

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 authentication 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. Cel projektu 4](#_Toc409692743)

[2. Zakres projektu 5](#_Toc409692744)

[2.1. Analiza przedmiotu projektu 5](#_Toc409692745)

[3. Założenia projektu 6](#_Toc409692746)

[3.1. Wykorzystywany język programowania 6](#_Toc409692747)

[3.2. Model MVC 6](#_Toc409692748)

[3.3. Wykorzystywane środowisko 7](#_Toc409692749)

[3.4. Wykorzystywany system bazy danych 7](#_Toc409692750)

[4. Dobór rozwiązań 8](#_Toc409692751)

[5. Część projektowa 9](#_Toc409692752)

[5.1 Opracowanie algorytmu 9](#_Toc409692753)

[5.1.1 Proces autoryzacji 9](#_Toc409692754)

[5.2 Opracowanie struktury bazy danych 10](#_Toc409692755)

[6. Implementacja 13](#_Toc409692756)

[6.1. Struktura plików i katalogów 13](#_Toc409692757)

[6.2. Ogólne informacje na temat kodu źródłowego 14](#_Toc409692758)

[6.3. Część administracyjna komponentu 16](#_Toc409692759)

[6.4. Część frontowa komponentu 22](#_Toc409692760)

[6.5. Proces autoryzacji 22](#_Toc409692761)

[7. Dokumentacja użytkownika 25](#_Toc409692762)

[7.1. Wymagania sprzętowe 25](#_Toc409692763)

[7.2. Instalacja 25](#_Toc409692764)

[7.3. Zarządzanie aplikacją 28](#_Toc409692765)

[7.4 Część frontowa witryny 36](#_Toc409692766)

[8. Podsumowanie i wnioski 37](#_Toc409692767)

[8.1 Podsumowanie 37](#_Toc409692768)

[8.2 Wnioski 37](#_Toc409692769)

[9. Literatura 38](#_Toc409692770)

[10. Ikonografia 39](#_Toc409692771)

# 1. Cel projektu

Celem niniejszego 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. Do tej pory nie znalazłem dla framework’u Joomla! rozwiązania, które w łatwy i przystępny sposób umożliwiałoby przypisywanie dla użytkowników odpowiednich uprawnień przy korzystaniu z witryny.

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.

# 2. Zakres projektu

## 2.1. Analiza przedmiotu projektu

Na początku tego podrozdziału warto wspomnieć czym właściwie jest autoryzacja. Autoryzacja polega na dodawaniu pewnych uprawnień użytkownikowi, zezwalających na dostęp do pewnych 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. Joomla! przewiduje 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.

Tematyką tworzonego projektu jest prosty interfejs służący do obsługi autoryzacji wyłącznie artykułów.

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

Problem, będący przedmiotem 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 jak najbardziej 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.

# 3. Założenia projektu

## 3.1. Wykorzystywany język programowania

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

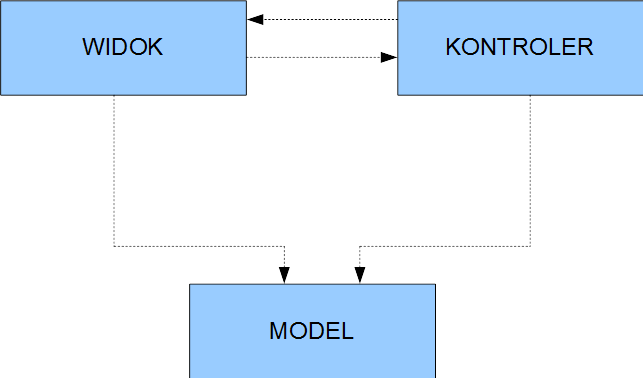
## 3.2. Model MVC

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, reakcja na akcje użytkownika wywołująca akcje zawarte w modelu oraz powodująca zmiany w widoku. 

Rysunek 1 Schemat wzorca MVC

## 3.3. Wykorzystywane środowisko

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

Jako systemu kontroli wersji projektu zdecydowano się użyć repozytorium *GitHub.*

## 3.4. Wykorzystywany system bazy danych

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

# 4. Dobór rozwiązań

W poprzednich rozdziałach zaznaczone zostało, 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. Są również najbardziej wyspecjalizowanymi rozszerzeniami.

Komponent jest najbardziej złożonym rozszerzeniem – 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*).

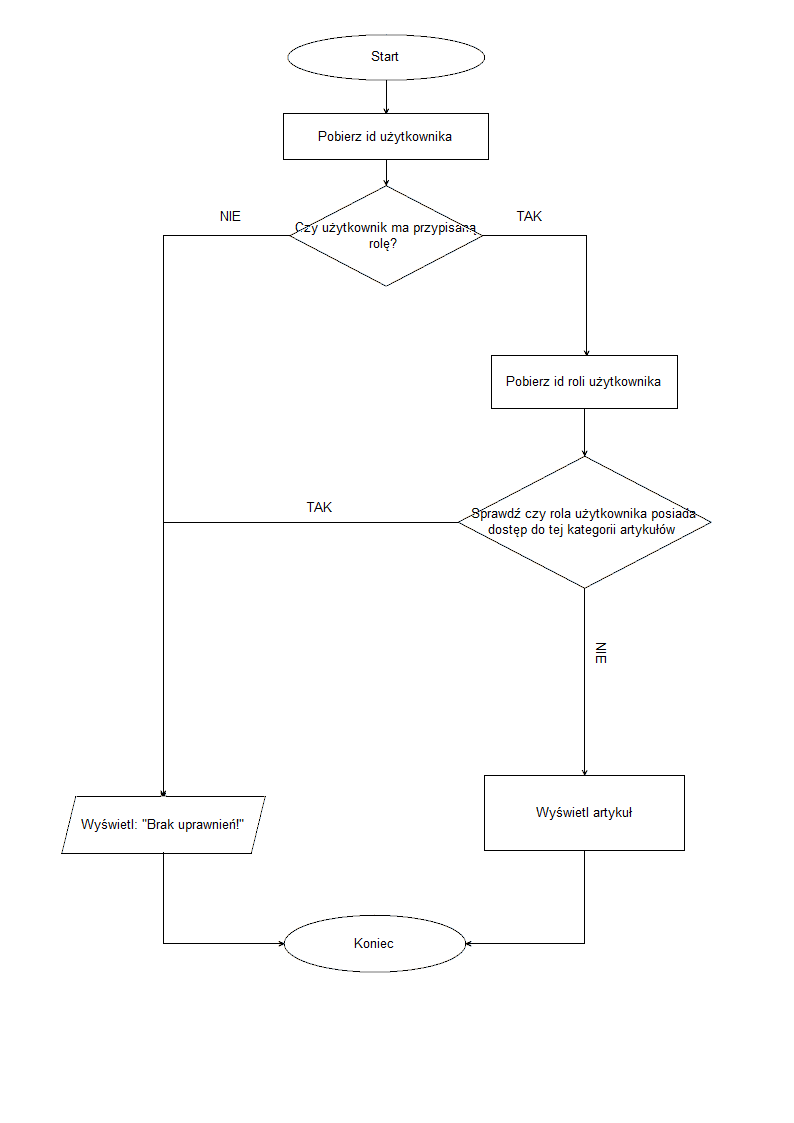
W części prezentacyjnej istnieje możliwość zarejestrowania się zainteresowanego gościa witryny. Po stronie administratora możliwa jest edycja nowo utworzonego konta, a także dodanie użytkownika z tego poziomu.

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

# 5. Część projektowa

## 5.1 Opracowanie algorytmu

### 5.1.1 Proces autoryzacji



Rysunek 2 Schemat blokowy - proces autoryzacji

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

1. Na początku zostaje pobrane ID użytkownika obecnie zalogowanego w naszej witrynie.
2. Na podstawie pobranego ID przeszukujemy odpowiednią tabelę w bazie danych aby sprawdzić, 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 wypadku 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 wypadku wyświetlany jest komunikat o nieudanej autoryzacji i również proces zostaje zakończony.

## 5.2 Opracowanie struktury 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 zostanie 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)**

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:

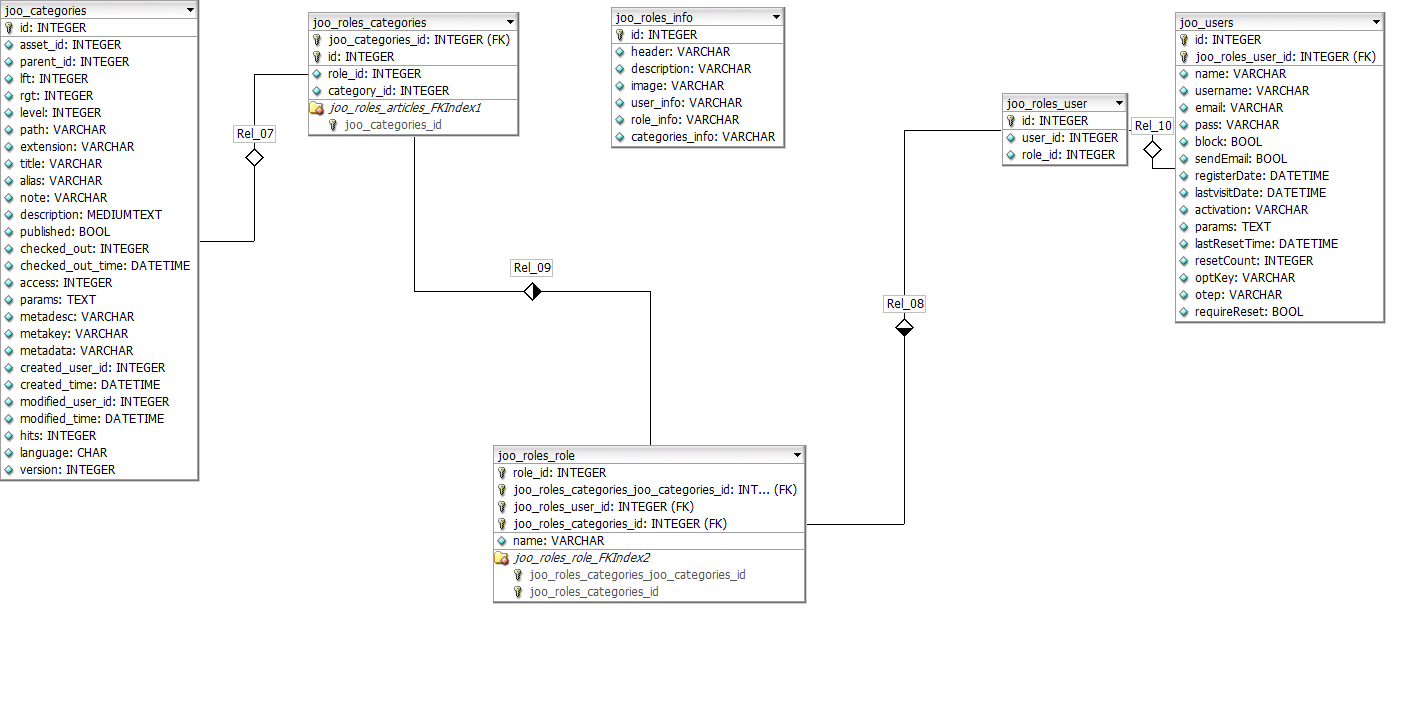
|  |  |
| --- | --- |
| **role\_id** | **name** |
| 1 | student |
| 2 | pracownik |
| 3 | vip |

Tabela 2 Struktura początkowa tabeli joo\_roles\_role

1. **roles\_articles (a właściwie joo\_roles\_articles)**

Jest to tabela przechowująca połączone w pary identyfikatory roli wraz z przypisanymi do nich identyfikatorami kategorii artykułów (pobranych z tabeli joo\_categories), na dostęp do których zezwala dana rola.

