Software Architecture Document

wersja 2.0-final

Marcin Mieteń Maciej Szarliński

studenci IV roku infromatyki

Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego

Projektowanie Obiektowych Systemów Informatycznych 2009/10

19 stycznia 2010

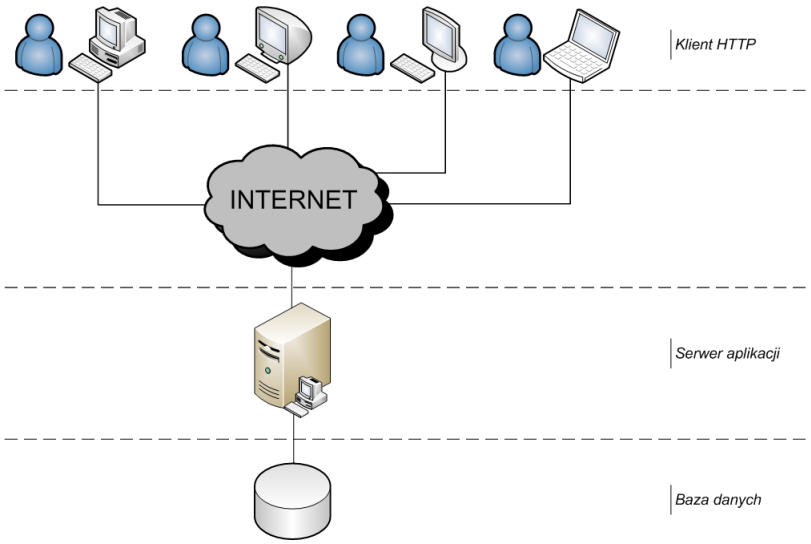
Spis treści

1 Architektura ogólna systemu

System będzie oparty o architekturę *thin-client* – cała logika aplikacji będzie zawarta w części serwerowej. Użytkownicy będą korzystali z usług serwera za pośrednictwem przeglądarki internetowej (protokół HTTP lub HTTPS). Rozwiązanie powinno zapewnić:

- dostęp do systemu niezależnie od systemu operacyjnego i bez konieczności instalacji oprogramowania po stronie użytkownika,
- trwałość i bezpieczeństwo danych,
- wielodostęp (do 1000 użytkowników jednocześnie przy czasie odpowiedzi co najwyżej 5s).

Ogólny diagram architektury systemu przedstawiony jest poniżej:



Preferowane technologie mające wpływ na architekturę fizyczną:

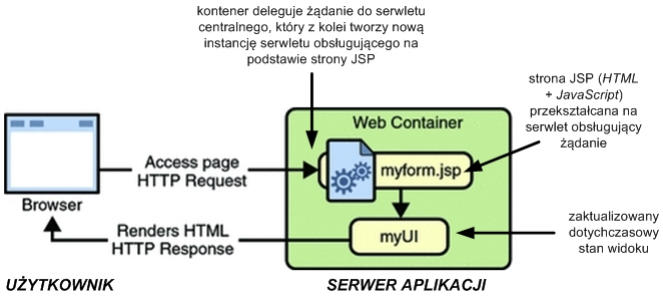
- system zarządzania bazą danych: *PostgreSQL*,
- serwer aplikacji: *Glassfish* lub inny zgodny ze specyfikacją *Java EE 5* (wystarczy kontener serwletów zgodny z *Java Servlet 2.5*),
- warstwa prezentacji: framework *JavaServer Faces 1.2* rozszerzony o AJAX (biblioteki *IceFaces*).

1.1 Obsługa żądań do serwera i prezentacja danych

1.1.1 Komunikacja z serwerem w JavaServer Faces

JavaServer Faces jest technologią wchodzącą w skład standardu Java Enterprise Edition, upraszczającą obsługę żądań HTTP po stronie serwera. Framework wzbogaca aplikacje internetowe o możliwość przechowywania stanu komponentów po stronie serwera (walidacja, konwersja i utrzymywanie sesji) oraz pozwala na odseparowanie warstwy widoku (strony JSP wzbogacone o dodatkowe biblioteki tagów) od logiki aplikacji. Dzięki temu programiści mogą pracować niezależnie nad różnymi częściami aplikacji.

Poniższy diagram prezentuje uproszczony model komunikacji użytkownika z aplikacją działającą na serwerze za pośrednictwem przeglądarki internetowej:



1.1.2 Żądania asynchroniczne i „bogate" komponenty

Technologia JavaServer Faces wymaga, aby każde żądanie do serwera było wynikiem akcji użytkownika (np. zatwierdzenie formularza przyciskiem), co uniemożliwia na przykład sprawdzanie poprawności danych już podczas ich wprowadzania. Technologia JavaServer Faces nie pozwala na:

- sprawdzać poprawności wprowadzanych danych już podczas wypełniania formularzy, przez co w przypadku przesyłania dużej ilości danych obsługa błędów jest uciążliwa,
- wyświetlać powiadomień w dowolnym momencie,
- budować stron internetowych w oparciu o komponenty znane z aplikacji stacjonarnych (Rich Internet Application).

