Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki

Praca dyplomowa inżynierska

IQUEST – SYSTEM ROZSZERZONYCH ANKIET STUDENCKICH

Krzysztof Marian Borowiak, 94269
Maciej Trojan, 94378
Krzysztof Urbaniak, 94381
Łukasz Wieczorek, 94385

Promotor
dr inż. Bartosz Walter

Poznań, 2013 r.

Tutaj przychodzi karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

Rozdział 1

Wprowadzenie

1.1 Opis problemu i koncepcja jego rozwiązania

1.1.1 Problem

W kontekście każdej organizacji, zrzeszającej większą grupę ludzi, takiej jak zakład pracy, czy jednostka szkolnictwa wyższego, jaką jest Politechnika Poznańska, niezbędne jest systematyczne badanie efektywności programów nauczania, sposobu kształcenia oraz celowości inwestycji. Przykładowo, brak wiedzy o potrzebach studentów i wykładowców dotyczących wyposażenia laboratoriów, sal wykładowych, programów komputerowych, może prowadzić do zbędnych zakupów, a tym samym do ponoszenia niepotrzebnych kosztów. Najprostszym i najbardziej popularnym narzędziem prowadzenia badań jest ankieta[1]. Pozwala ona na nieinwazyjne, anonimowe i masowe pozyskiwanie opinii respondentów na zadany temat, a dzięki zastosowaniu odpowiednich metod statystycznych, także na ocenę zdania całej populacji.

Biorąc pod uwagę ostatnie lata (2008-2012)[2], w zgodzie ze statutem[3], Politechnika Poznańska wykorzystywała szerokie spektrum narzędzi do pozyskiwania wiedzy o efektach podejmowanych działań. Jednak z uwagi na brak odpowiedniego systemu zapewniającego wymierność wyników (np. uniemożliwienie wielokrotnego udziału pojedynczej osoby w ankiecie, zablokowanie możliwości wypełnienia ankiety przez studenta innego wydziału/kierunku, itp.), uzyskane informacje mogły przyczyniać się do błędnej interpretacji sytuacji. Dla zobrazowania dotychczasowego systemu badania, posłuży przykład ankietowania studentów Wydziału Informatyki[4]. Student był poddawany badaniom ankietowym w formie elektronicznej na poziomach uczelni i wydziału, formie papierowej przeprowadzanej przez Samorząd Studentów na potrzeby uczelni, oraz dodatkowym ankietom w ramach niektórych zajęć dydaktycznych.

Wyniki ankiet realizowanych w ramach wyżej wymienionych systemów prowadzenia badań nie były ze sobą porównywane, co sprawiało, że nie mogły dać pełnego obrazu panującej sytuacji. Samo wdrożenie badań nie umożliwiało wyciągnięcia konstruktywnych wniosków. Brak odpowiednich mechanizmów analizy ich wyników, generował zagrożenie wystąpienia spadku jakości usług świadczonych przez uczelnię, a co za tym idzie, także jej prestiżu.

[1] [1][13]
[2] Mowa o okresie, który jest znany autorom niniejszej pracy dyplomowej.
[3] Zapis dotyczący dokonywania oceny nauczyciela akademickiego z uwzględnieniem oceny studentów[4].
[4] Do 2010 roku – Wydziale Informatyki i Zarządzania.

1.1.2 Proponowane rozwiązanie

Punktem wyjścia jest stworzenie jednego, globalnego, prostego w obsłudze systemu o nazwie *iQuest*, służącego do prowadzenia badań ankietowych dla różnych grup docelowych, obejmujących zarówno aktualnych studentów oraz absolwentów Politechniki Poznańskiej. Będzie on współpracować z uczelnianym systemem raportowania, umożliwiającym wygodne pozyskiwanie informacji na zadany temat.

Dla zapewnienia bezproblemowego wdrożenia systemu oraz jego dostępności, zostanie on wykonany za pomocą technologii internetowych. Umożliwi to jego obsługę z użyciem dowolnej popularnej przeglądarki internetowej dostępnej na rynku.

Ważnym elementem proponowanego rozwiązania jest objęcie badaniami również absolwentów uczelni. W zamian za wzięcie udziału w badaniach, uzyskają oni dostęp do unikalnych materiałów dydaktyczno-naukowych.

Przeprowadzane badania będą anonimowe. Jednakże, w celu zapewnienia prawidłowości i miarodajności wyników, niezbędne jest wprowadzenie mechanizmów uwierzytelniania użytkowników systemu. Ich autoryzacja będzie obejmować jedynie kwestię dostępu do badań i materiałów – dla ankietera, wyniki będą pozbawione danych identyfikujących poszczególnych respondentów.

