

Data oddania: _____

Ocena: _____

Mateusz Grotek	186816
Mateusz Jakóbczak	186819
Rafał Jurkiewicz	186822
Łukasz Kotyński	186829
Paweł Tarasiuk	186875

Kontrola dostępu w dynamicznych systemach informatycznych

1. Problem

Przedstawiony problem dotyczy tworzenia i analizy zaawansowanych, inteligentnych mechanizmów kontroli dostępu w dynamicznych systemach informatycznych. Celem projektu jest zapoznanie się ze stosowanymi rozwiązaniami, zdobycie aktualnej wiedzy na ten temat, a także wykonanie praktycznej implementacji wybranych rozwiązań.

2. Wprowadzenie

Pierwszą rzeczą, jaką należy zrobić w celu rozwiązania postawionego werbalnie problemu jest dokładne zrozumienie określeń użytych w jego specyfikacji. W wypadku kontroli dostępu w dynamicznych systemach informatycznych istotną rzeczą jest zwrócenie uwagi na pojęcia kontroli dostępu i dynamicznego systemu informatycznego. Jako, że główna część zagadnienia skupia się na zapewnieniu kontroli dostępu, wyjaśnimy najpierw drugie, mniej istotne pojęcie, jakim są dynamiczne systemy informatyczne.

Pojęcie dynamiki można tutaj rozumieć na różne sposoby, ale na pewno podstawowym aspektem jest tutaj wrażliwość na zmieniające się warunki, takie jak na przykład czas, lub pewne zdarzenia zachodzące zarówno w samym

systemie, jak i poza nim. Oznacza to, że system nie powinien robić żadnych założeń na temat tego co się aktualnie dzieje, a raczej reagować dynamicznie. Jako przykład możemy podać edycję stron typu wiki. W naszym projekcie nowe kategorie stron może dodawać tylko użytkownik posiadający rolę wikipedysty, i tylko taki, który utworzył przynajmniej 3 artykuły. Oznacza to, że dostęp do komponentu odpowiadającego za dodawanie nowych kategorii zostaje odblokowany dynamicznie w odpowiedzi na zajście opisanego zdarzenie, jakim jest utworzenie trzeciego artykułu. Dlatego stworzenie systemu dynamicznego jest dużo trudniejsze od stworzenia systemu statycznego.

Wróćmy teraz do pojęcia kontroli dostępu. Kontrola dostępu oznacza, że dostęp do pewnych zasobów jest, z jednej strony, dozwolony lub zabroniony, natomiast z drugiej, że jest on nadzorowany, czyli że na przykład każde skorzystanie z zasobów jest zapisywane. Wynika z tego, że elementami kontroli dostępu są nie tylko autentykacja i autoryzacja, ale także audyt.

3. Aktualny stan wiedzy

Jest wiele różnych sposobów na zapewnienie kontroli dostępu. Co więcej sposoby te niekoniecznie kolidują ze sobą, a często mogą ze sobą współpracować. Do podstawowych mechanizmów mogących zapewnić kontrolę dostępu należą (wg. [1] [2] [3]):

- kontrola dostępu uznaniowa *Discretionary Access Control*, *DAC*
- kontrola dostępu obowiązkowa *Mandatory Access Control*, *MAC*
- kontrola dostępu oparta na rolach *Role Based Access Control*, *RBAC*.

Każdy z tych modeli różni się elementem, który przydziela prawa dostępu, a także mechanizmami dostępu [3].

W przypadku **DAC** zarządzaniem uprawnieniami zajmują się właściciele danych. Często wykorzystywane są do tego listy kontroli dostępu (*Access Control Lists*, *ACLs*). Dobrym przykładem takiego rozwiązania jest dostęp do plików i katalogów na dysku w Uniksie/Linuksie. Właściciel pliku/katalogu może ustawić prawa dostępu dla siebie, grupy, do której należy dany plik, a także dla osób spoza grupy. Prawa te, to prawa odczytu, zapisu i wykonywania, a także pewne dodatkowe prawa o których nie będziemy tutaj pisać. Jednocześnie zaletą i wadą takiego rozwiązania jest fakt, że zarządzanie uprawnieniami dokonywane jest przez właścicieli zasobów. Jest to zaleta, gdyż łatwo jest zarządzać właścicielowi zasobami, które tylko on posiada, gdyż ich ilość jest ograniczona. Jest to wada, gdyż brak jest centralnego nadzoru nad uprawnieniami.

Przypadek **MAC** zakłada, że przydzielaniem uprawnień zajmuje się sam system na podstawie etykiet przydzielonych do osób i zasobów. Na przykład jeśli osoba ma przydzieloną etykietę „Secret”, to może ona przeglądać zasoby oznaczone jako „Secret”, a także zasoby na poziomach niższych. Nie może już za to przeglądać zasobów oznaczonych jako „Top Secret”. Sam system zapewnia dostęp do zasobów na podstawie ustalonej w trakcie jego tworzenia polityki bezpieczeństwa.

Przypadek **RBAC** bazuje na przydzieleniu dla każdego użytkownika jego roli w organizacji. Przydzielaniem ról i uprawnień do ról zajmuje się administrator. Następnie każdej roli są przydzielane dostępne dla niej zasoby.

Amerykańskie ministerstwo obrony (DoD) zdefiniowało w dokumencie Trusted Computer System Evaluation Criteria [4] (w skrócie TCSEC) następujące 4 poziomy dostępu:

- A – ochrona zweryfikowana
- B – ochrona obowiązkowa
- C – ochrona uznaniowa
- D – ochrona minimalna

Ochrona minimalna dotyczy systemów, które zostały ocenione, ale nie spełniły żadnych kryteriów. Kolejne poziomy oznaczają lepszą ochronę. W wypadku poziomu C możliwa jest tylko ochrona DAC. Dla poziomu B mamy także politykę ochrony, czyli dodatkowo MAC. W wypadku A procedury zostały formalnie zweryfikowane (np. przez podanie dowodu poprawności programu).

