Uniwersytet Zielonogórski

Wydział Informatyki, Elektrotechniki i Automatyki

Kierunek: *Informatyka*

Specjalność: *Sieciowe systemy informatyczne*

Student (Grupa): *Piotr Joński (431IDZ)*

Promotor: *dr inż. Andrzej Marciniak*

Konsultant:

Temat Pracy:

- wersja polska — ***System zarządzania projektami dedykowany metodyce Scrum***
- wersja angielska — ***Project management system dedicated to Scrum methodology***

Cel Pracy:
PRZEPISAĆ Z KARTY DYPLOMOWEJ

Zakres Pracy:

1. Zakres pracy 1
2. Zakres pracy 2
3. Zakres pracy 3

Data podjęcia tematu: 01.01.2000

Podpisy:

Promotor	Student

Data przyjęcia pracy	Podpis promotora

Wyrażam zgodę na wykorzystanie do celów dydaktycznych i badawczych wyników zawartych w pracy dyplomowej i powstałych programów komputerowych.

Data	Podpis studenta

Uniwersytet Zielonogórski

Wydział Informatyki, Elektrotechniki i Automatyki

Praca dyplomowa

Kierunek: Informatyka

System zarządzania projektami dedykowany metodyce Scrum

Piotr Joński

Promotor:
dr inż. Andrzej Marciniak

Zielona Góra, 02 2016

Piotr Joński
431IDZ
Wydział Informatyki,
Elektrotechniki i Automatyki UZ

Zielona Góra 10/02/2016

OŚWIADCZENIE

Świadomy odpowiedzialności karnej oświadczam, że przedkładana praca dyplomowa pt.

System zarządzania projektami dedykowany metodyce Scrum

została napisana przeze mnie samodzielnie i nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem dyplomu wyższej uczelni lub tytułów zawodowych. Jednocześnie oświadczam, że w/w praca nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994r. o prawie autorskim i prawach pokrewnych innych osób (DZ. U. z roku 2000 Nr 80 poz. 904) oraz dóbr osobistych chronionych prawem cywilnym.

Oświadczam również, że egzemplarz pracy dyplomowej w formie wydruku komputerowego jest zgodny z egzemplarzem pracy dyplomowej w formie elektronicznej.

.....

podpis

Streszczenie

Celem pracy był projekt oraz implementacja systemu do zarządzania projektami dedykowanego dla metodyki Scrum.

W efekcie pracy powstała aplikacja webowa, która umożliwia tworzenie projektów zgodnie z wytycznymi jakie narzuca metoda Scrum. System posiada możliwość zarządzania projektami jak i użytkownikami z poziomu panelu administracyjnego.

W celu realizacji projektu zostały użyte takie technologie jak EJB (Enterprise Java Bean), która umożliwia proste i przyjemne tworzenie logiki biznesowej oraz zarządzania transakcjami. Do składowania danych posłużono się specyfikacją JPA wraz z frameworkiem Hibernate. Za warstwę wizualną odpowiada technologia JSF oraz open sourcowy framework Primefaces. System został wdrożony na serwer aplikacji jakim jest Wildfly oraz używa zewnętrznej bazy Postgresql do przechowywania danych. Serwer wraz z bazą zostały umiejscowione w kontenerze Dockera, przez co czas instalacji i uruchomienia systemu jest niewielki.

Tak zaimplementowany system spełnia początkowe założenia, a poprzez architekturę uruchomieniową jest łatwy w użyciu.

Słowa kluczowe: system do zarządzania projektami, scrum, docker

Spis treści

Spis rysunków

Spis tabel

Rozdział 1

Wstęp

1.1. Wprowadzenie i przegląd literatury

Co tu dać?

1.2. Cel i zakres pracy

Celem pracy było wytworzenie systemu do zarządzania projektami dedykowanego metody Scrum. Obecnie na rynku jest wiele zarówno darmowych jak i płatnych narzędzi, które oferują możliwość prowadzenia projektów rożnymi metodykami. W zakres pracy wchodzą:

1. zapoznanie się z literaturą tematu,
2. opracowanie założeń projektu,
3. spis wymagań funkcjonalnych i niefunkcjonalnych systemu,
4. implementacja wszystkich funkcjonalności oraz usunięcie powstałych błędów,
5. przetestowanie systemu ,
6. wytworzenie części opisowej pracy.

1.3. Struktura pracy

Opis rozdziałów zostanie uzupełniony na końcu pracyitemize

Rozdział 2

Wprowadzenie

2.1. Szkic problemu

Szybki rozwój w dziedzinie informatyki spowodował wzrost zapotrzebowania na systemy, które pomogłyby rozwiązywać problemy kwestii organizacyjnej projektów. W dzisiejszych czasach zarządzanie projektami jest szerokim zagadnieniem, a na jego temat powstało wiele publikacji. Autorzy proponują takie rozwiązania dot. zarządzania projektami jak:

1. scrum,
2. lean Softrware Development,
3. feature Driven Development,
4. test Driven Develipment.

Nie jest to wyczerpująca lista metodyk - istnieje wiele innych metod tworzenia oprogramowania, jednak opis wszystkich wykracza po za zakres tej pracy. W mojej pracy dyplomowej poruszam temat systemu do zarządzania projektami, który jest dedykowany metodyce Scrum. Nie bez powodu wybrana została właśnie ta zwinna metodyka - jest to na chwilę obecną najbardziej popularna forma wytwarzania oprogramowania, a umiejętności pracy wraz z nią są pożądane przez większość pracodawców na całym świecie.