1.2 Ograniczenia i zagrożenia dla projektu

Analizie podano jedynie ograniczenia i zagrożenia związane ze stroną implementacyjną projektu, z uwzględnieniem następujących definicji:

Ograniczenie – granica, której przekroczenie jest niemożliwe lub zakazane. Obejmuje np. ograniczenia technologiczne oraz narzucone z góry przez osoby o kompetencjach decyzyjnych w kwestiach związanych z projektem.

Zagrożenie – sytuacja lub fakt mogący negatywnie wpłynąć na realizację projektu.

1.2.1 Ograniczenia

System *iQuest* realizowany jest dla Wydziału Informatyki. W związku z tym musi on spełniać szerokie spektrum wymagań ze strony takich organów, jak Dział Rozwoju Oprogramowania (DRO), Biuro Analiz i Rozwoju Usług (BAiRU), czy Dział Obsługi i Eksploatacji (DOiE). Jednym z warunków jest konieczność stosowania rozwiązań „otwartych” (ang. *open*) oraz „wolnych” (ang. *free*), co znacząco zawęża możliwe do zastosowania technologie i zasoby.

Za poboczne ograniczenie, ale mające istotny wpływ na przebieg prac, należy uznać z góry narzucony harmonogram, oraz restrykcje co do wielkości zespołu programistów.

1.2.2 Zagrożenia

Mimo braku wcześniejszego doświadczenia zespołu przy realizacji tego rodzaju projektów, zespół dynamicznie przystosowywał się do sytuacji, wykazując umiejętność szybkiego uczenia się. W dalszym ciągu zagrożeniem pozostaje fakt, że wykonawcami projektu są studenci uczelni technicznej, a nie wykwalifikowani specjaliści w dziedzinie technologii internetowych, posiadający

wieloletnią praktykę w realizacji podobnych zadań.

Poważną przeszkodą jest niekompletność dokumentacji niektórych systemów i technologii niezbędnych w projekcie[5] oraz niespodziewane zmiany w specyfikacji projektu i wymaganiach klienta, występujące w trakcie jego realizacji.

1.2.3 Wpływ ograniczeń i zagrożeń na projekt

Narzucona decyzja o zastosowaniu konkretnych typów technologii (rozwiązania „wolne i otwarte"), dokumentowanych nie przez wyspecjalizowanych fachowców, lecz tzw. „społeczność" (ang. *community*), wymusiła konieczność oparcia wielu działań na metodzie „prób i błędów", co znacząco wpłynęło na czas realizacji zadań poszczególnych członków zespołu projektowego.

Wspomniana w poprzedniej sekcji (1.2.2. – Zagrożenia) kwestia zmieniających się wymagań i specyfikacji sprawiła jednak najwięcej problemów. Niejednokrotnie zdarzały się sytuacje, kiedy prace projektowe były już w zaawansowanym stadium, a nowe wytyczne powodowały konieczność rozpoczynania pracy od podstaw. Było to główną przyczyną powstawania opóźnień w realizacji projektu.

1.3 Cele projektu

Celem projektu *iQuest* jest zbudowanie jasnego, prostego, przystępnego systemu umożliwiającego prowadzenie badań wśród studentów i absolwentów uczelni. System ten powinien:

- zapewnić spełnienie przez Uczelnię zapisów Ustawy „Prawo o Szkolnictwie Wyższym" dotyczących monitorowania rozwoju absolwentów Uczelni[3]
- ujednolicić uczelniany system pozyskiwania informacji
- oferować dużą elastyczność:
  - przy definiowaniu różnorodnych ankiet
  - przy tworzeniu i hierarchizacji grup respondentów
  - przy zachęcaniu do uczestnictwa w niej przez potencjalnych Respondentów
- integrować się z uczelnianym systemem raportowania
- odciążyć pracowników uczelni oraz Samorząd Studentów z obowiązków związanych z przeprowadzaniem konwencjonalnych („papierowych") ankiet

1.4 Osoby realizujące projekt

Projekt realizowany jest w ramach Software Development Studio (SDS), służącego realizacji projektów informatycznych dla Wydziału Informatyki Politechniki Poznańskiej. W rolę zespołu zarządzającego wcielają się studenci studiów magisterskich (specjalizacja *Software Engineering*). Zespół programistów tworzą studenci ostatniego semestru studiów inżynierskich, dla których udział w SDS jest ściśle związany z pracą dyplomową inżynierską. Wkład poszczególnych osób w projekt znajduje się w dodatku A.1.

Poniżej przedstawiono wykaz kompetencji poszczególnych zespołów:

[5] Rozwinięcie tej kwestii znajduje się w rozdziałach 6. oraz 8.

