Uniwersytet Zielonogórski

Wydział Informatyki, Elektrotechniki i Automatyki

Kierunek: *Informatyka*

Specjalność: *Sieciowe systemy informatyczne*

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

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:

Celem pracy jest wytworzenie systemu, który będzie umożliwiał intuicyjne zarządzanie projektem.

Zakres Pracy:

1. Opracowanie wymagań funkcjonalnych i niefunkcjonalnych dla projektowanego systemu,
2. Wytworzenie projektu i implementacja systemu,
3. Wykonanie testów aplikacji,
4. Wytworzenie dokumentacji i zredagowanie części opisowej pracy.

Data podjęcia tematu: 18.03.2015

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, luty 2016

Piotr Joński
431 IDZ
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 metodyce 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 otwarty szkielet aplikacyjny Primefaces. System został wdrożony na serwerze aplikacyjnym 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

Spis listingów

Rozdział 1

Wstęp

1.1. Wprowadzenie

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 i rozwiązań. Do najpopularniejszych rozwiązań dotyczących zarządzania projektami można zaliczyć m.in.: Scrum, Lean Software Development (LSD), Feature Driven Development (FDD) czy też Test Driven Development (TDD).

Nie jest to wyczerpująca lista metodyk – istnieje wiele innych metod tworzenia oprogramowania, jednak opis wszystkich wykracza poza zakres tej pracy. Wszystkie z nich zaliczają się do tzw. metodyk zwinnych, które charakteryzują wspólne cechy takie jak: prowadzenie historyjek użytkownika, planowanie interwałów i podsumowanie ich w postaci retrospektywy oraz wytwarzanie oprogramowania w sposób przyrostowy. 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.

Obecnie na rynku istnieje mnóstwo narzędzi do tego typu zadań, a liczba oferowanych szkoleń z zakresu zarządzania projektami oraz metodyki Scrum nie ma końca. Coraz więcej firm decyduje się na zakup drogich systemów wspomagających proces zarządzania projektami. Ale czy zakup takiego programu jest konieczny? W mojej pracy przedstawię system, który bez żadnych przeszkód mógłby być stosowany w niewielkich zespołach, czy firmach, które nie zamierzają wydawać fortuny na profesjonalne aplikacje.

1.2. Czym właściwie jest metodyka Scrum

Scrum jest zwinną metodyką wytwarzania oprogramowania, której początki sięgają połowy lat 80 [1]. Polega ona na przyrostowym wytwarzaniu produktu oraz stałej komunikacji z klientem bądź też jego reprezentantem - właścicielem produktu (*ang. product owner*). Zespół deweloperski (*ang. 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 właściciele produktu, 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:

- **właściciel produktu** (*ang. product owner*) - osoba odpowiedzialna za projekt,
- **scrum master** - osoba, która pomaga zespołowi w organizacji czasu pracy a także rozwiązuje powstałe problemy,
- **zespół deweloperski** (*ang. team*) - zespół programistów wspólnie pracujących nad wytworzeniem produktu,
- **rejestr produktu** (*ang. backlog*) - jest to zbiór zadań / funkcjonalności wchodzących w skład wytwarzanego projektu,
- **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,
- **historyjka** (*ang. story*) - najmniejsza jednostka podlegająca ocenie (wg. wybranej przez zespół skali),
- **zadanie** (*ang. task*) - zadania, które mogą być powiązane bezpośrednio z historyjką lub projektem,
- **retrospektywa** (*ang. retrospective*) - wydarzenie, które ma miejsce na koniec interwału. W tym momencie zespół deweloperski spisuje wszystkie wady i zalety minione-

go sprintu oraz wspólnie z scrum masterem starają się rozwiązać zaistniałe problemy. Jest to jeden z ważniejszych elementów Scruma.

1.3. Przegląd dostępnych narzędzi

Opracowując założenia projektowe dla realizowanego systemu skupiłem się, na tym, by jak najbardziej 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 wdrażać w chmurze (*ang. hosting*), gdyż potrzebują dużo płatnych zasobów.

Ostatnim, jednak nie mniej ważnym problemem jest ich czytelność. Oferują one szereg zbędnych zwykłemu użytkownikowi - programiście - funkcjonalności, która jest przydatna tylko w określonych warunkach. Nadmiar zakładek i 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. Nadałem mu nazwę **scrumus**, która ma charakter gry słownej – scrum us – w wolnym tłumaczeniu "wyskramuj nas".

Tab. 1.1. Porównanie wybranych systemów zarządzania projektami

Cecha systemu	Wybrane systemy			
	JIRA	Bitbucket	GitHub	scrumus
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 wersja lite

1.4. 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 przedstawionych w tabeli 1.1, 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 uruchamianie systemów,
2. przejrzysty interfejs użytkownika, który został osiągnięty dzięki darmowej bibliotece Primefaces,
3. bez zbędnej funkcjonalności - zostały zaimplementowane tylko obligatoryjne funkcje tego typu systemu oraz niewielka ilość udogodnień.

Praca w swoim zakresie obejmuje:

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

1.5. Struktura pracy

W rozdziale drugim przedstawiona zostanie analiza problemu oraz podstawowe założenia projektowe. Opisane zostaną również technologie, które były stosowane podczas implementacji i wdrażania projektu.

W trzecim rozdziale znajduje się dokumentacja projektowa z zawartymi przykładowymi implementacjami danych funkcjonalności oraz konfiguracji, uzupełniona o schematy stanów oraz diagramy UML.

W rozdziale czwartym przedstawiona zostanie konfiguracja z użyciem systemu Docker oraz analiza pracy pod względem jakości wraz ze statystykami i podsumowaniem osiągniętych celów.

Ostatnim rozdziałem jest podsumowanie zawierające ogólne wnioski z realizacji pracy oraz wskazówki dotyczące rozwoju projektu w architekturze mikroserwisów.

Rozdział 2

Analiza biznesowa problemu i założenia projektowe

Prezentowany przeze mnie system ma pewne założenia oraz wymagania. W tym rozdziale zajmę się opisem użytkowników, wymagań funkcjonalnych, niefunkcjonalnych oraz użytych technologii.

2.1. Role i użytkownicy systemu

W projektowanym systemie występują cztery rodzaje użytkowników:

1. Administrator - posiada największe uprawnienia w systemie,
2. Właściciel produktu - odzwierciedla rolę właściciela produktu w Scrumie,
3. Scrum master - odzwierciedla rolę scrum mastera w Scrumie,
4. Deweloper - posiada najmniejsze uprawnienia, jest częścią zespołu deweloperskiego.

Na tym etapie warto wspomnieć, że każdy użytkownik 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.

2.2. Wymagania funkcjonalne

We wcześniejszym podpunkcie zostały omówione role w systemie. Każda z tych ról ma pewne uprawnienia lub restrykcje. Każda taka cecha zostanie przedstawiona jako wy-

maganie funkcjonalne prezentowanego systemu.

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

> *Historyjka jest jednostką funkcjonalności w projektach XP (ang. eXtreme Programming). 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 trakcie 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*>[1]

Administrator

- jako administrator mogę tworzyć nowych użytkowników w celu dodania ich do systemu,
- jako administrator mogę usuwać użytkowników z systemu z wyjątkiem siebie samego,
- jako administrator mogę dodawać użytkowników do zespołu w celu modyfikacji zespołu,
- jako administrator mogę usuwać użytkowników z zespołu w celu modyfikacji zespołu,
- jako administrator mogę nadać uprawnienia administratora dowolnemu użytkownikowi,
- jako administrator mogę odebrać uprawnienia administratora dowolnemu administratorowi,
- jako administrator mogę przypisać właściciela produktu do projektu,
- jako administrator mogę usunąć właściciela produktu z projektu,

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

- jako administrator mogę utworzyć projekt w celu dodania go do systemu,
- 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,
- jako administrator mogę modyfikować projekt,
- jako administrator mogę tworzyć nowe zespoły w celu dodania ich do systemu,
- jako administrator mogę dodawać zespoły do projektów w celu przydzielenia uprawnień,
- jako administrator mogę usuwać zespoły z projektów w celu odebrania uprawnień,
- jako administrator mogę przypisać scrum mastera do zespołu,
- jako administrator mogę usunąć scrum mastera z zespołu,
- jako administrator mogę tworzyć, usuwać oraz edytować statusy zadań,
- jako administrator mogę tworzyć, usuwać oraz edytować priorytety zadań,
- jako administrator mogę tworzyć, usuwać oraz edytować typy zadań.

Właściciel produktu

- jako product owner mam wgląd w projekt, w którym jestem product, ownerem
- jako product owner mogę tworzyć zadania w projekcie, w którym jestem product ownerem.

Scrum master

- jako scrum master mogę tworzyć retrospektywy dla sprintów,
- jako scrum master mogę przenosić zadania ze story do backlogu,
- jako scrum master mogę przenosić zadani z backlogu do story,
- jako scrum master mogę tworzyć story w sprincie,
- jako scrum master mogę tworzyć sprint w projekcie.

Deweloper

- jako deweloper mogę tworzyć zadania,
- jako deweloper mogę usuwać zadania,
- jako deweloper mogę dodawać komentarze do zadań,
- jako deweloper mogę dodawać komentarze do retrospektyw,
- jako deweloper mogę edytować swój profil,
- jako deweloper mogę zmienić swoje hasło,
- jako deweloper mogę przeglądać profile innych użytkowników,
- jako deweloper mogę przeglądać wszystkie projekty, do których jestem przypisany,
- jako deweloper mogę przypisać zadanie do dowolnego użytkownika,

2.3. 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 systemu,
    - przejrzystość - intuicyjne poruszanie się po stronach projektun
    - 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 - obsługa wielu użytkowników na raz,
   - mieć dużą responsywność - czas reakcji interfejsu użytkownika powinien być jak najkrótszy,
   - działać szybko - wszystkie procesy biznesowe oraz akcje działające w tle powinny wykonywać się możliwie w jak najkrótszym czasie. Ponadto, komunikacja z bazą danych powinna być niezauważalna dla użytkownika.
5. **S**upportability - wsparcie, do którego należy:
   - prostota w instalacji,
   - łatwość konfiguracji,
   - adaptowalność, czyli możliwość zaadoptowania systemu do innych warunków,
   - testowalność, czyli testy jednostkowe oraz integracyjne.

2.4. Użyte technologie i narzędzia

Przy implementacji systemu scrumus wykorzystano wiele gotowych rozwiązań, które bez wątpienia podniosły wydajność pracy poprzez automatyczne budowanie, czy wdrażanie projektu na serwer. Również jakość aplikacji utrzymuje się na bardzo dobrym poziomie dzięki wielu testom jednostkowym, integracyjnym i w końcu testom użytkownika końcowego.

Dodatkowo przy implementowaniu systemu zostały wykorzystane narzędzia, które oferuje pakiet Student Developer Pack[2]. Aby uzyskać możliwość używania narzędzi z tego pakietu należało wysłać żądanie przydzielenia do tego projektu. Weryfikacja była oparta na uczelnianym adresie e-mail. Znalazł się tam między innymi system GitHub w wersji Micro z możliwością prowadzenia prywatnych repozytoriów.

2.4.1. Testowanie

Jak wiadomo w dzisiejszych czasach aplikacje, które nie posiadają testów, jak i również te, które testów nie przechodzą, nie mają prawa znaleźć się na serwerze produkcyjnym.

