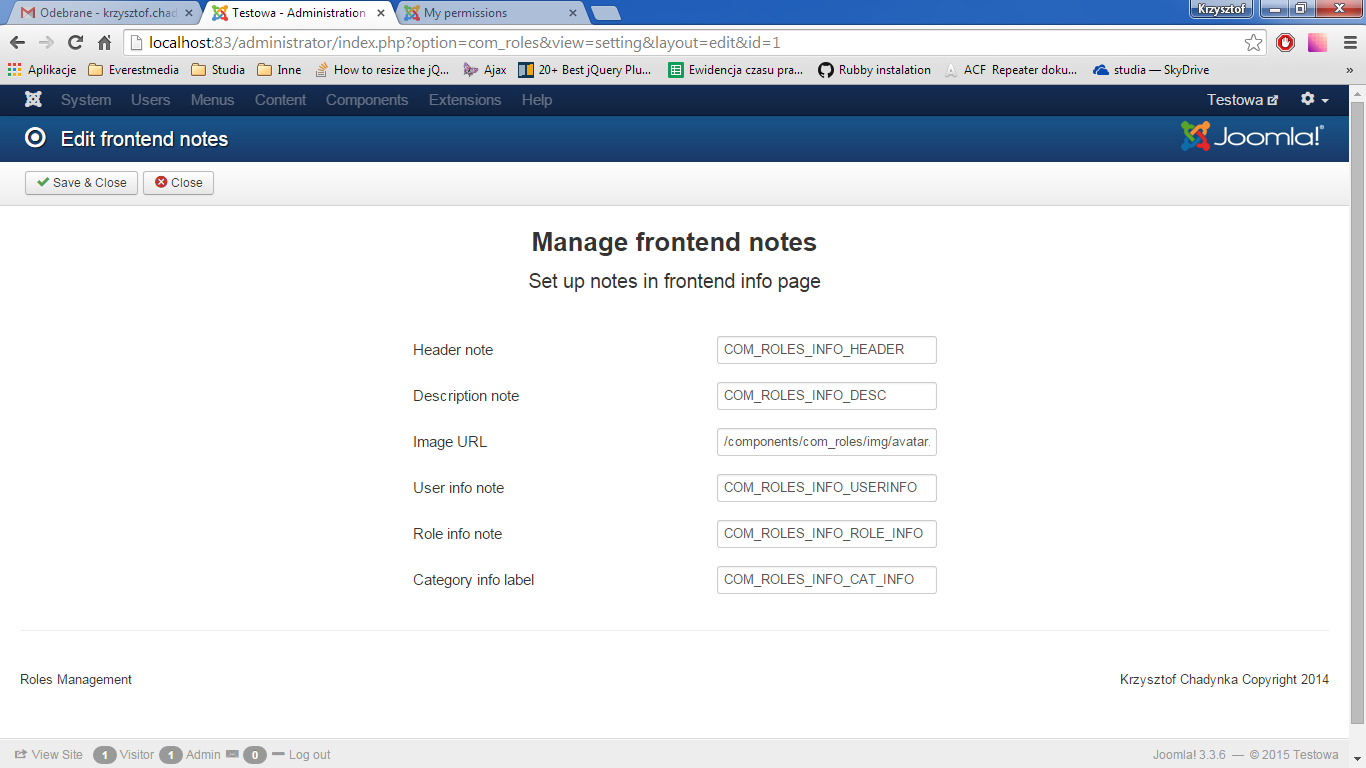
1. ***Frontend settings***

Jak zostało już zestawione wspomniane w tabeli 3, podstrona *Frontend settings* daje możliwość edycji treści w widoku szczegółów użytkownika korzystającego z administrowanej witryny. Na rysunku 17 przedstawiono zestawienie treści wyświetlonych po frontowej stronie komponentu opracowanego w ramach projektu.



Rysunek 17 Główny widok podstrony Frontend settings - panel administracyjny

W tej części programu dostępna jest opcja edycji poszczególnych treści. W tym celu należy wybrać opcję Edit. Edycję treści frontowej komponentu ilustruje rysunek  18.