2.2. Czym właściwie jest scrum

Scrum jest zwinną metodyką wytwarzania oprogramowania, której początki sięgają połowy lat 80. Polega ona na przyrostowym wytwarzaniu produktu oraz stałej komunika-

cji z klientem bądź też jego reprezentantem - właścicielem produktu (*ang. product owner*). Zespół deweloperski *team* podejmuje się przygotowania wybranych przez siebie funkcjonalności, których wybór może być zależny od właściciela produktu - to on ustala priorytety. Jednak w celu zminimalizowania ryzyka "wyższej władzy", czyli narzucaniu zbyt wysokich wymagań i nieosiągalnych terminów, co często czynią product ownerzy, została wprowadzona kolejna rola - scrum master, którą ciężko jest przetłumaczyć na język polski (z tego względu w dalszej części pracy będę posługiwał się właśnie tym terminem). To on "chroni" zespół przed przepracowaniem i rozwiązuje napotykane po drodze problemy. Aby prawidłowo przedstawić omawiany temat należy zapoznać się z kilkoma pojęciami, które są nieodłącznym elementem tej metody:

1. **właściciel produktu** (*ang. product owner*) - osoba odpowiedzialna za projekt,

2. **scrum master** - osoba, która pomaga zespołowi w organizacji czasu pracy a także rozwiązuje powstałe problemu,

3. **zespół deweloperski** (*ang. team*) - zespół programistów wspólnie pracujący nad wytworzeniem produktu,

4. **rejestr produktu** (*ang. backlog*) - jest to zbiór zadań / funkcjonalności wchodzące w skład wytwarzanego projektu,

5. **interwał** (*ang. sprint*) - interwały czasowe, w których realizowane są zadania. Bezpośrednio przed każdym sprintem występuje planowanie, czyli wybieranie funkcjonalności do zrealizowania w danym sprincie. Bezpośrednio po sprincie powinna wystąpić retrospektywa oraz demo produktu, które jest często pomijane przez wiele zespołów,

6. **historyjka** (*ang. story*) - najmniejsza jednostka podlegająca ocenie (wg. wybranej przez zespół skali).

7. **zadanie** (*ang. task*) - zadania, które mogą być powiązane bezpośrednio ze story lub projektem.

8. **retrospektywa** (*ang. retrospective*) - wydarzenie, które ma miejsce na koniec sprintu. W tym momencie zespół deweloperski spisuje wszystkie wady i zalety minionego sprintu oraz wspólnie z scrum masterem starają się rozwiązać zaistniałe problemy. Jest to jeden z ważniejszych elementów Scruma.

2.3. Przegląd dostępnych narzędzi

Wytwarzając ten system skupiłem się, aby wpasował się on w wybraną przeze mnie metodykę oraz rozwiązywał szereg mankamentów, które napotkałem podczas korzystania z obecnych już rozwiązań. W celu lepszego zrozumienia problemów postanowiłem opisać kilka z nich.

Pierwszym i niewątpliwie największym problemem tego typu systemów jest to, że są one płatne, przez co nie wszystkich na nie stać. Niektóre są darmowe, lecz tylko dla projektów typu open source, co nie sprawdza się podczas pracy nad wspólnymi projektami w firmie lub np. pracą dyplomową.

Kolejną wadą jest to, że nie są one proste w instalacji. Często występują jako potężne programy, przez co nie można ich bezpłatnie hostować w chmurze gdyż potrzebują dużo płatnych zasobów.

Ostatnim, jednak nie mniej ważnym problemem jest ich czytelność. Oferują one szereg zbędnej zwykłemu użytkownikowi - programiście - funkcjonalności, która jest przydatna tylko w określonych warunkach. Taki ogrom zakładek i okienek powoduje często zamęt i niezrozumienie wśród programistów.

Aby zobrazować wagę powyższych problemów zostały one zestawione w poniższej tabeli porównującej wybrane systemy wraz z systemem, który został wytworzony w ramach pracy dyplomowej.

Tab. 2.1. Porównanie wybranych systemów zarządzania projektami

Cecha systemu	Wybrane systemy			
	JIRA	Bitbucket	GitHub	Autorski system
Darmowy	do 10 użyt.	do 5 użyt.	open source	tak
Hosting	wew. / zew.	zew.	zew.	wew. / zew.
Czas instalacji	ok. 5 minut	nd.	nd.	ok. 2 minuty
Wymagania	baza danych	nd.	nd.	docker
Cechy	issue tracker wiki	issue tracker wiki	issue tracker wiki	issue tracker

2.4. Rozwiązanie problemu

Celem pracy było wytworzenie systemu do zarządzania projektami dedykowanego metody Scrum. Obecnie na rynku jest wiele zarówno darmowych jak i płatnych narzędzi przedstawionych w tabeli 2.1, które oferują możliwość prowadzenia projektów rożnymi metodykami. Wcześniej wymienione wady skłoniły mnie do opracowania własnego systemu, który ma za zadanie rozwiązać wspomniane problemy w następujący sposób:

1. łatwość instalacji - do tego celu zostało wykorzystane oprogramowanie docker, które wspiera błyskawiczne stawianie systemów,
2. przejrzysty interfejs użytkownika - został osiągnięty dzięki open sourcowej bibliotece PrimeFaces,
3. bez zbędnej funkcjonalności - zostały zaimplementowane tylko obligatoryjne funkcje tego typu systemu oraz niewielka ilość udogodnień.

Rozdział 3

Analiza biznesowa problemu i założenia projektowe

3.1. Role i użytkownicy systemu

W projektowanym systemie występują cztery rodzaje użytkowników:

1. administrator - posiada on największe uprawnienia w systemie,
2. właściciel produktu - odzwierciela rolę właściciela produktu w Scrumie,
3. scrum master - odzwierciela rolę scrum mastera w Scrumie,
4. deweloper - posiada najmniejsze uprawnienia, jest częścią zespołu deweloperskiego.