[2]Projekt Student Developer Pack dotyczy wyłącznie studentów. Więcej informacji można znaleźć pod adresem `https://education.github.com/pack/offers`

Co prawda przy projektowaniu mojej aplikacji serwer produkcyjny jak i testowy stanowił jedność, jednak bez testów obyć się nie mogło.

2.4.1.1. JUnit

JUnit jest frameworkiem, który umożliwia pisanie testów jednostkowych [2]. Stosuje się go do aplikacji napisanych w języku Java. Jego działanie sprawia, że uruchomienie testów trwa parę sekund, dzięki czemu programista może je uruchamiać na bieżąco, przez co jakość kodu jest stale monitorowana. Framework ten jest doskonale zintegrowany z systemami budowania projektów takimi jak Gradle czy Maven.

2.4.1.2. PowerMock

Jest to framework rozszerzający możliwości zwykłych testów jednostkowych. Oferuje on mockowanie klas, tzn. przykrywanie ich funkcjonalności na potrzeby testów, przez co możemy przykryć np. komunikację z bazą danych i zwrócić własne rezultaty. Takie możliwości są bardzo ważne przy pisaniu testów jednostkowych, które mają wykonywać się błyskawicznie i nie mogą polegać na zewnętrznych systemach t.j. bazy danych.

2.4.1.3. AssertJ

Jest to kolejny framework na potrzeby testów. Jego zaletą jest to, iż pozwala on tworzyć czytelne dla programisty asercje oraz ma możliwość szerokiej rozbudowy. Pozwala on na sprawdzenie wyników testów w jednej linii, co ma duży wpływ na czytelność.

2.4.1.4. Arquillian

Arquillian jest kolejnym frameworkiem do testowania, tym razem jednak integracyjnego [3]. Posiada on wsparcie dla wielu kontenerów aplikacji JEE, co czyni go pionierem pod tym względem. Dzięki jego mechanizmom możliwe jest uruchomienie testów bez konieczności posiadania serwera aplikacji, co znacznie przyśpiesza wykonywanie testów integracyjnych. Istnieje również możliwość uruchamiania testów na zdalnym serwerze i to właśnie tą ścieżkę wybrałem, ze względu na większe możliwości testowania oraz to, że aplikacja jest uruchamiana na żywym środowisku, przez co umożliwia to sprawdzenia zachowania aplikacji dokładnie na takim serwerze, na jakim będzie uruchomiona wersja produkcyjna. Dodatkową zaletą tego frameworku jest to, że nie musimy wdrażać całej aplikacji. Jeżeli testujemy tylko jedną klasę, która nie ma żadnych powiązań lub zależności to wystarczy,

że wdrożymy na serwerze tylko tę klasę.

2.4.2. Ciągła integracja oraz zarządzanie projektem

Koleje narzędzia z grupy ciągłej integracji umożliwiają budowanie, testowanie oraz monitorowanie stanu projektu na każdym etapie jego powstawania. W dzisiejszych czasach każda firma, a nawet osoby prywatne, posiada takie systemy, bez których nie byłoby możliwe tak szybkie wytwarzanie aplikacji.

2.4.2.1. Gradle

Gradle jest darmowym narzędziem do budowania projektów oraz zarządzania zależnościami, przez co programista może skupić się na pracy, a wszystkimi bibliotekami oraz zadaniami jakie należy wykonać podczas budowania zajmie się Gradle. W przeciwieństwie do Maven został on stworzony przy użyciu języka Groovy, co sprawia, że nie ma rzeczy niewykonalnych. Jego prostota i zarazem ogrom możliwości sprawiają, że powoli zastępuje on przestarzały framework Maven. Dzięki temu projektowi za pomocą jednej komendy skompilujemy, zbudujemy oraz uruchomimy testy naszej aplikacji. Gradle został przyswojony przez prawie każdy system budowania projektów, przez co był oczywistym wyborem przy planowaniu projektu.

2.4.2.2. GitHub

GitHub jest serwisem, który umożliwia prowadzenie projektów z użyciem systemu kontroli wersji - Git. Git jest rozproszonym systemem kontroli wersji [4]. Umożliwia on prowadzenie projektu w sposób przejrzysty i efektowny. Każda nasza zmiana jest rejestrowana w tym systemie z możliwością jej przywrócenia bądź też cofnięcia, przez co nie musimy obawiać się, że coś nam ucieknie. Dzięki pakietowi Student Developer Pack, otrzymałem rozszerzone konto GitHub, które zawiera do 5 prywatnych repozytoriów. Narzędzie to pozwoliło mi utrzymać kod w prywatności przed światem zewnętrznym jak i również udostępnić go dla promotora.

2.4.2.3. TeamCity

TeamCity jest potężnym narzędziem do budowy projektów, którego rozwojem zajmuje się firma JetBrains. Jest on darmowy dla małych projektów i w skład tej wersji wchodzi jeden agent, który wykonuje zadania. System umożliwia budowanie oraz testowanie aplika-

cji, generując przy tym raporty, przez co wszystkie błędy zostaną znalezione i poprawione. Ma on świetną integrację z systemami kontroli wersji takimi jak Git czy SVN oraz innymi narzędziami tej firmy, które zostały również wykorzystane podczas prac. Rysunek 2.1 przedstawia historię budowań projektu wraz z całkowitą liczbą testów. Dodatkowo prezentowane są informacje, które testy nie przeszły, czas budowy oraz wiele innych przydatnych informacji.

Branch	#	Results	Artifacts	Changes	Started	Duration
SCRUMUS-42-Akc...a-scrum-mastera	#178	Tests failed: 1, passed: 363	None	Piotr Joński (3)	29 Jan 16 21:31	7m:39s
SCRUMUS-42-Akc...a-scrum-mastera	#177	Tests failed: 1 (1 new), passed: 363	None	Piotr Joński (3)	29 Jan 16 19:45	7m:15s
poprawki-promotora	#176	Tests passed: 364	None	katiamar63 (1)	29 Jan 16 12:35	7m:20s
poprawki-promotora	#175	Tests passed: 364	None	Piotr Joński (10)	29 Jan 16 12:26	8m:11s
SCRUMUS-42-Akc...a-scrum-mastera	#174	Tests passed: 364	None	Piotr Joński (1)	28 Jan 16 20:20	7m:33s
SCRUMUS-42-Akc...a-scrum-mastera	#173	Tests passed: 364	None	Piotr Joński (1)	28 Jan 16 14:09	7m:34s
SCRUMUS-42-Akc...a-scrum-mastera	#172	Tests passed: 364	None	Piotr Joński (2)	27 Jan 16 19:28	7m:33s
SCRUMUS-42-Akc...a-scrum-mastera	#171	Tests passed: 364	None	No changes	27 Jan 16 18:09	7m:48s
master	#170	Tests passed: 364	None	Piotr Joński (14)	27 Jan 16 18:02	7m:16s
SCRUMUS-45-Zainicjowac-latexa	#169	Tests passed: 364	None	Piotr Joński (1)	27 Jan 16 17:54	7m:36s
SCRUMUS-45-Zainicjowac-latexa	*#168*	*Canceled (Tests failed: 5 (5 new), passed: 0)*	*None*	*Piotr Joński (1)*	*27 Jan 16 17:53*	*46s*
SCRUMUS-45-Zainicjowac-latexa	#167	Tests passed: 364	None	Piotr Joński (2)	27 Jan 16 13:24	6m:51s
SCRUMUS-45-Zainicjowac-latexa	#166	Tests passed: 364	None	Piotr Joński (2)	27 Jan 16 10:51	7m:19s
SCRUMUS-45-Zainicjowac-latexa	#165	Tests passed: 364	None	Piotr Joński (8)	26 Jan 16 17:45	7m
SCRUMUS-45-Zainicjowac-latexa	#164	Tests passed: 364	None	No changes	26 Jan 16 12:24	7m:04s
master	#163	Tests passed: 364	None	Piotr Joński (23)	26 Jan 16 12:17	7m:09s

Rys. 2.1. Historia budowań projektu w systemie TeamCity

2.4.2.4. YouTrack

YouTrack jest systemem do zarządzania projektami, którego funkcjonalność jest bardzo zbliżona do wcześniej przedstawionego systemu Jira. Jest on dedykowany ogólnie metodykom zwinnym, między innymi Scrumowi. Używałem go podczas implementacji aplikacji, co pozwoliło mi kontrolować zadania, które należy wykonać. Aplikacja umożliwia również generowanie raportów i wykresów dot. zadań, użytkowników czy projektów. System jest również rozwijany przez firmę JetBrains, przez co jest zintegrowany z TeamCity oraz IntelliJ. Pobiera on informacje o budowach projektów z systemu TeamCity oraz umożliwia przeglądania zadań w programie IntelliJ, przez co znacznie upraszcza pracę. Dodatkowo jest darmowy dla małych projektów (do 10 osób), przez co może być wykorzystywany przez małe firmy lub studentów.

2.4.2.5. IntelliJ

IntelliJ jest zintegrowanym środowiskiem deweloperskim na bardzo wysokim poziomie. Umożliwia on tworzenie, kompilację oraz testowanie kodu napisanego w języku Java lub innych językach, które działają na wirtualnej maszynie Javy np. Scala lub Groovy. Narzędzie to posiada szereg funkcjonalności ułatwiających i przyśpieszających programo-

wanie. Jest ono świetnie zintegrowane z frameworkami do testowania (JUnit, Arquillian), systemami do budowania (Gradle, TeamCity), czy zarządzania (YouTrack) przez co programista może się skupić wyłącznie na jednym oknie - oknie aplikacji IntelliJ. Ten system również jest darmowy dla studentów, co stwarza świetne możliwości dla tej grupy osób.

2.4.3. Warstwa prezentacji

Pod pojęciem warstwa prezentacji mam na myśli zarówno część wizualną jak i różne technologie i wzorce pozwalające na budowę własnych aplikacji prezentacji dla istniejącego już systemu. Przy projektowaniu interfejsu WWW pomogły frameworki JSF oraz PrimeFaces, zaś do umożliwienia rozbudowy systemu o dodatkowe aplikacje klienckie odpowiadają przykładowe endpointy w postaci RESTów.

2.4.3.1. JSF

JSF jest frameworkiem, który umożliwia tworzenia warstwy prezentacji dla aplikacji webowych napisanych w języku Java [5]. Posiada on szereg udogodnień, dzięki którym programowanie części wizualnej staje się proste. Framework umożliwia stworzenie warstwy oddzielającej logikę biznesową od widoku, co jest pożądanym, a wręcz wymaganym aspektem programowania aplikacji webowych.

2.4.3.2. PrimeFaces

PrimeFaces jest darmowym frameworkiem, który posiada szereg dostępnych z półki kontrolek służących do obsługiwania widoku. Dodatkowo wprowadza wiele usprawnień, takich jak generowanie tabel z danymi czy szablony css, przez co możemy zmienić wygląd naszej aplikacji za pomocą jednej linii kodu. Został on napisany jako częściowa nakładka na JSF. Wszystkie żądania potrafi obsługiwać synchronicznie i asynchronicznie dzięki zastosowaniu JavaScriptu.

2.4.4. Pozostałe

W tej sekcji zostaną omówione pozostałe aplikacje, frameworki oraz język Java EE.