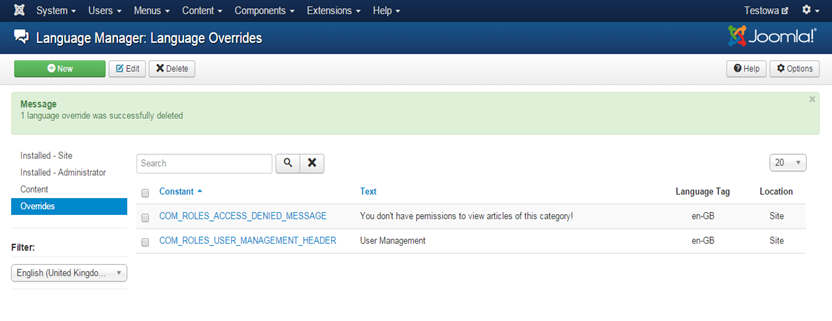


Rysunek 18 Edycja treści w widoku szczegółów użytkownika - panel administracyjny

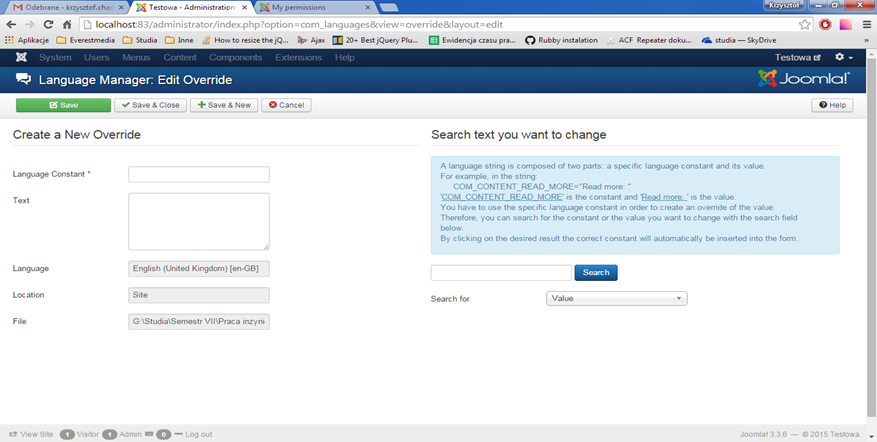
Każda treść (poza adresem url zdjęcia) została zapisana jako sekwencja, która w  następnej kolejności będzie przetłumaczona przez System Tłumaczeń framework’u Joomla!. Rysunek 19 ilustruje zestawienie dostępnych tłumaczeń. W celu wykonania tłumaczenia należy dodać daną treść i jej tłumaczenie do bazy poprzez wykonanie następujących kroków:

1. W górnym menu wybrać *Extension->Language Manager;*
2. Z bocznego menu po lewej stronie wybrać opcję Overrides;
3. Wybrać opcję New;
4. W polu Language Constant należy wpisać sekwencję przeznaczoną do przetłumaczenia, natomiast w polu Text zostaje wprowadzony tekst, które będzie tłumaczeniem.

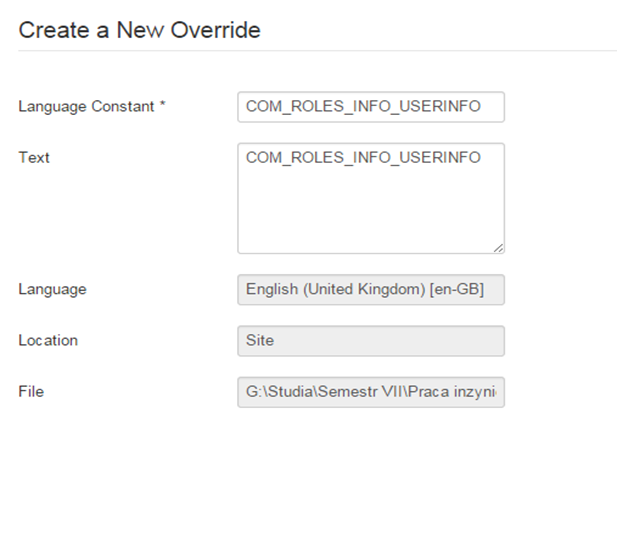
Kroki 2, 3 i 4 procesu tłumaczenia zaprezentowano na rysunkach 20 i 21.



