POLITECHNIKA ŁÓDZKA

INSTYTUT INFORMATYKI

PRACA DYPLOMOWA INŻYNIERSKA

Interaktywne nauczanie bazodanowych języków zapytań

Autor:
Łukasz Ochmański
Nr albumu: 183566

Promotor:
dr. Inż Krzysztof Myszkorowski

Łódź, 8 luty 2015

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

1.1. Wstęp

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.

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 – pierwszym silniku bazodanowym opartym o model relacyjny (jednak pierwszy komercyjny system RDBSM to wdrożenie firmy ORACLE w 1979 r.).

Jak sama nazwa wskazuje, SEQUEL 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.

Nazwa ewoluowała – 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.

Standardy, dialekty

Konkurencyjność rynku spowodowała konieczność ustandaryzowania języka SQL i na szczęście stało się to już w roku 1986, kiedy został opracowany przez ANSI pierwszy standard określany jako SQL:86. Podkreślam, że na szczęście tak szybko, bo choć istnieją istotne różnice np. w nazwach implementowanych funkcji, to ogólne zasady dla relacyjnych baz danych, różnych

producentów, są 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, jeśli poznasz T-SQL, odnajdziesz się szybko 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.

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).

1.2. Cel projektu

Celem głównym pracy jest zaprojektowanie i zbudowanie aplikacji internetowej przy użyciu dostępnych technologii typu open source w celu nauczania bazodanowych języków zapytań w standardzie SQL-92. Natomiast celem stworzenia aplikacji jest wsparcie studentów w opanowaniu biegłego posługiwania się językiem SQL, któgo 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. Opis struktury pracy

1.3.1. Rodział 1

Wstęp

W tym rozdziale jest opisana geneza powstania relacyjnych baz danych i języków zapytań.

Cel projektu

W tym rozdziale znajduje się charakterystyka ogólny opis prezentowanego projektu.

Opis struktury pracy

Skrócone zestawienie wszystkich rozdziałów.

1.3.2. Rozdział 2

Założenia dotyczące projektu

Założenia typu, 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 sie przyjrzeć i rozwiązać. Na tym będzie sie skupiać praca.

Dostęp do projektu

Ogólne informacje dotyczące lokalizacji aplikacji, sposobów dostępu, kod źródłowego, wymaganych uprawnień.

Wymagania sprzetowe

Specyfikacja wymagań technicznych, niezbędnych do uruchomienia aplikacji.

1.3.3. Rozdział 3

Architektura systemu

Ogólny zarys budowy aplikacji. Model graficzny.

Warstwa prezentacji (front-end)

Ogólny zarys, zasada działania i funkcje pełniące przez warstwę prezentacji.

Warstwa dostępu do danych (back-end)

Ogólny zarys, zasada działania i funkcje pełniące przez warstwę dostępu do danych.

Warstwa danych

Opis silnika bazodanowego, możliwości, trybów pracy, protokołów komunikacji i sposobów integracji z klientami. Utworzone instancje, schematy i użytkownicy.

Techniki, wzorce architektoniczne, wzorce programistyczne

Opis zastosowanych technik i wzorców projektowych (design patterns).

1.3.4. Rozdział 4

Schemat działania

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

Dokumentacja dla programistów chcących rozwijać istniejącą aplikację i dla osbób chcących korzystać z dostępnych metod API (Application Programming Inteface).

1.3.5. Rozdział 5

Przykłady działania aplikacji

Graficzna prezentacja, pokaz możliwości aplikacji.

Praca zespołowa

Prezentacja modułu do pracy grupowej.

1.3.6. Rozdział 6

Bezpieczeństwo

Poruszenie tematów związanych z separacją danych, uprawnień użytkowników, dostępu do ocen prze niepowołane osoby.

Dostępność i niezawodność

Tutaj zostaną poruszone kwestie dostępności do systemu w czasie i miejscu. Odporność na ataki, awarie, wirusy, zawieszenia, wyjątki, błędy. Akcja przedsięwzięte w celu uniknięcia problemów.

Analiza wydajności

Porównanie dostępnych na rynku baz oraz porównanie dostępnych protokołów, które mogą zastąpić isniejące.

1.3.7. Rozdział 7

Perspektywy rozwoju

Lista planowanych funkcjonalności

Wnioski i podsumowanie

Wnioski i podsumowanie.

1.3.8. Rozdział 8

Spis rysynków.

Spis tabel.