2.4.4.1. Java Enterprise Edition

Język Java powstał w roku 1995. Jego głównymi założeniami są m.in. niezależność od architektury oraz bezpieczeństwo i niezawodność [6]. Język ten jest dostępny w trzech

podstawowych pakietach:

- Java Micro Edition - służy do wytwarzania aplikacji na urządzenia, które posiadają niewielkie parametry systemowe. Obecnie wraz z rozwojem telefonów komórkowych i innych urządzeń mobilnych, powstaje coraz mniej aplikacji wykorzystującej ten pakiet.

- Java Standard Edition - jest podstawowym pakietem Javy, który można znaleźć na każdym komputerze z zainstalowaną Javą. Oferuje on możliwości programowania sieciowego i współbieżnego. Powstałe aplikacje mogą mieć postać programów uruchamialnych z linii komend lub też pełnoprawnych aplikacji desktopowych.

- Java Enterprise Edition - jest to najszerszy pakiet Javy, który oferuje możliwość tworzenia aplikacji webowych z wykorzystaniem różnych technologii takich jak JSF czy EJB [7][8]. Jest obecnie jednym z najbardziej popularnych narzędzi wykorzystywanych przy implementacji projektów serwerowych.

2.4.4.2. WildFly

WildFly jest darmowym kontenerem aplikacji, rozwijanym przez firmę RedHat [9]. Jest on w pełni zgodny ze standardem EJB. Serwer posiada wsparcie dla baz danych oraz zabezpieczeń aplikacji na poziomie kontenerów. Implementuje również specyfikację JAAS (Java Authentication and Authorization Service), dzięki której zapewnione jest bezpieczeństwo w systemie scrumus. Posiada on bardzo dobre parametry takie jak czas startu serwera - poniżej dwóch sekund, czy czas wdrożenia aplikacji - poniżej dziesięciu sekund.

2.4.4.3. Docker

Jest to narzędzie, które w ostatnim czasie zwróciło uwagę wielu firm i indywidualnych programistów. Co prawda koncepcja kontenerów sięga lat 80-tych, jednak dopiero w dobie mikrousług system ten miał możliwość upowszechnienia się. Pozwala on na budowanie tzw. obrazów opartych na systemach z rodziny UNIX [10]. Z takich obrazów można tworzyć kontenery, skalować je i zarządzać nimi. System daje możliwość zainstalowania dowolnych aplikacji, przy niewielkiej konfiguracji, co pozwala uruchomić wytworzony w ramach pracy system wraz z bazą danych za pomocą jednej komendy w czasie równym uruchomieniu serwera i wdrożenia aplikacji.

2.4.4.4. Postgresql

Postgresql jest darmową bazą danych o dużych możliwościach. Ma on niewielkie wymagania systemowe oraz oferuje wsparcie dla transakcji. Umożliwia przechowywanie danych o różnych typach i posiada szereg usprawnień optymalizacyjnych. Dodatkowo można definiować własne typy danych.

2.4.4.5. LaTeX

LaTeXjest oprogramowaniem do składania tekstu. Posiada szereg funkcjonalności takich jak automatyczne tworzenie spisów treści, tabel czy ilustracji. W łatwy sposób można utworzyć bibliografię i skorowidze. LaTeXpozwala autorowi odizolować się do wyglądu i formatowania dokumentu, a skupić jedynie na treści i strukturze tekstu. Został on użyty do wytworzenia części opisowej tejże pracy dyplomowej, co znacznie ułatwiło pracę.

2.4.4.6. DbSchema

Jest to program, który pozwala na wizualizację bazy danych oraz zmienianie jej struktury z poziomu programu, bez konieczności wykonywania skryptów SQL. Za pomocą tego programu został wygenerowany model bazy danych przedstawiony w dalszej części tej pracy.

2.4.4.7. Lombok

Framework Lombok służy do automatycznego generowania kodu Javy. Pozwala on za pomocą jednej adnotacji wstawić podstawowe metody do klasy takie jak gettery i settery, konstruktory czy też hashcode i equals. Dzięki temu projektowi nasze klasy zostaną znacznie odchudzone przez co zyskują na czytelności.

Rozdział 3

Dokumentacja projektowa

3.1. Architektura systemu

Wytworzony w ramach pracy 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 (*ang. Representational State Transfer*). 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 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 on jednak ograniczony tylko do klientów napisanych w języku Java i protokołu RMI (*ang. Remote Method Invocation*).

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 te wymagania, lecz w tym celu posłużymy się, specjalnie zaprojektowaną do tego rodzaju zadań, technologią JavaServer Faces (JSF). Rozwiązanie to pozwoli nam zaimplementować dodatkową warstwę, dedykowaną specjalnie dla interfejsu WWW. Warstwa ta to tzw. komponenty Java Beans zwane również Backing Beans (BB). Rysunek 3.1 przedstawia 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 być przekazane do klienta. Również w aplikacji WWW walidacja danych wejściowych powinna znajdować 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 FacesMessage i dodawane bezpośrednio do kontekstu aplikacji WWW.

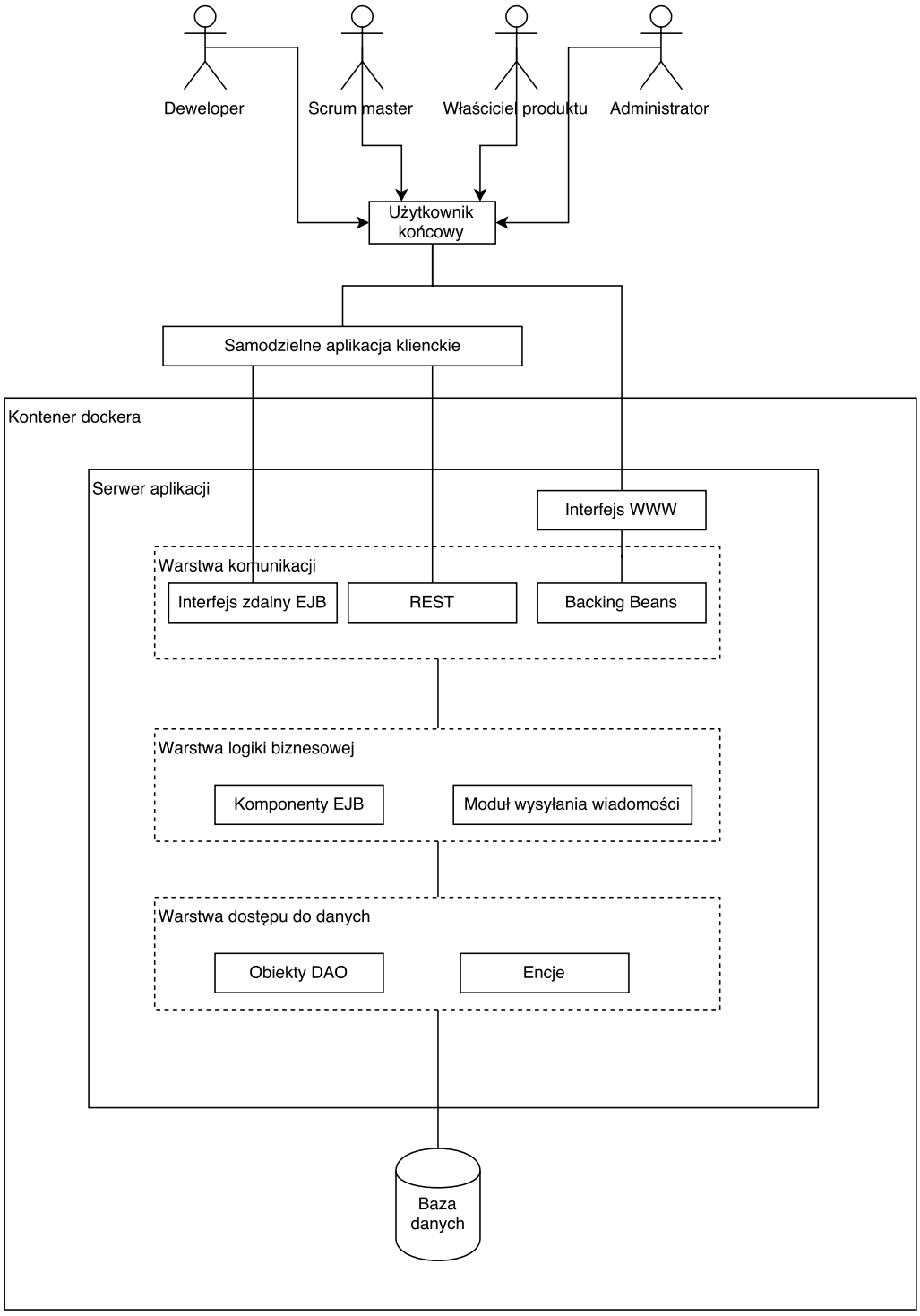
Wymagania odnośnie walidacji oraz obsługi błędów mogą skłaniać do wprowadzenia dodatkowej warstwy w architekturze systemu. Niemniej jednak taka warstwa nie została wprowadzona, gdyż spowodowałoby to nadmierny przyrost klas oraz niepotrzebne uogólnienie systemu. Zamiast tego logika biznesowa generuje 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.

3.1.1. 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*.



Rys. 3.1. Diagram systemowy aplikacji

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*.

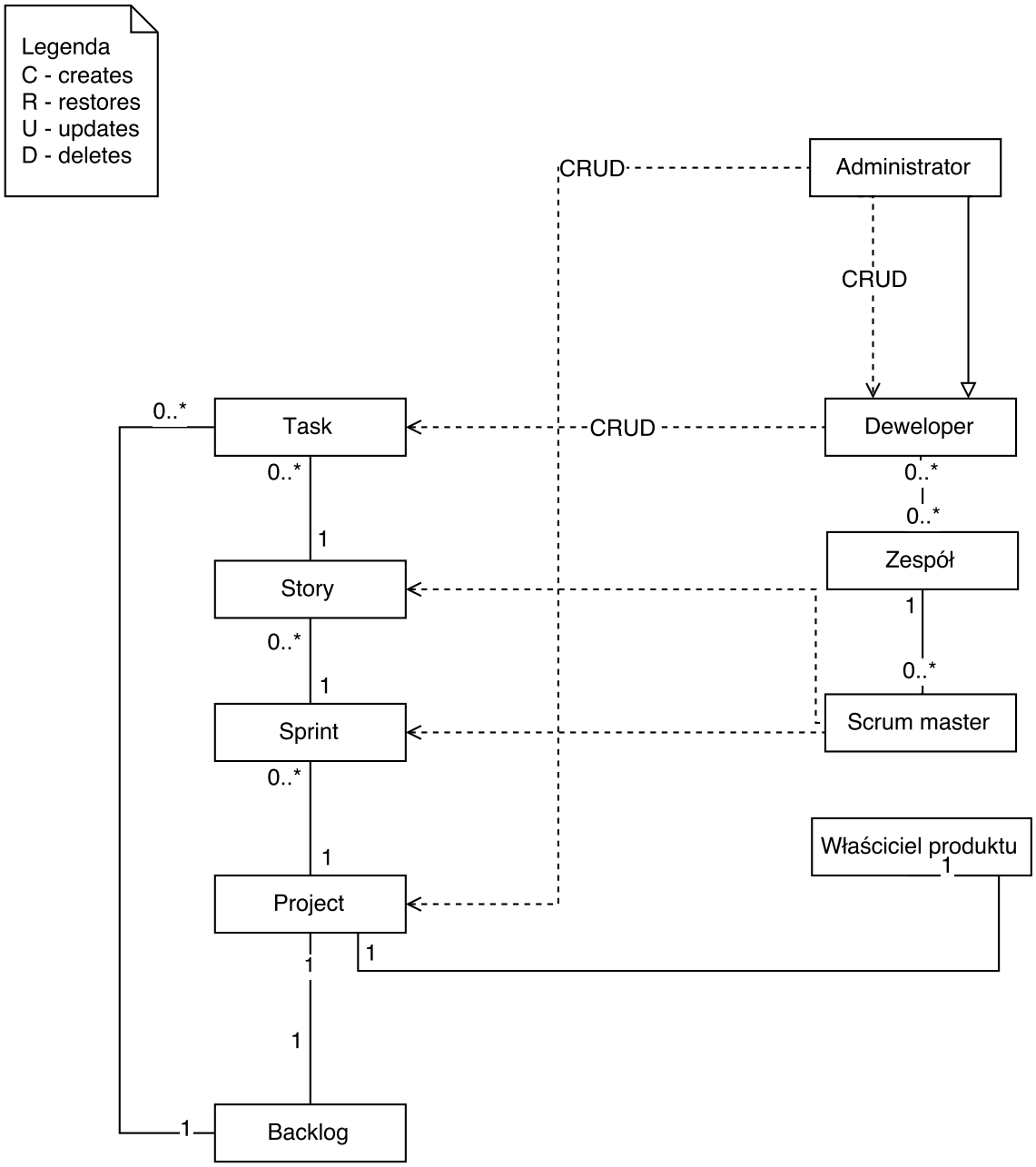
Sprint – jest tworzony przez *scrum mastera*. Posiada *story*.