Zespół zarządzający – dwóch studentów, pełniących w głównej mierze role *kierownika projektu* oraz *architekta*, ale także wykonujący obowiązki *analityka*. Kompetencje:

- Kontakt bezpośredni z klientem oraz jego przedstawicielami.
- Decydowanie o kierunku rozwoju projektu (w tym: wybór technologii, rozwiązań, itp.).
- Utworzenie dokumentacji architektury projektu.
- Tworzenie i przydzielanie zadań do zespołu programistów.

Zespół programistów – czterech studentów, pełniących role *programistów*. Kompetencje:

- Przyjmowanie i realizacja zadań przydzielonych zespołowi.
- Zgłaszanie problemów z implementacją założeń architektonicznych.
- Opiniowanie decyzji w procesie ich podejmowania.

1.5 Struktura pracy

Praca została podzielona umownie na trzy wzajemnie uzupełniające się części. Część pierwsza, obejmująca rozdziały 1., 8. i 9. oraz sekcję 6.2. rozdziału 6., odnosi się do całości projektu z punktu widzenia realizujących go osób. Opisane są w niej założenia, napotkane problemy oraz wyciągnięte wnioski. Część druga – rozdziały od 2. do 5. – dotyczy charakterystyki projektu oraz jego architektury. Oparto ją na danych pozyskanych od zespołu zarządzającego projektem[6], dostępnych dla zespołu programistów w ramach platformy *Redmine*. Część trzecia, zawierająca pozostałe rozdziały, tj. 6. i 7. (z wyłączeniem sekcji 6.2.), przedstawia projekt od strony implementacji. Opisuje szczegółowo budowę systemu iQuest.

[6] [12]

Rozdział 2

Opis procesów biznesowych

2.1 Wstęp

System *iQuest*, będący przedmiotem niniejszej Pracy Dyplomowej, jest nie tylko projektem edukacyjnym, ale również pełnoprawnym zadaniem biznesowym. Wykonany dla Wydziału Informatyki Politechniki Poznańskiej, traktowany jest dokładnie tak samo, jak w pełni profesjonalne zlecenia, z którymi jego uczestnikom przyjdzie się zmierzyć w przyszłości. Z tego względu, konieczna jest jego analiza w kontekście powiązanych procesów biznesowych.

2.2 Aktorzy

W systemie zdefiniowani są następujący aktorzy:

- System – opisywany system. *iQuest*.
- Administrator – zarządza sprawami technicznymi, związanymi platformą *Moodle*[1]. Funkcję mogą pełnić osoby mające podstawową wiedzę informatyczną, znający mechanizmy *Moodle'a*.
- Administrator Bazy Danych – zarządza sprawami technicznymi, związanymi z prawami do grup docelowych, ich tworzeniem i utrzymaniem. Funkcję mogą pełnić Pracownicy Uczelni/Dziekanatu oraz Administratorzy Systemów.
- Ankieter – tworzy ankiety, wskazuje grupy docelowe i rozsyła ankiety. Może też przeglądać raporty. Funkcję mogą pełnić: Prowadzący zajęcia, Pracownik Dziekanatu.
- Respondent – odpowiada na otrzymane ankiety. Funkcję mogą pełnić: Absolwenci, Studenci.

2.3 Obiekty biznesowe

W ramach systemu *iQuest*, zdefiniowanych jest sześć obiektów biznesowych. Mowa o Ankiecie, Badaniu, Grupie Docelowej, Katalogu Ankiet, Pytaniu i Raporcie. Poniższe opisy tych obiektów podane są w kolejności alfabetycznej.

2.3.1 Ankieta

Jest tworzona przez Ankieterów i wysyłana do Respondentów. Raz utworzona Ankieta zostaje zapisana w Katalogu Ankiet. Ankietę charakteryzują następujące atrybuty:

[1] ang. *Modular Object-Oriented Dynamic Learning Environment*

- Nazwa Ankiety,
- Wstęp,
- Podsumowanie,
- Przypisane Pytania.

2.3.2 Badanie

Jest to Ankieta wraz z wybranymi: grupą docelową i czasem trwania. Badanie determinują następujące atrybuty:

- Nazwa Badania,
- Data rozpoczęcia,
- Data zakończenia,
- Okresowość,
- Grupa docelowa,
- Przypisana Ankieta.

2.3.3 Grupa Docelowa

Grupa studentów lub absolwentów, do których skierowana jest ankieta. Atrybuty:

- Studenci/Absolwenci

2.3.4 Katalog Ankiet

Katalog Ankiet zawiera zbiór wszystkich Ankiet dostępnych dla danego Ankietera *iQuest*. Ankiety mogą być współdzielone z poziomu Katalogu Ankiet, duplikowane, oglądane, edytowane i/lub usuwane, w zależności od aktualnego statusu. Dla przykładu, nowo-utworzoną Ankietę bez odpowiedzi można bez problemu usunąć lub edytować, podczas gdy taka, na którą udzielono już odpowiedzi, dostępna jest w trybie *tylko do odczytu*.