Spis listingów

Bibliografia

Załaczniki

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,
- W poniższej pracy będę używał akronimu INBJZ jako skrót od „Interaktywne nauczanie bazodanowych języków zapytań".
- INBJZ jest aplikacją typu open source i można korzystać z kodu źródłowego w celach niekomercyjnych.
- Użyte przeze mnie bilbioteki 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.
- Podzas korzystania z aplikacji należy być ostrożnym, gdyż na pewno istnieje wiele możliwośći „złamania" systemu. Starałem 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 w bazie jest logowana (zakończona powodzeniem) i widoczna przez administratora systemu.
- Wszystkie próby (skuteczne i nieskuteczne) udzielenia odpowiedzi 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 cudzysłowów.
- 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 mna 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 interaktywne reagowanie na działania studentów.

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. Należało by 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 osoby, a efekt końcowy powinień być wspólny dla wszystkich, którzy ukończą wszsytkie 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 poufność 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 możliwości wyścigu oraz próby modyfikacji wspólnych

zasobów, gdyż wszyscy użytkownicy dzielą jedna instancję bazy danych. 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://5.135.146.42:8080/** .

Godziny dostepu

W planach system będzie dostępny całą dobę. Jednak w przypadku wystąpienia błędów wymagany będzie restart, co spowoduje unieruchomienie aplikacji na nie więcej niż 5 minut.

Docelowa grupa odbiorców

W początkowej fazie wdrożenia nie będzie ograniczeń co do osób korzystających z serwisu. Każdy może anonimowo korzystać z 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://ochman2000@code.google.com/p/interaktywne-nauczanie-jezykow-bazodanowych/

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 zaimiportować 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/interaktywne-nauczanie-jezykow-bazodanowych/

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 rodzielczość: 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 aplikacj 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.

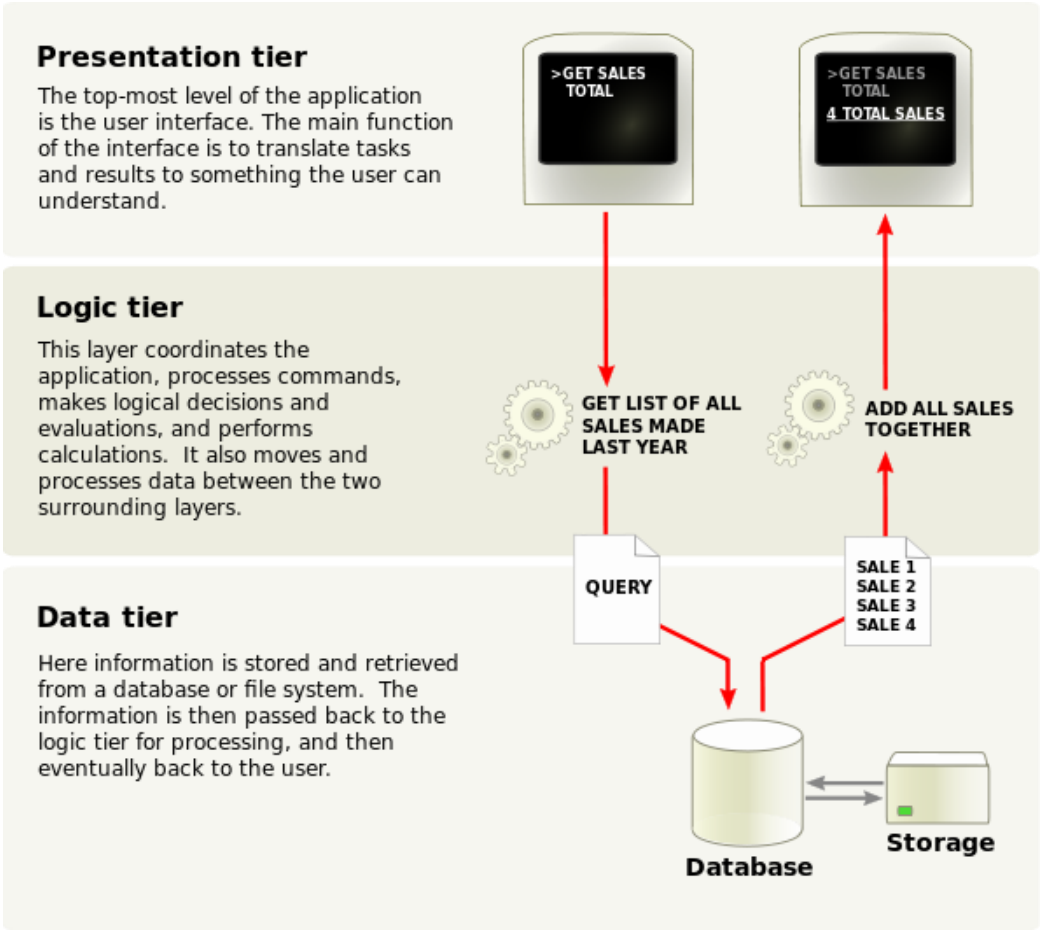
WebSocket

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, dzwięku 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 efyktywność 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