Story – jest agregatem *zadań*.

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

Zadanie – jest tworzone przez *użytkowników*.

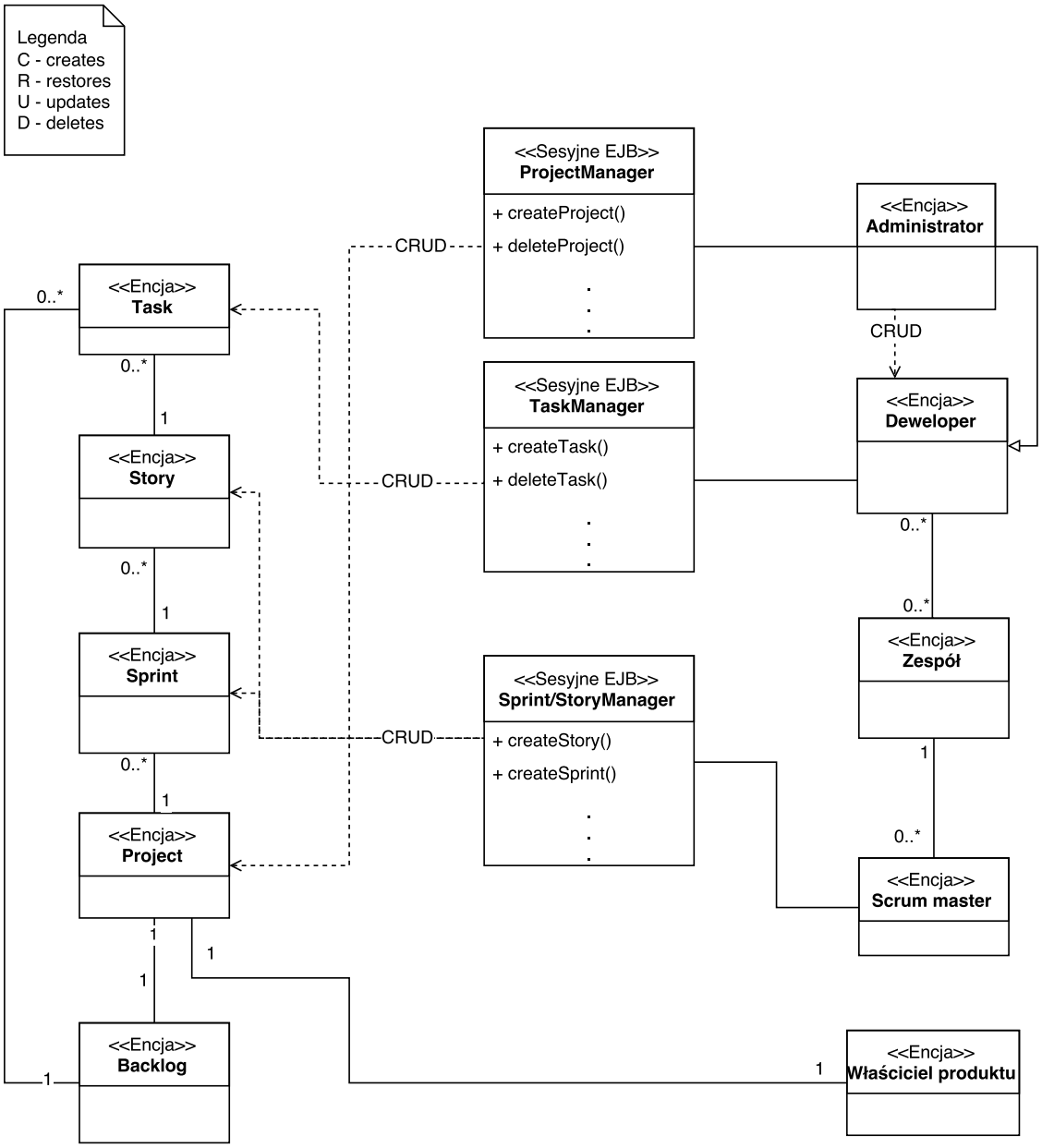
Należy wspomnieć, iż jest to częściowy opis obiektów biznesowych, który ma na celu zobrazowanie procesu powstawania aplikacji. Wyczerpujący opis tych obiektów byłby zbyt rozległy i nie stanowi on meritum części opisowej pracy. Wpływ poszczególnych jednostek biznesowych oraz ich wzajemne relacje przedstawiono na rysunku 3.2 w postaci diagramu modelu pojęciowego UML.



Rys. 3.2. Diagram jednostek biznesowych

3.2. Struktura projektu

W poprzedniej sekcji opisany został model dziedzinowy (koncepcyjny) projektowanej aplikacji. W niniejszym punkcie zostaną opisane wybrane komponenty. Zanim jednak to zrobimy zostanie przedstawiony bardziej szczegółowy diagram jednostek biznesowych, który uwzględnia funkcje, jakie można wykonywać w systemie. Diagram z rysunku 3.3 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. 3.3. Szczegółowy diagram wybranych klas

3.2.1. 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.

3.2.1.1. 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.

3.2.1.2. 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 sesyjny. Ta reguła również tutaj ma swoje zastosowanie. Zostały utworzone odpowiednie komponenty sesyjne, które są swego rodzaju menedżerami 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.

3.2.1.3. Komponenty sterowane komunikatami

Pomimo założenia projektowego, iż 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 (*ang. Message Driven Bean*) do generowania i wysyłania e-maili z hasłem po utworzeniu użytkownika.

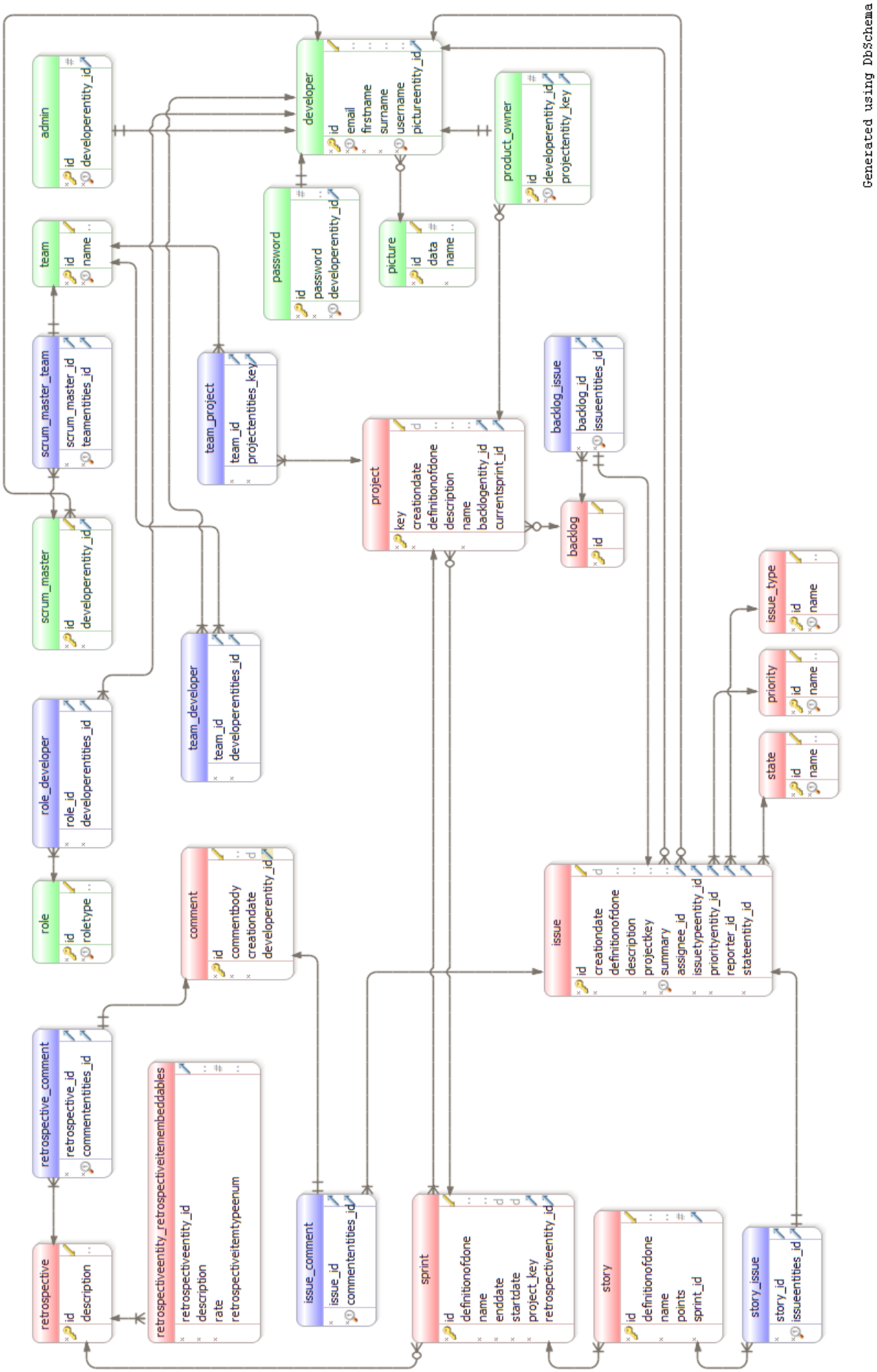
Taka wiadomość mogła by być rozgłoszona w systemie i dzięki temu różne komponenty mogłyby obsłużyć to zdarzenie. W systemie zrezygnowano jednak z tego typu technik.

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.

