



Politechnika Łódzka

Institut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

INTERAKTYWNE NAUCZANIE BAZODANOWYCH JĘZYKÓW ZAPYTAŃ

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

Promotor: dr inż. Krzysztof Myszkowski

Dyplomant: Łukasz Ochmański

Nr albumu: 183566

Kierunek: Informatyka

Specjalność: Inżynieria Oprogramowania i Analiza Danych

Łódź, 25 marca 2015



Institut Informatyki

90-924 Łódź, ul. Wólczańska 215, *budynek B9*

tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

Streszczenie

Informatyka

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

Praca dyplomowa inżynierska

Interaktywne nauczanie bazodanowych języków zapytań

Łukasz Ochmański

Nr albumu: 183566

Celem pracy jest stworzenie interaktywnego samouczka wspierającego polskich studentów w nauce bazodanowych języków zapytań. Główną zaletą systemu jest łatwość dostępu i brak konieczności instalacji silników bazodanowych typu Oracle database client czy też konfiguracji narzędzi typu SQL Server Management Studio. Niekiedy konfiguracja przerasta umiejętności studentów, a w najlepszych wypadku pochłania godziny lub dni. Tego typu działania nie są głównym celem dydaktycznym przedmiotu o nazwie „Podstawy baz danych”. Aby skupić uwagę studentów na nauce języka, postanowiłem stworzyć ten oto serwis.

Spis treści

Rozdział 1	6
1.1. Wprowadzenie.....	6
1.2. Cel projektu	7
1.3. Historia SQL.....	8
1.4. Standardy i dialekty.....	9
1.5. Kategorie SQL.....	9
1.6. Opis struktury pracy	10
Rozdział 2	12
2.1. Założenia dotyczące projektu	12
2.2. Sformułowanie problemu informatycznego	13
2.3. Dostęp do projektu.....	14
2.4. Wymagania sprzętowe.....	15
Rozdział 3	16
3.1. Architektura systemu.....	16
3.1.1. Sposoby komunikacji	16
3.1.2. Model trójwarstwowy.....	16
3.1.3. Data Access Object (DAO)	18
3.2. Warstwa prezentacji (front-end)	20
3.2.1. HTML 5.....	20
3.2.2. JavaScript	21
3.2.3. jQuery.....	22
3.2.4. WebSocket.....	23
3.2.5. STOMP.....	24
3.2.6. JSON	25
3.2.7. BootStrap.....	25
3.2.8. CSS 3	26
3.3. Warstwa dostępu do danych (back-end).....	27
3.3.1. JVM	27
3.3.2. Java 8	27
3.3.3. JDBC.....	28
3.3.4. Apache Tomcat 8	28
3.3.5. Spring framework.....	29
3.3.6. Spring boot.....	29
3.3.7. H2 embedded.....	30
3.3.8. Gradle.....	30
3.4. Warstwa danych	30
3.4.1. Instancja STUDENT.....	32
3.4.2. Instancja ADMINISTRATOR.....	33
Rozdział 4	34
4.1. Schemat działania.....	34
4.1.1. Przykładowy workflow	34
4.1.2. Weryfikacja odpowiedzi.....	34
4.1.3. Przyznanie punktów.....	35
4.2. Opis wybranych funkcjonalności	35
4.2.1. Praca grupowa.....	35
4.2.2. Praca indywidualna.....	36
4.2.3. Forum DISQUS	36

4.2.4.	Komunikaty.....	36
4.2.5.	Materiały naukowe	36
4.2.6.	HELP	36
4.3.	Dokumentacja techniczna.....	37
Rozdział 5	43
5.1.	Przykłady działania aplikacji oraz dokumentacja użytkownika	43
5.1.1.	Uruchomienie aplikacji	43
5.1.2.	Rozwiązywanie zadań.....	44
5.1.3.	Nawigacja	48
5.1.4.	Obsługa błędów	50
5.1.5.	Forum dyskusyjne.....	50
5.1.6.	Kontakt	52
5.1.7.	Komunikaty.....	53
5.1.8.	Panel sterowania	54
5.2.	Praca zespołowa.....	58
Rozdział 6	59
6.1.	Uzasadnienie wyboru pytań.....	59
Rozdział 7	63
7.1.	Perspektywy rozwoju.....	63
7.2.	Wnioski i podsumowanie.....	66
Spis rysunków	69
Bibliografia	70
Załączniki	72

Rozdział 1

1.1. Wprowadzenie

W XXI wieku sposób używania komputera znacząco zmienił się od sposobu, w jaki używano go w poprzednim stuleciu. Przez ostatnie kilka lat stały postęp technologiczny doprowadził do wynalezienia zupełnie nowych, pasjonujących zastosowań dla systemów baz danych [1]. Główną czynnikiem, któremu zawdzięczamy dynamiczny rozwój informatyki jest popularyzacja Internetu. Łatwy dostęp do sieci spowodował szybszy przepływ informacji pomiędzy ludźmi. Twórcy systemów, mając dostęp do nieograniczonej wiedzy, pracują wydajniej i piszą więcej programów, w krótszym czasie. W związku z tym systemy są lepsze, tańsze i bardziej przyjazne użytkownikowi. Zwiększyła się również konkurencja. Jest to jednak dobra rzecz, gdyż napędza to jeszcze szybszy rozwój. Niska cena dostępu do Internetu oraz ogromny przyrost szybkości łącz oferowanych przez dostawców usługi sprawił, że nie trzeba nawet instalować aplikacji na własnej maszynie. Większość programów jest dostępnych bez kompilacji, instalacji i konfiguracji natychmiast z każdego miejsca na Ziemi. Łączy się na tyle szybko, że można korzystać z aplikacji internetowych tak, jakby były zainstalowane na maszynie lokalnej. Nie potrzebne jest posiadanie własnych kopii programów desktopowych, aby korzystać z usług. Nie potrzebne jest aktualizowanie programów, dbanie o środowisko w jakim są uruchamiane. Nie jest nawet ważne czy nasz komputer jest zainfekowany i sprawny i czy posiadamy przestrzeń na dysku twardym. Nie jest wymagana duża ilość pamięci operacyjnej. Cały ciężar obliczeń został przeniesiony w inne miejsce i nie trzeba się martwić o efektywność naszej jednostki obliczeniowej. Wszystkie wymienione problemy przestają mieć znaczenie w obecnych czasach.

Każdy użytkownik dostaje dostęp do najnowszej wersji aplikacji, nie musi się martwić o przeprowadzenie niezbędnych aktualizacji. Zawsze ma dostęp do najnowszej aplikacji. Do tej samej wersji, do której mają dostęp wszyscy. Nie trzeba się już więcej martwić, że stracimy nasze dane w pożarze czy podczas zalania klawiatury lub jeśli ukradną nam laptopa. A co jeśli kolega, który ma jedynie komputer Apple, chce korzystać z pomysłów firmy Microsoft? Nie ma problemu. Nie jest wymagany wspólny system operacyjny. Aplikacje webowe działają przecież niezależnie od platformy. Taką możliwość zawdzięczmy wspólnym standardom ustalonym w ostatniej dekadzie przez World Wide

Web Consortium, w skrócie W3C, organizację, która zajmuje się ustanawianiem standardów pisania i przesyłu stron WWW.

Aby sprostać wymaganiom stawianym przez społeczeństwo należy tworzyć nowoczesne aplikacje wykorzystujące wszystkie dobrodziejstwa naszych czasów. Nauka studenta informatyki nie powinna wyglądać tak jak dwadzieścia lat temu, gdzie instalowano wszystkie programy, systemy, silniki na swojej maszynie. W dzisiejszych czasach maszyny obliczeniowe i programy desktopowe nie są niezbędne. Większą część funkcjonalności można osiągnąć mając jedynie dostęp do nowoczesnej przeglądarki. Do możliwości takich przeglądarek należy między innymi renderowanie grafiki 3D, dźwięku, video i rozwiązywanie skomplikowanych problemów obliczeniowych. Przeglądarki potrafią nawet komunikować się bezpośrednio z kartą graficzną i kooprocesorem, są wielozadaniowe i mogą tworzyć wiele procesów w systemie oraz korzystać z wielu rdzeni procesora.

1.2. Cel projektu

Celem mojej pracy jest zaprojektowanie i zbudowanie nowoczesnej aplikacji internetowej, spełniającej standardy dyktowane przez nasze czasy, przy użyciu dostępnych technologii typu open source w celu nauczania bazodanowych języków zapytań w standardzie ANSI SQL-92. Stworzenie takiej aplikacji wymaga zmiany sposobu podejścia do pisania programów. Program musi mieć rozbudowany interfejs graficzny i symulować działanie standardowego programu desktopowego. Użytkownik nie powinien odczuć różnicy pomiędzy korzystaniem z obu wersji. Wszystkie ważne funkcjonalności powinny zostać udostępnione, a wszystkie niepotrzebne funkcjonalności powinny zostać skutecznie ukryte przed użytkownikiem tak, aby mógł się skupić na tym co istotne. Cały proces konfiguracji, szczególnie związane z tworzeniem, zapisywaniem plików, autentykacją, tworzeniem połączeń oraz integracja z platformą powinny być wykonane po stronie serwera. Należy przeprowadzić selekcję operacji wykonanych podczas całego procesu i zautomatyzować jak najwięcej z nich, zwłaszcza tych niskopoziomowych, systemowych, które nie mają związku z nauką SQL. Operacje te powinny zostać oddelegowane do serwera, a efektem tej pracy powinien gotowy serwis dostępny dla użytkowników, chcących korzystać z usług. Nie ma potrzeby, aby wszyscy wykonywali tę samą pracę wielokrotnie. Wystarczy, że zrobi to tylko autor aplikacji po stronie serwera, zwłaszcza, że operacje istnieją jedynie ze względów historycznych. Niskopoziomowe operacje na systemach plików są standardem, który każdy powinien znać. Wszystkie programy instaluje się w niemal identyczny sposób, ale przejście przez proces instalacji jest konieczne, ponieważ nikt nie wymyślił automatycznego sposobu

na zrobienie tego bez udziału człowieka. Poza tym stworzenie takiego mechanizmu wymagało by ogromnych nakładów pracy, za które niewielu klientów jest chętnych zapłacić. Ten pomysł jest po prostu jest nieopłacalny. Twórcy tych produktów założyli, że użytkownik będzie posiadał niezbędną wiedzę i pomineli tę kwestię ze względów ekonomicznych, a nie dlatego, że tak zamierzeli. Jest to ciężar, na który trzeba się przygotować i ktoś musi go ponieść. Zazwyczaj jest do tego wyznaczona osoba zwana administratorem, której jedynym zadaniem jest rozwiązywanie tego typu problemów.

Misją jaka przyświeca mi podczas tworzenia tego projektu jest wsparcie studentów w opanowaniu biegłego posługiwania się językiem SQL, którego znajomość jest kluczowa w poznawaniu niezbędnej wiedzy teoretycznej z zakresu systemów baz danych.

Użytkownicy, zdobywają kompetencje swobodnego tworzenia dowolnych kwerend SQL i umiejętności korzystania z wiedzy zawartej w bazach. Poznają metodykę i narzędzia do sprawnego przetwarzania dużych ilości danych, wyciągania tylko istotnych informacji.

1.3. Historia SQL

Rozwój relacyjnych baz danych, który miał miejsce w latach 70-tych ubiegłego wieku uwarunkował konieczność opracowania języka do manipulacji, wyciągania i obsługi danych w bazach. Systemy baz danych zmięły się znacznie od ukazania się słynnego artykułu Teda Codda. Propozycja Codda polegała na umożliwieniu prezentowania użytkownikowi danych w postaci tabel, nazywanych relacjami [2].

Pierwszym oficjalnym językiem relacyjnych baz danych, był SEQUEL (Structured English Query Language), opracowany przez pracowników firmy IBM (Raymond F. Boyce oraz Donald Chamberline). Zaimplementowany w 1973 roku w SYSTEM R był pierwszym silnikiem bazodanowym opartym o model relacyjny, jednak pierwszym komercyjnym systemem RDBSM był produkt firmy ORACLE w 1979 r. [3].

Jak sama nazwa wskazuje, SEQUEL (Structured English Query Language) to język w domyśle przyjazny dla użytkownika, służący odpytywaniu baz. Jednym z założeń była łatwość tworzenia zapytań, operacji na zbiorach za pomocą słów kluczowych w języku angielskim. Język miał być intuicyjny i prostoty. Te cechy to także założenia samego modelu relacyjnego i chyba właśnie dlatego, systemy baz danych oparte o model relacyjny podbiły świat i są do dziś dominującymi środowiskami bazodanowymi.

Jednak nazwa SEQUEL, okazała się być nazwą zastrzeżoną przez brytyjską firmę przemysłu lotniczego. Stąd została skrócona do znanej obecnie formy czyli SQL (Structured Query Language).

Najważniejszymi systemami RDBMS (Relational DataBase Management System), w których podstawowym językiem jest SQL, to oczywiście: MS SQL Server, Oracle, DB2, MySQL, PostgreSQL, Sybase.

1.4. Standardy i dialekty

Konkurencyjność rynku spowodowała konieczność ustandaryzowania języka SQL i stało się to w roku 1986, kiedy został opracowany przez ANSI pierwszy standard określany jako SQL:86. Stało się to na tyle wcześnie, że choć istnieją istotne różnice np. w nazwach implementowanych funkcji, to ogólne zasady dla relacyjnych baz danych, różnych producentów, pozostały spójne. Ma to znaczenie szczególnie podczas integracji platform i dla nas, pracujących w różnych środowiskach.

Powstały, więc dialekty językowe. Transact-SQL (T-SQL) – historycznie wprowadzony przez Sybase, rozwijany do dziś przez Microsoft w SQL Server. Inne dialekty, mające duże znaczenie na rynku to oczywiście PL/SQL (firmy ORACLE) oraz SQL/PSM (najpopularniejszy silnik relacyjny w serwisach WWW – MySQL).

Pomimo różnic w dialektach, nazwach funkcji, typach danych – istnieje szeroki wspólny mianownik – relacyjny model oparty o teorię zbiorów. Dlatego, znając T-SQL, nie ma problemu z poruszeniem się np. w bazie MySQL czy Oracle.

Standardy ANSI są regularnie aktualizowane. Od 1986 roku zostało opublikowanych szereg wersji (aktualnie obowiązująca to ANSI SQL:2011/2011), wprowadzających porządek w nowych funkcjonalnościach. Np. w SQL:2003 zostały wprowadzone standardy związane z obsługą XML.

1.5. Kategorie SQL

Istnieją grupy istotnych akronimów, które powinny być znane każdemu użytkownikowi bazy danych. W zależności od zastosowań są to :

DDL – Data Definition Language – czyli komendy dot. tworzenia, modyfikacji obiektów w bazie np. CREATE TABLE, ALTER VIEW, DROP,

DML – Data Modification Language (UPDATE, INSERT, DELETE)

DCL – Data Control Language – kontrola uprawnień (GRANT, DENY, REVOKE)

TCL – Transaction Control Language – obsługa transakcji np. BEGIN TRANSACTION, COMMIT, ROLLBACK.

I w końcu DQL – Data Querying Language – czyli polecenie SELECT

Dotyczy on tylko podzbioru języka SQL – związanego z pisaniem zapytań (kwerend).

1.6. Opis struktury pracy

Rozdział 1 Wprowadzenie do tematu. W tej sekcji jest opisana geneza powstania relacyjnych baz danych i języków zapytań oraz ogólny opis prezentowanego projektu. Opis struktury pracy. Skrócone zestawienie wszystkich rozdziałów.

Rozdział 2 Założenia dotyczące projektu, wymagana składnia, niedozwolone operacje, możliwości i ograniczenia techniczne, czego można oczekiwać od systemu, a czego nie. Przedstawienie problemu informatycznego. Sformułowanie najistotniejszych zagadnień, problemów którym trzeba się przyjrzeć i rozwiązać. Ogólne informacje dotyczące lokalizacji aplikacji, sposobów dostępu, kod źródłowego, wymaganych uprawnień. Wymagania sprzętowe Specyfikacja wymagań technicznych niezbędnych do uruchomienia aplikacji.

Rozdział 3 Opis architektury systemu. Ogólny zarys budowy aplikacji. Model graficzny. Zasada działania i funkcje pełniące przez warstwę prezentacji. Funkcje pełniące przez warstwę dostępu do danych. Opis silnika bazodanowego, możliwości, trybów pracy, protokołów komunikacji i sposobów integracji z klientami. Utworzone instancje, schematy i użytkownicy.

Rozdział 4 Schemat działania aplikacji. Zasada funkcjonowania i przepływu informacji pomiędzy poszczególnymi modułami. Opis funkcjonalności. Szczegółowe wyjaśnienie zasady działania najważniejszych funkcjonalności systemu. Szczegółowy opis implementacji. Analiza kodu źródłowego. Dokumentacja techniczna.

Rozdział 5 Przykłady działania aplikacji. Graficzna prezentacja, pokaz możliwości systemu. Prezentacja modułu do pracy grupowej.

Rozdział 6 Uzasadnienie wyboru pytań. Poruszenie tematów związanych z dydaktyką. W jaki sposób dobierano pytania i dlaczego ułożono je w takiej kolejności.

Rozdział 7 Perspektywy rozwoju. Lista planowanych funkcjonalności. Wnioski i podsumowanie.

Rozdział 8 Spis rysunków. Spis tabel. Spis listingów. Bibliografia. Załączniki

Rozdział 2

2.1. Założenia dotyczące projektu

- Aplikacja została stworzona do użytku wszystkich zainteresowanych i może być używana nieodpłatnie przez wszystkich studentów w celach dydaktycznych,
- Stworzona aplikacją typu open source i można korzystać z kodu źródłowego w celach niekomercyjnych.
- Użyte przeze mnie biblioteki i frameworki H2, Apache Tomcat, Spring są typu Open Source i korzystanie z kodu źródłowego mojej aplikacji zobowiązuje wszystkich jego użytkowników do udostępnienia całego kodu i podania autora na zasadach omówionych w licencji.
- Podczas korzystania z aplikacji należy być ostrożnym, gdyż na pewno istnieje wiele możliwości „złamania” systemu. Staralem się jednak jak najbardziej taką możliwość ograniczyć i wszystkie zmiany wykonane na głównym schemacie są odwracane przy każdym zapytaniu.
- Aby dokonać trwałych zmian należy stworzyć swój schemat i to powinno zwiększyć ochronę danych.
- Nie ma gwarancji, ani zabezpieczeń, że inny użytkownik nie usunie informacji wprowadzonych przez innego użytkownika.
- Każda skuteczna zmiana struktury bazy i danych w niej zawartych jest logowana (zakończona powodzeniem) i widoczna przez administratora systemu.
- Wszystkie próby (skuteczne i nieskuteczne) udzielenia odpowiedzi do zadań są logowane i widoczne przez administratora systemu.
- Dialekt SQL bazy danych H2 powinien być zgodny z dialektem MySQL oraz MS SQL Server.
- Baza wspiera rozróżnianie wielkości znaków, jeśli używa się cudzysłówów.
- Zarówno producent silnika bazodanowego H2 jak i autor aplikacji (Łukasz Ochmański) nie ponoszą odpowiedzialności za problemy spowodowane użytkowaniem aplikacji.
- Wspierane kodowanie to UTF-8. Możliwa jest również obsługa Unicode. Można zatem śmiało wprowadzać polskie znaki.
- Dla poprawnego działania aplikacji zalecane jest używanie najnowszej przeglądarki Firefox.
- Stabilność systemu nie jest gwarantowana. Projekt został stworzony jedynie do domowego użytku.
- W razie jakichkolwiek problemów, należy się kontaktować ze mną poprzez forum na stronie lub e-mail.

2.2. Sformułowanie problemu informatycznego

Jak już wcześniej wspomniałem głównym celem projektu jest stworzenie aplikacji wspomagającej naukę języka SQL. W obecnej chwili nie ma na polskim rynku dostępnych narzędzi w naszym ojczystym języku służących wyłącznie do tego celu. Będzie to mały serwis, którego celem jest nauczanie i interaktywna współpraca ze studentem.

W projekcie pojawia się kilka problemów, z którymi należy się zmierzyć. Pierwszym z nich jest opracowanie sposobu kierowania uczniem krok po kroku tak, jak robi to istota żywa. Student powinien budować swoje umiejętności wraz z liczbą przebytych etapów. Należy odpowiednio dobrać pytania i zwiększać ich trudność dopiero po opanowaniu poprzedniego tematu. System powinien inteligentnie dobierać pytania w zależności od poziomu wiedzy prezentowanej przez ucznia. Należałoby też zadbać, aby system wracał do tematów, które wydają się słabo opanowane w celu przypomnienia materiału. Kurs każdego z użytkowników powinien wyglądać inaczej i powinien być dostosowany do poziomu wiedzy prezentowanej przez osobę, a efekt końcowy powinien być wspólny dla wszystkich, którzy ukończą wszystkie etapy. Aplikacja powinna zostać wyposażona w system podpowiadający w razie długotrwałych problemów tak, aby nikt nie utknął na jednym z pytań.

Następną istotną kwestią jest zapewnienie bezpieczeństwa danych osobowych oraz poufności ocen. Nie można dać użytkownikom zupełnej swobody, gdyż mogliby łatwo dostać się do informacji należących do innych osób. Najważniejszym aspektem jest pozbawienie użytkowników możliwości modyfikacji swoich ocen. Trzeba rozsądnie zdecydować nad sposobami logowania, przypominania haseł i zakładania kont. Tym samym należy tak tworzyć ograniczenia, aby nie utrudniały one pracy, gdyż to było głównym założeniem całego projektu.

Jednocześnie należy zadbać, aby system nie dyskryminował nikogo. Każdy człowiek na świecie, bez względu na lokalizację stacji roboczej, sposób dostępu do sieci, platformę czy przeglądarkę powinien być w stanie poruszać się po systemie bez zakłóceń.

Kolejnym wyzwaniem jest zadbanie o trwałość danych, gdyż oceny nie mogą ginąć podczas awarii. Dane powinny być replikowane i cyklicznie backupowane. Istnieje tutaj kilka rozwiązań. Najprostszym z nich jest przeniesienie tej odpowiedzialności na samego użytkownika, poprzez umożliwienie zapisu stanu aplikacji po stronie klienta.

Największym jednak problemem jest zapewnienie stabilności systemu podczas jego długotrwałej, nieprzerwanej pracy z użytkownikami. Do tego celu będzie wymagana seria testów jednostkowych, integracyjnych oraz testy obciążeniowe, sprawdzające możliwości serwera. Trzeba również wziąć pod uwagę ryzyko wystąpienia wyścigu oraz możliwość naruszenia integralności danych, gdyż wszyscy użytkownicy dzielą wspólne zasoby. Do tego celu należy wykorzystać dostarczone przez twórców języka Java oraz bazy H2 mechanizmy ochrony przed przeplotem, jednocześnie nie ograniczając przetwarzania współbieżnego, gdyż przetwarzanie sekwencyjne sparaliżowałoby pracę całego systemu. Najskuteczniejszym rozwiązaniem wydaje się zastosowanie transakcyjności z mieszanką pessimistic locking oraz optimistic locking.

2.3. Dostęp do projektu

Adres

Wersja beta aplikacji jest dostępna pod adresem: <http://37.187.43.124>.

Godziny dostępu

W planach system będzie dostępny całą dobę. Jednak w przypadku wystąpienia błędów wymagany będzie restart, co spowoduje unieruchomienie aplikacji na nie więcej niż pięć minut.

Docelowa grupa odbiorców

W początkowej fazie wdrożenia nie będzie ograniczeń co do osób korzystających z serwisu. Każdy może anonimowo korzystać z systemu Interaktywnego Nauczania Bazodanowych Języków Zapytań (INBJZ). Docelowo będą to studenci przedmiotu „Podstawy baz danych” na wydziale Fizyki Technicznej Informatyki i Matematyki Stosowanej Politechniki Łódzkiej kierunku informatyka. W obecnej wersji można się przedstawić poprzez wpisanie swojego identyfikatora w prawym górnym rogu ekranu. Nie ma przeprowadzanej żadnej autentykacji, więc można się podać za dowolną osobę i nie ma z tego tytułu żadnych negatywnych konsekwencji.