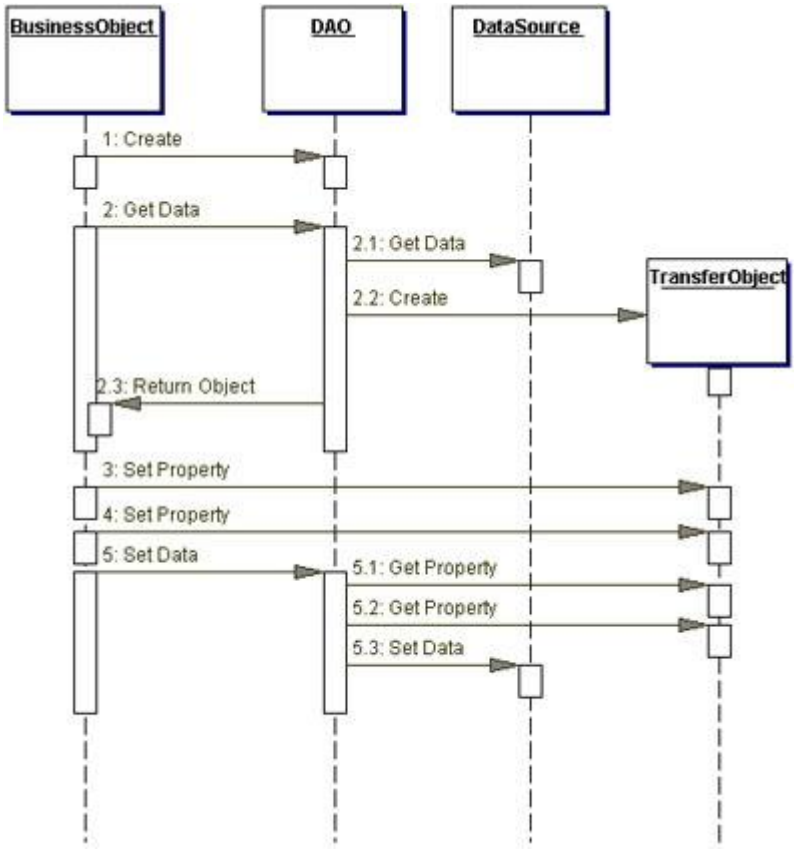
Cała architektura projektu opiera się o klasyczny model trójwarstowy (Three-Tier-Architecture), gdzie moduły systemu zostały oddzielone i uniezależnione od siebie. Każdy z modułów jest samodzielny i może w każdej chwili zostać wymieniony. Poniższy rysynek prezentuje zastosowany model:



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 dostarczać dane w taki sam sposób jak wcześniej. Jedyna różnica będzie polegać na efektywności i skutkach działania programu. Ważne jest, aby zaplanować za wczasu metody i klasy, które mogą ulec zmianie. Wówczas zmiany w kodzie będą minimalne. Diagram interakcji przedstawiony na rysunku poniżej:



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

W przypadku mojej aplikacji isnieje bardzo duże prawdopodobieństwo, że baza danych zostanie wymieniona, gdyż H2 jest niewystarczająca do wielu rozwiązań. Z tego powodu w moim kodzie zostały wydzielony interfejs 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 inerfejsu 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.

