POLSKO-JAPOŃSKA WYŻSZA SZKOŁA
TECHNIK KOMPUTEROWYCH

Wydział informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Bartłomiej Wójtowicz
Nr albumu s9831

Elementy grywalizacji w zarządzaniu projektem informatycznym

Praca magisterska napisana
pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, czerwiec 2013

Streszczenie

Niniejsza praca omawia zagadnienia związane z zarządzaniem projektami informatycznymi oraz problemami jakie temu towarzyszą. Autor poddał analizie obecne rozwiązania i próbował zaobserwować jak sprawują się one na tle motywacji pracowników w zespołach programistycznych.

Przedstawiono zestaw standardowych środków potrzebnych do kontrolowania projektu oraz koncepcję rozwiązania postawionego problemu za pomocą grywalizacji.

Finalną fazą pracy jest działający prototyp, który umożliwia posługiwanie się elementami grywalizacji w kontekście zarządzania projektami. Został on wykonany przy użyciu technologii Ruby on Rails.

Spis treści

1 Wstęp

Obecnie na rynku istnieje duża ilość narzędzi, zarówno darmowych jak i płatnych, które wspomagają proces zarządzania projektami. Oferują one zazwyczaj następujące możliwości:

- zarządzanie zadaniami
- kontrola i zarządzanie budżetem
- planowanie
- alokacja zasobów
- zarządzanie dokumentacją i specyfikacją
- wymiana informacji
- generowanie raportów

Powiększa się także grono użytkowników z dostępem do sieci za pomocą urządzeń mobilnych. Taki przyrost sprawia, że ludzie coraz częściej zastępują swoje komputery małymi, przenośnymi smartphone'ami, czego następstwem jest zwiększony popyt na oprogramowanie mobilne.

1.1 Cel pracy

Celem pracy jest opracowanie koncepcji narzędzia, które wspomoże proces zarządzania projektem informatycznym oraz zwiększy efektywność pracy ludzi pracujących w zespołach programistycznych.

Istniejące rozwiązania bardzo dobrze sprawują się w kwestii zarządzania projektem jednak brakuje im czynnika motywującego pracowników, który zwiększy produktywność i zminimalizuje niepowodzenia, które często występują podczas tworzenia aplikacji m.in.:

- niedotrzymywanie terminów oddawania poszczególnych części implementacji (co przekłada się na niezadowolenie klienta)
- niedokładność wykonywanych zadań (obniżenie jakości)

1.2 Zarys koncepcji

Kreowany system będzie zawierał zestaw standardowych modułów, niezbędnych do sprawnego zarządzania projektem. Dodatkowo będzie on dostępny w wersji mobilnej, tak by zaspokoić rzeszę użytkowników mobilnych i podążać najnowszymi trendami.

Jednym z najważniejszych aspektów koncepcji jest wprowadzenie elementów grywalizacji (z ang. *gamification*) jako czynnika motywującego pracowników.

Grywalizacja wprowadzi techniki z gier komputerowych jako skuteczne środki do manipulacji ludzkim zachowaniem. Element ten może znacząco podnieść efektywność pracowników oraz polepszyć ich samopoczucie. Dodatkowo sprawi też, że będą wykonywać zadania z większym zaangażowaniem i przyjemnością, nawet jeśli wcześniej te zadania były dla nich nudne.

1.3 Rezultaty pracy

Rezultatem pracy jest stworzona koncepcja oprogramowania, które wspomoże proces zarządzania projektem wraz z elementami grywalizacji. Sporządzona koncepcja zostanie ukazana w aplikacji demonstracyjnej. W skład dema będą wchodzić: główny system webowy, dostępny za pomocą przeglądarki internetowej oraz aplikacja mobilna na platformę Android.

System ten będzie pozwalał na zarządzanie zadaniami, wymianę informacji i plików, monitorowanie terminów ukończenia projektów, generowanie raportów oraz definiowanie dynamicznych elementów grywalizacji.

1.4 Organizacja pracy

Na początku przedstawiono ogólnie problem dotyczący efektywności pracy zespołów programistycznych oraz możliwe jego rozwiązanie.

W dalszej części pracy opisano czym jest grywalizacja i jakimi prawami się rządzi. Kolejno poddano analizie obecne rozwiązania wspomagające proces zarządzania projektem – wyszczególniono ich podobieństwa oraz różnice. Na samym końcu nakreślono ich wady.

Po wnikliwej analizie opisano krótko zestaw technologii użytych do budowy koncepcji, która pomoże rozwiązać wcześniej przedstawione niedoskonałości.

W rozdziale 4 w pełnych detalach zaprezentowano koncepcję rozwiązania postawionego problemu, natomiast rozdział następny przedstawia już prototyp owej koncepcji. Następnie zademonstrowano przykład użycia powstałego prototypu.

W zakończeniu ujęto podsumowanie całej pracy, wyliczono wady i zalety przyjętego rozwiązania oraz wyznaczono plany rozwojowe na przyszłość.

2 Analiza istniejących rozwiązań

Na rynku istnieje szeroka gama oprogramowania do zarządzania projektami – systemy te są zazwyczaj bardzo do siebie zbliżone pod kątem funkcjonalności – różnią je natomiast podejścia, w jaki sposób prezentowane są dane oraz interfejs użytkownika, które wpływają ich na użyteczność (z ang. usability). Funkcjonują także systemy oparte o mechanizm grywalizacji, które próbują zwiększać efektywność pracowników.

Zostaną omówione następujące aplikacje:

- Redmine
- YouTrack
- Basecamp
- JIRA
- Visual Studio Achievements Extension
- PROPS to you

2.1 Wstęp

Niniejszy rozdział przedstawia mechanizm grywalizacji oraz zasdy działania jego składowych.

2.1.1 Definicja grywalizacji

Grywalizacja zbiera w sobie wszystkie wątki związane z tworzeniem gier i przenosi je na grunt niezwiązany bezpośrednio z grami. W ten sposób łączy koncepcje rzeczywistych gier, gier reklamowych i gier dotyczących zmian w spójny system, który korzysta z najnowszych odkryć psychologii behawioralnej i analizy sukcesów gier społecznościowych [1].

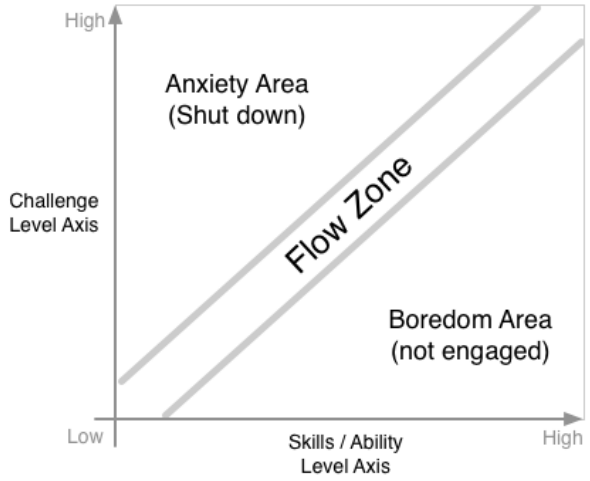
Inaczej zwana też gamifikacją lub gryfikacją. Wprowadzenie mechanizmów grywalizacji do danej sytuacji pozwala zachęcić ludzi do realizacji czegoś co w rzeczywistości sprawiałoby im mniej przyjemności. Odpowiedni zestaw tych czynników prowadzi do zwiększenia zaangażowania i podniesienia efektywności pracy [1, 2].

2.1.2 Ludzki motywator

Podstawowym elementem grywalizacji jest gracz. Analiza gracza i jego zrozumienie jest środkiem do stworzenia systemu, który odniesie sukces.

Ważnym elementem podczas projektowania takiego systemu jest zapewnienie odpowiedniego balansu gry. Innymi słowy jak dobrać czynniki gry, aby gracz chciał pozostać w grze dłużej.

Autorzy książki [1] zobrazowali ten bilans w postaci wykresu widocznego na rysunku 1.



Rysunek 1: Balans gry w postaci diagramu
Źródło: `http://twobenches.wordpress.com/2012/09/23/flow/`

Oś Y reprezentuje poziom wyzwania gry, natomiast oś X poziom umiejętności gracza. Po środku jest strefa przepływu pomiędzy obszarem znudzenia (*Boredom Area*), a obszarem frustracji (*Anxiety Area*). Jakiekolwiek odchylenie od strefy przepływu powoduje, że gracz nie kontynuuje gry. W przypadku zejścia do obszaru znudzenia gracz traci zaangażowanie – umiejętności gracza są wysokie, ale gra za jest za łatwa, przestaje być wyzwaniem. W przypadku obszaru frustracji gracz odchodzi – nie zdobył jeszcze dużych umiejętności, a gra staje się trudna [1].

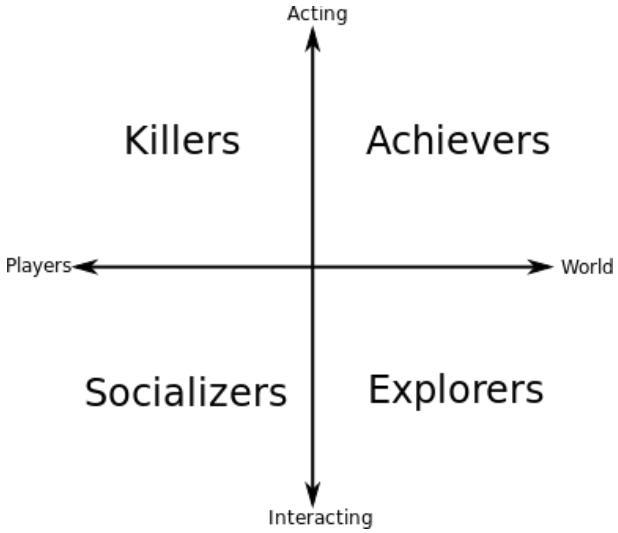
2.1.3 Rodzaje graczy

Każdy człowiek jest inny dlatego analiza graczy nie jest łatwa, a od niej zależy dostrojenie rozgrywki. Jedną z metod, która pomoże zidentyfikować graczy jest „Test Bartle'a" [1, 3, 4].

Według tego testu możemy wyróżnić 4 główne rodzaje graczy:

- **Odkrywca** (*Explorer*) – dla gracza ważny jest świat i wszystko co w nim ukryte, musi być pierwszy w odkrywaniu, aby móc później pochwalić się tym innym
- **Zdobywca** (*Achiever*) – typ gracza, który lubi zawsze coś wygrywać, interesują go tablice wyników – ze względu nastawienia na ciągłe zdobywanie czegoś, w wyniku porażki może stracić zainteresowanie grą
- **Społecznik** (*Socializer*) – społecznik gra by zdobywać punkty w kontaktach towarzyskich, jego głównym celem jest interakcja z ludźmi – co nie znaczy, że nie interesuje go wygrana bo też ma to dla niego znaczenie, jednak nie tak wielkie jak długotrwałe relacje społeczne
- **Zabójca/Niszczyciel** (*Killer*) – statystycznie najmniej liczny typ gracza z czterech wymienionych, ale także bardzo ważny – chęć wygrania u zabójców jest zbliżona do zdobywców jednakże sama wygrana nie jest wystarczająca – zabójca musi widzieć jak ktoś przegrywa, a najlepiej jak wszystko dzieje się na oczach innych graczy

Obecnie jest więcej typów graczy, wyżej wymieniono tylko główne rodzaje. Zależności między nimi można zobaczyć na rysunku 2.



Rysunek 2: Rodzaje graczy
Źródło: http://freemiumdesign.blogspot.com/2013/04/my-transition-from-world-of-warcraft-to.html

Warto też zaznaczyć, że typy te nie wykluczają się wzajemnie. Każdy gracz może składać się ze wszystkich typów lub kliku w określonej proporcji – wtedy cecha dominująca determinuje gracza.

2.1.4 Mechanika

Podczas tworzenia mechanizmów grywalizacji stosuje się wiele technik z projektowania gier. Te najczęściej zapożyczane zostały zaprezentowane poniżej.

- **Wyzwania i zadania** – zadania te mogą być przedstawione jako misje, jakie gracz musi wykonać – najczęściej są to zadania, które są wymagane do wykonania przez użytkowników, ale zaprezentowane w innej formie. Zróżnicowanie polega na rozszerzeniu misji, którą np. należy wykonać w grupie (angażuje to więcej osób razem)
- **Statystyki i paski postępu** – takie paski pozwalają śledzić użytkownikowi swój postęp i informować go jak blisko jest awansu na następny poziom lub czy jest bliski ukończenia zadania
- **Odznaczenia** – odznaczenia są bardzo ważnym elementem, który sygnalizują status społeczny (wyróżnienie) – najczęściej użytkownik zdobywa odznaki za ukończone zadanie lub dodatkowe osiągnięcia
- **Poziomy trudności** – aby gracz ciągle pozostawał w grze należy odpowiednio stopniować poziom gry i trudność zdobywania nowych odznak lub np. pasków doświadczenia – im dłużej gra tym powinno być trudniej, ale niezbyt przesadnie – należy znaleźć balans przejścia od nowicjusza do mistrza

- **Rywalizacja** – rywalizacja prowadzi do tzw. „wyścigu szczurów" – ludzie są skłonni zrobić więcej by być tym pierwszym lub uzyskać specjalne wyróżnienie
- **Współpraca** – przeciwieństwem rywalizacji z kolei jest współpraca – zmuszenie użytkowników do osiągnięcia nowego celu wspólnymi siłami – taki zabieg polepsza relacje społeczne
- **Rankingi** – rankingi są bardzo ważnym elementem – użytkownik musi widzieć na jakiej jest pozycji – dzięki temu ciągle dąży, aby być pierwszym
- **Punkty** – punkty lub wirtualne pieniądze, którymi nagradzany jest gracz – walutę często może wymienić na coś innego. Przyznawane punkty powinny być adekwatnie wysokie do trudności zadań, aby gracz nie stracił zaangażowania.
- **Wirtualne przedmioty** – wirtualne przedmioty, skarby i wszelakie różności, którymi gracz może różnić się od pozostałych graczy są równie prestiżowe co odznaczenia.
- **System nagród/komunikacji** – tego typu systemy służą do poszerzania relacji społecznych między ludźmi

Taki zestaw elementów zaczerpniętych z gier należy także wzbogacić o odpowiednią atmosferę. Dzięki temu osoba poddana technice grywalizacji poczuje się jak gracz w grze, którego celem są różne zadania. Efektem będzie większe zaangażowanie.

2.1.5 Przykłady aplikacji wykorzystujące elementy grywalizacji

Naświetlone w poprzednim rozdziale elementy mechaniki zostały już użyte w wielu aplikacjach, które odniosły sukces.

- **Foursqure** – jest to społecznościowa aplikacja na smartphone'y, której zadaniem jest informowanie przyjaciół o lokalizacji naszego pobytu (funkcja „*check-in*"). Użytkownik za rejestrację w danym położeniu otrzymuje punkty i czasami odznaki. Foursquare posiada poziomy statusu użytkownika – w zależności od zaawansowania zyskuje on dodatkowe przywileje. Istotnym faktem jest, że przed sukcesem Foursquare istniała aplikacja Dodgeball, która miała podobny cel, ale poniosła porażkę. Dopiero wprowadzenie grywalizacji przyniosło żądany efekt [1].
- **Nike Plus** – aplikacja mobilna, której celem jest motywowanie biegaczy. Tablice wyników, gdzie gracze mogą porównywać swoje wyniki oraz osiągnięcia (nawet ze sławnymi osobami) to pole do rywalizacji. Nie znaczy to, że Nike Plus jest nastawione tylko na konkurencję. Podczas biegu system wysyła powiadomienie na platformę Facebook. Gdy ktoś ze znajomych zobaczy powiadomienie i kliknie „Lubię to" to na telefonie biegacza odtworzy się krótka ścieżka dźwiękowa z oklaskami, co podbudowywuje i dodaje sił podczas wysiłku – pozwala to na pogłębianie więzi międzyludzkich [1].
- **Yahoo! Answers** – platforma służąca do zadawania pytań i szukania odpowiedzi. System promował zadawanie pytań, jednak po czasie okazało, że pojawiało się dużo

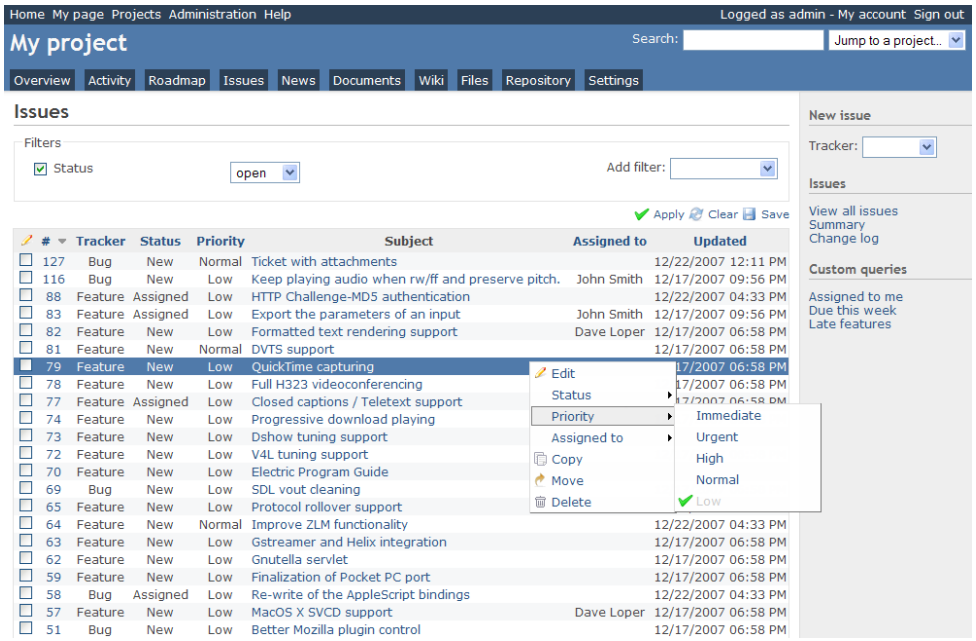
bezsensownych pytań oraz duplikatów dlatego został poddany modernizacji. Po 2005r wprowadzano zmiany mechaniki punktów. Za zadanie pytania użytkownik płaci 5 punktów, a za zamknięcie otrzymuje 3 punkty[1]. Obecnie Yahoo Answers wciąż jest bardzo popularną witryną, która angażuje masę ludzi [1]. Podobną mechanikę można zauważyć w systemie StackOverflow przeznaczonym dla branży IT.

2.2 Redmine

System Redmine jest elastycznym narzędziem służącym do zarządzania projektami dostępnym za pomocą przeglądarki internetowej. Został stworzony w technologii Ruby On Rails. Jest darmowy, oparty na licencji GPL w wersji 2.