Wszystkie te problemy adresuje framework IceFaces, który jest rozszerzeniem mechanizmu JavaServer Faces o komunikację asynchroniczną i bibliotekę „bogatych" komponentów. Do obsługi AJAX push przez serwer Glassfish wymagana jest darmowa wtyczka Grizzly.

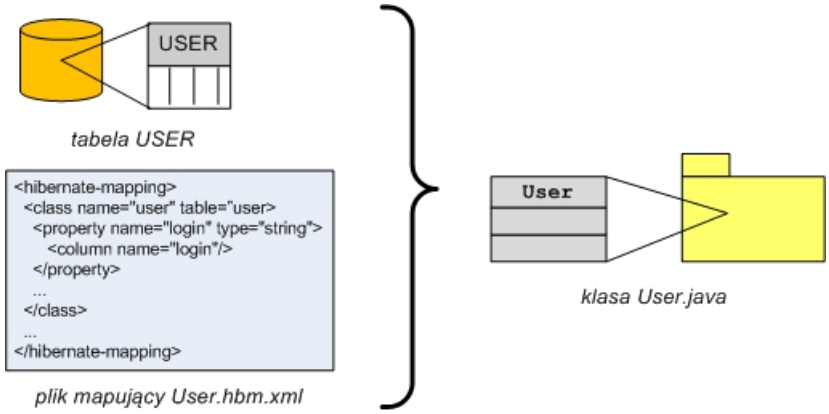
1.1.3 Leniwa inicjalizacja obiektów

Aby zapobiec wczytywaniu wielu danych z bazy danych przy każdym zapytaniu, zastosujemy mechanizm leniwej inicjalizacji asocjacji i kolekcji. Rozwinięcie kolekcji dopiero w widoku będzie możliwe dzięki zastosowaniu wzorca *Open session in view*, którego implementację dostarcza framework Spring.

1.2 Trwałość danych i transakcyjność

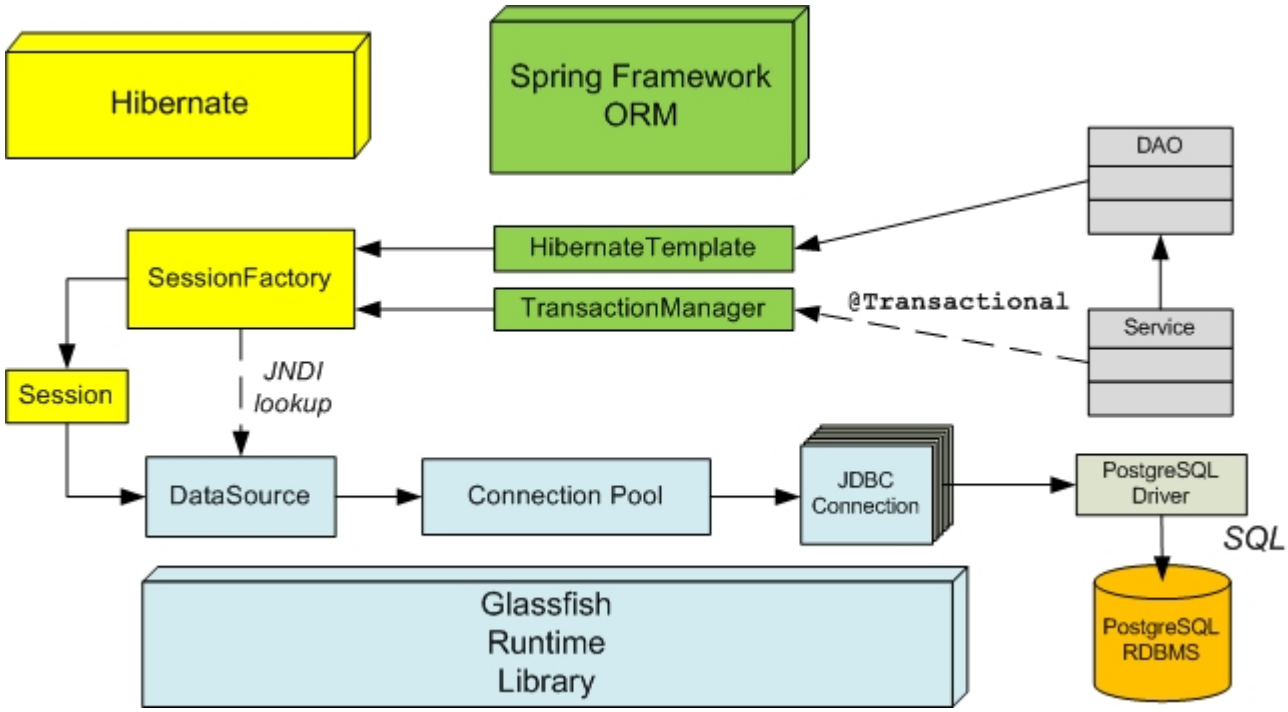
1.2.1 Mapowanie obiektowo-relacyjne

Każda encja ma swoje odzwierciedlenie w systemie w postaci klasy zgodnej ze standardem JavaBeans. Mechanizm mapowania będzie dostarczony przez framework Hibernate:

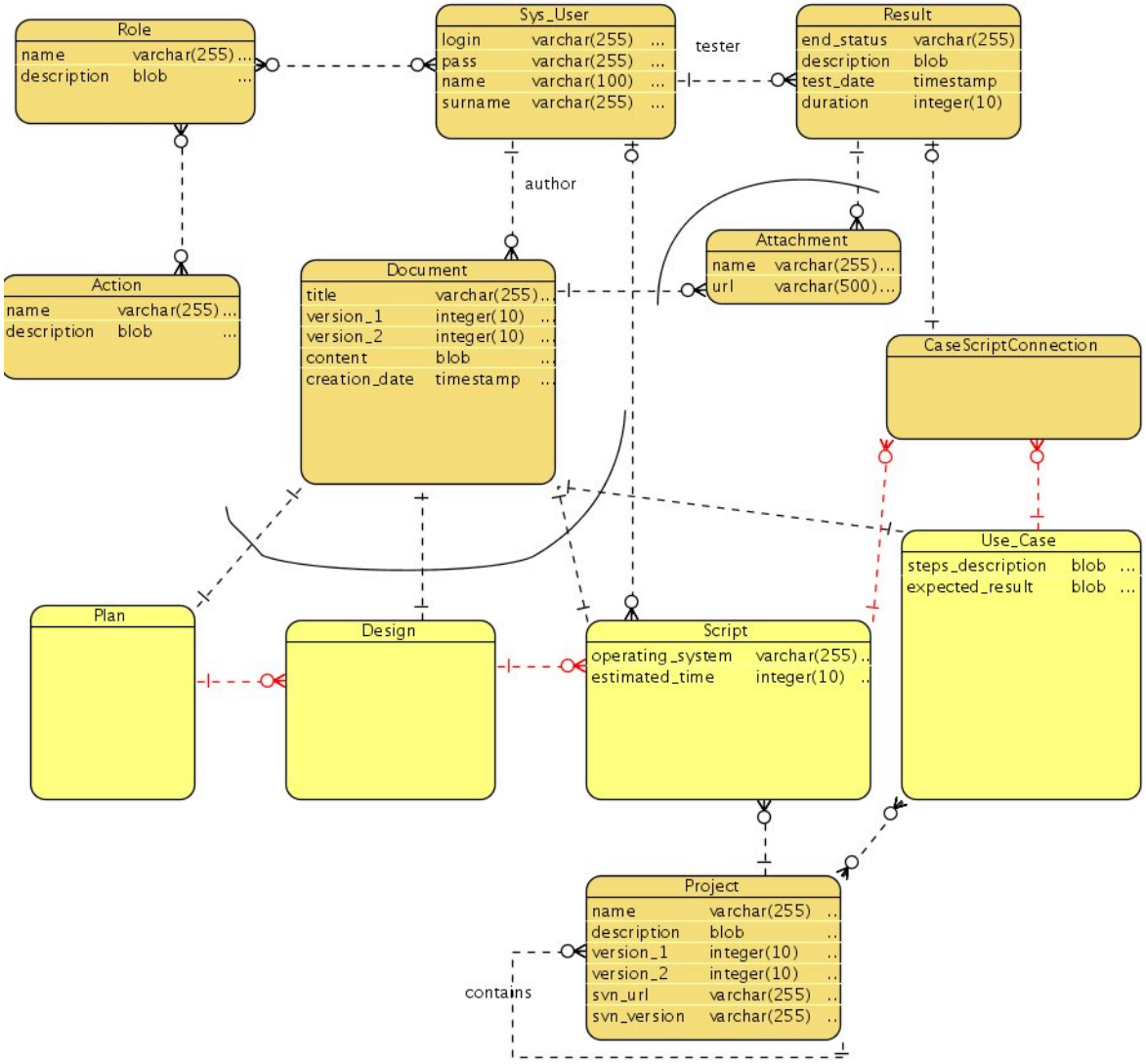


1.2.2 Zarządzanie połączeniami i transakcjami

Odpowiedzialność za utrzymywanie puli połączeń do bazy danych oraz synchronizację transakcji przeniesiemy na serwer aplikacji. Z poziomu aplikacji baza danych będzie widoczna jako zasób *DataSource*, który będzie dostępny poprzez interfejs JNDI. Demarkacja transakcji zostanie zdefiniowana za pomocą interfejsu dostarczanego przez Spring Framework.