2.3.5 Pytanie

Pytanie jest elementarną jednostką Ankiety. Może składać się jedynie z treści (w przypadku pytań otwartych), lub treści i dostępnych odpowiedzi (dla Pytań zamkniętych). Pytanie w ogólności charakteryzują:

- Treść Pytania,
- Rodzaj Pytania,
- Dostępne odpowiedzi (dla Pytań zamkniętych).

2.3.6 Raport

Raport jest zbiorem odpowiedzi z jednego lub z kilku badań. Może zawierać wykresy i zestawienia. Funkcjonalność raportów pozwala na selektywne uzyskiwanie informacji na zadany temat.

2.4 Biznesowe przypadki użycia

Poniżej przedstawione zostały biznesowe przypadki użycia. Obejmują one dwa główne zagadnienia: zbieranie informacji oraz zarządzanie Grupami Docelowymi.

2.4.1 BC01: Zbieranie informacji o Absolwentach

Przypadek użycia: BC01: Zbieranie informacji o Absolwentach
Aktorzy: Ankieter, Respondent
Pre: Ankieter chce ankietować Absolwentów
Post: Ankieta, Raport
Scenariusz Główny
1. Ankieter tworzy Ankietę (UC01) 2. Ankieter wybiera Absolwentów, do których chce rozesłać Ankietę (UC03) 3. Ankieter uruchamia Ankietę (UC04) 4. System powiadamia Respondentów o Ankiecie 5. Respondent wypełnia Ankietę (UC05) 6. Ankieter sprawdza podsumowanie Ankiety (UC06)

2.4.2 BC02: Zbieranie informacji o Studentach

Przypadek użycia: BC02: Zbieranie informacji o Studentach
Aktorzy: Ankieter, Respondent
Pre: Ankieter chce ankietować Studentów
Post: Ankieta, Raport
Scenariusz Główny
1. Ankieter tworzy Ankietę (UC01) 2. Ankieter wybiera Studentów, do których chce rozesłać Ankietę (UC03) 3. Ankieter uruchamia Ankietę (UC04) 4. System powiadamia Respondentów o Ankiecie 5. Respondent wypełnia Ankietę (UC05) 6. Ankieter sprawdza podsumowanie Ankiety (UC06)

2.4.3 BC03: Zarządzanie Grupami Docelowymi

Przypadek użycia: BC03: Zarządzanie Grupami Docelowymi
Aktorzy: Administrator
Pre: Ankieter chce ankietować Studentów
Post: Ankieta, Raport
Scenariusz Główny
1. Ankieter zgłasza potrzebę stworzenia Grupy Docelowej Administratorowi 2. Administrator podaje nazwę Grupy Docelowej, którą zamierza utworzyć 3. Administrator dodaje/usuwa członków Grupy Docelowej 4. Administrator potwierdza chęć stworzenia Grupy Docelowej 5. System tworzy Grupę Docelową 6. Ankieter może korzystać z Grupy Docelowej

