Laboratorium 12 - SOA.
Tematyka: Projekt weryfikujący zdobytą wiedzę.

Zadanie polega na napisaniu aplikacji zarządzania bazą danych – edytor postaci (opisującej elementy gry fantasy) wykorzystując mechanizm persistence w warstwie EJB oraz JSF do wyświetlania interfejsu użytkownika. Aplikacja musi składać się z osobnych modułów EJB i WEB. Katalog będzie składać się z dwóch encji, gdzie pierwsza będzie kategoriami a druga elementami należącymi do tych kategorii.
Specyfikacja katalogu i zapytań znajduje się na końcu instrukcji. Prócz wymienionych pól każda encja musi być wyposażona w klucz prywatny typu int lub Integer. Wszystkie akcje biznesowe muszą być zrealizowane w warstwie EJB i wystawione poprzez beany w warstwie WEB

Kamień milowy 1

1. Wyświetlanie danych w katalogu z podziałem na kategorie. Kategorie i elementy mają być dostarczone przez beana JSF. Elementy muszą być pogrupowane względem kategorii. (1 punkt)

2. Dodawanie nowych elementów i kategorii do katalogu. Należy przygotować podstrony pozwalające na dodawanie nowych kategorii i elementów. Formularze powinny udostępniać pola tekstowe pozwalające na ustawienie wszystkich atrybutów. Przypisanie do kategorii powinno się odbywać poprzez listę jednokrotnego wyboru z wykorzystaniem odpowiedniego konwertera. (1 punkt)

3. Usuwanie elementów i całych kategorii. Na stronie wyświetlającej elementy pogrupowane względem kategorii należy wyświetlić przy każdej z kategorii i każdym z elementów przycisk pozwalający na usunięcie. (1 punkt)

4. Modyfikacja elementów i kategorii katalogu. Należy przygotować podstrony pozwalające na edycję istniejących elementów i kategorii. Zarówno dodawanie jak i edycja powinny być zrealizowane tymi samymi stronami i beanami. Przypisanie do kategorii powinno się odbywać poprzez listę jednokrotnego wyboru z wykorzystaniem odpowiedniego konwertera. (1 punkt)

5. Realizacja zapytania w postaci NamedQuery. (1punkt)

Rozszerzenie aktualnej wersji aplikacji tak aby dostęp do aplikacji posiadali jedynie zalogowani użytkownicy. W tym celu serwis musi być zabezpieczony za pomocą np. mechanizmów JAAS (lub można skorzystać z innych form autoryzacji i autentykacji).

6. Należy zmodyfikować bazę dodając encję przedstawiającą użytkownika.

Każdy użytkownik może mieć kilka kategorii (wieża/las/jaskinia w zależności od numeru indeksu), każda z kategorii może mieć tylko jednego właściciela (użytkownika). Następnie należy zdefiniować dwie role: użytkownik i administrator. (1 punkt)

7. Dostęp do serwisu ma być ograniczony tylko dla zalogowanych użytkowników. Należy wykorzystać zabezpieczenie na poziomie web. (1 punkt)

8. *U*żytkownik może edytować tylko swoje kategorie (i ich elementy), natomiast administrator może edytować wszystkie elementy ze wszystkich kategorii. Zarówno użytkownik jak i administrator mają korzystać z tych samych stron JSF, rozróżnienie ról ma odbywać się w metodach EJB. (1 punkty)

9. Użytkownik powinien mieć możliwość zmiany swojego hasła, natomiast administrator powinien być w stanie zmieniać hasła wszystkich użytkowników. Hasła nie mogą być trzymane jako plain text. (1 punkt)

10. Aplikacja powinna blokować możliwość logowania z wybranych domen webowych - funkcjonalność proszę zrealizować z wykorzystaniem idei Filtrów - opis w dołączonym dokumencie. (1 punkt)

11. Nie ma możliwości realizacji więcej niż jedna sesja zalogowanego użytkownika. Jeśli nastąpiła by próba zalogowanie w trakcie pracy - zwracany jest albo kontekst sesyjny albo informacji że dany użytkownik jest już zalogowany.