Hibernate
Spring Framework
ORM
DAO
HibernateTemplate
SessionFactory
TransactionManager
@Transactional
Service
Session
JNDI
lookup
DataSource
Connection Pool
JDBC
Connection
PostgreSQL
Driver
SQL
Glassfish
Runtime
Library
PostgreSQL
RDBMS

2 Diagram Encji



2.1 Sys_User

Encja odpowiadająca użytkownikowi systemu.

2.2 Role

Encja odpowiadająca roli użytkownika w systemie np. tester, projektant testów, kierownik testów.

2.3 Action

Encja odpowiadająca pojedynczej aktywności związanej z konkretną akcją w systemie. Każda rola (tester, projektant testów, kierownik testów itp.) będzie posiadała zbiór akcji, do których będzie miała prawa dostępu.

2.4 Document

Encja odpowiadająca abstrakcyjnemu dokumentowi związanemu z procesem testowania oprogramowania. Każdy taki dokument posiada atrybuty:

- name – nazwa dokumentu
- content – główna zawartość dokumentu (tekst opisujący wszystkie kwestie związane z typem dokumentu)
- version_1 – główny nr wersji dokumentu
- version_2 – nr dla podwersji dla danej wersji głównej.
- creation_date – data i czas utworzenia dokumentu.

2.5 Plan

Encja odpowiadająca dokumentowi planu testów.

2.6 Design

Encja odpowiadająca dokumentowi projektu testów.

2.7 Script

Encja odpowiadająca skryptowi testów. Każdemu skryptowi można przyporządkować listę pojedynczych testów (Use_Case), projekt, testera, który będzie zobowiązany do przeprowadzenia wszystkich testów powiązanych ze skryptem. Dodatkowo można zdefiniować system operacyjny, na którym będą przeprowadzane testy i szacowany czas trwania przeprowadzenia testów skryptu.

2.8 Use_Case

Encja odpowiadająca pojedynczemu testowi. Istotnymi atrybutami są:

- steps_description – dokładny opis krok po kroku czynności, które powinien wykonać tester.
- expected_result – ostateczny oczekiwany wynik po wykonaniu wszystkich kroków opisanych w steps_description.

Każdy case tyczy się wykonania testu na konkretnym obszarze projektu. Każdy projekt dzieli się na podprojekty, z których możemy wybrać te których dotyczy dany case.

2.9 CaseScriptConnection

Encja odpowiadająca połączeniu wiele do wielu między encją Case i Script. W kontekście takiego połączenia tworzone są wyniki danego testu (Case).

2.10 Project

Encja odpowiadająca projektowi, który jest przedmiotem testów. Każdy projekt może posiadać podprojekty.

2.11 Attachment

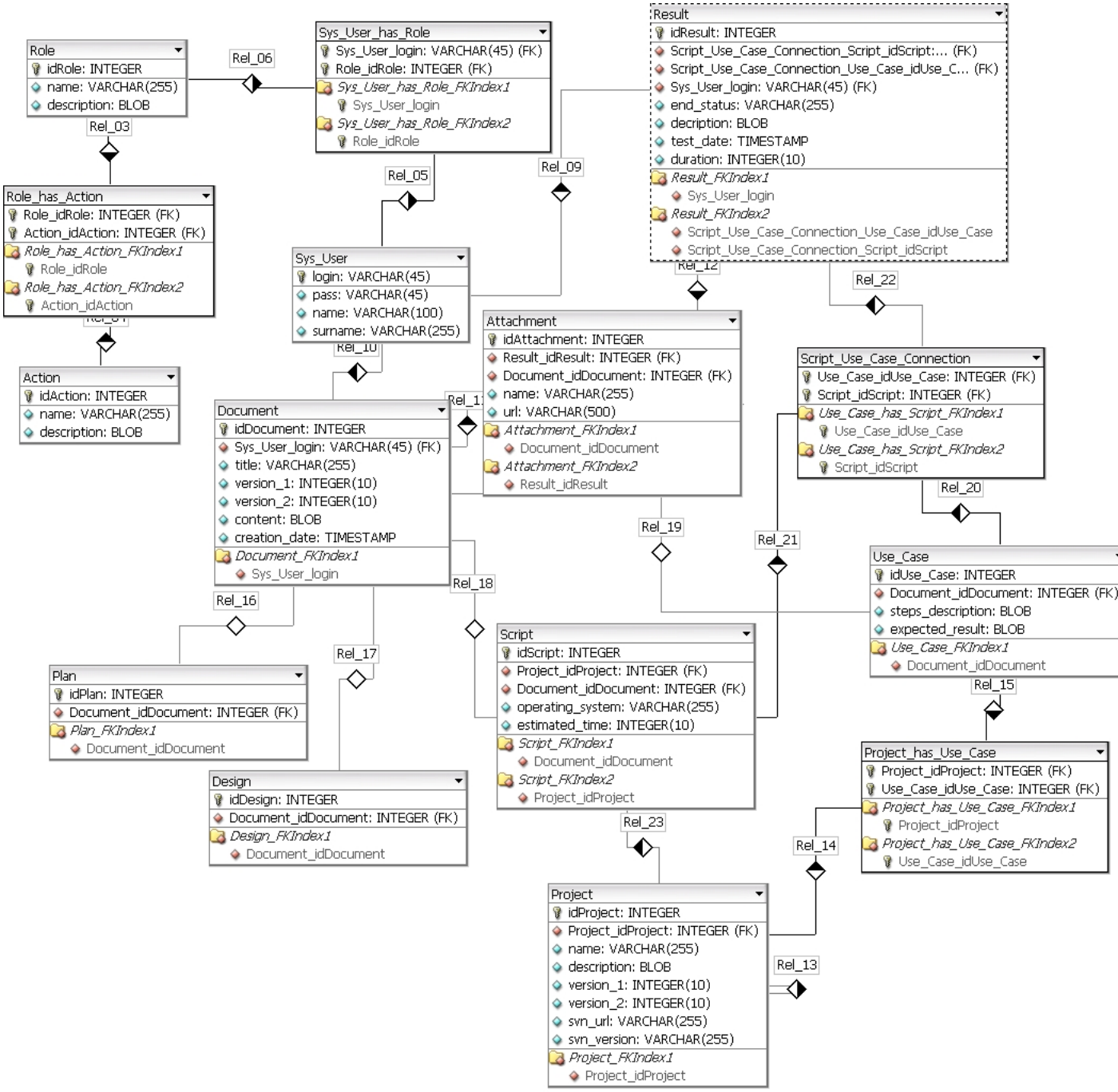
Encja odpowiadająca załącznikowi. Załączniki można dodawać zarówno do rezultatów testów (encja Result) jak i do dokumentów (encja Dokument).