Rozdział 3

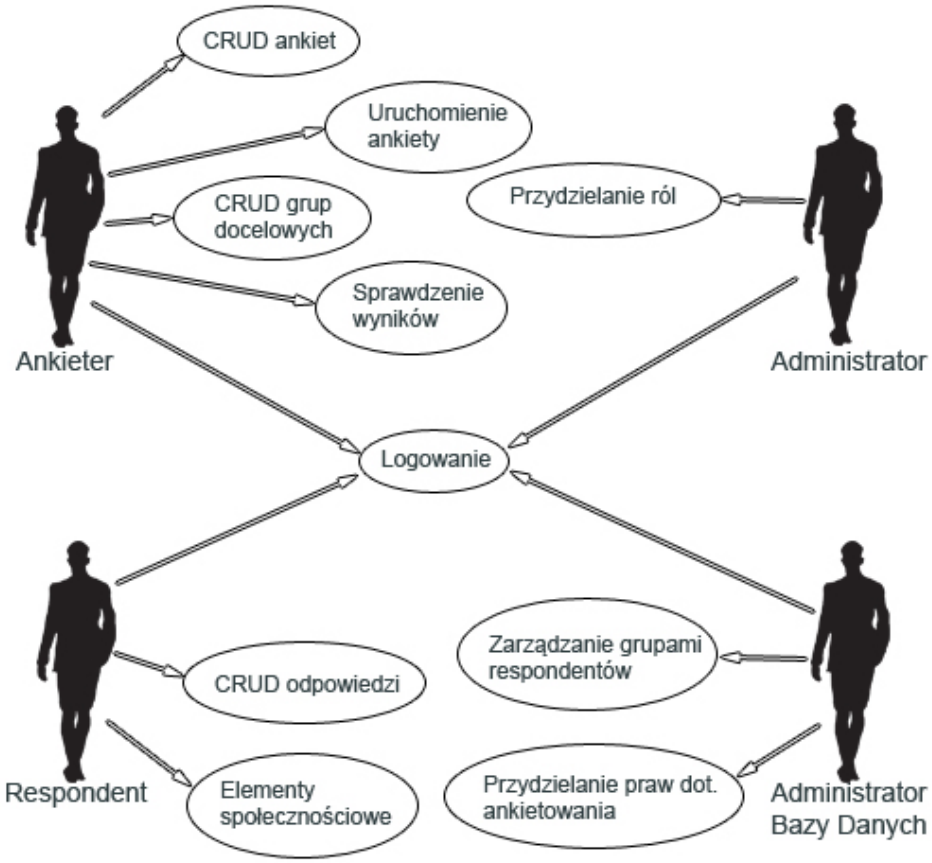
Wymagania funkcjonalne

3.1 Wstęp

Sprawna realizacja celów warunkujących powstanie systemu *iQuest* uzależniona jest od spełnienia szeregu wymagań funkcjonalnych. Należą do nich m.in. tworzenie i prowadzenie (zarówno ze strony Ankietera, jak i Respondenta) badań i ankiet, a także analiza powstałych na ich podstawie zestawień danych. Ze względu na charakter projektu *iQuest*, niezwykle ważną funkcjonalnością okazał się najbardziej podstawowy ze wszystkich mechanizmów – logowanie do systemu. Prawidłowa autoryzacja użytkowników systemu, podobnie jak ich uwierzytelnianie, a także współpraca z innymi systemami Uczelni są niezbędne dla prawidłowości i wymierności prowadzonych badań.

3.2 Diagram przypadków użycia

Na rysunku 3.1 przedstawiono diagram przypadków użycia. W ramach systemu udostępniane są różne funkcje, możliwe do wykonania przez różnych aktorów. Dla przykładu, ankieter może tworzyć badania i analizować ich statystyki oraz powiązane raporty, podczas gdy respondent może brać udział w badaniach odpowiadając na pytania zadawane w ramach skierowanych do niego ankiet; Administrator zarządza przydzielaniem ról; Administrator Bazy Danych (utożsamiany też z Administratorem Danych) zajmuje się grupami Respondentów i przydzielaniem użytkownikom praw i zezwoleń.



Rysunek 3.1: Diagram przypadków użycia (wyk. zespół zarządzający)

3.3 Ankieter

Poniżej przedstawiono przypadki użycia dotyczące Ankietera.

3.3.1 UC01: Stworzenie Ankiety

Przypadek użycia: UC01: Stworzenie Ankiety
Aktorzy: Ankieter
Pre: Ankieter jest zalogowany w Systemie i chce utworzyć Ankietę
Post:
Scenariusz Główny
1. Ankieter uruchamia interfejs tworzenia Ankiety. Podaje atrybuty Ankiety: nazwę Ankiety, wstęp, podsumowanie 2. System prezentuje stronę umożliwiającą dodawanie pytań 3. Ankieter wybiera typ pytania 4. Ankieter podaje treść pytania 5. Ankieter podaje możliwe odpowiedzi 6. Ankieter akceptuje Ankietę 7. System prezentuje podsumowanie ankiety i zapisuje ją w Katalogu Ankiet ankietera
Rozszerzenia
4.A Typ pytania: pytanie otwarte 4.A.1 Ankieter pomija krok 5. 6.A Ankieter chce dodać kolejne pytanie 6.A.1 Powrót do kroku 3.

3.3.2 UC02: Edycja Ankiety

Przypadek użycia: UC02: Edycja Ankiety
Aktorzy: Ankieter
Pre: 1. Ankieta znajduje się w Systemie i jest dostępna dla Ankietera 2. Ankieta nie jest częścią czynnego Badania 3. Ankieter jest zalogowany w Systemie i chce zmodyfikować istniejącą Ankietę
Post:
Scenariusz Główny
1. Ankieter wybiera Ankietę do modyfikacji 2. System prezentuje wskazaną Ankietę 3. Ankieter modyfikuje lub usuwa dostępne pytania 4. Ankieter potwierdza chęć zapisu zmienionej Ankiety 5. System zapisuje zmienioną Ankietę
Rozszerzenia
3.A. Edycja możliwych odpowiedzi do pytań 3.A.1 Ankieter edytuje możliwe odpowiedzi do pytań

