Akademia Górniczo-Hutnicza
Wydział Elektroniki, Automatyki, Informatyki i Elektroniki

Implementacja modułu bezpieczeństwa dostępu do danych na poziomie encji w zależności od przypisanej roli użytkownika.

Moduł wykorzystuje programowanie aspektowe, pozwalające na zastosowanie w dowolnym systemie gdzie wykorzystano mapowanie O-R w technologii JavaScript. Role systemu posiadają strukturę drzewiastą.

Marcin Michna
Marek Majerski
Hubert Frączek
Marek Krauze

Kraków, 2019/2020

Spis treści

1. Założenia systemu

Moduł bezpieczeństwa ma za zadanie zabezpieczyć dostęp do danych w tabelach przed użytkownikami którzy nie mają uprawnień dostępu.

Aby przygotować moduł do działania wywołać trzeba metodę inicjalizującą, która ustawi dane logowania do bazy danych.

Przy każdorazowej zmianie roli użytkownika wywołać trzeba metodę ustawiającą aktualną rolę w module bezpieczeństwa. Po ustawieniu roli, za każdy razem, gdy wywołane zostanie zapytanie do bazy danych, argumenty funkcji zostaną przechwycone i zmodyfikowana zostanie treść zapytania tak, aby zwrócone z bazy danych rekordy nie zawierały tych do których użytkownik nie ma dostępu.

O uprawnieniach decyduje rola przypisana do użytkownika. Role mają strukturę drzewiastą, więc każda rola znajdująca się wyżej w hierarchii drzewa posiada wszystkie uprawnienia roli swoich dzieci.

Informacje na temat dostępu użytkownika z daną rolą do encji w tabeli będą przechowywane w liście ACL. Dodatkowo w osobnej tabeli ACL przetrzymywane będą informacje na temat uprawnień do dodawania rekordów do tabel w zależności od roli.

2. Opis systemu

2.1 Przechwytywanie zapytań

Podstawową funkcją modułu bezpieczeństwa jest przechwytywanie zapytań do bazy danych, modyfikowanie ich, a następnie zwracanie tylko tych rekordów do których użytkownik ma dostęp. Schemat działania dla każdego zapytania do bazy danych:

1. użytkownik ustawia rolę użytkownika
2. użytkownik wysyła zapytania do bazy danych
3. używając programowania aspektowego przechwycone zostaje zapytanie
4. ustalane zostają uprawnienia użytkownika w zależności od roli
5. zapytanie do bazy danych zostaje zmodyfikowane
6. wykonuje się właściwe zapytanie do bazy danych, które zwraca wynik

2.2 Role

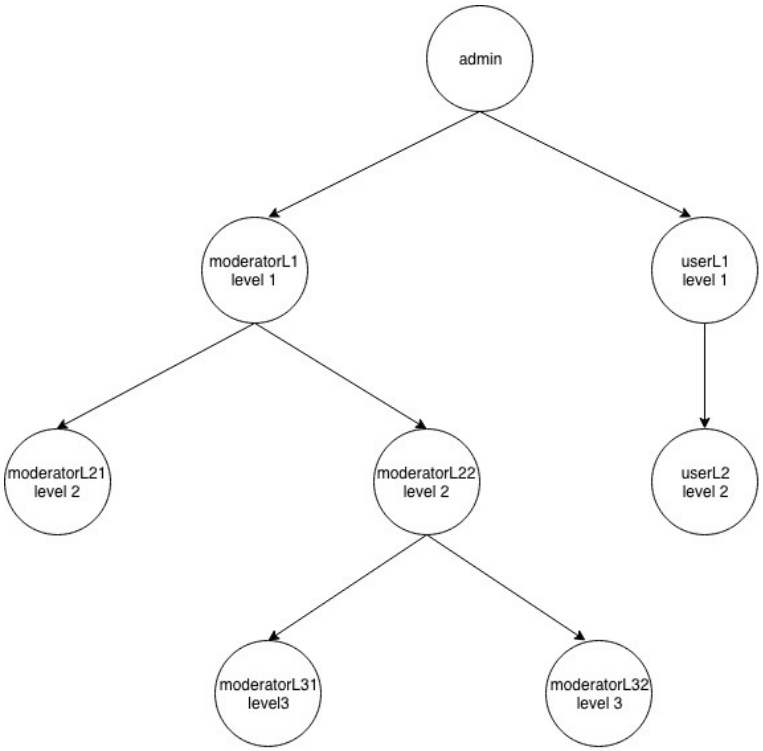

Figure 2.1: Przykładowa struktura drzewiasta roli (ta wykorzystana w przykładowej aplikacji)

2.2.1 Model roli w systemie

Do zarządzania rolami wykorzystywane są 2 tabele w bazie danych:

- roles
- role_tree

W tabeli **roles** przetrzymywane są nazwy dostępnych roli, do których przypisane są unikalne id.