VCS (Version Control System)

Repozytorium kontroli wersji jest prowadzone na bieżąco w systemie GIT. Można sklonować sobie całe repozytorium poprzez wydanie polecenia:

```
git clone https://code.google.com/p/inbjz/
```

Aby jednak brać czynny udział w rozwoju projektu należy mnie o to poprosić. Wtedy udostępnię zasoby.

Kod źródłowy i użyte narzędzia

Projekt został napisany w języku Java. Aby go uruchomić należy posiadać Java Development Kit 8. Następnie gotowy projekt można zaimportować do dowolnego środowiska programistycznego za pomocą systemu budowania (Build Automation Software) Gradle. Gotowy projekt jest skonfigurowany do działania w IntelliJ 14. Kod źródłowy można również podejrzeć za pomocą dowolnej przeglądarki internetowej pod adresem:

<https://code.google.com/p/inbjz/>

2.4. Wymagania sprzętowe

Przeglądarka

Wymagana przeglądarka zdolna do obsługi JavaScript, HTML5, CSS3 oraz WebSocket. Strona działa na wszystkich najnowszych przeglądarkach, ale zalecana jest Mozilla Firefox.

Strona kodowania

UTF-8

System operacyjny

Dowolny system operacyjny z graficznym interfejsem użytkownika i działającą przeglądarką.

Rozdzielczość ekranu

Zalecana rozdzielczość: 1366x768 pikseli.

Pamięć operacyjna

Zdolna uruchomić przeglądarkę Firefox czyli około 512MB

Rozdział 3

3.1. Architektura systemu

W tym rozdziale postaram się omówić główne cechy systemu i zastosowane rozwiązania do wymiany danych pomiędzy poszczególnymi modułami.

3.1.1. Sposoby komunikacji

W aplikacji są używane trzy główne protokoły komunikacji:

HTTP

Poprzez ten protokół serwowane są statyczne strony HTML, pliki CSS oraz pliki z kodem JavaScript do klienta i od klienta. Za pomocą prostych metod GET i POST można uzyskać prawie wszystkie funkcjonalności omawianej aplikacji. Jednak należy podkreślić, iż zbudowana przeze mnie aplikacja w bardzo małym stopniu opiera się na HTTP. Cały ciężar przesyłu danych postanowiłem przenieść do nowszego i bardziej wydajnego protokołu opisanego poniżej.

WebSocket

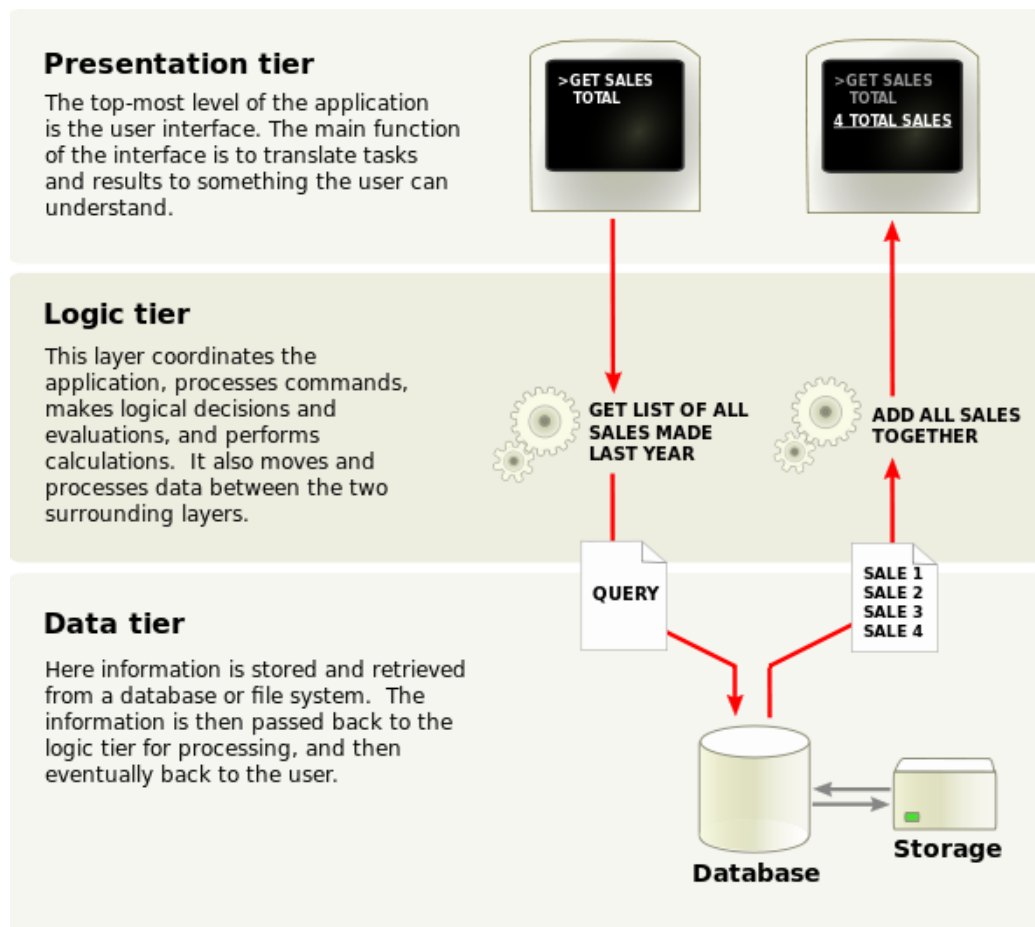
Poprzez ten protokół następuje wymiana danych pomiędzy serwerem, a klientem. Wszystkie zapytania i komendy wpisywane w polu tekstowym na stronie HTML mogą być mapowane do formatu JSON (JavaScript Object Notation). W przypadku obrazów, dźwięku lub video dane mogą być konwertowane na strumień bajtów. Następnie wysyłane do serwera na jeden z dostępnych portów bezpośrednio w niższej warstwie modelu TCP/IP w porównaniu z protokołem HTTP. Zapewnia to najwyższą efektywność przy jednoczesnym zapewnieniu pewności transmisji.

JDBC (Java Database Connectivity) komunikuje się ze sterownikiem bazy danych i tłumaczy wszystkie zapytania z postaci tekstowej na natywne komendy udostępnione przez API producenta bazy danych. Następnie dane (Result Set) są zwracane poprzez sterownik do programu wysokiego poziomu i interpretowane przez programistę, czyli mnie.

3.1.2. Model trójwarstwowy

Cała architektura projektu opiera się o klasyczny model trójwarstwowy (Three-Tier-Architecture) [4], gdzie moduły systemu zostały oddzielone i uniezależnione od siebie.

Każdy z modułów jest samodzielny i może w każdej chwili zostać wymieniony. Na rysunku 1 zaprezentowano zastosowany model:

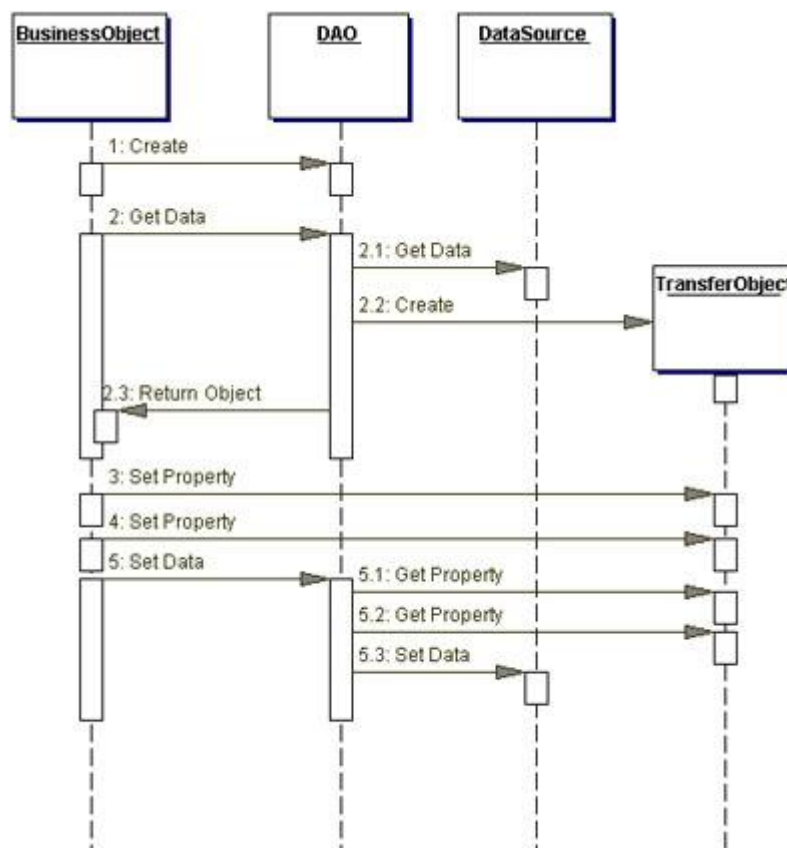


Rysunek 1: Three-Tier-Architecture

źródło: http://en.wikipedia.org/wiki/Multitier_architecture

3.1.3. Data Access Object (DAO)

Aby umożliwić modularność projektu należało użyć techniki hermetyzacji obiektów, dzięki której jest możliwa łatwa zamiana dostępnych modułów przedstawionych w punkcie powyżej. Technika polega ta na wydzieleniu fragmentów kodu, które łączą się z poszczególnymi warstwami i stworzeniu standardowego interfejsu, który jest zawsze używany do komunikacji pomiędzy modułami. Interfejs w języku Java jest to zbiór definicji metod poprzez, które programista zawsze odwołuje się w swoim programie. Implementacja metod jest dostarczana w zależności od potrzeby. W ten sposób można jedną linią kodu przełączyć źródło pobierania danych. Może zdarzyć się, że zaistnieje potrzeba stworzenia wersji systemu, która będzie pobierała dane jedynie z plików tekstowych. Innym razem zaistnieje potrzeba pobierania danych z odległego serwera poprzez gniazda TCP/IP, a jeszcze innym razem z zewnętrznego Webservice'u. Ani użytkownik, ani programista nie zauważą różnicy w funkcjonowaniu programu. Wszystkie metody będą działać i będą właściwie dostarczać dane w taki sam sposób jak wcześniej. Jedyna różnica będzie polegać na efektywności i skutkach działania programu. Ważne jest, aby zaplanować za czasu metody i klasy, które mogą ulec zmianie. Wówczas zmiany w kodzie będą minimalne. Diagram interakcji przedstawiono na rysunku 2.



Rysunek 2: Data Access Object

źródło: <http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>

W przypadku mojej aplikacji istnieje bardzo duże prawdopodobieństwo, że baza danych zostanie wymieniona, gdyż H2 jest niewystarczająca do wielu rozwiązań. Z tego powodu w moim kodzie zostały wydzielone interfejsy DatabaseDAO.java, który zawiera metody takie jak:

```

public interface DatabaseDao {

    List<String[]> executeQuery(String sql) throws SQLException;
    String executeStmt(String sql) throws SQLException;
    String update(String sql);
    String[] getLabels(String sql);
}

```

Kolejnym przykładem użycia techniki DAO jest wydzielenie interfejsu dostępu do repozytorium zadań. Nie wiadomo gdzie zadania i odpowiedzi będą się znajdowały w przyszłości. Może to być plik XML, baza danych, dokument tekstowy lub statyczna klasa Java'owa. Jak na razie wybrałem składowanie odpowiedzi w bazie danych.

```
public interface TaskRepository {

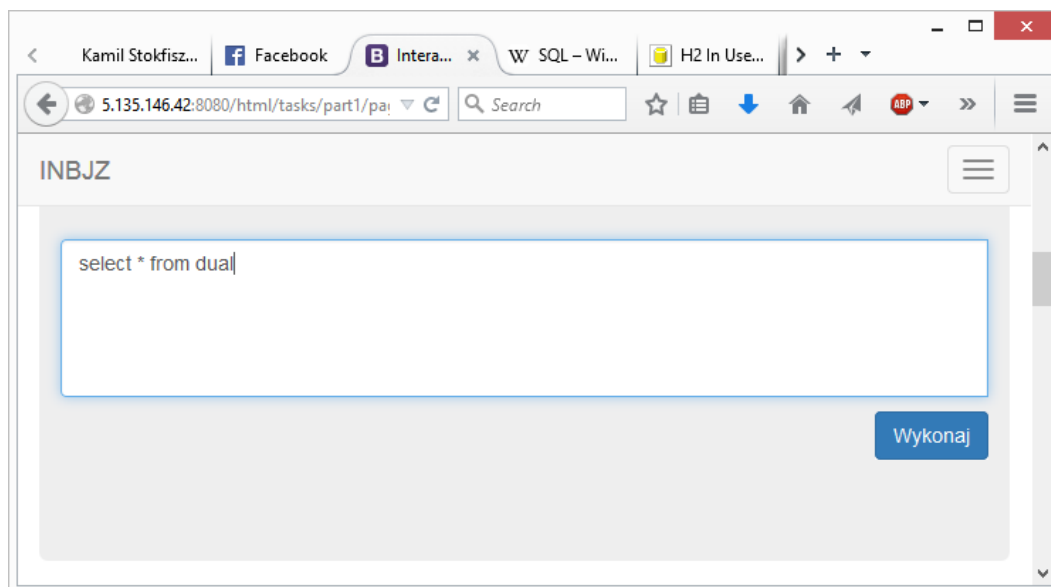
    Task getTaskById();
    Task getTask(int chapter, int number);
    List<Task> getTasks(int chapter);
    List<Task> getTasks();
}
```

3.2. Warstwa prezentacji (front-end)

W tym rozdziale zajmę się opisem warstwy prezentacji (presentation-tier) wymienioną w punkcie 7.2. Warstwa prezentacji jest w bezpośrednim kontakcie z użytkownikiem i zazwyczaj ma formę graficzną. Dawniej, zanim powstały przeglądarki internetowe i kolorowe monitory używano terminali TTY do wydawania koment. W dzisiejszych czasach odpowiednikiem takiego teminala jest przeglądarka, która wchodzi w interakcję z użytkownikiem i rozpoznaje jego zamiary. Poprzez kliknięcia na odpowiednie przyciski interpretuje chęci internauty i oddelegowuje komendy do serwera, który przetwarza żądanie, a następnie odpowiada przeglądarce. Ta z kolei wyświetla żądany rezultat.

3.2.1. HTML 5

Rysunek 3 przedstawia warstwę prezentacji wykonaną w technologii HTML 5



Rysunek 3: Warstwa Front-end

HTML to język znaczników, który pozwala opisać strukturę strony internetowej. HTML przeszedł przez wiele wersji. Początkowo w pierwszej publicznie dostępnej specyfikacji w internecie (nazwanej uprzednio „HTML Tags”) zawierała 22 znaczniki, z których do

dzisiaj nadal używanych jest 13. Pierwsza wersja była oparta o język SGML (ang. Standard Generalized Markup Language). Kolejne wersje na przestrzeni 5 lat były ciągle zmieniane aż do 1997 roku i wersji HTML 4.0. Jedną z wersji, która najdłużej wyznaczała standard w sieci była specyfikacja języka HTML 4.0. Od tamtej pory pojawiło się także kilka odmian HTML 4.0 takich jak np. XHTML 1.0. Wprowadzenie HTML5 zajęło dużo czasu, ze względu na dopracowanie HTML5 do poziomu, który mógłby być akceptowalnym standardem używanym przez wiele rodzajów urządzeń. Wersja HTML5 została zatwierdzona cztery miesiące temu w październiku 2014 roku. HTML5 zawiera w obecnej specyfikacji 89 znaczników. Jedną z głównych zalet języka, jest niezależność od systemu operacyjnego oraz sprzętu komputerowego. Pozwala to uruchamiać aplikacje HTML na szerokiej gamie urządzeń specjalistycznych i konsumenckich.

3.2.2. JavaScript

Język JavaScript jest prawdopodobnie najczęściej używanym językiem programowania na świecie. Do zalet języka należy prostota składni, łatwość dostępu i wizualny efekt działania. Najczęściej JavaScript używa się do wykonywania animacji na stronie HTML. Niektórzy twierdzą, że kolejna wersja języka JavaScript odegra ważną rolę także w innych dziedzinach, gdyż przyrost mocy obliczeniowej urządzeń elektronicznych jest tak duży, że prawie każdy procesor jest w stanie poradzić sobie z interpretowaniem kodu JavaScript w czasie rzeczywistym.

W JavaScript istnieje też realny problem: JavaScript jest absurdalnie liberalnym językiem. Zamierzeniem twórcy tego języka było sprawienie, aby był on jak najłatwiejszy w użyciu dla początkujących programistów. Jednak w rzeczywistości to tylko utrudnia znajdowanie problemów w programach, ponieważ system ich nie pokazuje [5].

Drugim problemem jest to, że JavaScript jest kompilowany dynamicznie, co oznacza, że nie można dowiedzieć się o istniejących problemach przed uruchomieniem. Brak silnego typowania utrudnia rozpoznanie oczekiwanych wartości funkcji. Obiekty są rzutowane na różne typy w zależności od sytuacji.

Mimo swojej nazwy, język JavaScript ma niewiele wspólnego z językiem Java. Zbieżność ta jest wynikiem marketingowych zabiegów i nie powstała w wyniku racjonalnego wyboru. W 1995 r., gdy firma Netscape opublikowała JavaScript, język Java był intensywnie promowany i zdobywał ogromną popularność. W ten sposób powstała ta nazwa.

Gdy obsługę JavaScriptu wprowadzono także w innych przeglądarkach niż Netscape, sporządzono dokument zawierający opis, jak dokładnie ten język powinien działać. Język, który w nim opisano został nazwany ECMAScript, po nazwie organizacji standaryzacyjnej, która zajęła się napisaniem tego standardu.

Standard ECMAScript opisuje język programowania ogólnego przeznaczenia i nie ma w nim ani słowa o jego integracji z jakąkolwiek przeglądarką internetową. W związku z tym JavaScript to ECMAScript plus dodatkowe narzędzia do manipulowania stronami internetowymi i oknami przeglądarki.

JavaScript wciąż ewoluuje. W najnowszych przeglądarkach obsługiwana jest wersja ECMA Script 5. W czerwcu 2015 planowane jest wprowadzenie nowej wersji ECMA Script 6.

3.2.3. jQuery

Do 2005 roku JavaScript, choć był dostępny w przeglądarkach internetowych, służył najczęściej jedynie do tworzenia bardzo prostych animacji, walidacji formularzy lub projektowania rozwijanego menu. Te nieskomplikowane zadania nie wymagały użycia specjalnych bibliotek, bo najczęściej konieczna ilość kodu mieściła się w kilkunastu wierszach [6].

Wraz z popularyzacją technologii AJAX, a także bogatych, interaktywnych interfejsów aplikacji internetowych, objętość kodu drastycznie wzrosła. Posługiwanie się językiem JavaScript dodatkowo utrudnił fakt, że przeglądarki internetowe (w szczególności Internet Explorer) w różny sposób implementowały szczegóły dostępu do dokumentu HTML (model DOM), deklaracje zdarzeń lub tworzenie obiektu XMLHttpRequest.

Mimo to apetyt deweloperów rósł, a JavaScript nie zawsze potrafił za nim nadążyć. Zapaleńcom dawał się we znaki brak kilku podstawowych funkcji znanych z innych języków, choćby wyszukiwania, czy dany obiekt znajduje się w tablicy oraz usuwania z tekstu spacji.

Wymienione utrudnienia stały się powodem powstania kilkunastu bibliotek przeznaczonych dla popularnego języka skryptowego. Jedne starały się rozwiązać palące problemy, rozszerzając wbudowane obiekty, inne udostępniały coś na kształt dodatkowych klas i modułów. Większość jako swój podstawowy cel stawiała ułatwienie obsługi drzewa DOM i ukrycie różnic w interpretacji kodu przez przeglądarki.

Obecnie biblioteki JavaScript można podzielić na dwa rodzaje: ułatwiające dynamizację istniejącego kodu HTML poprzez uproszczenie dostępu do DOM, CSS i AJAX oraz definiujące widgety i budujące cały interfejs użytkownika bezpośrednio w JavaScript. jQuery należy do pierwszej z wymienionych kategorii – udostępnia interfejs doskonale współpracujący z „klasycznymi” dokumentami HTML. Jeśli chcemy tworzyć aplikacje internetowe generujące interfejs użytkownika w JavaScript (jak np. Gmail), prawdopodobnie wygodniej będzie skorzystać z innych bibliotek, np. Dijit lub Ext-JS.

3.2.4. WebSockets

WebSockets służy do nawiązywania połączenia dwukierunkowego przez TCP. Protokół ten został ustandaryzowany przez organizację IETF (ang. Internet Engineering Task Force) w 2011 roku. Web Socket może być zastosowany wewnątrz każdego rodzaju klientów i serwerów internetowych. Całe połączenie przez protokół jest odseparowane od protokołu HTTP. Co pozwala na tworzenie wszelkiego rodzaju aplikacji czasu rzeczywistego. W konsekwencji komunikacja między przeglądarką użytkownika a serwerem może być realizowana w czasie rzeczywistym bez narzutu nagłówków protokołu HTTP. Wraz z wprowadzeniem WebSockets zaprezentowano dwa nowe schematy URI - ws: i wss: do połączeń szyfrowanych.

Do tej pory aplikacje bazujące na HTML komunikowały się z serwerem synchronicznie: zapytanie i odpowiedź. Jeśli serwer potrzebował więcej czasu na wykonanie zadania to można było czekać albo dokonywać co pewien czas zapytań o stan procesu (ang. polling; metoda przeglądania). Dzięki dwukierunkowej komunikacji powstała możliwość powiadamiania przeglądarki o chęci zmiany za pośrednictwem jednego gniazda TCP. Za pomocą standardowego protokołu HTTP było to dotychczas niemożliwe. W odróżnieniu od polling nowa metoda nosi nazwę push, ponieważ chęć wysłania wiadomości nie musi być poprzedzona żądaniem. Serwer wypycha wiadomość do klienta bez jego woli, a nie serwuje tak jak dawniej.

Aby używać technologii WebSocket wymagana jest przeglądarka, która wspiera ten protokół oraz serwer. W moim przypadku serwerem jest Apache Tomcat 8, który już od wersji 7 posiadał wbudowane wsparcie.

Aby ustanowić połączenie z poziomu przeglądarki internetowej w aplikacji web'owej musimy wykonać jedną czynność przez HTTP, którą jest „handshake”. Handshake zwykle wykonywany jest na porcie 80, aby serwer HTTP mógł go przetworzyć. Przykładowa próba połączenia widnieje poniżej.

Zapytanie klienta przez HTTP:

```
GET / chat HTTP /1.1
Host : server . example . com
Upgrade : websocket
Connection : Upgrade
Sec - WebSocket - Key : x3JJHmDL1EzLkh9GBhXDw ==
Sec - WebSocket - Protocol : chat , superchat
Sec - WebSocket - Version : 13
Origin : http :// example . com
```

Odpowiedź serwera przez HTTP, zawierająca pole „Sec-WebSocket-Accept”:

```
HTTP /1.1 101 Switching Protocols
Upgrade : websocket
Connection : Upgrade
Sec - WebSocket - Accept : HSmrc0sMIYUkAGmm5OPpG2HaGWk =
Sec - WebSocket - Protocol : chat
```

W zapytaniu klienta pole klucza „Sec-WebSocket-Key” to losowa liczba enkodowana skrótem base64. W odpowiedzi na zapytanie serwer dokonuje konkatencji klucza z unikalnym polem GUID. Następnie wartość jest zakodowana algorytmem skrótu SHA-1, ponownie enkodowana przez base64 i zwracana w odpowiedzi w polu „Sec-WebSocket-Accept”. Połączenie na tym etapie jest ustanowione.

Dane mogą być wymieniane w ramach z niewielkim narzutem na nagłówek i payload. Długość wysyłanych danych może być dowolna, aż do odczytania końcowego bitu FIN. Protokół używa prefixów ws:// oraz wss:// - oznaczają one połączenie Web Socket oraz Web Socket Secure, które są odpowiednikami HTTP oraz HTTPS, czyli połączenia nieszyfrowanego oraz szyfrowanego.

3.2.5. STOMP

STOMP w skrócie: Simple (or Streaming) Text Orientated Messaging Protocol.