3.3. Schemat danych

System korzysta z zewnętrznej bazy danych, której schemat jest przedstawiony na rysunku 3.4. 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. Schemat danych został wygenerowany bezpośrednio z klas Javy, za pomocą frameworka Hibernate oraz specyfikacji JPA (*ang. Java Persistence API*).



Rys. 3.4. Model bazy danych projektu scrumus

3.4. Szczegóły implementacji

Przy omawianiu szczegółów implementacji przejdziemy przez cały zaprojektowany system w kierunku od bazy danych poprzez logikę biznesową, warstwę prezentacji a w następnym rozdziale przejdziemy do sposobu uruchomienia aplikacji wraz z opisem środowiska testowego i produkcyjnego, które w przypadku tego projektu stanowiły jedność. Zanim jednak przystąpimy do opisu poszczególnych warstw, w celu zrozumienia wszystkich omawianych zagadnień, warto zapoznać się ze strukturą plików i folderów w projekcie scrumus, która została przedstawiona na rysunku 3.5.

3.4.1. Baza danych i model encyjny

Baza danych i model encyjny został zaprojektowany w postaci klas Javy, tak zwanych encji, a następnie automatycznie wygenerowany za pomocą frameworka Hibernate. Dodatkowo zaraz po wdrożeniu aplikacji na serwer wykonywany jest skrypt SQL, który wprowadza do bazy testowe wartości i tworzy użytkowników z różnymi uprawnieniami. Przykładowy kod klasy encji został przedstawiony na listingu 3.1, a konfiguracja Hibernate na listingu 3.2.

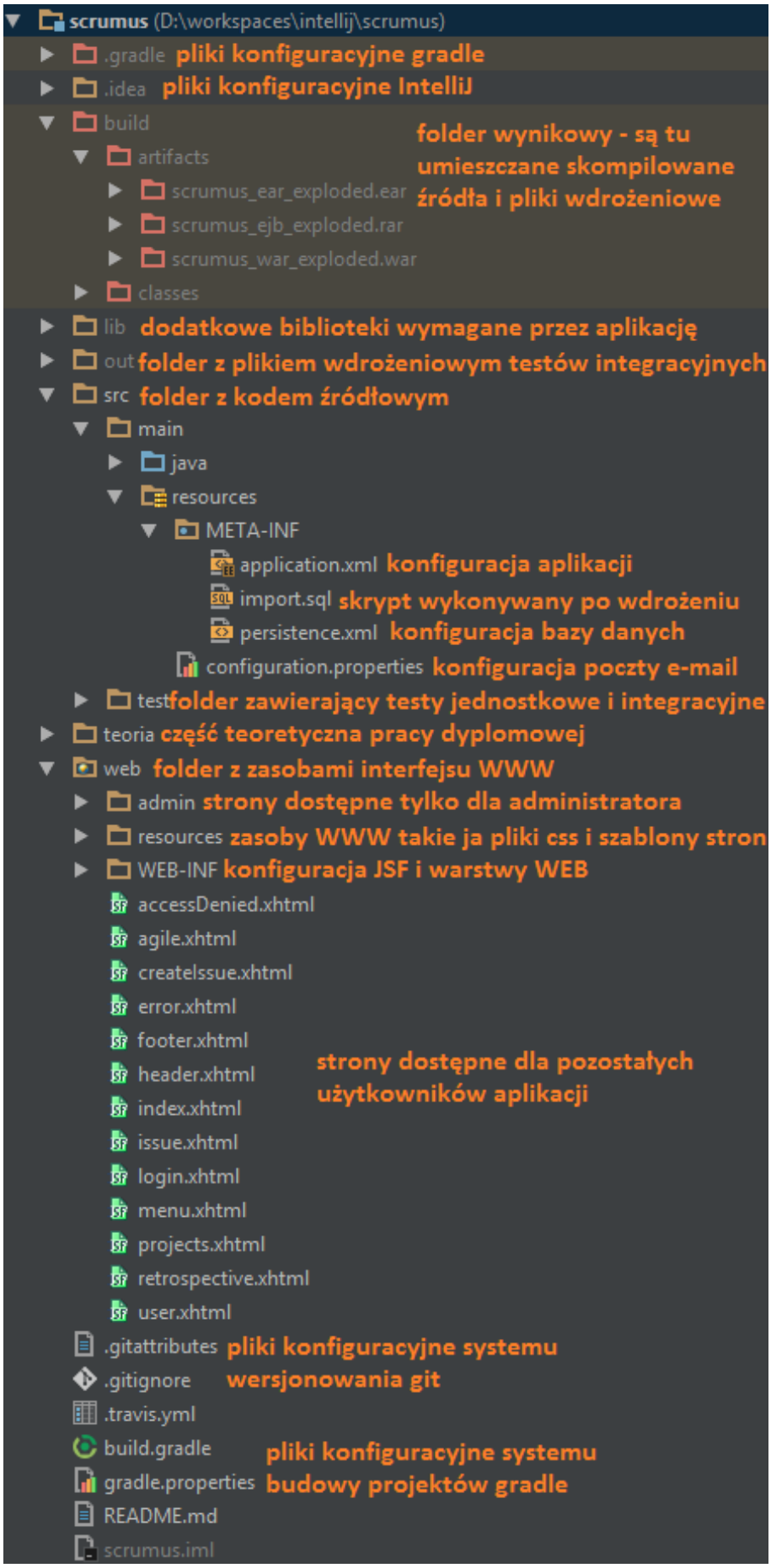
```
@Data                          // adnotacja lombok dodaje gettery
@NoArgsConstructor             // settery oraz hashcode i equals
@Entity                        // oznaczenie klasy jako encji
@Table(name = "project")       //nazwa tabeli w bazie danych
@NamedQueries({@NamedQuery(name = ProjectEntity.FIND_ALL,
                           query = ProjectEntity.FIND_ALL_QUERY),
               //pozostale zapytania uzywane przez DAO})
public class ProjectEntity {

// zapytania nazwane

// mapowanie
@Id
@Column(length = 8, nullable = false, unique = true)
private String key;

// pozostale pola
}
```

Listing 3.1. Przykładowa klasa encji



Rys. 3.5. Struktura plików i folderów projektu scrumus

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             version="2.0">
  <persistence-unit name="PostgresDS">
    <jta-data-source>java:jboss/datasources/PostgresDS</jta-data-source>
    <properties>
      <property name="hibernate.show_sql"
                value="false"/>
      <property name="hibernate.format_sql"
                value="true"/>
      <property name="hibernate.dialect"
                value="org.hibernate.dialect.PostgreSQLDialect"/>
      <property name="hibernate.hbm2ddl.auto"
                value="create-drop"/>
      <property name="javax.persistence.sql-load-script-source"
                value="import.sql"/>
      <property name="hibernate.connection.CharSet"
                value="utf8"/>
      <property name="hibernate.connection.characterEncoding"
                value="utf8"/>
      <property name="hibernate.connection.useUnicode"
                value="true"/>
      <property name="hibernate.event.merge.entity_copy_observer"
                value="allow"/>
    </properties>
  </persistence-unit>
</persistence>
```

Listing 3.2. Konfiguracja Hibernate

Kolejnym krokiem było zaprojektowanie warstwy DAO, która jest odpowiedzialna za pobieranie, utrwalania i usuwanie obiektów z bazy. Klasy obsługujące te żądania zostały zaprojektowane zgodnie ze wzorcem DAO [7][11], który pozwolił zaoszczędzić sporo kodu poprzez zastosowanie typów generycznych i klas abstrakcyjnych. Dodatkowo podczas wykonywanych operacji występuje mapowanie pomiędzy obiektami encji, które są przechowywane w bazie danych, a obiektami biznesowymi, które są bezpośrednio używane w warstwach wyższych. Takie rozgraniczenie jest zgodne z regułą OCP[1].

[1] R. C. Martin, *Czysty kod. Podręcznik dobrego programisty*, Helion 2010, s.160

3.4.2. Warstwa logiki biznesowej

W tej warstwie wykonują się wszystkie operacje biznesowe. Dzięki zastosowaniu EJB, poprzez ustanowienie klas jako ziaren sesyjnych, wprowadzone zostały transakcje, które umożliwiają obsługę błędnych operacji oraz uniemożliwiają wykonanie takiej operacji. To znaczy, że jeżeli chcemy dodać do bazy danych użytkownika, którego adres e-mail został już wykorzystany, wtedy cała transakcja zostaje cofnięta. Usuwane są również inne obiekty, które powstały na skutek tej operacji jak np. powiązane z użytkownikiem hasło. Zaletą tego rozwiązania jest to, że nie dopuszcza ono możliwości wprowadzenia błędnych danych do systemu. Dla przykładu zostanie omówiona operacja dodawania nowego użytkownika, której diagram sekwencji został przedstawiony na rysunku 3.6.

Na diagramie nie została uwzględniona operacja anulowania transakcji, ze względu na czytelność. Przerwanie procesu może nastąpić przy zapisywaniu użytkownika do bazy - błąd komunikacji, generowaniu hasła - brak algorytmu SHA-256, zapisywaniu hasła do bazy - błąd komunikacji lub też przy wysyłaniu e-maila z hasłem w sytuacji, gdy wprowadzimy złe ustawienia podczas procesu konfiguracji. W każdym z tych przypadków generowany jest odpowiedni wyjątek, który następnie jest obsługiwany przez warstwę JSF, a użytkownikowi zostaje wyświetlona stosowna informacja.

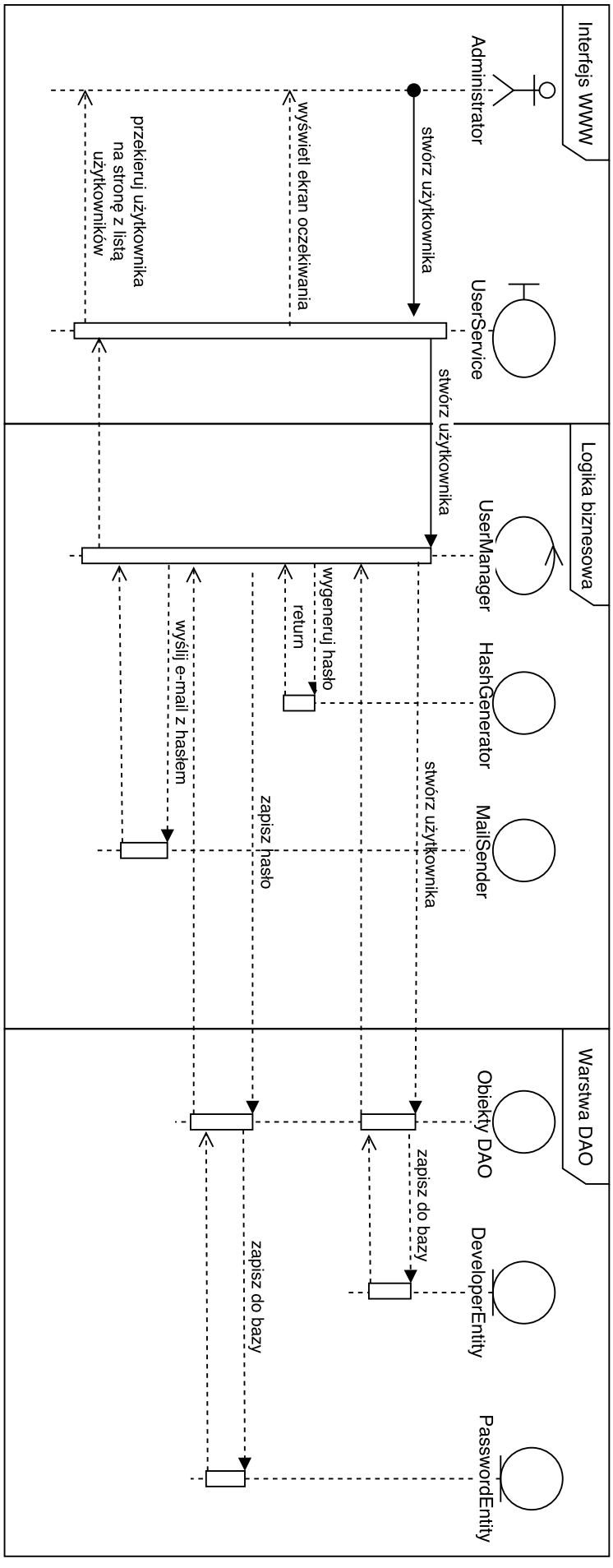
Generowanie haseł odbywa się poprzez klasę narzędziową UUID z pakietu java.util. Generuje ona co prawda numer UUID, lecz z powodzeniem można go wykorzystać jako hasło początkowe użytkownika, wystarczy tylko usunąć wszystkie znaki myślnika (-) z tak wygenerowanego ciągu znaków. Następnie hasło jest szyfrowane za pomocą algorytmu SHA256 i w takiej postaci jest zapisywane w bazie danych. Cały kod odpowiedzialny za generowanie i szyfrowanie hasła został zamieszony na listingu 3.3.