Ostatecznie, struktura wykorzystywanych tabel bazy danych (wraz z odpowiednimi relacjami), przedstawia się następująco (*Rysunek 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.

# 6. Implementacja

## 6.1. Struktura plików i katalogów

Poniżej w skróconej formie zostanie przedstawiona struktura plików i katalogów utworzonego komponentu:

* **Główny katalog komponentu (*com\_roles*)**
  + **Część administracyjna komponentu (*admin*)**
    - ***controllers***
* Kontrolery używane w zapleczu administracyjnym
  + - ***css***
      * *style.css* – arkusz styli dla zaplecza administracyjnego
    - ***language***
      * Pliki językowe służące do zarządzania tłumaczeniami
    - ***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 pojedynczy widok (dla przykładu *roles*)
        + ***tmp***

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 styli dla części frontowej komponentu
    - ***img***
      * Obrazki wykorzystywane w części frontowej komponentu
    - ***language*** 
      * Pliki językowe służące do zarządzania tłumaczeniami
    - ***models***
      * Modele używane w części frontowej
    - ***views***
      * **struktura katalogów i plików opisująca pojedynczy widok** (analogicznie do zaplecza administracyjnego)
    - *controller.php* - główny kontroler części frontowej komponentu
    - *roles.php* – główny plik zarządzający częścią frontową
  + Plik konfiguracyjny całego komponentu (*roles.xml*)

## 6.2. 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 Ograniczenie dostępu do pliku

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

Głównym plikiem używanym podczas procesu instalacji jest plik *roles.xml.*

Listing Plik rolex.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>

Powyższy plik przechowuje informacje na temat komponentu, takie jak informacje 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 poniżej (Listing 3)

COM\_ROLES\_INFO\_HEADER="Your permission"

COM\_ROLES\_INFO\_DESC="Here is a place for beautifull description"

COM\_ROLES\_INFO\_USERINFO="Your username"

COM\_ROLES\_INFO\_ROLE\_INFO="Role assigned to you"

COM\_ROLES\_INFO\_CAT\_INFO="Articles categories assigned to your role"

COM\_ROLES\_INFO\_NOT\_LOGGED\_USER="Please login first"

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

## 6.3. 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 tutaj 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.

Powyższe 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.

**Zasady dotyczące nazywania klas:**

* **modele**

NazwaKomponentuModelNazwaModelu

(np. *RolesModelRole*);

* **kontrolery**

NazwaKomponentuControllerNazwaKontrolera

(np. *RolesControllerRole*);

* **klasy widoków**

NazwaKomponentuViewNazwaWidoku

(np. RolesViewRole);

* **klasy tabel**

NazwaKomponentuTableNazwaTabeli

(np. RolesTableRole);

**Wybrane fragmenty kodu:**

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');

Powyższy kod jest zestawem zapytań do bazy danych w trakcie instalacji komponentu.

Powyżej (Listing 5) przedstawiono klasę stanowiącą kontroler listy kategorii.

class RolesControllerCategories extends JControllerAdmin

{

public function getModel($name = 'Category', $prefix = 'RolesModel')

{

return parent::getModel($name, $prefix, array('ignore\_request' => true));

}

}

Listing Klasa RolesControllerCategories

Przedstawiona klasa (Listing 6) jest reprezentacją modelu służącego do pobierania z bazy danych informacji niezbędnych do operacji na kategoriach.

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 Klasa RolesModelCategories

Listing 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);

}

}

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 Klasa RolesModelCategory

## 6.4. Część frontowa komponentu

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

## 6.5. Proces autoryzacji

Cały proces autoryzacji odbywa się w pliku */administrator/components /com\_content/controller.php*, który na potrzeby projektu został zmodyfikowany. Poniżej (Listing 9) zamieszczone zostały zmienione lub dodane na potrzeby autoryzacji fragmenty kodu:

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();

}

Listing Funkcja geturrentCategoryID

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

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();

}

Listing Funkcja getCurrentRoleID

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

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;

}

Listing Funkcja checkIfCategoryExists

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 posiadają 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 wygląda następująco:

Listing 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 opisana wyżej funkcję (Listing 12) i, na podstawie zwróconej przez nią wartości, udostępnia się lub blokuje daną kategorię.

# 7. Dokumentacja użytkownika

## 7.1. Wymagania sprzętowe

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

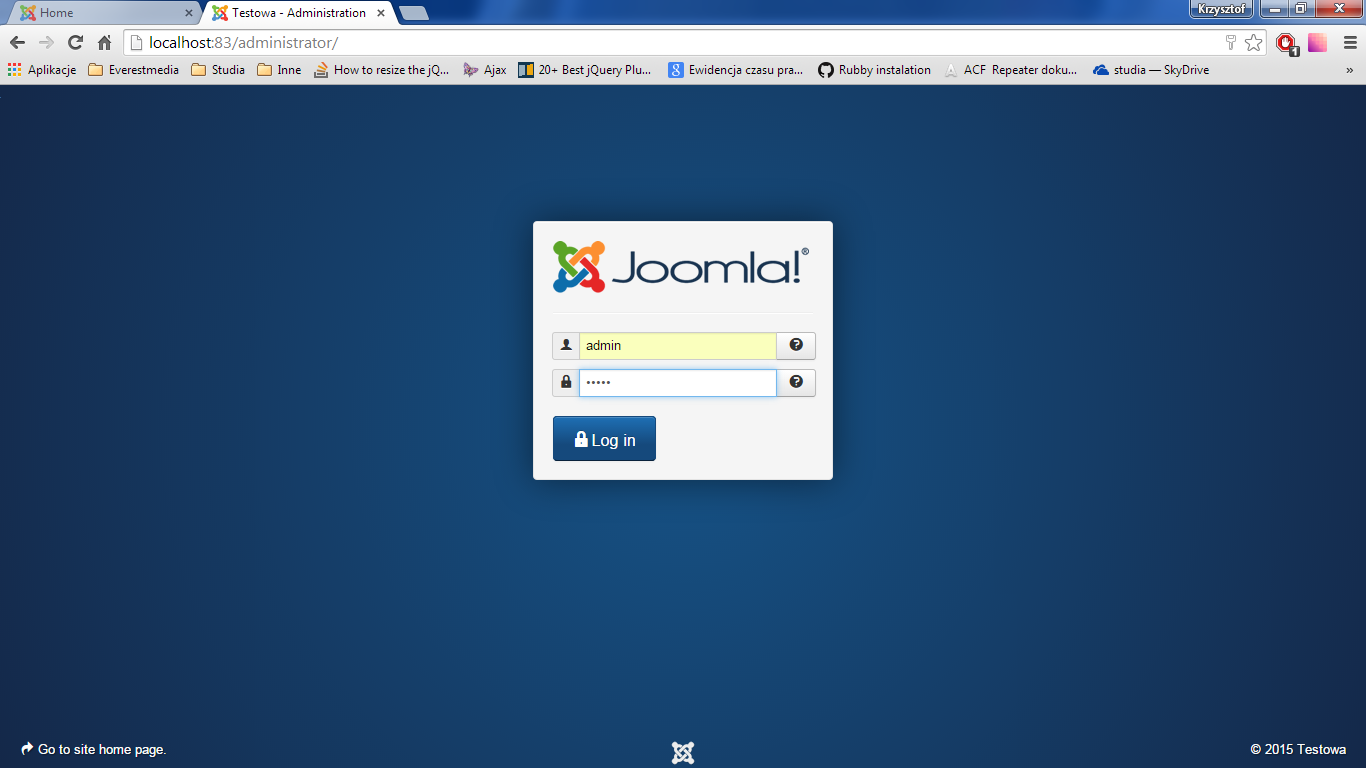
* obsługa PHP w wersji 5.3.10 lub wyższej;
* Obsługa bazy danych w standardzie MySQL 5.1 (lub nowsza);
* Na serwerze musi być zainstalowana witryna internetowa oparta na framework’u Joomla! wersji 3.1 lub wyższej (Proces instalacji szczegółowo opisany jest na stronie producenta – [*www.wiki.joomla.pl*](http://www.wiki.joomla.pl)).