Główne funkcjonalności jakie zawiera Redmine to:

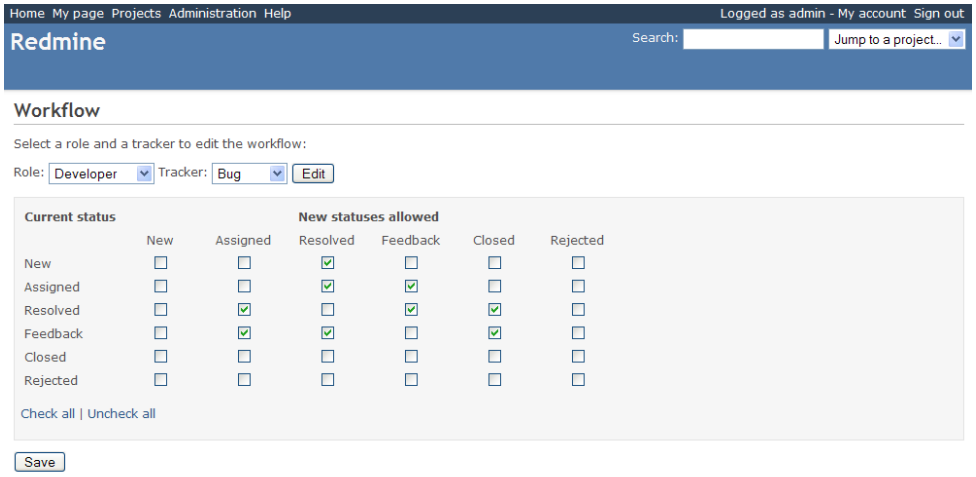
- zarządzanie zadaniami (rysunek 3) – system zezwala na tworzenie zadań i podzadań dla projektów, dla danego projektu można zdefiniować własne statusy i typy zadań, ponadto można sterować przepływem statusów (z ang. *workflow*) (rysunek 4)



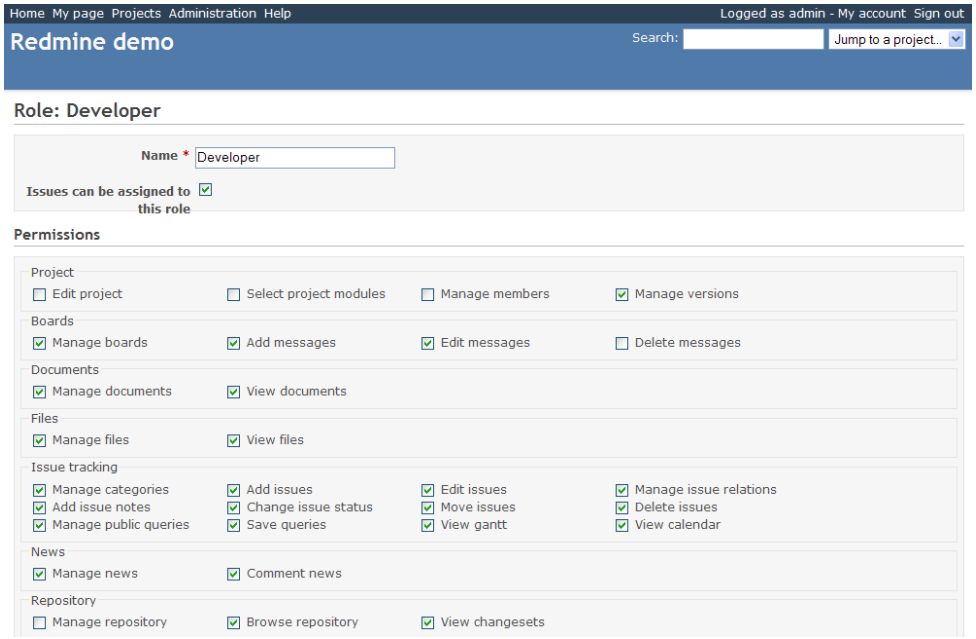
Rysunek 3: Widok listy zadań dla systemu Redmine
Źródło: Opracowanie własne

- kalendarz – widok kalendarza pozwala na łatwe monitorowanie ważnych dat wliczając w to ukończenie projektu czy poszczególnych jego zadań
- wsparcie dla wielu projektów – w Redmine można zarządzać wieloma projektami naraz, dodatkowo każdy projekt może mieć podprojekty
- elastyczny podział użytkowników na role i prawa dostępu (rysunek 5) – użytkownik może mieć różne role dla różnych projektów oraz zawężone lub poszerzone prawa dostępu

[1]Aktualny system punktów dla Yahoo! Answers znajduje się tutaj: `http://answers.yahoo.com/info/scoring_system`

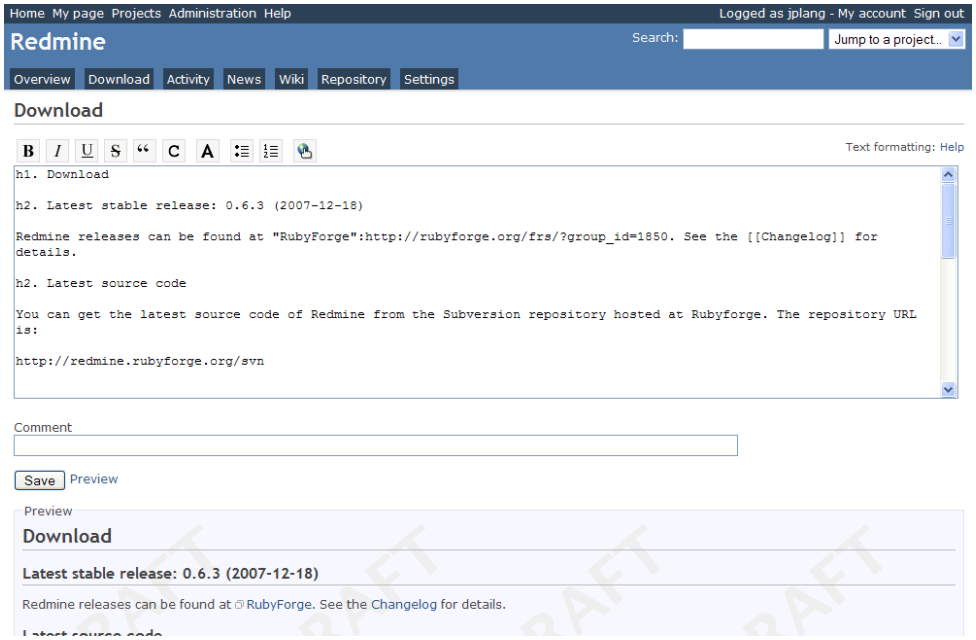


Rysunek 4: Sterowanie przepływem zadań w systemie Redmine
Źródło: Opracowanie własne

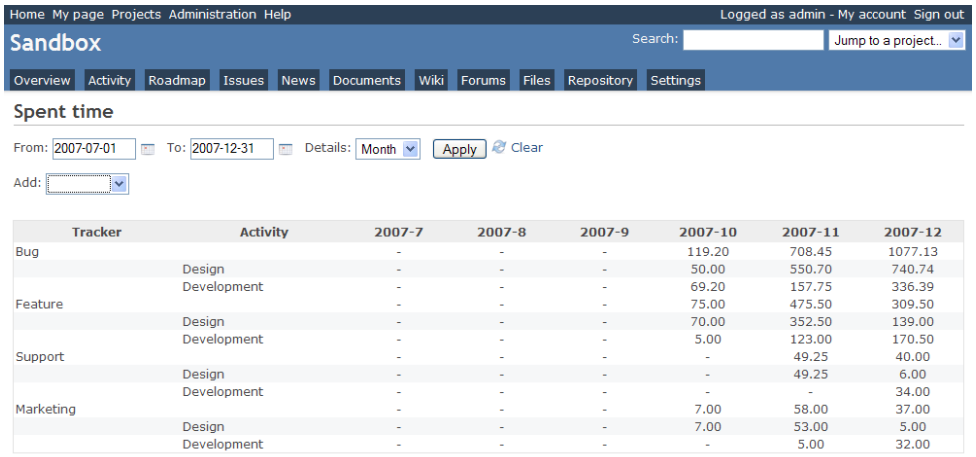


Rysunek 5: Ustalanie praw dla ról w projekcie w systemie Redmine
Źródło: Opracowanie własne

- zarządzanie dokumentami i plikami (rysunek 6) – można wgrać pliki na platformę lub do projektu, system zawiera także zarządzanie dokumentacją projektu, gdzie można przechowywać np. ważne informacje lub konfiguracje
- forum – w projekcie można odblokować forum, aby móc usprawnić wymianę komunikacji lub poruszać ważne sprawy czy problemy
- monitorowanie czasu (rysunek 7) - użytkownicy systemu mogą logować czas spędzony nad zadaniami do systemu, dzięki czemu osoby odpowiedzialne za projekt (użytkownicy wyżsi prawami) mogą monitorować przebieg czasu



Rysunek 6: Edycja dokumentu w systemie Redmine
Źródło: Opracowanie własne

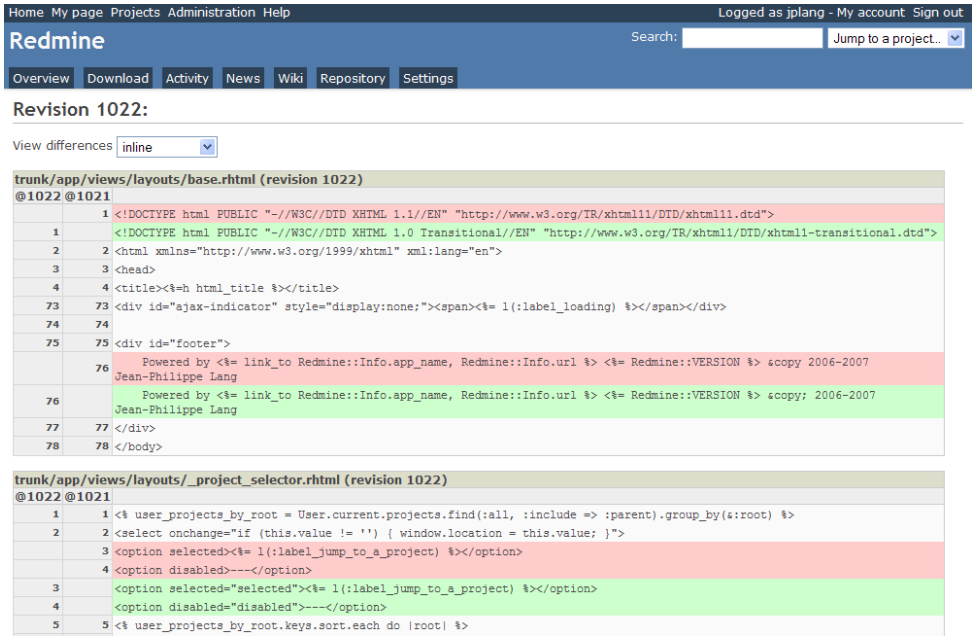


Tracker	Activity	2007-7	2007-8	2007-9	2007-10	2007-11	2007-12
Bug		-	-	-	119.20	708.45	1077.13
	Design	-	-	-	50.00	550.70	740.74
	Development	-	-	-	69.20	157.75	336.39
Feature		-	-	-	75.00	475.50	309.50
	Design	-	-	-	70.00	352.50	139.00
	Development	-	-	-	5.00	123.00	170.50
Support		-	-	-	-	49.25	40.00
	Design	-	-	-	-	49.25	6.00
	Development	-	-	-	-	-	34.00
Marketing		-	-	-	7.00	58.00	37.00
	Design	-	-	-	7.00	53.00	5.00
	Development	-	-	-	-	5.00	32.00

Rysunek 7: Monitorowanie czasu spędzonego nad zadaniami w systemie Redmine
Źródło: Opracowanie własne

- Integracja z systemami kontroli wersji (rysunek 8) – system Redmine można zintegrować z kilkoma systemami kontroli wersji dzięki czemu można przeglądać kod bezpośrednio w aplikacji

Redmine jest dobrym narzędziem do zarządzania projektem w firmie, zawiera dużo funkcjonalności jednak jest już nieco przestarzały - sposób prezentacji danych i dostępu do nich jest bardzo archaiczny co sprawia, że korzysta się z niego trudniej i ciężej.



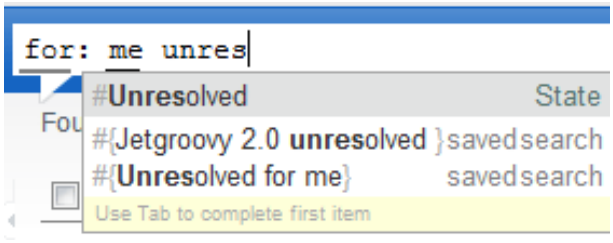
Rysunek 8: Podgląd kodu w systemie Redmine
Źródło: Opracowanie własne

2.3 YouTrack

YouTrack jest komercyjnym narzędziem do zarządzania projektami. Dostęp do systemu odbywa się poprzez przeglądarkę internetową. Oprogramowanie to zostało stworzone w technologii Java.

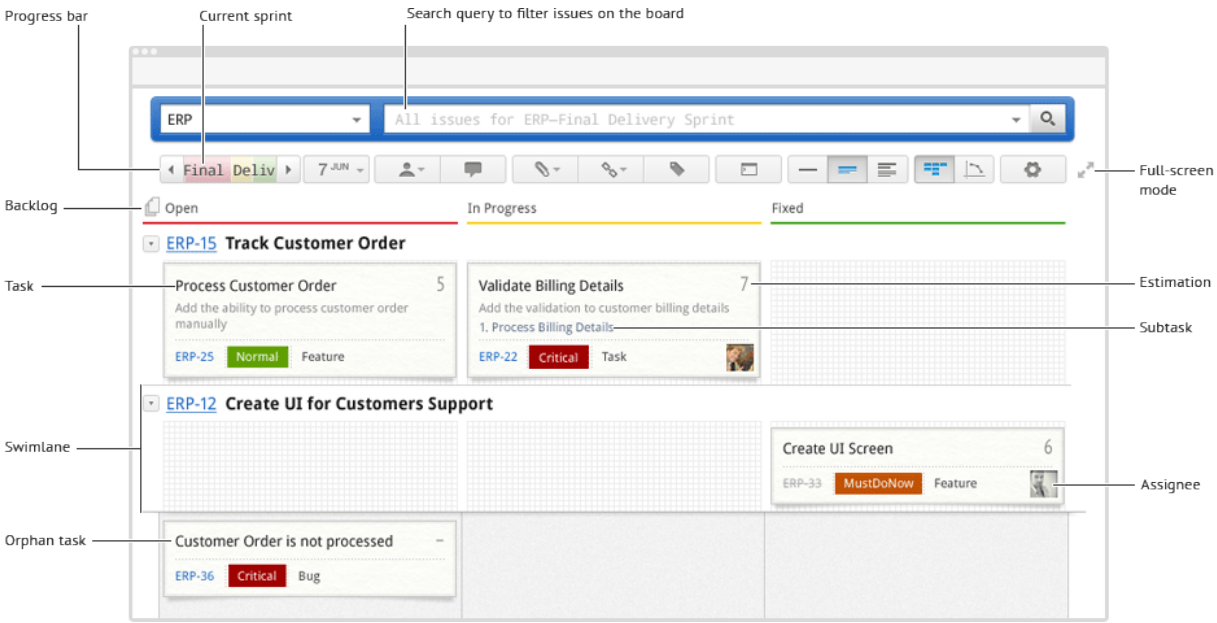
Główne funkcjonalności jakie zawiera YouTrack to:

- sterowanie klawiaturą – system jest bardziej nastawiony na posługiwanie się nim za pomocą klawiatury co w praktyce pozwala zaoszczędzić trochę czasu i zwiększa jego użyteczność – w odróżnieniu od systemów gdzie głównie należy posługiwać się kursorem myszy
- zarządzanie zadaniami – jak prawie wszystkie systemy do zarządzania projektami, YouTrack pozwala na tworzenie zadań oraz ich szczegółowych opisów z możliwością komentowania przez użytkowników. Zadania te można tagować, co zwiększa ich czytelność na liście. Ponadto istnieje możliwość tworzenia połączeń między zadaniami (np. łączenie duplikatów pod jednym zadaniem)
- wieleprojektowość – obecnie to już standard raczej we wszystkich oprogramowaniach tego typu – YouTrack także służy do zarządzania wieloma projektami naraz
- zaawansowany język zapytań do wyszukiwania (rysunek 9) – jest to jedna z ważniejszych zalet tego systemu, elastyczny system wyszukiwania z autouzupełnieniem składni pozwala na bardzo szybkie i efektywne znajdowanie danych, których szukamy. Z racji, że YouTrack jest zorientowany także na korzystanie z klawiatury, używany w nim język zapytań sprawdza się znacznie lepiej niż każde inne zaawansowane wyszukiwanie w innych systemach, gdzie większość parametrów określa się za pomocą kliknięć myszy - tak powstałe reguły wyszukiwań można zapisać jako stałe zapytania i korzystać z nich później



Rysunek 9: Wyszukiwanie w systemie YouTrack
Źródło: http://www.jetbrains.com/youtrack/

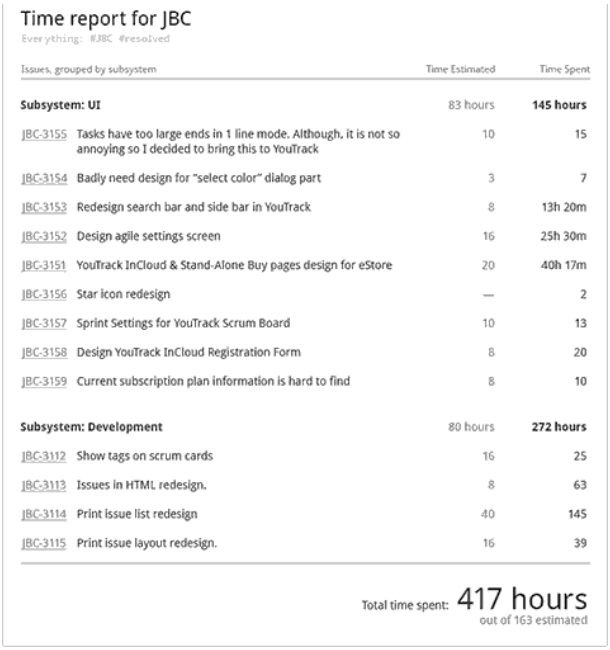
- zaawansowany język do edycji reguł przepływu – w systemie za pomocą specjalnego języka można sterować przepływem tworząc odpowiednie reguły, które aktywują się w odpowiedzi na określone zdarzenia. Przykładowo można stworzyć reguły, które będą aktywować się gdy zadanie przekroczy swój termin realizacji
- wizualizacja danych (rysunek 10) – dane zaprezentowane w systemie są bardziej przejrzyste i nowoczesne dla użytkownika co pozwala łatwiej śledzić postęp na projekcie. Dane są przedstawione jako kolumny (*swim lanes*), na których rozmieszcza się zadania – całość można dostosowywać do swoich potrzeb



Rysunek 10: Przykładowy widok kolumnowy w systemie YouTrack
Źródło: http://www.jetbrains.com/youtrack/

- eksport i import danych – w YouTrack można eksportować dane do zewnętrznych plików oraz importować je z różnych systemów wliczając w to Redmine i JIRA omówione w niniejszej pracy
- raportowanie (rysunek 11) – sam YouTrack w sposób jaki przedstawia niektóre dane jest już małym raportem ponieważ taka reprezentacja od razu pokazuje co jeszcze należy zrobić w projekcie lub gdzie leży problem. Oczywiście system pozawala także na generowanie

różnych raportów m.in. czas spędzony nad zadaniami czy różnorodne zestawienia w postaci macierzy



Time report for JBC

Everything: #JBC #resolved

Issues, grouped by subsystem		Time Estimated	Time Spent
Subsystem: UI		83 hours	**145 hours**
JBC-3155	Tasks have too large ends in 1 line mode. Although, it is not so annoying so I decided to bring this to YouTrack	10	15
JBC-3154	Badly need design for "select color" dialog part	3	7
JBC-3153	Redesign search bar and side bar in YouTrack	8	13h 20m
JBC-3152	Design agile settings screen	16	25h 30m
JBC-3151	YouTrack InCloud & Stand-Alone Buy pages design for eStore	20	40h 17m
JBC-3156	Star icon redesign	—	2
JBC-3157	Sprint Settings for YouTrack Scrum Board	10	13
JBC-3158	Design YouTrack InCloud Registration Form	8	20
JBC-3159	Current subscription plan information is hard to find	8	10
Subsystem: Development		80 hours	**272 hours**
JBC-3112	Show tags on scrum cards	16	25
JBC-3113	Issues in HTML redesign.	8	63
JBC-3114	Print issue list redesign	40	145
JBC-3115	Print issue layout redesign.	16	39

Total time spent: 417 hours
out of 163 estimated

Rysunek 11: Przykładowy raport czasu w YouTrack
Źródło: `http://www.jetbrains.com/youtrack/`

- automatyczne zarządzanie czasem – dzięki automatycznym procedurom zarządzania czasem użytkownik nie musi ręcznie wpisywać czasu jaki spędził nad danym zadaniem, tylko w odpowiednim momencie naciskać przyciski Śtart"i Śtop-- co pozwala bardzo dokładnie monitorować czas
- API oraz integracja z zewnętrznymi narzędziami – system posiada możliwość stworzenia dodatkowych narzędzi i automatyzacji pracy poprzez API, które udostępnia. Ponadto jest zintegrowany z kilkoma istniejącymi już narzędziami, także przydatnymi w procesie tworzenia oprogramowania np. serwer ciągłej integracji (z ang. *continuous integration*)

System YouTrack oferuje dużo funkcjonalności dla zarządzania projektem, które są precyzyjnie wykonane. Sposób prezetnacji danych i dostęp do nich jest na wysokim poziomie co ułatawia i uprzyjemnia z nim pracę.

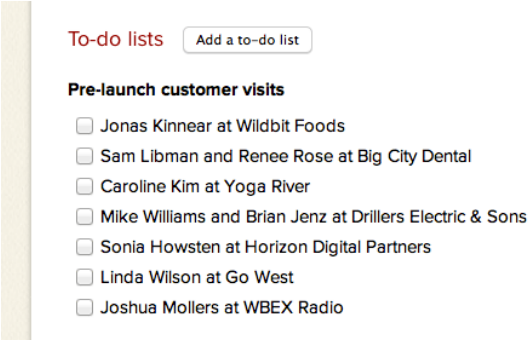
2.4 Basecamp

Basecamp to komercyjne oprogramowanie do zarządzania projektami, stworzone i aktywnie rozwijane w technologii Ruby On Rails przez twórców tego framework'a.