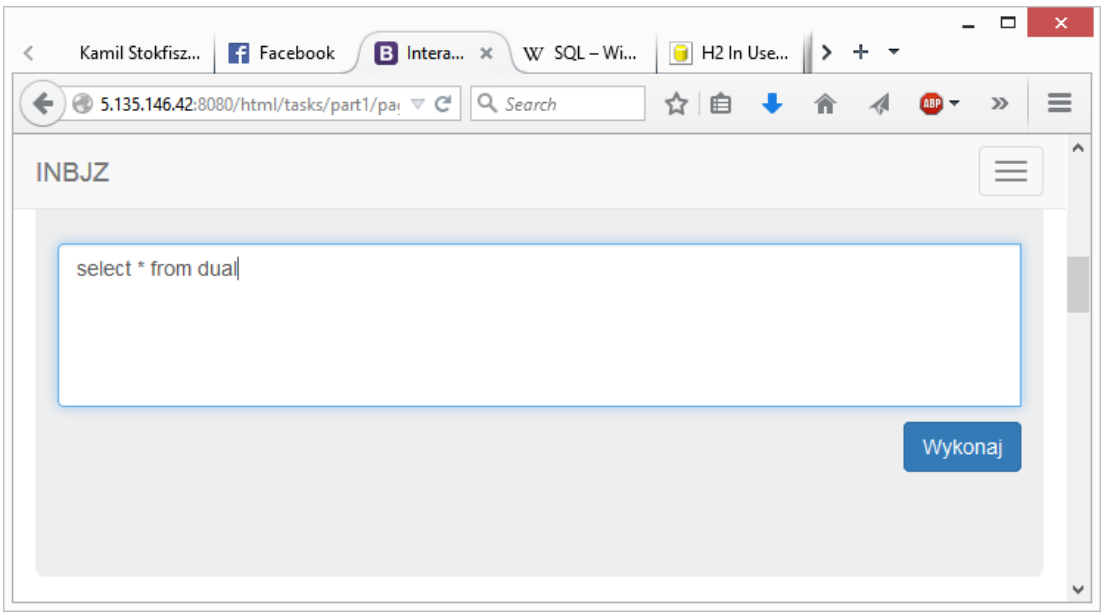
```
public interface TaskRepository {

    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 żadanie, a następnie odpowiada przeglądarce. Ta z kolei wyświetla żadany rezultat.

3.2.1. HTML 5



HTML to jezyk znaczników, który pozwala opisać strukture strony internetowej. HTML przeszedł przez wiele wersji. Poczatkowo 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 jezyk SGML (ang. Standard Generalized Markup Language). Kolejne wersje na przestrzeni 5 lat były ciągle zmieniane aż do 1997 roku i wersji HTML 4.0. Jedna z wersji, która najdłużej wyznaczała standard w sieci była specyfikacja jezyka HTML 4.0. Od tamtej pory pojawiło sie także kilka odmian HTML 4.0 takich jak np. XHTML 1.0. Wprowadzenie HTML5 zajeło duzo czasu, że wzgledu na dopracowanie HTML5 do poziomu, który mógłby być akceptowalnym standardem używanym przez wiele rodzajów urządzeń. Wersja HTML5 został zatwierdzony 3 miesiace temu w pazdzierniku 2014 roku. HTML5 zawiera w obecnej specyfikacji [W3C ref] 89 znaczników. Jedna z głównych zalet jezyka, jest niezależnosc od systemu operacyjnego oraz sprzetu komputerowego. Pozwala to uruchamiac aplikacje HTML na szerokiej gamie urzadzen 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.

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.

Wraz z popularyzacją technologii AJAX, a także bogatych, interaktywnych interfejsów aplikacji internetowych, objętość kodu drastycznie wzrosła. Posługiwanie się nim 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 znajduję 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. WebSocket

WebSockets służy do nawiązywania połączenia dwukierunkowego przez TCP. Protokół ten został ustandaryzowany przez organizacje IETF (ang. Internet Engineering Task Force) w 2011 roku. Web Socket moze byc zastosowany wewnatrz 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 jest to Apache Tomcat 8, który już od wersji 7 posiadał wbudowane wsparcie.

Aby ustanowic połaczenie z poziomu przegladarki internetowej w aplikacji web'owej musimy wykonać jedna czynność przez HTTP, którą jest 'handshake'. Handshake wykonywany jest na porcie 80, aby serwer HTTP mógł go przetworzyc. Przykładowa próba połaczenia widnieje ponizej.

Zapytanie klienta przez HTTP:

```
GET / chat HTTP /1.1
Host : server . example . com
Upgrade : websocket
Connection : Upgrade
Sec - WebSocket - Key : x3JJHMbDL1EzLkh9GBhXDw ==
Sec - WebSocket - Protocol : chat , superchat
Sec - WebSocket - Version : 13
Origin : http :// example . com
```

Odpowiedz serwera przez HTTP, zawierajaca pole „Sec-WebSocket-Accept":