Na tym etapie warto wspomnieć, że każdy użytkownik jest jest deweloperem. Każdy projekt może mieć tylko jednego właściciela produktu, a każdy z nich może być product ownerem tylko raz. Dodatkowo zespół deweloperski ma przydzielonego tylko jednego scrum mastera, przy czym scrum master może być przypisany do wielu zespołów jednocześnie.

3.2. Wymagania funkcjonalne

Prezentowany przeze mnie system ma pewne założenia oraz wymagania. W tym rozdziale zajmę się opisem wymagań funkcjonalnych.

We wcześniejszym akapicie zostały omówione role w systemie. Każda z tych ról ma pewne uprawnienia lub restrykcje. Każda taka cecha zostanie przedstawiona jako wymaganie funkcjonalne prezentowanego systemy.

Jednym ze sposobów na prowadzenie dokumentacji projektu, jak i zbioru wymagań jest utrzymywanie rejestru historyjek użytkownika. Historyjki użytkownika są częścią zwinnych metody prowadzenia projektu. Jako że wytwarzanemu systemowi towarzyszy metodyka Scrum, nie mogło tutaj zabraknąć tego elementu.

> *Historyjka jest jednostką funkcjonalności w projektach XP. Pokazujemy postęp prac, dostarczając przetestowany i zintegrowany kod, który składa się na implementację danej historyjki. Historyjka powinna być zrozumiała i wartościowa dla klientów, testowalna przez programistów i na tyle mała, żeby programiści mogli zaimplementować sześć historyjek w takcie jednej iteracji*[1].

Historyjki mogą mieć wiele wzorców. W tej pracy są stosowane dwa z nich:

- Jako *<typ użytkownika>* mogę *<nazwa zadania>*.
- Jako *<typ użytkownika>* mogę *<nazwa zadania>* w celu *<cel>*[6]

Administrator

1. jako administrator mogę tworzyć nowych użytkowników w celu dodania ich do systemu,
2. jako administrator mogę usuwać użytkowników z systemu z wyjątkiem siebie samego,
3. jako administrator mogę dodawać użytkowników do zespołu w celu modyfikacji zespołu,
4. jako administrator mogę usuwać użytkowników z zespołu w celu modyfikacji zespołu,
5. jako administrator mogę nadać uprawnienia administratora dowolnemu użytkownikowi,
6. jako administrator mogę odebrać uprawnienia administratora dowolnemu administratorowi,
7. jako administrator mogę przypisać właściciela produktu do projektu,
8. jako administrator mogę usunąć właściciela produktu z projektu,
9. jako administrator mogę utworzyć projekt w celu dodania go do systemu,

[1] K. Beck, M. Fowler, *Planning Extreme Programming*, Addison-Wesley, Boston 2000, s.42

10. jako administrator mogę usunąć projekt w celu usunięcia z systemu oraz powiązanych z nim elementów t.j. sprint, story, backlog oraz inne. Operacja usuwania odbywa się kaskadowo,
11. jako administrator mogę modyfikować projekt,
12. jako administrator mogę tworzyć nowe zespoły w celu dodania ich do systemu,
13. jako administrator mogę dodawać zespoły do projektów w celu przydzielenia uprawnień,
14. jako administrator mogę usuwać zespoły z projektów w celu odebrania uprawnień,
15. jako administrator mogę przypisać scrum mastera do zespołu,
16. jako administrator mogę usunąć scrum mastera z zespołu,
17. jako administrator mogę tworzyć, usuwać oraz edytować statusy zadań,
18. jako administrator mogę tworzyć, usuwać oraz edytować priorytety zadań,
19. jako administrator mogę tworzyć, usuwać oraz edytować typy zadań.

Product owner

1. TODO

Scrum master

1. TODO

Developer

1. jako deweloper mogę tworzyć zadania,
2. jako deweloper mogę usuwać zadania,
3. jako deweloper mogę dodawać komentarze do zadań,
4. jako deweloper mogę dodawać komentarze do retrospektyw,
5. jako deweloper mogę edytować swój profil,
6. jako deweloper mogę zmienić swoje hasło,
7. jako deweloper mogę przeglądać profile innych użytkowników,
8. jako deweloper mogę przeglądać wszystkie projekty, do których jestem przypisany,
9. jako deweloper mogę przypisać zadanie do dowolnego użytkownika,

3.3. Wymagania niefunkcjonalne

Kolejnym zagadnieniem, które zostanie poruszone w tym rozdziale będą wymagania niefunkcjonalne. Do analizy zostanie wykorzystana metoda FURPS. Oto pełny spis wymagań niefunkcjonalnych systemu:

1. **F**unctionality - funkcjonalność, system powinien:
    - spełniać wszystkie wymagania funkcjonalne,
    - mieć możliwość administracji poprzez panel administracyjny,
    - posiadać audyt w postaci logów systemu,
    - być łatwo rozszerzalny.
2. **U**sability - używalność, system powinien posiadać następujące cechy:
    - ergonomia - łatwość używania,
    - look & feel, czyli estetyczność oraz możliwość modyfikacji wyglądu,
    - internacjonalizacja - wiele wersji językowych.
3. **R**eliability - niezawodność, w której skład wchodzą:
    - dostępność, czyli czas pomiędzy awariami,
    - odzyskiwalność - ile czasu zajmie ponowne uruchomienie systemu.
4. **P**erformance - wydajność, system powinien:
    - być przepustowy ,
    - mieć dużą responsywność,
    - działać szybko.
5. **S**supportability - wsparcie, do którego należy:
    - prostota w instalacji,
    - łatwość konfiguracji,
    - adaptowalność, czyli możliwość zaadoptowania systemu do innych warunków,
    - testowalność, czyli testy jednostkowe oraz integracyjne.