Główne funkcjonalności oferowane przez Basecamp:

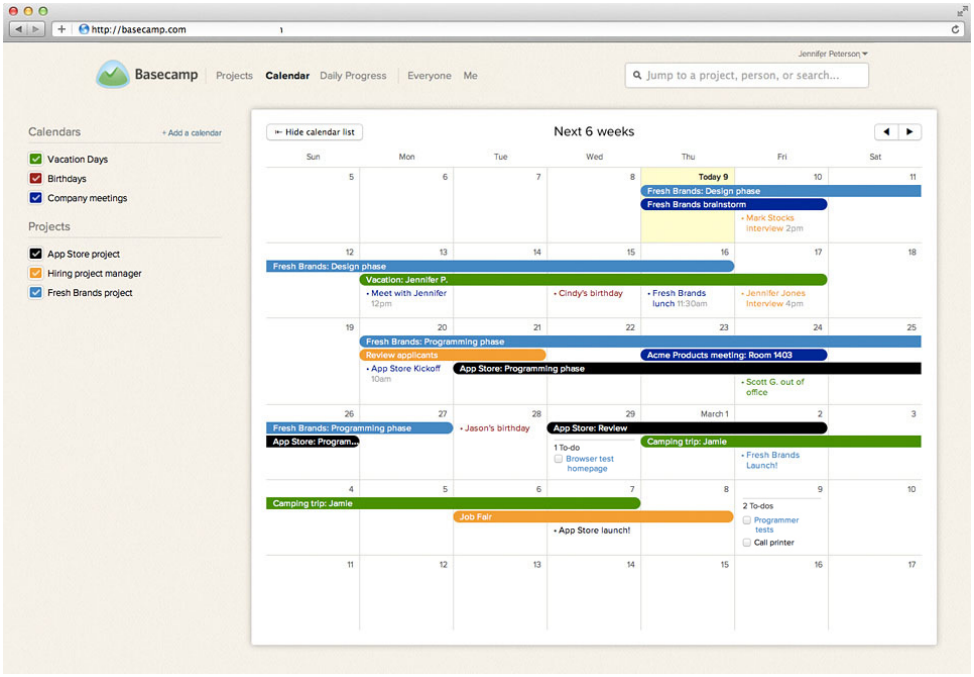
- wieloprojektowość – zarządzanie wieloma projektami
- zarządzanie zadaniami (rysunek 12) – platforma Basecamp bardzo mocno skupia się na zadaniach i ich prezentacji, zadania przedstawione są jako listy „TODO", które można

łatwo i szybko modyfikować poprzez dodawanie nowych zadań do list lub porządkowanie ich kolejności na liście



Rysunek 12: Lista zadań w Basecamp
Źródło: `http://basecamp.com/one-page-project`

- kalendarz (rysunek 13) – moduł kalendarza jest bardzo rozbudowany w tym systemie, duża część elementów integruje się z nim. Dodatkowo można wykonywać akcje z kalendarza oraz grupować różne kalendarze tematycznie

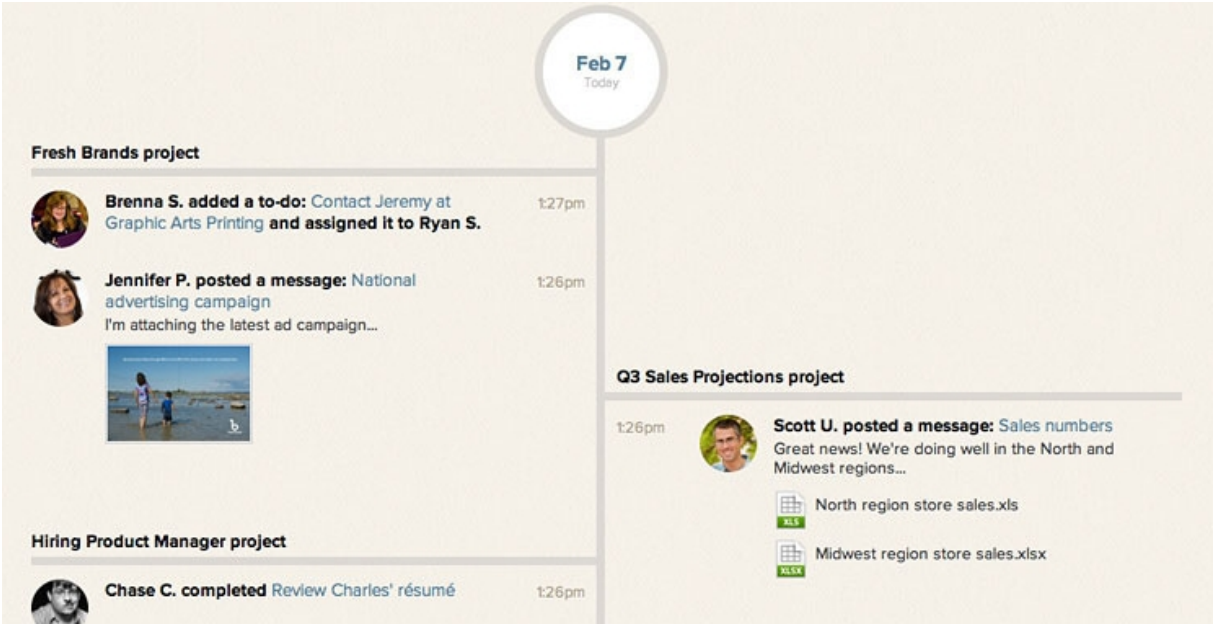


Rysunek 13: Widok kalendarza w systemie Basecamp
Źródło: `http://basecamp.com/calendar`

- zarządzanie plikami – Basecamp łatwo pozwala przeglądać i wyszukiwać pliki. Dostęp do plików może odbywać się poprzez odpowiednie zadania lub przegląd wszystkich plików w projekcie – czego czasem brakuje w innych aplikacjach do zarządzania projektem
- silna kolaboracja i dyskusje - system ten jest silnie nastawiony także na aktywne dyskusje, w wielu miejscach można rozpocząć rozmowę z innymi użytkownikami lub zostawić

komentarz np. przy wgranym pliku

- raporty – Basecamp dostarcza dzienne raporty z aktualnych wydarzeń jakie miały miejsce, tak by informacje o postępie cały czas były dostępne dla osób odpowiedzialnych za projekt
- historia aktywności (rysunek 14) – zapisywane jest każde wydarzenie jakie miało miejsce w systemie, dzięki czemu łatwo można prześledzić co się wydarzyło i kiedy



Rysunek 14: Przebieg wydarzeń w Basecamp
Źródło: http://basecamp.com/timeline

- organizacja maili (rysunek 15) – zazwyczaj większość komunikacji z klientami za granicą odbywa się drogą elektroniczną (z powodu odległych lokalizacji). Jednym z większych problemów jakie rozwiązuje Basecamp jest organizacja tych wiadomości oraz angażowanie osób zewnętrznych, które nie biorą bezpośrednio udziału w projekcie. Basecamp pozwala dołączyć do rozmowy osoby, które nie posiadają nawet konta w systemie i cała rozmowa poprzez emaile trafia do ich bazy, by móc później z niej korzystać jako referencję

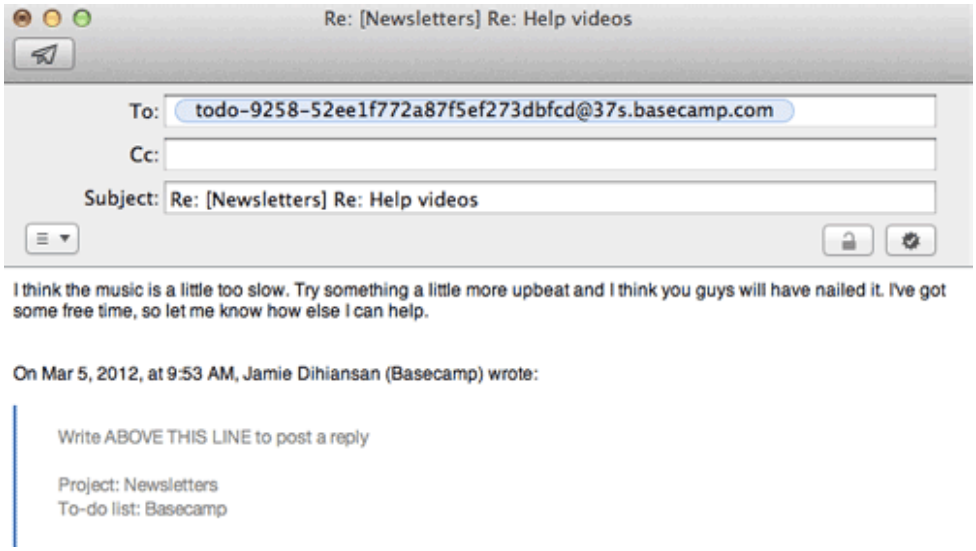
Basecamp posiada bardzo przyjazny interfejs użytkownika, jest przemyślanie zaprojektowany by korzystało się z niego jak najłatwiej i jak najszybciej, jednak przy ogromnych projektach sposób prezentacji danych może zacząć przeszkadzać.

2.5 JIRA

JIRA to bardzo potężne, komercyjne narzędzie stworzone w technologii Java dostępne za pomocą przeglądarki internetowej.

Główne funkcjonalności jakie oferuje JIRA:

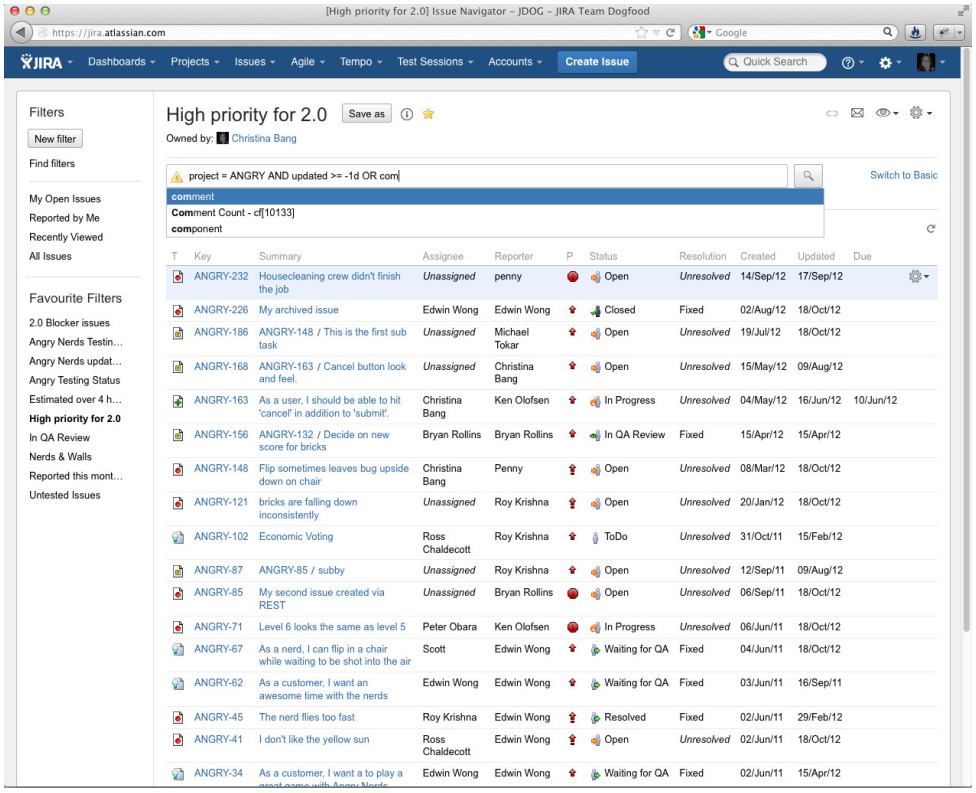
- zarządzanie zadaniami – zadania można w pełni dostosowywać, tworzyć połączenia między nimi, tworzyć podzadania, określać ich typ, dodawać obserwatorów, urządzać głosowanie.



Rysunek 15: Automatyczna organizacja wiadomości w Basecamp
Źródło: http://basecamp.com/loop-in

Ponadto można ustalać własny harmonogram pracy lub łączyć zadania z kodem źródłowym tworzonego programu

- sterowanie klawiaturą – wbudowane skróty klawiaturowe ułatwiają i przyśpieszają pracę
- komunikacja – wewnętrzny system komunikacji pozwala na sprawną wymianę informacji
- integracja IDE – JIRA posiada rozszerzenia, które można zainstalować dla środowisk programistycznych, dzięki czemu można np. śledzić błędy bezpośrednio z IDE
- zewnętrzne systemy – umożliwia integrację z dodatkowymi narzędziami
- zaawansowany język zapytań (rysunek 16) – tak samo jak w przypadku YouTrack, JIRA zawiera własny język do szybkiego wyszukiwania danych
- raportowanie – mini gadżety dołączane do systemu pozwalają na bieżący monitoring poszczególnych elementów projektu, można także generować złożone raporty dla projektów
- definiowanie przepływów (rysunek 17) – możliwość wizualnego tworzenia przepływów upraszcza modelowanie procesu
- import danych – JIRA pozwala zaimportować dane z wielu konkurencyjnych aplikacji
- tworzenie dokumentów z bogatymi treściami
- integracja z systemem kontroli wersji – tak jak w przypadku systemu Redmine można przeglądać kod z poziomu platformy
- weryfikacja kodu – dzięki JIRA można weryfikować kod źródłowy programów i dodawać komentarze co usprawnia wymianę informacji między programistami (zwłaszcza w przypadku pracy zdalnej)
- śledzenie aktywności – wydarzenia zachodzące w systemie są zapisywane i można je przeglądać



Rysunek 16: Lista zadań oraz język zapytań dla systemu JIRA
Źródło: `http://www.atlassian.com/project-management-software`

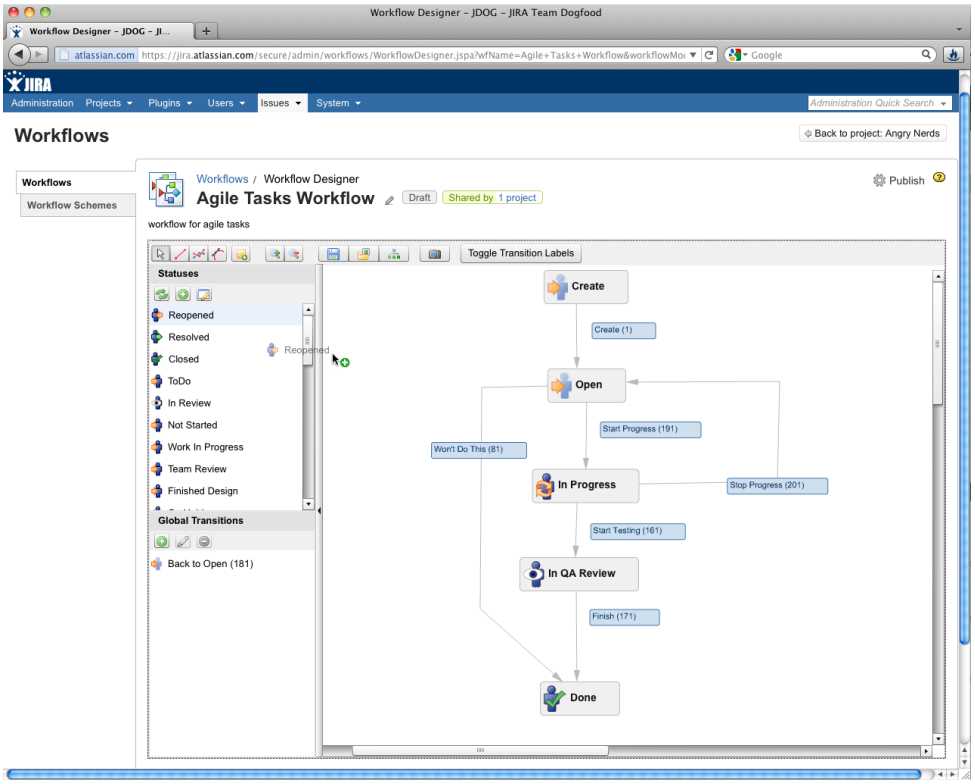
JIRA posiada ogrom możliwości, kumuluje niemal wszystkie funkcjonalności wyżej opisanych systemów oraz bardzo dobrze sprawdza się dla wielkich projektów.

2.6 Visual Studio Achievements Extension

Visual Studio Achievements Extension co prawda nie jest aplikacją do zarządzania projektem, ale jest bardzo dobrym przykładem jak elementy grywalizacji potrafią wpływać na wydajność programistów w sektorze IT.

Wtyczka jest zintegrowana ze środowiskiem pracy programisty i działa w tle. Zbiera informacje za każdym razem, gdy programista używa swojego IDE lub kompiluje program. W tle wyliczane są odznaczenia. Gdy użytkownik zdobędzie nową nagrodę jest o tym powiadamiany, a jego wynik publikowany. Dodatkowo ma możliwość podzielenia się tą informacją na serwisie społecznościowym.

Co więcej takimi odznakami można nawet lekko sterować jakością oprogramowania. Jeżeli programista zdobędzie pozytywną odznakę to wie, że robi coś dobrze. Dzięki czemu za pomocą systemu wynagrodzeń można wymuszać tzw. "*best practices*" na realizowanych projektach czyli zwiększać jakość oprogramowania.



Rysunek 17: Definiowanie przepływu w JIRA
Źródło: http://www.atlassian.com/project-management-software

2.7 PROPS to you

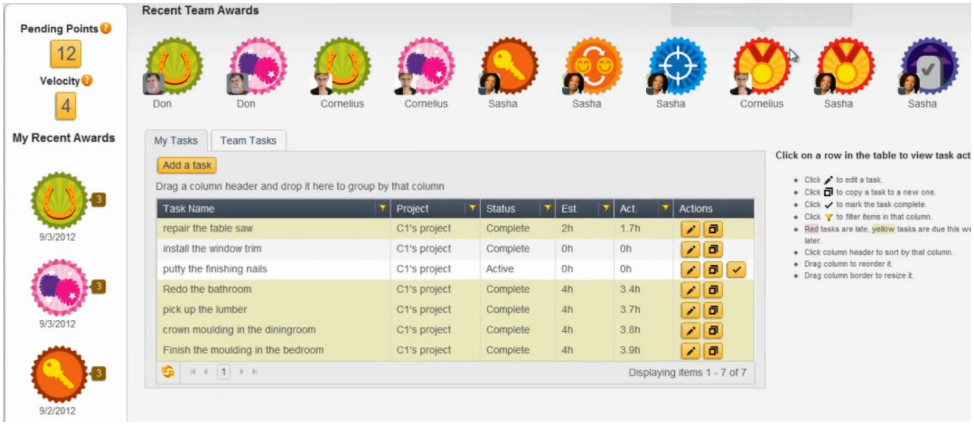
„PROPS to you" lub PTY to aplikacja dostępna z poziomu przeglądarki internetowej. Jest to system, które integruje w sobie cechy narzędzia do zarządzania projektami oraz elementy grywalizacji.

W kontekście projektów IT zawiera podstawowy zestaw funkcjonalności:

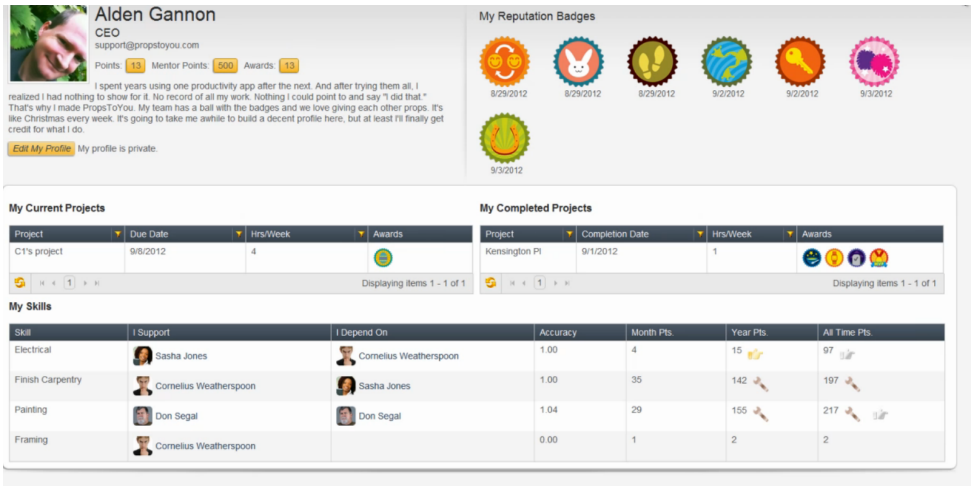
- tworzenie/edycja zadań – można zarządzać prostymi zadaniami, każde zadanie jest przypisane do konkretnej osoby i posiada opis (przykładowa lista zadań znajduje się na rysunku 18)
- monitorowanie terminów projektów – śledzenie dat zakończenia projektów
- wgrywanie plików – PTY pozwala na wgranie załączników do projektu
- generowanie raportów – można generować raporty z przebiegu pracy i szacować termin ukończenia projektu
- monitorowanie czasu

Aplikacja ta w odróżnieniu od systemów przedstawionych w rozdziałach 2.2, 2.3, 2.4, 2.5 posiada system odznak i osiągnięć, jakimi nagradzani są użytkownicy. Zdobywanie nagród polega na wykonywaniu zadań. Czasami jednak ktoś może ofiarować nagrodę za pomoc lub oznaczyć zadanie specjalną odznaką, którą inny użytkownik może zdobyć poprzez udzielenie pomocy. Swoje osiągnięcia można śledzić na własnym profilu (Rysunek 19).

System PTY w dużym stopniu nastawiony jest na grywalizację, jednak jest trochę za mały i za ubogi pod kątem zarządzania projektem.



Rysunek 18: Widok listy zadań w PTY
Źródło: Opracowanie własne



Rysunek 19: Widok własnego profilu w PTY
Źródło: Opracowanie własne

2.8 Wady istniejących rozwiązań

Redmine, YouTrack, Basecamp, JIRA bardzo dobrze spełniają swoje założenia w dziedzinie zarządzania projektem informatycznym. Oferują szereg różnorakich funkcjonalności niezbędnych do sprawnej pracy zespołu m.in.: zarządzanie zadaniami, terminami, czasem, generowanie raportów, wymiana informacji lub danych. Funkcjonalności te są wykonane w unikalny sposób dla każdego z systemów jednak żaden z nich nie posiada elementu eliminującego jedno z największych zagrożeń dla tworzonego projektu – zahamowanie motywacji pracownika.