STOMP dostarcza kompatybilny format wiadomości, dzięki któremu wszyscy klienci Stomp mogą komunikować się z brokerami (Stomp Message Brokers) w łatwy sposób, bez względu na stosowany język, używaną platformę, czy rodzaj istniejącego brokera.

STOMP jest bardzo prostym i łatwym do wdrożenia protokołem, wzorującym się na protokole HTTP. Być może zimplementowanie tego protokołu po stronie serwera nie jest

trywialne, ale na pewno niewielkim nakładem sił możliwe jest stworzenie modułu klienta, który połączy się z usługą, w bardzo przystępnym czasie. Dowodem na moje słowa jest fakt, że wystarczy użyć usługi Telnet, aby zalogować się do dowolnego brokera Stomp i wejść z nim w interakcję.

Wielu deweloperów twierdzi [7], że udało im się napisać klienta Stomp w kilka godzin, w ich konkretnym języku na ich środowisku na ich platformie. Zatem jeśli środowisko programistyczne nie oferuje narzędzi do współpracy z STOMP, można bez większego problemu napisać takie narzędzie samemu.

3.2.6. JSON

JSON, JavaScript Object Notation jest formatem tekstowym, bazującym na podzbiorze języka JavaScript. Typ MIME dla formatu JSON to application/json. Format został opisany w dokumencie RFC 4627 [8].

JSON jest jednym z nieformalnych sposobów przekazywania danych do aplikacji opartych o AJAX [9]. W typowych przypadkach dane w formacie JSON są pobierane z serwera jako tekst przy wykorzystaniu obiektu XMLHttpRequest języka JavaScript, a następnie przekształcane w obiekt. Tekst powinien być kodowany za pomocą UTF-8, który jest w JSON domyślny.

Komunikat JSON jest literałem obiektu języka Javascript, który w tym języku jest tablicą asocjacyjną. Wszystkie dane są zmiennymi (nie stanowią kodu wykonywalnego), a nazwy składników (właściwości) obiektów są otoczone cudzysłowami. Wartości mogą być typu string (napis otoczony cudzysłowem), number (liczba typu double), stanowić jedną ze stałych: false null true, być tablicą złożoną z takich elementów lub obiektem. Obiekty i tablice mogą być dowolnie zagnieżdżane. Cały komunikat jest kodowany w unikodzie i domyślnie jest to UTF-8.

3.2.7. Bootstrap

Bootstrap powstał w 2010 roku i swoje istnienie zawdzięcza ówczesnym deweloperom pracującym przy Twitterze. Idea tego frameworka jest całkiem prosta i sprowadza się do dostarczenia elastycznych reguł CSS do tworzenia szybkich layoutów. Pewnym uzupełnieniem jest kod JavaScript, dzięki któremu możemy dodać naszym wystylizowanym elementom trochę dynamiki [10].

CSS przygotowany przez autorów frameworka obejmuje większość znaczników HTML. Część z nich, tych bardziej podstawowych, odpowiedzialnych za formatowanie tekstu

posiada style przypisane bezpośrednio do konkretnych tagów. W automatyczny sposób sformatowane mamy więc np. nagłówki, paragrafy, czy też cytaty. W pozostałych przypadkach musimy zastosować konkretne klasy.

Twórcy Bootstrapa pomyśleli o naprawę wielu różnych problemach które spadają na webdeveloperów i dostarczyli nam klasy dzięki którym stworzymy praktycznie wszystko, poczynając od ładnych przycisków, poprzez piękne tabele i formularze, aż do całych responsywnych layoutów.

3.2.8. CSS 3

Kaskadowe arkusze stylów (CSS - Cascading Style Sheets) są nieodłącznym elementem współczesnej wersji języka HTML. Język CSS odpowiada za wizualną prezentację stron internetowych w przeglądarkach.

Można wprawdzie stworzyć witrynę, używając jedynie czystego HTML, jednak będzie to witryna prosta i mało atrakcyjna. Style pomogą znacznie ją wzbogacić, dlatego ich znajomość jest dzisiaj niezbędna. Ponadto stosowanie stylów oszczędza wiele pracy koniecznej dawniej do sformatowania dokumentów za pomocą tradycyjnych technik - nierzadko zmiana jednego drobnego parametru w arkuszu stylów pozwala zmienić wygląd całej witryny.

Zaledwie kilka lat temu twórcy stron WWW korzystali z języka HTML i wyłącznie za jego pomocą „rzeźbili” ostateczny kształt projektowanej witryny. Pomimo ogromnych ograniczeń, jakie oferowała ta metoda, powstawały bardzo ciekawe projekty łamiące wszelkie ograniczenia.

Wraz z wprowadzeniem pierwszej specyfikacji kaskadowych arkuszy stylów, a później jej drugiej odsłony twórcy stron złapali wiatr w żagle. Od teraz języki HTML oraz jego bezpośredni następca XHTML stały się jedynie zbiorem elementów odpowiedzialnych za określanie właściwości poszczególnych składników strony, np. nagłówków, akapitów czy tabel. Natomiast cały proces formatowania ich wyglądu lub umiejscowienia na stronie został zlecony kaskadowym arkuszom stylów. Rozwiązanie takie pozwoliło na znaczną poprawę komfortu tworzenia stron oraz odchudzenie kodu [11].

Zacięta konkurencja na rynku przeglądarek doprowadziła do sytuacji, w której ich autorzy zostali zmuszeni do ścisłego trzymania się różnych specyfikacji i natychmiastowego wprowadzania nowości i poprawek. Świetnym przykładem działania konkurencji na rynku

przeglądarek jest fakt, że wiele z nich aktualnie obsługuje elementy wstępnej wersji specyfikacji CSS 3, która nie została jeszcze skończona i oficjalnie zatwierdzona.

3.3. Warstwa dostępu do danych (back-end)

3.3.1. JVM

Java Virtual Machine (JVM) to rodzaj wirtualnego komputera, który ma swój zestaw rejestrów, zestaw instrukcji, stos i pamięć dla programów.

Dzięki standaryzacji maszyny wirtualnej, programy napisane w Javie są uniwersalne, tzn. wykonują się identycznie w każdym systemie operacyjnym.

Programy napisane w Javie są kompilowane do poziomu kodu pośredniego, nazywanego kodem bajtowym Javy (bytecode). Kod bajtowy jest interpretowany przez wirtualną maszynę JVM do postaci programu wykonywalnego dla danego systemu operacyjnego.

3.3.2. Java 8

Java 8 jest przełomowym wydaniem SDK (Software Development Kit) z wielu powodów. Zamierzam wymienić dwa, które są moim zdaniem najważniejsze. Wprowadzono Stream API, które pozwala wykonywać operacje na zbiorach podobnie jak w językach deklaratywnych typu SQL. Drugim powodem jest wprowadzenie wyrażeń Lambda, które zrewolucjonizowały sposób pisania programów w języku Java. Dodatkowo zamieszczam listę najistotniejszych zmian i wprowadzonych udogodnień w wersji ósmej:

- 101 Generalized Target-Type Inference
- 103 Parallel Array Sorting
- 104 Annotations on Java Types
- 107 Bulk Data Operations for Collections
- 109 Enhance Core Libraries with Lambda
- 117 Remove the Annotation-Processing Tool (apt)
- 120 Repeating Annotations
- 122 Remove the Permanent Generation
- 126 Lambda Expressions & Virtual Extension Methods
- 135 Base64 Encoding & Decoding
- 150 Date & Time API
- 153 Launch JavaFX Applications
- 155 Concurrency Updates
- 160 Lambda-Form Representation for Method Handles
- 161 Compact Profiles
- 162 Prepare for Modularization
- 174 Nashorn JavaScript Engine
- 184 HTTP URL Permissions
- 185 JAXP 1.5: Restrict Fetching of External Resources

3.3.3. JDBC

Java, jako uniwersalny język programowania, daje możliwość dostępu do baz danych. Przenośność aplikacji bazodanowych tworzonych w Javie zapewnia interfejs JDBC opracowany przez firmę Sun Microsystems w 1996 roku. Umożliwia on konstruowanie i wykonywanie poleceń SQL'owych z poziomu kodu Javy [12]. Dzięki JDBC aplikacje bazodanowe napisane w Javie są niezależne od sprzętu oraz stosowanej bazy danych (niezależność od systemu operacyjnego zapewnia sama Java). Należy jednak zaznaczyć, iż nadal możliwe jest stworzenie specyficznych wywołań lub użycie typów danych, występujących tylko w przypadku konkretnej bazy danych.

3.3.4. Apache Tomcat 8

Apache Tomcat jest serwerem aplikacji na licencji Apache Software License pozwalającym uruchomić nam aplikacje webowe napisane w JAVA. Spełnia specyfikację firmy SUN jeśli chodzi o Java Servlets oraz Java Server Pages [13]. Zaletą tego serwera jest to, że jest w całości napisany w JAVA stąd jest dostępny na wszystkie platformy. Tomcat jest kontenerem webowym przechowującym aplikacje napisane w Javie EE (Java Enterprise Edition). Jest to Javowski odpowiednik serwera Apache HTTP (Apache HTTP jest wykorzystywany w udostępnianiu stron napisanych w języku HTML). Dostęp do aplikacji

jest identyczny jak w przypadku zwykłych stron internetowych. Z punktu widzenia użytkownika/klienta, korzystanie z aplikacji nie różni się od tych które zostały napisane w języku PHP, Java czy Python, ponieważ Tomcat bazuje na protokole HTTP.

3.3.5. Spring Framework

Spring Framework jest biblioteką tworzenia aplikacji dostępną na zasadach open source, która umożliwia uproszczenie procesu tworzenia oprogramowania biznesowego za pomocą języka Java. Biblioteka ta pozwala osiągnąć ten cel przez udostępnienie programistom modelu komponentów oraz zbioru postych i spójnych API, które w efektywny sposób izolują ich od złożonego kodu podstawowego, wymaganego w skomplikowanych aplikacjach [14].

W ostatnich dziewięciu latach zakres tej biblioteki znacznie się zwiększył, ale pomimo tego pozostała prosta i łatwa w użyciu. Obecnie składa się ona z około dwudziestu modułów, które mogą być podzielone na 6 podstawowych obszarów funkcjonalnych:

- Dostęp do danych (integracja),
- Sieć WWW,
- Programowanie aspektowe (AOP),
- Instrumentacja,
- Podstawowy kontener,
- Testy

3.3.6. Spring Boot

Spring Boot jest zbiorem ponad czterdziestu mikroprojektów dostępnych na oficjalnej stronie Spring. Są one bardzo wygodne dla osób rozpoczynających pracę ze Springiem. Gotowe pakiety pozwalają na uruchomienie aplikacji w mniej niż piętnaście minut. Mikroprojekty Spring Boot nadają się świetnie do integracji z istniejącymi projektami. Można łatwo dołączyć je do swojej aplikacji używając narzędzi do automatycznego budowania, takich jak Maven, Gradle. Nie trzeba już szukać w internecie przykładów zastosowania i modyfikować ich do własnych potrzeb. Spring Boot dostarcza zbiór działających aplikacji, których nie trzeba modyfikować. Można je jedynie rozwijać według własnych potrzeb. Nie ma również konieczności zawierania wymaganych bibliotek, konfigurowania zewnętrznych serwerów, czy nawet środowisk programistycznych. Wystarczy dołączyć zależność do pliku POM w przypadku Maven lub .gradle w przypadku Gradle i aplikacja powinna zadziałać natychmiast.

3.3.7. H2 embedded

H2 jest jednym z najlżejszych systemów zarządzania relacyjną bazą danych. H2 została całkowicie napisana w języku Java i jej głównymi zaletami są: wieloplatformowość i szybkość działania. Baza H2 jako jedyna może działać całkowicie wewnątrz wirtualnej maszyny Javy. Może zostać wbudowana w aplikację i przechowywana w pamięci operacyjnej. Rozmiar samej biblioteki bazy danych wynosi jedynie 1.5MB i jest dostępna w postaci pliku z rozszerzeniem *.jar. H2 jest również dostępna w pakiecie SpringBoot, co gwarantuje bardzo niską barierę wejścia. Doświadczonemu programiście wystarczy zaledwie dziesięć minut, aby znaleźć bibliotekę, zainstalować i uruchomić aplikację komunikującą się z bazą H2 poprzez JDBC.

3.3.8. Gradle

Gradle jest potężnym narzędziem do budowy projektów. Gradle łączy w sobie cechy innych narzędzi takich jak Ant oraz Maven, eliminując przy tym nadmiarowość kodu jaki należy napisać podczas konfiguracji projektu. Do tej pory trudność sprawiało zaimplementowanie jakiegokolwiek algorytmu w skrypcie budującym; trudno było wyrazić nawet tak podstawowe konstrukcje jak if czy for; ciężko było wykonać nawet pozornie proste zadania, których wykonywanie nie zostało przewidziane przez developerów Ant'a ani Maven'a. Potrzebna była dogłębna znajomość tych narzędzi, by zrozumieć złożony skrypt budujący. Poza tym podejście "build by convention" nie jest wspierane przez Ant, albo skutecznie utrudnia konfigurację w przypadku Maven. Ponadto wsparcie dla budowy projektów wielomodułowych jest niewystarczające, skrypty budujące są niepotrzebnie duże i nieczytelne z uwagi na narzut samego języka XML [15].

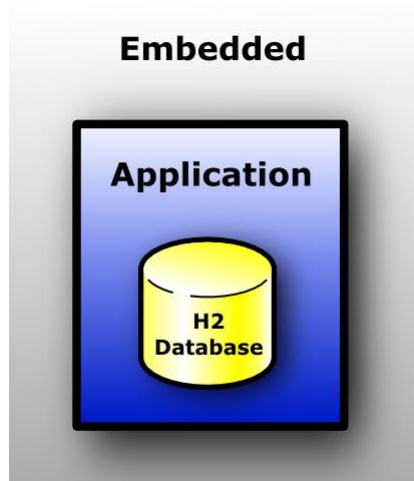
Składnia zaczerpnięta z języka Groovy pozwala na większą elastyczność i daje większe możliwości. Można dosłownie napisać program, który będzie czyścił, instalował, kompilował i budował warunkowo, w zależności od okoliczności, czasu i miejsca nasze projekty dokładnie tak jak sobie tego życzymy. Ponadto w Gradle zostały podjęte zdecydowane kroki, aby wyeliminować problemy pojawiające się w poprzednich wersjach narzędzi do budowy projektów. Do tej pory należało uważnie umieszczać zależności, które mogły powodować występowanie cykli. W Gradle użyto skierowanego acyklicznego grafu do rozwiązania problemu.

3.4. Warstwa danych

Jak już wcześniej wspomniałem w projekcie użyto bazy danych H2. Technologia została wybrana ponieważ zapewnia najszybszy dostęp do aplikacji, które działają na tej samej

instancji wirtualnej maszyny Java JVM. Zgodnie ze specyfikacją producenta H2 może działać w trzech trybach:

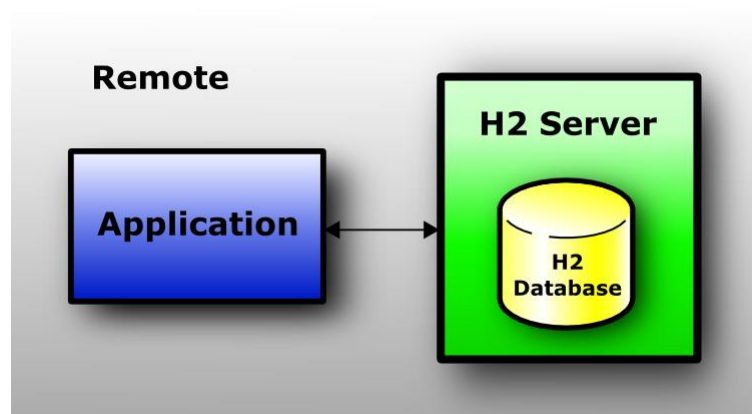
- Wbudowany (Rysunek 4)



Rysunek 4: Embedded mode

Ten tryb pozwala na najszybszy dostęp do danych, gdyż cała baza może być przechowywana w pamięci operacyjnej.

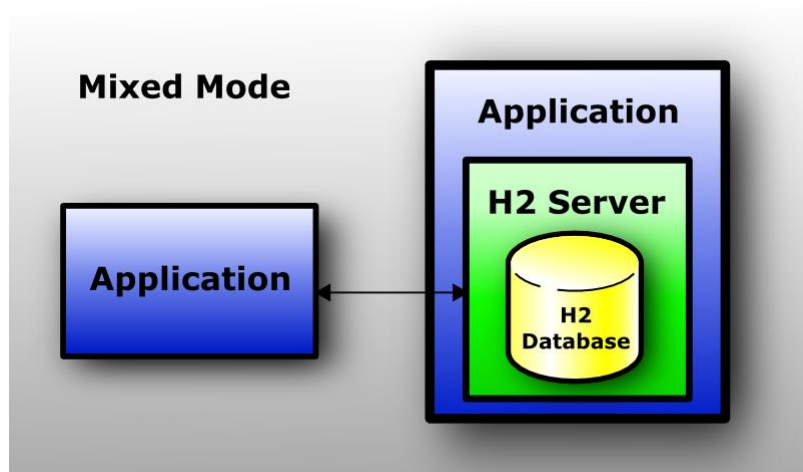
- Serwer (Rysunek 5)



Rysunek 5: Server mode

Standardowy tryb typu klien-serwer, gdzie jest dostępna liczba połączeń i należy się łączyć poprzez sterownik JDBC lub ODBC. Komunikacja odbywa się poprzez protokół TCP/IP. W razie nadmiaru żądań klienci są kolejkowani i muszą czekać na odpowiedź.

- Mieszany (Rysunek 6)



Rysunek 6: Mixed mode

Ten tryb pozwala na korzystanie z zasobów bazy danych poprzez kilku użytkowników jednocześnie. Mogą to być osobni klienci, czy osobne procesy w systemie łączące się przez standardowy sterownik JDBC i jednocześnie sama aplikacja działająca wewnątrz tego samego procesu w tej samej wirtualnej maszynie Java. Oczywiście proces korzystający z zasobów z wewnątrz JVM będzie działał kilkadziesiąt, może nawet kilkaset razy szybciej.

3.4.1. Instancja STUDENT

Do potrzeb projektu INBJZ należało użyć dwie osobne instancje bazy danych. Jedna z nich będzie służyć do manipulacji i nauki i będzie w całości dostępna dla studentów. Wewnątrz bazy utworzono dwóch użytkowników: SA (Super Administrator) oraz Student. W późniejszym etapie implementacji zdecydowałem się zaniechać korzystania z użytkownika Student, gdyż nie przynosi on żadnych korzyści. Aby studenci mogli używać wszystkich funkcji systemu, w tym tworzyć tabele, schematy, indeksy zdecydowałem się odblokować wszystkie zasoby tej instancji bazy na ich użytek. Natomiast ochrona głównego schematu została oddelegowana do samej warstwy dostępu do danych. To aplikacja dba o spójność danych testowych. Do tego celu wykorzystałem prostą metodę odwracania wszystkich zmian po każdej transakcji w głównym schemacie. W ten sposób żadnej użytkownik systemu nie ma szans usunięcia choćby krotki danych. Wszystkie dane są zachowane w skrypcie, który jest przechowywany na samym serwerze i może być modyfikowany tylko przez programistę lub administratora serwera.

3.4.2. Instancja ADMINISTRATOR

Druga instancja służy do przechowywania pytań, odpowiedzi, logów, loginów, haseł i innych używanych przez aplikację. Baza została podzielona na dwóch użytkowników, podobnie jak w poprzednim przypadku są to SA (Super Administrator) i Student. W tym przypadku oba konta są aktywnie używane. Konto Student ma ograniczone uprawnienia i może wykonywać jedynie selecty na czterech tabelach: LOGS, TASKS, STUDENTS, LOGIN_EVENTS.

Konto SA ma natomiast nieograniczone uprawnienia i dodatkowo dostęp do oczekiwanych odpowiedzi w tabelach LOGGED_ANSWERS i ANSWERS. To konto jest używane przez aplikację bezpośrednio w celu weryfikacji odpowiedzi oraz służy do zapisywania aktywności studentów.

Dzięki instancji ADMINISTRATOR można uzyskać informacje o udzielonych odpowiedziach. Można zobaczyć liczbę błędów, liczbę poprawnych odpowiedzi. Można także pogrupować informacje według nr albumu, jednocześnie ograniczając datę i godzinę dodania wpisu. W ten sposób powstanie bardzo spójny raport przydatny wykładowcy, na podstawie którego można dokonać oceny pracy uczniów. Wyniki można sortować i agregować zgodnie z zasadami panującymi w SQL. Możliwości są zatem nieograniczone. Można również pozyskać dane historyczne na temat wyników studentów z poprzednich lat i zestawzić je z wynikami osiąganymi przez obecnych. W ten sposób powstanie bardzo ciekawa statystyka, która może stać się cenną informacją dla prowadzącego zajęcia.

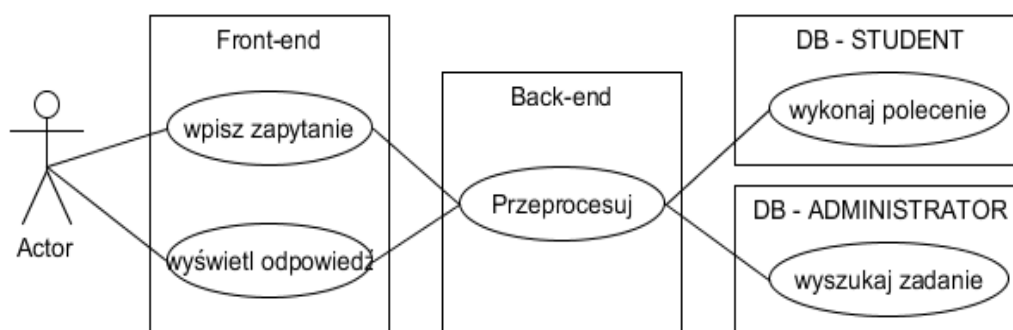
Rozdział 4

4.1. Schemat działania

W tym rozdziale opiszę dokładny sposób funkcjonowania i przepływu informacji pomiędzy poszczególnymi modułami oraz przedstawię sposób realizacji podanych rozwiązań.

4.1.1. Przykładowy workflow

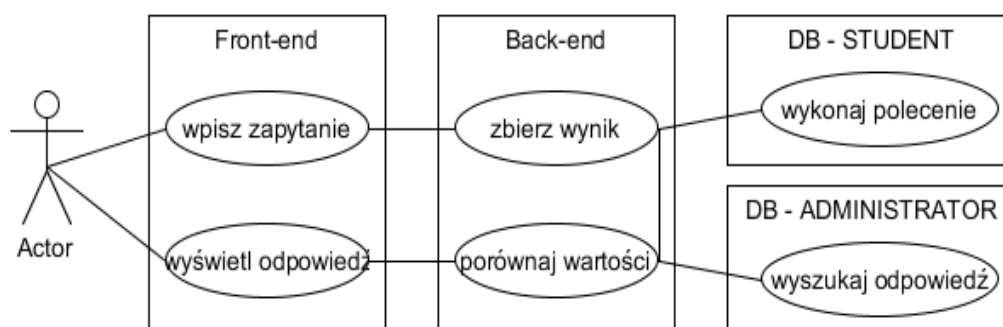
Na rysunku 7 przedstawiono ogólny sposób przepływu informacji pomiędzy modułami systemu.