Rozdział 4

Dokumentacja projektowa

4.1. Architektura systemu

Wytworzony przeze mnie system jest aplikacją webową, która korzysta z bazy danych do odczytu i zapisu niezbędnych informacji. Z systemu będą korzystać cztery, wcześniej omówione, rodzaje użytkowników poprzez interfejs WWW. Dodatkowo system został zaprojektowany w taki sposób, aby była możliwość rozbudowy o dodatkowe zdalne aplikacje klienckie umożliwiające komunikację z systemem. System wraz z bazą danych znajdują się w kontenerze Dockera.

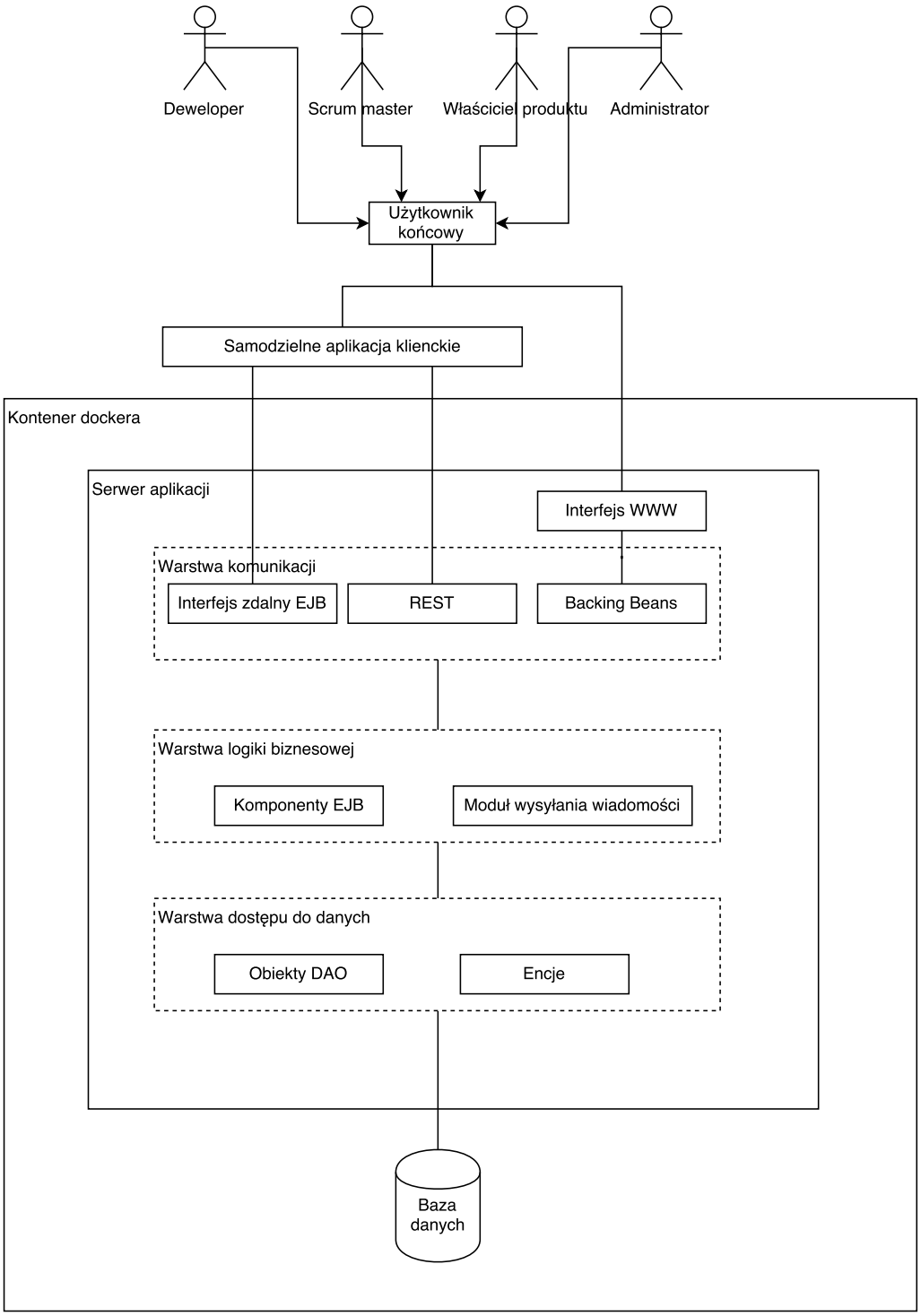
Jak wiadomo, połączenie ze zdalnymi klientami nie jest gwarantowane, co skłania do zastosowania mechanizmu komunikacji Java Message Service (JMS). Jednak pomimo, że połączenie nie jest ani stabilne, ani nawet w ogóle nie wiadomo czy zostało nawiązane, zależy nam, aby informować użytkowników o różnego rodzaju błędach, lub przesyłać im dane na bieżąco. Co prawda JMS umożliwia realizację tego zadania lecz do komunikacji z samodzielną aplikacją zostały udostępnione inne ścieżki komunikacji.

Pierwsza z nich to warstwa usługi REST. Dzięki takiemu rozwiązaniu z aplikacją może komunikować się dowolny system niezależnie od tego w jakim języku został napisany. Warstwa ta nie została zaimplementowana w całości, lecz jedynie w jako fragment funkcjonalności w celu zobrazowania komunikacji ze zdalnymi klientami.