## 7.2. Instalacja

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

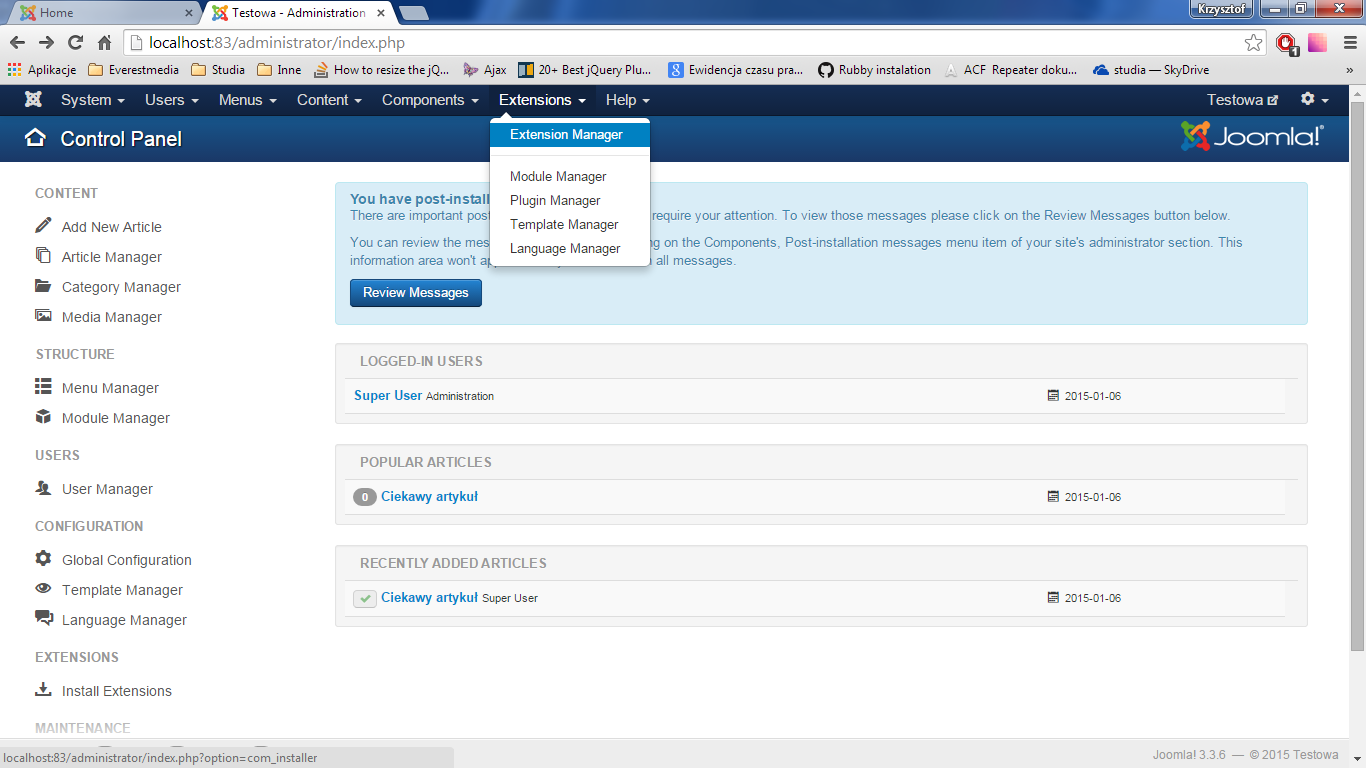
W tym celu należy:

1. zalogować się do panelu administracyjnego, korzystając z danych podanych podczas instalacji Joomla!. :



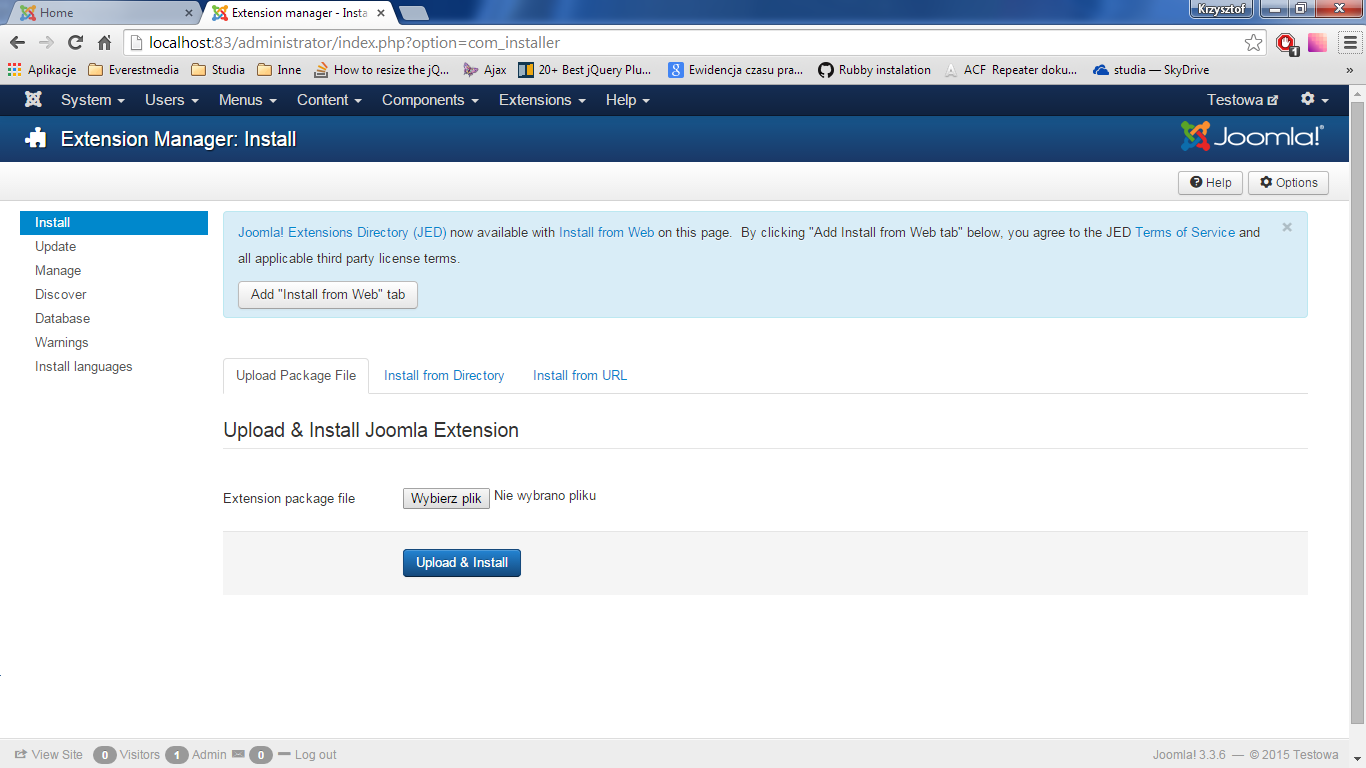
Rysunek 4 Instalacja komponentu - logowanie do panelu administracyjnego

1. Wybrać z górnego menu pozycję Extensions -> Extension Manager:



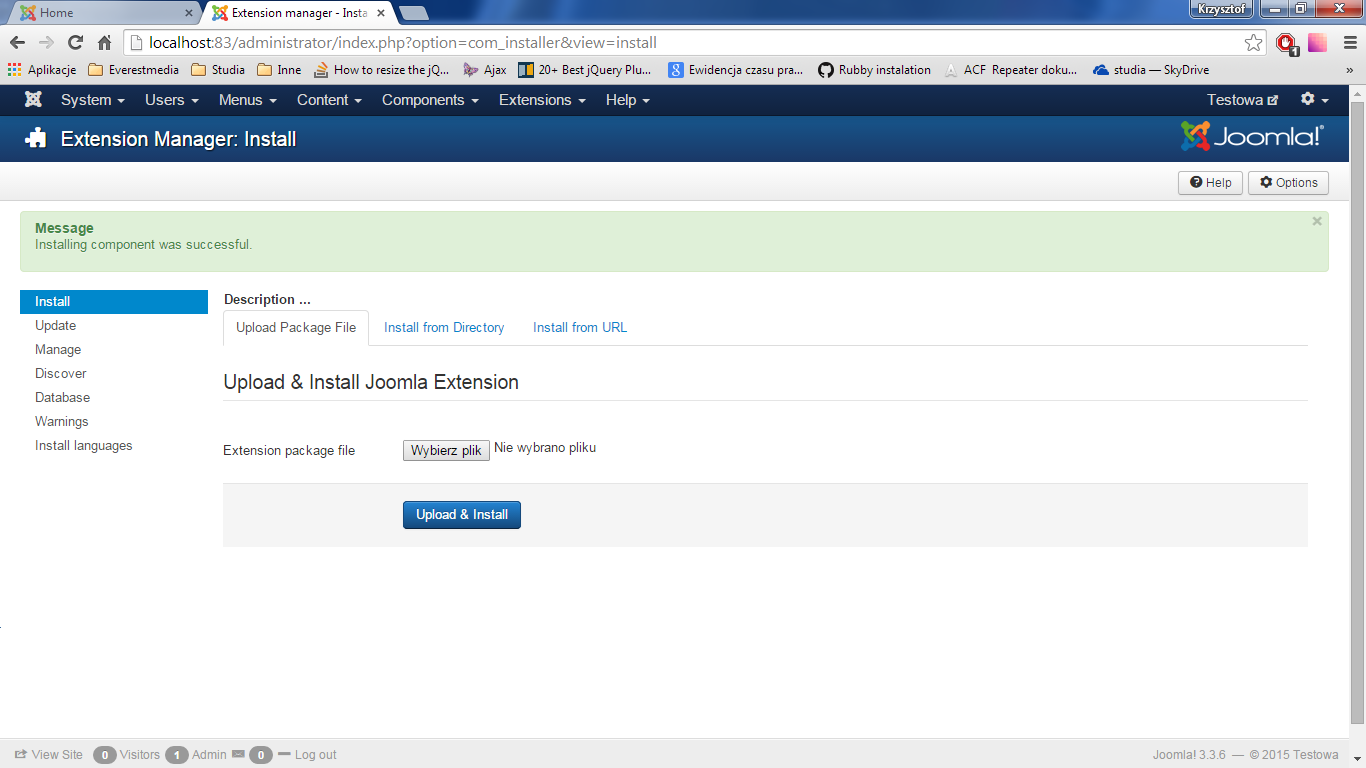
Rysunek 5 Instalacja komponentu - przejście do odpowiedniej zakładki