Jakość tworzonego oprogramowania w zespołach programistycznych często spada po jakimś czasie, ludzie stają się przemęczeni i znudzeni. Obniżona jest produktywność, przez co dostarczana jest gorsza jakość, co bezpośrednio przekłada się na rezultat projektu.

Opisane rozwiązania nie są w stanie poradzić sobie z tym problemem. W przypadku systemu Redmine nie jesteśmy w stanie tego dobrze wyśledzić, jedyne co można zauważyć to np. przekroczone terminy lub dużą ilość błędów na liście zadań do rozwiązania. Odstraszający wygląd Redmine wcale nie zachęca do ich zamykania.

System YouTrack posiada natomiast lepszy wygląd i bardziej zachęca do realizacji zadań poprzez *agile board*. Dla przykładu załóżmy, że zadania mogą znaleźć się w 3 stanach (Otwarte → w trakcie realizacji → Zrobione) – widzimy wtedy rozłożenie zadań w 3 kolumnach. Te 3 kolumny podświadomie przypominają wyścig, który jest małym czynnikiem motywującym, ale nieco pomaga ponieważ żaden z pracowników nie chce zostać sam (czyli ostatni) na liście po lewej (linia startu) tylko chce przed innymi znaleźć się na ostatniej liście (linia mety). Tak drobny czynnik natomiast nie wystarczy na dłuższy okres czasu dlatego zaangażowanie pracowników również spadnie po czasie.

Basecamp zachęca swoim wyglądem. Listy zadań przedstawione są w przyjazny sposób. Zamknięcie zadania odbywa się pojedynczym kliknięciem dlatego użytkownicy lubią je zamykać, ale żeby zamknąć zadanie trzeba je jeszcze wykonać. Basecamp posiada atrakcyjny wygląd, z którym dobrze się pracuje, ale jako system nie posiada czynników motywujących.

JIRA także posiada *agile board* co lekko zwiększa motywację pracowników. Ma dość bogaty system powiadomień, ale duża liczba wiadomości, które są tekstem tylko sztywno opisującym co się dzieje na projekcie nie zwiększa zaangażowania, a czasem przytłacza i działa niekorzystnie.

Przedstawione narzędzia nie posiadają elementów, które pobudzą pracowników do działania i produktywność ludzi będzie spadać. Zapobiec takiemu scenariuszowi można poprzez wprowadzenie mechanizmów motywujących, które spowodują, że produktywność wzrośnie – takim motywatorem jest grywalizacja.

Visual Studio Achievements Extension oraz PROPS to you radzą sobie z problemem motywacji znacznie lepiej, jednak są zbyt ubogie w funkcjonalności w porównaniu z systemami, które skupiają się wyłącznie na zarządzaniu projektem.

3 Wykorzystane narzędzia i technologie

Bieżący rozdział opisuje narzędzie i technologie wykorzystane podczas pracy.

3.1 Aplikacja Webowa

W niniejszym rozdziale zawarto krótki opis technologii użytych do budowy aplikacji webowej.

3.1.1 Ruby

Ruby to obiektowy, interpretowany język programowania z dynamicznym typowaniem, który został stworzony przez Yukihiro Matsumoto (pseudonim Matz) w 1995r. Kreator języka chciał stworzyć język z idealną składnią (bazując na jego ulubionych językach m.in. Smalltalk czy Perl) [5, 6].

Ruby jest bardzo elastycznym językiem programowania, pozwala programistom w pełni zmieniać, dodawać czy usuwać swoje części (programista nie jest ograniczony niczym – zawsze może nadpisać domyślne zachowanie czegokolwiek).

Wybrane cechy języka:

- dynamiczne typowanie
- pojedyncze dziedziczenie (dodatkowe rozszerzanie klas przez mixiny)
- garbage collector
- domknięcia, bloki, iteratory, generatory
- przeciążanie operatorów
- meta-programowanie
- literały
- obsługa wyjątków
- wątki
- parametry domyślne
- możliwość pisania rozszerzeń w C
- wieloplatformowość
- duża ilość bibliotek standardowych

3.1.2 Ruby On Rails

Ruby on Rails – jest to framework MVC o otwartym kodzie źródłowym napisanym w języku Ruby używany do tworzenia aplikacji webowych. Został on stworzony przez duńskiego programistę David'a Heinemeier'a Hansson'a. Framework Ruby On Rails (potocznie Rails lub RoR) został wypuszczony do sieci w 2005r i szybko zyskał popularność wśród programistów, dzięki czemu sam język Ruby został szeroko rozpowszechniony [7].

Lista wybranych bibliotek (GEMs) użytych podczas pracy:

- pg – adapter do obsługi bazy danych PostgreSQL
- devise – moduł do autentykacji
- paperclip – moduł generowania miniatur obrazków i wgrywania plików
- aws-sdk – SDK dla języka Ruby pozwalające na połączenia z usługami AWS
- formtastic – DSL dla budowy formularzy webowych
- friendly_id – generator unikalnych URL na bazie nazw
- jquery-fileupload-rails – biblioteka pozwalająca na wygodne wgrywanie plików przez użytkownika
- inherited_resources – moduł wspomagający tworzenie kontrolerów
- ckeditor – edytor WYSIWYG
- highcharts-rails – biblioteka ułatwiająca wyświetlanie wykresów
- rabl – DSL dla budowy API
- rufus-lua – biblioteka pozwalająca na wykonywanie skryptów w języku Lua
- redcarpet – biblioteka zamieniająca tzw. „markdown" na postać HTML

3.1.3 PostgreSQL

PostgreSQL (lub Postgres) to darmowa, relacyjna baza danych. Działa na bardzo wielu systemach. W pełni kompatybilna z właściwościami ACID. Implementuje większość standardu SQL:2008 [8].

3.1.4 Lua

Lua to szybki i lekki język skryptowy, który często osadza się w innych językach programowania. Zawiera podstawowe struktury danych oraz instrukcje sterujące. Często używany w grach jako warstwa skryptowa [9].

3.1.5 HTML

HTML (HyperText Markup Language) – hipertekstowy język znaczników, obecnie jest standardem do budowania zawartości i struktury stron www. HTML składa się ze specjalnych znaczników tzw. tagów, z których buduje się szkielet strony. Na ich podstawie przeglądarki internetowe interpretują strukturę dokumentu i wyświetlają odpowiednią sformatowaną treść [10].

3.1.6 CSS/SCSS

CSS (Cascading Style Sheets) – kaskadowe arkusze stylów, definiują, w jaki sposób przeglądarka internetowa ma wyświetlić poszczególne znaczniki HTML, może je dodatkowo upiększyć – zmienić znacząco ich wygląd. CSS zostało zaprojektowane, aby odseparować treść

dokumentu od warstwy prezentacji dokumentu. Usprawnia to współdzielenie formatowania między różnymi stronami w danym systemie [10].

SCSS (Sassy CSS) – jest to język zbudowany na bazie CSS. Składnia SCSS pozwala na łatwiejsze czytanie stylów (przejrzysty kod źródłowy), z których później generowane są pliki CSS [11].

3.1.7 JavaScript/jQuery

JavaScript to język skryptowy, został stworzony, aby rozbudować możliwości interakcji użytkownika z interfejsem opartym na samym HTML. Jest to język obiektowy z dynamicznym typowaniem. Ze względu na to, że niektóre przeglądarki obsługują JavaScript w różnych wersjach lub nie obsługują niektórych jego funkcji powstaje wiele bibliotek, które zapewniają wykonanie właśnie takich funkcji niezależnie od przeglądarki [12].

Jedną z takich bibliotek jest jQuery, która zapewnia nam manipulację strukturą dokumentu HTML zawsze w taki sam sposób niezależnie od używanej przeglądarki. jQuery wykrywa przeglądarkę i wykonuje specyficzny kod dla niej, zwalniając programistę z pisania tych warunków czego efektem jest przejrzysty kod [13].

3.1.8 Twitter Bootstrap

Twitter Bootstrap to biblioteka stworzona przez programistów serwisu Twitter. W skład biblioteki wchodzą narzędzia, które ułatwiają konstrukcję interfejsów. Biblioteka składa się z plików CSS i JavaScript [14].

3.1.9 ACE editor

Edytor ACE to edytor kodu źródłowego, który jest osadzony w przeglądarce. Został on stworzony w JavaScript [15].

3.1.10 CKEditor

CKEditor to edytor HTML, który został stworzony, aby ułatwić tworzenie bogatych treści na stronach www. Jest to edytor typu WYSIWYG. Zawiera wiele dodatków, które rozszerzają jego możliwości [16].

3.2 Aplikacja Mobilna

Poniżej zaprezentowano zestaw technologii użytych do stworzenia aplikacji mobilnej.

3.2.1 Java

Java to język stworzony przez firmę Sun Microsystems. Kod javy jest kompilowany do tzw. bajt kodu, który jest wykonywany przez maszynę wirtualną Javy, co sprawia, że kod piszemy raz i z założenia każda platforma z zainstalowaną maszyną wirtualną Javy będzie w stanie uruchomić program. Obecnie Javę rozwija firma Oracle [17].

Wybrane cechy Javy:

- silne typowanie zmiennych
- darmowa
- niezależność od platformy
- bezpieczeństwo oraz niezawodność
- obiektowość
- szerokie pole zastosowań
- automatyczne zarządzanie pamięcią

3.2.2 Android SDK

Android SDK to zestaw bibliotek i narzędzi potrzebnych do tworzenia aplikacji na platformę Android [18].

3.2.3 AQuery

AQuery (Android-Query) – to lekka biblioteka, która ułatwia wykonywania zadań asynchronicznych, tworzenie cache'u dla obrazków, a także manipulację interfejsem graficznym [19].

3.2.4 GSON

Gson to biblioteka stworzona w języku Java, która ułatwia konwersję danych z formatu JSON do odpowiedniego obiekty Javy i odwrotnie [20].

3.3 Narzędzia i środowisko pracy

Niniejszy rozdział poświęcony jest omówieniu narzędzi wykorzystanych do implementacji prototypu.

3.3.1 Sublime Text 2

Sublime Text to prosty, bardzo konfigurowalny edytor kodu źródłowego. Jego największą zaletą jest, że działa bardzo wydajnie [21].

3.3.2 Eclipse Juno z ADT

Eclipse to środowisko programistyczne, które jest modularne i łatwo rozszerzalne przez inne wtyczki. Głównym jego przeznaczeniem jest język Java, jednak istnieją wtyczki, które pozwalają bezproblemowo używać innych języków programowani [22].

ADT to zestaw wtyczek przygotowanych przez Google dla Eclipse, które ułatwia pracę przy platformie Android.

3.3.3 Git

Git to darmowy, rozproszony system kontroli wersji. Działa bardzo szybko i wydajnie [23].

4 Propozycja nowego rozwiązania

Jak wspomniano w rozdziale 1.1 celem pracy jest opracowanie koncepcji narzędzia zwiększającego motywację pracowników, które pomoże usprawnić jakość tworzonych projektów. W dalszej części rozdziału przedstawiono wymagania jakie musi ono spełniać, a także opis podejścia, które pomoże osiągnąć założenia.

4.1 Panel administracyjny

W panelu administracyjnym istnieje możliwość zarządzania projektami oraz użytkownikami. Są to rzadko wykonywane czynności ponieważ często nie tworzy się nowych projektów jak również użytkowników. Jednakże są one niezbędne do funkcjonowania systemu i rozpoczęcia pracy z nim.

Moduł projektów w sekcji administracyjnej pozwala na tworzenie i edycję projektów. Dla projektu można zdefiniować nazwę, logo będące identyfikatorem wizualnym oraz listę użytkowników, którzy mają dostęp do niego.

Moduł użytkowników natomiast umożliwia ich tworzenie, edycję oraz ustawienia ról, które determinują zakres działań w całym systemie.

Ponadto każdy użytkownik ma dostęp do swojego profilu gdzie może edytować swoje dane logowania oraz imię wraz z nazwiskiem.

4.2 Projekty

W momencie gdy istnieją już projekty użytkownicy mogą korzystać z ich wewnętrznych funkcjonalności przeznaczonych do codziennego użytku. W ich skład wchodzą następujące moduły:

- Zadania
- Pliki
- Wiadomości
- Treści
- Raporty
- Dynamiczne elementy grywalizacji

Wszystkie te moduły zostaną omówione w dalszej części rozdziału.

Ponadto każdy projekt składa się ze sprintów. Sprint to jedna z faz projektu realizowana w określonym czasie. Tworząc sprint należy podać jego nazwę oraz datę rozpoczęcia i zakończenia. Każdy projekt zawiera specjalny sprint tzw. *backlog*, gdzie umieszcza się zadania jeszcze nieokreślone lub planowane na przyszłość.

4.3 Zadania

Zadania są najważniejszą składową każdego systemu do zarządzania projektami. Każde zadanie składa się z:

- nazwy – krótkie zdefiniowanie zadania
- rodzaju określającego kategorię zadania:
  - błąd (*Bug*)
  - funkcjonalność (*Feature*)
  - usprawnienie (*Improvement*)
  - zwykłe zadanie (*Task*)
  - scenariusz (*User story*)
  - super usprawnienie (*Epic*)
- statusu – jako jednego z poniższych:
  - do zrobienia (*TODO*)
  - w trakcie realizacji (*In progress*)
  - gotowe (*Done*)
- procentu wykonania
- priorytetu:
  - niski
  - średni
  - wysoki
- osoby odpowiedzialnej za wykonanie
- estymacji czasowej na jego realizację
- opisu – dokładny opis zadania (opcjonalny)

Dodatkowo musi mieć przypisany sprint lub znaleźć się w backlog'u. Opcjonalnie może one być przypisane do innego zadania jako podzadanie. System powinien także umożliwiać przenoszenie zadań z backlog'a do konkretnego sprintu.

4.4 Pliki

Podczas pracy niejednokrotnie zachodzi potrzeba wymiany plików. Ludzie często wymieniają pliki poprzez e-mail, program Skype lub nośniki zewnętrzne. Są to raczej złe praktyki. Każdy plik dotyczący projektu powinien się znaleźć w jego przestrzeni. Kreowane narzędzie powinno umożliwiać dodanie plików do projektu, tak by każdy z pracowników miał do niego dostęp. Pliki powinny być pogrupowane względem czasu ich wgrania.

4.5 Wiadomości

Centralizacja wiadomości do systemu zarządzania projektem ma kilka zalet. Jedną z nich jest śledzenie historii wypowiedzi – nie trzeba szukać starych wiadomości w email'ach lub chat'ach gdy chcemy się do czegoś odwołać bo wszystko znajduje się w stałym miejscu.

Inną zaletą przechowywania rozmów jest to, że członkowie zespołu mogą śledzić co się dzieje aktualnie na projekcie nawet jak ta rozmowa ich nie dotyczy – eliminuje to potrzeby informowania pracowników o tym co ustalono w pewnych kwestiach – sami mogą do tego dotrzeć.

System wiadomości musi przyjąć postać forum, aby móc kategoryzować rozmowy przynajmniej na poziomie tematów.

4.6 Treści

Moduł treści to miejsce gdzie można przechowywać wszelakie informacje na temat projektu wliczając w to obszerne dokumentacje, pliki konfiguracyjne, instrukcje instalacji lub wdrożenia oraz wiele innych. Jest to tzw. „wiki" projektu. Aby zapewnić dowolność takich treści powinny one być tworzone za pomocą uproszczonego języka znaczników, który pozwoli formatować tekst („*markdown language*").

4.7 Raporty

Kreowany system musi pozwolić na minimalny zestaw wykresów o przebiegu projektu, aby można było szybko zwizualizować stan projektu bez wchodzenia w szczegóły.

4.8 Dynamiczne elementy grywalizacji

Zbudowanie aspektów motywujących pracowników w systemie do zarządzania projektami nie jest łatwym przedsięwzięciem. Czynniki motywujące powinny starać się całkowicie wyeliminować istniejącą nudę w pracy, monotonię oraz wszystkie czynniki, jakie wpływają na pomniejszenie zaangażowania ludzi. Osiągnąć można to poprzez wprowadzenie elementów grywalizacji.

Nawet przy opracowaniu planu rozwiązania tego problemu i stworzeniu mechaniki gry, który zapewni sukces realizowanego projektu – pozostaje pytanie jak wdrożyć te elementy do oprogramowania, którego używamy.

Esencją jest stworzenie dynamicznych elementów grywalizacji, aby móc ciągle dostosowywać rozgrywkę dla graczy i bilansować strefę przepływu jak opisano w rozdziale 2.1.2. Bilans można uzyskać poprzez opisywanie zachowań tych elementów za pomocą skryptów. Podobną techniką posługują się programiści gier komputerowych, którzy mając do dyspozycji prawie gotową grę lub silnik gry używają skryptów do parametryzacji rozgrywki, stopnia trudności i udoskonalenia tzw. „gameplay'u".

W dalszej części rozdziału przedstawiono właściwości potrzebne do definiowania dynamicznych elementów grywalizacji, aby móc stworzyć różnorodne mechaniki gier. Każdy projekt posiada własny zestaw tych właściwości co powoduje, że każda rozgrywka będzie inna.

4.8.1 Role graczy

Role pozwolą na lepszą identyfikację graczy w rozgrywce. Gracz to pracownik, który wykonuje zadania na danym projekcie. Rozgrywka to realizowany projekt. Rola polega na nadaniu

dodatkowej nazwy członkowi projektu, która przenika ze światem gry. Gracz czuje się potrzebny i bardziej zaangażowany ponieważ przydzielono mu fikcyjną postać, którą będzie grał.

4.8.2 Pamiętnik

Pamiętnik jest odzwierciedleniem tego co dzieje się w grze (wirtualnym świecie projektu). Wszystkie wpisy w nim są generowane automatycznie na bazie przebiegu rozgrywki. Zawiera on informacje takie jak:

- opisy elementów fabuły
- opis zdarzeń
- misje gry
- zdobyte/utracone punkty przez gracza
- przedmioty jakie gracz dostał/utracił
- zdobyte odznaki

Nad pamiętnikiem gracz powinien widzieć swój aktualny status w grze razem ze wszystkimi właściwościami jakie zostały zdefiniowane dla danej gry.

Ponadto z pamiętnika można przejść do tabeli wyników gdzie gracze mogą porównywać się między sobą i sprawdzać kto najlepiej sprawdza się w danej kategorii.

4.8.3 Kolekcje

Kolekcje to wszystkie wirtualne przedmioty oraz dobra jakie gracz może zyskać lub znaleźć w grze.

Tworzenie obiektów gry jakie gracz może kolekcjonować polega na uzupełnieniu informacji obiektu. Każdy obiekt składa się z następujących atrybutów:

- unikalny identyfikator – referencja do obiektu w projekcie
- nazwa – nazwa przedmiotu
- ikona – obrazek reprezentujący wirtualny przedmiot

Istnieją tylko dla danego projektu. Mogą być one przyznawane za rzadko występujące czynności wykonane na projekcie np. przyznanie złota za rozwiązanie zadania, które ma wysoki priorytet i jest typu „super zadanie", ale nie jest to sztywną regułą – mogą być przyznawane częściej.

4.8.4 Punkty

Punkty dzielą się na dwa typy: zwykłe punkty oraz paski postępu. Zwykłe punkty to wartość liczbowa mająca początkową wartość, mogą rosnąć w nieskończoność – przykładem takiego rodzaju punktów są punkty doświadczenia. Paski postępu to wartość liczbowa określona na przedziale dwóch liczb, także może mieć swoją wartość początkową – przykładem jest pasek

zdrowia postaci, który jako wartość minimalną przyjmuje 0, a jako wartość maksymalną 100. Istnieją tylko dla danego projektu.

Stworzenie systemu punktów polega na:

- określeniu unikalnego identyfikatora w projekcie
- nadaniu nazwy
- ustawienia wartości początkowej
- ustawienia wartości minimalnej
- ustawienia wartości maksymalnej (opcjonalne jeśli są to punkty rosnące w nieskończoność)

W kontekście zarządzania projektem jedną z opcji może być analiza zadania pod kątem jego typu. W projekcie pojawia się zadanie typu „błąd". Pamiętnik powinien wypisać informację o tym fakcie (np. „Użytkownik został zaatakowany przez potwora") oraz wyświetlić odpowiedni obrazek (rysunek potwora). Aby zbliżyć się do mechaniki gier komputerowych można odjąć kilka punktów zdrowia w celu mobilizacji gracza do zmierzenia się z nim. Po pokonaniu potwora (zakończeniu zadania) użytkownik może odzyskać stracone punkty zdrowia i zdobyć kilka punktów doświadczenia. Jeszcze lepiej jak pamiętnik wyświetli obrazek martwego potwora, co zwiększy satysfakcję gracza.

4.8.5 Obrazki

Obrazki to wizualne obiekty gry. Zapewniają lepsze wrażenia i odczucia podczas gry niż sam tekst. Obrazek składa się z:

- unikalnego identyfikatora
- nazwy
- ikony

4.8.6 Odznaki

Odznaczenia pozwalają graczom wyróżnić się na tle innych. Każda odznaka ma swoją nazwę i jest przyznawana graczowi za jakieś osiągnięcie. W odróżnieniu od pozostałych właściwości tj. punkty czy kolekcje przyznane odznaki pojawiają się na globalnym profilu użytkownika, tak aby każdy gracz mógł się pochwalić jakie odznaczenie, za co i w których projektach zdobył.