3.3.3 UC03: Wybór Grupy Docelowej

Przypadek użycia: UC03: Wybór Grupy Docelowej
Aktorzy: Ankieter, Administrator
Pre: 1. Ankieta znajduje się w Systemie i jest dostępna dla Ankietera 2. Grupa Docelowa znajduje się w Systemie i jest dostępna dla Ankietera 3. Ankieter jest zalogowany w Systemie i chce wybrać Grupę Docelową
Post:
Scenariusz Główny
1. Ankieter wybiera opcję tworzenia Badania 2. System prezentuje listę Grup Docelowych, do których Ankieter ma uprawnienia 3. Ankieter wybiera Grupy Docelowe dla danego Badania 4. Ankieter akceptuje powiązanie Grup Docelowych z Ankietą
Rozszerzenia
3.A. Zawężenie Grupy Docelowej 3.A.1 Ankieter wybiera członków Grupy Docelowej, do której ma być skierowana Ankieta. 3.A.2 Powrót do kroku 4. 3.B. Grupa Docelowa poszukiwana przez Ankietera nie istnieje 3.B.1 Ankieter próbuje połączyć kilka Grup Docelowych lub ich fragmentów 3.B.2 W przypadku niepowodzenia kroku rozszerzenia 3.B.1, bądź wystąpienia takiej konieczności, Ankieter informuje Administratora, że nie ma praw do wysyłania ankiet do wskazanych osób i/lub powiadamia go (za pomocą poczty elektronicznej) o potrzebie stworzenia Grupy Docelowej o konkretnych atrybutach (BC03) 3.B.3 W przypadku pominięcia kroku rozszerzenia 3.B.2, powrót do kroku 4., w przeciwnym razie, powrót do kroku 1.

3.3.4 UC04: Uruchomienie Badania

Przypadek użycia: UC04: Uruchomienie Badania
Aktorzy: Ankieter, Respondent
Pre: 1. Ankieta znajduje się w Systemie i jest dostępna dla Ankietera 2. Ankieta jest powiązana z Grupą Docelową 3. Ankieter jest zalogowany w Systemie i chce rozesłać istniejącą Ankietę
Post: Respondenci powiadomieni o Ankiecie
Scenariusz Główny
1. Ankieter ustawia czas rozpoczęcia i zakończenia Badania oraz grupę docelową (UC3) 2. Ankieter potwierdza chęć uruchomienia Badania 3. System udostępnia Ankietę Respondentom 4. System powiadamia Respondentów o Ankiecie
Rozszerzenia
1.A. Ankieter chce prowadzić cykliczną ankietyzację 1.A.1 Ankieter ustawia częstotliwość powtarzania Badania

3.3.5 UC06: Sprawdzenie wyników

Przypadek użycia: UC06: Sprawdzenie wyników
Aktorzy: Ankieter
Pre: 1. Ankieta znajduje się w Systemie, zawiera odpowiedzi od Grupy Docelowej i jest dostępna dla Ankietera 2. Ankieter jest zalogowany w Systemie i chce pozyskać informacje od Studentów/Absolwentów
Post: Wygenerowany Raport
Scenariusz Główny
1. Ankieter wybiera Ankietę, której wyniki chce poznać 2. Ankieter wybiera typ Raportu, który chciałby zobaczyć 3. System generuje i wyświetla Raport

3.4 Respondent

Poniżej przedstawiono przypadki użycia dotyczące Respondenta.

3.4.1 UC05: Udzielenie odpowiedzi

Przypadek użycia: UC05: Udzielenie odpowiedzi
Aktorzy: Respondent
Pre: 1. Respondent dostaje powiadomienie o Ankiecie (link bezpośredni do Ankiety) 2. Respondent jest zalogowany w Systemie i chce wypełnić Ankietę
Post:
Scenariusz Główny
1. Respondent wybiera Ankietę, w której chce wziąć udział 2. System prezentuje wybraną Ankietę Respondentowi 3. Respondent udziela odpowiedzi na pytania 4. Respondent zatwierdza wypełnioną Ankietę 5. System zapisuje odpowiedzi
Rozszerzenia
2.A. Przedawniona Ankieta 2.A.1 System informuje, że Ankieta już się zakończyła 5.A. Brak odpowiedzi na niektóre pytania 5.A.1 System pozwala powrócić do pytań, na które nie udzielono odpowiedzi 5.A.2 Powrót do kroku 1.

3.5 Administrator

Poniżej przedstawiono przypadki użycia dotyczące Administratora.