Rysunek 7: Diagram przypadków użycia – Przykładowy workflow

4.1.2. Weryfikacja odpowiedzi

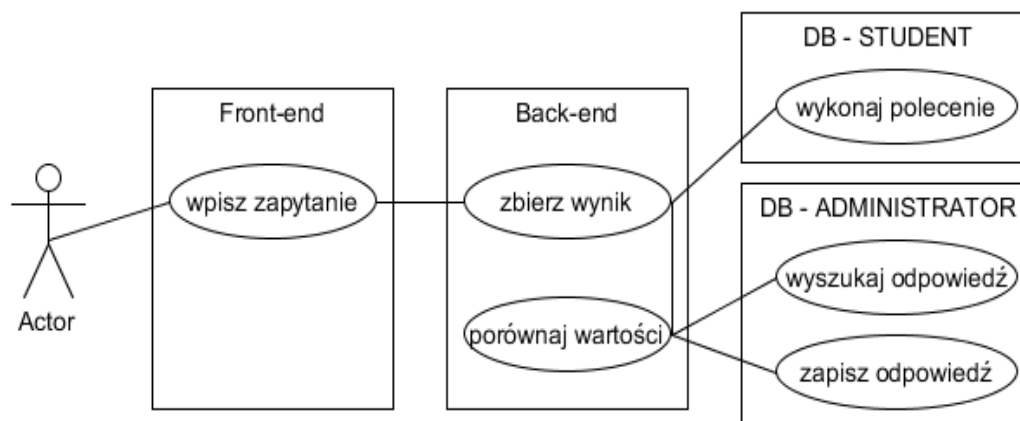
Na rysunku 8 przedstawiono sposób weryfikacji poprawności udzielonej odpowiedzi.



Rysunek 8: Diagram przypadków użycia – Weryfikacja odpowiedzi

4.1.3. Przyznanie punktów

Na rysunku 9 przedstawiono sposób przyznawania punktów za udzielenie poprawnej odpowiedzi.



Rysunek 9: Diagram przypadków użycia – Przyznanie punktu

4.2. Opis wybranych funkcjonalności

4.2.1. Praca grupowa

Dzięki technologii push przeglądanie stron internetowych nie polega już tylko na odpytywaniu serwera o pliki w formacie HTML. Teraz serwery mają możliwość wymuszenia zmiany widoku strony w przeglądarce. Jest to komunikacja dwustronna, gdzie każdy z uczestników połączenia ma możliwość przesyłania danych w dowolnym momencie. Technika tak jest bardzo pomocna podczas pracy zdalnej, gdzie użytkownicy spotkania chcą na żywo obserwować i reagować na wydarzenia mające miejsce w odległej lokalizacji. W INBJZ został zaimplementowany moduł do pracy zespołowej, którego zastosowanie można znaleźć w dwóch szczególnych przypadkach.

Jednym z tych przypadków może być możliwość uczęszczania na zajęcia laboratoryjne lub wykładowe z domu lub akademika. Każdy, kto ma w danej chwili otwartą stronę z zakładką Praca zespołowa w systemie INBJZ, może na bieżąco śledzić i brać czynny udział w zajęciach poprzez rozwiązywanie zadań. Wszyscy uczestnicy widzą wpisywane zapytania i rezultaty wydanych komend przez inne osoby. Działa to dokładnie tak jak chat.

Drugim pomysłem na użycie modułu pracy zespołowej może być praca podczas zajęć, gdzie studenci mogą po kolei wykonywać zapytania na swojej stacji roboczej bez wstawiania

od stanowiska, tak jakby byli wywoływani do tablicy w dawnych czasach. Każdy uczestnik zajęć będzie widział co wpisuje kolega, tak jakby pisał to właśnie na tablicy. Jak wiadomo pisanie na tablicy wiąże się z problemami, zwłaszcza dla osób, które mają słaby wzrok i nie potrafią jednocześnie robić notatek i słuchać. Jeśli polecenia są wykonywane na ekranie, można je bez problemu skopiować i zachować.

4.2.2. Praca indywidualna

W odróżnieniu od modułu do pracy zespołowej, istnieje standardowy moduł do pracy indywidualnej, który identyfikuje tożsamość poprzez wpisany numer albumu. Można oczywiście zadania wykonywać w trybie incognito, jeśli nie poda się swojego numeru. Jest on celowo opcjonalny. Należy jednak pamiętać, że bez podania numeru, nie ma możliwości składania reklamacji w przypadku braku punktów.

Moduł ten ma służyć do pracy samodzielnej i efekty pracy są logowane w bazie danych, dostępnej przez prowadzącego zajęcia.

4.2.3. Forum DISQUS

Na użytek studentów mających problem z rozwiązywaniem zadań umieściłem forum dyskusyjne. Każdy kto chce się czegoś dowiedzieć od osób, potrzebuje pomocy, ma uwagi lub sugestie może wpisać dowolny komentarz pod każdym zadaniem w INBJZ.

4.2.4. Komunikaty

Aby szybko dowiedzieć się o zbliżających się terminach egzaminów, zmianach sal, odwołanych spotkaniach, przesuniętych godzinach konsultacji wystarczy zajrzeć do zakładki komunikaty.

4.2.5. Materiały naukowe

Dla ułatwienia pracy z bazą danych umieściłem listę odsyłaczy do przydatnych stron związanych z tematyką baz danych. Są to fora oraz anglojęzyczne strony z przykładami i odpowiedziami na najczęstsze pytania.

4.2.6. HELP

Do dyspozycji studentów jest również komenda HELP, którą można wydać z każdego formularza na stronie z zadaniami. Po jej wpisaniu wyświetli się lista dostępnych funkcji i poleceń danej bazy danych.

4.3. Dokumentacja techniczna

W tym rozdziale opiszę w usługi dostępne dla osób chcących korzystać z wewnętrznego API (Application Programming Interface). Kod źródłowy kontrolerów znajduje się w załączniku numer 1.

4.3.1. TaskController

```
@PostMapping("/task/query")
@SendToUser("/queue/position-updates")
public InbhzResultSet executeQuery(Request message, MessageHeaders
messageHeaders) { }
```

Aby skorzystać ze metody opisanej powyżej należy subskrybować się do usługi w następujący sposób:

```
stompClient.subscribe('/user/queue/position-updates', queryCallback);
```

a następnie wywołać usługę poprzez:

```
stompClient.send("/app/task/query", headers, JSON.stringify({
    'query': query, 'taskId': taskId, 'mode':mode, 'studentId':studentId})
);
```

Jako queryCallback należy przekazać funkcję, która ma zostać wywołana na po zrealizowaniu żądania. Do obiektu typu Request należy przekazać nr zadania, polecenie SQL, natomiast drugi argument służy do autentykacji użytkownika. Należy tam przekazać nr albumu, login i ewentualnie hasło. Usługa jest dostępna dla pojedynczego użytkownika. System przyjmuje żądania i odsyła je w standardowy sposób.

```
@ResponseBody
@RequestMapping("/sessionId")
@SendToUser("/queue/sessionId")
public String sessionId() { }
```

Aby skorzystać ze metody opisanej powyżej należy subskrybować się do usługi w następujący sposób:

```
stompClient.subscribe('/user/queue/sessionId', queryCallback);
```

a następnie wywołać usługę poprzez:

```
stompClient.send("/app/sessionId", headers, JSON.stringify({
    'query': query, 'taskId': taskId, 'mode':mode, 'studentId':studentId})
);
```

Jako `queryCallBack` należy przekazać funkcję, która ma zostać wywołana na po zrealizowaniu żądania. Do obiektu typu `Request` należy przekazać nr zadania, polecenie SQL, natomiast drugi argument służy do autentykacji użytkownika. Należy tam przekazać nr albumu, login i ewentualnie hasło. Usługa jest dostępna dla pojedynczego użytkownika i system przyjmuje żądania i odsyła je w standardowy sposób. Usługa służy do pokazania identyfikatora sesji użytkownika przydzielonego przez serwer.

```
@PostMapping("/getTaskById")
@SendToUser("/queue/getTaskById")
public Task getTaskById(int id) throws Exception { }
```

Aby skorzystać ze metody opisanej powyżej należy subskrybować się do usługi w następujący sposób:

```
stompClient.subscribe('/user/queue/getTaskById', queryCallBack);
```

a następnie wywołać usługę poprzez:

```
stompClient.send("/app/getTaskById", headers, JSON.stringify({
    'query': query, 'taskId': taskId, 'mode': mode, 'studentId': studentId})
);
```

Jako `queryCallBack` należy przekazać funkcję, która ma zostać wywołana na po zrealizowaniu żądania. Do obiektu typu `Request` należy przekazać nr zadania, polecenie SQL, natomiast drugi argument służy do autentykacji użytkownika. Należy tam przekazać nr albumu, login i ewentualnie hasło. Usługa jest dostępna dla pojedynczego użytkownika i system przyjmuje żądania i odsyła je w standardowy sposób. Usługa służy do znalezienia odpowiedniego zadania w repozytorium zadań. Bedzie to bardzo przydatna funkcjonalność w przypadku rozbudowy modułu automatycznego dodawania pytań. Front-end i back-end będą musiały wymienić się serią informacji, aby wyświetlić treść zadania na stronie w sposób dynamiczny, a nie w sposób statyczny, tak jak jest to obecnie wykonane. W tej chwili treść zadań jest umieszczona w dokumentach HTML.

```
@PostMapping("/getTask")
@SendToUser("/queue/getTask")
public Task getTask(int chapter, int number) throws Exception { }
```

Aby skorzystać ze metody opisanej powyżej należy subskrybować się do usługi w następujący sposób:

```
stompClient.subscribe('/user/queue/getTask', queryCallBack);
```

a następnie wywołać usługę poprzez:

```
stompClient.send("/app/getTask", headers, JSON.stringify({
    'query': query, 'taskId': taskId, 'mode': mode, 'studentId': studentId})
);
```

Jako queryCallback należy przekazać funkcję, która ma zostać wywołana na po zrealizowaniu żądania. Do obiektu typu Request należy przekazać nr zadania, polecenie SQL, natomiast drugi argument służy do autentykacji użytkownika. Należy tam przekazać nr albumu, login i ewentualnie hasło. Usługa jest dostępna dla pojedynczego użytkownika i system przyjmuje żądania i odsyła je w standardowy sposób.

```
@PostMapping("/getAllTasks")
@SendToUser("/queue/getAllTasks")
public List<Task> getTasks() throws Exception { }
```

Aby skorzystać ze metody opisanej powyżej należy subskrybować się do usługi w następujący sposób:

```
stompClient.subscribe('/user/queue/getAllTasks', queryCallback);
```

a następnie wywołać usługę poprzez:

```
stompClient.send("/app/getAllTasks", headers, JSON.stringify({
    'query': query, 'taskId': taskId, 'mode': mode, 'studentId': studentId})
);
```

Jako queryCallback należy przekazać funkcję, która ma zostać wywołana na po zrealizowaniu żądania. Do obiektu typu Request należy przekazać nr zadania, polecenie SQL, natomiast drugi argument służy do autentykacji użytkownika. Należy tam przekazać nr albumu, login i ewentualnie hasło. Usługa jest dostępna dla pojedynczego użytkownika i system przyjmuje żądania i odsyła je w standardowy sposób. Ten serwis ma za zadanie zwracać listę wszystkich zadań bez ich treści. Jedynie metadane.

4.3.2. QueryController

```
@PostMapping("/query")
@SendToUser("/queue/position-updates")
public InjzResultSet executeQuery(Request message, MessageHeaders
messageHeaders) { }
```

Aby skorzystać ze metody opisanej powyżej należy subskrybować się do usługi w następujący sposób:

```
stompClient.subscribe('/user/queue/position-updates', queryCallback);
```

a następnie wywołać usługę poprzez:

```
stompClient.send("/app/query", headers, JSON.stringify({
    'query': query, 'taskId': taskId, 'mode': mode, 'studentId': studentId})
);
```

Jako queryCallBack należy przekazać funkcję, która ma zostać wywołana po zrealizowaniu zapytania. Do obiektu typu Request należy przekazać polecenie SQL, nr zadania, tryb pracy: real/protected oraz nr albumu. Natomiast drugi argument służy do autentykacji użytkownika. Należy tam przekazać nr albumu, login i ewentualnie hasło. Usługa jest dostępna dla pojedynczego użytkownika i system przyjmuje zapytania i odsyła je w standardowy sposób, dzięki przedrostkowi /user/. Jest to standardowa usługa używana w głównym module systemu do pracy indywidualnej. Usługa przyjmuje wszelkie rodzaje poleceń, które są następnie interpretowane. W zależności od rodzaju polecenia zbiór wyników jest zwracany lub nie. W odpowiedzi znajduje się odpowiednia informacja, mówiąca o typie odpowiedzi. Odpowiedź zawiera również takie informacje jak:

```
String info;
int taskId;
String status;
boolean correct;
String type;
String consoleOutput;
String errorMessage;
String[] expectedHeaders;
List<String[]> expected;
String[] actualHeaders;
List<String[]> actual;
List<Integer> missingRowIds;
List<Integer> extraRowIds;
List<String[]> missingRows;
List<String[]> extraRows;
```

```
@ResponseBody
@RequestMapping("/sessionId")
public String sessionId() { }
```

Opisana wyżej metoda służy do zwrócenia identyfikatora sesji protokołu HTTP. Może ona zostać wykorzystana przez klienta do stworzenia pośrednika pomiędzy front-endem a back-endem.

```
@RequestMapping("/teamHello")
@SendTo("/topic/greetings")
public InbhzResultSet greeting(Request message) throws Exception { }
```


Aby skorzystać ze metody opisanej powyżej należy subskrybować się do usługi w następujący sposób:

```
stompClient.subscribe('/topic/greetings', greetingsCallBack);
```

a następnie wywołać metodę:

```
stompClient.send("/app/teamHello", {}, JSON.stringify({  
    'message': message})  
);
```

Jako queryCallBack należy przekazać funkcję, która ma zostać wywołana na po zrealizowaniu żądania. Do obiektu typu Request należy przekazać tekst przywitania, natomiast drugi argument służy do autentykacji użytkownika. Należy tam przekazać nr albumu, login i ewentualnie hasło. Usługa jest dostępna dla wielu użytkowników i system przyjmuje żądania, następnie odsyła je do wszystkich subskrybowanych osób. Przysyłany komunikat tekstowy znajduje się w polu info zwracanego obiektu.

```
@MessageMapping("/teamExecute")  
@SendTo("/topic/execute")  
public InbjzResultSet execute(Request message) throws Exception {    }
```

Aby skorzystać ze metody opisanej powyżej należy subskrybować się do usługi w następujący sposób:

```
stompClient.subscribe('/topic/execute', executeCallBack);
```

a następnie wywołać metodę:

```
stompClient.send("/app/teamExecute", {}, JSON.stringify({  
    'query': query, 'taskId': taskId, 'mode': mode})  
);
```

Jako executeCallBack należy przekazać funkcję, która ma zostać wywołana na po zrealizowaniu żądania. Do obiektu typu Request należy przekazać nr zadania, polecenie SQL, tryb real/protected natomiast drugi argument służy do autentykacji użytkownika. Należy tam przekazać nr albumu, login i ewentualnie hasło. Usługa jest dostępna dla wielu użytkowników na raz. System przyjmuje żądania i odsyła je do wszystkich subskrybowanych osób. Jest to szczególnie przydatne podczas pracy grupowej. W odróżnieniu od metody `/teamQuery`, metoda `/teamExecute` nie ma tak dużego narzutu w przypadku wykonywania poleceń SQL nie zwracających zbioru wynikowego. Jeśli chcemy jedynie wykonać INSERT lub UPDATE, należy wywołać właśnie tę usługę.

```
@MessageMapping("/teamQuery")  
@SendTo("/topic/query")  
public InbjzResultSet select(Request message) throws Exception {    }
```

Aby skorzystać ze metody opisanej powyżej należy subskrybować się do usługi w następujący sposób:

```
stompClient.subscribe('/topic/query', queryCallBack);
```

a następnie wywołać metodę:

```
stompClient.send("/app/teamQuery", {}, JSON.stringify({  
    'query': query, 'taskId': taskId, 'mode': mode})  
});
```

Jako queryCallBack należy przekazać funkcję, która ma zostać wywołana na po zrealizowaniu żądania. Do obiektu typu Request należy przekazać nr zadania, polecenie SQL, tryb real/protected natomiast drugi argument służy do autentykacji użytkownika. Należy tam przekazać nr albumu, login i ewentualnie hasło. Usługa jest dostępna dla wielu użytkowników na raz. System przyjmuje żądania i odsyła je do wszystkich subskrybowanych osób. Jest to szczególnie przydatne podczas pracy grupowej. W przeciwieństwie do usługi `/teamExecute` usługa `/teamQuery` może być służyć dla wszystkich rodzajów poleceń SQL. Program sprawdzi jaki jest rodzaj polecenia i zwróci zbiór wynikowy bądź nie. Oczywiście odbywa się to większym kosztem. Usługa `/teamQuery` wykonuje dokładnie taką samą operację jak `/query`.

Rozdział 5

5.1. Przykłady działania aplikacji oraz dokumentacja użytkownika

5.1.1. Uruchomienie aplikacji

Aplikację można uruchomić na wiele sposobów. W poniższym przewodniku zaprezentuję dwie z nich:

Instalacja na maszynie lokalnej

Aby uruchomić aplikację należy najpierw pobrać plik `interactive-sql-learning-0.1.1.jar`, który jest dostępny pod adresem:

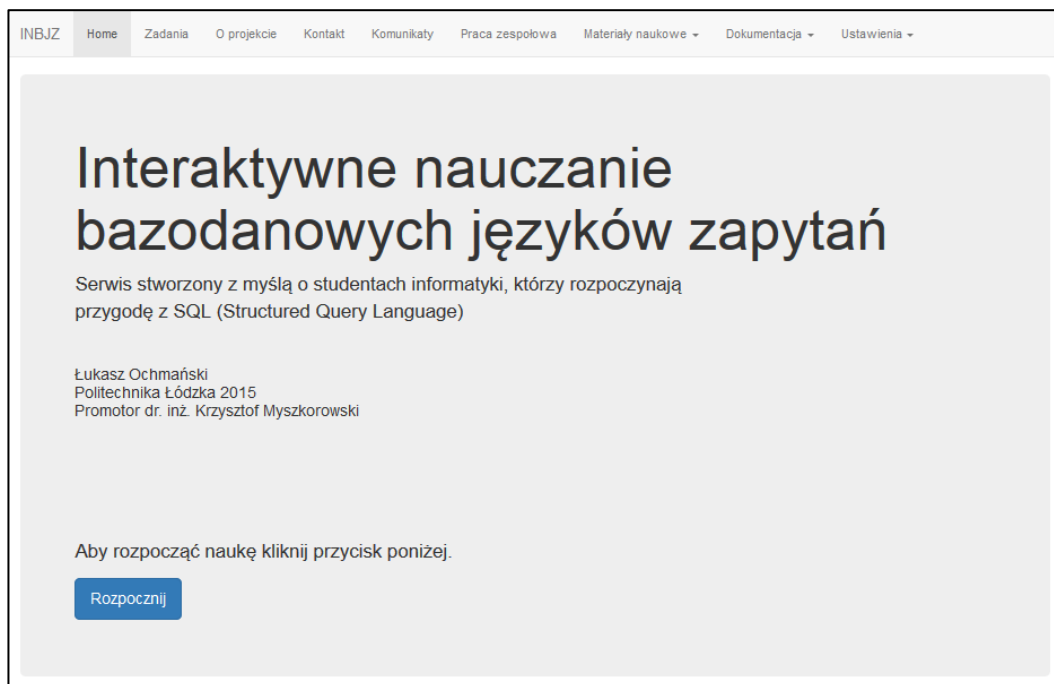
https://code.google.com/p/inbjz/source/browse/stable_release

Następnie należy uruchomić plik. Jeśli używasz systemu operacyjnego Windows wystarczy dwukrotnie kliknąć myszką na ikonkę i poczekać około trzydzieści sekund. Zakładamy, że posiadasz zainstalowaną wirtualną maszynę Javy wersji 8 (Java Runtime Environment 8). Jeśli nie posiadasz najnowszej wersji nie uda ci się uruchomić programu. W przypadku problemów należy użyć wiersza poleceń (cmd) systemu windows i wykonać polecenie `java -jar interactive-sql-learning-0.1.1.jar`.

Po pomyślnym uruchomieniu serwera należy otworzyć dowolną przeglądarkę internetową i wpisać <http://localhost>. Zakładamy, że port 80 jest dostępny. Po zakończeniu używania aplikacji należy zlokalizować proces w menedżerze zadań i go zakończyć. Proces zostanie również zakończony przy zamknięciu systemu. W przypadku korzystania z wiersza poleceń lub konsoli w systemie Linux należy użyć kombinacji klawiszy `Ctrl+c`, co powoduje zakończenie procesu w obu przypadkach.

Wersja WWW

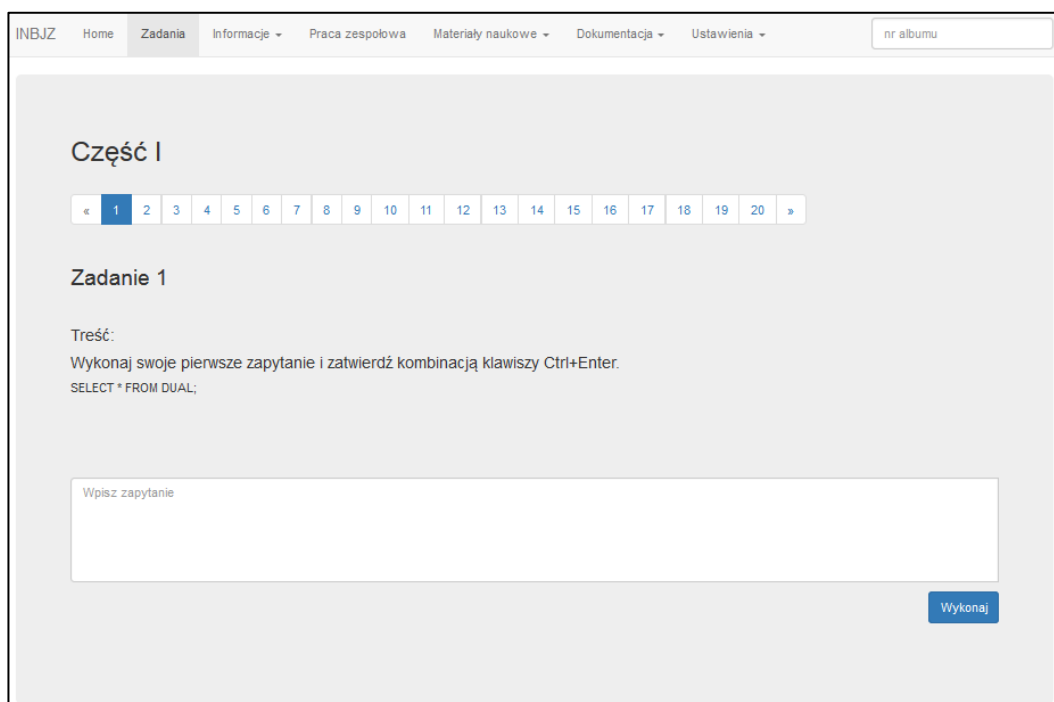
Wszystkie opisane powyżej kroki są zbędne jeśli posiadasz szybki dostęp do internetu. Używanie dostępnej aplikacji w internecie jest zalecaną metodą. W pełni działająca aplikacja jest dostępna pod adresem: <http://37.187.43.124>. Po pomyślnym wczytaniu strony na ekranie powinien pojawić się widok jak na załączonym rysunku 10.



Rysunek 10: Strona główna aplikacji.

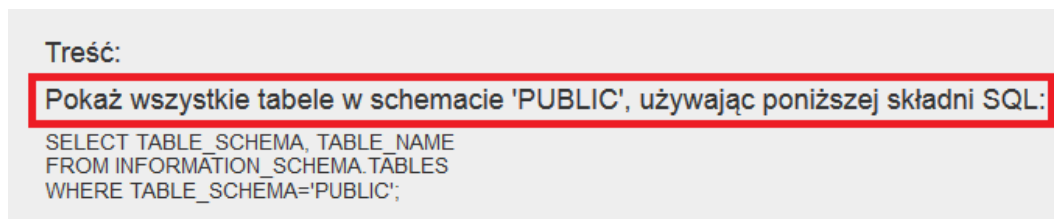
5.1.2. Rozwiązywanie zadań

Aby rozpocząć rozwiązywanie zadań należy kliknąć na przycisk „Rozpocznij” na stronie głównej. Zostaniemy automatycznie przeniesieni do pierwszego zadania (rysunek 11).



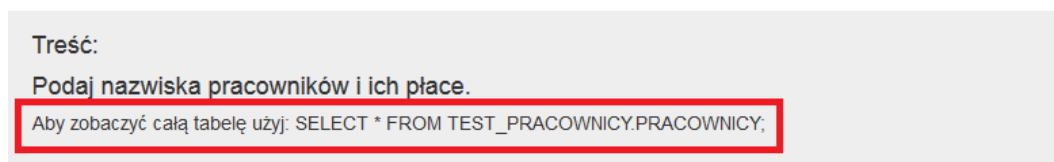
Rysunek 11: Przykładowe zadanie.