```
public class HashGenerator {

public String generateHash() {
return UUID.randomUUID().toString().replaceAll("-", "");
}

public String encodeWithSHA256(String text)
throws NoSuchAlgorithmException, UnsupportedEncodingException {
return String.format("%064x", new BigInteger(1,
MessageDigest.getInstance("SHA-256").update(text.getBytes("UTF-8")).digest());
}
}
```

Listing 3.3. Klasa generująca i szyfrująca hasła



Rys. 3.6. Diagram sekwencji przedstawiający proces tworzenia nowego użytkownika

3.4.3. Warstwa prezentacji

Ostatnią warstwą, która zostanie omówiona w tym rozdziale jest widok użytkownika, czyli warstwa prezentacji. Równolegle z tą warstwą idą w parze usługi REST, za pomocą których można komunikować się z systemem. Są jednak one rozwinięte w bardzo małym stopniu - jedynie w celach demonstracyjnych.

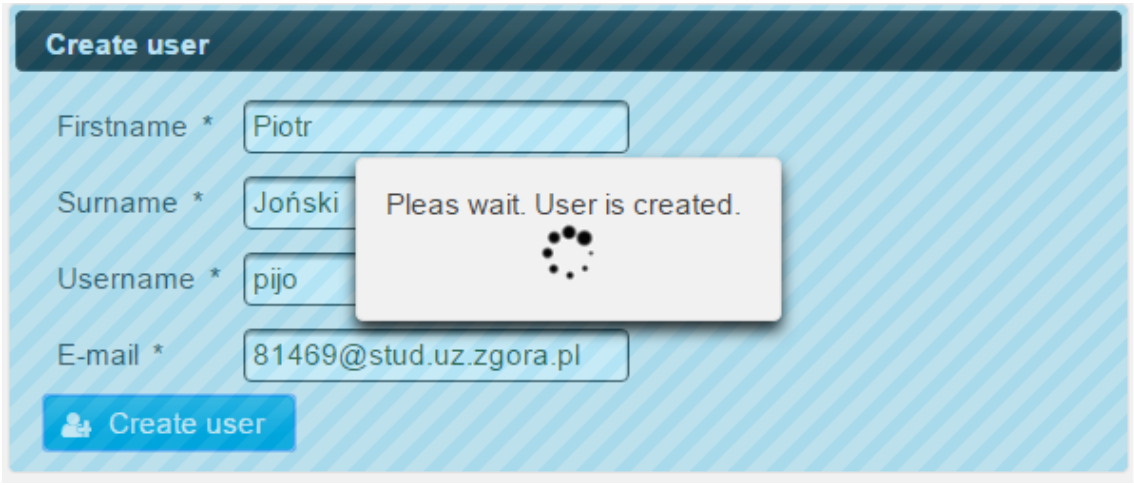
Widoki użytkownika opierają się na technologii JSF 2.2 [12] oraz darmowym frameworku Primefaces w wersji 5.3. Dzięki jego funkcjonalności zostały zaimplementowane tabele, menu, czy różnego rodzaju listy dostępne na stronach systemu. Całość operacji wykonywanych na stronie odbywa się za pomocą JavaScriptu, który został użyty przez framework oraz ziaren Backing Beans. Ziarna te mają dodatkową funkcjonalność - są odpowiedzialne za dostarczanie danych do widoku oraz obsługę żądań - to w nich znajduje się kod, który wykonuje się po wciśnięciu przycisku. Dla przykładu na listingu 3.4 został przedstawiony częściowy kod formularza, napisany w xhtml z użyciem JSF oraz Primefaces odpowiedzialny za wykonanie akcji tworzenia użytkownika. Na listingu został pominięty fragment odpowiedzialny za wyświetlanie pól, w których wprowadza się imię, nazwisko, nazwę użytkownika oraz adres e-mail.

```
<h:form id="createUserForm">
<p:panel id="createUserPanel" header="#{i18n['page.admin.user.create']}">
<p:focus context="createUserPanel" />
<p:messages id="messages" globalOnly="true closable="true" />
<h:panelGrid columns="3" cellpadding="5">
.................
</h:panelGrid>
<p:commandButton id="createButton" value="#{i18n['page.admin.user.create']}"
action="#{userService.createUser}" ajax="false" icon="fa fa-user-plus"
validateClient="true" onclick="PF('block').show()" />
</p:panel>
<p:blockUI block="createUserPanel" trigger="createButton" widgetVar="block">
<p:outputLabel value="#{i18n['page.admin.user.create.process']}" />
<br />
<p:graphicImage library="primefaces" name="/outputpanel/images/loading.gif" />
</p:blockUI>
</h:form>
```

Listing 3.4. Kod formularza tworzenia użytkownika

Najważniejszym elementem kodu jest przycisk utworzony za pomocą znacznika Primefaces <p:commandButton ... />. Jego wartość (value), czyli wyświetlany tekst jest

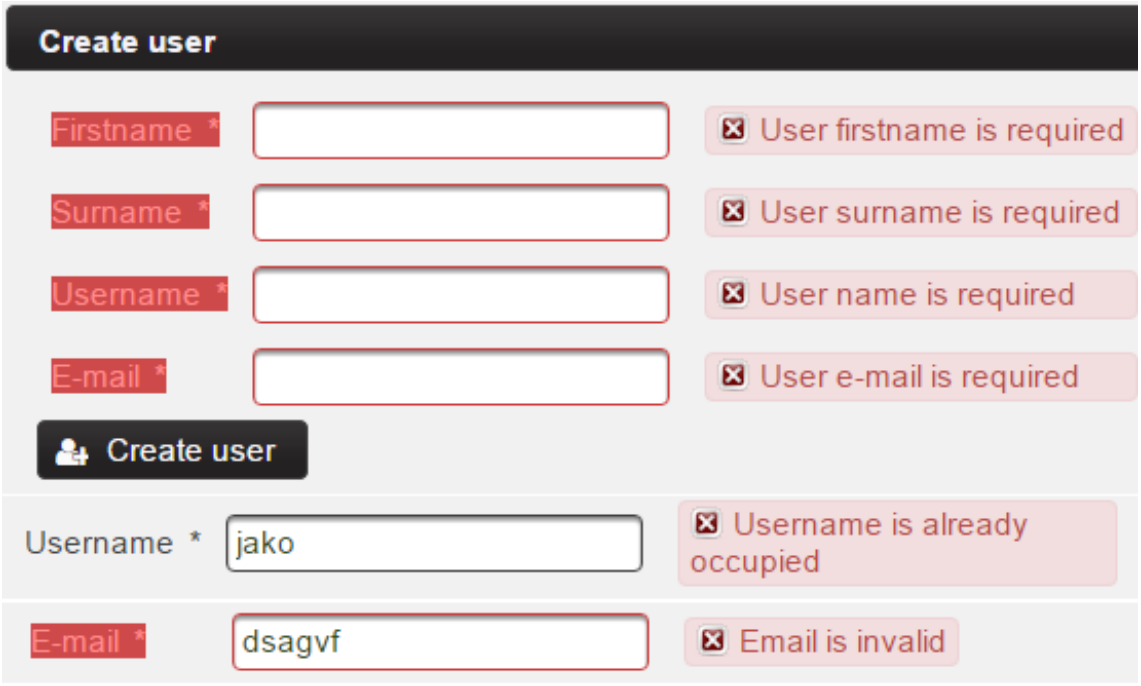
pobierany za pomocą zasobu o nazwie **i18n**, co sprawia, że możliwa jest internacjonalizacja systemu. Ta funkcjonalność również została wykonana w ramach pracy dyplomowej. Kolejnym elementem jest akcja (action) przycisku, która wskazuje na metodę *createUser* z klasy *UserService* zadeklarowanej jako ziarno o zasięgu widoku (view scoped). Po naciśnięciu przycisku zostaje wykonana JavaScriptowa akcja uruchomiona za pomocą obsługi onclick. Jej zadaniem jest wyświetlenie okna poczekania. Krok ten odpowiada wysłaniu żądania utworzenia użytkownika z widoku do ziarna UserService, a następnie wyświetleniu animacji przez ten serwis. Przebieg akcji został przedstawiony we wcześniejszej sekcji na rysunku 3.6. Samo okno poczekania zostało zaprezentowane na ilustracji 3.7.



Rys. 3.7. Ekran poczekania podczas tworzenia użytkownika

Po poprawnej walidacji nowy użytkownik zostaje utworzony, na jego skrzynkę trafia e-mail z wygenerowanym hasłem, a administrator zostaje przeniesiony na stronę z listą użytkowników. Jak wiadomo, wprowadzane dane nie zawsze muszą być poprawne. W takim wypadku użytkownikowi zostają wyświetlone komunikaty o błędach, jakie popełnił podczas wprowadzania danych do formularza, co przedstawia rysunek 3.8.

Kolejnym aspektem widoku jest to, że został on zaprojektowany w technologi responsywnej, co oznacza, że widok jest skalowany w zależności od rozmiaru okna przeglądarki. Dzięki temu użytkownik zawsze będzie miał wgląd w na całą zawartość strony. To sprawia, że przeglądanie treści systemu jest równie łatwe z poziomu przeglądarek desktopowych jak i urządzeń mobilnych. Technologia ta działa na zasadzie dynamicznego dostosowywania szerokości wyświetlanych elementów i jest w pełni oparta na systemie CSS (Cascading Style Sheets). Została ona zintegrowana z frameworkiem Primefaces, co sprawia, że jej użycie ogranicza się do zastosowania odpowiednich klas komponentów. Strony systemu zostały zaprojektowane z użyciem szablonów, które również są dostępne dzięki bibliotece



Rys. 3.8. Komunikaty o błędzie podczas tworzenia użytkownika

Primefaces. Cały szablon strony został opisany w jednym pliku, przez co w bardzo łatwy sposób możemy modyfikować układ poszczególnych grup elementów na stronie takich jak menu górne, treść właściwa czy stopka.

System posiada również zabezpieczenia, sprawiające, że część stron jest widoczna tylko dla użytkowników z rolą administratora, a pozostałe treści dla zalogowanych. Zabezpieczenia zostały oparte na specyfikacji JAAS (Java Authentication and Authorization Service), która jest implementowana przez używany kontener - Wildfly. Konfiguracja została przedstawiona na listingu 3.5 oraz 3.6.