1. Wybrać plik z dysku (komponent *com\_roles* w formacie .zip), po czym kliknąć przycisk *Upload & Install*:



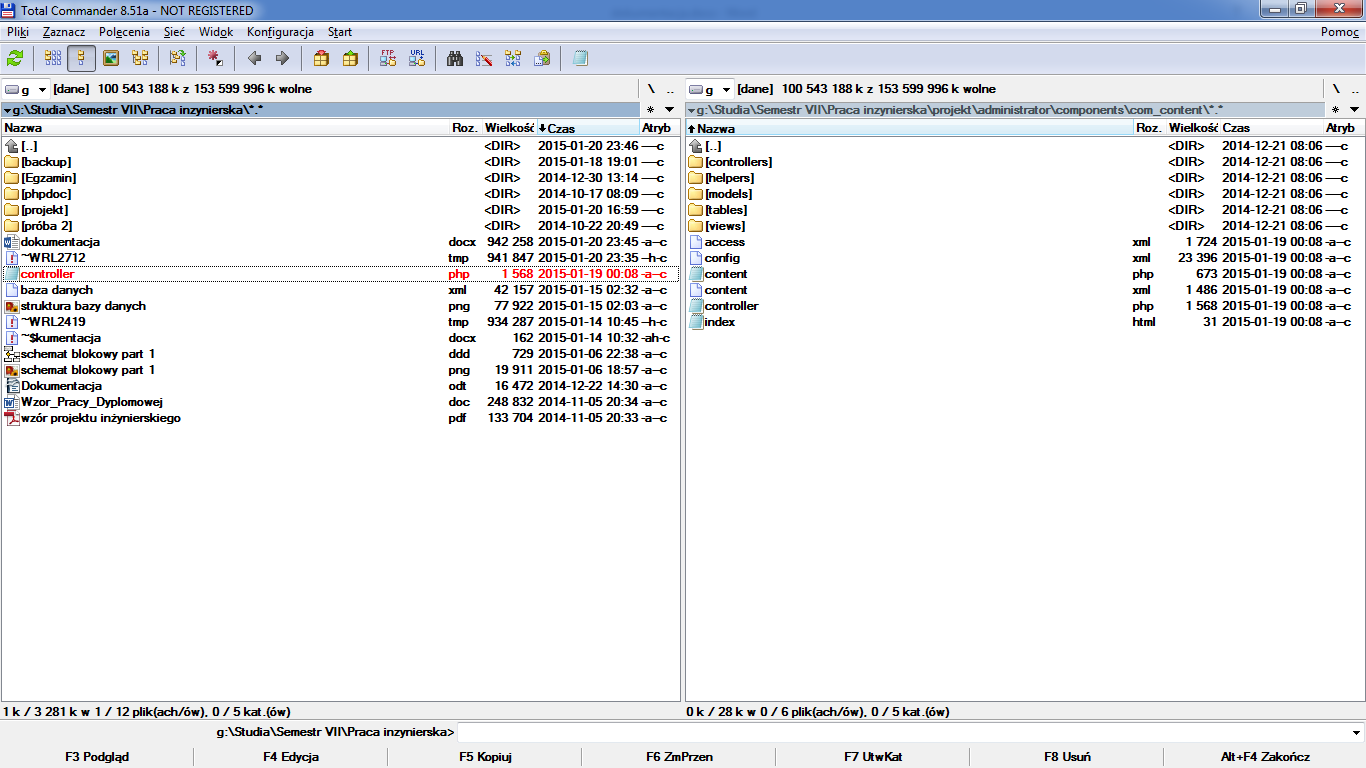
Rysunek 6 Instalacja komponentu - wybór pliku

1. Jeżeli podczas instalacji nie wystąpił żaden błąd, w górnym obszarze strony pojawi się komunikat informujący o zakończonej sukcesem instalacji komponentu:



Rysunek 7 Instalacja komponentu - informacja o pozytywnym wyniku instalacji.

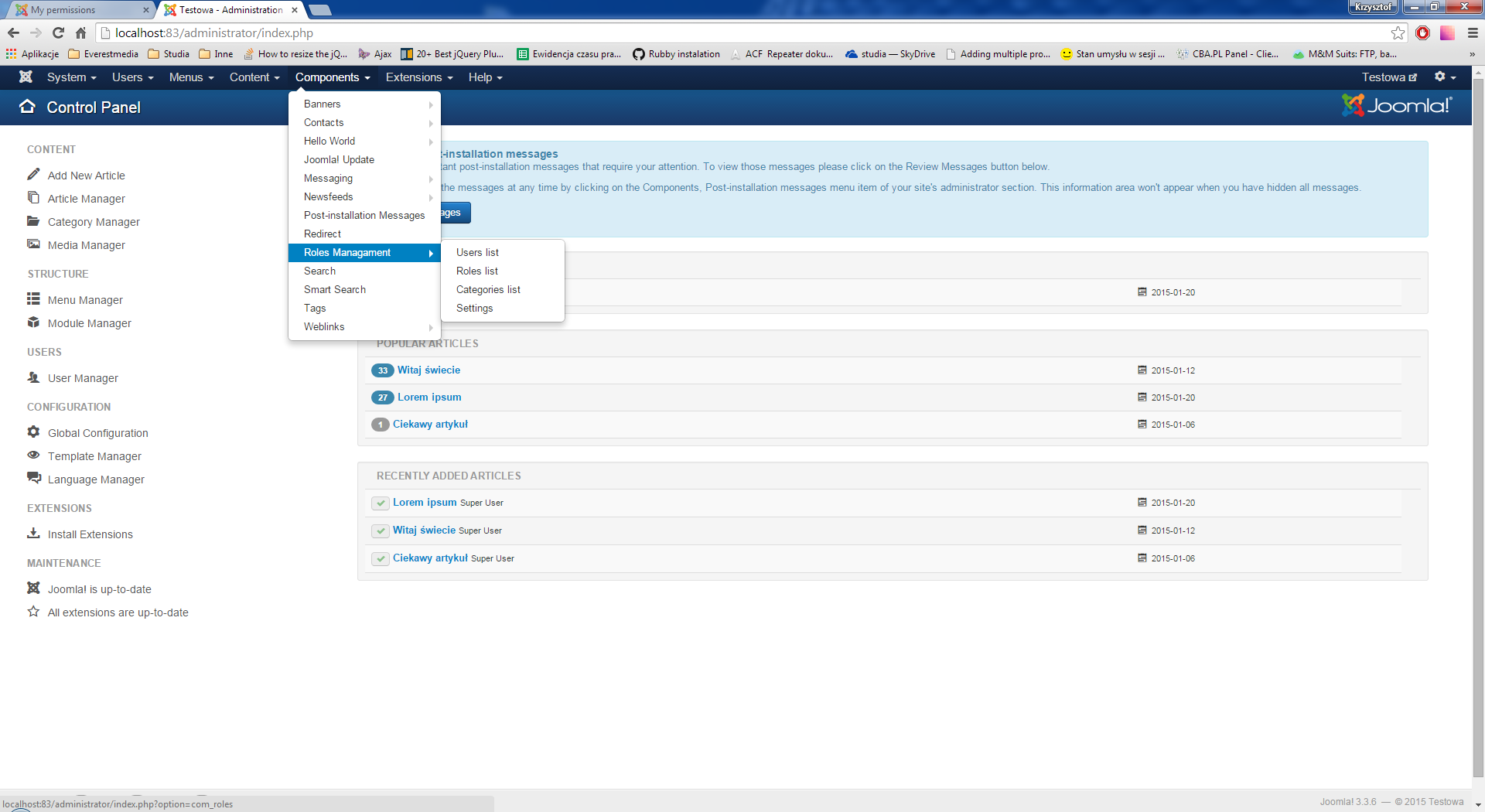
1. Po zakończonej instalacji należy, za pomocą protokołu narzędzia obsługującego protokół FTP (np. Total Commander), przesłać na zdalny serwer plik *controller.php (*plik ten znajduje się także na płycie cd dołączonej do projektu) do katalogu *administrator/components/com\_content*. Spowoduje to nadpisanie 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.



Rysunek 8 Przesłanie pliku controller.php na zdalny serwer

## 7.3. Zarządzanie aplikacją

Po udanej instalacji komponentu, w menu głównym, w zakładce *Components,* pojawi się dodatkowa pozycja *Roles Management*



Rysunek 9 Położenie komponentu w menu głównym - panel administracyjny

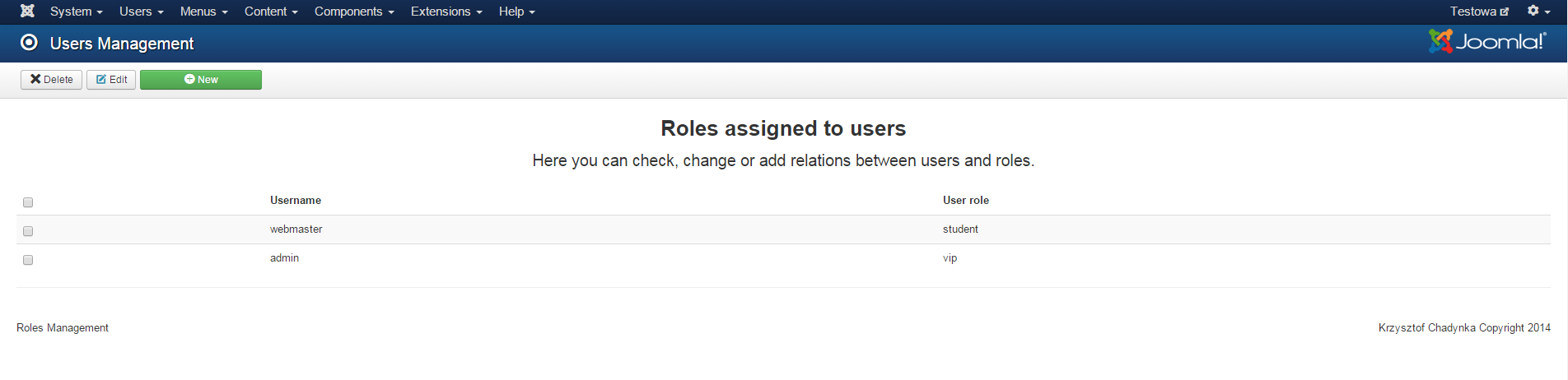
Jak zostało udokumentowane na powyższym zrzucie ekranu (*Rysunek 9),* dostępne są cztery elementy menu, z których każdy odnosi się do podstrony odpowiedzialnej za fragment funkcjonalności komponentu. Zestawienie tych funkcjonalności zostało przedstawione w tabeli:

|  |  |
| --- | --- |
| **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 3 Zestawienie podstron i ich funkcjonalności - panel administracyjny