Definicja odznaki:

- unikalny identyfikator
- nazwa
- ikona

Przykładowo użytkownik rozwiązał bardzo szybko zadanie na projekcie – jako nagrodę mógłby otrzymać odznaczenie z odpowiednim obrazkiem oraz tytułem „Najszybciej rozwiązane zadanie". Innym podejściem mogłaby być analiza słów kluczowych zadania i przyznawanie odznak tematycznych.

4.8.7 Logika gry

Logika gry to serce całej mechaniki. Jest to warstwa skryptowa, która pozwala dynamiczne sterować właściwościami gry, które opisano wyżej. Skrypty gry są tworzone dla każdego sprintu z osobna, aby wymusić większą różnorodność rozgrywki oraz kreowanych elementów gry.

Dla każdego skryptu należy stworzyć jego kod źródłowy oraz oznaczyć jako aktywny, żeby był interpretowany w systemie do zarządzania projektami – można stworzyć wiele małych plików lub jeden duży – w zależności od preferencji. Aktywne skrypty są ewaluowane podczas tworzenia i aktualizacji zadań.

Mając już gotowy pomysł fragmentu gry trzeba go przenieść na logikę gry. Jej kod jest tworzony w języku skryptowym Lua. Tworząc go mamy do dyspozycji kilka ważnych informacji na temat sprintu, aby móc podejmować decyzje w skrypcie, a także informacje skąd nadeszło zdarzenie. Wbudowane API grywalizacji w skrypty pozwala na manipulację elementami użytkownika tj. punkty, kolekcje czy odznaki. Istnieje także możliwość pisania do pamiętnika.

Wyjątkową zaletą takiego podejścia jest to, że nie ogranicza nas żaden kod sztywno zaszyty w aplikacji i możemy tworzyć dowolne typy gier, budować różne fabuły i kusić graczy (pracowników) by chcieli grać więcej. Jedyny limit to nasza wyobraźnia – należy jednak zwrócić uwagę na to jak kształtuje się rozgrywka, aby dostosować grę do graczy.

4.8.8 Historyjki

Historyjki to luźne paragrafy tekstu połączone z grafiką, których zadaniem jest rozbudowa fabuły gry. Mogę one zawierać także definicję zadań lub misji. Po napisaniu fragmentu historyjki jest ona publikowana w pamiętniku gry.

Nie wnoszą one nic specjalnego do mechaniki poza budową nastroju i zachęcenia graczy do głębszego poznania świata gry, a więc zwiększają zaangażowanie. W miarę regularne ich pisanie poprawi znacząco jakość gry.

4.9 Aplikacja mobilna

Aplikacja mobilna służy do monitorowania zadań projektu. Nie powinna ona umożliwiać żadnych zmian na projekcie, jest tylko narzędziem do szybkiego sprawdzenia stanu projektu i wykazaniu małych zestawień. Jej funkcjonalności to:

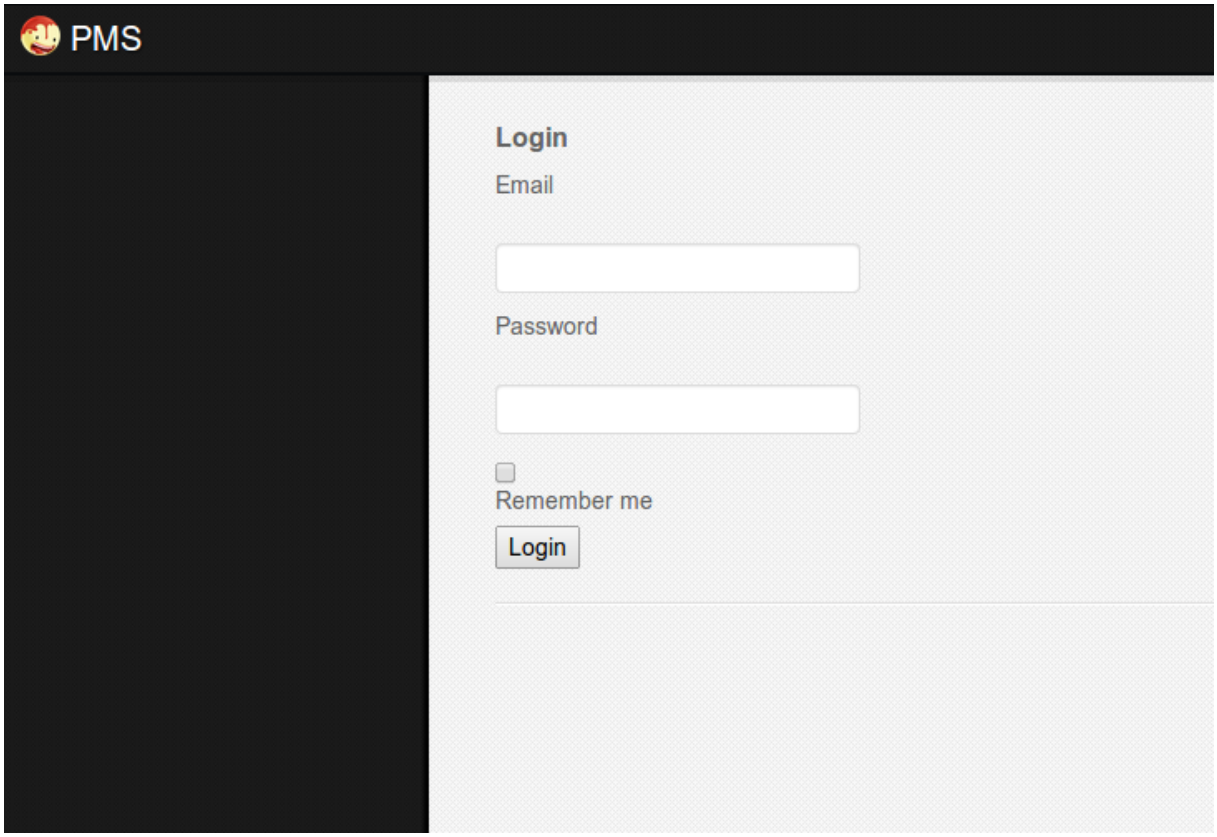
- przeglądanie projektów
- przeglądanie sprintów
- przeglądanie zadań
- przeglądanie raportu dla sprintów

5 Prototyp

W niniejszym rozdziale zawarto propozycję rozwiązania, służącego do zarządzania projektem informatycznym z wykorzystaniem elementów grywalizacji. Prototyp przyjął nazwę roboczą PMS od *Project Management Software*.

5.1 Interfejs użytkownika

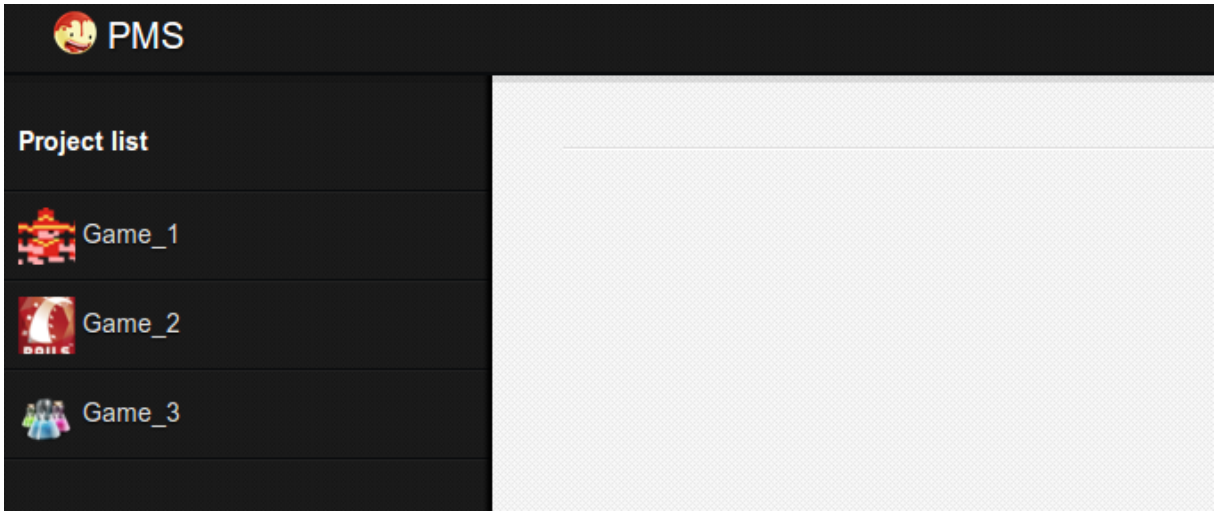
Po pierwszym uruchomieniu przeglądarki internetowej użytkownik zobaczy ekran logowania. Na tym ekranie musi on podać swój adres email oraz hasło, aby zalogować się do systemu (Rysunek 20). Cały interfejs użytkownika został podzielony na 4 części:



Rysunek 20: Logowanie do systemu
Źródło: Opracowanie własne

- górny pasek narzędzi – edycja profilu, wylogowanie, przejście do części administracyjnej
- lewy pasek boczny – głównie menu
- pozostały obszar – służący do wyświetlania treści
- stopka

Po zalogowaniu użytkownik powinien widzieć listę projektów w lewej części ekranu tak jak przedstawiono na rysunku 21.



Rysunek 21: Lista projektów dla użytkownika
Źródło: Opracowanie własne

Wybranie projektu z menu spowoduje przejście do jego głównego widoku, który znajduje się na rysunku 22. W lewym pasku znajduje się menu projektu, spośród, którego można przejść do poszczególnych modułów:

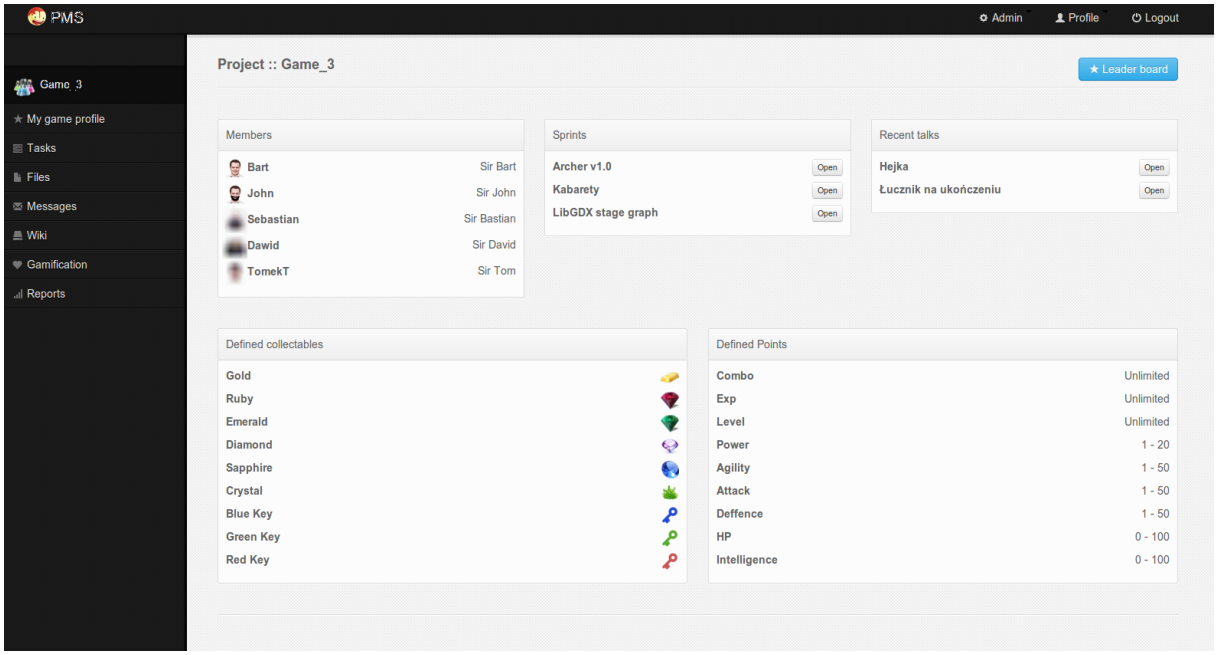
- Profil gry
- Zadania
- Pliki
- Wiadomości
- Tworzenie treści
- Grywalizacja
- Raporty

Na stronie głównej projektu znajduje się 5 ramek, które zawierają odpowiednio: listę członków projektu, listę sprintów, ostatnie rozmowy, zdefiniowane elementy kolekcji dla grywalizacji oraz punkty. Klikając w prawym górnym rogu na przycisk „Leader board" możemy bezpośrednio przejść do profilu gry do tablicy wyników graczy.

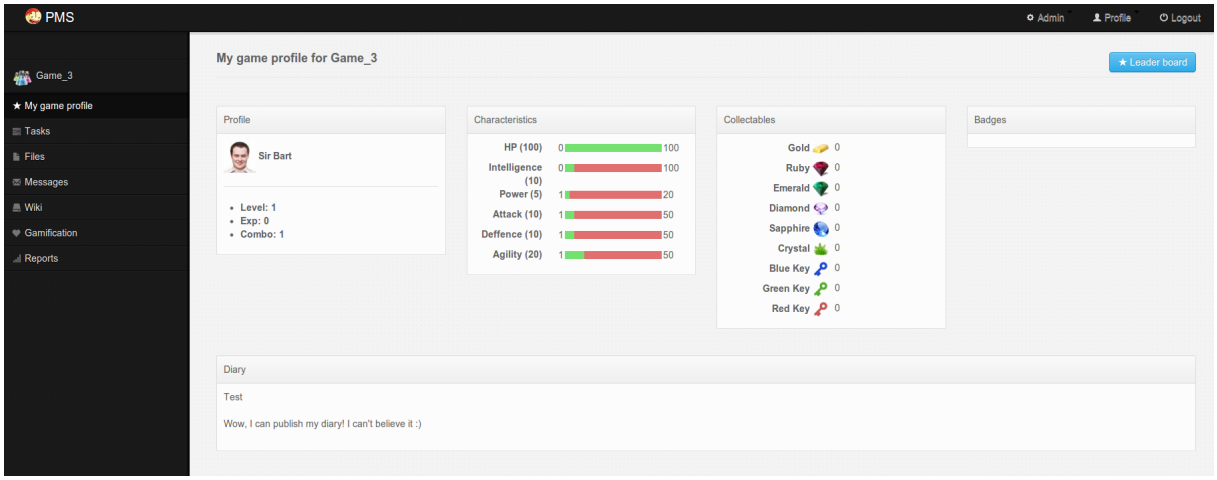
Profil gry znajdujący się na rysunku 23 składa się z 4 wewnętrznych ramek. Pierwsza ramka zawiera zdjęcia użytkownika, jego rolę w grze, a także zwykłe punkty. Druga z kolei punkty postępu. W trzeciej znajdują się kolekcje, a w ostatniej lista zdobytych odznaczeń w bieżącym projekcie.

Poniżej ramek jest pamiętnik, w którym są wszystkie wpisy z gry. Z tego widoku można także przejść do tablic wyników. Tablice wyników (rysunek 24) wyświetlają podsumowania dotyczące każdego z graczy na temat jego punktów oraz kolekcji.

Kolejnym elementem menu są zadania (rysunek 25). Widok ten zawiera listę sprintów dla projektu oraz *backlog*. Z tego widoku można dodać kolejne sprinty. Zadania są zaprezentowane



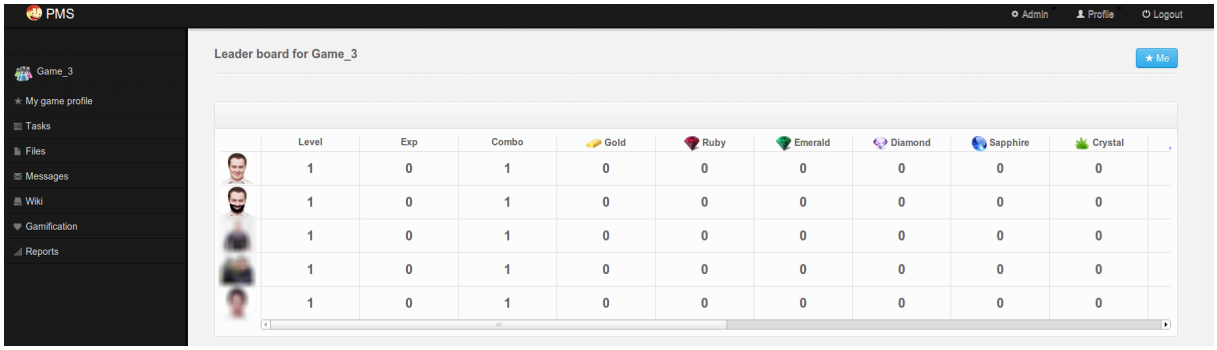
Rysunek 22: Główny widok projektu
Źródło: Opracowanie własne



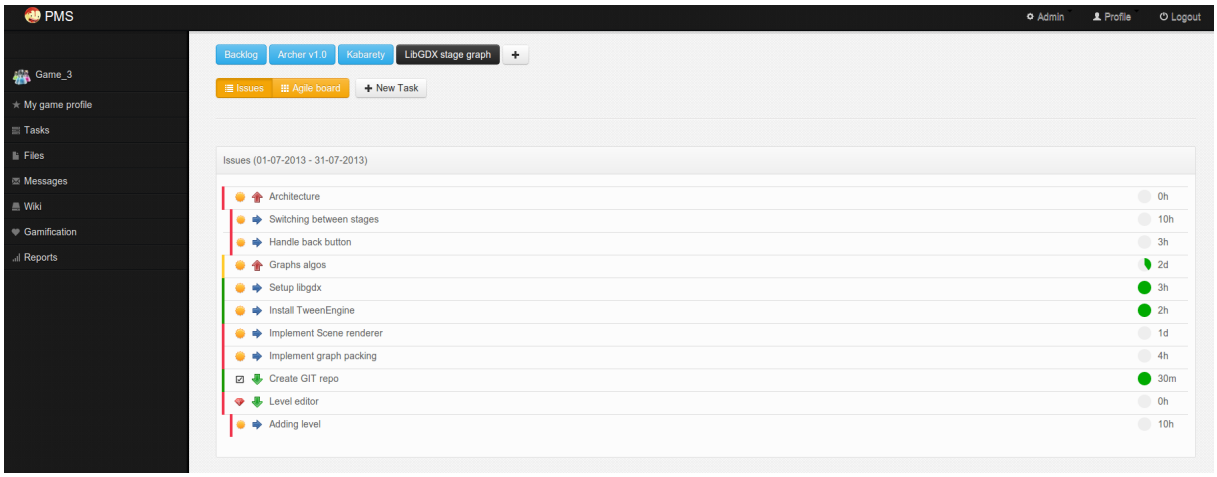
Rysunek 23: Profil gry
Źródło: Opracowanie własne

w postaci listy lub w postaci *agile baord'a* (rysunek 26). Lista zawiera wszystkie zadania, a każde z nich skrócony opis (summary), odpowiednie ikonki determinujące jego typ, priorytet, procent wykonania lub status. W przypadku widoku kolumnowego (*agile board*) widać listę tylko zadań nadrzędnych (brak podzadań). Zadanie te są zgrupowane po ich statusie.

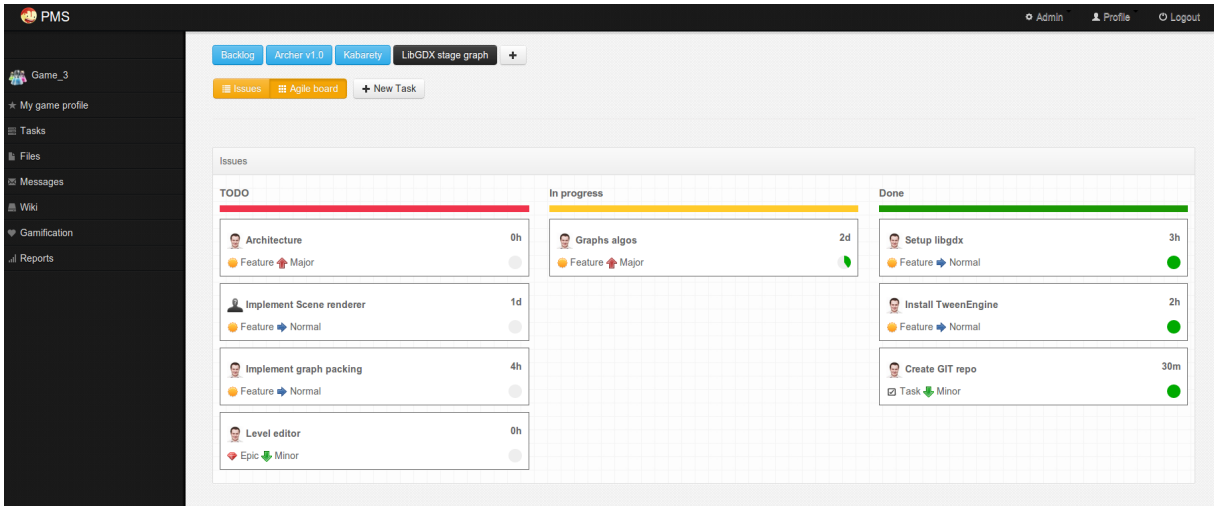
Nowe zadanie można stworzyć klikając przycisk „New Task". Użytkownik zostanie przekierowany do widoku znajdującego się na rysunku 27. Tworząc nowe zadanie trzeba określić ogólną jego treść, rodzaj, status, procent wykonania, priorytet, osobę oraz estymację. Nieobowiązkowo można przypisać je do innego i dodać długi opis.