Rysunek 19 Dostępne tłumaczenia - panel administracyjny



Rysunek 20 Dodawanie nowego tłumaczenia - panel administracyjny

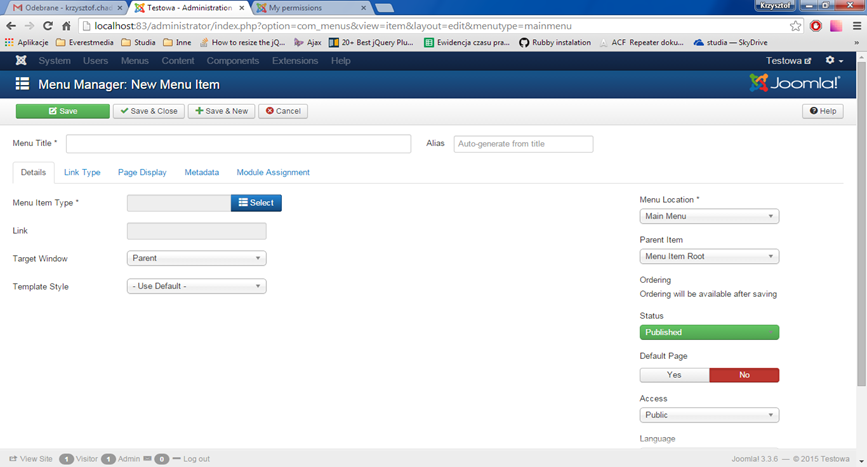


Rysunek 21 Wprowadzanie nowgo tłumaczenia - panel administracyjny

Można oczywiście pominąć krok tłumaczenia, dodając do bazy bezpośrednie treści. Uniemożliwi to jednak automatyczne tłumaczenie strony w przypadku witryny wielojęzycznej.

Aby podstrona ze szczegółami użytkownika została wyświetlona w frontowej części witryny, należy dodać nową pozycję w menu.

W tym celu należy wybrać z menu głównego pozycję *Menus -> (jedno z dostępnych Menu) -> Add new Menu Item*.

**

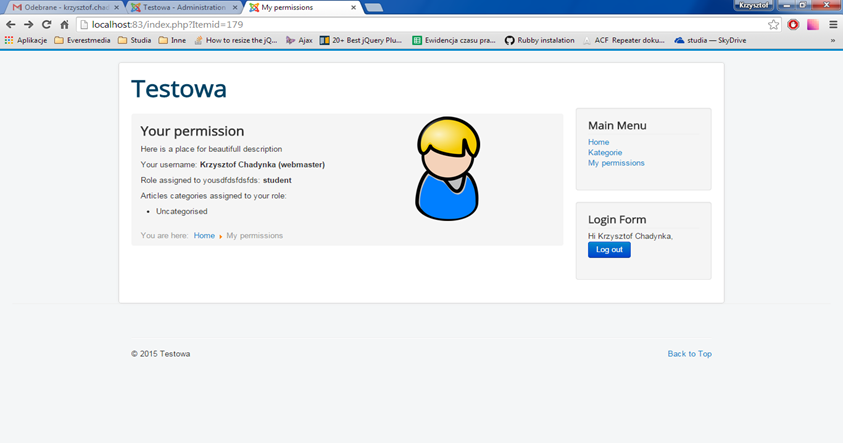
Rysunek 22 Dodanie nowej pozycji w menu - panel administracyjny

W polu Menu Item Type wybiera się opcję roles -> User info, natomiast w polu Menu Title zostaje wpisana dowolna treść, która będzie wyświetlana jako odnośnik we  frontowej części witryny. Widok ekranu tworzenia nowej pozycji menu zamieszczono na rysunku 22.