```
<security-constraint>
<web-resource-collection>
<web-resource-name>scrumusSecurity</web-resource-name>
<description>Dostep tylko dla administratora</description>
<url-pattern>/admin/</url-pattern>
<http-method>GET</http-method>
</web-resource-collection>
<auth-constraint>
<role-name>ADMIN</role-name>
</auth-constraint>
</security-constraint>
```

Listing 3.5. Konfiguracja zabezpieczeń systemu scrumus - administrator

```
<security-constraint>
<web-resource-collection>
<web-resource-name>scrumusSecurity</web-resource-name>
<description>Dostep tylko dla zalogowanych uzytkownikow</description>
<url-pattern>/</url-pattern>
<http-method>GET</http-method>
<http-method>POST</http-method>
<http-method>HEAD</http-method>
<http-method>PUT</http-method>
</web-resource-collection>
<auth-constraint>
<role-name>DEVELOPER</role-name>
</auth-constraint>
</security-constraint>
```

Listing 3.6. Konfiguracja zabezpieczeń systemu scrumus - zalogowany użytkownik

Kolejnym możliwym sposobem komunikacji z aplikacją jest usługa REST, która jest oparta na protokole HTTP. Po wysłaniu zapytania na wybrany adres otrzymujemy odpowiedź w postaci obiektu JSON (*ang. JavaScript Object Notation*). Protokół HTTP oferuje zbiór metod, które w REST API (*ang. Application Programming Interface*) umownie używa się do wymienionych czynności: GET - pobranie zasobów, POST - wysłanie zapytania z ciałem, które jest używane przy zapisywaniu obiektów oraz DELETE - usuwanie obiektów. Listing 3.7 przedstawia konfigurację takiej usługi, która mogłaby zostać zaimplementowana na nowej lub istniejącej już warstwie JSF. Takie rozwiązanie umożliwia rozbudowę systemu o dodatkowe klienty.

```
@GET
@Path("/users/{userId}")
public Developer findUser(String userId) {
    try {
    int userInId = Integer.parseInt(userId);
      Optional<Developer> userOptional = userManager.findByUserId(userInId);
      return userOptional.orElse(null);
    } catch (NumberFormatException e) {
      return null;
    }
}
```

Listing 3.7. Przykładowa usługa REST

Rozdział 4

Dokumentacja wdrożeniowa i testowa

W tym rozdziale opiszę proces budowania projektu z zastosowaniem specjalnie przeznaczonego do tego celu systemu Gradle. Następnie prześledzimy przebieg wdrażania z wykorzystaniem kontenerów Dockerów, a na końcu przedstawię statystyki oraz wyniki testów projektu.

4.1. Gradle

We wcześniejszych rozdziałach został opisany system Gradle, który jest używany podczas budowy projektu wytwarzanego w ramach pracy dyplomowej. Skupię się tutaj na opisaniu komend użytecznych podczas pracy z kodem oraz przedstawię wyniki uruchomienia budowy na lokalnym serwerze TeamCity.

Na początek jednak pragnę przedstawić tabelę porównawczą systemu Gradle oraz Maven[1], ilustrującą kryteria na podstawie których został dokonany wybór narzędzia dla ciągłej integracji kodu. Wyniki zostały zawarte w tabeli 4.1.

System Gradle rozpoczyna budowę projektu po uruchomieniu komendy:

```
gradle build
```

Istnieje też możliwość definiowania własnych zadań oraz uruchamiania ich po kolei. Przykładowe wywołanie zadań: zbuduj, przetestuj jednostkowo, przetestuj integracyjnie, wgraj na serwer można uruchomić za pomocą komendy:

[1]Wyczerpujące porównanie wybranych systemów można znaleźć pod adresem `http://gradle.org/maven_vs_gradle/`

Tab. 4.1. Porównanie systemu Gradle vs Maven

	Gradle	Maven
Liczba linii w pliku konfiguracyjnym systemu scrumus	54	150
Wykonywanie zadań	Tak	Niemożliwe bez dodatków
Graficzny interfejs użytkownika	Tak (opcja -gui)	Nie
Automatycznie wykrywanie zmian i budowanie projektu	Tak	Nie
Dynamiczne zależności	Tak	Nie
Rozwiązywanie konfliktów zależności	Tak	Nie
Przyrostowe budowanie	Tak	Nie
Daemon	Tak	Nie
Równoległe wykonywanie zadań	Tak	Nie

```
gradle build test integrationTest deploy
```

Projekt wytworzony podczas pracy dyplomowej korzystał z lokalnego serwera do budowy projektów jakim jest TeamCity. W celu wykonania budowy została użyta komenda `gradle clean build`, która dawała następujące wyniki:

```
[20:20:00]Skip checking for changes - changes are already collected
[20:20:00]Clearing temporary directory: E:\study\TeamCity\buildAgent\buildTmp
[20:20:00]Publishing internal artifacts
[20:20:00]Checkout directory: E:\study\TeamCity\buildAgent\21cb64e16
[20:20:00][Updating sources] server side checkout
[20:20:00][Updating sources] Using vcs information from agent ...
[20:20:00][Updating sources] Building incremental patch for VCS root:...
[20:20:01][Updating sources] Repository sources transferred: 36.14 KB total
[20:20:01][Updating sources] Updating E:\study\TeamCity\buildAgent\21cb64e16
[20:20:01]Step 1/1: Gradle (7m:31s)
[20:20:01][Step 1/1] Starting: "C:\Program Files\gradle-2.6\bin\gradle.bat" ...
[20:20:01][Step 1/1] in directory: E:\study\TeamCity\buildAgent\21cb64e16
[20:20:03][Step 1/1] Starting Gradle in TeamCity build 174
[20:20:03][Step 1/1] :clean
[20:20:04][Step 1/1] :compileJava (2s)
```

```
[20:20:07][Step 1/1] :processResources
[20:20:07][Step 1/1] :classes
[20:20:07][Step 1/1] :war
[20:20:07][Step 1/1] :assemble
[20:20:08][Step 1/1] :compileTestJava (1s)
[20:20:10][Step 1/1] :processTestResources
[20:20:10][Step 1/1] :testClasses
[20:20:10][Step 1/1] :test (7m:22s)
[20:27:32][Step 1/1] :check
[20:27:32][Step 1/1] :build
[20:27:32][Step 1/1]
[20:27:32][Step 1/1] BUILD SUCCESSFUL
[20:27:32][Step 1/1]
[20:27:32][Step 1/1] Total time: 7 mins 30.796 secs
[20:27:32][Step 1/1] Process exited with code 0
[20:27:32]Publishing artifacts
[20:27:32]Publishing artifacts
[20:27:32]Publishing artifacts
[20:27:32]Generating coverage report...
[20:27:33]Calculating coverage statistics...
[20:27:33]Publishing internal artifacts
[20:27:33]Build finished
```

Jak widać, na załączonym powyżej logu z historii budowy, system TeamCity po dostaniu sygnału o zmianie stanu repozytorium zdalnego (zawartego na GitHubie) rozpoczął proces aktualizacji źródeł. Następnie po transferze wykonała się komenda Gradle uruchamiająca budowę projektu. W skład tej fazy wchodzą następująco:

- **clean** - wyczyszczenie folderu wynikowego,
- **compileJava** - kompilacja źródeł Java używając kompilatora javac,
- **processResources** - kopiowanie plików produkcyjnych zasobów do folderu docelowego,
- **classes** - gromadzenie klas produkcyjnych i folderów produkcyjnych zasobów,
- **war** - gromadzenie zasobów WAR,

- **assemble** - gromadzenie wszystkich archiwów w projekcie,
- **compileTestJava** - kompilacja źródeł testowych Java używając kompilatora javac,
- **processTestResources** - kopiowanie plików testowych zasobów do folderu docelowego,
- **testClasses** - gromadzenie klas testowych i folderów testowych zasobów,
- **test** - uruchomienie testów jednostkowych oraz integracyjnych,
- **check** - uruchomienie wszystkich zadań weryfikujących,
- **build** - wykonanie pełnej budowy projektu.

Po wykonaniu wszystkich wyżej wymienionych zadań dostajemy informację o powodzeniu zakończenia budowy projektu, co oznacza, że wszystkie testy przeszły oraz nie wystąpił błąd podczas żadnej fazy budowy. Na końcu system TeamCity publikuje wyniki testów – udostępnia je samemu sobie, tak aby były możliwe do wyświetlenia na stronie.

4.2. Docker

Docker jest kolejnym systemem użytym podczas wytwarzania pracy dyplomowej. Służy on do pakowania aplikacji w gotowe jednostki uruchomieniowe. Jego podstawowymi koncepcjami jest utrzymanie zawsze stabilnej, niezmiennej paczki aplikacji, którą da się bez problemu uruchomić na każdym systemie rodziny UNIX zawierającym oprogramowanie Docker.

Obrazy systemów są tworzone z plików *Dockerfile*, które opisują jakie oprogramowanie ma zostać wgrane w dany obraz. Następnie definiujemy zmienne środowiskowe oraz komendy, które mają zostać uruchomione podczas budowania obrazu. Po tych czynnościach następuje deklaracja co ma zostać wykonane po starcie kontenera.

Każdy obraz podczas swojego powstawania tworzy tzw. między obrazy. Są to obrazy, które powstają po wykonaniu komend modyfikujących aktualny obraz. Najlepiej wyjaśnić to na przykładzie:

1. W pliku Dockerfile definiujemy z jakiego obrazu ma powstać nasz, np. z obrazu systemu ubuntu. W tym momencie powstaje pierwszy między obraz.
2. Po wgraniu systemu ubuntu "mówimy" Dockerowi, aby zaaktualizował nasz system. Po tej aktualizacji system utworzy kolejny między obraz z wgranymi aktualizacjam.

3. Kolejnym krokiem będzie zainstalowanie np. bazy danych. Po jej instalacji zostanie wygenerowany kolejny między obraz itd.

Głównym powodem, który zadecydował o wybraniu tej technologii była jej łatwość użycia i możliwość uruchomienia systemu scrumus, mając jedynie kilkunastolinijkowy plik konfiguracyjny. Na listingu 4.1 został przedstawiony plik Dockerfile systemu scrumus.