2.12 Result

Encja odpowiadająca rezultatowi pojedynczego testu wykonanego w kontekści danego skryptu (CaseScriptConnection). Każdemu rezultatowi przyporządkowuje się testera, który ostatecznie wykonał dany test (zazwyczaj ten sam, który został przydzielony do skryptu).

- status – status przeprowadzenia danego testu (np. pass, acceptable, fail, not tested).
- description – komentarze i dodatkowe opisy odnośnie przebiegu testu (co i gdzie było nie tak jak być powinno).
- test_date – data i czas rozpoczęcia testu.
- duration – czas trwania testu w sekundach.

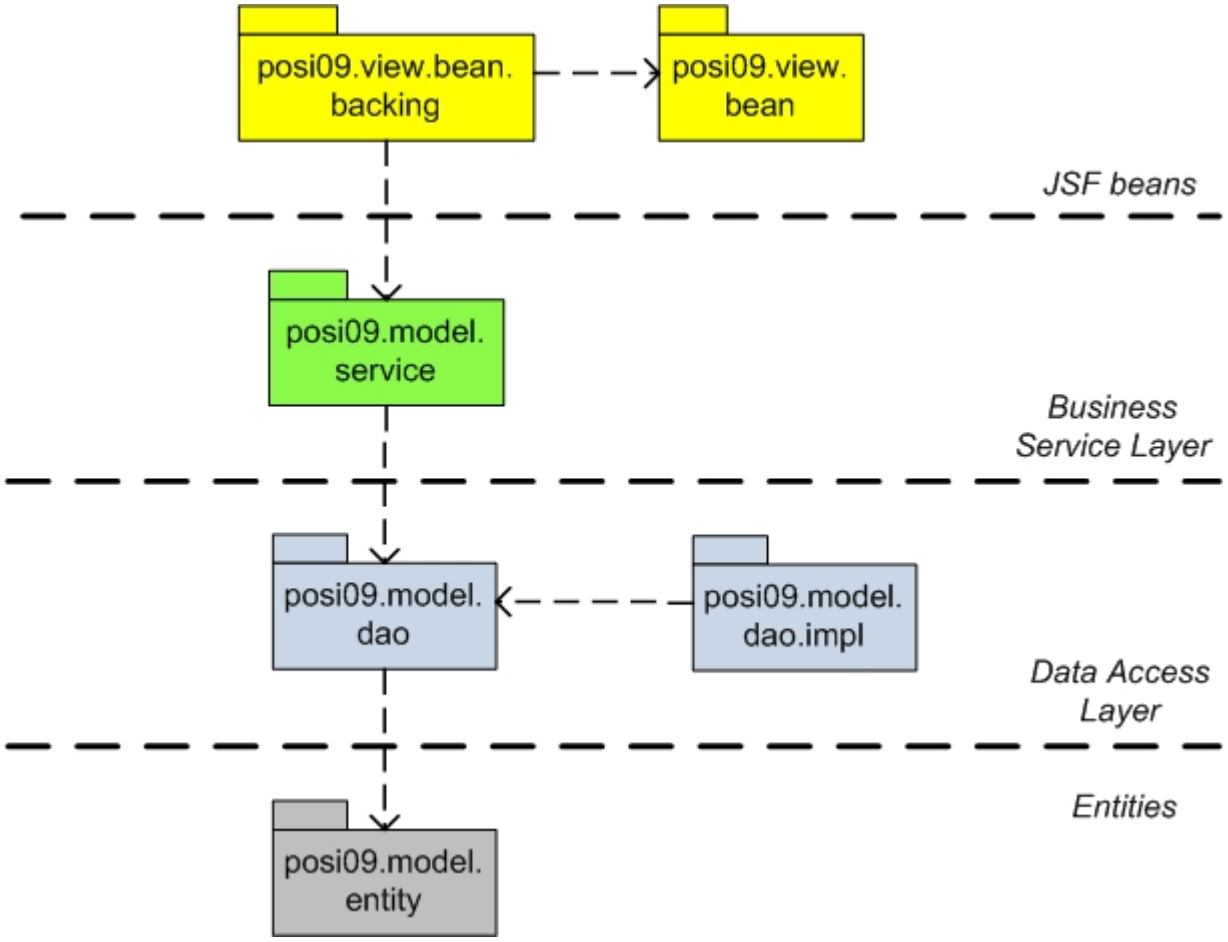
3 Schemat bazy danych



4 Dekompozycja obiektowa systemu

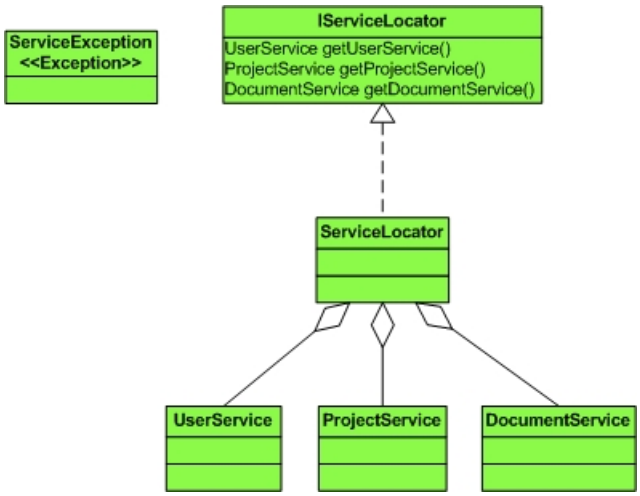
4.1 Pakiety