Kolejna ścieżka komunikacji to zdalny interfejs komponentu EJB (Enterprise JavaBean). Mechanizm ten wprowadza możliwość komunikacji asynchronicznej z aplikacją. Jest one jednak ograniczony tylko do klientów napisanych w języku Java.

Wyżej wymienione rozwiązania stanowią warstwę do komunikacji z aplikacjami klienckimi. Kolejnym krokiem było zaprojektowanie warstwy do komunikacji z klientem WWW.

Co prawda warstwa REST spełniałaby swoje wymagania, lecz w tym celu posłużymy się, specjalnie zaprojektowaną do tego rodzaju zadań, technologią JavaServer Faces (JSF), która będzie wymagała dodatkowej warstwy z którą będzie się komunikować. Warstwa ta to tzw. komponenty Java Beans zwane również Backing Beans (BB). Rysunek 4.1 przedstawia diagram systemowy aplikacji:



Rys. 4.1. Diagram systemowy aplikacji

Jak wiadomo, interfejsy graficzne, a co za tym idzie – sposób ich obsługi będą się różniły w zależności od klienta. Najważniejszą różnicą będzie sposób walidacji danych oraz obsługi błędów. W samodzielnej aplikacji klienckiej walidacja danych oraz obsługa błędów powinna wystąpić możliwie szybko, aby nie generować zbędnego ruchu sieciowego. Jeżeli w trakcie przetwarzania żądania wystąpią jakiekolwiek błędy, to powinny one zostać zamienione na wyjątki aplikacji oraz przekazane do klienta. Również w aplikacji WWW walidacja danych wejściowych powinna znajdywać się na poziomie klienta. Jednak nie można wykluczyć sytuacji, w której błędy pojawią się po stronie serwera nawet po poprawnej walidacji danych. Z tego względu błędy powinny być konwertowane po stronie serwera (przez dodatkową warstwę obsługi JSF) na obiekty typu FaceMessage i dodawane bezpośrednio do kontekstu aplikacji WWW.

Wymagania odnośnie walidacji oraz obsługi błędów mogą skłaniać do wprowadzenia dodatkowej warstwy w architekturze systemu. Nie mniej jednak taka warstwa nie została wprowadzona, gdyż spowodowało by to nadmierny przyrost klas oraz niepotrzebne uogólnienie systemu. Zamiast tego logika biznesowa generuje wyjątki aplikacji, gdy zajdzie taka potrzeba oraz przekazuje je warstwie wyżej lub klientom, które wywołują dane komponenty logiki biznesowej. Oznacza to tyle, że obsługa błędów aplikacji w samodzielnej aplikacji Javy powinna być zaimplementowana po stronie samej aplikacji, natomiast interfejs WWW posiada dodatkową warstwę w postaci komponentów JavaBeans, które takową obsługę zapewnią. Aby zrozumieć taką modyfikację należy wykonać bardziej szczegółowy diagram systemu. Rysunek 3 przedstawia zmodyfikowany diagram systemu.

4.1.1. Identyfikacja komponentów biznesowych

W oparciu o artefakty metodyki Scrum oraz wymagania funkcjonalne z systemu wyłaniają się następujące komponenty biznesowe:

Deweloper – użytkownik aplikacji, może przeglądać oraz modyfikować *zadania* w *projektach*, do których należy.

Scrum master – użytkownik aplikacji, jest przypisany do jednego lub wielu *zespołów*.

Właściciel produktu – użytkownik aplikacji, jest przypisany tylko do jednego *projektu*.

Administrator – użytkownik aplikacji, zarządza całym systemem. Może tworzyć nowe *projekty* oraz *zespoły*.

Zespół – zbiór *deweloperów*, może być powiązany z *projektem*.

Projekt – zbiór *zadań*, posiada *właściciela produktu*.

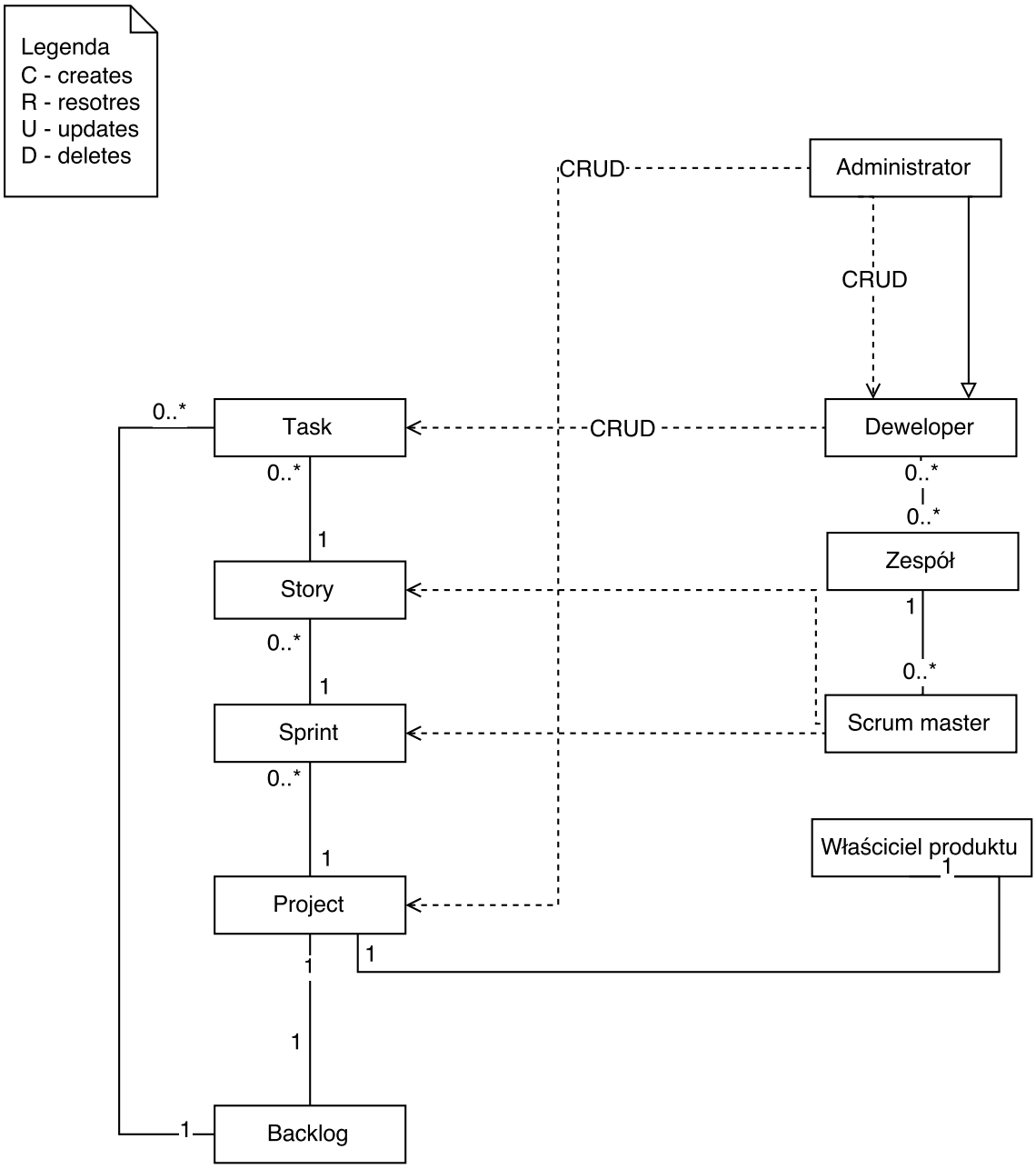
Sprint – jest tworzone przez *scrum mastera*. Posiada story.