```
FROM jboss/wildfly

USER root

ENV HOME /opt/jboss
ENV JBOSS_HOME /opt/jboss/wildfly

RUN yum install -y \
 postgresql \
 postgresql-contrib \
 pgadmin3 \
 telnet \
 curl

RUN /opt/jboss/wildfly/bin/add-user.sh admin admin --silent

COPY /postgresql /opt/jboss/wildfly/modules/system/layers/base/org/postgresql

EXPOSE 9990:9990 8080:8080

CMD ["/opt/jboss/wildfly/bin/standalone.sh",
       "-b", "0.0.0.0",
       "-bmanagement", "0.0.0.0"]
```

Listing 4.1. Plik Dockerfile systemu scrumus

W przedstawionym wyżej listingu tworzymy obraz Dockera oparty na obrazie *jboss/wildfly*, co mówi klauzula *FROM*. Następnie przełączamy się na użytkownika *root* i ustawiamy dwie zmienne środowiskowe za pomocą komendy *ENV*. Kolejnym krokiem jest aktualizacja systemu oraz zainstalowanie bazy danych Postgresql i skryptu telnet, który jest przydatny przy sprawdzaniu poprawności działania systemu. Następnie tworzymy użytkownika o nazwie *admin*. Po tych operacjach kopiujemy konfigurację bazy zawartą w folderze *postgres*, wystawiamy porty na zewnątrz kontenera. Komenda *CMD* mówi nam, co ma się wykonać po uruchomieniu kontenera. W naszym przypadku jest to uruchomienie

serwera wildfly w wersji standalone, na której uprzednio skonfigurowany został system scrumus.

4.3. Wyniki testów

Kolejnym, a zarazem ostatnim elementem jakie zostaną opisane w pracy dyplomowej będzie opis testów jednostkowych i integracyjnych. Podczas pisania testów korzystano z konwencji *given-when-then*, czyli mając (given) wartości, kiedy (when) wykonana zostanie dana operacje, wtedy (then) upewnij się, że coś się zgadza lub coś się wykonało. Zostaną również przedstawione wyniki pokrycia kodu oraz krótkie statystyki dot. projektu.

4.3.1. Testy jednostkowe

W pracy do jednostkowego przetestowania kodu użyto frameworka JUnit wspartego kolejną biblioteką do "podszywania się obiektów" (*ang. mocking objects*) PowerMockito. Testy jednostkowe stosuje się, jak sama nazwa mówi, do przetestowania jednostkowej funkcjonalności np. dodawania liczb, generowania ciągu znaków czy wywoływania metod. W celu poprawy czytelności kodu zastosowano funkcje projektu AssertJ ułatwiającego asercje z bogatym wsparciem dla list, obiektów *Optional* czy strumieni z Java 8. Kod przykładowego testu został przedstawiony na listingu 4.2.

```
@Test
public void shouldEncodeWithSHA256() throws Exception {
// given
String stringToHash = "123";
String expectedHash = "a665a45920422f9d417e4867efdc4fb8a04"
                    + "a1f3fff1fa07e998e86f7f7a27ae3";
HashGenerator hashGenerator = new HashGenerator();

// when
String result = hashGenerator.encodeWithSHA256(stringToHash);

// then
assertThat(result).isEqualTo(expectedHash);
}
```

Listing 4.2. Kod przykładowego testu jednostkowego

4.3.2. Testy integracyjne

Następnym rodzajem testów były testy integracyjne, które w odróżnieniu od jednostkowych służą do sprawdzenia integracji, czyli współpracy różnych komponentów systemu. Istnieją różne interpretacje testów integracyjnych - niektóre z nich mówią, że są to bardziej rozbudowane testy jednostkowe, które nadal nie mogą korzystać z zewnętrznych źródeł, inne zaś, iż komunikacja z np. bazą danych jest dozwolona.

W pracy zastosowano podejście, które umożliwia komunikację z bazą danych. Dzięki temu mogło zostać przetestowane również mapowanie się obiektów na struktury oraz zachowanie się JPA podczas próby wprowadzenia błędnych danych, które nie zawsze były oczywiste.

Testy integracyjne korzystają z frameworku Arquillian, który jak wcześniej opisano pozwala tworzyć pliki wdrożeniowe i uruchamiać je na serwerach wbudowanych, lokalnych oraz zdalnych. Przykład utworzenia testowego pliku wdrożenia został przedstawiony na listingu 4.3.

```
public static WebArchive createDeployment() {
    return ShrinkWrap.create(WebArchive.class, "test.war")
                     .addPackages(true, "edu.piotrjonski.scrumus")
                     .addPackages(true, "org.assertj.core")
                     .addPackages(true, "org.mockito")
                     .addPackages(true, "org.objenesis")
                     .addPackages(true, "org.apache.commons.collections4")
                     .addAsResource("META-INF/persistence.xml")
                     .addAsResource("configuration.properties")
                     .as(WebArchive.class);
}
```

Listing 4.3. Przykład przygotowania pliku wdrożenia Arquillian

Przedstawiony wyżej listnig tworzy plik WAR (*ang. Web Archive Resource*) oraz dodaje rekurencyjnie do niego pakiety znajdujące się w pakietach: *edu.piotrjonski.scrumus, org.assertj.core, org.mockito, org.obejnesis, org.apache.commons.collections4*. Dodatkowo kopiowane są pliki ustawień bazy danych - *persistence.xml* oraz plik konfiguracji systemu. Wszystko jest wdrażane pod nazwą *test.war*.

Na listingu 4.4 przedstawiono przykładowy test integracyjny. W pierwszej kolejności tworzone są dwa nowe zadania, które następnie zostają zapisane w bazie danych za pomocą obiektu DAO. Obiekt DAO został wstrzyknięty (*ang. injected*) do klasy testu, co było możliwe dzięki uruchomieniu testu w kontenerze. Kolejnym krokiem jest wyszukanie za-

dań z odpowiednim statusem - to właśnie metoda o nazwie *findAllIssuesWithIssueType* jest tutaj testowana. Na końcu następuje sprawdzenie, czy zwrócona lista zawiera dwa elementy.

```
@Test
public void shouldFindAllIssuesWithGivenIssueType() {
// given
Issue issue1 = createIssue();
Issue issue2 = createIssue();
issueDAO.saveOrUpdate(issue1);
issueDAO.saveOrUpdate(issue2);

// when
List<Issue> result = issueDAO.findAllIssuesWithIssueType(issue1.getIssueType()
.getName());

// then
assertThat(result).hasSize(2);
}
```

Listing 4.4. Przykład testu integracyjnego

4.3.3. Podsumowanie wyników testów oraz statystyki

Projekt został oparty na wielu różnych frameworkach dedykowanych językowi Java oraz z użyciem komponentów wspierających ciągłą integrację i wdrażanie systemu.

Podczas pracy napisano 103 (składających się z 2774 linii kodu) testy jednostkowe oraz 261 testów integracyjnych, które zawierają 7285 linii kodu. Kod produkcyjny jest złożony z 6349 linii, które zostały zoptymalizowane dzięki wykorzystaniu wzorców.

Łącznie napisano 16428 linii kodu Javy, 2137 xhtml, 1086 LaTeX, 292 xml oraz 83 css, co daje 20026 powodów do radości. Sumaryczny czas wytworzenia projektu, wraz z częścią teoretyczną, wyniósł 98 dni. W tym czasie rozwiązano 43 zadania, wykonano 196 zdalnych budowań projektów oraz 262 commity, co daje około 204 linie kodu, 0.4 zadania, 2 budowy projektu i 2.7 commita dziennie.

Kolejnym faktem jest, iż aplikację projektowano opierając się na dobrych wzorcach programistycznych [13][14][15], a podczas implementacji korzystano z języka Java w wersji 8 oraz wykorzystywano wszystkie jego mechanizmy takie jak strumienie czy kontenery wartości Optional. Dzięki tym kontenerom w kodzie produkcyjnym nie ma ani jednego słówka kluczowego **null**, a przynajmniej w całej logice biznesowej.

Łączne pokrycie kodu nie przekracza poziomu 72% klas, 43% metod oraz 32% linii kodu. Taki wynik jest dalece rozbieżny ze stanem faktycznym przez problemy z kompatybilnością frameworku Arquillian, PowerMockito oraz narzędzia mierzącego pokrycie kodu jakim jest IntelliJ Code Coverage Runner. W rzeczywistości testy integracyjne pokrywają więcej metod, niż jest to wskazane przez używane narzędzia. Spowodowane jest to tworzeniem obiektów proxy przez Arquilliana i dlatego pokrycie wywoływanych metod nie jest w pełni mierzone. Inne narzędzia służące do pomiaru takie jak JaCoCo czy Cobertura również miały z tym faktem problemy.

Rozdział 5

Podsumowanie

W trakcie pracy nad systemem wykorzystano wiele nowych technologii, które przyczyniły się do jego jakości oraz stabilności. Pierwszym celem jaki został postawiony systemowi była jego użyteczność – zostały zaimplementowane tylko obligatoryjne funkcje systemu, jaki powinien posiadać projekt tej kategorii. W łatwy i przejrzysty sposób pozwala on zarządzać projektem, który jest prowadzony za pomocą metodyki Scrum.

W celu zapewnienia trwałości wszystkie dane systemu zapisywane są w darmowej i popularnej bazie Postgres. Do zapewnienia interakcji aplikacji z bazą danych wykorzystana została specyfikacja JPA oraz jej implementacja oparta na Hibernate. Takie rozwiązanie sprawia, że modyfikacja i pielęgnacja już istniejącego kodu nie powinna sprawiać żadnych problemów.

Do osiągnięcia kolejnego celu pracy - przejrzystości interfejsu użytkownika - zastosowano popularny szkielet aplikacji JSF wzbogacony o darmowe komponenty z biblioteki Primefaces. Umożliwiło to skupienie się na wytwarzaniu funkcjonalności systemu zapewniając jednocześnie jego prostotę i elegancję.

Ostatnim celem była prosta konfiguracja i zarządzanie systemem oraz szybka reakcja na niedostępność systemu. Został on osiągnięty dzięki wykorzystaniu systemu Docker, który jest pionierem jeżeli chodzi o szybie wytwarzanie i uruchamianie środowisk zarówno deweloperskich jak i testowych.

Całość została uzupełniona o testy jednostkowe oraz integracyjne. W kodzie wykorzystano szereg udogodnień, jakie wprowadza wersja ósma Javy. Dodatkowo praktyczna znajomość wzorców projektowych pozwoliła na efektywniejszą pracę i szybszą implementację funkcjonalności przy jak najmniejszym wysiłku.

Oczywiście, jak każdy projekt informatyczny, tak i ten posiada pewne wady. Został

on napisany jako monolityczna aplikacja webowa, co sprawia, że dołączenie nowej funkcjonalności powoduje konieczność ponownego wdrożenia całego systemu.

System może zostać zrefaktoryzowany do postaci szeregu mikroserwisów, z których każdy posiada swoją odpowiedzialność. Na przykład serwis użytkowników odpowiadający za autentykację i autoryzację, serwis warstwy prezentacji korzystający z serwisu danych np. zadań lub projektów. Całość mogłaby również zostać oparta na Dockerach oraz na technologi mikroserwisów, którą wspaniale wspierają takie projekty jak SpringBoot.

Aktualna implementacja systemu również sprawia, że jest on przejrzysty, prosty w konfiguracji oraz estetycznie wyglądający. Są to najważniejsze cechy takich projektów, które mogą przyciągnąć uwagę wielu potencjalnych użytkowników.

Literatura

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

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

[3] Oficjalna witryna internetowa frameworka Arquillian. `http://arquillian.org`

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

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

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

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

[8] Bill Burke, Richard Monson-Haefel. *Enterprise JavaBeans 3.0.* Helion 2007

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

[10] Oficjalna witryna internetowa systemu Docker. `https://www.docker.com`

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

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

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

[14] Joshua Kerievsky. *Refaktoryzacja do wzorców projektowych.* Helion 2005

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