W formularzu w dolnej części strony należy wpisać polecenie SQL, o które poproszono w treści zadania. Treść zadania jest przedstawiona pogrubionym drukiem na rysunku 12:



Rysunek 12: Treść zadania.

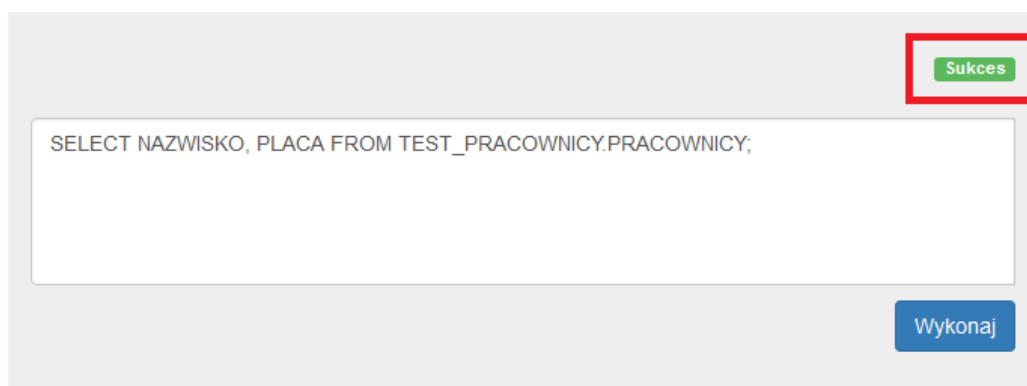
Poniżej treści często widoczna jest podpowiedź lub uwaga, której przeczytanie nie jest wymagane. Niemniej jednak jej znajomość jest wymagana do udzielenia poprawnej odpowiedzi. Przykład na rysunku 13.



Rysunek 13: Podpowiedzi.

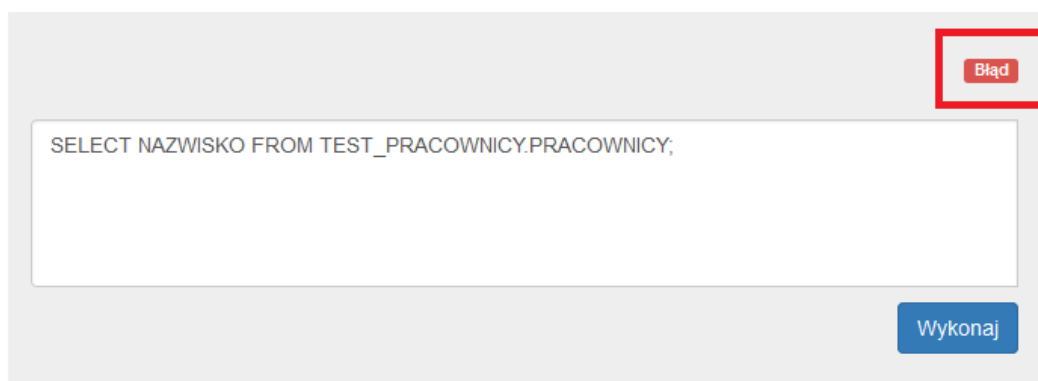
Teoretycznie w zaprezentowanym powyżej przykładzie nie ma potrzeby stosowania podpowiedzi, gdyż użytkownik w pierwszym etapie kursu dowiaduje się o sposobach odnajdowania nazw schematów, tabel i kolumn. Nie chcemy jednak, aby spędził nad rozwiązaniem tego zadania zbyt dużo czasu i dajemy mu małą wskazówkę w postaci przykładowego zapytania.

Po wpisaniu odpowiedzi zgodnej z kluczem system informuje o tym użytkownika poprzez wyświetlenie zielonej ikonki po prawej stronie ekranu, widocznej na rysunku 14:



Rysunek 14: Poprawna odpowiedź.

W przypadku udzielenia niepoprawnej odpowiedzi pojawia się czerwona ikonka informująca o popełnionym błędzie (rysunek 15).



Rysunek 15: Niepoprawna odpowiedź.

Każdy użytkownik systemu ma nieograniczone możliwości odpytywania bazy danych używając dozwolonych kwerend. Wyniki zapytań są natychmiast wyświetlane na stronie w postaci tabeli HTML. Za niewłaściwe odpytywanie bazy danych nie ma żadnych kar ani negatywnych konsekwencji. W systemie przyjęto strategię uwzględniania jedynie poprawnych odpowiedzi. Powodem takiej strategii jest zachęcenie użytkowników do jak najczęstszego ćwiczenia na danych w dowolny wybrany przez siebie sposób. Przykładowy wynik zapytania został przedstawiony na rysunku 16.

INBJZ	Home	Zadania	Informacje ▾	Praca zespołowa	Materiały naukowe ▾	Dokumentacja ▾	Ustawienia ▾	nr albumu
-------	------	---------	--------------	-----------------	---------------------	----------------	--------------	-----------

Wynik zapytania:

```
select * from test_pracownicy.pracownicy;
```

NR_AKT	NAZWIŚKO	STANOWISKO	KIEROWNIK	DATA_ZATR	DATA_ZIWOL	PLACA	DOD_FUNKCYJNY	PROWIZJA	ID_DZIAŁU
8901	KROL	PREZES	null	1989-07-01 00:00:00.0	null	7000.00	4000.00	null	10
8902	MICHALSKI	DYREKTOR	8901	1989-08-15 00:00:00.0	null	5000.00	1500.00	null	40
8904	SKALSKI	GŁÓWNY INFORMATYK	8901	1989-08-18 00:00:00.0	null	3500.00	2500.00	null	10
8910	MONIUSZKO	DYREKTOR	8901	1989-09-01 00:00:00.0	null	5000.00	1500.00	null	20
8911	WRZOSEK	OPERATOR	8910	1989-11-10 00:00:00.0	null	1500.00	null	null	20
8913	KOWALSKA	CZŁONEK ZARZĄDU	8901	1989-11-15 00:00:00.0	null	5000.00	2000.00	null	10
8920	WOJCIK	TECHNOLOG	8910	1989-12-01 00:00:00.0	null	1800.00	null	null	20
8932	BRZOSKA	GŁÓWNY KSIĘWY	8901	1989-12-08 00:00:00.0	null	3000.00	2000.00	null	60
9010	WISNIEWSKA	GRAFIK	8902	1990-02-12 00:00:00.0	null	1800.00	null	null	40
9011	WIERZBICKI	INFORMATYK	8902	1990-03-20 00:00:00.0	null	2000.00	null	null	40
9025	MALIK	LOGISTYK	9121	1990-06-01 00:00:00.0	null	1500.00	null	null	30
9027	GADULA	LOGISTYK	9121	1990-06-20 00:00:00.0	null	1500.00	null	null	30

Rysunek 16: Wynik zapytania.

Gdy użytkownik zapozna się już ze strukturą bazy i jest gotowy przystąpić do rozwiązywania zadania, może skorzystać z przygotowanego klucza odpowiedzi. Użytkownik nie ma możliwości podejrzenia zapytania SQL, ale ma możliwość porównania swojego wyniku z oczekiwanym rezultatem (rysunek 17). Autor treści zadań ma obowiązek dostarczyć poprawne odpowiedzi. Wówczas komputer w binarny sposób porównuje zawartość obu tabel wynikowych, komórka po komórce. Ilość prób, jaką użytkownik może wykorzystać do uzyskania oczekiwanego rezultatu jest nieograniczona.

INBJZ	Home	Zadania	Informacje ▾	Praca zespołowa	Materiały naukowe ▾	Dokumentacja ▾	Ustawienia ▾	nr albumu
-------	------	---------	--------------	-----------------	---------------------	----------------	--------------	-----------

Oczekiwany rezultat:

* zapytanie ukryte	
NAZWISKO	PLACA
KROL	7000.00
MICHALSKI	5000.00
SKALSKI	3500.00
MONIUSZKO	5000.00
WRZOSEK	1500.00
KOWALSKA	5000.00
WOJCIK	1800.00
BRZOZKA	3000.00
WISNIEWSKA	1800.00
WIERZBICKI	2000.00
MALIK	1500.00
GADULA	1500.00
LESZCZYNSKI	1200.00
KOWALCZYK	1200.00
RYBAK	1800.00

Rysunek 17: Oczekiwany rezultat.

5.1.3. Nawigacja

Użytkownik systemu może poruszać się po wszystkich dostępnych ćwiczeniach bez żadnych ograniczeń. Zalecane jest jednak, aby zachować kolejność rozwiązywania przygotowanych zadań, gdyż ułatwi to pracę. Zadania zostały ułożone rosnąco pod względem trudności. Aby podejrzeć listę zadań można skorzystać z paska menu w górnej części strony widocznej na rysunku 18:

INBJZ	Home	Zadania	O projekcie	Kontakt	Komunikaty	Praca zespołowa	Materiały naukowe ▾	Dokumentacja ▾	Ustawienia ▾
-------	------	---------	-------------	---------	------------	-----------------	---------------------	----------------	--------------

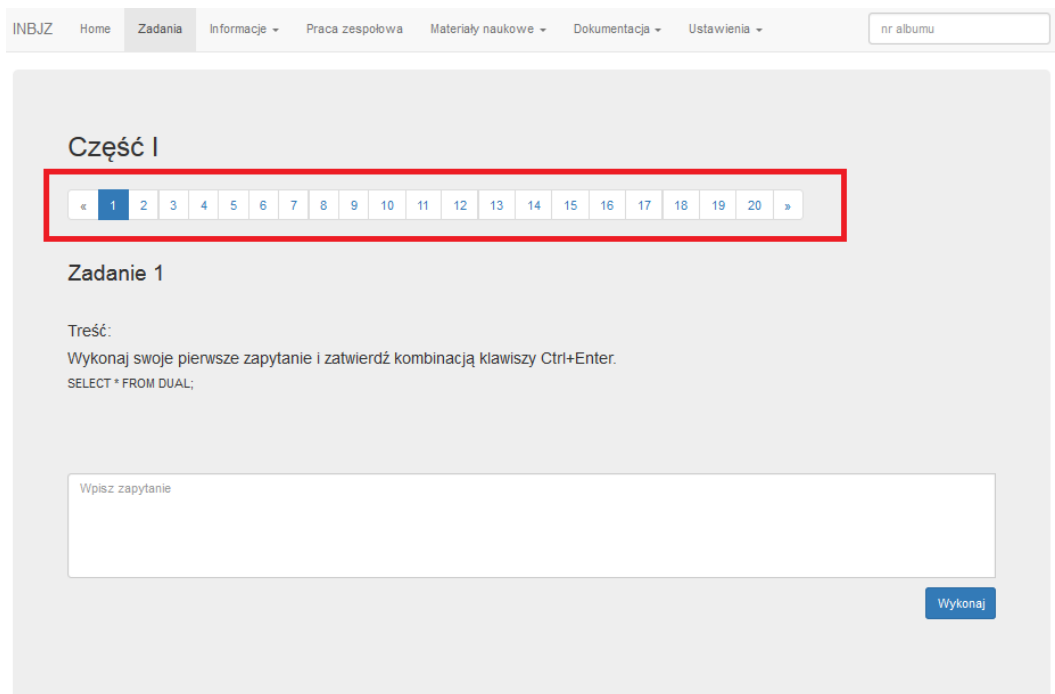
Lista zadań

Część I

- [Zadanie 1](#)
- [Zadanie 2](#)
- [Zadanie 3](#)
- [Zadanie 4](#)
- [Zadanie 5](#)
- [Zadanie 6](#)
- [Zadanie 7](#)
- [Zadanie 8](#)
- [Zadanie 9](#)
- [Zadanie 10](#)
- [Zadanie 11](#)
- [Zadanie 12](#)
- [Zadanie 13](#)
- [Zadanie 14](#)
- [Zadanie 15](#)
- [Zadanie 16](#)
- [Zadanie 17](#)
- [Zadanie 18](#)
- [Zadanie 19](#)
- [Zadanie 20](#)

Rysunek 18: Strona z zadaniami.

Dodatkowo istnieje druga możliwość przełączania się pomiędzy zadaniami. W obrębie etapu nie trzeba wracać do zakładki „Zadania”. Gdy znajdzie potrzeba przeniesienia się na inną stronę użytkownik może skorzystać z paska nawigacji na każdej stronie z zadaniem. Wystarczy kliknąć numer który nas interesuje, tak jak zaprezentowano na rysunku 19:



Rysunek 19: Pasek nawigacji.

Aby przejść do kolejnego etapu, należy powrócić do menu „Zadania” i wybrać dowolne zadanie z pożądanego etapu. Następnie można kontynuować korzystanie z paska nawigacji, aby poruszać się pomiędzy ćwiczeniami danej części kursu.

5.1.4. Obsługa błędów

W przypadku wpisania niepoprawnej składniowo komendy SQL, system wyświetla komunikat zwrócony przez procesor zapytań w postaci okienka pod napisem: „Console output” (rysunek 20).



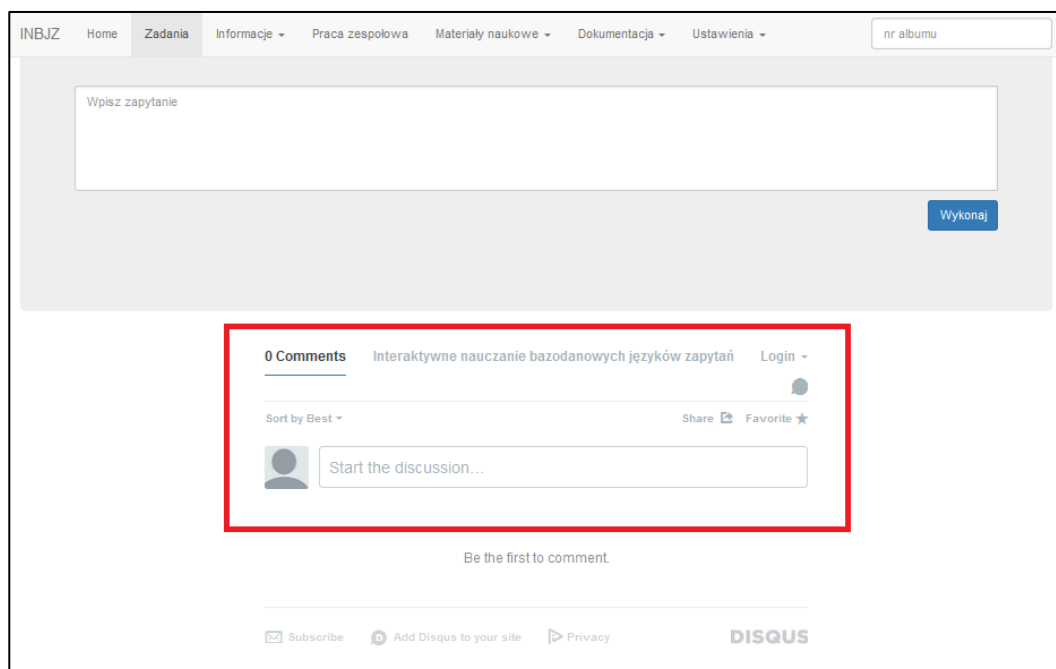
The screenshot shows a web-based SQL query interface. At the top, it says "Treść:" followed by the instruction "Podaj nazwiska pracowników i ich płace." and a hint "Aby zobaczyć całą tabelę użyj: SELECT * FROM TEST_PRACOWNICY.PRACOWNICY;". Below this is a text input field containing the query "select * from test_pracownicy.pracownik". To the right of the input field is a red button labeled "Błąd". Below the input field is a blue button labeled "Wykonaj". At the bottom, there is a "Console output:" section with a red border, containing the error message: "Table 'PRACOWNIC' not found; SQL statement: select * from test_pracownicy.pracownik [42102-182]".

Rysunek 20: Komunikat błędu.

5.1.5. Forum dyskusyjne

Jednym z najciekawszych rozwiązań zaprezentowanych w systemie jest użycie forum dyskusyjnego. Niestety forum nie jest zupełnie anonimowe i wymaga zarejestrowania się w systemie, ale mam nadzieję, że nie przeszkodzi to w prowadzeniu rozmaitych dyskusji przez studentów na tematy związane przedmiotem. Ma to swoje zalety, gdyż pomoże to w identyfikowaniu nieuczciwych bądź wulgarnych osób.

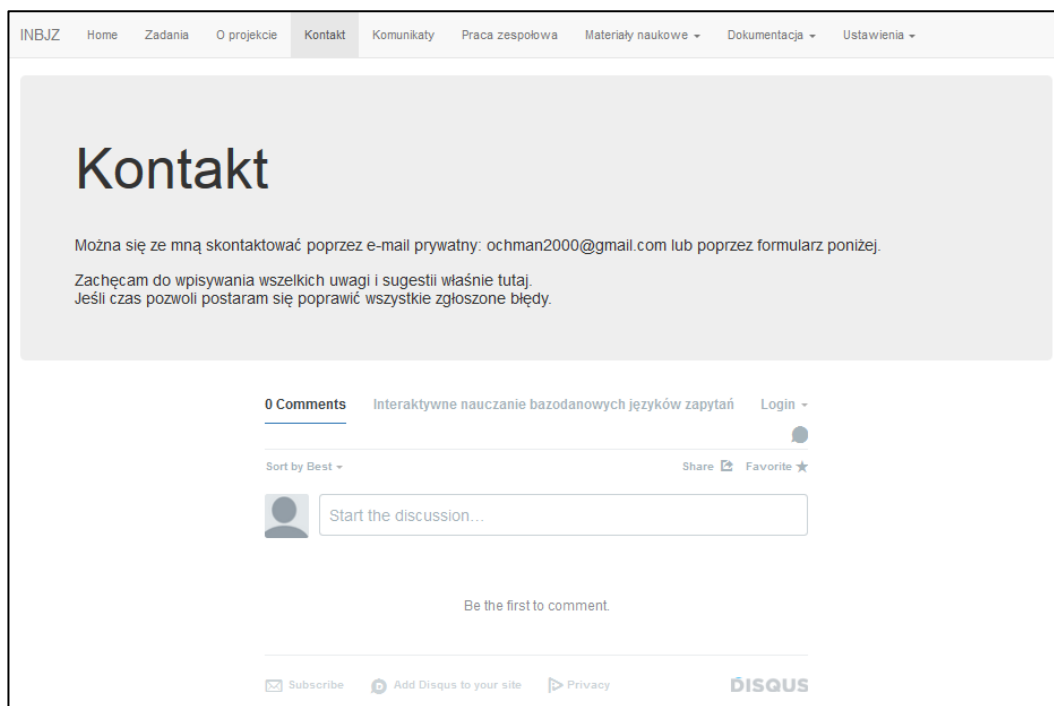
Dzielenie się wiedzą uważam za podstawę kształcenia. Studenci będą mogli prowadzić konwersacje na żywo w trakcie rozwiązywania pytań. W przypadku wątpliwości co do któregoś kolwiek zadania można w bardzo łatwy sposób umieścić wiadomość w polu tekstowym poniżej każdego zadania (rysunek 21) i oczekiwać wsparcia innych.



Rysunek 21: Forum dyskusyjne.

5.1.6. Kontakt

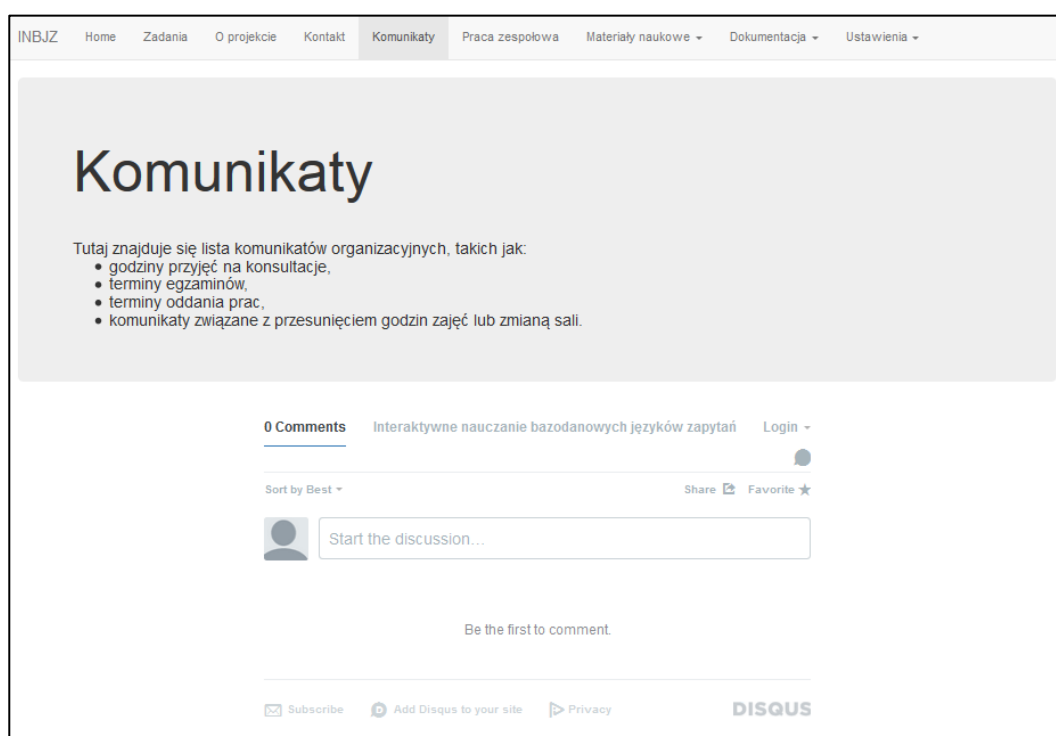
W przypadku jakichkolwiek problemów, uwag lub sugestii można skontaktować się z twórcą strony poprzez dane umieszczone w zakładce kontakt. Jest ona dostępna w pasku menu w górnej części strony. Wygląd zakładki kontakt widoczny na rysunku 22.



Rysunek 22: Strona kontaktowa.

5.1.7. Komunikaty

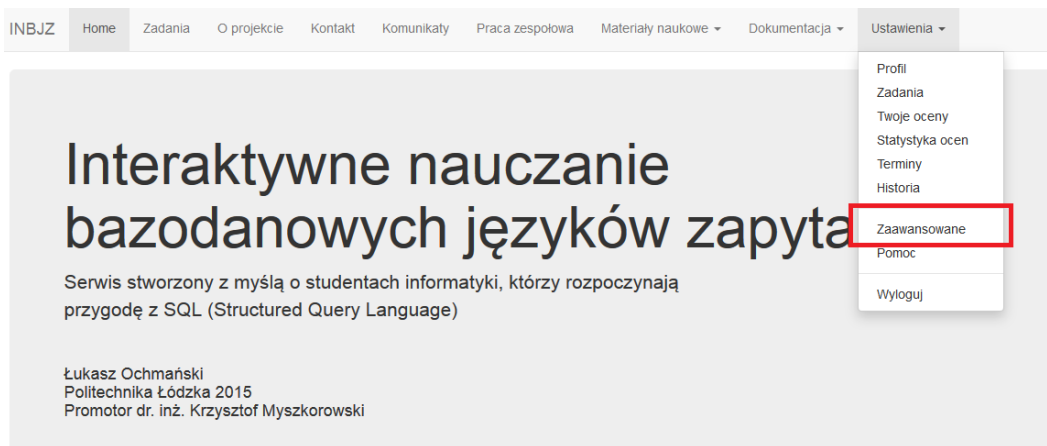
W celu ułatwienia dostępu do informacji odnośnie prowadzonych zajęć można umieszczać komunikaty organizacyjne bezpośrednio na stronie (rysunek 23). Nie ma potrzeby tworzenia osobnych stron, na potrzeby prowadzonych zajęć, co jest częstą praktyką wśród wykładowców. Mogą się tutaj również znaleźć linki do tych stron lub bezpośrednio informacje, które są związane z terminami zjazdów terminami zaliczeń i tym podobne. Jest to również alternatywa dla powstałych serwisów informacyjnych, które często są niedostępne z powodu prac konserwacyjnych takich jak Wikamp.