W naszym projekcie postanowiliśmy użyć połączenia modeli RBAC i MAC. Z modelu RBAC weźmiemy pojęcie roli. Każdy użytkownik będzie przypisany do określonej roli, której przysługują określone uprawnienia. Jednakże nie będą to uprawnienia statyczne, ale dynamiczne, co oznacza, że w pewnych warunkach pewne uprawnienia będą nieważne lub ograniczone. W tym celu zastosowane zostaną elementy modelu MAC. Inaczej mówiąc model RBAC będzie miał za zadanie zapewnić pewne statyczne uprawnienia, które następnie będą modyfikowane dynamicznie przez odpowiednie zdarzenia zachodzące w systemie i poza nim. Oznacza to, że nasz system uprawnień będzie miał strukturę dwuwarstwową. Pierwszą warstwę będą stanowić przypisane przez administratora systemu role i ich uprawnienia. Nazwijmy takie uprawnienia **możliwościami** roli. To że rola ma możliwość nie oznacza automatycznie, że ma ona także **uprawnienie**. Zdecyduje o tym system na podstawie aktualnych zdarzeń.

Model RBAC został wstępnie opisany w artykule Ferraiolo i Kuhna [6], a dalsze szczegóły można znaleźć w artykule Sandhu, Ferraiolo i Kuhna [7]. Amerykański Narodowy Instytut Standardów i Technologii (National Institute of Standards and Technology) opracował model [5], którego celem jest standaryzacja systemów bazujących na rolach. Role stanowią element pośredni, ułatwiający zarządzanie. Do jednej roli może być przyporządkowana więcej niż jedna osoba. Ponadto do jednej roli mogą być przyporządkowane różne zasoby.

Standard NIST podzielił systemy RBAC na 4 klasy, przy czym każda następna klasa jest rozszerzeniem poprzedniej:

- płaski RBAC
- hierarchiczny RBAC
- RBAC z więzami
- symetryczny RBAC

Najprostszą odmianą jest płaski RBAC, który zawiera powiązania między rolami a użytkownikami i uprawnieniami. RBAC hierarchiczny dodatkowo zawiera powiązania między rolami, takie jak na przykład dziedziczenie. W RBAC z więzami dodatkowo jest zapewniona spójność ról i podział odpo-

wiedzialności, a także ograniczenia na powiązania między użytkownikami i rolami, a także pomiędzy jedną rolą a drugą. W symetrycznym RBAC dodatkowo jest możliwość nakładania ograniczeń na powiązania między rolami a uprawnieniami. Jest także możliwość przeglądania i zaawansowanego zarządzania uprawnieniami.

Nasz system chcemy zrealizować jako system z uprawnieniami bazującymi na regułach. Silniki reguł biznesowych to popularne rozwiązanie w firmach. Służą tam one na przykład to zarządzania przepływem dokumentów, a także zarządzania zasobami ludzkimi. Najpopularniejszym rozwiązaniem jest Drools [8], który jest systemem reguł przeznaczonym dla środowiska Java i rozwijanym przez firmę RedHat. Jednakże w naszym projekcie chcemy użyć innego sprawdzonego rozwiązania, jakim jest język Prolog [9]. Jest to język programowania deklaratywnego w logice, który jest bardzo dobrze dostosowany do rozwiązywania skomplikowanych problemów w których występuje wiele reguł.

4. Opis rozwiązania problemu

Jak opisaliśmy powyżej, aby zapewnić naszemu rozwiązaniu maksymalną elastyczność zastosowaliśmy reguły opisane jako reguły Prologowe. Dzięki temu możliwe stało się dowolne ustalenie zależności między uprawnieniami. Zastosowaliśmy model oparty o role, jednakże nasz system pozwala także przypisywać uprawnienia bezpośrednio konkretnym użytkownikom. Co więcej można zupełnie dowolnie kształtować zależności między rolami. Na przykład osobie posiadającej rolę administratora wiki można automatycznie przypisać rolę wikipedysty itp.

Omówiliśmy powyżej sposób organizacji zarządzania uprawnieniami. Jednakże uprawnienia służą przecież do zarządzania określonymi zasobami. Optymalnym rozwiązaniem jest zdefiniowanie ogólnej postaci zasobu, i umożliwieniu zarządzania uprawnieniami do niej. Dzięki postawieniu problemu na takim poziomie ogólności byliśmy w stanie oddzielić proces definiowania i zarządzania uprawnieniami, od procesu tworzenia zasobów. Dzięki temu nasz system uprawnień można dopasować praktycznie do dowolnych zasobów, które jesteśmy w stanie przedstawić w określonej postaci ogólnej.

Aby omówić tę postać ogólną należy zauważyć, że każdy zasób, aby był dostępny dla użytkownika za pomocą strony internetowej, musi mieć przypisany mu określony widok, za pomocą którego użytkownik wchodzi w interakcję z tym zasobem. Podstawowymi operacjami, jakie można wykonać na zasobie są operacje czytania zasobu i zapisywania zasobu. Wszystkie inne operacje można rozumieć jako pewne wersje tych dwóch operacji. Jako przykład podamy chociażby dostęp do kolekcji zasobów. Dodawanie elementu do kolekcji to po prostu pisanie do kolekcji. Przeglądanie listy zasobów w kolekcji to odczyt z kolekcji. Tworząc odpowiednią ilość zasobów i bazując na tych dwóch operacjach jesteśmy w stanie opisać praktycznie dowolny zasób.