```
HTTP /1.1 101 Switching Protocols
Upgrade : websocket
Connection : Upgrade
Sec - WebSocket - Accept : HSmrc0sMlYUkAGmm5OPpG2HaGWk =
Sec - WebSocket - Protocol : chat
```

W zapytaniu klienta pole klucza 'Sec-WebSocket-Key' to losowa liczba enkodowana skrótem base64. W odpowiedzi na zapytanie serwer dokonuje konkatenacji klucza z unikalnym polem GUID. Nastepnie wartosc jest zakodowana algorytmem skrótu SHA-1, ponownie enkodowana

przez base64 i zwracana w odpowiedzi w polu 'Sec-WebSocket-Accept'. Połaczenie na tym etapie jest ustanowione.

Dane moga byc wymieniane w ramkach z niewielkim narzutem na nagłówek i payload. Dlugosc wysyłanych danych moze byc dowolna, az do odczytania koncowego bitu FIN. Protokół uzywa prefixów ws:// oraz wss:// - oznaczaja one połaczenie Web Socket oraz Web Socket Secure, które sa odpowiednikami HTTP oraz HTTPS, czyli połaczenia nieszyfrowanego oraz szyfrowanego.

3.2.5. STOMP

STOMP is the Simple (or Streaming) Text Orientated Messaging Protocol.

STOMP provides an interoperable wire format so that STOMP clients can communicate with any STOMP message broker to provide easy and widespread messaging interoperability among many languages, platforms and brokers.

STOMP is a very simple and easy to implement protocol, coming from the HTTP school of design; the server side may be hard to implement well, but it is very easy to write a client to get yourself connected. For example you can use Telnet to login to any STOMP broker and interact with it!

Many developers have told us that they have managed to write a STOMP client in a couple of hours to in their particular language, runtime or platform into the STOMP network. So if your favored language/runtime of choice does not offer a good enough STOMP client don't be afraid to write one.

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.

JSON jest jednym z nieformalnych sposobów przekazywania danych do aplikacji opartych o AJAX. 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.

CSS przygotowany przez autorów frameworka obejmuje większość znaczników HTML. Cześć 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 naprawdę wielu różnych problemach które spadają na webdeveloperów i dostarczyli nam klasy dzięki którym stworzymy praktycznie wszystko, począwszy 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 wszelakie 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.