**Obsługa poszczególnych podstron:**

1. **Users management**

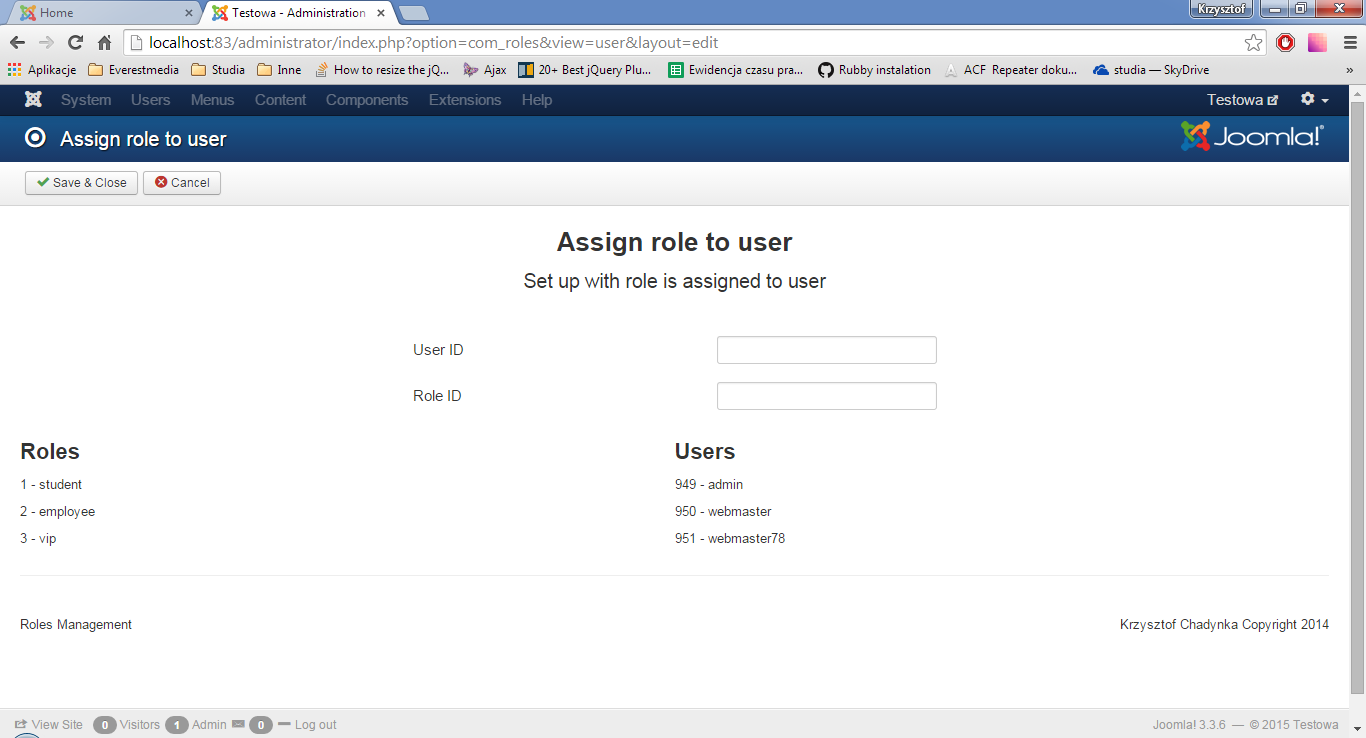
****

Rysunek 10 Główny widok podstrony Users Management – panel administracyjny

Powyższy zrzut ekranu (Rysunek 10) prezentuje główny widok podstrony, czyli listę powiązań użytkowników z rolami.

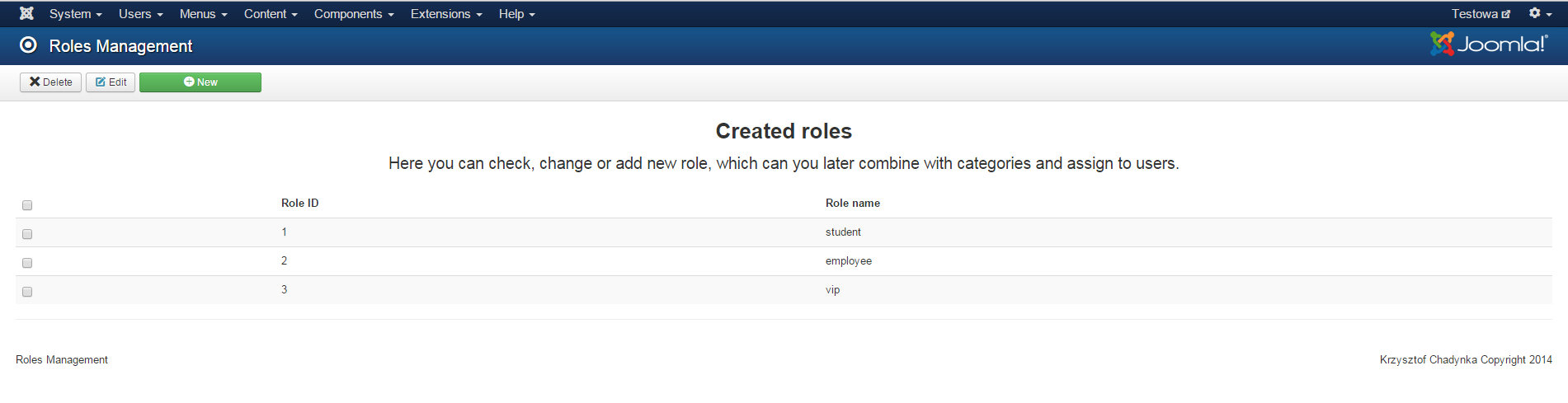
W owym widoku dostępne są trzy akcje:

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



Rysunek Edycja / dodawanie nowej roli

1. **Roles list**

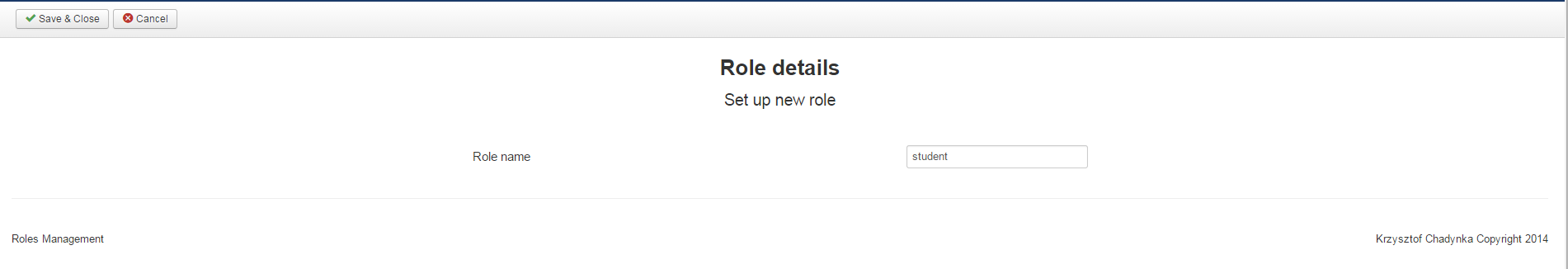
****

Rysunek 12 Główny widok podstrony Roles management – panel administracyjny

Jak zostało udokumentowane na powyższym zrzucie ekranu (Rysunek 12), główny widok prezentuje listę dostępnych ról.

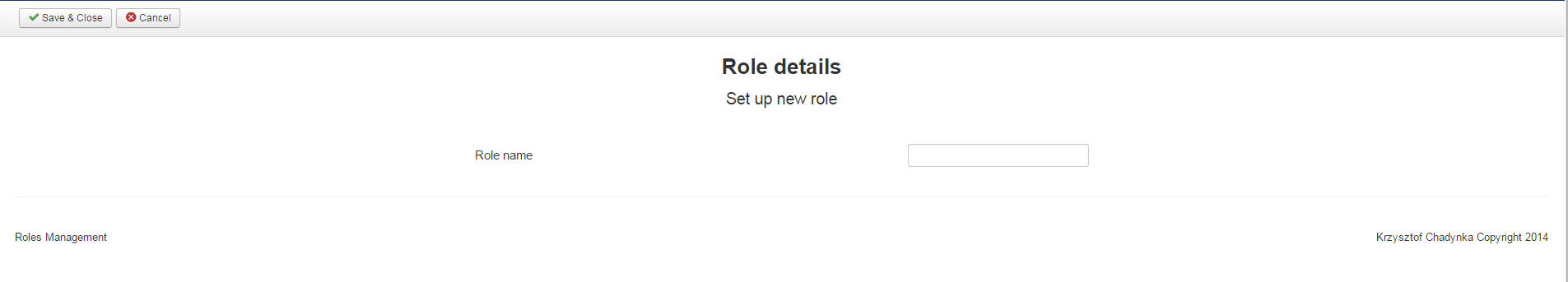
Dostępne są trzy akcje:

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



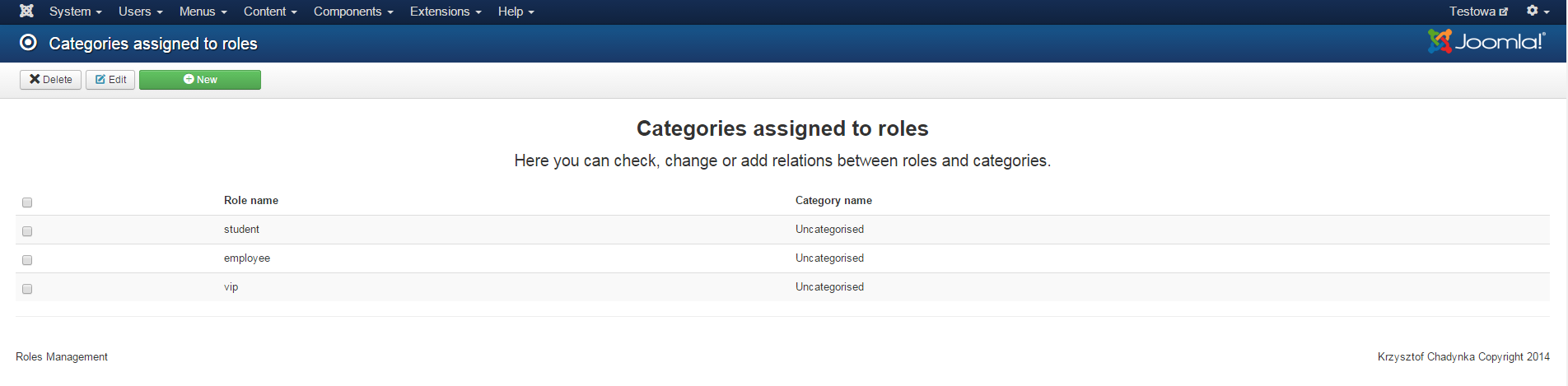
Rysunek 13 Edycja istniejącej roli – panel administracyjny