W związku z powyższym każdą interakcję użytkownika z zasobem możemy opisać za pomocą czterech parametrów zasobu i dwóch parametrów użytkownika:

- parametry zasobu:
 - unikalny identyfikator zasobu
 - typ zasobu
 - unikalny identyfikator widoku
 - typ widoku
- parametry użytkownika
 - nazwa użytkownika
 - hasło

Używając tych danych możemy już zaproponować pewien system kontroli dostępu. Jednakże system taki byłby niepełny, gdyż nie możliwe byłoby w nim zdefiniowanie zależności od innych parametrów systemu w aktualnym jego stanie. Dlatego nasz system umożliwia dostęp do dowolnego beana z poziomu Prologu. Dzięki takiemu rozwiązaniu możemy na bieżąco podczas weryfikacji uprawnień odpytywać dowolne elementy systemu o ich stan.

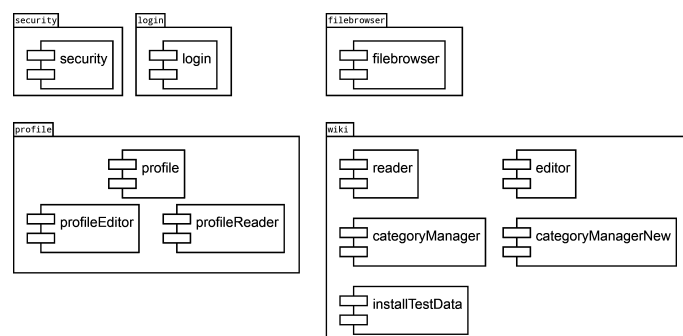
5. Opis implementacji

Implementacja rozwiązania problemu została wykonana na platformie *JavaEE* w technologii *JSF* i *PrimeFaces*. Implementację można podzielić na dwie zasadnicze części. Pierwszą jest zestaw przykładowych komponentów, którymi możemy zarządzać. Poniżej opiszemy każdy z tych elementów. Drugą jest silnik reguł odpowiedzialny za dostęp do komponentów i zasobów w nich wyświetlanych.

5.1. Zestaw komponentów

5.1.1. Opis ogólny

Każdy komponent został wykonany przy użyciu technologii *JSF Component Components*. Lista komponentów jest przedstawiona na rysunku 1.



Rysunek 1. Diagram komponentów

Każdy komponent jest zestawem widoków, w których można przeglądać określone typy zasobów. Poniżej prezentujemy listę stworzonych komponentów:

- filebrowser – przeglądanie plików na serwerze
- login – logowanie

- profile – przeglądanie profili użytkowników
- security – definiowanie uprawnień
- wiki – funkcjonalność wikipedii

Każdy komponent jest wykonany według określonego szablonu przedstawionego w postaci skrótowej poniżej:

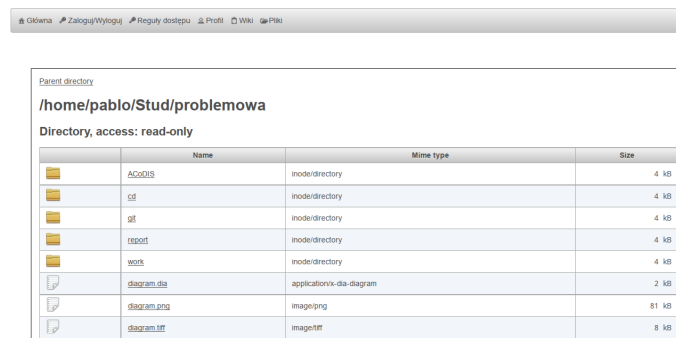
- nagłówek komponentu – zawiera nagłówek xml i niezbędne elementy,
- część dostępna dla uprawnień specjalnych – wyświetlana, gdy użytkownik ma uprawnienia na poziomie **special**,
- część dostępna dla uprawnień do zapisu – wyświetlana, gdy użytkownik ma uprawnienia na poziomie **write**,
- część dostępna dla uprawnień do odczytu – wyświetlana, gdy użytkownik ma uprawnienia na poziomie **read**,
- część dostępna dla braku uprawnień – wyświetlana, gdy użytkownik ma uprawnienia na poziomie **no**,
- stopka komponentu – zamknięcie tagów otwartych w nagłówku.

W katalogu projektu `/src/main/webapp/resources/example` znajduje się przykładowy, poglądowy moduł złożony z dwóch zagnieżdżonych komponentów. Wszystkie pozostałe komponenty wykonane są zgodnie z tym szablonem.

5.1.2. Komponent filebrowser

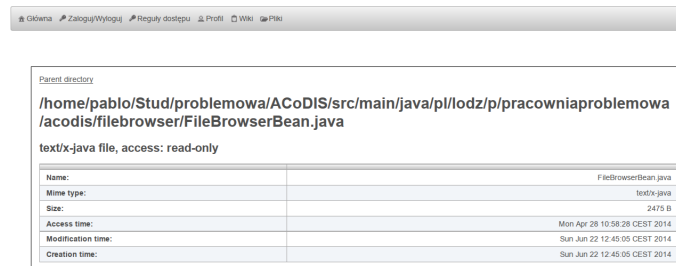
Komponent `filebrowser` stanowi narzędzie pozwalające korzystać z uprawnień do zasobów jakimi są węzły systemu plików na serwerze. Rozróżniane są dwa istotnie różne typy węzłów:

- *Zwykłe pliki* – dla tego typu węzłów wyświetlane są ich własności takie jak typ MIME, rozmiar pliku oraz daty utworzenia, modyfikacji i dostępu. Informacje o poszczególnych danych wybierane są spośród wyników metody `java.nio.file.Files.readAttributes`, zaś typ MIME odgadywany jest przez (wdrożoną w Javie 7) metodę `java.nio.file.Files.probeContentType`.
- *Katalogi* – dla tego typu węzłów wyświetlana jest lista szczegółowa z elementami ich zawartości. Dla każdego węzła będącego synem przeglądane katalogu wyświetlana jest jego nazwa, typ MIME oraz rozmiar. Wybranie nazwy węzła z listy powoduje przejście do niego.