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 z dwóch powodów:
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 najważniejszych 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. 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. 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. 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 podstawoweg, wymaganego w skomplikowanych aplikacjach.

W ostatnich dziewięci latach zakres tej biblioteki znacznie się zwiększył, ale pomimo tego pozostałą prosta i łatwa w użyciu. Obecnie składa się ona z koło 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 40 mikroprojektów dostępnych na oficialnje stronie Spring. Są one bardzo wygodne dla osób rozpoczynających pracę ze Springiem. Gotowe pakiety pozwalają na uruchomienie aplikacji w mniej niż 15 minut. Mikroprojekty Spring Boot nadają się świetnie do integracji z isniejącymi 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 10 minut, aby znaleźć bibliotekę, zainstalować i uruchomić aplikację komunikującą się z bazą H2 poprzez JDBC.

3.3.8. Gradle

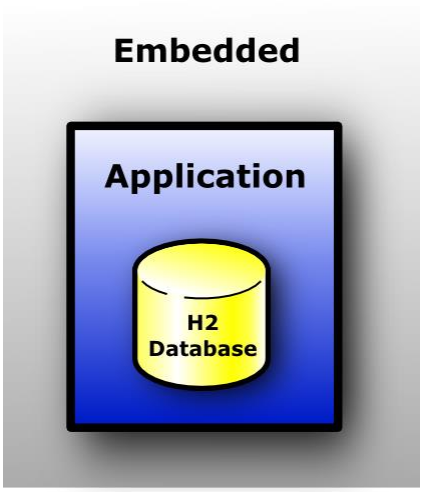
Gradle jest potężnym narzędziem do budowy projektów. Gradle łączy w sobie cechy innych narzędzi takich jak Ant oraz Maven, eliminując przy tym nadmiarowość kodu jaki należało napisać podczas konfiguracji projektu. Do tej pory trudność sprawiało zaimplementowanie jakiegokolwiek algorytmu w skrypcie budującym; trudno było wyrazić nawet tak podstawowe konstrukcję jak if czy for, ciężko było wykonać nawet pozornie proste taski, których wykonywanie nie zostało przewidziane przez developerów Anta/Mavena, które nie są dostępne w postaci gotowych tasków lub pluginów. Potrzebna jest dogłębna znajomość tych narzędzi, by zrozumieć złożony skrypt budujący. Poza tym podejście "build by convention" nie jest wspierane (Ant), albo skutecznie utrudnia konfigurację (Maven). Ponadto wsparcie dla budowy projektów wielomodułowych jest niewystarczające, skrypty budujące są niepotrzebnie duże i nieczytelne z uwagi na narzut samego języka (XML).

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

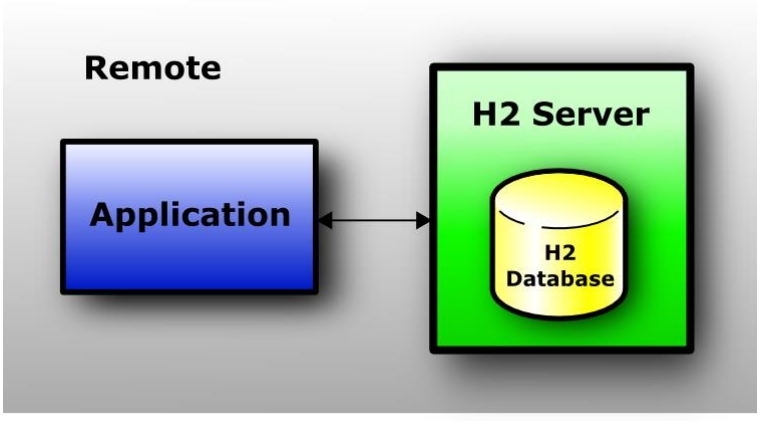