3.5.1 UC07: Tworzenie Grupy Docelowej

Przypadek użycia: UC07: Tworzenie Grupy Docelowej
Aktorzy: Administrator
Pre: 1. Ankieter chce wysyłać Ankiety do określonych Respondentów w prosty sposób 2. Administrator jest zalogowany w Systemie i chce utworzyć nową Grupę Docelową
Post: Nowa Grupa Docelowa w Systemie
Scenariusz Główny
1. Administrator wybiera opcję tworzenia Grup Docelowych 2. System prezentuje formularz tworzenia Grupy Docelowej 3. Administrator wprowadza nazwę tworzonej Grupy Docelowej, wybiera Grupę Nadrzędną oraz Respondentów do dodania do Grupy Docelowej 4. Administrator potwierdza chęć stworzenia Grupy Docelowej 5. System zapisuje nową Grupę Docelową
Rozszerzenia
4.A Brak Grupy Nadrzędnej 4.A.1 Administrator Bazy Danych nie uzupełnia Grupy Nadrzędnej

3.5.2 UC08: Edycja Grupy Docelowej

Przypadek użycia: UC08: Edycja Grupy Docelowej
Aktorzy: Administrator
Pre: 1. Grupa Docelowa znajduje się w Systemie 2. Administrator jest zalogowany w Systemie i chce zmodyfikować Grupę Docelową
Post: Zmodyfikowana lista członków Grupy Docelowej
Scenariusz Główny
1. Administrator wybiera Grupę Docelową 2. Administrator wybiera członka/członków Grupy Docelowej do edycji/usunięcia 3. Administrator dodaje/edytuje/usuwa członka/członków Grupy Docelowej 4. Administrator potwierdza chęć wprowadzenia zmian 5. System zapisuje zmiany

3.6 Wszyscy Użytkownicy

Poniżej przedstawiono przypadki użycia dotyczące wszystkich Użytkowników.

3.6.1 UC09: Logowanie do Systemu

Przypadek użycia: UC09: Logowanie do Systemu
Aktorzy: Użytkownik (Ankieter, Administrator, Respondent)
Pre: Użytkownik posiada konto w Systemie i posiada poprawne dane logowania
Post: Użytkownik jest zalogowany w Systemie
Scenariusz Główny
1. System wyświetla Użytkownikowi formularz logowania 2. Użytkownik podaje login lub adres e-mail oraz hasło 3. System uwierzytelnia Użytkownika
Rozszerzenia
2.A Użytkownik chce się zalogować przy pomocy *eKonta* 2.A.1 Użytkownik wybiera opcję logowania przez *eKonto* 2.A.2 System przekierowuje Użytkownika na stronę logowania przez *eKonto* (*eLogin*) 2.A.3 Użytkownik wprowadza dane logowania do *eKonta* 2.A.4 Powrót do kroku 3.

Rozdział 4

Wymagania pozafunkcjonalne

4.1 Wstęp

W niniejszym rozdziale zostaną zaprezentowane i krótko opisane wymagania pozafunkcjonalne dotyczące systemu *iQuest*. Dodatkowo, zostały one przydzielone do wybranych przez zespół zarządzający charakterystyk oprogramowania.

4.2 Opis wymagań

Wymagania pozafunkcjonalne przypisane do wyżej opisanych charakterystyk oprogramowania przedstawione zostały w tabeli 4.1. Dodatkowo, określono za pomocą etykiety *ważność (W)* oraz *trudność implementacji (TI)* dla każdego z wymagań. Wartości w ramach tego oznaczenia wyrażone są w następującej skali:

- H (ang. *high*) – wysoka
- M (ang. *medium*) – średnia
- L (ang. *low*) – niska

Analizy, zakończonej etykietowaniem, dokonał zespół zarządzający, rozpatrując to wedle wytycznych przedstawionych w trakcie zajęć akademickich na 1. roku studiów II stopnia.