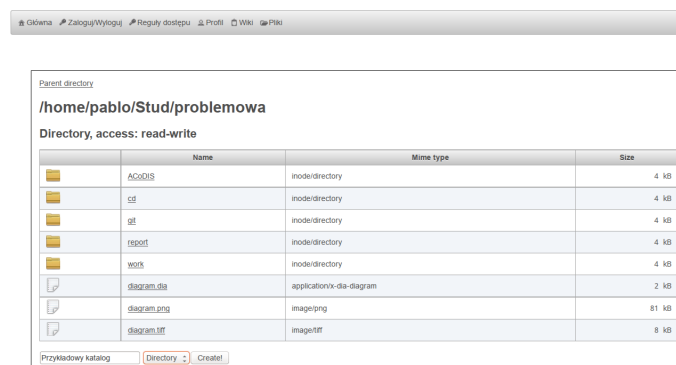
	Name	Mime type	Size
	ACeUS	inode/directory	4 kB
	cd	inode/directory	4 kB
	git	inode/directory	4 kB
	report	inode/directory	4 kB
	work	inode/directory	4 kB
	diagram.da	application/x-dia-diagram	2 kB
	diagram.png	image/png	81 kB
	diagram.tif	image/tif	8 kB

Rysunek 2. Przeglądanie katalogu z prawami tylko do odczytu

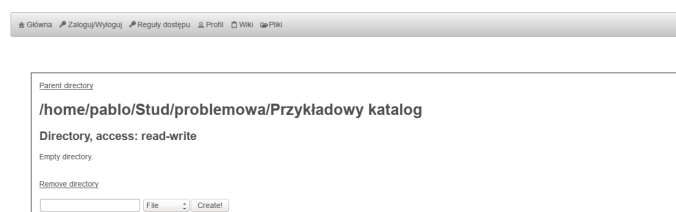


Rysunek 3. Przeglądanie pliku z prawami tylko do odczytu

Oczywiście aby jakiegokolwiek informacje zostały wyświetlone, trzeba mieć przynajmniej uprawnienia na poziomie *read*. Dodatkową możliwością wspólną dla wszystkich typów węzłów jest możliwość przejścia do węzła nadrzędnego. Wspomniane „przejście” pomiędzy jednym węzłem a drugim może oczywiście skończyć się informacją o braku uprawnień, w zależności od tego jakie uprawnienia do węzła docelowego posiada dany użytkownik. Dla łatwości demonstracji działania systemu, jeżeli nie jest sprecyzowane, jaki zasób ma być wyświetlony (w naszej aplikacji dotyczy to tylko pierwszego uruchomienia komponentu), wybierany jest specjalny katalog na serwerze przypisany do aktualnie zalogowanego użytkownika. Jeżeli takowy nie istnieje to jest on tworzony.



Rysunek 4. Dodawanie nowych węzłów w katalogach z uprawnieniem *write*



Rysunek 5. Usuwanie pustych katalogów uprawnieniem *write*

Dodatkowe możliwości pojawiają się gdy uprawnienia do danego zasobu mają poziom *write*. Wtedy dla plików i pustych folderów pojawia się możliwość usunięcia bieżącego węzła, oraz dla folderów (niezależnie od tego czy są

puste czy nie) pojawia się możliwość tworzenia pustych plików oraz pustych folderów wewnątrz nich.

Podobnie jak we wszystkich pozostałych elementach naszej implementacji systemu ACoDIS, uprawnienia te mogą być specyficzne dla danego użytkownika lub wynikać z jego roli oraz dodatkowo zależeć od ścieżki do danego zasobu oraz wszelkiego rodzaju czynników zmieniających się w sposób dynamiczny. Klasycznym przypadkiem jest tutaj wpływający czas (i jedna z reguł demonstrujących możliwości systemu jest oparta właśnie na tym).

Jak zostało wspomniane w ogólnym opisie modułu `filebrowser`, za każdym razem odczytywana jest aktualna informacja o serwerowym systemie plików. Zważywszy że struktura tego systemu plików może ulegać ciągłym zmianom (między innymi wynikającym z działań użytkowników posiadających uprawnienia na poziomie *write* do pewnych węzłów), sam ten fakt czyni zachowanie niniejszego modułu dynamicznym. Kluczowy jest jednak dla nas dynamizm zdefiniowany wprost poprzez reguły dostępu – opisany poniżej przykład 3. realizuje tę myśl w sposób specyficzny dla modułu `filebrowser`.