Rysunek 24: Tabele wyników
Źródło: Opracowanie własne

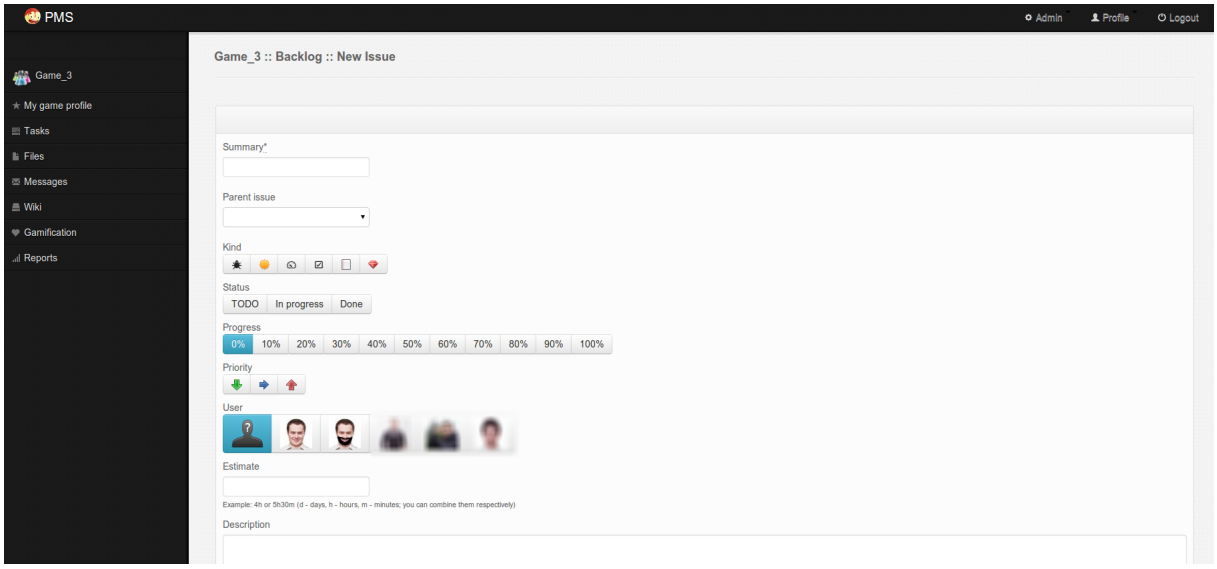


Rysunek 25: Zadania w postaci listy
Źródło: Opracowanie własne



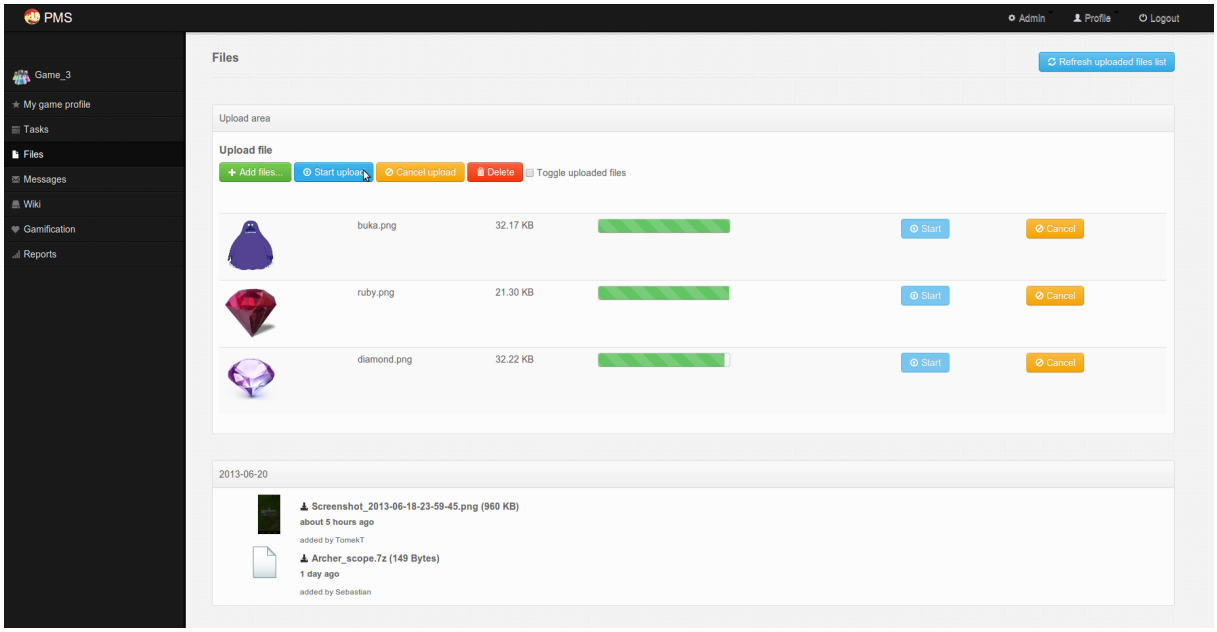
Rysunek 26: Zadania w postaci AgileBoard
Źródło: Opracowanie własne

Następnym modułem dla projektu są pliki. Składają się one tylko z pojedynczego widoku, na którym jest formularz oraz lista wgranych plików. Podgląd funkcjonalności znajduje się na



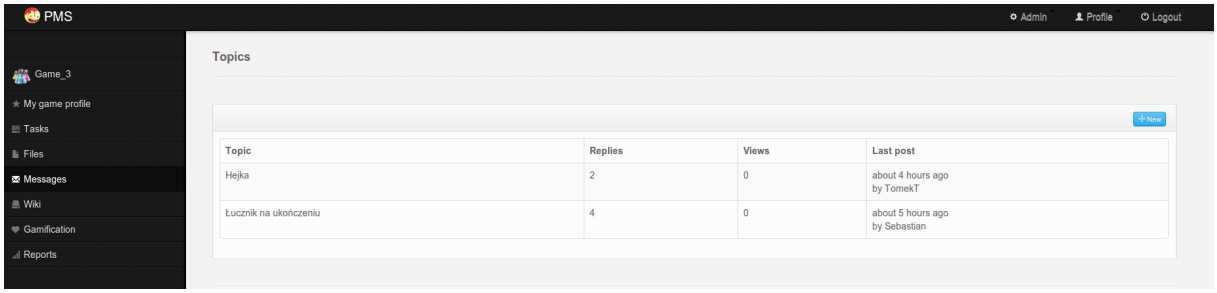
Rysunek 27: Tworzenie nowego zadania
Źródło: Opracowanie własne

rysunku 28. Istnieje możliwość wgrania wielu plików jednocześnie – wystarczy tylko je dodać do kolejki i kliknąć przycisk „start".



Rysunek 28: Panel do zarządzania plikami
Źródło: Opracowanie własne

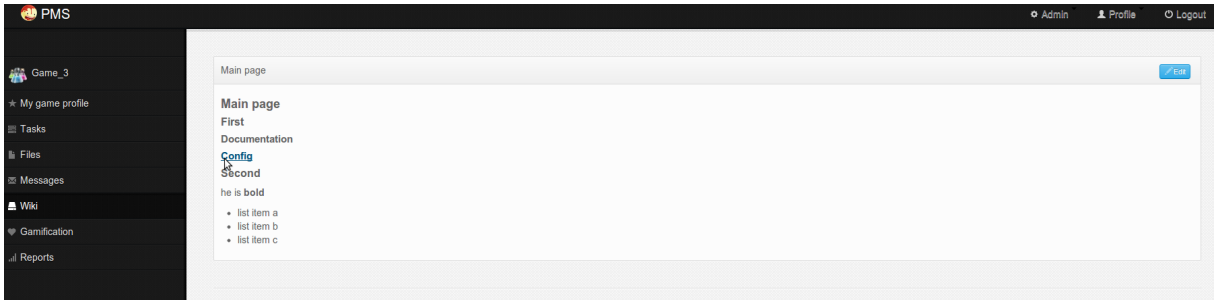
Projekt zawiera także system wiadomości w postaci forum. Rozmowy toczące się na projekcie są zgrupowane tematycznie (rysunek 29). Obok każdego wątku rozmowy wyświetlana jest ilość odpowiedzi, ilość wyświetleń oraz czas ostatniej wypowiedzi. Po przejściu do wątku wyświetli się lista wiadomości w postaci chat'u (rysunek 30) skąd możemy dodać nową wiadomość.



Rysunek 29: List aktualnych tematów rozmów
Źródło: Opracowanie własne

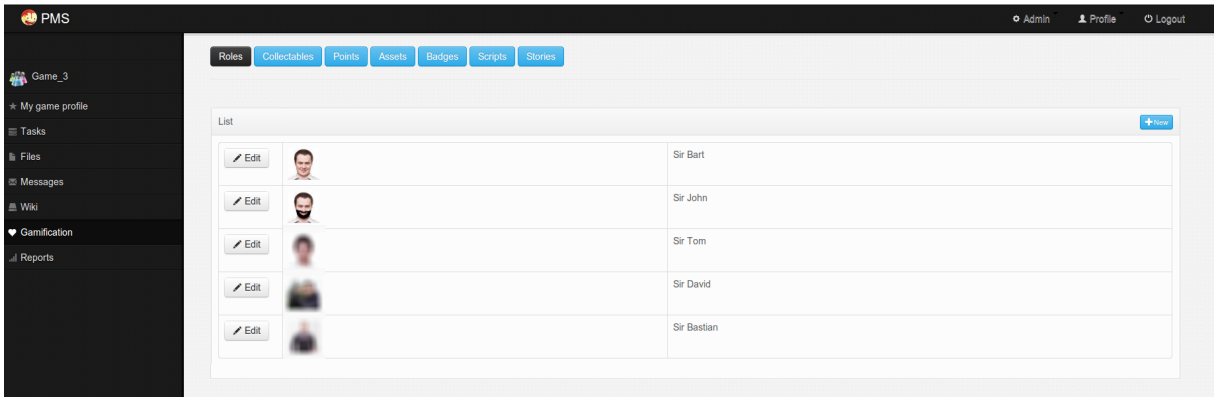
Rysunek 30: Wiadomości w wątku
Źródło: Opracowanie własne

Do przechowywania treści niezwiązanych z zadaniami służy moduł Wiki. Moduł ten pozwala na tworzenie stron oraz formatowanie ich za pomocą lekkiego języka znaczników (*markdown*). Nowe strony powinny być tworzone poprzez dodanie linków do nich na stronie-rodzicu. Po wyświetleniu jej należy kliknąć link podstrony i tam edytować jej treść.



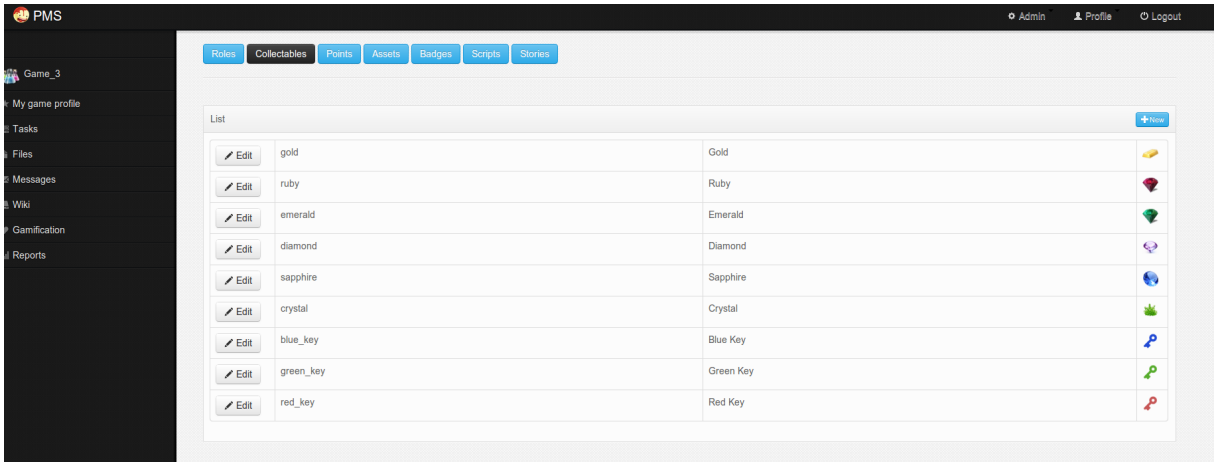
Rysunek 31: Sformatowana strona w module Wiki
Źródło: Opracowanie własne

Najważniejszym elementem systemu jest moduł grywalizacji skąd dokonywane są zmiany odnośnie rozgrywki. Pierwszym krokiem jest zdefiniowanie postaci dla graczy (rysunek 32), które będą reprezentowały użytkownika w danym projekcie. Następnie można określić listę kolekcji.



Rysunek 32: Role w grze
Źródło: Opracowanie własne

Dla pojedynczej kolekcji należy zdefiniować obrazek ją najlepiej przedstawiający. Na rysunku 33 przedstawiono przykładową listę kolekcji. W jej skład mogą wchodzić różne dobra materialne, kryształy, rzeczy, które mogą się przydać podczas rozgrywki tj. klucze – występuje tutaj pełna dowolność – zazwyczaj będzie to zależało od kontekstu gry.

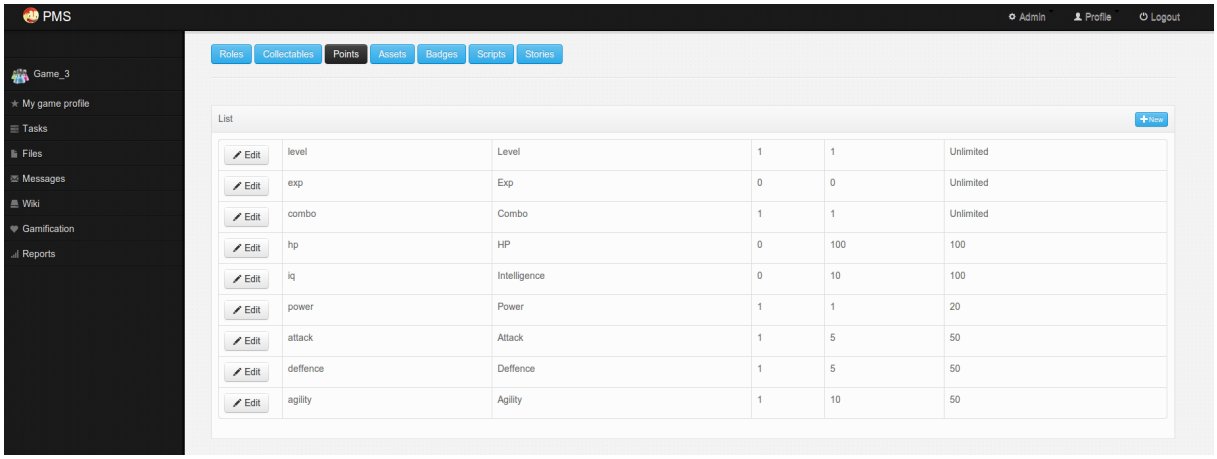


Rysunek 33: Lista kolekcji w grywalizacji
Źródło: Opracowanie własne

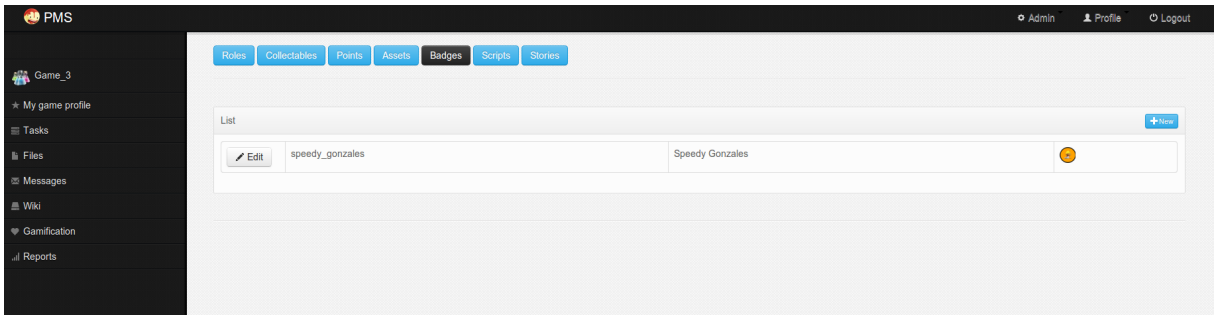
Nieodłącznym aspektem grywalizacji są punkty. Widok punktów przedstawia rysunek 34. Punkty dzielą się na punkty nielimitowane oraz zakresowe. Na liście punktów wyświetla się informacja jakie to są punkty, ich wartość początkową dla gracza oraz przedział, na jakim operują.

Kolejnym elementem grywalizacji są odznaczenia. Definiując je należy zawsze ustawić dla nich obrazek i nazwę. Sam powód przyznania danego odznaczenia użytkownikowi jest już określany przez logikę gry. Na rysunku 35 zaprezentowano listę odznaczeń.

Istotą dynamicznych elementów grywalizacji jest warstwa skryptowa. Widok dla skryptów gry przedstawia rysunek 36. Na widoku tym trzeba określić, dla którego sprintu będzie tworzona logika. Wynika to z faktu unikania monotonności rozgrywki. Dzięki temu gra nie będzie do końca

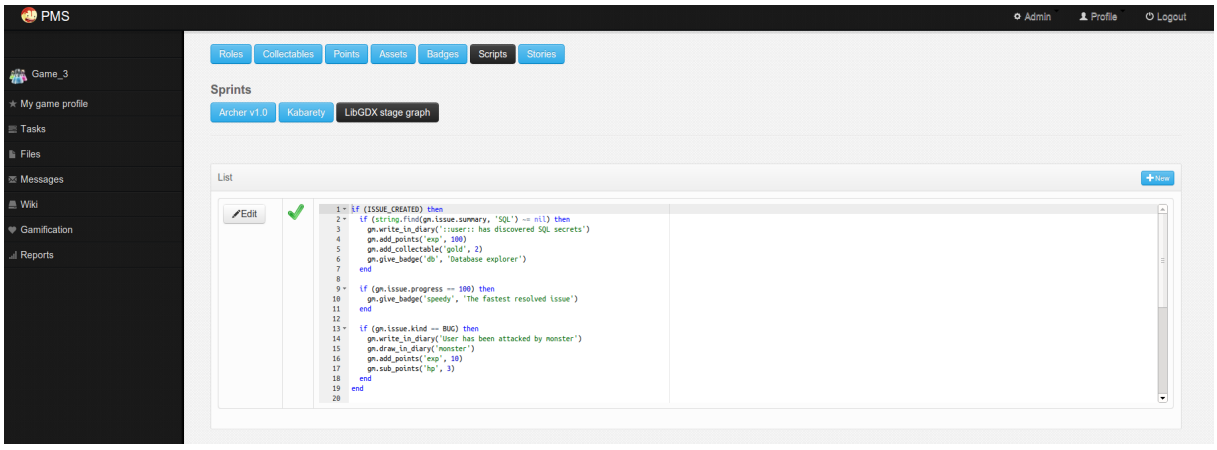


Rysunek 34: Lista punktów w grywalizacji
Źródło: Opracowanie własne



Rysunek 35: Lista odznaczeń
Źródło: Opracowanie własne

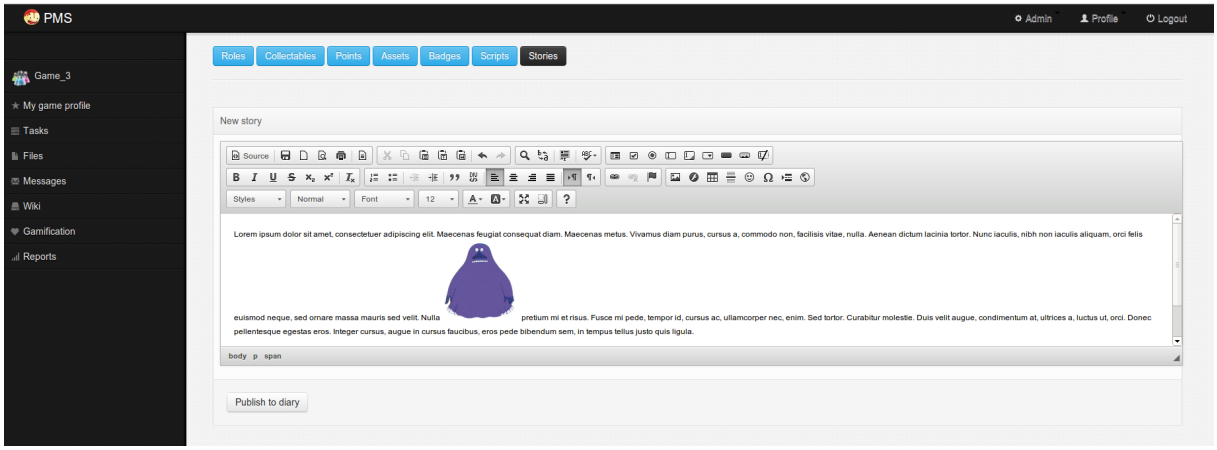
przewidywalna dla graczy i nie będą mogli jej łatwo oszukać.