Rysunek 23: Komunikaty organizacyjne.

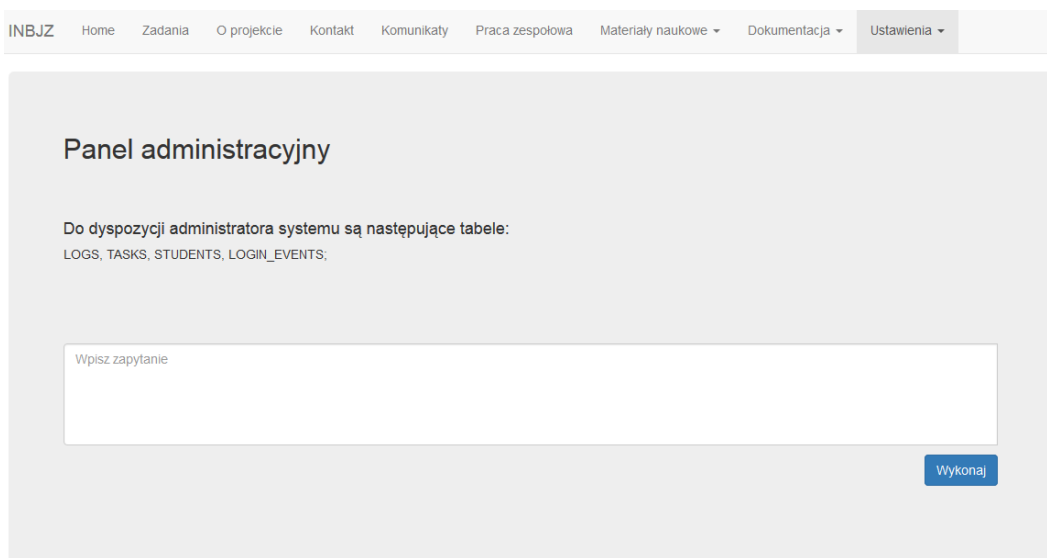
5.1.8. Panel sterowania

Kluczowym modulem systemu INBJZ jest panel sterowania. Aby się do niego dostać należy wybrać menu „Ustawienia”, następnie „Zaawansowane” tak, jak pokazano na rysunku 24.



Rysunek 24: Menu Zaawansowane.

Po wybraniu powyższej opcji użytkownik powinien zobaczyć poniższy ekran. W wersji podstawowej systemu nie ma przygotowanych gotowych widoków dla prowadzących zajęcia, jednak system zapewnia pełny dostęp do usług poprzez zaprezentowany interfejs na rysunku 25.



Rysunek 25: Panel Administracyjny.

Dodatkowo w ten sposób nauczyciel ma większe możliwości wyciągania istotnych informacji. Na przykład, aby sprawdzić odpowiedzi wszystkich uczniów można wykonać zapytanie:

```
SELECT STUDENT_ID, ANSWER, CORRECT, LOG_DATE
FROM LOGS L
JOIN LOGGED_ANSWERS A ON L.ANSWER_ID=A.ID;
```

Wynik zapytania widoczny na rysunku 26.

Wynik zapytania:

select student_id, answer, correct, log_date from logs l join logged_answers a on l.answer_id=a.id;			
STUDENT_ID	ANSWER	CORRECT	LOG_DATE
183566	select * from test_pracownicy.pracownicy;	FALSE	2015-02-22 17:18:46.648
183566	SELECT * FROM TEST_PRACOWNICYPRACOWNICY;	FALSE	2015-02-22 17:23:59.409
183566	SELECT NAZWISKO, PLACA FROM TEST_PRACOWNICYPRACOWNICY;	TRUE	2015-02-22 17:24:18.793
183566	SELECT NAZWISKO FROM TEST_PRACOWNICYPRACOWNICY;	FALSE	2015-02-22 17:28:48.973

Rysunek 26: Udzielone odpowiedzi.

Następnie odpowiedzi można pogrupować i sprawdzić ile rekordów zawiera poprawną odpowiedź, a ile rekordów niepoprawną:

```
SELECT STUDENT_ID, CORRECT, COUNT(*)
FROM LOGS L
JOIN LOGGED_ANSWERS A ON L.ANSWER_ID=A.ID
GROUP BY STUDENT_ID, CORRECT;
```

Wynik zapytania widoczny na rysunku 27.

select student_id, correct, count(*) from logs l join logged_answers a on l.answer_id=a.id group by student_id, correct;		
STUDENT_ID	CORRECT	COUNT(*)
183566	TRUE	1
183566	FALSE	3

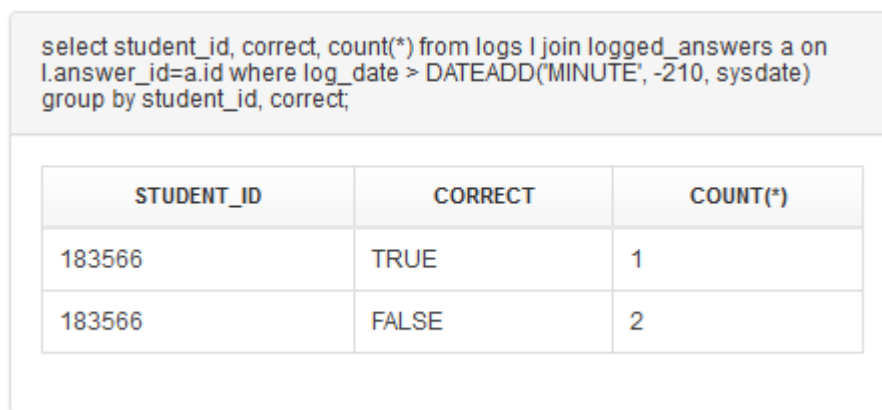
Rysunek 27: Pogrupowane odpowiedzi każdego z uczniów.

Pogrupowane odpowiedzi można posortować po liczbie poprawnych odpowiedzi i na podstawie sporządzonego rankingu wystawić oceny końcowe.

W przypadku gdy chcemy sprawdzić ile z tych odpowiedzi zostało udzielonych podczas trwającego właśnie kolokwium można nieinwazyjnie to sprawdzić wykonując podobne zapytanie z ograniczeniem czasu do ostatnich 120 minut:

```
SELECT STUDENT_ID, CORRECT, COUNT(*)  
FROM LOGS L  
JOIN LOGGED_ANSWERS A ON L.ANSWER_ID=A.ID  
WHERE LOG_DATE > DATEADD('MINUTE', -120, SYSDATE)  
GROUP BY STUDENT_ID, CORRECT;
```

Wynik zapytania widoczny na rysunku 28.



The screenshot shows a SQL query and its results. The query is: `select student_id, correct, count(*) from logs l join logged_answers a on l.answer_id=a.id where log_date > DATEADD('MINUTE', -210, sysdate) group by student_id, correct;` The results are displayed in a table with three columns: STUDENT_ID, CORRECT, and COUNT(*). There are two rows of data.

STUDENT_ID	CORRECT	COUNT(*)
183566	TRUE	1
183566	FALSE	2

Rysunek 28: Wyniki studentów z trwającego kolokwium.

Używając panelu sterowania można wykazać się nieograniczoną wyobraźnią i przeprowadzać różnego rodzaju badania. Można porównać odpowiedzi uczniów i wyszukać dwóch studentów, którzy mają najbardziej podobne odpowiedzi, na podstawie analizy ciągu znaków. Wtedy wiadomo, że od siebie ściągali.

Można też w bardzo prosty sposób odpytać bazę o studentów, którzy wykonali zadania najszybciej. Jeśli komuś udało się rozwiązać 20 zadań w mniej niż 5 minut to jest duże prawdopodobieństwo że nie będzie potrafił się z tego wytłumaczyć i nie należy takiej osobie przyznawać oceny.

```
SELECT STUDENT_ID,  
DATEDIFF('SECOND', MIN(LOG_DATE), MAX(LOG_DATE)) AS SECONDS  
FROM LOGS L  
GROUP BY L.STUDENT_ID  
ORDER BY SECONDS ASC
```

Wynik zapytania widoczny na rysunku 29.


```
select student_id, DATEDIFF('SECOND', min(log_date), max(log_date)) as  
seconds from logs l group by l.student_id order by seconds
```

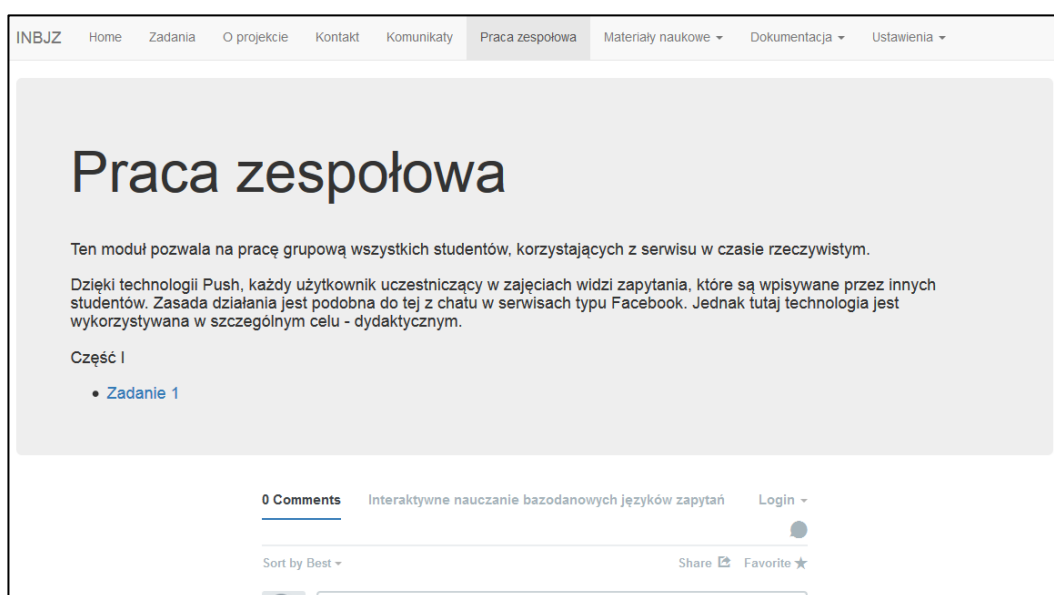
STUDENT_ID	SECONDS
123457	9
123458	14
123456	97
183566	602

Rysunek 29: Ranking studentów rozwiązujących zadania najszybciej.

Pomimo, że koszt napisania takiego zapytania jest stosunkowo duży dla niewprawionej osoby, to jednak jest to koszt jednorazowy. Wprowadzone zapytania można sobie zapisać i wielokrotnie używać. Poza tym zapewnia to ogromną elastyczność. Bardzo trudno byłoby napisać interfejs graficzny, który potrafiłby pokazać potrzebne raporty w każdej sytuacji. Zakładam, że prowadzący zajęcia umie się posługiwać SQL'em, choć nie wykluczam dodania najpotrzebniejszych wykazów w postaci widoków czy wykresów SVG w niedalekiej przyszłości.

5.2. Praca zespołowa

W systemie dostępny jest moduł pracy grupowej, który może posłużyć do rozwiązywania zadań w trakcie zajęć laboratoryjnych lub zdalnie. Czasem występuje sytuacja, w której rzutnik jest niedostępny, a prowadzący ma potrzebę zaprezentowania swojego zapytania uczniom. Innym razem zdarza się że uczniowie są wywoływani do tablicy, ale w przypadku ćwiczeń z podstaw baz danych pisanie na tablicy kredą nie ma sensu. W takich sytuacjach przydaje się moduł pracy grupowej. W ten sposób wywoływani uczniowie mogą wspólnie z grupą rozwiązywać problemy tak jak za dawnych czasów, ale teraz nie muszą odchodzić od swojego stanowiska. Aby skorzystać z tej funkcjonalności należy wybrać zakładkę „Praca zespołowa”. Rysunek 30 przedstawia moduł:



Rysunek 30: Moduł pracy zespołowej.

Rozdział 6

6.1. Uzasadnienie wyboru pytań

Pytania zostały podzielone na sześć grup. W części pierwszej znajdują się pytania obejmujące podstawowe zagadnienia związane z obsługą baz danych. Ten rozdział został podzielony na cztery etapy. W etapie pierwszym uczeń poznaje podstawowe polecenia SQL, dowiaduje się jak poruszać się po schematach i jak znajdować tabele w nich umieszczone. Etap drugi ma na celu zaprezentowanie podstawowych poleceń DML (Data Modification Language) takich jak CREATE, ALTER, DROP. Student powinien opanować tworzenie tabel, definiowanie kolumn o różnych typach oraz powinien nauczyć się je modyfikować oraz usuwać. Etap trzeci ma na celu wprowadzenie podstawowych poleceń DDL (Data Definition Language) takich jak INSERT, UPDATE, DELETE. Użytkownik powinien opanować podstawowe techniki edycji danych. W etapie czwartym zaprezenowane są nieco bardziej złożone polecenia DML pomocne w definicji struktur danych i ograniczeń. Przykładem takich poleceń są: ALTER TABLE ADD CONSTRAINT NOT NULL, ALTER TABLE ADD CONSTRAINT PRIMARY KEY.

Część druga samouczka została podzielona na trzy etapy. Tutaj zadania również skupiają się na definicji struktury danych oraz ich manipulacją w zaawansowany sposób. Pierwszy etap ma na celu pokazanie użycia ograniczeń typu CHECK, PRIMARY KEY oraz FOREIGN KEY w trakcie tworzenia tabel. Drugi etap natomiast skupia się na definiowaniu podobnych ograniczeń na etapie dalszej pracy z bazą danych. Bardzo często zachodzi potrzeba wprowadzenia zmian istniejących ograniczeń w późniejszym momencie niż podczas modelowania struktury. Pojawiają się czasem problemy z poprawnością danych, które należy rozwiązać. W ostatnim etapie tego rozdziału wymagane jest zastosowanie zaawansowanych technik utrzymywania spójności danych. Tworzenie relacji poprzez definiowanie kluczy obcych i kluczy głównych. Zakładanie indeksów unikatowych i nieunikatowych, jednokolumnowych i wielokolumnowych.

Część trzecia to wprowadzenie do języka zapytań DQL (Data Query Language). Ta część kursu została podzielona na cztery etapy. W pierwszym etapie zaprezentowane są najprostsze zapytania SQL selekcjonujące jedynie najpotrzebniejsze informacje. Drugi etap składa się z zadań zawierających funkcje matematyczne typu COUNT(), MIN(), MAX() oraz funkcje operujące na ciągach znaków SUBSTR(), LENGHT(). W kolejnym etapie

wykorzystywane są poznane wcześniej funkcje do agregowania danych za pomocą klauzuli GROUP BY. Na tym poziomie po raz pierwszy pojawia się konieczność tworzenia podzapytań. Dodatkowo należy rozważyć zastosowanie klauzuli HAVING, choć przy odpowiednich zdolnościach w pisaniu podzapytań i złączeń można tego oczywiście uniknąć. System nie weryfikuje sposobu rozwiązywania problemów w żaden konkretny sposób. Oceniana jest jedynie poprawność odpowiedzi, a zatem jej skuteczność. W ostatnim etapie pytania pojawiają się trudniejsze zadania wymagające zastanowienia się nad sposobem otrzymania poprawnej odpowiedzi. Wymagane jest zastosowanie złączeń oraz wykluczeń zbiorów. Może zająć konieczność użycia klauzuli IN, EXISTS lub JOIN. Pytania są bardzo krótkie i konkretne, np. Podaj nazwiska szefów. Aby tego dokonać należy wykazać się sprytem. Dodatkowo na tym poziomie pojawia się konieczność zasięgnięcia informacji z innych źródeł, literatury lub internetu. Jest wielce nieprawdopodobne, że uda się rozwiązać tego typu problemy bez pozyskania dodatkowej wiedzy.

Część czwarta skupia się mocno na operacjach złączeń zbiorów. Pierwszy etap składa się wyłącznie z zapytań wymagających prostych złączeń typu INNER JOIN. Występują tu złączenia proste, które można obsłużyć za pomocą klauzuli NATURAL JOIN jak i złączenia kolumn o różnych nazwach. W zadaniu czwartym występuje konieczność złączenia co najmniej trzech zbiorów w jednym zapytaniu. W kolejnym ćwiczeniu student jest proszony o wykonanie złączenia wewnątrz zbioru typu SELF JOIN i tym ćwiczeniem zostanie zakończony pierwszy etap. W kolejnym etapie nastąpi dogłębna analiza złączeń typu LEFT JOIN. Zadania będą bardzo przypominały treścią zadania z pierwszego etapu, jednak będą się różnić warunkami brzegowymi. System INBJZ zadba o to, aby każdy uczeń zrozumiał istotę problemu. Dane są tak przygotowane, aby oba przypadki złączeń zwracały inny wynik. Tylko poprawne złączenie zewnętrzne zbiorów będzie nagrodzone punktem za zadanie. Podobnie jak w pierwszym etapie jest jedno zadanie gdzie tabele są łączone po takiej samej nazwie, jedno po różnych nazwach, jedno po wielu tabelach i jedno z samym sobą. Dodatkowo jedno zadanie sprawdzające umiejętność znajdowania dopełnienia zbioru. Trzeci etap to krótki przegląd pozostałych sposobów operowania na zbiorach, czyli RIGHT JOIN, wykluczenie części wspólnej oraz suma UNION, oraz UNION ALL. Podobnie jak w poprzednim etapie użytkownik jest proszony o szczegółowe zbadanie problemu, gdyż dane są tak przygotowane, aby w każdym z przypadków zwracały różny wynik. Ostatnim etapem części czwartej jest analiza warunkowych złączeń zewnętrznych. Chodzi o to, aby pokazać, że warunki złączenia mogą być podawane w różnych formach.

W przypadku gdy identyczny warunek jest podawany tuż po definicji złączenia JOIN, a tuż przed słowem kluczowym WHERE wtedy zbiory wynikowe różnią się od tych, w których warunek został umieszczony po słowie kluczowym WHERE. Dodatkowym problemem, o którym mowa w zadaniach jest zjawisko łączenia w przypadku kolumny zawierającej wartość pustą NULL. Temat został poruszony w sposób wystarczający jak na ten poziom nauki. Pytania są tak skonstruowane, aby wyjaśnić tę subtelną różnicę. Ponieważ nie są to łatwe zagadnienia, pytania zostały przedstawione w jak najprostrzej formie. Odpowiedzi na nie wymagają napisania zaledwie kilku linii. Poziom zadań nie jest więc tak wysoki, na jaki mogło by się to wydawać.

Cześć piąta kursu to głównie utrwalanie wiedzy nabytej we wcześniejszych rozdziałach. Etap pierwszy to wprowadzenie do nowego bardziej rozbudowanego schematu. Pierwsze siedem zadań prowadzi do poznania wszystkich tabel, jednocześnie ucząc drobnych elementów składni SQL np. klauzula TOP, NULLS LAST, LIKE, LENGTH(), UPPER(), LOWER(), w połączeniu z sortowaniem według określonych warunków. W kolejnych zadaniach pojawiają się coraz trudniejsze zagadnienia. W zadaniu ósmym występuje klauzula IN, w dziewiątym DATEADD(), DATEDIFF(). Zadania 10 i 11 to sprawdzian wiedzy ze złączeń. Zadanie dwunaste to kolejny sprawdzian wiedzy z użycia klauzuli IN. Zadanie 13 to sprawdzian wiedzy z podstawowych zapytań z zaprzeczeniem warunku na NULL. Zadanie 14 to ćwiczenie na konkatenację ciągu znaków oraz używanie wieloznaczników tzw. WILDCARDS. Zadanie 15 to kombinacja złączeń wielokolumnowych z warunkiem WHERE oraz sortowaniem. Zadanie 16 to jedno z najbardziej złożonych ćwiczeń w całym kursie. Wymaga ono użycia podzapytań, funkcji agregujących AVG() oraz MAX() dla GROUP BY oraz DISTINCT. W alternatywnej formie może ono zawierać złączenia i klauzule IN lub EXISTS. Podobnie jak w większości zadań istnieje kilka sposobów rozwiązania problemu, jednak oceniany jest jedynie efekty końcowy, tzn. równość zbiorów oczekiwanego i uzyskanego. Zadanie 17 to standardowe zapytanie z użyciem klauzuli GROUP BY oraz funkcji daty, gdyż ćwiczenie wymaga wyodrębnienia dnia tygodnia. W zadaniu 18 po raz kolejny zaistnieje słuszną możliwość zastosowania klauzuli HAVING. Oczywiście można tego uniknąć rozpisując zapytanie na kilka zagnieżdżonych podzapytań złączonych warunkowo, jednak w tym przypadku nastąpi widoczny zysk z zastosowania HAVING i jest to polecany sposób rozwiązania zadania. W kolejnym zadaniu istnieje jasny związek z zapytaniem zastosowanym w poprzednim ćwiczeniu. Można użyć całego zapytania jak warunek WHERE i uzyskać poprawną odpowiedź. Na zakończenie całego rozdziału dodano małe ćwiczenie na klauzulę TOP.

Ostatnia szósta część samouczka jest podsumowaniem wiedzy zdobytej w rozdziałach 1-5. Tutaj jednak poziom jest znacząco wyższy. Stosowane są wszelkie kombinacje poznanych wcześniej technik. W tym rozdziale nie jest sprawdzana wiedza lecz jest ona pogłębiana poprzez systematyczne stosowanie ich w połączeniu z innymi. Pierwsze zadanie jest łatwym złączeniem czterech tabel z warunkiem na ciąg znaków. W kolejnym zadaniu następuje proste użycie funkcji `count(*)` wraz z warunkiem korzystającym z wieloznacznika `WILDCARD`. Zadanie numer trzy ma na celu zaprezentowanie problemu pojawiającego się podczas używania funkcji agregujących typu `count(*)`. Agregowanie danych poprzez `GROUP BY` wyklucza wiersze które są puste (`NULL`), zatem funkcja `count(*)` nie pokazuje tych rekordów, które faktycznie zwracają liczbę 0. Należy mieć świadomość jakie są skutki użwania `GROUP BY` i znać alternatywne rozwiązania. Zadanie 5 wymaga zastosowania złączeń, podzapytania, klauzuli `IN` oraz `DISTINCT`. W zadaniu szóstym następuje konieczność użycia zależności funkcji `MAX()` i `COUNT()` w odpowiedniej kolejności. Zadanie numer 8 jest odpowiednikiem zadania numer 7, ale ze zmienionym warunkiem brzegowym, określonym w treści zadania. Wymaga to przepisania całego zapytania i rozwiązania problemu w zupełnie inny sposób. W zadaniu dziewiątym kolejny test na licznosc rekordów zawierający również licznosc tych, które są ignorowane przez `GROUP BY`. Zadanie dziesiąte ma na celu poznanie nowych technik manipulacji dat i przedziałów czasowych. Następne zadanie to kolejny test wiedzy z technik liczenia rekordów. Tym razem wystarczy zastosować funkcję agregującą i `GROUP BY`, aby uzyskać zadowalający rezultat. Zadanie dwunaste ma na celu przywrócenie czujności użytkownika i znów wracamy do złączeń zbiorów. To pytanie jest podchwytliwe i może wymagać powtórzenia wiedzy z poprzednich rozdziałów. Zadanie trzynaste to nowość i wprowadza nowy, ale bardzo przydatny sposób grupowania przy użyciu zaawansowanych funkcji daty. Zadanie należy pogrupować po miesiącach, a miesiące trzeba wyciągnąć z dat w formacie `TIMESTAMP`. Zadanie czternaste ma na celu zwrócenie uwagi użytkownika na subtelna różnicę pomiędzy grupowanie po miesiącu, a grupowaniu po miesiącu i roku. W zadaniu piętnastym wymagane jest połączenie dwóch funkcji dat oraz jeden funkcji `count()` w klauzuli `GROUP BY` dla uzyskania pożądanego efektu. Ostatnie zadanie całego kursu uczy jak tworzyć histogramy za pomocą SQL.

Rozdział 7

7.1. Perspektywy rozwoju

Moduł logowania

Aby poprawnie weryfikować tożsamość użytkowników, przydatna jest możliwość logowania. Podawane hasła powinny być zapisywane w bazie za pomocą funkcji haszujących oraz z użyciem soli kryptograficznej w celu uniknięcia ataków słownikowych typu „rainbow table”.