Przykładowe zaawansowane reguły związane z niniejszym modułem to:

1. Użytkownik posiada dostęp na poziomie *write* do każdego węzła, którego nazwa zaczyna się od jego loginu. O ile teoretycznie można sobie wyobrazić system uprawnień do plików w którym możemy dawać prawa do edycji zasobu pojedynczym innym użytkownikom poprzez ustawienie odpowiedniej nazwy, to rozwiązanie takie trudno nazwać optymalnym. Celem stworzenia tej reguły jest jednak pokazanie, że możliwości takie jak niniejsza mogą być wprowadzane do naszego systemu za pomocą pojedynczych linii kodu w języku Prolog.
2. Użytkownik posiada dostęp na poziomie *read* do każdego węzła, który posiada przodka o nazwie zaczynającej się od jego loginu. Tak zdefiniowana reguła może zostać w praktycznym zastosowaniu przetłumaczona na to, że każdy użytkownik ma dostęp przynajmniej na poziomie *read* do całej zawartości swojego katalogu domowego.
3. Dla stworzonego na potrzeby demonstracji użytkownika `test3` definiowaliśmy reguły dostępu polegające na tym, że użytkownik ten może czytać wszystkie pliki na serwerze, oraz dodatkowo posiada uprawnienie na poziomie *write* do plików, które zostały utworzone nie dawniej niż 60s przed bieżącą chwilą. Reguła ta w dosłowny sposób realizuje dynamizm systemu uprawnień, a także ma największy potencjał jeżeli chodzi o zastosowania praktyczne – nietrudno wyobrazić sobie system, w którym użytkownik po dodaniu nowego zasobu ma pewien ograniczony czas na wycofanie się z tego i usunięcie go. Aby to uzyskać, można wykorzystać zaprojektowaną przez nas regułę nawet bez wprowadzania do niej zmian, gdyż stanowiące osobne zagadnienie ograniczenia polegające na tym żeby użytkownik mógł usuwać tylko pliki zamieszczone przez siebie mogłyby być realizowane przez oddzielne reguły.

5.1.3. Komponent login

Komponent ten umożliwia zalogowanie się użytkownika. Co ciekawe całą logikę logowania udało się zaimplementować jako kilka reguł Prologowych.

Oznacza to, że sam komponent nie posiada wewnętrznej logiki, a cała logika zawarta jest w systemie reguł opisujących proces logowania. Zanim użytkownik się zaloguje komponent wyświetla się w trybie braku dostępu (**no**). Jednakże po wpisaniu prawidłowego loginu i hasła użytkownik uzyskuje dostęp do komponentu w trybie odczytu (**read**). W tym trybie komponent wyświetla informacje o aktualnie zalogowanym użytkowniku. Za takie działanie komponentu odpowiada tylko jedna reguła dostępowa opisana poniżej.

```
canAccessAt(user(Username, Password),
            resource('login','login','login','login'),
            readAccess) :-
    isUsernameAndPassword(Username, Password).
```

Powyższą regułę można zapisać w języku polskim jako: „Jeżeli Username i Password są poprawnymi danymi logowania, to użytkownik o loginie Username i hasle Password ma dostęp do zasobu o identyfikatorach login, login, login na poziomie odczytu. Bardziej szczegółowy opis i postać ogólna reguł Prologu jest dostępna w sekcji opisującej silnik reguł. Natomiast procedura wylogowania użytkownika polega po prostu na usunięciu jego zapamiętanych danych. W takim wypadku warunek w powyższej regule nie jest już spełniony i użytkownik traci automatycznie prawo dostępu na poziomie odczytu. Dokładniejszy opis jest zawarty w sekcji opisującej silnik reguł.

5.1.4. Komponent profile

Komponent służy do przeglądania oraz edycji znajdujących się w systemie danych o użytkownikach. Jego implementację można podzielić logicznie na trzy modelowe warstwy: prezentacji, logiki i danych.

Warstwa prezentacji to utworzony z pomocą JSF front-end w postaci plików **.xhtml* zgromadzonych w folderze *profile* komponentu. Znajdują się tam wszystkie elementy istotne dla działania komponentu, każdy z nich posiada strukturę umożliwiającą kontrolę dostępu opisaną w poprzednich rozdziałach. Do najważniejszych jego elementów należą *profileReader* odpowiedzialny za wyświetlanie danych dostępnych tylko dla użytkowników z prawami do czytania treści oraz *profileEditor* wprowadzający dodatkową funkcjonalność dla posiadaczy praw do zapisu. Pozostałe elementy agregują dodatkowe treści udostępniane w przeglądarce profili, tj. komunikaty wyświetlane użytkownikom i dodatkowe informacje o kontaktach.

Logika komponentu opiera się o pakiet *pl.lodz.p.pracowniaproblemowa.acodis.profile* napisany w technologii Java. Składa się on z trzech klas:

- *Profile* - zawiera model danych dotyczący pojedynczego profilu. Definiuje wszystkie pola opisujące dane profilu oraz metody do obsługi tych danych.
- *ProfileDataBean* - klasa przechowuje wszystkie dane które zmieniają się dynamicznie wraz z użytkowaniem systemu. Przechowuje listę wszystkich profili, aktualnie wyświetlany profil, odniesienie do danych użytkownika wykorzystanych do logowania oraz stan wyświetlanego aktualnie profilu. W metodach klasy kryją się funkcje odpowiedzialne za odczyt i zapis z warstwy danych oraz operacje odczytu i dokonania zmian stanu przeglądarki profili oraz samych danych.

- *ProfileConverter* - z uwagi na to, że przeglądarka danych posiada listener ustawiony na element *selectOneMenu* służący do wyboru pożądanego profilu zaistniała potrzeba implementacji konwertera skupionego wprost na klasie *Profile*. Zatem w klasie *ProfileConverter* znajduje się kod implementujący interfejs *Converter*, służący do skutecznego porównania i wyboru odpowiednich instancji profili. Dzięki temu możliwa jest transformacja danych w postaci stringów z front-endu do obiektów Javy i na odwrót.