Rysunek 36: Lista skryptów logiki gry
Źródło: Opracowanie własne

Dodatkową możliwością jest publikowanie historyjek do pamiętnika gry. Proces publikowania przedstawia rysunek 37. Historyjki są tworzone w edytorze WYSIWYG i po zatwierdzeniu automatycznie publikowane. W edytorze można posługiwać się obrazkami wgranymi do modułu

grywalizacji, aby zwiększyć ich atrakcyjność.



Rysunek 37: Publikowanie historyjek do gry
Źródło: Opracowanie własne

5.2 Aplikacja mobilna

Stworzona aplikacja mobilna jest prostą implementacją pozwalającą na śledzenie zadań w projekcie. Jest ona tylko odczytu danych.

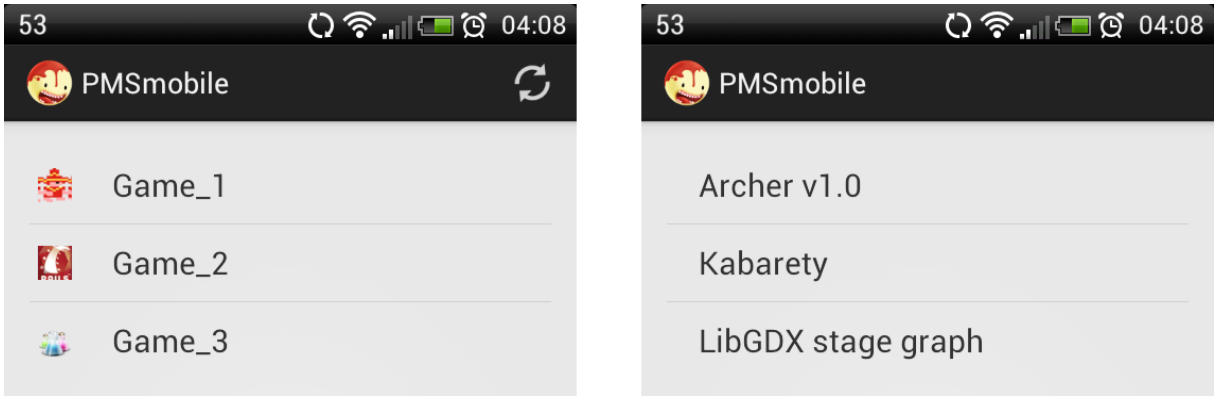
Aplikacja składa się z 4 widoków:

- lista projektów
- lista sprintów
- lista zadań
- raport z zadań

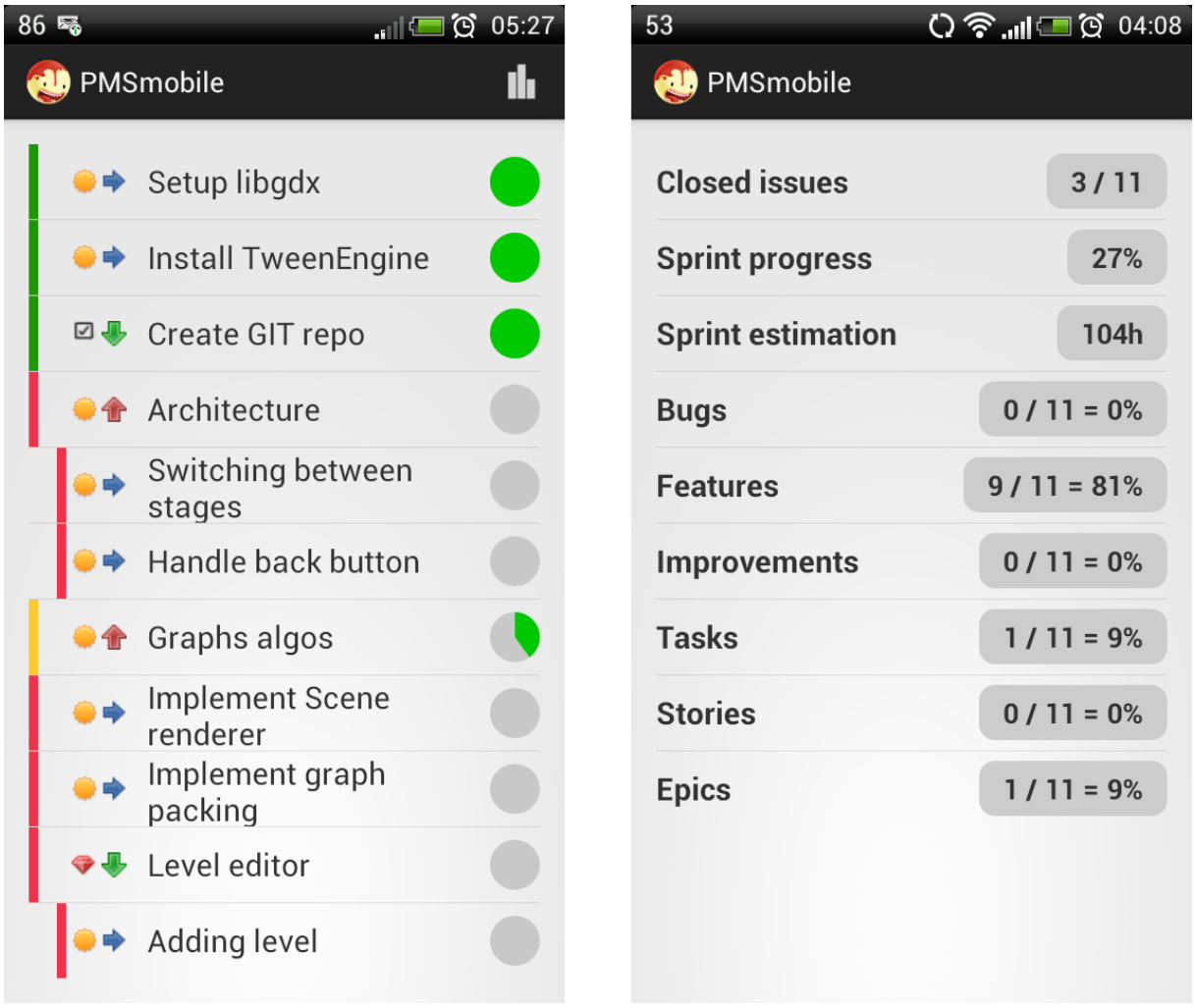
Po zalogowaniu się do aplikacji ściągane są dane o projektach, które są prezentowane na pierwszym ekranie (rysunek 38) – wyświetlana jest nazwa oraz logo. Synchronizacja danych jest manualna i odbywa się poprzez naciśnięcie przycisku w prawym górnym rogu na pasku akcji. Po wejściu w dany projekt pobierane są jego sprinty. Z listy sprintów (rysunek 38) można przejść do zadań, które też są ściągane. Ich lista znajduje się na rysunku 39. Ponadto na pasku akcji w widoku zadań znajduje się przycisk przełączający ze sprintu do raportu (rysunek 39). Raport zawiera następujące informacje:

- ilość zamkniętych zadań w stosunku do otwartych
- postęp sprintu
- łączna estymacja sprintu
- procentowy udział poszczególnych rodzajów zadań w sprint'cie

Wszystkie dane po ściągnięciu są trzymane w pamięci podręcznej dlatego aplikacja może działać w trybie offline.



Rysunek 38: Fragment widoku projektów oraz sprintów
Źródło: Opracowanie własne



Rysunek 39: Widoku zadań oraz raport
Źródło: Opracowanie własne

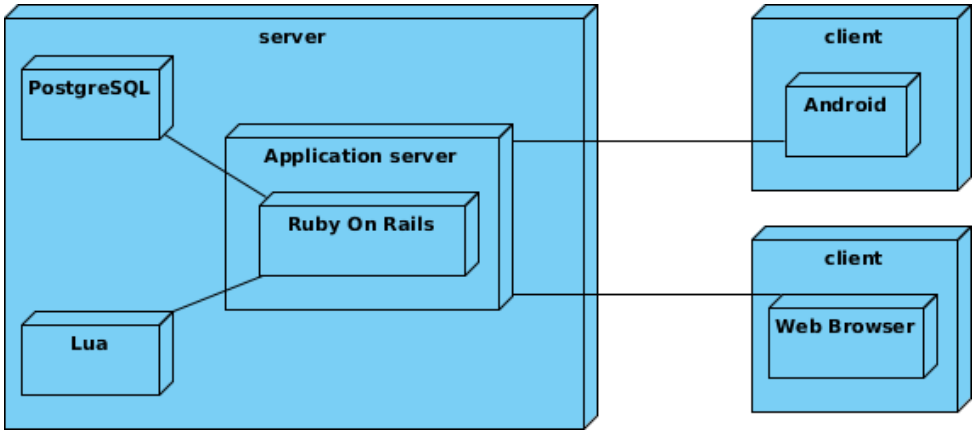
5.3 Opis implementacji

W niniejszym rodziale przedstawiono techniczne aspekty implementacji.

5.3.1 Architektura

Na rysunku 40 przedstawiono ogólny zarys architektury. Głównymi elementami proponowanego rozwiązania są:

- aplikacja Ruby On Rails
- interpreter Lua
- baza danych (PostgreSQL)



Rysunek 40: Ogólny diagram architektury
Źródło: Opracowanie własne

Prototyp został oparty o Ruby On Rails w wersji 3.2.13. Jak opisano w rozdziale 3.1.2 framework ten używa wzorca MVC , który pomaga w strukturyzacji aplikacji. Bazową klasą dla modeli jest *ActiveRecord::Base*, która opakowuje zarządzanie połączeniami z bazą danych oraz mapowanie danych na obiekty.

W systemie wykorzystano zewnętrzną bibliotekę, która ułatwia uruchamianie interpretera Lua. Do poprawnego działania skryptów wymagana jest Lua w wersji co najmniej 5.1.4. Wszystkie aktywne skrypty Lua na projektach są uruchamiane podczas tworzenia zadań lub ich aktualizacji. Błędy powstale podczas ich wykonania są zapisywane w bazie danych i mogą być później przeglądane w celu analizy niepowodzenia.

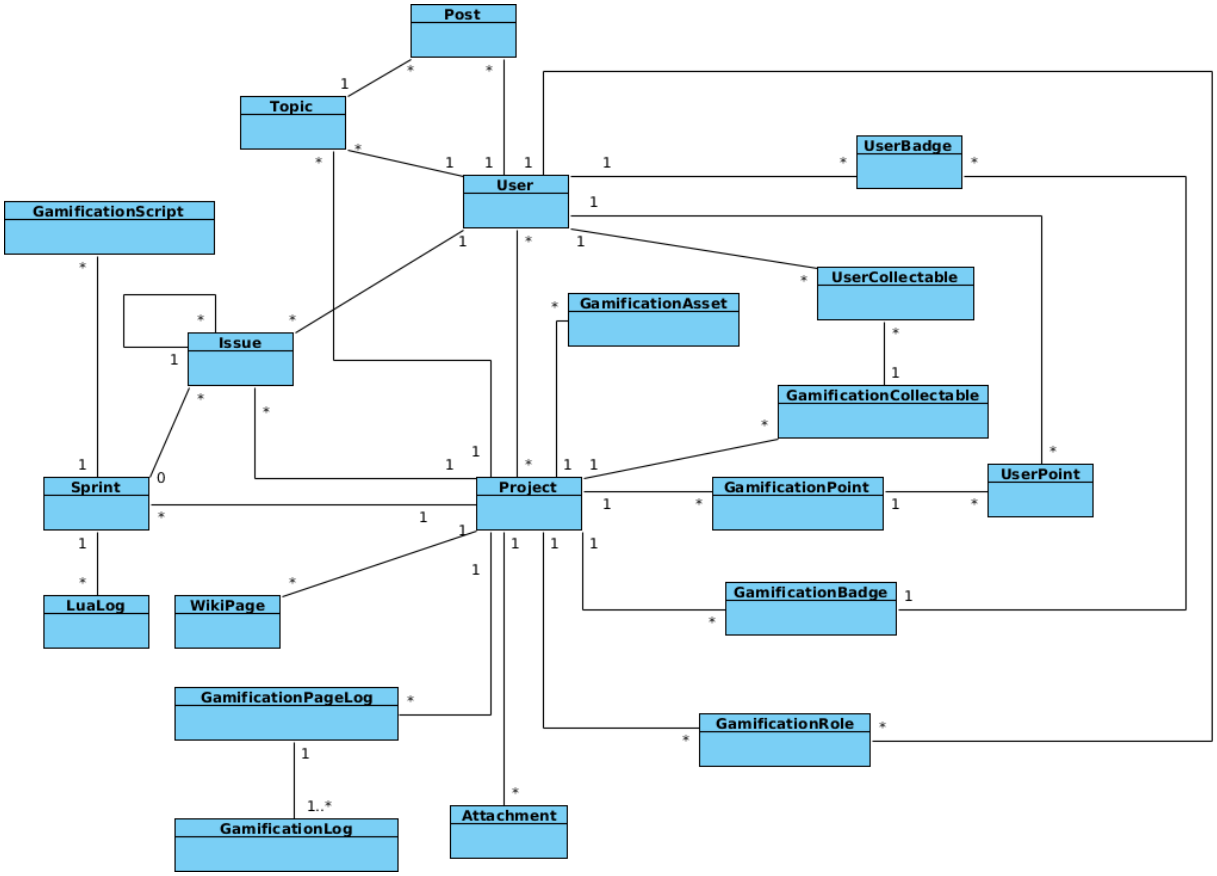
Z aplikacją mogą łączyć się klienci. Klient to urządzenie z sytemem Android z minimalną wersją systemu 4.0.3 lub przeglądarka interentowa. Zalecaną przeglądarką jest Google Chrome.

5.3.2 Model dziedziny

Model dziedziny zawiera odwzorowanie obiektów biznesowych na modele aplikacji. Na rysunku 41 zaprezentowano ten zbiór oraz zależności jego składowych.

Na diagramie przedstawiono następujące modele:

- Project – realizowany projekt
- Sprint – faza projektu



Rysunek 41: Model dziedziny
Źródło: Opracowanie własne

- User – użytkownik systemu
- Issue – zadanie zdefiniowane w projekcie
- Attachment – wgrany plik
- WikiPage – strona z treścią
- Topic – wątek forum
- Post – pojedyncza wypowiedź w wątku forum
- GamificationPageLog – strona z pamiętnika gry, agregująca wpisy
- GamificationLog – pojedynczy wpis pamiętnika
- GamificationScript – skrypty odpowiedzialne za logikę gry
- GamificationAsset – pliki graficzne do gry
- GamificationPoint – zdefiniowane punkty grywalizacji w projekcie
- GamificationBadge – zdefiniowane odznacznie dla projektu
- GamificationRole – zdefiniowane role dla użytkowników w projekcie
- GamificationCollectable – zdefiniowane kolekcje grywalizacji w projekcie

- UserBadge – przyznane odznaczenia dla użytkownika
- UserCollectable – aktualne wartości kolekcji dla użytkowników
- UserPoint – aktualne wartości punktów dla użytkowników
- LuaLog – zapis błędów skryptów

5.3.3 Wybrane fragmenty kodu źródłowego

W niniejszym podrozdziale przedstawiono sposób działania skryptów Lua.

Jak wspomniano wcześniej skrypty są wykonywane podczas tworzenia zadań oraz ich aktualizacji. Na listingu 1 oraz 2 zawarto metody kontrolera zapewniające te dwie operacje.

```
def create
  @issue = @project.issues.build(params[:issue])
  @issue.sprint = @sprint

  if @issue.save
    process_scripts(@project, @sprint, @issue, nil) if @sprint

    redirect_to redirect_page, :notice => "Issue created"
  else
    flash.now[:error] = "Can't create issue"
    render :action => :new
  end
end
```

Listing 1: Tworzenie zadania

```
def update
  @issue_old = @issue.dup

  if @issue.update_attributes(params[:issue])
    process_scripts(@project, @sprint, @issue, @issue_old) if
       @sprint
    redirect_to redirect_page, :notice => "Issue update"
  else
    flash.now[:error] = "Can't update issue"
    render :action => :edit
  end
end
```

Listing 2: Aktualizacja zadania

Akcja *create* kontrolera buduje obiekt zadania z parametrów, a następnie przypisuje mu sprint. Kolejno stara utworzyć się obiekt w bazie danych. Gdy to się nie uda to wyświetlany jest ponownie formularz zadania z zaznaczonymi błędami. W przypadku sukcesu przetwarzane są skrypty Lua, ale tylko pod warunkiem, że sprint istnieje – jeślie nie tzn. że zadanie jest tworzone w backlogu. Sprawdzenie tego warunku jest konieczne ponieważ kontroler jest współdzielony.

Akcja *update* na samym początku wykonuje duplikat obiektu zadania w pamięci, aby móc przekazać poprzednie wartości w celu późniejszej analizy przez skrypty. W przypadku sukcesu aktualizacji wykonywane są skrpyty jeśli nie jest to backlog.

Listing 3 przedstawia metodę przetwarzania skryptów. Na początku jest tworzona klasa ewaluatora z parametrami. Następnie znajdowane są wszystkie aktywne skrypty oraz są one uruchamiane. Po zakończeniu warstwy skryptowej metoda sprawdza czy wystąpiły jakieś błedy podczas ewaluacji. Jeśli tak to zapisuje je do bazy, jeśli nie to w jednej transakcji tworzy nowe wpisy w pamiętniku oraz zapisywane są zmiany elementów grywalizacji użytkownika.

```
def process_scripts(project, sprint, issue, issue_old)
  lua = Lua::Exec.new(current_user, project, sprint, issue,
     issue_old)

  scripts = sprint.gamification_scripts.where(:active => true).all
  lua.run(scripts)
  if lua.errors.any?
    LuaLog.log!(sprint, lua.errors.join("\n"))
  else
    Issue.transaction do
      page_log = @project.gamification_page_logs.create
      if page_log.persisted?
        lua.process(page_log)
      end

    end
  end

  lua
end
```

Listing 3: Przetwarzanie skryptów

Kolejnym istotnym fragmentem aplikacji jest kod ewaluatora skryptów. Listing 4 przedstawia ciało metody *run*. Jak można zauważyć ewaluowana jest cała tablica skryptów. Ważne jest, aby każdy skrypt miał własny stan i był uruchomiony w niezależnym od siebie środowisku – inaczej mogłoby to powodować kolizje zmiennych lub doprowadziłoby do niepożądanego zachowania. Każdy błąd wykonania jest dodawany do tablicy błędów, które później są zapisane do bazy. Przed ewaluacja skryptów tworzone jest API (*bindings*). Przykładową budowę API zawiera

listing 5. Podczas wiązania metod Lua z kodem Ruby ważne jest, aby operacje, które zmieniają coś w bazie danych nie były wykonywane od razu tylko opóźniane, a później wykonane w jednej transakcji – zapewniając atomowość operacji. Problem ten został rozwiązany poprzez użycie metody *post_delayed* znajdującej się na listingu 6. Metoda ta zbiera blok kodu i dodaje go do tablicy. Tablica ta później jest przetwarzana i każda operacja zostaje wykonana w takiej kolejności jakiej została opóźniona. Odpowiada za to metoda *process* we wspomnianym już listingu 3.

```
  def run(scripts)
    scripts.each do |script|
      begin
        lua = Rufus::Lua::State.new

        prepare_bindings(lua)

        lua.eval(script.code)
      rescue => e
        @errors << e.message
      ensure
        lua.close
      end
    end
  end
```

Listing 4: Uruchomienie skryptów

```
  lua.eval("gm = {}")
...
  lua.eval("SPRINT_STARTS_AT = #{@sprint.starts_at.to_i}")
  lua.eval("SPRINT_ENDS_AT = #{@sprint.ends_at.to_i}")
...
  lua.function("gm.write_in_diary") do |text|
    post_delayed do
      log = GamificationLog.log(@page_log, @user, @project, "text",
          text)
      log.save
    end
      nil
  end

  lua.function("gm.draw_in_diary") do |text|
    post_delayed do
```

```
      log = GamificationLog.log(@page_log, @user, @project, "img",
          text)
      log.save
    end

    nil
  end
...
```

Listing 5: Budowanie API

```
  def post_delayed
    @process_delayed << lambda { yield }
  end
```

Listing 6: Opóźnianie wykonania kodu

6 Przykład użycia

W niniejszym rozdziale zostanie przedstawione API grywalizacji oraz przykład wykorzystania logiki gier w zarządzaniu projektem.

6.1 Omówienie API

API składa się z pomocniczych zmiennych globalnych, obiektu zadania oraz metod pozwalających na manipulację elementami grywalizacji. Poniżej zostanie ono dokładniej opisane.