W module tym powinna być możliwość resetowania haseł w bezpieczny sposób. Do tego celu należy użyć alternatywnego adresu e-mail bądź telefonu komórkowego, na który będzie wysłany telekod. Aby to wszystko miało sens należy uprzednio zweryfikować wiarygodność podanego adresu e-mail lub użytego telefonu.

Alternatywnym rozwiązaniem jest podanie serii pytań pomocniczych, na które odpowiedzi zna tylko posiadacz konta. Jest to metoda bardzo skuteczna, lecz zawodna, gdyż użytkownicy często nie udzielają poprawnie odpowiedzi. Po trzykrotnej próbie zalogowania się niepoprawnym hasłem następuje zablokowanie konta na okres 30 minut. Dodatkową możliwością jest zintegrowanie modułu logowania z istniejącymi serwisami Single Sign-On typu Facebook, Google Accounts.

Moduł oceniania

Automatyczny system oceniania studenta, który polegałby na analizie statystycznej wyników wszystkich osób rozwiązujących zadania w danym przedziale czasowym, np. semestrze. System wyciągałby średnią wszystkich wyników i próbował dopasować oceny zgodnie z rozkładem normalnym. Nauczyciel mógłby również wpisać samodzielnie przedziały lub ocenić uczniów ręcznie. Student z poziomu swojego interfejsu powinien mieć możliwość zobaczenia statystyki swoich ocen, wykresy, dystrybuanty, estymację oceny końcowej i pozycję w rankingu na tle grupy. Dodatkowo przewiduje możliwość uzupełnienia statystyki o ilość poświęconego czasu.

Moduł zarządzania pytaniami

Przydatną funkcjonalnością byłoby rozszerzenie funkcjonalności istniejącego panelu administracyjnego, tak aby administratorzy mogli wprowadzać nowe pytania i odpowiedzi oraz konfigurować liczbę etapów dostępnych dla uczniów. Dodatkowo powinna być możliwość zmiany wyglądu, formatowania i podglądu zadań na bieżąco za pomocą edytora

WYSIWYG (What you see is what you get). Jednak problemem, który się tutaj pojawia jest duże ryzyko związane z wprowadzaniem nieodpowiednich treści. Nieumiejętnie wprowadzone dane do systemu spowodują jego zawieszenie na długi czas. Może to powodować frustrację wśród użytkowników i większe kłopoty związane z brakiem możliwości wypełnienia terminów przez uczniów. Jest to powód dla, którego stworzenie tego modułu zostało przesunięte na późniejszy okres, gdyż musi ono zostać wykonane z najwyższą starannością.

Pomocna przydałaby się funkcja rotacji pytań, dla utrudnienia kopiowania odpowiedzi przez studentów. Każdy student dostawał by inne pytania ze wspólnego zbioru wszystkich pytań. Podobnie jak ma to miejsce podczas państwowego egzaminu na prawo jazdy.

Moduł raportowania

System powinien przygotowywać cykliczne raporty wysyłane do nauczycieli poprzez dostępne kanały e-mail, sms, serwis www. Dodatkowo przydałaby się możliwość wysyłania wiadomości e-mail z przypomnieniami o zbliżającym się terminie wykonania zadania. Przypomnienia powinny być wysyłane do uczniów oraz nauczycieli.

Internacjonalizacja

Umożliwienie korzystania z serwisu użytkownikom z innych krajów. W szczególności wprowadzenie języków: angielski, niemiecki.

Integracja z platformą Moodle

Wystawić API, lub Webservice'y, aby można było wykorzystać funkcjonalności na platformach edukacyjnych, typu Wikamp. Ponadto istnieje możliwość rozszerzenia zakresu użycie frameworka do innych zastosowań, nauki innych przedmiotów, np. do systemów operacyjnych, programowanie komputer-człowiek, podstawy programowania, HTML5, SVG, Js, XML, może nawet matematyka, na podstawie Webwork, tutorial Linuxa, Javy, fizyki, itd.

Moduł wykrywania plagiatu

System mógłby przeprowadzać analizę logów i porównywać odpowiedzi udzielone przez studentów. Jeśli weźmiemy pod uwagę nazwy aliasów, formatowanie, wielkość liter, sposób pisania podzapytań, możemy jednoznacznie rozpoznać każdego użytkownika systemu. Wykrywanie plagiatu stałoby się wtedy bardzo proste i szybkie. Ponieważ złożoność obliczeniowa tego problemu jest dość duża trzeba przeprocesować wszystkie informacje w tle, a raporty cyklicznie wysyłać prowadzącemu zajęcia.

Personalizacja wyglądu

Jeśli popularność strony będzie wystarczająco duża rozważam wprowadzenie personalizacji wyglądu opartej o kaskadowe arkusze stylów CSS. Do tego celu będzie potrzebne dodanie tabeli do bazy danych, która będzie miała za zadanie przechowywać wszystkie wybrane opcje przez użytkownika. Do takich opcji będą między innymi należeć: kolejność wyświetlania zakładek, liczba i widoczność funkcjonalności, wielkość i kolor czcionki, kolor lub obraz tła. Dodawanie zdjęć swojego profilu, zapisywanie ostatnio rozwiązanych zadań. Przechowywanie informacji na temat przeczytanych porad i popupów. Ustawienia języka i lokalizacji np. dla walut lub dat.

Interaktywność

W kolejnej odsłonie bardzo przydałaby się pewna inteligencja w zachowaniu aplikacji. Nie mówię tutaj o sztucznej inteligencji, lecz o prostym warunkowym postępowaniu w przypadku problemów z udzieleniem odpowiedzi. Na podstawie statystyk prowadzonych przez moduł oceniania, można by wywnioskować poziom prezentowany przez użytkownika i dostosowywać pytania do jego potrzeb. Należałoby wspierać ucznia w potrzebie, gdy utknie na pytaniu i pokazywać mu dodatkowe lekcje lub dawać dodatkowe ćwiczenia z danej dziedziny.

Zapamiętywanie wpisywanych zapytań

Jednym z elementów personalizacji zaawansowanej mogłaby się stać możliwość zapamiętywania wpisanych wcześniej ciągów znaków i reużywania ich w trakcie pisania. Dane musiałby być indeksowane, aby były dostępne szybko podczas pisania na klawiaturze. Podpowiedzi składniowe tego typu są szczególnie potrzebne, gdy mamy do wyboru wiele nazw zaczynających się na tę samą literę. Podobnie działają wszystkie duże systemy, np. wyszukiwarka Google lub wyszukiwarka Facebook. Zawsze w pierwszej kolejności pokazuje nasze ostatnio wyszukiwane rezultaty, bo zazwyczaj te są właśnie najbardziej użyteczne. Dodatkowo bardzo pomocna okazałaby się funkcja zapamiętywania wypełnionych pytań tak, aby nie ginęły podczas odświeżania strony. Ponadto użytkownik powinien być przenoszony do swojego ostatniego rozwiązanego pytania, podczas każdego logowania.

Podpowiedzi składni w czasie rzeczywistym

Podpowiedzi składni w czasie rzeczywistym są największym wyzwaniem ze względów technicznych, ponieważ wymaga to bardzo dużej mocy obliczeniowej serwera. Wszystkie zapytania mogły by być interpretowane poprzez silnik wyrażeń regularnych i kompilowane

w locie, zanim zostaną faktycznie wykonane. Jednak jeśli system zyska dużą popularność będzie wymagane zastosowanie mocniejszej jednostki oraz zwiększenie przepustowości łącza. Komunikacja obustronna jest możliwa dzięki technologii WebSocket, tak jak komunikuje się normalny system desktopowy, ale zawsze będą pojawiały się znaczne opóźnienia i przynajmniej przez najbliższe dziesięć lat nic w tej kwestii nie da się zrobić.

Podpowiadanie nazw obiektów

Bardzo prostym sposobem na osiągnięcie kompromisu pomiędzy podpowiadaniem składni w czasie rzeczywistym, a podpowiadaniem leniwym (na sam koniec). Jest zastosowanie trybu buforowanego. Gdzie nazwy obiektów oraz historia zapytań będą co jakiś czas przesyłane do klienta w paczkach i przechowywane bezpośrednio w pamięci operacyjnej użytkownika. Jest to stosunkowo łatwe do wykonania, gdyż JavaScript tak, jak każdy język ma swoją przestrzeń pamięci, która może być wypełniona i wielokrotnie odczytywana bez obciążania łącza. Wadą tego rozwiązania jest brak możliwości natychmiastowego podpowiadania nazw obiektów, które zostały niedawno dodane. Odświeżanie musi odbywać się cyklicznie w rozsądnych odstępach czasu.

Zmiana interfejsu graficznego

Obecnie warstwa prezentacji front-end jest oparta na czystym HTMLu z dodatkami jQuery. Nie jest to optymalne rozwiązanie, gdyż wymagane jest wiele przeładowań statycznych stron *.html. Do profesjonalnego wyglądu strony należy użyć jednego z frameworków. Do dyspozycji mamy Backbone.js, AngularJS, Ember.js i wiele innych. Ich poznanie wiąże się z kilkutygodniową nauką, ale jeśli aplikacja ma wyglądać nienagannie jest to konieczność, na którą należy zagospodarować czas.

7.2. Wnioski i podsumowanie

Zaprezentowany w pracy sposób wykorzystania dostępnych technologii może posłużyć jako podstawa do tworzenia lepszych, bardziej rozbudowanych i bardziej zaawansowanych narzędzi lub nawet frameworków wspomagających naukę w wielu dziedzinach. Jestem pewny, że w przyszłości nauczanie będzie polegać w głównej mierze na użytkowaniu podobnych systemów. Programiści już teraz mają do dyspozycji technologie pozwalające tworzyć niemalże nieograniczone funkcjonalnie aplikacje webowe.

Problemem jest jednak koszt rozwoju takiej aplikacji. W kontekście nauczania podstaw baz danych, okazuje się, że koszt przygotowania takiego interaktywnego kursu przez jedną osobę jest znacznie większy niż przeprowadzenie standardowego szkolenia lub zorganizowania serii wykładów w ciągu jednego semestru.

Jeśli jednak weźmiemy pod uwagę fakt, że nad tym zagadnieniem pracują setki osób w całym kraju, wtedy sumaryczny koszt ich pracy jest większy. Zatem idealnym rozwiązaniem było by połączenie wysiłku wszystkich osób pracujących nad tym zagadnieniem, tak aby materiał nie był duplikowany. Dodatkowo powodem, dla którego warto zastanowić się nad takim rozwiązaniem, jest fakt, że praca musi być cyklicznie powtarzana co kilka miesięcy przez te same osoby. Oczywiście jest, że lepiej było by poświęcić ten czas na zgłębianie tematu lub poprawę jakości i objętości materiału i lepiej było by wykorzystać wiedzę swoich poprzedników.

Wzorując się na portalach społecznościowych, gdzie treść jest generowana przez ogół użytkowników, a nie przez twórców systemu, można stworzyć podobny system o charakterze edukacyjnym. Rozwój takiego systemu nie wymagał by szczególnych nakładów finansowych, gdyż treść była by dodawana na zasadzie ad-hoc, w razie potrzeby przez społeczność i zajmowała by zaledwie kilka minut przeciętnemu użytkownikowi. Wszyscy uczestnicy kursu mieli by możliwość tworzenia takiego portalu nie posiadając żadnych umiejętności programistycznych. System powinien budować się samodzielnie i potęgować swoje możliwości wraz z liczbą użytkowników korzystających z niego.

W przypadku niszowych tematów, np. związanych z informatyką, dzielenie się informacjami jest szczególnie ważne, gdyż informacje są trudno dostępne, a ludzie zajmujący się tematem zawodowo są zwykle pochłonięci pracą, więc nie można skorzystać z ich wiedzy. Takie portale dbały by o to, żeby trudno dostępne informacje stawały się łatwo osiągalne.

W odniesieniu do tematu pracy połączenie wysiłku wszystkich osób zajmujących się bazami danych umożliwiłoby powstanie bardzo rozbudowanego systemu promującego nauczanie języków zapytań. Upowszechnienie wiedzy o bazach danych w Polsce i na świecie jest bardzo pożądane. Specjaliści od baz danych są jednymi z lepiej opłacanych w branży IT, a to dlatego, że jest na nich duże zapotrzebowanie. Aby sprostać wymaganiom rynku potrzebne jest szybsze kształcenie specjalistów. Aby szybciej kształcić ludzi, należy im to ułatwić. Zatem wniosek jest jeden: rozwój systemów takich jak ten jest bardzo potrzebny. Jednak trzeba to robić z rozsądkiem, gdyż powielanie tej samej pracy nie przynosi wiele dobrego. Ma to pewne pozytywne strony, jak na przykład podnoszenie jakości poprzez zdrową rywalizację pomiędzy twórcami aplikacji. Jednak w tym konkretnym wypadku nie ma jednak o co konkurować, bo większość podobnych systemów jest darmowych i tworzonych z misją a nie dla chęci zysku. Jeśli chcemy tę misję wypełnić należy się

zjednoczyć i wspólnie brać udział w tworzeniu uniwersalnego systemu dostępnego dla wszystkich.

Podsumowując, zaprezentowana praca jest pokazem możliwości dostępnych dla osób zajmujących się dydaktyką. Poza pisanem bardzo rozbudowanych książek dla studentów z całego świata powinno kłaść się nacisk na część praktyczną i systematyczną weryfikację wiedzy. Literatura techniczna jest bardzo ważnym elementem kształcenia, ale teoria niepoparta ćwiczeniami praktycznymi jest mniej skuteczna. Aby osiągnąć lepsze rezultaty należy skupić się na rozwoju interaktywnych narzędzi do nauki.

Podczas tworzenia aplikacji nauczyłem się wielu pożytecznych technik, poznałem wiele nowych bibliotek i frameworków, między innymi Spring Framework.

Cel wyznaczony przez standardy kształcenia ECTS został osiągnięty. Pozyskanie wiedzy i informacji dotyczących nauczania języków bazodanowych oraz wykorzystanie wiedzy nabytej podczas toku studiów doprowadziło do powstania pożytecznego projektu, który może służyć innym. Opanowanie materiału niezbędnego do stworzenia aplikacji był czasochłonne, ale bardzo rozwijające i satysfakcjonujące. Projekt nie jest niczym odkrywczym choć spełnił swój cel i zmusił mnie do wykonania sporego wysiłku. Uważam że stworzenie tego projektu było bardzo potrzebne w mojej karierze jako programisty.

Spis rysunków

Rysunek 1: Three-Tier-Architecture	17
Rysunek 2: Data Access Object.....	19
Rysunek 3: Warstwa Front-end	20
Rysunek 4: Embedded mode	31
Rysunek 5: Server mode	31
Rysunek 6: Mixed mode	32
Rysunek 7: Diagram przypadków użycia – Przykładowy workflow	34
Rysunek 8: Diagram przypadków użycia – Weryfikacja odpowiedzi.....	34
Rysunek 9: Diagram przypadków użycia – Przyznanie punktu	35
Rysunek 10: Strona główna aplikacji.	44
Rysunek 11: Przykładowe zadanie.	44
Rysunek 12: Treść zadania.	45
Rysunek 13: Podpowiedzi.	45
Rysunek 14: Poprawna odpowiedź.	46
Rysunek 15: Niepoprawna odpowiedź.....	46
Rysunek 16: Wynik zapytania.....	47
Rysunek 17: Oczekiwany rezultat.....	48
Rysunek 18: Strona z zadaniami.	48
Rysunek 19: Pasek nawigacji.	49
Rysunek 20: Komunikat błędu.	50
Rysunek 21: Forum dyskusyjne.	51
Rysunek 22: Strona kontaktowa.	52
Rysunek 23: Komunikaty organizacyjne.	53
Rysunek 24: Menu Zaawansowane.	54
Rysunek 25: Panel Administracyjny.	54
Rysunek 26: Udzielone odpowiedzi.	55
Rysunek 27: Pogrupowane odpowiedzi każdego z uczniów.....	55
Rysunek 28: Wyniki studentów z trwającego kolokwium.....	56
Rysunek 29: Ranking studentów rozwiązujących zadania najszybciej.	57
Rysunek 30: Moduł pracy zespołowej.....	58

Bibliografia

Wszystkie normy RFC dostępne są na stronie WWW: <http://www.rfc-editor.org/>

Wszystkie dokumenty IEEE dostępne są na stronie: <http://ieeexplore.ieee.org/>

Wszystkie dokumenty ACM dostępne są na stronie: <http://portal.acm.org/>

- [1] R. Elmasri i S. Navathe, Wprowadzenie do systemów baz danych, Gliwice: Helion, 2005.
- [2] J. D. Ullman i J. Widom, Podstawowy wykład z systemów baz danych, Warszawa: WNT, W-wa, (seria: Klasyka Informatyki), 2000.
- [3] J. Kasprzak, „Język SQL – historia, standardy,” Styczeń 2013. [Online]. Available: <http://www.sqlpedia.pl/jezyk-sql-historia-standardy/>.
- [4] W. W. Eckerson, „Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications,” *Open Information Systems*, tom 3, nr 20, 1995.
- [5] M. Haverbeke, „JavaScript i wszystko jasne,” San Francisco, CA, William Pollock, 2011.
- [6] R. Jońca, „Kurs jQuery. Wprowadzenie. Zalety, podstawowe zasady, pierwszy skrypt i rozszerzenia,” Webhosting.pl, 2009.
- [7] STOMP, „The Simple Text Oriented Messaging Protocol,” 22 październik 2012. [Online]. Available: <https://stomp.github.io/>.
- [8] IETF, „The application/json Media Type for JavaScript Object Notation (JSON),” July 2006. [Online]. Available: <http://tools.ietf.org/html/rfc4627>.
- [9] JSON, „JavaScript Object Notation,” December 1999. [Online]. Available: <http://www.json.org/>.

- [10] J. Piechowiak, „Jak szybko stworzyć ładny CSS z frameworkiem Bootstrap,” ALT CONTROL DELETE, Poznań, 2014.
- [11] B. Danowski, „Wstęp do CSS,” 9 marzec 2010. [Online]. Available: <http://webmaster.helion.pl>.
- [12] C. Bronny, „Technologia JDBC w praktyce,” Uniwersytet Warszawski, Warszawa, 2006.
- [13] M. Granat i A. Rozmysłowicz, „Niezbędne serwery aplikacji. Wprowadzenie do technologii JBoss i Apache Tomcat,” Fatal Error, Zielona Góra, 2009.
- [14] W. Wheeler i J. White, „Spring w praktyce,” Manning Publication, Gliwice, 2013.
- [15] T. Kaczanowski, „Gradle – Mocarne narzędzie do budowy projektów,” Java exPress, Kraków, 2009.

Załączniki

1. Oświadczenie o samodzielnym wykonaniu pracy dyplomowej
2. Oświadczenie dotyczące zgodności płyty CD z wydrukiem pracy dyplomowej.
3. Płyta CD
4. Kod źródłowy.

Załącznik nr 4

Dwa główne controllery: służące do komunikacji z Front-end'em:

```
@Controller
public class QueryController {

    @Autowired
    private QueryService queryService;
    @Autowired
    private SimpMessagingTemplate simpMessagingTemplate;
    @Autowired
    private HttpSession session;

    private static final Logger logger = LoggerFactory.getLogger(QueryController.class);

    @Autowired
    public QueryController(QueryService queryService, SimpMessagingTemplate
                           simpMessagingTemplate, HttpSession session) {
        this.queryService = queryService;
        this.simpMessagingTemplate = simpMessagingTemplate;
        this.session = session;
    }

    @MessageMapping("/query")
    @SendToUser("/queue/position-updates")
    public InbhzResultSet executeQuery(Request message, MessageHeaders messageHeaders) {
        String clientId = getClientString(messageHeaders);
        logger.info("Client ID: "+clientId);
        return queryService.select(message, clientId);
    }

    private String getClientString(MessageHeaders messageHeaders) {
        if (messageHeaders==null) { return null; }
        LinkedMultiValueMap<String, String> nativeHeaders =
            (LinkedMultiValueMap<String, String>) messageHeaders.get("nativeHeaders");
        if (nativeHeaders==null) { return null; }
        List<String> strings = nativeHeaders.get("client-id");
        if (strings==null || strings.size()==0) { return null; }
        return strings.get(0);
    }

    private String getTimestamp() {
        LocalDateTime date = LocalDateTime.now();
        return date.format(DateTimeFormatter.ISO_DATE_TIME);
    }

    @ResponseBody
    @RequestMapping("/sessionId")
    public String sessionId() {
        return this.session.getId();
    }

    @MessageMapping("/teamHello")
    @SendTo("/topic/greetings")
    public InbhzResultSet greeting(Request message) throws Exception {
        return queryService.greeting(message);
    }

    @MessageMapping("/teamExecute")
    @SendTo("/topic/execute")
    public InbhzResultSet execute(Request message) throws Exception {
        return queryService.execute(message);
    }

    @MessageMapping("/teamQuery")
    @SendTo("/topic/query")
    public InbhzResultSet select(Request message) throws Exception {
        return queryService.select(message);
    }
}
```

oraz

```
@Controller
public class TaskController {

    @Autowired
    private AdmService admService;
    @Autowired
    AdmStudentService admStudentService;
    @Autowired
    private SimpMessagingTemplate simpMessagingTemplate;
    @Autowired
    private HttpSession session;

    private static final Logger logger = LoggerFactory.getLogger(QueryController.class);

    @Autowired
    public TaskController(AdmService admService,
                        AdmStudentService admStudentService,
                        SimpMessagingTemplate simpMessagingTemplate,
                        HttpSession session) {
        this.admService = admService;
        this.admStudentService = admStudentService;
        this.simpMessagingTemplate = simpMessagingTemplate;
        this.session = session;
    }

    @PostMapping("/task/query")
    @SendToUser("/queue/position-updates")
    public InbJzResultSet executeQuery(Request message, MessageHeaders messageHeaders) {
        String clientId = getClientString(messageHeaders);
        logger.info("Client ID: "+clientId);
        return admStudentService.select(message);
    }

    private String getClientString(MessageHeaders messageHeaders) {
        if (messageHeaders==null) { return null; }
        LinkedMultiValueMap<String, String> nativeHeaders =
            (LinkedMultiValueMap<String, String>) messageHeaders.get("nativeHeaders");
        if (nativeHeaders==null) { return null; }
        List<String> strings = nativeHeaders.get("client-id");
        if (strings==null || strings.size()==0) { return null; }
        return strings.get(0);
    }

    private String getTimestamp() {
        LocalDateTime date = LocalDateTime.now();
        return date.format(DateTimeFormatter.ISO_DATE_TIME);
    }

    @ResponseBody
    @RequestMapping("/task/sessionId")
    public String sessionId() {
        return this.session.getId();
    }

    @PostMapping("/getTaskById")
    @SendToUser("/queue/getTaskById")
    public Task getTaskById(int id) throws Exception {
        return null;
    }

    @PostMapping("/getTask")
    @SendToUser("/queue/getTask")
    public Task getTask(int chapter, int number) throws Exception {
        return null;
    }

    @PostMapping("/getAllTasks")
    @SendToUser("/queue/getAllTasks")
    public List<Task> getTasks() throws Exception {
        return null;
    }
}
```

```
}
```

Klasy konfiguracyjne:

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic/", "/queue/");
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/inbjz")
            .withSockJS()
            .setInterceptors(new HttpSessionIdHandshakeInterceptor());
    }
}

@Configuration
@ComponentScan(value={"pl.lodz.p.components"})
public class SpringConfiguration {

}
```

Klasy Serwisy:

```
@Service
public class AdmService extends DbService {
    private boolean criticalError = false;
    private static final Logger logger = LoggerFactory.getLogger(AdmService.class);