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



Rysunek 14 Dodawanie nowej roli – panel administracyjny

1. **Categories assigned to roles**

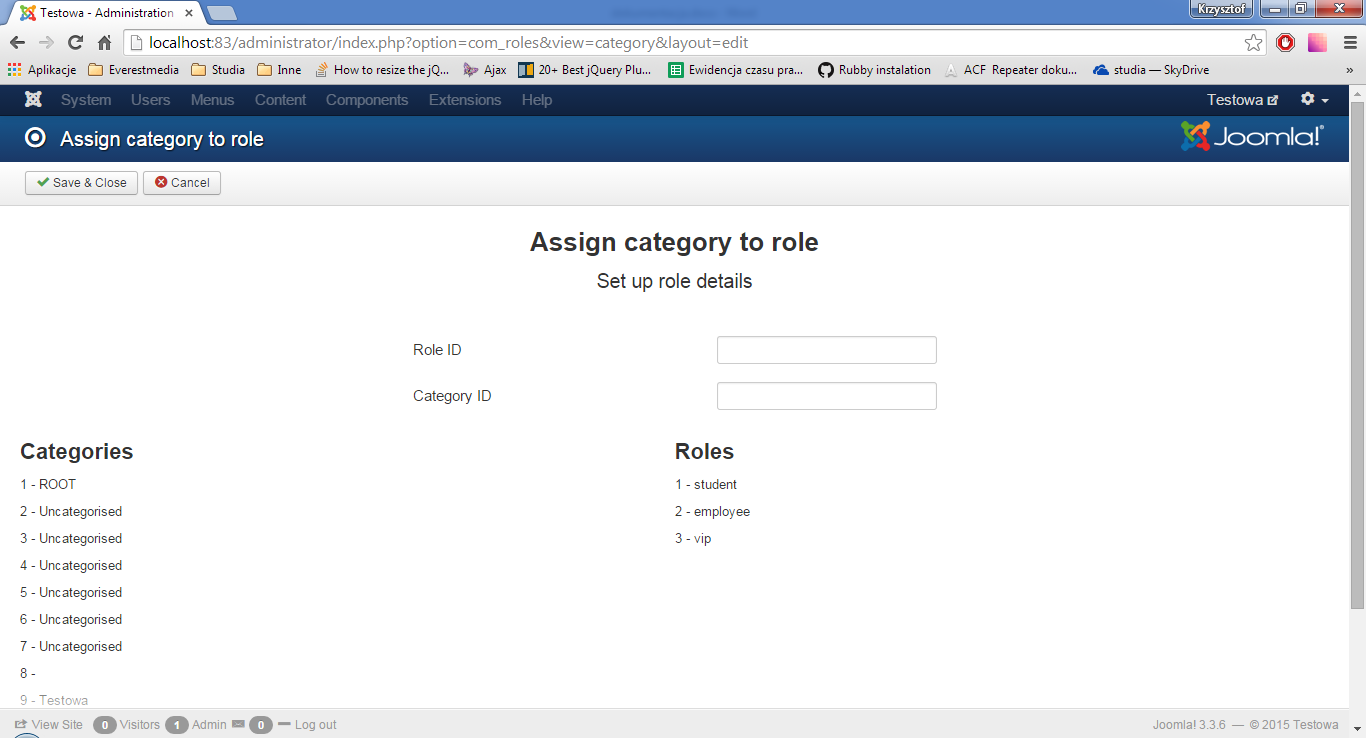
****

Rysunek 15 Główny widok podstrony Categories assigned to roles - panel administracyjny

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

Dostępne są trzy akcje:

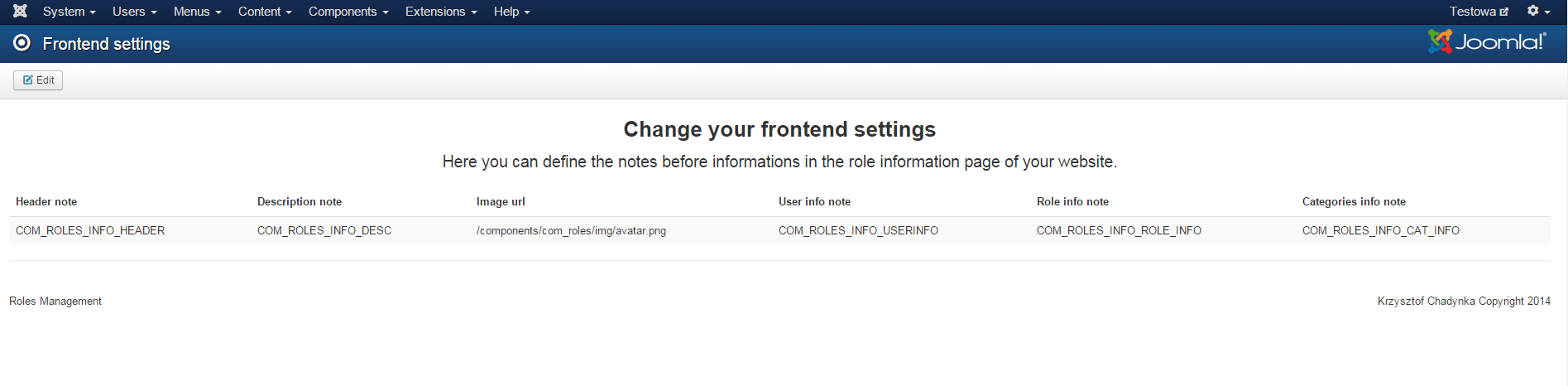
* 1. Usuwanie (należy zaznaczyć element, a następnie wybrać opcję *Delete*)
  2. Edycja (należy zaznaczyć element, a następnie wybrać opcję *Edit*)
  3. Dodawanie nowego powiązania pomiędzy rolą, a kategorią (Należy wybrać opcję *New)*



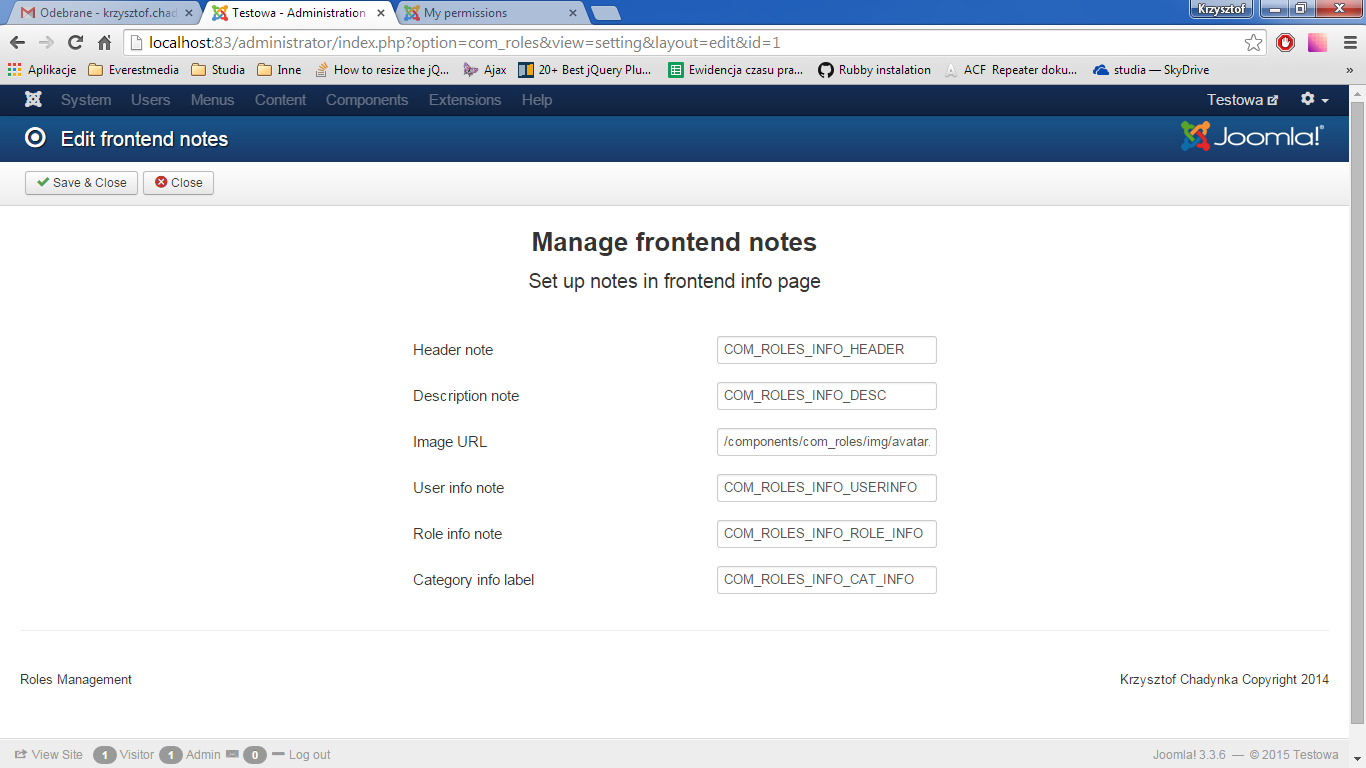
Rysunek Edycja / dodawanie nowej kategorii

1. **Frontend settings**

Jak zostało już wspomniane w tabeli (Tabela 3), podstrona ta daje możliwość edycji treści w widoku szczegółów użytkownika korzystającego z administrowanej witryny.



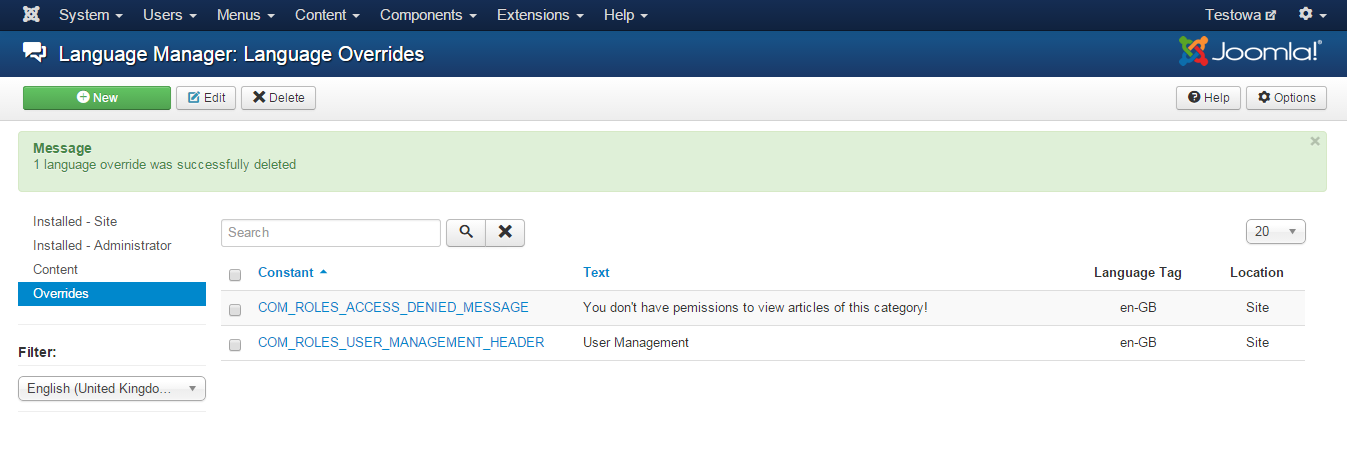
Rysunek 17 Główny widok postrony Frontend settings - panel administracyjny