Kolumny tabeli **roles**:

- id - identyfikator roli
- name - nazwa roli

id	name
1	admin
2	moderatorL1
3	moderatorL21
4	moderatorL22
5	moderatorL31
6	moderatorL32
7	userL1
8	userL2

Figure 2.2: Przykładowa tabela roles (ta wykorzystana w przykładowej aplikacji)

W tabeli **role_tree** przetrzymywana jest cała struktura drzewiasta ról. Pojedynczy rekord jest zdefiniowaniem dziecka dla danej roli. Jedna rola może nie mieć żadnych dzieci - wtedy jest najniżej w hierarchii, a także może mieć 1 lub więcej dzieci. Kolumny tabeli **role_tree**:

- id - identyfikator przypisania roli do dziecka
- parentid - identyfikator roli wyżej w hierarchii
- child - identyfikator roli niżej w hierarchii

2.2.2 Określanie uprawnień

Dla roli znajdowane są wszystkie jej dzieci, a następnie sprawdzane jest czy którykolwiek z nich nie ma wyższych uprawnień, jeżeli tak się zdarzy, to uprawnienia są dodawane.
Przykład na podstawie grafiki Figure 2.1
Mając rolę moderatorL1, sprawdzane są uprawnienia rang moderatorL22, moderatorL31, moderatorL32. Jeżeli którakolwiek z tych ról ma większe uprawnienia w jakimś zakresie niż nasza rola, to uprawnienia są dodawane.

id	parentid	child
1	1	2
2	1	7
3	2	3
4	2	4
5	4	5
6	4	6
7	7	8

Figure 2.3: Przykładowa tabela role_tree (ta wykorzystana w przykładowej aplikacji)

2.3 Listy ACL

Dostęp użytkowników do poszczególnych encji w bazie danych jest zapisany w listach ACL. Domyślnie użytkownik nie ma prawo do odczytywania wszystkich encji z bazy danych, oraz nie ma modyfikowania i dodawania nowych danych.

Listy ACL zaimplementowane są jako 2 tabele w bazie danych:

- acl
- acl_table_permission

2.3.1 Tabela acl

Określa która rola ma uprawnienia inne niż domyślne

Kolumny:

- idpermission - id uprawnienia, klucz publiczny
- idrole - identyfikator roli dla której uprawnienie definiujemy
- idrow - identyfikator rekordu w tabeli do której zmieniamy uprawnienia
- read - określa, czy można odczytywać wartości rekordu w bazie danych
- update - określa, czy można modyfikować wartości rekordu w bazie danych

idpermission	idrole	idtable	idrow	read	update
1	1	1	3	1	1
2	1	1	2	1	1
3	1	1	1	1	1
4	2	2	3	1	1
5	4	2	3	1	1
6	6	2	3	1	0
7	8	2	3	0	1
8	8	1	7	1	0

Figure 2.4: Przykładowa tabela acl (ta wykorzystana w przykładowej aplikacji)

2.3.2 Tabela acl_table_permission

Określa do których tabel użytkownik może dodawać nowe rekordy
Kolumny:

- id - identyfikator uprawnienia, klucz publiczny
- idrole - identyfikator roli dla której uprawnienie definiujemy
- idtable - identyfikator tablicy którą pozwalamy edytować

id	idrole	idtable
1	8	1
2	5	1
3	6	1
4	3	2

Figure 2.5: Przykładowa tabela acl_table_permission (ta wykorzystana w przykładowej aplikacji)

3. Instalacja

3.1 Wykorzystane technologie

- JavaScript
- Node.js
- MySQL
- Express.js
- Node Package Manager(npm)

3.2 Podłączenie modułu do własnej aplikacji

Przed przystąpieniem do podłączenia modułu należy mieć w bazie danych wszystkie table, które zostały opisane w poprzedniej sekcji(nazwy tych tabel muszą być identyczne) oraz dodatkowe, takie do których aplikujemy moduł bezpieczeństwa.