Story – jest agregatem *zadań*.

Backlog – przynależy do *projektu*, posiada *zadania*.

Zadanie – jest tworzone przez *użytkowników*.

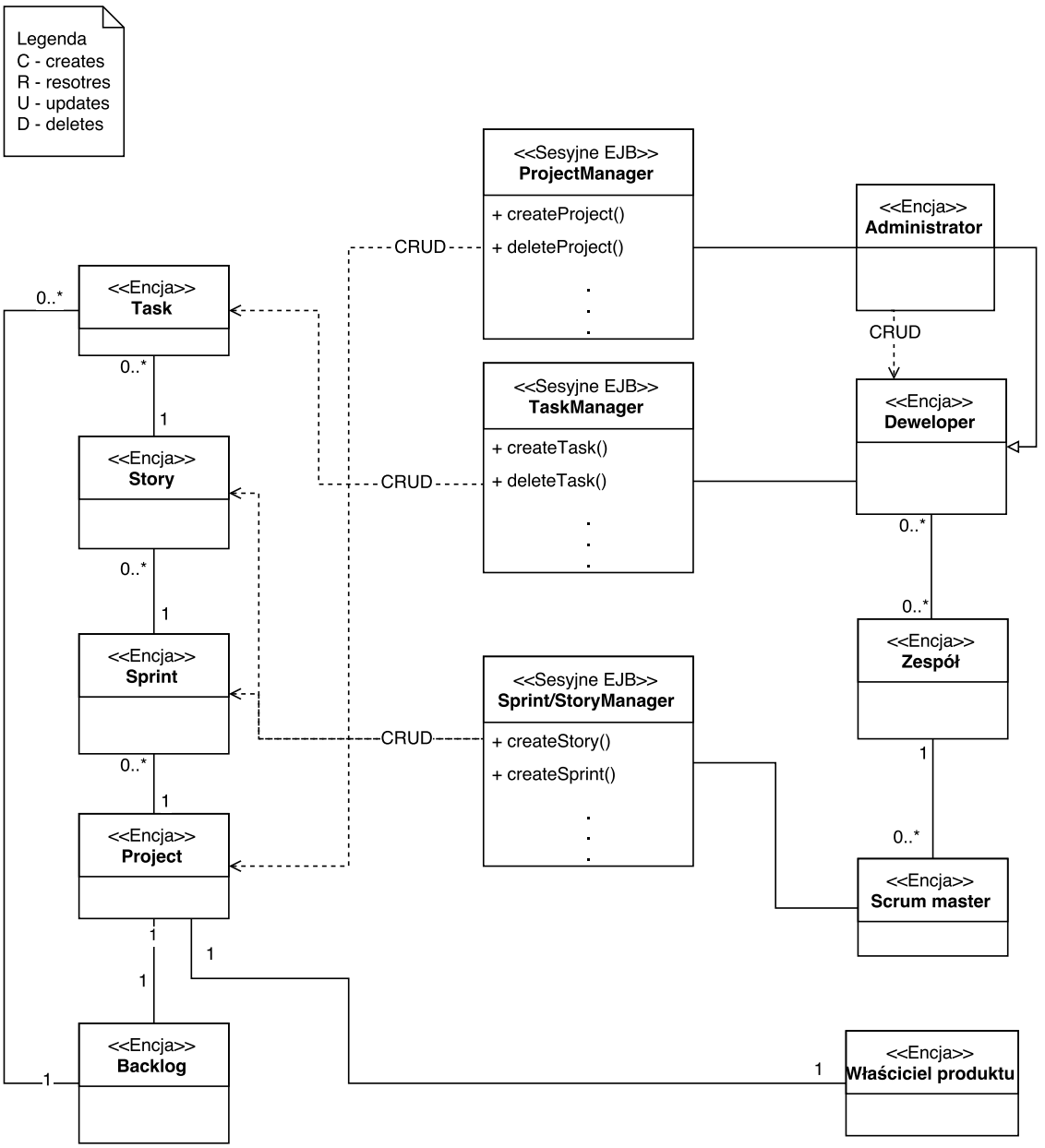
Należy wspomnieć, iż jest to częściowy opis obiektów biznesowych, który ma na celu zobrazowanie procesu powstawania aplikacji. Wyczerpujący opis tych obiektów byłby zbyt rozległy i nie jest on meritum części opisowej pracy. Aby zobrazować wpływ poszczególnych jednostek biznesowych oraz ich wzajemne relacje należy przedstawić ich diagram UML:



Rys. 4.2. Diagram jednostek biznesowych

4.2. Struktura projektu

W poprzedniej sekcji została opisana ogólna architektura systemu umożliwiająca pogląd na całą aplikację. Teraz zostaną opisane wybrane komponenty. Zanim jednak to zrobimy zostanie przedstawiony bardziej szczegółowy diagram jednostek biznesowych, który uwzględnia funkcje, jakie można wykonywać w systemie. Diagram powstał w oparciu o wymagania funkcjonalne, które szczegółowo opisują, co dany użytkownik może robić, oraz o identyfikację komponentów biznesowych omówionych wcześniej. Także i tutaj należy zwrócić uwagę, iż nie jest to wyczerpujący diagram klas opracowanych w systemie:



Rys. 4.3. Szczegółowy diagram wybranych klas

4.2.1. Klasyfikacja komponentów EJB

W tej sekcji zostaną opisane komponenty EJB używane w aplikacji oraz takie, które nie zostały użyte wraz z podaniem argumentów dlaczego je pominięto.

4.2.1.1. Komponenty encyjne

Reprezentują rekordy utrwalone w bazie danych. Komponenty encyjne mogą być używane do reprezentowania rzeczowników lub rzeczy z opisu funkcjonalnego. Jeżeli jednostka biznesowa posiada odpowiednik w rzeczywistości, to jest to prawdopodobnie komponent encyjny.

4.2.1.2. Komponenty sesyjne

Podczas gdy komponenty encyjne są rzeczami w aplikacji, komponenty sesyjne określają czynności jakie można na tych rzeczach wykonać. Są one jednostkami kontrolującymi procesy biznesowe. Zatem, aby wyodrębnić ten rodzaj komponentów należy się skupić na tym, co aplikacja może robić. Przyglądając się diagramowi klas można zauważyć, że ProjectManager może wykonywać operacje Create Restore Update Delete (CRUD) na projektach. Gdy w dowolnej aplikacji funkcjonalności skupiają się zazwyczaj wokół jednej lub więcej encji, to znaczy, że prawdopodobnie jest to komponent sesyjne. Ta reguła również tutaj ma swoje zastosowanie. Zostały utworzone odpowiednie komponenty sesyjne, które są rodzaju menadżerem spinającym funkcjonalności biznesowe, które są ze sobą powiązane. Ponieważ komponent sesyjny definiuje zbiór zachowań, to każde takie zachowanie można podporządkować jednej metodzie.

Głównym zadaniem aplikacji będzie przeglądanie projektów oraz zadań. W tym celu zostały utworzone odpowiednie komponenty sesyjne dla każdego rodzaju obiektu biznesowego, które jednak nie zostały przedstawione na szczegółowym diagramie, ze względu na ich obszerność. Każdy taki komponent posiada metody CRUD, które umożliwiają wykonywanie dowolnych operacji na tychże obiektach.

Tworzona aplikacja skupia się na zarządzaniu projektami, więc bez konfiguracji wstępnej – utworzenia użytkowników, przyznanie praw właściciela produktu, czy administratora – zarządzanie, a nawet samo utworzenie projektu będzie nie możliwe. Ponieważ tymi rzeczami zajmuje się administrator, więc w działającym systemie musi istnieć już użytkownik z uprawnieniami administratora. Dzięki temu system jest gotowy do działania zaraz po uruchomieniu.

4.2.1.3. Komponenty sterowane komunikatami

Pomimo iż zostało postanowione, że w aplikacji nie będą użyte komponenty sterowane komunikatami, nic nie stoi na przeszkodzie, aby głębiej przemyśleć możliwość wykorzystania takich komponentów. Zastanówmy się, w jakich sytuacjach mogłyby one być korzystne.