Dostępna jest opcja edycji poszczególnych treści (należy wybrać opcję *Edit)* 

Rysunek 18 Edycja treści w widoku szczegółów użytkownika - panel administracyjny

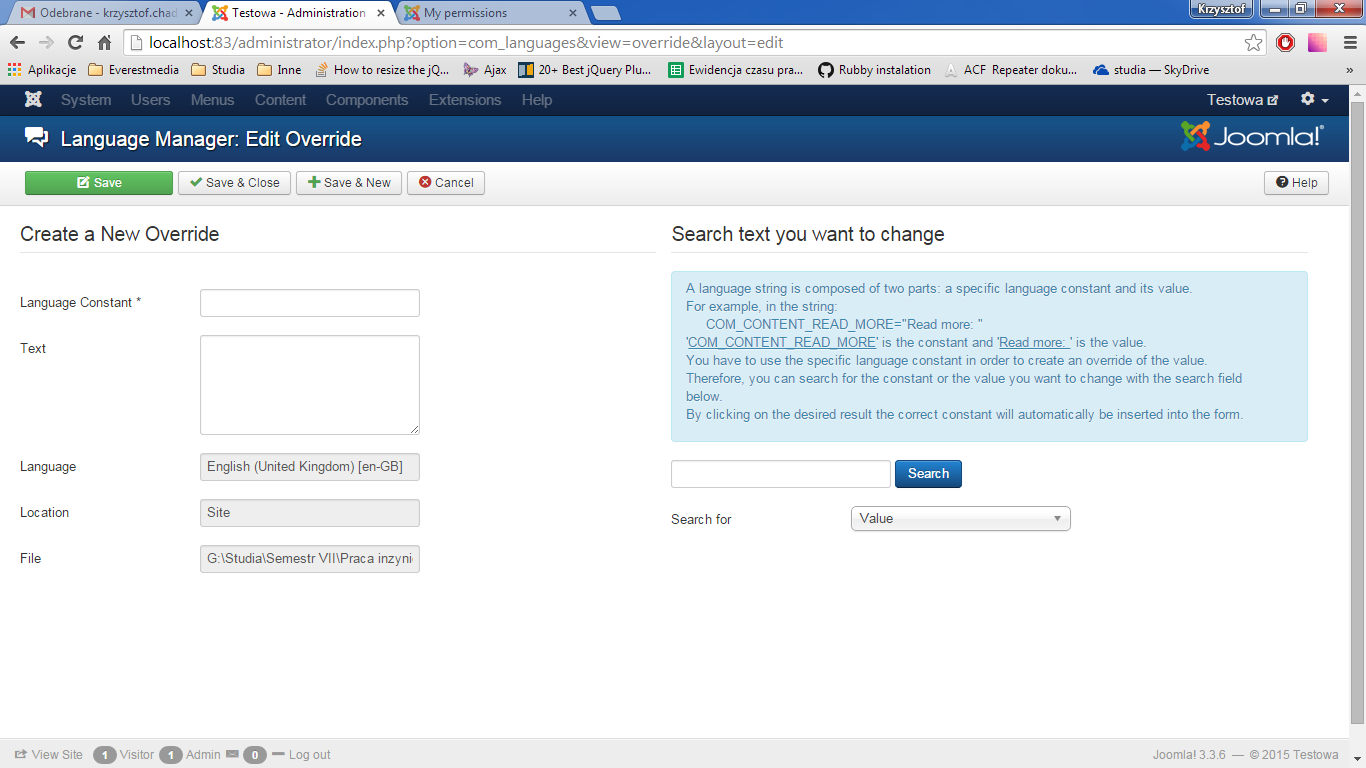
Każda treść (poza adresem url zdjęcia) została zapisana jako sekwencja, która w następnej kolejności będzie przetłumaczona przez System Tłumaczeń framework’u Joomla!. Aby jednak tłumaczenie nastąpiło, należy dodać daną treść i jej tłumaczenie do bazy:

* 1. W górnym menu wybrać *Extension -> Language Manager*
  2. Z bocznego menu po lewej stronie wybrać opcję *Overrides*



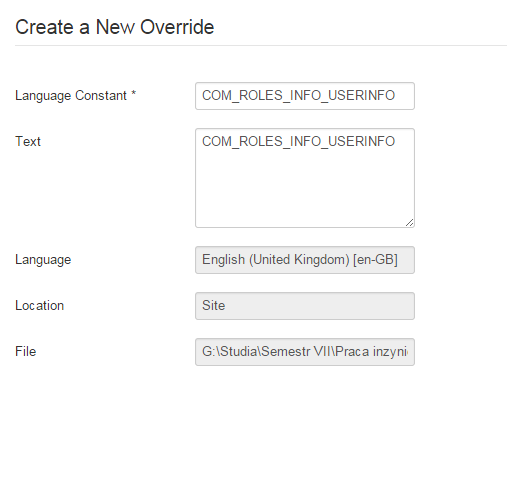
Rysunek 19 Dostępne tłumaczenia - panel administracyjny

* 1. Wybrać opcję New



Rysunek 20 Dodawanie nowego tłumaczenia - panel administracyjny

* 1. W polu *Language Constant* należy wpisać sekwencje do przetłumaczenia, natomiast w polu *Text* zostaje wprowadzony tekst, które będzie tłumaczeniem.

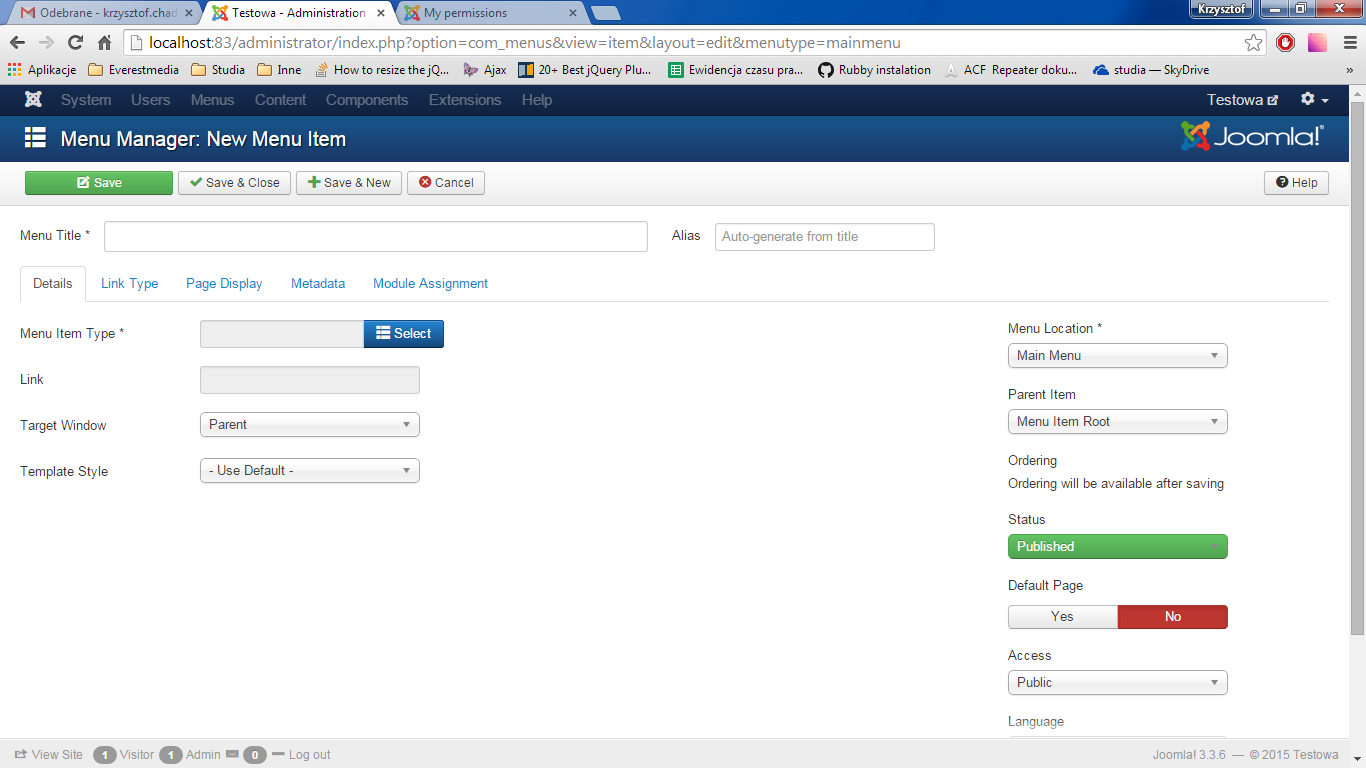


Rysunek 21 Przykładowe tłumaczenie tesktu - panel administracyjny

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 we 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*