## Część frontowa witryny

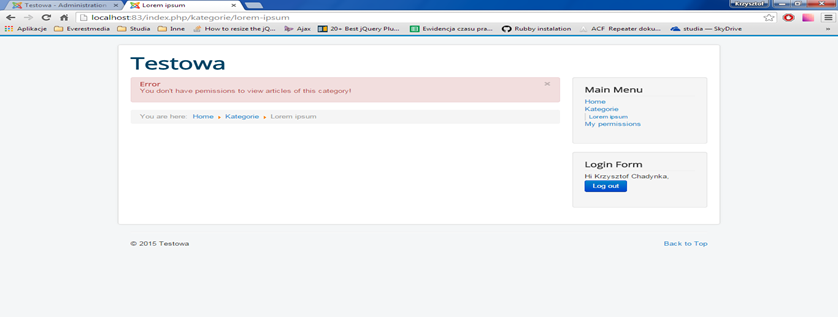
1. **Informacje na temat użytkownika**

Po wyborze opcji z menu (dodanej uprzednio przez administratora witryny), użytkownik zostaje przekierowany na stronę informacji na temat swoich uprawnień, widoczną na rysunku 23.



Rysunek 23 Podstrona informacji o użytkowniku - część frontowa

1. **Informacja o nieudanej autoryzacji**



Rysunek 24 Informacja o nieudanej autoryzacji - część frontowa

W przypadku braku uprawnień, zostanie wyświetlony panel informujący o nieudanej autoryzacji, co ilustruje rysunek 24.

# Podsumowanie i wnioski

## Podsumowanie

Celem projektu było opracowanie i zaimplementowanie funkcjonalności umożliwiającej administratorowi witryny internetowej zarządzanie w sposób łatwy i przejrzysty rolami użytkowników, polegające na przydzielaniu im odpowiednich uprawnień.

W ramach projektu, zgodnie z założeniami, wykonano komponent, pozwalający na autoryzacje użytkowników zarejestrowanych w witrynie opartej na framework’u Joomla!, zatem cel projektu zrealizowano w przewidzianym zakresie.

Na przygotowanie działającej aplikacji złożyły się następujące czynności:

* Opracowano strukturę bazy danych, w ramach czego utworzono nowe tabele, opracowano zależności między nimi, a także opracowano zależności z tabelami istniejącymi już w systemie;
* Zaprojektowano algorytm autoryzacji;
* Utworzono zaplecze administracyjne komponentu;
* Utworzono część frontową komponentu;
* Zastosowano system tłumaczeń w obydwu częściach;
* Zaimplementowano funkcjonalność autoryzacji.

## Wnioski

Wybór framework’u Joomla jako systemu, na którym oparto tworzoną funkcjonalność okazał się wyborem trafnym. Implementacja wzorca MVC w systemie jest przemyślana i logiczna.

Projekt ma swoją perspektywę rozwoju. W przyszłości planowane jest zaimplementowanie następujących dodatkowych funkcjonalności:

* Wysyłanie zapytań przez użytkownika do administratora systemu z prośbą o  przydzielenie mu odpowiednich uprawnień;
* Zakup licencji przez użytkownika obejmującej pojedynczą rolę lub zestaw ról;
* Połączenie z systemem PayPal lub podobnym, w celu łatwego dokonywania płatności za licencje.

# Literatura

|  |  |
| --- | --- |
| [1] | D. Rahmel, Joomla! Profesjonalne tworzenie stron WWW, Helion, 2010. |
| [2] | M. Mroczek, „http://ai.ia.agh.edu.pl/wiki/\_media/pl:miw:2007:miw07-softeng\_mvc.pdf,” [Online]. [Data uzyskania dostępu: 21 styczeń 2015]. |
| [3] | „Elektroniczna Biblioteka Dokumentacji Joomla,” [Online]. Available: www.wiki.joomla.pl. [Data uzyskania dostępu: 06 styczeń 2015]. |
| [4] | „Joomla! Documentation,” [Online]. Available: www.docs.joomla.org. [Data uzyskania dostępu: 04 styczeń 2015]. |

# Ikonografia

**SPIS ILUSTRACJI**

[Rysunek 1 Schemat wzorca MVC 7](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\Dokumentacja%20-%20wersja%20finalna.docx#_Toc409913085)