Poniższy diagram przedstawia podział klas na pakiety:



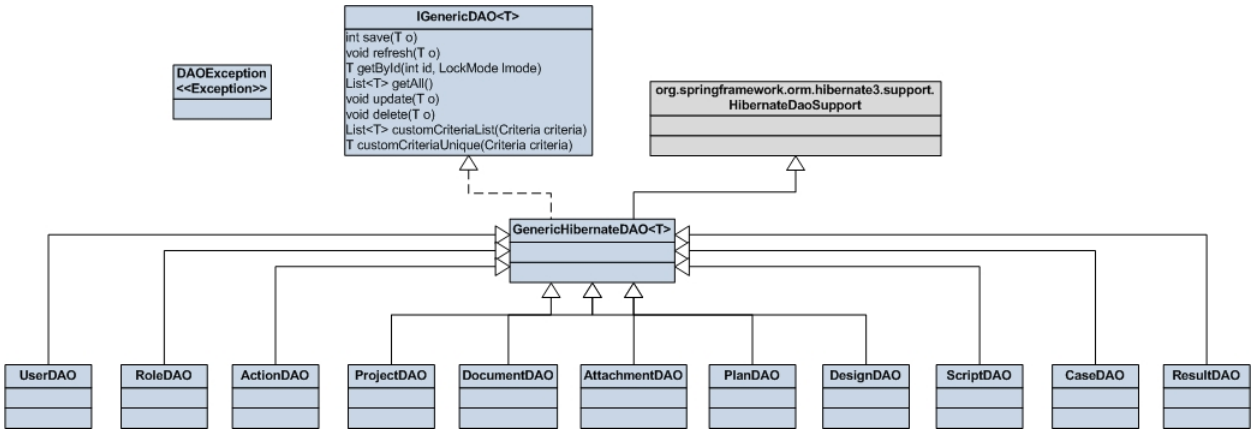
- posi09.view – zbiór klas odpowiedzialnych za obsługę zdarzeń aplikacyjnych oraz przechowywanie danych dla stron JSP,
- posi09.model.service – implementacja wzorca ServiceLocator,
- posi09.model.dao – implementacja wzorca Data Access Object,
- posi09.model.entity – klasy reprezentujące encje.

4.2 Serwisy biznesowe



Klasa implementująca interfejs `IServiceLocator` jest odpowiedzialna za dostarczanie odpowiedniego serwisu każdemu, kto potrzebuje przeczytać lub zaktualizować dane przechowywane w bazie danych. Serwisy zawierają referencje do obiektów DAO i inicjalizują transakcje bazodanowe, które są przenoszone na metody DAO dzięki `TransactionManager`'owi.