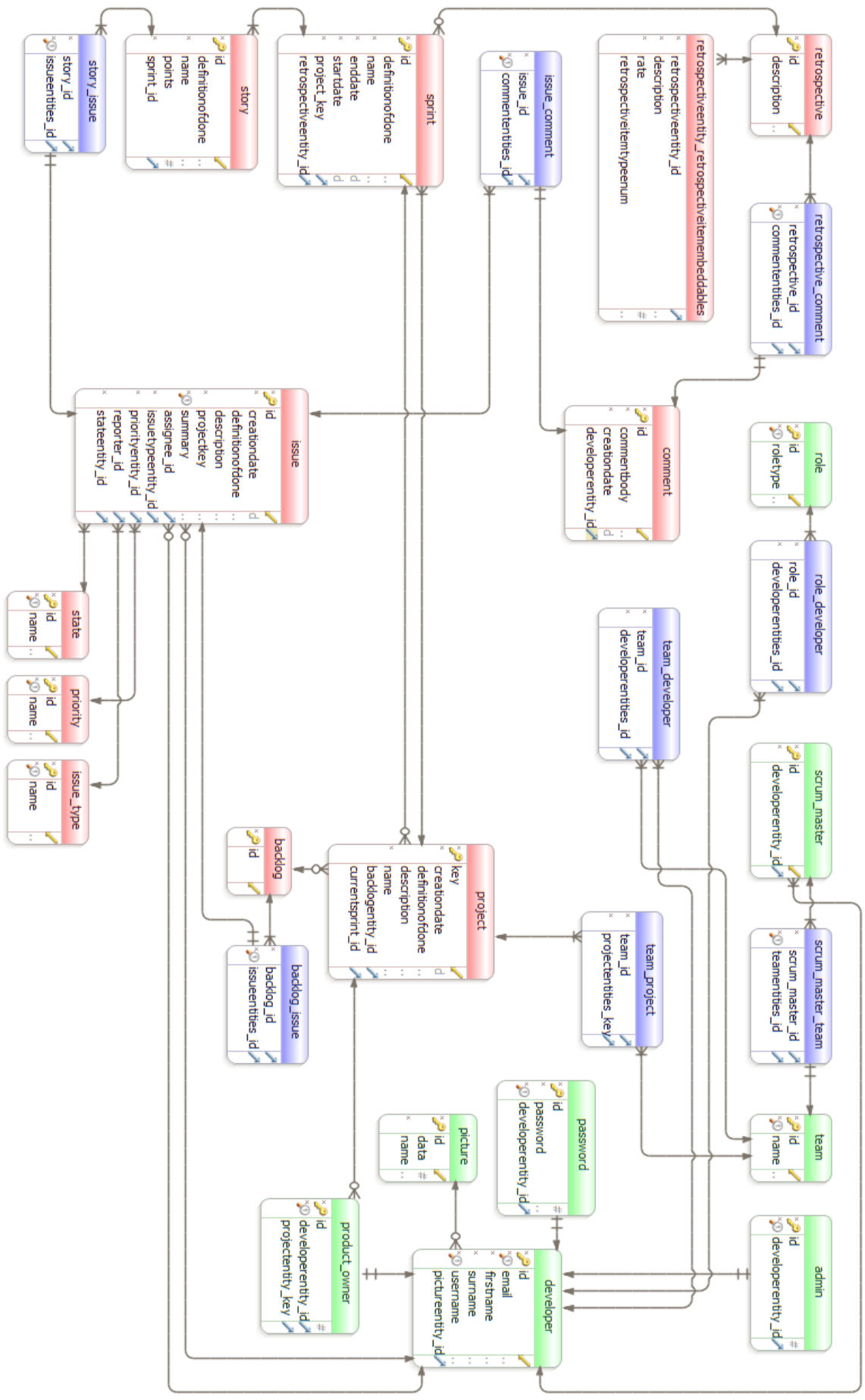
Pierwszą rzeczą, która nasuwa się na myśl jest wykorzystanie MDB (Message Driven Bean) generowania i wysyłania e-maili z hasłem po utworzeniu użytkownika. Taka wiadomość mogła by być rozgłoszona w systemie i dzięki temu różne komponenty mogłyby obsłużyć to zdarzenie. W systemie zrezygnowano jednak z tego typu technik.

Drugim możliwym zastosowaniem jest możliwość wysyłania rozgłoszeń w systemie. Jednak że aplikacja nie wprowadza funkcjonalności wiadomości systemowych również i tu ten komponent nie ma zastosowania.

Na tym etapie warto wspomnieć, że komponenty sterowane komunikatami można wprowadzić na każdym etapie udoskonalania projektu jednak warto zwrócić uwagę na to, jak zachować odpowiedni stopień bezpieczeństwa przy tego typu komunikatach.

4.3. Model bazy danych

System korzysta z zewnętrznej bazy danych, której model jest przedstawiony na rysunku 4.4. Został on wygenerowany za pomocą testowej wersji programu DbSchema. Kolorem zielonym zostały oznaczone tabele dotyczące użytkowników oraz uprawnień. Niebieski kolor oznacza tabele złączeniowe, które nie mają odwzorowania w kodzie. Czerwone zaś to pozostałe struktury występujące w projekcie – są to obiekty na których operują użytkownicy. W pozostałej części pracy opiszę jak został wygenerowany model bazy danych.



Rys. 4.4. Szczegółowy diagram wybranych klas

Bibliografia

[1] JavaServer Faces i Eclipse Galileo. Tworzenie aplikacji WWW *Andrzej Marciniak* Helion 2010

[2] Core JavaServer Faces. Wydanie II *David Geary, Cay S. Horstmann* Helion 2008

[3] Enterprise JavaBeans 3.0 *Bill Burke & Richard Monson-Haefel* Helion 2007

[4] JBoss AS 7. Tworzenie aplikacji *Francesco Marchioni* Helion 2014

[5] Core J2EE. Wzorce projektowe *Deepak Alur, John Crupi, Dan Malks* Helion 2004

[6] Scrum. O zwinnym zarządzaniu projektami. Wydanie II rozszerzone *Mariusz Chrapko* Helion 2015

[7] Java. Kompedium programisty. Wydanie VIII *Herbert Schildt* Helion 2012

[8] Zwinne wytwarzanie oprogramowania. Najlepsze zasady, wzorce i praktyki *Robert C. Martin* Helion 2015

[9] Git. Rozproszony system kontroli wersji *Włodzimierz Gajda* Helion 2013

[10] Mistrz czystego kodu. Kodeks postępowania profesjonalnych programistów *Robert C. Martin* Helion 2013

[11] Czysty kod. Podręcznik dobrego programisty *Robert C. Martin* Helion 2010

[12] Rusz głową! Wzorce projektowe *Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra* Helion 2011

[13] Refaktoryzacja do wzorców projektowych *Joshua Kerievsky* Helion 2005