Warstwa danych składa się z pliku *profiles.xml* znajdującego się w folderze zasobów systemu. Służy on za podręczną bazę wiedzy zaczytywaną do komponentu profili przy użyciu odpowiednich metod z klasy *ProfileDataBean*. Jest to wydajny i przenośny sposób na agregację danych o użytkownikach, jednocześnie nie rzutujący na eskalację rozmiarów systemu. Jednocześnie rozwiązanie to jest łatwo wymienialne na inne technologie wraz z rozwojem i zmianami w systemie.

Komponent profili z perspektywy użytkowej składa się w zasadniczej mierze z elementu *selectOneMenu* służącego do wyboru pożądanego profilu oraz obszaru wyświetlania danych o wybranej pozycji. Użytkownikowi z prawami *write* została udostępniona dodatkowa funkcjonalność edycji danych wyświetlanego profilu, która jest obsługiwana za pomocą przycisków "edytuj", "zapisz" oraz "anuluj" wyświetlanych przy właściwych stanach przeglądarki profili. Dzięki zaimplementowanym mechanizmom komponent reaguje dynamicznie na aktualny stan warstwy danych oraz uzależnia sposób swojej reakcji od czasu oraz zalogowanego aktualnie użytkownika.

5.1.5. Komponent security

Komponent security jest głównym komponentem odpowiedzialnym za dynamiczną modyfikację reguł w trakcie działania systemu. Zestaw reguł jest podzielony na dwie części. Pierwszą z nich jest biblioteka gotowych reguł zawierająca dwa zestawy elementów:

- elementy odpowiedzialne za podstawową funkcjonalność reguł i logowania
- gotowe reguły, które mogą być użyte przez administratora jako warunki dostępu (jako przykład można podać chociażby regułę zwracającą ilość utworzonych przez danego użytkownika stron na wiki, której można użyć jako warunku dostępu, chociażby do modułu tworzenia kategorii)

Część biblioteczna jest stała i nie zmienia się podczas działania systemu. Nowe wersje systemu mogą łatwo dodawać nowe elementy, gdy zostaną dodane do systemu nowe moduły udostępniające nową funkcjonalność.

Drugą częścią jest zestaw aktualnie obowiązujących reguł. Ta część systemu może być modyfikowana dynamicznie w trakcie jego działania przez użytkownika o odpowiednich uprawnieniach. Komponent security umożliwia edycję i zapis reguł w tej części.

5.1.6. Komponent wiki

Zbiór komponentów odpowiedzialnych za zarządzanie oraz prezentację artykułów wiki. W jego skład wchodzi:

- *wikiReader* - komponent odpowiedzialny za odczyt artykułów
- *wikiCategoryManager* - komponent zarządzający kategoriami wiki

- wikiEditor - komponent odpowiedzialny za tworzenie i edycję artykułów wiki

System wykonany jest w oparciu o dwie warstwy: model - widok, izolując metody niskopoziomowe i pozwalając na łatwą podmianę technologii przechowywania danych. Obecnie wykorzystywany jest system plików. Podział na kategorie wykonany jest przy pomocy podziału na katalogi, natomiast każdy artykuł jest oddzielnym plikiem, gdzie początek pliku zawiera nagłówek przechowujący informacje o autorze artykułu. Większość operacji niskopoziomowych wykonywana jest przez klasę *pl.lodz.p.pracowniaproblemowa.acodis.wiki.WikiUtils*.

W skład WikiReadera wchodzi następujące moduły:

- wikiReader.xhtml - opakowanie komponentu
- resources/wiki/reader.xhtml - widok prezentujący dane
- *pl.lodz.p.pracowniaproblemowa.acodis.wiki.WikiReaderBean* - klasa odpowiedzialna za obsługę logiki odczytywania danych

Podział artykułów na kategorie i artykuły pozwala na zapewnienie rozgraniczenia w dostępie zarówno do poszczególnych stron wiki jak i całych kategorii.

W skład WikiCategoryManager wchodzi następujące moduły:

- wikiCategoryManager.xhtml - opakowanie komponentu zarządzania istniejącymi kategoriami
- wikiCategoryManagerNew.xhtml - opakowanie komponentu tworzenia kategorii
- resources/wiki/categoryManager.xhtml - widok prezentujący formularz zarządzania istniejącymi kategoriami
- resources/wiki/categoryManagerNew.xhtml - widok prezentujący formularz tworzenia nowej kategorii
- *pl.lodz.p.pracowniaproblemowa.acodis.wiki.WikiCategoryManagerBean* - klasa odpowiedzialna za obsługę logiki zarządzania kategoriami

Zarządzanie kategoriami jest podzielone na tworzenie oraz usuwanie istniejących kategorii oraz istnieje również możliwość dla osób nieuprzywilejowanych ograniczenia funkcjonalności do samego podglądu kategorii.

W skład WikiEditora wchodzi następujące moduły:

- wikiEditor.xhtml - opakowanie komponentu
- resources/wiki/editor.xhtml - widok prezentujący formularz edycji/tworzenia dokumentu
- *pl.lodz.p.pracowniaproblemowa.acodis.wiki.WikiEditorBean* - klasa odpowiedzialna za obsługę logiki tworzenia nowego artykułu

5.1.7. Oprawa wizualna

Do Stworzenia oprawy wizualnej, wykorzystany został framework JSF i biblioteka komponentów PrimeFaces. JSF umożliwia tworzenie szablonów, wykorzystywanych później przez strony komponentów. Primefaces dostarcza wielu użytecznych komponentów, gotowych do wykorzystania, jak na przykład menu, rozwijane panele, czy okienka powiadomień.