Jak już wcześniej wspomniałem w projekcie użyto bazy danych H2. Technologia została wybrana ponieważ zapewnia najszybszy dostęp do aplikacji, które działają na tej samej instancji wirtualnej maszyny Java JVM. Zgodnie ze specyfikacją producenta H2 może działać w trzech trybach:

- Wbudowany (embedded)



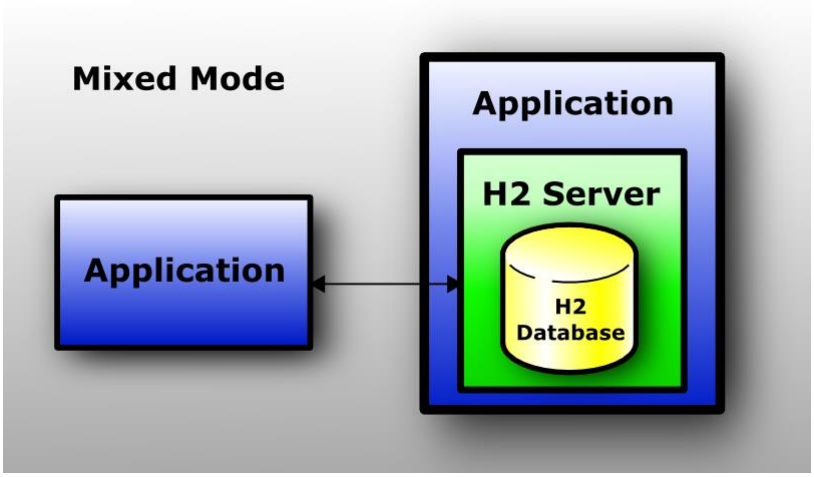
Ten tryb pozwala na najszybszy dostęp do danych, gdyż cała baza może być przechowywana w pamięci operacyjnej.

- Serwer



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

- Mixed



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 poźniejszym 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 sie 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 tej 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 dokonac 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 zestawić je z wynikami osiąganymi przez obecnych. W ten sposób powstanie bardzo ciekawa statysyka, która może stać się cenną informacją dla prowadzącego zajęcia.

3.5. Techniki, wzorce architektoniczne oraz programistyczne

3.5.1. DAO

3.5.2. MVC

3.5.3. Subscribe

3.5.4. Listener

3.5.5. Singleton

3.5.6. Wyrażenia Lambda

Pozi

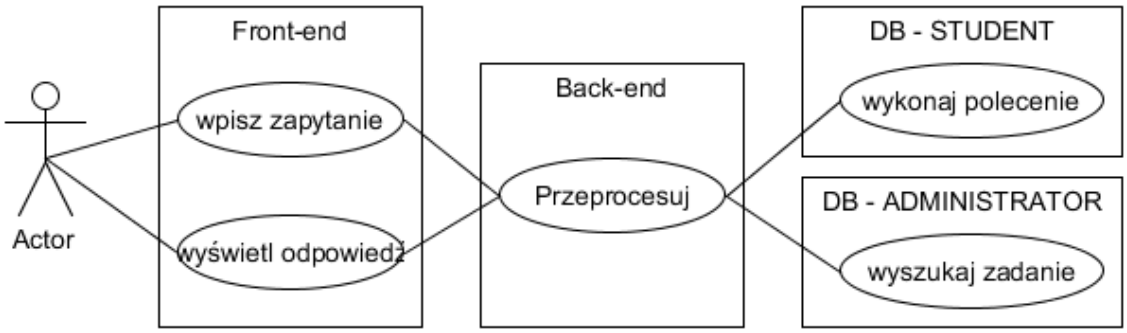
Rozdział 4

4.1. Schemat działania

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

4.1.1. Przykładowy workflow

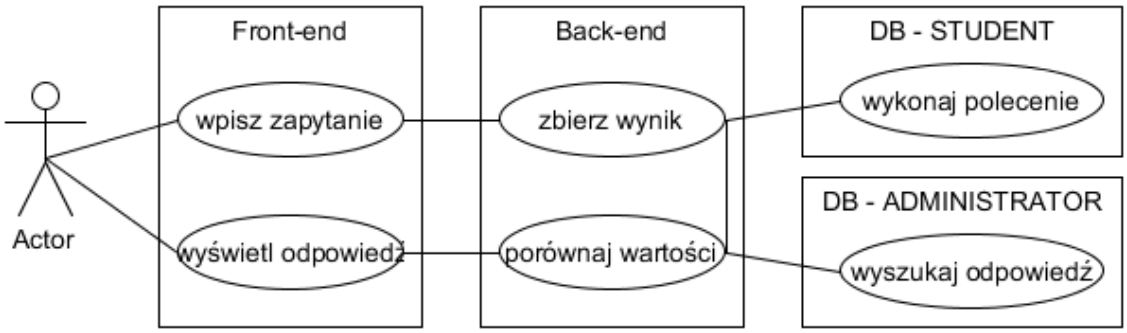
Poniżsy rysunek przedstawia ogólny sposób przypływu informacji pomiędzy modułami systemu.



Rysynek 4.1 Diagram przypadków użycia – Przykładowy workfolow

4.1.2. Weryfikacja odpowiedzi

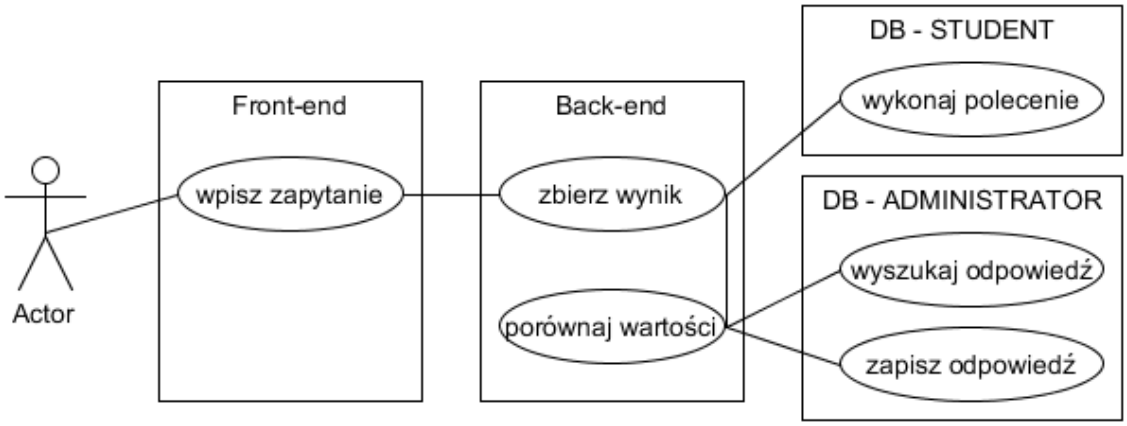
Poniższy rysunek przedstawia sposób weryfikacji poprawności udzielonej odpowiedzi.



Rysunek 4.2 Diagram przypadków użycia – Weryfikacja odpowiedzi

4.1.3. Przyznanie punktów

Poniższy rysunek przedstawia sposób przyznawania punktów za udzielenie poprawnej odpowiedzi.



Rysunek 4.3 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 wstawania 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 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 prowadzacego 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 egazminów, zmianach sal, odwołanych spotkaniach, przesuniętych godzinach konsultacji wystarczy zajrzeć do zakłaki 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. Szczegółowy opis implementacji

W tej sekcji zamierzam opisać i przeanalizować fragmenty kodu aplikacji.

Dwa główne controllery: służące do komunikacji z Front-endem:

```
@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 InbjzResultSet 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 InbjzResultSet greeting(Request message) throws Exception {
        return queryService.greeting(message);
    }

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

    @MessageMapping("/teamQuery")
    @SendTo("/topic/query")
    public InbjzResultSet 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;
    }

    @MessageMapping("/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();
    }

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

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

    @MessageMapping("/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 occured. 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 = "2lkj2lkj";
            String sha512 = "ACE1533DF199C233735FA02F4AC80410F49D3DD" +

"5CBEDEF4D1DDCFC972ACE8BF84F099B9374B50542ADFF6AE211ED839E703F24E7C2298B6A33E42DD4
FE8A5A97";
            dataSource.setPassword(sha512);
        }
        if (user==User.STUDENT) {
            dataSource.setUsername("STUDENT");
            dataSource.setPassword("abc");
        }
        return dataSource;
    }

    public static DatabaseAdmImpl getInstance(User user) {
        if (User.SA==user) {
            return instanceSA = instanceSA == null ? new DatabaseAdmImpl(user) :
instanceSA;
        } else {
            return instanceStudent = 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 contatins drop schema");
            return true;
        }
        if (sql.contains("drop user")) {
            Logger.error("Input contatins drop user");
            return true;
        }
        if (sql.contains("drop all")) {
            Logger.error("Input contatins 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 = "2lkj2lkj";
            String sha512 = "ACE1533DF199C233735FA02F4AC80410F49D3DD" +

"5CBEDEF4D1DDCFC972ACE8BF84F099B9374B50542ADFF6AE211ED839E703F24E7C2298B6A33E42DD4
FE8A5A97";
            dataSource.setPassword(sha512);
        }
        if (user==User.STUDENT) {
            dataSource.setUsername("STUDENT");
            dataSource.setPassword("abc");
        }
        return dataSource;
    }

    public static DatabaseStudImpl getInstance() {
        return instance = 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)+"...";
        }
//        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 contatins drop schema");
            return true;
        }
        if (sql.contains("drop user")) {
            Logger.error("Input contatins drop user");
            return true;
        }
        if (sql.contains("drop all")) {
            Logger.error("Input contatins 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;
    }
}
```

4.4. Dokumentajca techniczna

API

Pozi

Rozdział 5

5.1. Przykłady działania aplikacji oraz dokumentacja użytkownika

Okablowanie Poziome

5.2. Praca zespołowa

Pozi

Rozdział 6

Rozdział 7

7.1. Perspektywy rozwoju

Moduł logowania

Resetowanie haseł

Przypominanie zapomnianych haseł

Single Sign-On, Kerberos, LDAP, Active Directory

Blokowanie dostępu

Czarna lista

Moduł oceniania

Wykresy ocen, SVG

Rozkład normalny, SVG

Statystyka grupy, roku

Archiwum ocen

Wykrywanie plagiatu

Rozbudowa panelu administracyjnego

Automatyczne dodawanie pytań

Modyfikacja kont użytkowników

Analiza logów

Personalizacja wyglądu

Integracja z platformą Moodle

Interaktywność

Zapamiętywanie wpisywanych zapytań

Podpowiedzi składni w czasie rzeczywistym

Podpowiadanie nazw obiektów

Analiza stylu i składni

Angielska wersja językowa.

7.2. Wnioski i podsumowanie

Okablowanie Poziome

Pozi

Spis rysynków.

Spis tabel.

Spis listingów

Bibliografia

[1] Jakub Kasprzak
Język SQL – historia, standardy
Styczeń 2013
URL http://www.sqlpedia.pl/jezyk-sql-historia-standardy/

[2] Wikipedia

http://pl.wikipedia.org/wiki/SQL

[3] http://www.h2database.com/html/links.html#projects

[4] Core J2EE Patterns - Data Access Object

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

[5] Mutli-their-architecture

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

[6] Marijn Haverbeke

JavaScript I wszystko jasne

http://www.bt4.pl/kursy/javascript/wszystko-jasne

[7] Kurs jQuery. Część 1: Wprowadzenie. Zalety, podstawowe zasady, pierwszy skrypt i rozszerzenia
http://webhosting.pl/Kurs.jQuery.Czesc.1.Wprowadzenie.Zalety.podstawowe.zasady.pierwszy.skrypt.i.rozszerzenia

[8] STOMP The Simple Text Oriented Messaging Protocol

https://stomp.github.io/

[9] JSON, JavaScript Object Notation

http://pl.wikipedia.org/wiki/JSON

[10] Bootstrap

http://www.altcontroldelete.pl/artykuly/jak-szybko-stworzyc-ladny-css-z-frameworkiem-bootstrap/

[11] Bartosz Danowski Wstęp do CSS

http://webmaster.helion.pl/index.php/pcss-wstep

[12] Cezary Bronny Technologia JDBC w praktyce

http://students.mimuw.edu.pl/~zbyszek/bazy-danych/JDBC-CB227567.pdf

[13] W. Wheeler, J. White, Spring w praktyce, Manning Publication 2014

Załaczniki