[Rysunek 2 Schemat blokowy procesu autoryzacji 9](#_Toc409913086)

[Rysunek 3 Schemat bazy danych 12](#_Toc409913087)

[Rysunek 4 Instalacja komponentu - logowanie do panelu administracyjnego 26](#_Toc409913088)

[Rysunek 5 Instalacja komponentu - przejście do odpowiedniej zakładki 26](#_Toc409913089)

[Rysunek 6 Instalacja komponentu - wybór pliku 27](#_Toc409913090)

[Rysunek 7 Instalacja komponentu - informacja o pozytywnym wyniku instalacji 27](#_Toc409913091)

[Rysunek 8 Przesłanie pliku controller.php na serwer 28](#_Toc409913092)

[Rysunek 9 Położenie komponentu w menu głównym - panel administracyjny 28](#_Toc409913093)

[Rysunek 10 Główny widok podstrony Users management - panel administracyjny 30](#_Toc409913094)

[Rysunek 11 Dodawanie nowej roli 30](#_Toc409913095)

[Rysunek 12 Główny widok podstrony Roles Management 31](#_Toc409913096)

[Rysunek 13 Edycja istniejących roli - panel administracyjny 31](#_Toc409913097)

[Rysunek 14 Dodawanie nowej roli - panel administracyjny 31](#_Toc409913098)

[Rysunek 15 Główny widok Categories assigned to roles - panel administracyjny 32](#_Toc409913099)

[Rysunek 16 Dodawanie powiązania pomiędzy rolą, a kategorią - panel administracyjny 32](#_Toc409913100)

[Rysunek 17 Główny widok podstrony Frontend settings - panel administracyjny 33](#_Toc409913101)

[Rysunek 18 Edycja treści w widoku szczegółów użytkownika - panel administracyjny 33](#_Toc409913102)

[Rysunek 19 Dostępne tłumaczenia - panel administracyjny 34](#_Toc409913103)

[Rysunek 20 Dodawanie nowego tłumaczenia - panel administracyjny 34](#_Toc409913104)

[Rysunek 21 Wprowadzanie nowgo tłumaczenia - panel administracyjny 35](#_Toc409913105)

[Rysunek 22 Dodanie nowej pozycji w menu - panel administracyjny 36](#_Toc409913106)

[Rysunek 23 Podstrona informacji o użytkowniku - część frontowa 37](#_Toc409913107)

[Rysunek 24 Informacja o nieudanej autoryzacji - część frontowa 37](#_Toc409913108)

**SPIS TABEL**

[Tabela 1 Struktura początkowa tabeli JOO\_ROLES\_INFO 11](#_Toc409913109)

[Tabela 2Struktura początkowa tabeli JOO\_ROLES\_ROLE 12](#_Toc409913110)

[Tabela 3 Zasady dotyczące nazywania klas - panel administracyjny 17](#_Toc409913111)

[Tabela 4 Zestawienie podstron i ich funkcjonalności - panel administracyjny 29](#_Toc409913112)

**SPIS LISTING-ÓW KODU**

[Listing 1 Ograniczenie dostępu do pliku 14](#_Toc409913113)

[Listing 2 plik roles.xml 15](#_Toc409913114)

[Listing 3 plik site/language/en-GB/en-GB.com\_roles.ini 16](#_Toc409913115)

[Listing 4 plik install.mysql.utf8.sql 18](#_Toc409913116)

[Listing 5 klasa RolesControllerCategory 18](#_Toc409913117)

[Listing 6 klasa RolesModelCategories 19](#_Toc409913118)

[Listing 7 klasa RolesViewCategories 20](#_Toc409913119)

[Listing 8 klasa RolesModelCategory 21](#_Toc409913120)

[Listing 9 funkcja getCurrentCategoryID 22](#_Toc409913121)

[Listing 10 funkcja GetCurrentRoleID 22](#_Toc409913122)

[Listing 11 funkcja checkIfCategoryExists 23](#_Toc409913123)

[Listing 12 Weryfikacja autoryzacji 23](#_Toc409913124)