Główny moduł strony składa się z następujących elementów:

- nagłówek, w którym znajduje się menu, umożliwiające nawigację do stron poszczególnych komponentów

- stopki, nie pełniącej żadnej kluczowej funkcji, ale ładnie dopełniającej kompozycję strony
- części środkowej, wyświetlającej zawartość strony komponentu
- opcjonalnego, rozwijanego panelu bocznego, wyświetlanego w miarę potrzeby (gdy jest sens użycia go w komponencie)

Oczywiście, w razie potrzeby, można dodawać kolejne elementy, a nawet je zagnieżdżać (przykładem zagnieżdżenia jest lewy panel, stanowiący fragment środkowej części strony, dzięki czemu nie nachodzi on na nagłówek i stopkę). Wykorzystanie wspomnianego wyżej panelu bocznego jest widoczne w komponencie Wikipedii. Znajdują się na nim linki, umożliwiające tworzenie i zarządzanie zawartością wikipedii, a także drzewko zawartości strony (kategorie i artykuły). Wyodrębnienie tych elementów ze strony głównej pozwala na wygodniejszą i efektywniejszą nawigację i zarządzanie stroną. Boczne menu jest skalowalne, w związku z czym można je dopasować do indywidualnych potrzeb. Istnieje też opcja zwinięcia go, co umożliwia wyświetlanie treści na niemal całej szerokości strony.

Innymi przykładami wykorzystania komponentów PrimeFaces są rozwijane obszary informacji o autorach projektu, widoczne na stronie głównej, czy też okienka informacyjne, powiadamiające o braku uprawnień do przeglądania określonych treści. W tym drugim przypadku widać możliwości konfiguracyjne komponentów PrimeFaces. Tutaj, do okienka informacyjnego, zostało też dodane wyciemnienie tła, uniemożliwiające interakcję ze stroną aż do momentu zamknięcia okna powiadomienia.

5.2. Silnik reguł Prologowych

5.3. Opis ogólny

Sercem systemu jest silnik reguł Prologowych. Użyliśmy tutaj silnika *tuProlog*, gdyż jest on napisany w Javie i dobrze integruje się z tą platformą. Rozważaliśmy też użycie silnika *SWIProlog*, jednakże wstępna weryfikacja pokazała, że jego integracja z Javą jest niepełna. Choć silnik *tuProlog* nie jest tak zaawansowaną platformą jak *SWIProlog*, to do naszych zastosowań okazał się wystarczający. Jego dodatkowym atutem, który okazał się bardzo istotny jest możliwość dostępu w prosty sposób do dowolnych obiektów i klas Javy.

Tak jak wspomnieliśmy wcześniej użyte reguły podzielone zostały na dwa podzbiory. Pierwszy to reguły *biblioteczne*, które nie zmieniają się w trakcie działania systemu. Opisują one dwa ważne elementy systemu, jakim jest proces logowania i obsługa podstawowych reguł, a także zawierają zestaw gotowych, użytecznych reguł, które mogą być użyte w drugim podzbiórze jako warunki od których możemy uzależnić dostęp. Drugi podzbiór jest dynamiczny i może być na bieżąco modyfikowany przez osobę o odpowiednich uprawnieniach. Zestaw ten jest składany na deklarację określonych, konkretnych uprawnień dla konkretnych ról i użytkowników. Można powiedzieć, że część biblioteczna jest frameworkiem w którym tworzone są już konkretne reguły dla konkretnego systemu. Natomiast reguły biblioteczne mają postać ogólną i nie zmieniają się. Poniżej krótko opisujemy oba zestawy.

Przed przedstawieniem konkretnych reguł wypada powiedzieć kilka słów o języku Prolog, co pomoże zrozumieć podane niżej przykłady. Ogólna postać reguły w języku Prolog wygląda następująco:

```
wniosek :- warunek1, warunek2, ..., warunekN.
```

Reguła taka tłumaczy się na język polski następująco: „Jeżeli zachodzi **warunek1** i **warunek2** i ... i **warunekN**, to zachodzi **wniosek**”. Warunki i wnioski mają postać relacji n -argumentowych, przy czym n może być zerem. A więc postać ogólna wniosku lub warunku jest jedną z następujących:

— **relacja**

— **relacja(argument1, argument2, ..., argumentN).**

Jako argumenty mogą zostać użyte zmienne oznaczające nieznane obiekty, które charakteryzują się tym, że zaczynają się zawsze od wielkiej litery (wyjątkiem jest `_`, która oznacza po prostu zupełnie dowolny obiekt). Poza tym można jako argumentów użyć funkcji (w Prologu funkcje to nie to samo co w innych językach programowania, odpowiednikiem prologowych funkcji są struktury danych w innych językach), które same mogą mieć dowolną ilość argumentów (także zero, w takim wypadku jest to po prostu stała). A więc najbardziej ogólnie można regułę opisać jako:

```
relacja1(Zmienna1, stala1, funkcja1(Zmienna2, stala2)) :-  
    relacja2(Zmienna2, stala2, funkcja2(Zmienna2, stala1)),  
    relacja3(Zmienna3).
```

Nazwy relacji są tak dobrane, aby ich znaczenie było oczywiste. Należy zwrócić także uwagę, że Prolog podstawia pod zmienne wartości w momencie, gdy zostają one wyznaczone z postaci reguł.