4.3 Warstwa dostępu do danych



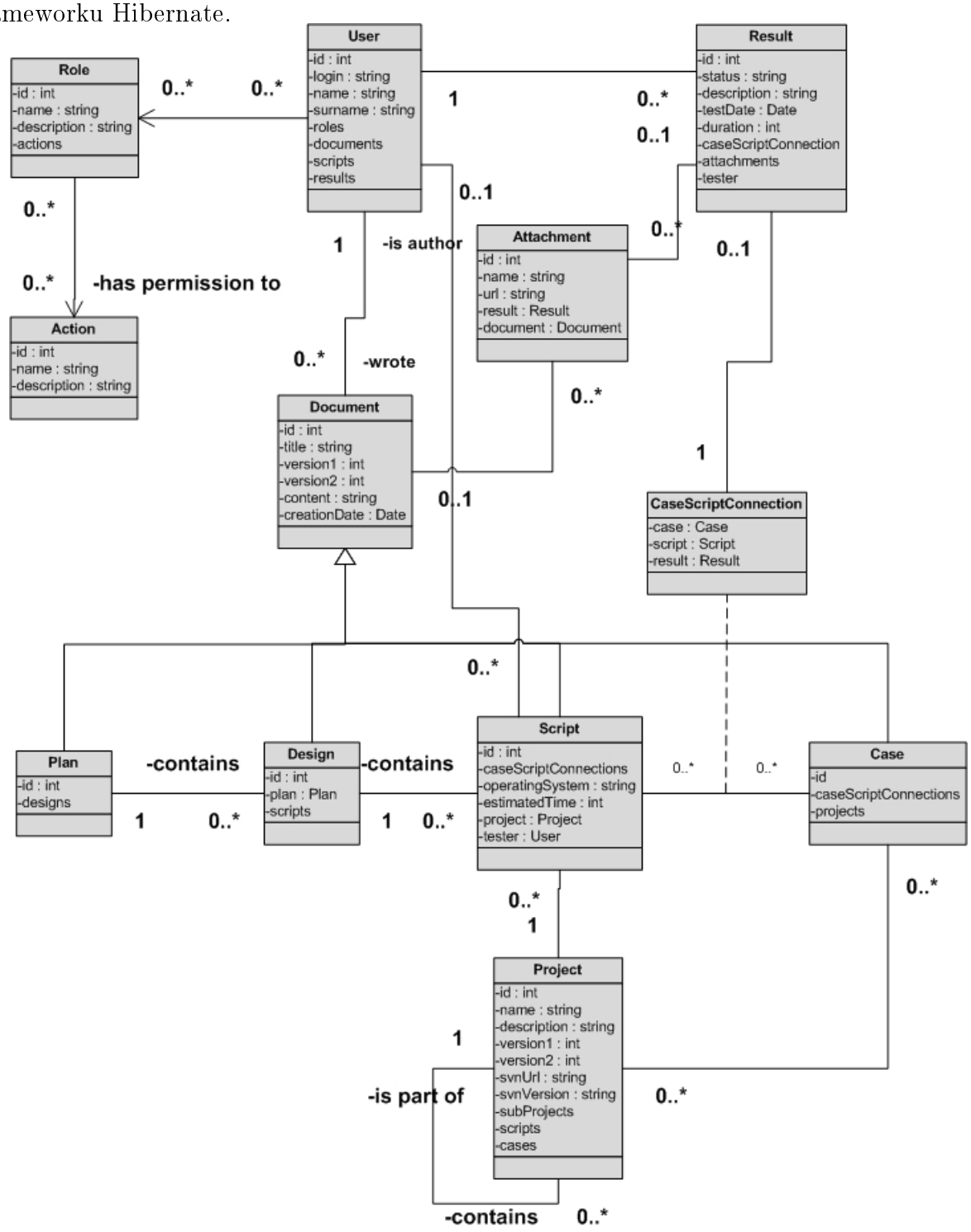
Interfejs `IGenericDAO<T>` stanowi abstrakcję nad interfejsem bazodanowym udostępniając następujące metody:

- `List<T> getAll(LockMode lock)` – zwraca wszystkie obiekty encji `T`,
- `T getById(int id, LockMode lock)` – zwraca obiekt encji `T` o identyfikatorze `id`,
- `int save(T o)` – utrwala obiekt `o` i zwraca przyznany mu identyfikator,
- `void update(T o)` – uaktualnia stan obiektu `o` w baziw danych,
- `void refresh(T o)` – odświeża atrybuty obiektu `o` danymi trwałymi,
- `void remove(T o)` – usuwa obiekt `o` z bazy danych,
- `List<T> customCriteriaList(Criteria criteria)` – zwraca listę obiektów spełniających warunek criteria,
- `T customCriteriaUnique(Criteria criteria)` – zwraca obiekt spełniający warunek criteria (wyjątek w przypadku, gdy takich obiektów jest więcej niż 1).