6.1.1 Zmienne globalne

Zmienne globalne to pomocnicze dane wspomagające pisanie skryptów. Wśród nich można wyróżnić:

- **ISSUE_CREATED** – stała typu *boolean* przechowująca czy zadanie zostało właśnie utworzone
- **ISSUE_UPDATED** – stała typu *boolean* przechowująca czy zadanie zostało właśnie zaktualizowane (wzajemnie wyklucza się z ISSUE_CREATED)
- **SPRINT_STARTS_AT** – przechowuje datę rozpoczęcia sprintu w formacie ilości sekund jakie upłynęły od 1970r
- **SPRINT_STARTS_AT** – przechowuje datę zakończenia sprintu w formacie ilości sekund jakie upłynęły od 1970r
- **NOW** – przechowuje aktualną datę w formacie ilości sekund jakie upłynęły od 1970r
- **ISSUES_OPENED** – ilość zadań otwartych w sprint'cie, do którego należy zadanie
- **ISSUES_CLOSED** – ilość zadań zamkniętych w sprint'cie, do którego należy zadanie
- **gm.user.id** – identyfikator liczbowy przechowujący aktualnie zalogowanego użytkownika
- **BUG** – stała reprezentująca typ zadania jako „błąd"
- **FEATURE** – stała reprezentująca typ zadania jako „funkcjonalność"
- **IMPROVEMENT** – stała reprezentująca typ zadania jako „usprawnienie"
- **TASK** – stała reprezentująca typ zadania jako „zadanie zwykłe"
- **STORY** – stała reprezentująca typ zadania jako „scenariusz"
- **EPIC** – stała reprezentująca typ zadania jako „super usprawnienie"
- **TODO** – stała reprezentująca stan zadania jako „do zrobienia"
- **INPROGRESS** – stała reprezentująca stan zadania jako „w trakcie realizacji"
- **DONE** – stała reprezentująca stan zadania jako „gotowe"
- **MINOR** – stała reprezentująca priorytet zadania jako „niski"
- **NORMAL** – stała reprezentująca priorytet zadania jako „średni"
- **MAJOR** – stała reprezentująca priorytet zadania jako „wysoki"

6.1.2 Obiekt zadania

API posiada dostęp do obiektu zadania, aby móc analizować co zmienia się w systemie. Dostęp do tego obiektu można uzyskać za pomocą instrukcji:

- **gm.issue** – obiekt zadania
- **gm.issue.old** – obiekt zadania przed modyfikacją w celu porównania zmian (jest pusty gdy zadanie jest tworzone, należy używać tylko w połączeniu z ISSUE_UPDATED)

Atrybuty zadania:

- **id** – identyfikator zadania
- **kind** – typ zadania, jeden z poniższych:
  - BUG
  - FEATURE
  - IMPROVEMENT
  - TASK
  - STORY
  - EPIC
- **status** – status zadania
  - TODO
  - INPROGRESS
  - DONE
- **progress** – postęp zadania (0-100)
- **priority** – priorytet zadania:
  - MINOR
  - NORMAL
  - MAJOR
- **estimate** – estymacja zadania (w sekundach)
- **parent_id** – identyfikator zadania-rodzica (w przypadku gdy brak rodzica to parent_id jest równy zeru)
- **user_id** – identyfikator użytkownika zadania
- **created_at** – czas utworzenia zadania (w sekundach od 1970r)
- **updated_at** – czas modyfikacji zadania (w sekundach od 1970r)
- **summary** – tytuł zadania
- **description** – treść całego zadania

6.1.3 Metody

Metody zawierają dostęp do kolekcji, punktów, odznaczeń oraz pamiętnika.

- **Kolekcje**
  - gm.add_collectable('UUID', value) – dodanie wartości (value) do kolekcji (UUID)
  - gm.sub_collectable('UUID', value) – odjęcie wartości (value) od kolekcji (UUID)
  - gm.set_collectable('UUID', value) – ustawienie wartości (value) dla kolekcji (UUID)
  - gm.get_collectable('UUID') – pobranie wartości kolekcji (UUID)
- **Punkty**
  - gm.add_points('UUID', value) – dodanie wartości (value) do punktów (UUID)
  - gm.sub_points('UUID', value) – odjęcie wartości (value) od punktów (UUID)
  - gm.set_points('UUID', value) – ustawienie wartości (value) dla punktów (UUID)
  - gm.get_points('UUID') – pobranie punktów (UUID)
- **Odznaczenia**
  - gm.give_badge('UUID', 'title') – nadanie odznaczenia (UUID) użytkownikowi, określając tytuł (title)
- **Pamiętnik**
  - gm.write_in_diary('hello ::uuid::') – dodanie tekstu do pamiętnika, każdy identyfikator (uuid) otoczony podwójnym dwukropkiem jest zamieniany na odpowiednią nazwę kolekcji, punktów, obrazków lub odznak
  - gm.draw_in_diary('uuid1 & uuid2') – wyświetlenie obrazków (uuid1, uuid2) w pamiętniku, obrazki oddzielane są znakiem „&”

6.2 Logika gier w kontekście zarządzania projektem

Przed rozpoczęciem projektu osoba odpowiedzialna za grywalizację na projektach musi zdefiniować grę dla programistów. Pierwszym krokiem jest nadanie wszystkim ról (postaci w grze). Następnie za pomocą historyjek publikuje fabułę gry do pamiętnika i nakreśla główne misje. Gracze (programiści) mogą zapoznać się z klimatem gry czytając pamiętnik.

Kolejnym krokiem jest zdefiniowanie dynamicznych elementów grywalizacji. Na potrzeby przykładowego projektu należy więc zdefiniować:

- **Punkty doświadczenia (exp)** – nieograniczone punkty jakie użytkownik może zebrać
- **Punkty zdrowia (hp)** – punkty od 0 do 100 reprezentujące zdrowie gracza, z domyślną wartością 100
- **Kolekcję złota (gold)** – złoto jakie użytkownik może zbierać podczas rozgrywki
- **Odznaka „Speedy Gonzales” (speedy)**

- **Odznaka „Mistrz baz danych" (db)**
- **Obrazek z przeciwnikiem (monster)**
- **Obrazek z martwym przeciwnikiem (dead_monster)**

Po określeniu elementów grywalizacji można przejść do projektowania mechaniki gry. Dla wybranego sprintu w projekcie trzeba dodać skrypty odpowiadające za logikę.

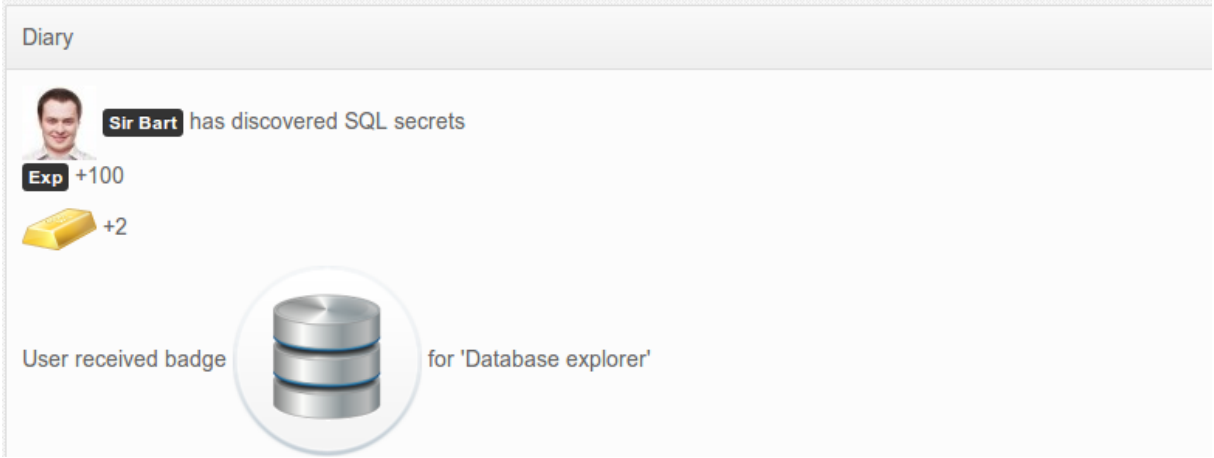
```
if (ISSUE_CREATED) then
  if (string.find(gm.issue.summary, 'SQL') ~= nil) then
    gm.write_in_diary('::user:: has discovered SQL secrets')
    gm.add_points('exp', 100)
    gm.add_collectable('gold', 2)
    gm.give_badge('db', 'Database explorer')
  end

  if (gm.issue.progress == 100) then
    gm.give_badge('speedy', 'The fastest resolved issue')
  end

  if (gm.issue.kind == BUG) then
    gm.write_in_diary('User has been attacked by monster')
    gm.draw_in_diary('monster')
    gm.add_points('exp', 10)
    gm.sub_points('hp', 3)
  end
end

if (ISSUE_UPDATED) then
  if (
    (gm.issue.kind == BUG) and
    (gm.issue.status == DONE) and
    (gm.issue.old.status ~= DONE)
    )
  then
    gm.write_in_diary('User killed monster')
    gm.draw_in_diary('dead_monster')
    gm.add_points('exp', 10)
    gm.add_points('hp', 3)
  end

  if (gm.issue.kind == EPIC and gm.issue.priority == MAJOR) then
    gm.write_in_diary('::user:: has found some gold')
```

```
    gm.add_collectable('gold', 1000)
  end
end
```

Listing 7: Przykładowa mechanika gry

Na listingu 7 przedstawiono przykładową mechanikę. Podzielona jest ona na dwie sekcje. Pierwsza z nich wykonuje się gdy zadanie jest tworzone w systemie (ISSUE_CREATED). Druga natomiast gdy jakieś zadanie jest aktualizowane (ISSUE_UPDATED). Należy ją interpretować następująco:

- **ISSUE_CREATED**
  - **if (string.find(gm.issue.summary, 'SQL') ≃ nil)** – w tym przypadku gra sprawdza czy tworzone zadanie w treści ogólnej zawiera słowo „SQL", jeśli tak to w pamiętniku zostanie wypisana informacja, że gracz odkrył tajniki SQL, dostanie za to 100 punktów doświadczenia, 2 sztuki złota oraz odznaczenie „Eksploratora baz danych" (rysunek 42)



Rysunek 42: Zachowanie pamiętnika, gdy użytkownik utworzy zadanie ze słowem SQL
Źródło: Opracowanie własne

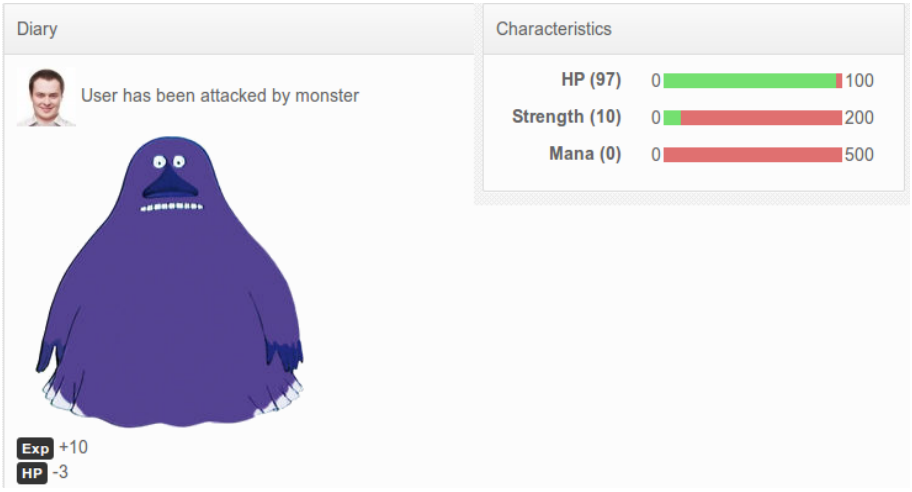
  - **if (gm.issue.progress == 100)** – gdy podczas tworzenia zadania okaże się, że gracz od razu ustawi postęp zadania na 100% to dostanie on za to odznaczenie „Najszybciej rozwiązane zadanie".
  - **if (gm.issue.kind == BUG)** – jeśli zadanie okaże się błędem to pamiętnik wypisze, że gracz został zaatakowany przez potwora, który także zostanie narysowany. Gracz dostanie 10 punktów doświadczenia, ale straci 3 punkty zdrowia (rysunek 43).

- **ISSUE_UPDATED**
  - **if ((gm.issue.kind == BUG) and (gm.issue.status == DONE) and (gm.issue.old.status ≃ DONE))** – jeśli zadanie podczas aktualizacji jest błędem,

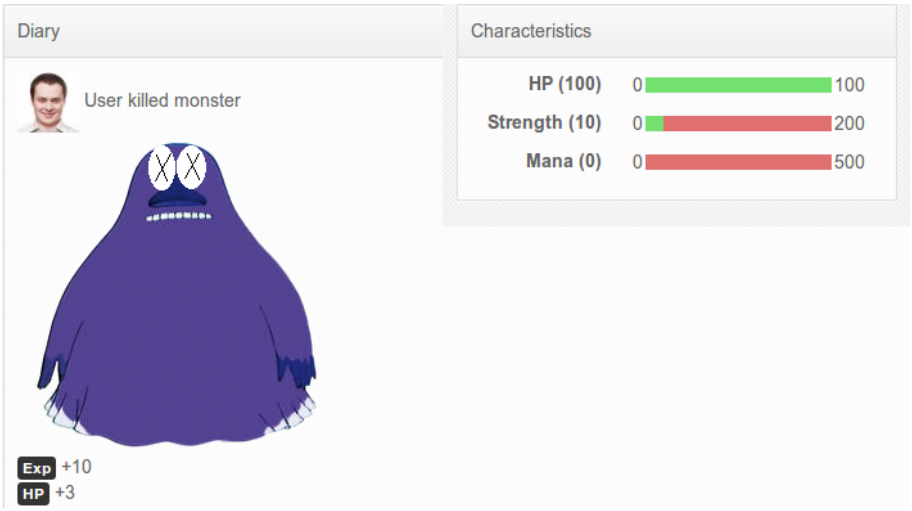
jego status jest ustawiony jako „Gotowe”, a poprzednia wersja zadania miała status inny niż „Gotowe” (warunek zakończenia zadania) pamiętnik wypisze, że gracz pokonał potwora i narysuje jego uśmiercony odpowiednik. Gracz zdobędzie 10 punktów doświadczenia, a jego zdrowie podniesie się o 3 jednostki (rysunek 44).

- **if (gm.issue.kind == EPIC and gm.issue.priority == MAJOR)** – za każdym razem gdy modyfikowane jest zadanie, które jest oznaczone wysokim priorytetem i jest „super zadaniem” to gracz zyska 1000 sztabek złota.

Po zadeklarowaniu skryptów gry dla wybranego sprint'u. Można dodać kolejną historyjkę do pamiętnika określającą dokładniej dany sprint by rozbudowywać fabułę gry i wzmacniać zainteresowanie graczy.



Rysunek 43: Zachowanie pamiętnika, gdy użytkownik utworzy zadanie typu „błąd”
Źródło: Opracowanie własne



Rysunek 44: Zachowanie pamiętnika gdy użytkownik zakończy rozwiązywać zadanie typu „błąd”
Źródło: Opracowanie własne

7 Podsumowanie

W podsumowaniu zawarto wykryte wady i zalety zastosowanego rozwiązania oraz plany rozwojowe na przyszłość.

7.1 Zalety i wady rozwiązania

Po zakończeniu pracy i przetestowaniu koncepcji autor może stwierdzić jakie są zalety przyjętego rozwiązania oraz jakie wykryto wady.

Największą zaletą okazała się elastyczność elementów grywalizacji. Dynamiczne ich tworzenie oraz skryptowanie logiki pozwala na skonstruowanie niemalże dowolnej mechaniki gry. Dodatkowe wzbogacenie elementów grywalizacji o historyjki poprawia jakość rozgrywki, a przyjęte reguły są adaptowane przyjaźniej.

Niestety zapewnienie takiego komfortu na projekcie zajmuje dużo czasu. Problemem staje się ciągła praca opiekuna projektu lub osoby odpowiedzialnej nad doskonaleniem gry i jej poprawnym balansem. Innym problemem znów jest znajomość języka skryptowego Lua – nie każda osoba zarządzająca projektem potrafi programować.

Choć sama rozgrywka przez swą nieprzewidywalność oraz dynamiczność potrafi dużo zdziałać to brakuje większej interakcji graczy między sobą. Ten problem można rozwiązać rozszerzając funkcjonalności systemu.

Ponadto po testach manualnych przeprowadzonych przez kilku użytkowników zewnętrznych wykryto błędy użyteczności systemu. Część z nich udało się wyeliminować jednak prototyp wciąż wymaga szlifowania.

7.2 Plan rozwoju

Przedstawione problemy skłoniły autora do opracowania planu rozwoju prototypu.
Rzeczy wymagające usprawnienia zawarto w poniższej liście:

- interakcja między graczami – obecnie stoi na niskim poziomie, ale ten problem można wyeliminować poprzez dodanie systemu decyzyjnego oraz wymianę kolekcji z innymi graczami, aby poszerzać interakcje. Skrypt gry mógłby zadać użytkownikowi pytanie z kilkoma odpowiedziami podczas jakiegoś wydarzenia. Wtedy w profilu gracza pojawiałoby się pytanie i w zależności od wyboru, gracz mógłby sam zadbać o swój los.
- brak izolacji od zarządzania projektem – obecne rozwiązanie wygląda jak zwykła aplikacja webowa do zarządzania projektem. Aby zwiększyć zaangażowanie graczy do częstszego jej używania powinna ona przypominać wyglądem interfejs gry komputerowej. Wszystkie informacje oraz nazwy tj. „Zadania”, „Pliki” powinny być konfigurowalne i prezentowane graczowi jako jej element np. „Zadania” to misje, „Pliki” to ładunek – lepiej będą współgrały z interfejsem.
- większa ilość danych i zdarzeń – system umożliwia jedynie dostęp do kilku danych oraz odpowiada na dwa zdarzenia – stworzenie zadania i jego aktualizację. Powinien

on odpowiadać na więcej zdarzeń np. użytkownik dodał wiadomość lub wgrał plik, aby premiować inne dobre praktyki. Ponadto w skryptach powinno być jeszcze więcej danych, aby robić dokładniejsze analizy i ciekawsze mechaniki.

- jeszcze większy dynamizm – poziom dynamiczności jest wystarczający na chwilę obecną, ale można byłoby usprawnić grę poprzez definiowanie struktur danych dla obiektów. Przykładowo obiekt potwora z własnym paskiem energii i siłą ataku umożliwiłby symulację walk lub nawet wprowadzenie sztucznej inteligencji.
- limitowane obiekty – w systemie należy rozważyć elementy kolekcji lub odznaczenia, które można zdobyć tylko jeden raz lub kilka – wtedy satysfakcja i chęć z ich posiadania wzrastają.

7.3 Zakończenie

Zarządzanie projektem to szeroka gałąź w informatyce. Stoi za nią wiele wyzwań, z którymi trzeba sobie jakoś radzić, a koordynowanie zespołów programistycznych nie należy do łatwych czynności. W niniejszej pracy autor przedstawił różne narzędzia wspomagające ten proces i zaproponował koncepcję, która zwiększy produktywność pracowników przy użyciu technik grywaliazacji.

Bibliografia

[1] Christopher Cunningham Gabe Zichermann. *Gamification by Design. Implementing Game Mechanics in Web and Mobile Apps.* O'Reilly Media, 2011.

[2] Jesse Schell. *The Art of Game Design: A book of lenses.* 2008.

[3] `http://www.mud.co.uk/richard/hcds.htm` (15.06.2013).

[4] `http://www.gamerdna.com/quizzes/bartle-test-of-gamer-psychology` (15.06.2013).

[5] `http://ruby-doc.org/` (15.06.2013).

[6] Hal Fulton. *Ruby. Tao programowania w 400 przykładach.* 2007.

[7] `http://rubyonrails.org/` (15.06.2013).

[8] `http://www.postgresql.org/` (15.06.2013).

[9] Craig Cook David Schultz. *HTML, XHTML i CSS. Nowoczesne tworzenie stron WWW.* Wydawnictwo Helion, 2008.

[10] Roberto Ierusalimschy. *Programming in Lua.* 2006.

[11] `http://sass-lang.com/` (15.06.2013).

[12] Stoyan Stefanov. *JavaScript Programowanie Obiektowe.* Wydawnictwo Helion, 2010.

[13] `http://jquery.com/` (15.06.2013).

[14] `http://twitter.github.io/bootstrap/` (15.06.2013).

[15] `http://ace.ajax.org/` (15.06.2013).

[16] `http://ckeditor.com/` (15.06.2013).

[17] Stoyan Stefanov. *Java Efektywne programowanie, Wydanie II.* Wydawnictwo Helion, 2009.

[18] `http://developer.android.com/sdk/index.html` (15.06.2013).

[19] `https://code.google.com/p/android-query/` (15.06.2013).

[20] `https://code.google.com/p/google-gson/` (15.06.2013).

[21] `http://www.sublimetext.com/3` (15.06.2013).

[22] `http://www.eclipse.org/` (15.06.2013).

[23] Włodzimierz Gajda. *Git. Rozproszony system kontroli wersji.* Wydawnictwo Helion, 2013.

[24] Paweł Tkaczyk. *GRYWALIZACJA. Jak zastosować reguły gier w działaniach marketingowych.* 2012.

[25] Jeannie Novak Marianne Krawczyk. *Game Development Essentials: Game Story & Character Development*. 2006.

[26] `http://stackoverflow.com/` (15.06.2013).

Spis rysunków