    @Override
    protected DatabaseDao getDatabase(Request request) {
        return getDatabase();
    }

    protected DatabaseDao getDatabase() {
        return DatabaseAdmImpl.getInstance(User.SA);
    }

    public String getAnswer(int taskId) {
        DatabaseDao database = getDatabase();
        List<String[]> actual = null;
        try {
            actual = database.executeQuery(getQuery(taskId));
        } catch (UncategorizedSQLException e) {
            logger.error(e.getCause().getMessage());
            return "Could not retrieve answer for this ID:" + taskId;
        } catch (DuplicateKeyException | JdbcSQLException e) {
            logger.error(e.getCause().getMessage());
            return "Could not retrieve answer for this ID:" + taskId;
        } catch (DataIntegrityViolationException | SQLException e) {
            logger.error(e.getCause().getMessage());
            return "Could not retrieve answer for this ID:" + taskId;
        } catch (Exception e) {
            logger.error(e.getCause().getMessage());
            return "Could not retrieve answer for this ID:" + taskId;
        }
        if (actual != null && actual.size() == 1
            && actual.get(0) != null && actual.get(0).length == 1) {
            return actual.get(0)[0];
        } else {
            return null;
        }
    }
}
```

```

    }
}

private String getQuery(int taskId) {
    return "select a.answer from tasks t" +
           "join answers a on t.answer_id=a.id where t.id="+taskId;
}

public String getType(int taskId) {
    DatabaseDao database = getDatabase();
    List<String[]> actual = null;
    try {
        actual = database.executeQuery("select type from tasks where id="+taskId);
    } catch (UncategorizedSQLException e) {
        logger.error(e.getCause().getMessage());
        return "Could not retrieve type for this ID:"+taskId;
    } catch (DuplicateKeyException | JdbcSQLException e) {
        logger.error(e.getCause().getMessage());
        return "Could not retrieve type for this ID:"+taskId;
    } catch (DataIntegrityViolationException | SQLException e) {
        logger.error(e.getCause().getMessage());
        return "Could not retrieve type for this ID:"+taskId;
    } catch (Exception e) {
        logger.error(e.getCause().getMessage());
        return "Could not retrieve type for this ID:"+taskId;
    }
    if (actual!=null && actual.size()==1
        && actual.get(0)!=null && actual.get(0).length==1) {
        return actual.get(0)[0];
    } else {
        return null;
    }
}

public int logPoint(int taskId, String clientId, String givenAnswer, boolean correct) {
    DatabaseDao database = getDatabase();
    int answerId = getNextAnswerSeq();
    if (answerId==-1) {
        logger.error("Could not retrieve a sequence number for logged_answer");
        return -1;
    }
    String answer;
    if (!criticalError) {
        answer = givenAnswer.replaceAll("'", "");
    } else {
        answer = convertToUnicode(givenAnswer);
    }
    try {
        database.executeStmt("insert into logged_answers values (" +
                             answerId + ", '"+answer+"')");

        database.executeStmt("insert into logs " +
                             "(id, student_id, client_id, task_id, answer_id, correct) " +
                             "values (" +
                             "LOGS_SEQ_ID.nextval," +
                             clientId + ", " +
                             session_id + ", " +
                             clientId + ", " +
                             taskId + ", " +
                             answerId + ", " +
                             (correct ? "TRUE" : "FALSE")
                             +");");
    } catch (SQLException e) {
        if (!criticalError) {
            criticalError = true;
            logger.error("Critical exception has occurred. Trying to save as unicode...");
            logPoint(taskId, clientId, givenAnswer, correct);
        }
        logger.error(e.getCause().getMessage());
    }
    logger.info("Student "+clientId+" has answered question ID "+taskId+" correctly.");
    return 0;
}

```



```

private String convertToUnicode(String s) {
    StringBuilder sb = new StringBuilder();
    s.codePoints().forEach((i) -> {
        sb.append("\\u");
        sb.append(Integer.toHexString(i));
    });

    if (s.length() >= 2000) {
        s = s.substring(0, 2000-4);
    }
    return s;
}

private int getNextAnswerSeq() {
    DatabaseDao database = getDatabase();
    List<String[]> actual = null;
    try {
        actual = database.executeQuery("select LOGGED_ANSWERS_SEQ_ID.nextval;");
    } catch (UncategorizedSQLException e) {
        Logger.error(e.getCause().getMessage());
    } catch (DuplicateKeyException | JdbcSQLException e) {
        Logger.error(e.getCause().getMessage());
    } catch (DataIntegrityViolationException | SQLException e) {
        Logger.error(e.getCause().getMessage());
    } catch (Exception e) {
        Logger.error(e.getCause().getMessage());
    }
    if (actual != null && actual.size() == 1
        && actual.get(0) != null && actual.get(0).length == 1) {
        return Integer.parseInt(actual.get(0)[0]);
    } else {
        return -1;
    }
}
}

```

```

@Service
public class QueryService extends DbService {

    private static final Logger logger = LoggerFactory.getLogger(QueryService.class);

    @Autowired
    private AdmService admService;

    @Autowired
    public QueryService(AdmService admService) {
        this.admService = admService;
    }

    @Override
    protected DatabaseDao getDatabase(Request request) {
        DatabaseDao database;
        if ("real".equals(request.getMode())) {
            database = DatabaseStudImpl.getInstance();
        } else {
            database = new DatabaseStudImpl();
        }
        return database;
    }

    public InbjzResultSet select(Request request, String clientId) {
        DatabaseDao database = getDatabase(request);
        String[] actualHeaders;
        List<String[]> actual;
        String answer = admService.getAnswer(request.getTaskId());
        String definedType = admService.getType(request.getTaskId());
        String[] expectedHeaders = new String[] {
            "Nie znaleziono odpowiedzi do tego zadania."
        };
        List<String[]> expected = null;
        InbjzResultSet res = new InbjzResultSet();
        res.setTaskId(request.getTaskId());
    }
}

```

```

        res.setType(Type.QUERY);
        try {
            actual = database.executeQuery(request.getQuery());
            actualHeaders = database.getLabels(request.getQuery());
            if ("QUERY".equals(definedType) && answer!=null) {
                expected = database.executeQuery(answer);
                expectedHeaders = database.getLabels(answer);
            }
        } catch (UncategorizedSQLException e) {
            if (e.getSQLException().getErrorCode()==90002){
                return fallBackUpdate(request, res);
            } else {
                return handleException(e, res);
            }
        } catch (DuplicateKeyException | JdbcSQLException e) {
            return handleException(e, res);
        } catch (DataIntegrityViolationException | SQLException e) {
            return handleException(e, res);
        } catch (Exception e) {
            return handleException(e, res);
        }
        res.setActualHeaders(actualHeaders);
        res.setActual(actual);
        res.setExpected(expected);
        res.setExpectedHeaders(expectedHeaders);
        res.setTaskId(request.getTaskId());
        res.setStatus(Status.OK);
        res.setCorrect("QUERY".equals(definedType) ? equals(actual, expected) : true);
        res.setContent("String representation of this result");
        try {
            admService.logPoint(request.getTaskId(), clientId, request.getQuery(),
                               res.isCorrect());
        } catch (Throwable t) {
            Logger.error("Problem with a logging module.");
            if (t.getCause()!=null) {
                Logger.error(t.getCause().getMessage());
            }
        }
        return res;
    }

    public AdmService getAdmService() {
        return admService;
    }
}

```

```

@Service
public class DbService {

    private static final Logger logger = LoggerFactory.getLogger(DbService.class);

    // @Autowired
    // private AdmService admService;

    @Transactional
    public InbjzResultSet select(Request request) {
        DatabaseDao database = getDatabase(request);
        List<String[]> actual = null;
        String[] actualHeaders = new String[]{"null"};
        InbjzResultSet res = new InbjzResultSet();
        res.setTaskId(request.getTaskId());
        res.setType(Type.QUERY);
        try {
            actual = database.executeQuery(request.getQuery());
            actualHeaders = database.getLabels(request.getQuery());
        } catch (UncategorizedSQLException e) {
            if (e.getSQLException().getErrorCode()==90002){
                return fallBackUpdate(request, res);
            } else {
                return handleException(e, res);
            }
        }
    }
}

```

```

    } catch (DuplicateKeyException | JdbcSQLException e) {
        return handleException(e, res);
    } catch (DataIntegrityViolationException | SQLException e) {
        return handleException(e, res);
    } catch (Exception e) {
        return handleException(e, res);
    }
    res.setActualHeaders(actualHeaders);
    res.setActual(actual);
    String[] expectedHeaders = actualHeaders;
    res.setExpectedHeaders(expectedHeaders);
    List<String[]> expected = actual;
    res.setExpected(expected);
    res.setTaskId(request.getTaskId());
    res.setStatus(Status.OK);
    res.setCorrect(equals(actual, expected));
    res.setContent("String representation of this result");
    return res;
}

protected InbjzResultSet handleException(Throwable t, InbjzResultSet res) {
    logger.error(t.getClass() + " " + t.getMessage());
    res.setStatus(Status.ERROR);
    res.setCorrect(false);
    res.setErrorMessage(t.getCause().getMessage());
    return res;
}

protected InbjzResultSet fallBackUpdate(Request request, InbjzResultSet res) {
    try {
        return update(request);
    } catch (DuplicateKeyException e1) {
        return handleException(e1, res);
    } catch (DataIntegrityViolationException | UncategorizedSQLException
            | SQLException e1) {
        return handleException(e1, res);
    } catch (Exception e1) {
        return handleException(e1, res);
    }
}

protected boolean equals(List<String[]> actual, List<String[]> expected) {
    if (actual==null && expected==null) {
        return true;
    }
    if (actual==null || expected==null) {
        return false;
    }
    if (actual.size()!=expected.size()) {
        return false;
    }
    if (actual.size()>0) {
        if (actual.get(0).length!=expected.get(0).length) {
            return false;
        }
    }
    for (int i=0; i<actual.size(); i++) {
        for (int j=0; j<actual.get(i).length; j++) {
            if (actual.get(i)[j]==null && expected.get(i)[j]==null){
                continue;
            }
            if (actual.get(i)[j]==null || !actual.get(i)[j].equals(expected.get(i)[j])) {
                return false;
            }
        }
    }
    return true;
}

public InbjzResultSet greeting(Request request) {
    DatabaseDao database = getDatabase(request);
    List<String[]> result = null;
    try {

```

```

        result = database.executeQuery(request.getQuery());
    } catch (SQLException e) {
        Logger.error(e.getMessage());
    }
    StringBuilder sb = new StringBuilder();
    for (String[] row : result) {
        sb.append(Arrays.toString(row));
        sb.append("\n");
    }
    return new InbjzResultSet(sb.toString());
}

public InbjzResultSet execute(Request request) {
    DatabaseDao database = getDatabase(request);
    InbjzResultSet res = new InbjzResultSet();
    String output;
    res.setTaskId(request.getTaskId());
    res.setType(Type.EXECUTE);
    try {
        output = database.executeStmt(request.getQuery());
    } catch (DuplicateKeyException | UncategorizedSQLException | JdbcSQLException e) {
        return handleException(e, res);
    } catch (SQLException | BadSqlGrammarException e) {
        return handleException(e, res);
    } catch (Exception e) {
        return handleException(e, res);
    }
    res.setTaskId(request.getTaskId());
    res.setStatus(Status.OK);
    res.setCorrect(true);
    res.setConsoleOutput(output);
    res.setContent("String representation of this result");
    return res;
}

public InbjzResultSet update(Request request) throws SQLException {
    DatabaseDao database = getDatabase(request);
    InbjzResultSet res = new InbjzResultSet();
    String output;
    res.setTaskId(request.getTaskId());
    res.setType(Type.EXECUTE);
    try {
        output = database.update(request.getQuery());
    } catch (DuplicateKeyException | UncategorizedSQLException e) {
        return handleException(e, res);
    } catch (BadSqlGrammarException e) {
        return handleException(e, res);
    }
    res.setTaskId(request.getTaskId());
    res.setStatus(Status.OK);
    res.setCorrect(true);
    res.setConsoleOutput(output);
    res.setContent("String representation of this result");
    try {
        getAdmService().logPoint(request.getTaskId(), request.getStudentId(),
request.getQuery(), res.isCorrect());
    } catch (Throwable t) {
        Logger.error("Problem with a logging module.");
    }
    res.setContent("String representation of this result");
    return res;
}

protected DatabaseDao getDatabase(Request request) {
    DatabaseDao database;
    if ("real".equals(request.getMode())) {
        database = DatabaseStudImpl.getInstance();
    } else {
        database = new DatabaseStudImpl();
    }
    return database;
}

```

```

    public AdmService getAdmService() {
        return null;
    }
}

```

```

public class DatabaseAdmImpl implements DatabaseDao {

    private JdbcTemplate jdbcTemplate;
    private static final Logger logger = LoggerFactory.getLogger(DatabaseAdmImpl.class);
    private static DatabaseAdmImpl instanceSA;
    private static DatabaseAdmImpl instanceStudent;

    private DatabaseAdmImpl(User user) {
        SimpleDriverDataSource dataSource = getDataSource(user);
        jdbcTemplate = new JdbcTemplate(dataSource);
    }

    private SimpleDriverDataSource getDataSource(User user) {
        SimpleDriverDataSource dataSource = new SimpleDriverDataSource();
        dataSource.setDriverClass(org.h2.Driver.class);
        dataSource.setUrl("jdbc:h2:./adm");

        if (user==User.SA) {
            dataSource.setUsername("SA");
            @SuppressWarnings("unused")
            String salt = "Politechnika";
            @SuppressWarnings("unused")
            String password = "*****";
            String sha512 = "*****";
            dataSource.setPassword(sha512);
        }
        if (user==User.STUDENT) {
            dataSource.setUsername("STUDENT");
            dataSource.setPassword("*****");
        }
        return dataSource;
    }

    public static DatabaseAdmImpl getInstance(User user) {
        if (User.SA==user) {
            return instanceSA == null
                ? new DatabaseAdmImpl(user) : instanceSA;
        } else {
            return instanceStudent == null
                ? new DatabaseAdmImpl(user) : instanceStudent;
        }
    }

    @Override
    public List<String[]> executeQuery(String sql) throws SQLException {
        String trimmed = sql;
        if (sql.length() > MAX_CHAR) {
            trimmed = sql.substring(0, MAX_CHAR)+"...";
        }
        logger.info("Querying: " + trimmed);
        if (hasProhibitedCommand(sql)) {
            Throwable t = new Throwable("Nice try :)");
            throw new SQLException("Nice try :", t);
        }
        return jdbcTemplate.query(sql,
            (rs, rowNum) -> {
                int columnCount = rs.getMetaData().getColumnCount();
                String[] row = new String[columnCount];
                for (int i = 0; i < columnCount; i++) {
                    row[i] = rs.getString(i + 1);
                }
                return row;
            });
    }

    private boolean hasProhibitedCommand(String sql) {

```

```

        sql = sql.replaceAll("\\s+", " ").toLowerCase();
        if (sql.contains("drop schema superuser")) {
            Logger.error("Input contains drop schema");
            return true;
        }
        if (sql.contains("drop user")) {
            Logger.error("Input contains drop user");
            return true;
        }
        if (sql.contains("drop all")) {
            Logger.error("Input contains drop all");
            return true;
        }
        return false;
    }

    @Override
    public String executeStmt(String sql) {
        jdbcTemplate.execute(sql);
        return sql + " executed.";
    }

    @Override
    public String update(String sql) {
        int rows = jdbcTemplate.update(sql);
        return rows + " row" + (rows==1 ? "" : "s") + " affected.";
    }

    @Override
    public String[] getLabels(String sql) {
        SqlRowSet rs = jdbcTemplate.queryForRowSet(sql);
        if (rs==null) return new String[] {"null"};
        SqlRowSetMetaData metaData = rs.getMetaData();
        String[] columns = metaData.getColumnNames();
        String[] columnNames = new String[columns.length];
        for (int i=0; i<columns.length; i++) {
            String columnLabel = metaData.getColumnLabel(i+1);
            columnNames[i] = columnLabel != null ? columnLabel : metaData.getColumnName(i+1);
        }

        return columnNames;
    }
}

```

```

public class DatabaseStudImpl implements DatabaseDao {

    private JdbcTemplate jdbcTemplate;
    private static final Logger logger = LoggerFactory.getLogger(DatabaseStudImpl.class);
    private static DatabaseStudImpl instance;

    public DatabaseStudImpl() {
        SimpleDriverDataSource dataSource = getDataSource(User.SA);
        jdbcTemplate = new JdbcTemplate(dataSource);
        init(jdbcTemplate);
    }

    private SimpleDriverDataSource getDataSource(User user) {
        SimpleDriverDataSource dataSource = new SimpleDriverDataSource();
        dataSource.setDriverClass(org.h2.Driver.class);
        dataSource.setUrl("jdbc:h2:./mem");

        if (user==User.SA) {
            dataSource.setUsername("SA");
            @SuppressWarnings("unused")
            String salt = "Politechnika";
            @SuppressWarnings("unused")
            String password = "*****";
            String sha512 = "*****";
            dataSource.setPassword(sha512);
        }
        if (user==User.STUDENT) {

```

```

        dataSource.setUsername("STUDENT");
        dataSource.setPassword("*****");
    }
    return dataSource;
}

public static DatabaseStudImpl getInstance() {
    return instance == null ? new DatabaseStudImpl() : instance;
}

private void init(JdbcTemplate jdbcTemplate) {
    jdbcTemplate.execute(DatabaseUtils.getHrSchema());
    jdbcTemplate.execute(DatabaseUtils.getHrData());
    jdbcTemplate.execute(DatabaseUtils.getTworzPracownicy());
    jdbcTemplate.execute(DatabaseUtils.getWstawDanePracownicy());
}

@Override
public List<String[]> executeQuery(String sql) throws SQLException {
    if (sql==null || sql.trim().equalsIgnoreCase("")) {
        Throwable t = new Throwable("Brak polecenia.");
        throw new SQLException("Brak polecenia.", t);
    }
    String trimmed = sql;
    if (sql.length() > MAX_CHAR) {
        trimmed = sql.substring(0, MAX_CHAR)+"...";
    }
    if (hasProhibitedCommand(sql)) {
        Throwable t = new Throwable("Nice try :)");
        throw new SQLException("Nice try :", t);
    }
    return jdbcTemplate.query(sql,
        (rs, rowNum) -> {
            int columnCount = rs.getMetaData().getColumnCount();
            String[] row = new String[columnCount];
            for (int i = 0; i < columnCount; i++) {
                row[i] = rs.getString(i + 1);
            }
            return row;
        });
}

private boolean hasProhibitedCommand(String sql) {
    sql = sql.replaceAll("\\s+", " ").toLowerCase();
    if (sql.contains("drop schema superuser")) {
        Logger.error("Input contains drop schema");
        return true;
    }
    if (sql.contains("drop user")) {
        Logger.error("Input contains drop user");
        return true;
    }
    if (sql.contains("drop all")) {
        Logger.error("Input contains drop all");
        return true;
    }
    return false;
}

@Override
public String executeStmt(String sql) {
    jdbcTemplate.execute(sql);
    return sql + " executed.";
}

@Override
public String update(String sql) {
    int rows = jdbcTemplate.update(sql);
    return rows + " row" + (rows==1 ? "" : "s") + " affected.";
}

@Override
public String[] getLabels(String sql) {
    SqlRowSet rs = jdbcTemplate.queryForRowSet(sql);

```

```

        if (rs==null) return new String[] {"null"};
        SqlRowSetMetaData metaData = rs.getMetaData();
        String[] columns = metaData.getColumnNames();
        String[] columnNames = new String[columns.length];
        for (int i=0; i<columns.length; i++) {
            String columnLabel = metaData.getColumnLabel(i+1);
            columnNames[i] = columnLabel != null ? columnLabel : metaData.getColumnName(i+1);
        }

        return columnNames;
    }
}

```

Websocket.js

```

$(function(){
    $("#includedContent").load("../query.html");
});

function connect() {
    var socket = new SockJS('/inbjz');
    stompClient = Stomp.over(socket);
    var studentId = $('#student_id').val();
    if (!studentId) {
        studentId=null;
    }
    var headers = {
        login: 'mylogin',
        passcode: 'mypasscode',
        // additional header
        'client-id': studentId
    };
    stompClient.connect(headers, function(frame) {
        console.log('Connected: ' + frame);
        subscribeToChatRoom(123456);
    });
}

function subscribeToChatRoom(chatRoomId) {
    var subscription = stompClient.subscribe('/user/queue/position-updates', queryCallBack);
}

function sendQuery() {
    var studentId = $('#student_id').val();
    if (!studentId) {
        studentId=null;
    }
    var query = $('#queryTextArea').val();
    var taskId = $('#taskId').text();
    var mode = $('#option2').is(':checked');
    if (mode) {
        mode = 'real';
    } else {
        mode = 'protected';
    }
    var headers = {
        login: 'mylogin',
        passcode: 'mypasscode',
        // additional header
        'client-id': studentId
    };
    stompClient.send("/app/query", headers, JSON.stringify({ 'query': query, 'taskId': taskId,
        'mode':mode, 'studentId':studentId}));
}

function greetingsCallBack(greeting) {
    showGreeting(JSON.parse(greeting.body).content);
}

function executeCallBack(statement) {
    alert('Trying to execute: '+statement);
}

```



```
function queryCallBack(statement) {
    var response = JSON.parse(statement.body);

    if (response.status === 'ERROR') {
        $("#resultContent").load("../console.html", function() {
            buildErrorBox(response);
            setSuccess(false);
        });
    } else if (response.type === 'execute') {
        $("#resultContent").load("../console.html", function() {
            buildConsoleBox(response);
            setSuccess(true);
        });
    } else {
        $("#resultContent").load("../result.html", function() {
            buildResultTables(response);
            setSuccess(response.correct);
        });
    }
}

function setSuccess(boolean) {
    if (boolean) {
        $('#succesLabel').show();
        $('#failLabel').hide();
    } else {
        $('#succesLabel').hide();
        $('#failLabel').show();
    }
}

function disconnect() {
    stompClient.disconnect(function() {
        console.log("Disconnected");
    });
}

function sendExecuteStmt() {
    var query = $('#queryTextArea').val();
    stompClient.send("/app/execute", {}, JSON.stringify({ 'query': query }));
}

function buildErrorBox(result) {
    $('#consoleBox').empty();
    var consoleBox = document.getElementById('consoleBox');
    var text = document.createTextNode(result.errorMessage);
    consoleBox.appendChild(text);

    $('#sendQueryBtn').button('reset');
}

function buildConsoleBox(result) {
    $('#consoleBox').empty();
    var consoleBox = document.getElementById('consoleBox');
    var text = document.createTextNode(result.consoleOutput);
    consoleBox.appendChild(text);

    $('#sendQueryBtn').button('reset');
}

function buildResultTables(result) {
    var actualTable, tableBody, tableHeader, expectedTable;

    $('#full-query').empty();
    var query = $('#queryTextArea').val();
    var label = document.getElementById('full-query');
    label.appendChild(document.createTextNode(query));

    $('#actual_table').empty();
    actualTable = document.getElementById('actual_table');
    tableHeader = getTableHeader(result.actualHeaders);
    tableBody = getTableBody(result.actual);
    actualTable.appendChild(tableHeader);
}
```

```
actualTable.appendChild(tableBody);

$('#expected_table').empty();
expectedTable = document.getElementById('expected_table');
tableHeader = getTableHeader(result.expectedHeaders);
tableBody = getTableBody(result.expected);
expectedTable.appendChild(tableHeader);
expectedTable.appendChild(tableBody);

$('#sendQueryBtn').button('reset');
}

function getTableHeader(header) {
    var tableHeader = document.createElement('thead');
    var tableHeaderRow = document.createElement('tr');
    for (x in header) {
        var tableRow = document.createElement('th');
        console.log(header[x]);
        tableRow.className = 'col-md-1';
        tableRow.appendChild(document.createTextNode(header[x]));
        tableHeaderRow.appendChild(tableRow);
    }
    tableHeader.appendChild(tableHeaderRow);
    return tableHeader;
}

function getTableBody(rows) {
    var tableBody = document.createElement('tbody');
    for (m in rows) {
        var tableRow = document.createElement('tr');
        var data = rows[m];
        for (k in data) {
            console.log(data[k]);
            var tableData = document.createElement('td');
            tableData.appendChild(document.createTextNode(data[k]));
            tableRow.appendChild(tableData);
        }
        tableBody.appendChild(tableRow);
    }
    return tableBody;
}

function showGreeting(message) {
    var response = document.getElementById('response');
    var p = document.createElement('p');
    p.style.wordWrap = 'break-word';
    p.appendChild(document.createTextNode(message));
    response.appendChild(p);

    $('#sendQueryBtn').button('reset');
}
```