Interfejs zostanie zaimplementowany za pomocą frameworku Hibernate i biblioteki Spring ORM – klasa `GenericHibernateDAO<T>`. Dla każdej encji istnieje dokładnie jedna odpowiadająca klasa DAO wywiedziona z `GenericHibernateDAO` z doprecyzowaniem typu.

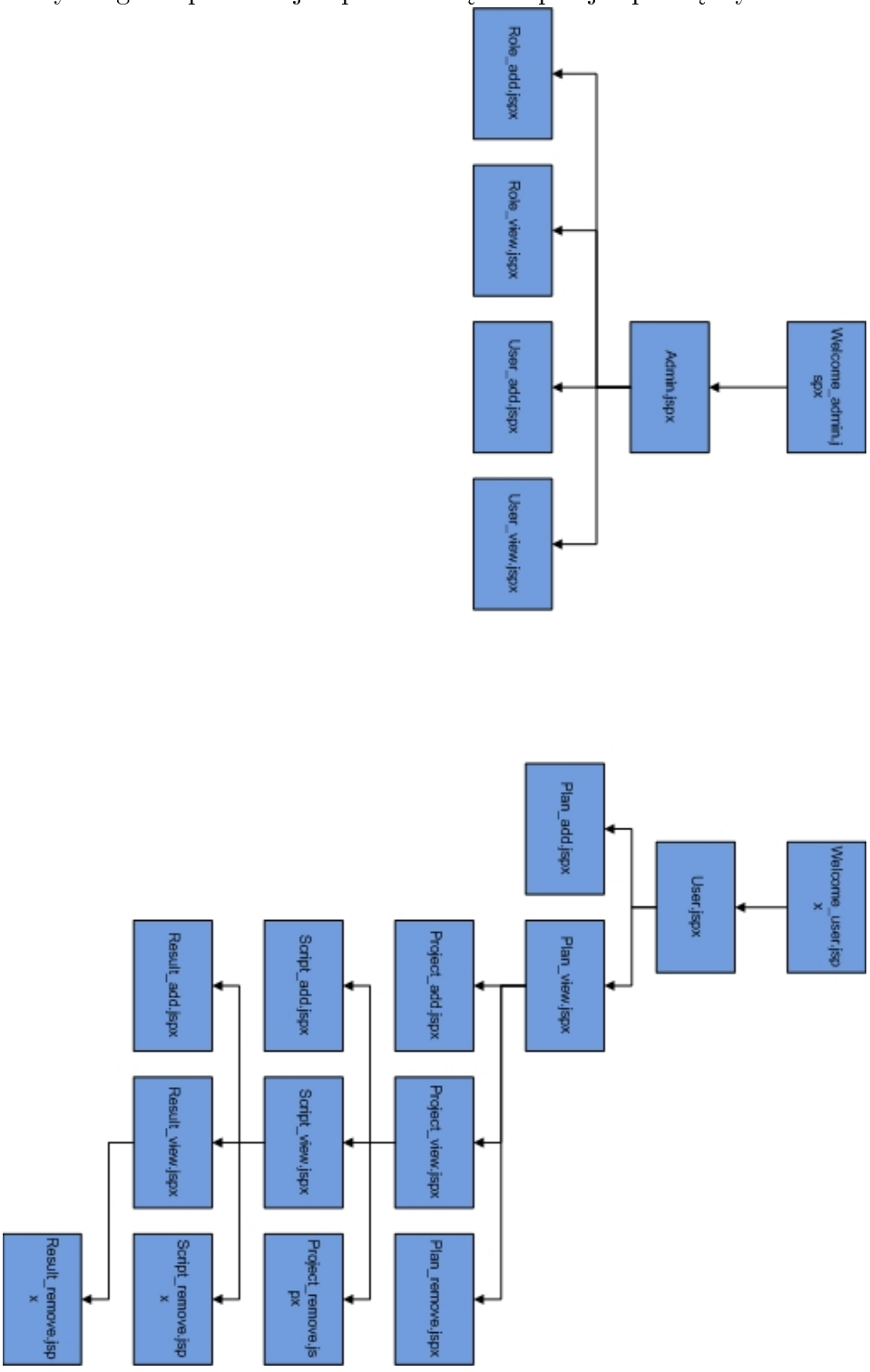
4.4 Encje

Klasy te reprezentują encje w bazie danych, ich obiekty są tworzone i zarządzane za pomocą frameworku Hibernate.



5 Interfejs – page flow

Poniższy diagram prezentuje uproszczoną sieć przejść pomiędzy stronami WWW.



6 Historia zmian

2009-10-27
wersja 0.1

2009-11-17
wersja 1.0

2009-12-01
wersja 1.1

2010-01-19
wersja 2.0-final