Aby podłączyć moduł do własnej aplikacji należy, w pierwszej kolejności w poprawny sposób podać dane do bazy danych do metody **securityInit(config)** z klasy **SecurityModule**:

```
securityInit({
  host: "hostname",
  user: "user",
  password: "password",
  database: "database"
});
```

Figure 3.1: Prawidłowe wywołanie

Po wykonaniu tych czynności można już wysyłać zapytania do bazy. Aby wykonać zapytanie należy najpierw ustawić role w metodzie **setRole(role)** klasy **SecurityModule**:

```
setRole("admin");
```

Figure 3.2: Prawidłowe wywołanie

Następnie należy podać samo zapytanie do metody aspektowej **sendQuery(query)** z klasy **SecurityModule**:

```
sendQuery("select * from table");
```

Figure 3.3: Prawidłowe wywołanie

4. Przykład użycia

Krok 1. Użytkownik musi wywołać metodę init w głównej klasie modułu, czyli w **SecurityModule**. SecurityModule.init(dbHost, dbUser, dbPassword, dbName);

Argumenty przyjmowane przez metodę init:

- dbHost - adres bazy danych
- dbUser - user bazy danych
- dbPassword - hasło do bazy danych
- dbName - nazwa bazy danych

Krok 2. Użytkownik musi ustawić role użytkownika poprzez wywołanie metody setRole w głównej klasie modułu **SecurityModule.setRole(role)**.

Argument przyjmowany przez motedę setRole:

- role - rola użytkownika

Krok 3. Wywołanie zapytania.

Użytkownik po poprawnym wykonaniu kroków 1 i 2 może wywołać metodę **SecurityModule.sendQuery(userQuery)**. W przypadku nie wykonania któregoś z kroków zostanie zwrócona informacja z błędem.

5. Przykładowa aplikacja

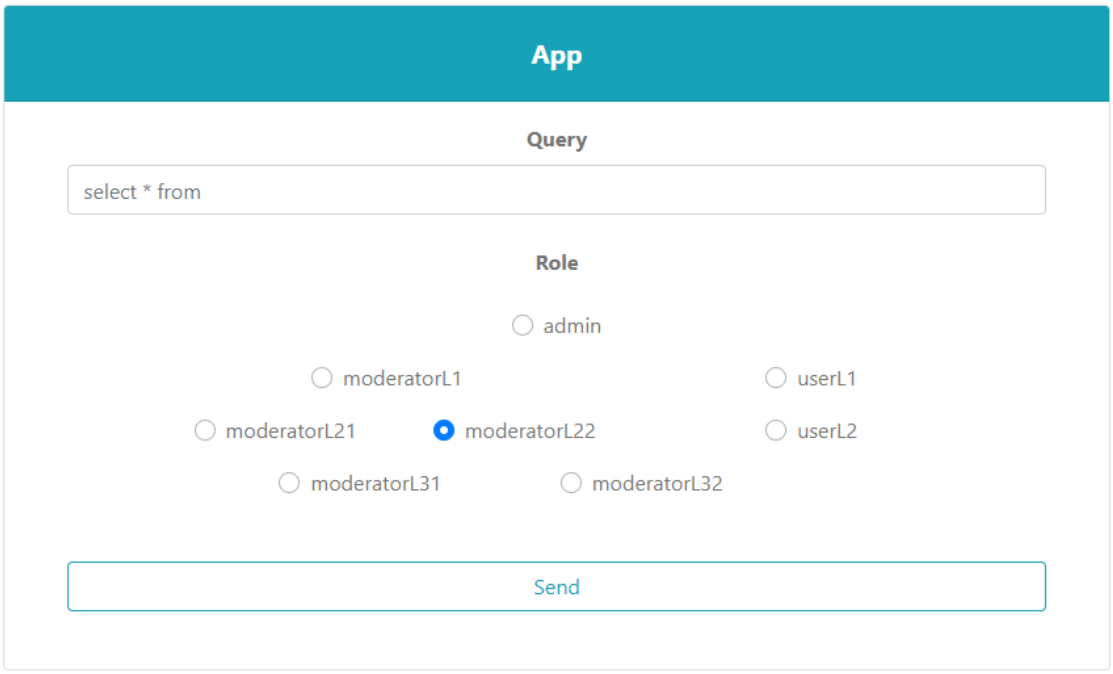

Figure 5.1: UI przykładowej aplikacji

Przykładowa aplikacja składa się z prostego formularza.