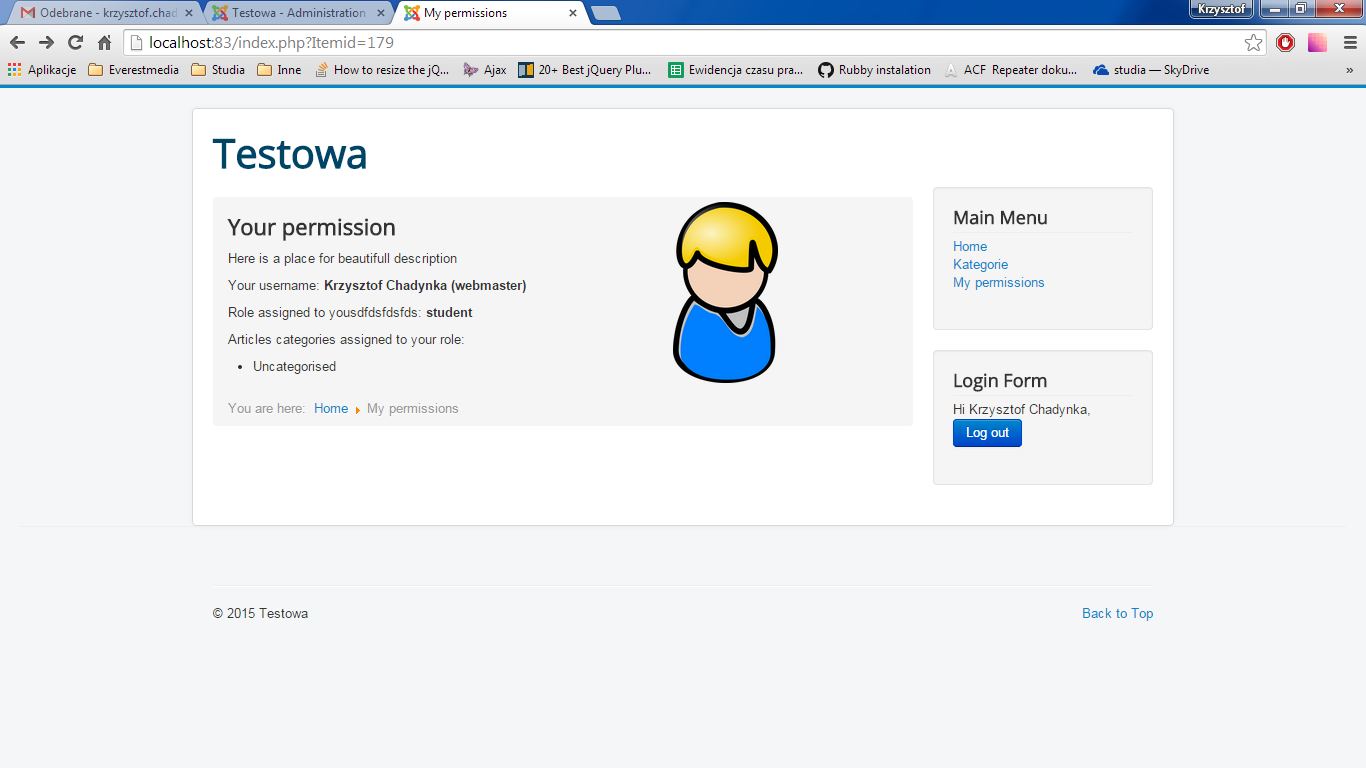
Rysunek 22 Dodanie nowej pozycji w menu - panel administracyjny

W polu *Menu Item Type* wybiera sięopcję *roles -> User info*, natomiast w polu *Menu Title* zostaje wpisana dowolna treść, która będzie wyświetlana jako odnośnik we frontowej części witryny.

## 7.4 Część frontowa witryny

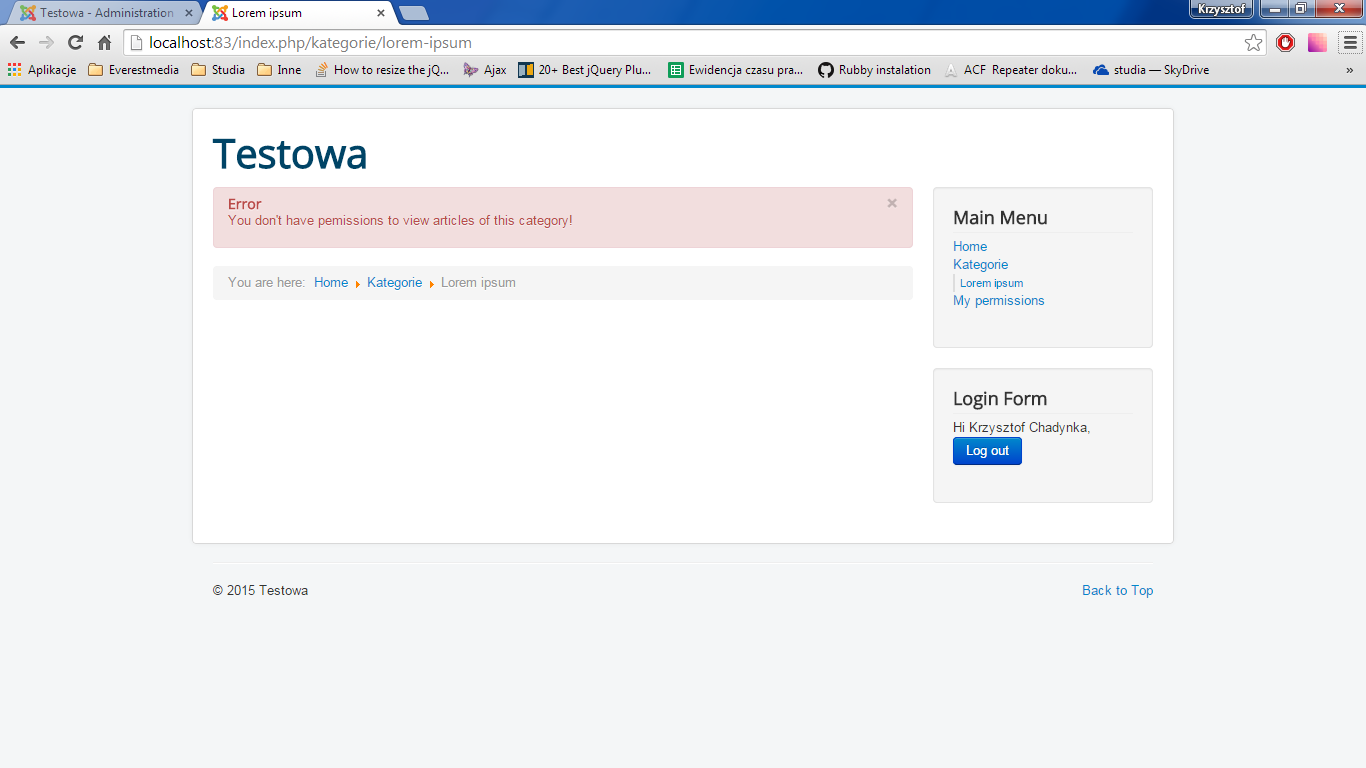
1. **Informacje na temat użytkownika**

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



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 (Rysunek 24).

# 8. Podsumowanie i wnioski

## 8.1 Podsumowanie

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

Na przygotowanie działającej aplikacji złożyły się następujące czynności:

* Opracowano strukturę bazy danych (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.

## 8.2 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.

# 9. Literatura

**Literatura Zwarta**

1. Dan Rahmel: *Joomla! Profesjonalne tworzenie stron WWW*, wyd. Helion 2010

**Inne**

1. <http://ai.ia.agh.edu.pl/wiki/_media/pl:miw:2007:miw07-softeng_mvc.pdf> (dostęp z dnia 21.01.2015).
2. [*www.docs.joomla.org*](http://www.docs.joomla.org)(dostęp z dnia 04.01.2015);
3. [*www.wiki.joomla.pl*](http://www.wiki.joomla.pl) (dostęp z dnia 06.01.2015);

# 10. Ikonografia

Spis ilustracji

[Rysunek 1 Schemat wzorca MVC 7](#_Toc409692664)

[Rysunek 2 Schemat blokowy - proces autoryzacji 9](#_Toc409692665)

[Rysunek 3 Schemat bazy danych 12](#_Toc409692666)

[Rysunek 4 Instalacja komponentu - logowanie do panelu administracyjnego 25](#_Toc409692667)

[Rysunek 5 Instalacja komponentu - przejście do odpowiedniej zakładki 26](#_Toc409692668)

[Rysunek 6 Instalacja komponentu - wybór pliku 26](#_Toc409692669)

[Rysunek 7 Instalacja komponentu - informacja o pozytywnym wyniku instalacji. 27](#_Toc409692670)

[Rysunek 8 Przesłanie pliku controller.php na zdalny serwer 28](#_Toc409692671)

[Rysunek 9 Położenie komponentu w menu głównym - panel administracyjny 28](#_Toc409692672)

[Rysunek 10 Główny widok podstrony Users Management – panel administracyjny 29](#_Toc409692673)

[Rysunek 11 Edycja / dodawanie nowej roli 30](#_Toc409692674)

[Rysunek 12 Główny widok podstrony Roles management – panel administracyjny 30](#_Toc409692675)

[Rysunek 13 Edycja istniejącej roli – panel administracyjny 31](#_Toc409692676)

[Rysunek 14 Dodawanie nowej roli – panel administracyjny 31](#_Toc409692677)

[Rysunek 15 Główny widok podstrony Categories assigned to roles - panel administracyjny 31](#_Toc409692678)

[Rysunek 16 Edycja / dodawanie nowej kategorii 32](#_Toc409692679)

[Rysunek 17 Główny widok postrony Frontend settings - panel administracyjny 32](#_Toc409692680)

[Rysunek 18 Edycja treści w widoku szczegółów użytkownika - panel administracyjny 33](#_Toc409692681)

[Rysunek 19 Dostępne tłumaczenia - panel administracyjny 33](#_Toc409692682)

[Rysunek 20 Dodawanie nowego tłumaczenia - panel administracyjny 34](#_Toc409692683)

[Rysunek 21 Przykładowe tłumaczenie tesktu - panel administracyjny 34](#_Toc409692684)

[Rysunek 22 Dodanie nowej pozycji w menu - panel administracyjny 35](#_Toc409692685)

[Rysunek 23 Podstrona informacji o użytkowniku - część frontowa 36](#_Toc409692686)

[Rysunek 24 Informacja o nieudanej autoryzacji - część frontowa 36](#_Toc409692687)

Spis tabel

[Tabela 1 Struktura początkowa tabeli joo\_roles\_info 11](#_Toc409692661)

[Tabela 2 Struktura początkowa tabeli joo\_roles\_role 12](#_Toc409692662)

[Tabela 3 Zestawienie podstron i ich funkcjonalności - panel administracyjny 29](#_Toc409692663)

Spis listing-ów kodu

[Listing 1 Ograniczenie dostępu do pliku 14](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692649)

[Listing 2Plik rolex.xml 15](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692650)

[Listing 3 Plik site/language/en-GB/en-GB.com\_roles.ini 16](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692651)

[Listing 4 Plik install.mysql.utf8.sql 18](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692652)

[Listing 5 Klasa RolesControllerCategories 19](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692653)

[Listing 6 Klasa RolesModelCategories 19](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692654)

[Listing 7 Klasa RolesViewCategories 20](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692655)

[Listing 8 Klasa RolesModelCategory 21](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692656)

[Listing 9 Funkcja geturrentCategoryID 22](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692657)

[Listing 10 Funkcja getCurrentRoleID 22](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692658)

[Listing 11 Funkcja checkIfCategoryExists 23](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692659)

[Listing 12 Weryfikacja autoryzacji 24](file:///G:\Studia\Semestr%20VII\Praca%20inzynierska\projekt\Krzysztof%20Chadynka%20projekt%20inżynierski.docx#_Toc409692660)