Kamień milowy 2

12. Edycja elementów powinna być wieloetapowa (przynajmniej dwie podstrony). Należy zapewnić możliwość edycji kilku elementów w tym samym czasie w różnych zakładkach. Należy do tego wykorzystać beany CDI o odpowiednich zasięgach. (1 punkt)

13. Usuwanie i modyfkacja elementów powinno powodować przeładowanie jedynie kontrolki odpowiedzialnej za wyświetlanie elementów. W tym celu należy wykorzystać odpowiednie kontrolki PrimeFaces oraz partial processing. (1 punkt)
- tutaj proszę się zaznajomić z stroną http://primefaces.org/

14. Na głównej stronie aplikacji powinna być wyświetlana lista najlepiej uzbrojonych elfów lub najpotężniejszych magów lub najbogatszych smoków.
Bean dostarczający dane do wyświetlenia powinien w sposób ciągły przechowywać listę i aktualizować ją tylko w przypadku dodania, usunięcia lub modyfikacji elfa/maga/smoka. W tym celu należy zastosować model zdarzeń CDI oraz mechanizm push z PrimeFaces. (1 punkt)

15. Zakładamy, że środowisko magów/elfów/smoków jest bardzo negatywnie nastawione do nowo przybyłych. Niedopuszczalne jest aby nowy mag/elf/smok przewyższał swoją mocą/liczbą strzał/bogactwem osobnika o najwyższej wartości. W tym celu metodę dodawania nowego elementu należy opatrzyć interceptorem CDI,

który porówna wartość dodawanego elementu z tymi już istniejącymi w bazie danych i w razie potrzeby obniży ją do odpowiedniego poziomu. (1 punkt)

16. Właściciel każdej postaci powinien być informowany (za pomocą JMS) o pojawieniu się nowej postaci takiego samego typu dodanej przez innego użytkownika. Oznacza to, że jeśli dodamy nowego maga to wszyscy właściciele magów powinni być o tym poinformowani. Właściciele innych postaci nie. (1 pkt)

17. Przygotowanie usługi typu REST pozwalającej na pobranie wszystkich kategorii (wieża, las, jaskinia) wraz z elementami (mag, elf, smok) lub wybranej kategorii i elementów do niej należących. Usługa może zwracać jeden z formatów: XML lub JSON. (1 punkt)

18. Przygotowanie usługi typu REST pozwalającej na dodawanie nowego elementu do wybranej kategorii. Element przesyłany jako XML lub JSON. Przy oddawaniu zdania można wykorzystać gotowego klienta (np. curl). (1 punkt)

19. Realizacja klienta powyższych usług jako aplikacja Java SE. (1 punkt)

20. Zabezpieczenie wybranych usług mechanizmem JAAS. Przy oddawaniu zadania można zaprezentować działanie w przeglądarce. (1 punkt)

21. Realizacja dynamicznej negocjacji treści dla usługi z punktu 17. Należy przygotować obsługę przynajmniej dwóch wybranych języków i podczas zwracania wyników zastosować tłumacza. Tłumacz może być wykonany jako atrapa nie realizująca tłumaczenia a jedynie modyfikująca wszystkie zwracane obiekty typu String w zauważalny sposób. (1 punkt)

22. Nasza gra powinna być rozwijalna. Chcemy mieć możliwość dodawania nowych kategorii i postaci. Przygotowanie usług SOAP pozwalających na dodawania nowych kategorii oraz nowych postaci w danej kategorii do naszej gry - forma realizacji dowolna . Realizacja oczywiście jako element w oddzielnym deploy. (1pkt)

23. Nasz edytor może być wykorzystywany w dowolnym silniku gry. Zasymulujmy działania poprzez napisanie zewnętrznej usługi SOAP pozwalającej na modyfikacje parametru wybranej postaci (ilość złota dla smoka, ilość strzał dla elfa lub mana dla maga) (1pkt).

24. Przyjmijmy że po modyfikacji (wywołanej za pomocą usługi z punktu 23) każda postać po około 1 minucie wysyła do właściwej gry nową wartość mocy (wartość mocy jest zmieniana o losowa wylosowaną wartość z przedziału od -2 do 2) (1pkt)

Specyfikacja katalogu z danymi:

· Dla osób z numerem indeksu podzielnym przez 3 (niepodzielnym

przez 2):
    - jaskinia (kategoria);
        - powierzchnia: int;
    - smok (element):
        - imię: String,
        - złoto: int,
        - kolor: enum;
        - moc: enum;

- Dla osób z numerem indeksu podzielnym przez 2 (niepodzielnym przez 3):
    - wieża (kategoria):
        - wysokość: int;
    - mag (element):
        - imię: String,
        - mana: int,
        - żywioł: enum;
        - moc: enum;

- Dla pozostałych osób:
    - las (kategoria):
        - liczba drzew: int;
    - elf (element):
        - imię: String,
        - liczba strzał: int,
        - rodzaj łuku: enum;
        - moc: enum;