W polu tekstowym "Query" wpisujemy zapytanie jakie chcemy wysłać do bazy danych.
Obsługiwane są zapytania SELECT, UPDATE i INSERT

Następnie z listy "Role" wybieramy jaką rolę posiadamy, po czym zatwierdzamy przyciskiem Send.

Wysyłane zostaje zapytanie do serwera, który przechwytuje zapytanie, ustawia role, a następnie wysyła zapytanie do bazy danych i zwraca odpowiedź, która wyświetla się pod formularzem.

6. Zastosowane wzorce

6.1 Kompozyt

Używany będzie do przechodzenia po drzewie ról i ustalaniu upoważnień użytkownika. Do klasy **Role** przekazywana jest nazwa roli. Pola tej klasy to: nazwa roli, id roli, tablica dzieci. Id roli ustalane jest na podstawie tabeli roles. Tablica dzieci składa sie z obiektów klasy **Role**, jest ona ustalana za pomocą metody **getChildren**. Metoda **buildTree** pozawla na utworzenie drzewa zależności między rolami, dokładniej zwraca listę wszystkich rekordów we wszystkich tablicach do których poszczególne role nie mają dostępu. Następnie wyliczana jest część wspólna, która oznaczać będzie to, do czego faktycznie nie mamy dostępu.

```
constructor(role) {
    this.role = role;
    this.role_id = this.getRoleId();
    this.children = this.getChildren();
}
```

Figure 6.1: Konstruktor klasy Role

6.2 Singleton

Klasę Singleton realizuje plik **DatabaseManager**. Za każdym razem gdy będziemy chcieli połączyć się z bazą danych, nastąpi sprawdzenie czy Singleton już został utworzony, tutaj czy istnieją dane do połączenia z bazą. Jeśli tak to można połączyć się z bazą poprzez metodę **getConnection()**. Jeśli natomiast nie będzie istnieć, to w pliku **DatabaseManager** zostaną utworzone nowe dane, po utworzeniu tych danych będzie możliwość połączenia się z bazą.

```
let instance = null;

You, a few seconds ago | 2 authors (You and others)
class DatabaseManager {
  constructor() {
    this.config = null;
  }

  static getInstance() {
    if (instance === null) {
      instance = new DatabaseManager();
    }

    return instance;
  }

  setConfig(conf) {
    this.config = conf;
  }

  getConnection() {
    return new MySql(this.config);
  }
}
```

Figure 6.2: Implementacja klasy Singleton

```
let connection = DatabaseManager.getInstance().getConnection();
```

Figure 6.3: Tworzenie oraz sprawdzenie instancji

6.3 Builder

Builder wykorzystywany będzie do budowania zmodyfikowanego zapytania. Na bieżąco dodawane będą ograniczenia do zapytania, a gdy wszystkie zostaną dodane stworzone zostanie pełne zapytanie.

UML:
[h]

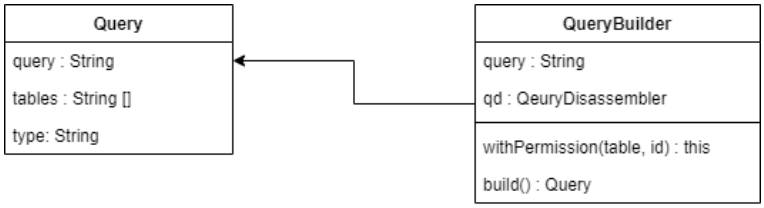

Figure 6.4: UML QueryBuildera

Przykład wykorzystania:

```
// modifying query
let queryBuilder = new QueryBuilder(queryStr);

for (let perm of permissionsFiltered) {
  queryBuilder.withPermission(perm[1], perm[0]);
}
let q = queryBuilder.build().query;
```

Figure 6.5: Przykład wykorzystania

- tworzony jest obiekt QueryBuildera
- w pętli dodajemy kolejne ograniczenia dostępu
- tworzymy finalne query