5.4. Reguły biblioteczne

Nie będziemy tu opisywać wszystkich reguł bibliotecznych, gdyż zajęłoby to zbyt dużo miejsca, a jedynie podamy reprezentatywne przykłady. Należy przypomnieć, że reguły biblioteczne działają na niskim poziomie i są częścią implementacji, więc ich postać nie musi być tak czytelna jak reguł dostępowych. Poniższa reguła sprawdza, czy jej argument jest aktualnym użytkownikiem.

```
isUser(user(Username, Password)) :-
    isLoginBean(LoginBean),
    LoginBean <- getUsername returns UsernameOrVariable,
    convertsToString(UsernameOrVariable, Username),
    LoginBean <- getRealPassword returns PasswordOrVariable,
    convertsToString(PasswordOrVariable, Password).
```

Natomiast poniższa reguła daje użytkownikowi dostęp do zasobów (z wyłączeniem komponentu logowania), o ile należy on do określonej roli i ta rola ma dostęp do tych zasobów:

```
canAccessAt(user(Username, Password), Resource, AccessLevel) :-
    Resource \= resource('login','login','login','login'),
    isUsernameAndPassword(Username, Password),
    hasRole(Username, Role),
    roleCanAccessAtIfLogged(Role, Resource, AccessLevel).
```

Ze wszystkimi regułami bibliotecznymi można zapoznać się w pliku `/src/main/resources/pl/loz/p/pracowniaproblemowa/acodis/resources/lib.pl`

5.5. Reguły dostępowe

Reguły dostępowe przedstawię bardziej dokładnie. Ich pełną postać można zobaczyć wchodząc z odpowiednimi uprawnieniami do komponentu **security**. Relacje, które stanowią sedno systemu ograniczającego dostęp należą do poniższych:

- `isUsernameAndPassword(Username, Password)` – deklaruje, że hasłem dla użytkownika `Username` jest `Password`.
- `hasRole(Username, Role)` – deklaruje, że użytkownik `Username` ma przypisaną rolę `Role`.
- `userCanAccessAtIfLogged(Username, Resource, AccessMode)` – deklaruje, że użytkownik `Username` ma dostęp do zasobu `Resource` na poziomie `AccessMode`.
- `roleCanAccessAtIfLogged(Role, Resource, AccessMode)` – deklaruje, że rola `Role` ma dostęp do zasobu `Resource` na poziomie `AccessMode`.
- `userCannotAccessAtIfLogged(Username, Resource, AccessMode)` – deklaruje, że użytkownik `Username` ma odcięty dostęp do zasobu `Resource` na poziomie `AccessMode`.
- `roleCannotAccessAtIfLogged(Role, Resource, AccessMode)` – deklaruje, że rola `Role` ma odcięty dostęp do zasobu `Resource` na poziomie `AccessMode`.

Należy zwrócić uwagę, że odcięcia dostępu mają priorytet nad prawami dostępu. Niektóre z powyższych relacji są spełnione tylko pod pełnymi warunkami. Nie będziemy wymieniać ich wszystkich i ograniczymy się do przykładowych:

- `hasFilledProfile(Username)` – użytkownik `Username` wypełnił swój profil.
- `isHour(Hour)` – aktualna godzina to `Hour`.
- `userCreatedNumberOfArticles(Username, Number)` – użytkownik `Username` utworzył `Number` artykułów.

6. Dyskusja rozwiązania i podsumowanie

Nasze rozwiązanie postawionego problemu uznajemy za udane. Jednakże gdybyśmy następnym razem wykonywali podobny projekt, to nie zastosowalibyśmy Javy, ani tym bardziej technologii JSF. Problemem technologii JSF jest jej stanowość. Przystępując do wykonania projektu i wybierając JavęEE i JSF kierowaliśmy się faktem, że wszyscy członkowie zespołu znają język Java i relatywnie łatwe będzie dla nich wdrożenie się w technologię JSF. Jednakże technologia ta nie spełniła pokładanych w niej oczekiwań. Jej głównym problemem jest fakt, że każda odsłona strony powoduje wczytanie zapamiętanego na serwerze drzewa komponentów. Powoduje to, że komponenty te są statyczne i trudno było zmusić tą technologię do dynamicznego podmieniania komponentów pomiędzy wyświetleniami strony. Udało się to w końcu zrealizować, ale dość wysokim kosztem. Dlatego w następnych podobnych projektach nie wykorzystalibyśmy już Javy, ale bardziej dynamiczne środowiska np. Smalltalk i Seaside, Ruby on Rails lub Pythona i Django.

7. Udział w projekcie

Pracami podzieliliśmy się w następujący sposób:

- Mateusz Grotek – projekt systemu, zarządzanie uprawnieniami, Prolog
- Mateusz Jakubczak – komponent profili użytkownika
- Rafał Jurkiewicz – szablon strony i warstwa wizualna
- Łukasz Kotyński – komponent wikipedii
- Paweł Tarasiuk – komponent obsługi plików

Literatura

- [1] Access Control Cheat Sheet https://www.owasp.org/index.php/Access_Control_Cheat_Sheet
- [2] Computer access control http://en.wikipedia.org/wiki/Computer_access_control
- [3] Access Control Systems w *Fundamentals of Information Systems Security* http://en.wikibooks.org/wiki/Fundamentals_of_Information_Systems_Security/Access_Control_Systems
- [4] Trusted Computer System Evaluation Criteria http://en.wikipedia.org/wiki/Trusted_Computer_System_Evaluation_Criteria
- [5] Role Based Access Control (RBAC) and Role Based Security <http://csrc.nist.gov/groups/SNS/rbac/>
- [6] Ferraiolo, D. F., Kuhn, D.R., 1992. Role-Based Access Control. 15th National Computer Security Conference
- [7] Sandhu, R., Ferraiolo D., Kuhn R. 2000. The NIST Model for Role Based Access Control: Toward a Unified Standard. Proceedings, 5th ACM Workshop on Role Based Access Control, Lipiec 26–27, 2000, Berlin, pp. 47–63
- [8] Drools – The Business Logic integration Platform <https://www.jboss.org/drools/>
- [9] Kluźniak F., Szpakowicz S. 1985. Prolog For Programmers.