Charakterystyka	**Wymaganie**	**W**	**TI**
Analizowalność	Komentarze w kodzie źródłowym powinny być w języku angielskim.	M	L
Analizowalność	Kod źródłowy sytemu powinien być utworzony zgodnie ze standardami *Moodle*.	M	L
Analizowalność	System powinien zawierać testy jednostkowe.	M	M
Analizowalność	System powinien rejestrować stack trace i rodzaj błędu (fatal, warning).	H	L
Analizowalność	Wraz z kodem źródłowym systemu należy dostarczyć dokumentację.	H	M
Analizowalność	System powinien logować niewłaściwe wywołania metod.	H	L
Autentyczność	Gdy student staje się absolwentem, należy umożliwić mu dalsze logowanie się do systemu bez użycia *eKonta*.	H	M
Bezpieczeństwo (wolność od ryzyka)	Dane (opinie respondentów) przechowywane w systemie muszą być uzyskiwane poprzez wbudowane mechanizmy ankietowania.	H	L
Bezpieczeństwo (wolność od ryzyka)	Projekt systemu należy poddać analizie z wykorzystaniem metody ATAM.	M	M
Charakterystyka czasowa	Wyświetlenie ankiety powinno trwać nie dłużej niż 4 sekundy.	H	L
Charakterystyka czasowa	Generowanie raportów powinno odbywać się ze średnio nie dłużej niż 1 godzinę.	M	M
Charakterystyka czasowa	Należy zdefiniować klasy operacji, w zależności od czasu ich trwania. Klasy: • bez komunikatu potwierdzającego wykonanie • z potwierdzeniem wykonania • wykonywane na serwerze w tle	M	M
Dostępność personalna	Przewidzieć, na poziomie architektury, możliwość rozbudowy np. o interfejs dla osób niedowidzących.	L	M
Dostępność techniczna	System może mieć przerwę serwisową, lecz musi wówczas prezentować specjalny ekran informujący o czasie jej trwania.	H	L
Estetyka Interfejsu Użytkownika	Środowisko ma być przyjazne i czytelne dla użytkownika końcowego.	H	M
Funkcjonalna poprawność	Wszystkie wartości mają być prezentowane z dokładnością do 2 miejsc po przecinku.	H	L
Identyfikowalność	System ma umożliwiać identyfikowanie podmiotów (osobno: administratorów, ankieterów, respondentów), podejmujących konkretne działania: tworzenie ankiet, odpowiadanie w ankietach, itp.	H	M
Integralność	Baza danych powinna być chroniona przed nieuprawnionym dostępem [modyfikacją/usunięciem] w następujący sposób: logowanie za pomocą loginu i hasła.	H	L

Integralność	System powinien być odporny na następujące próby nielegalnego dostępu: nieuprawniony dostęp fizyczny do serwera.	M	L
Integralność	Należy chronić/szyfrować dane przesyłane z i do systemu.	M	L
Interoperacyjność	System ma wymieniać potrzebne dane z systemami uczelnianymi: *eKonto*, *eDziekanat*, *ePoczta*. Dane mają być aktualne.	H	M
Interoperacyjność	System ma pobierać dane z systemu *eDziekanat* w następujący sposób: *SOAP*.	H	M
Interoperacyjność	System ma przesyłać dane o wiadomościach do wysłania do systemu *ePoczta* w następujący sposób: *SOAP*.	H	M
Interoperacyjność	System ma pobierać dane do autoryzacji z systemu *eKonto* w następujący sposób: *SOAP*.	H	M
Interoperacyjność	System ma przesyłać wyniki ankiet do systemu *BI*[1].	H	M
Kompletność kontekstowa	System ma działać w przeglądarkach: *IE 7.0+*, *Firefox 15*, *Opera 12*.	H	M
Kompletność kontekstowa	Należy przygotować raport jak system zachowuje się na platformach mobilnych.	L	L
Łatwość adaptacji	Należy utworzyć raport z łatwości adaptacji oraz gdzie znajdują się adaptery.	M	L
Łatwość instalacji	System musi umożliwiać łatwą aktualizacje, przy założeniu, że wersja platformy *Moodle* pozostaje bez zmian.	H	H
Łatwość nauczenia się	Interfejs użytkownika (dla ankietowanych) powinien być całkowicie intuicyjny.	H	M
Łatwość nauczenia się	Interfejs użytkownika (dla ankietera) może wymagać nieznacznego doszkolenia obsługujących.	M	L
Łatwość testowania	„Atrapy" (ang. *mock*) systemów zewnętrznych (m.in. *eKonto*, *ePoczta*).	M	M
Łatwość zamiany	Należy umożliwić przełączanie systemu między trybem testowym i produkcyjnym.	M	L
Łatwość zamiany	System powinien umożliwiać wczytanie wszystkich danych z poprzedniej wersji.	H	M
Łatwość zamiany	Procedura wymiany oprogramowania powinna trwać nie dłużej niż 2 dni i odbywać się w następujący sposób: zgodność z instrukcją.	M	L
Łatwość zamiany	Podczas projektowania systemu, należy brać pod uwagę możliwość wprowadzenia wielojęzyczności interfejsu.	M	M
Łatwość zmiany	System powinien być przygotowany na wprowadzenie następujących zmian: nowe typy raportów, nowe typy pytań, modyfikacje interfejsów systemów zewnętrznych.	M	M
Niezaprzeczalność	System musi posiadać logi (zalogowanie w systemie, stworzenie/edycja/usunięcie/wysłanie/wypełnienie ankiety), aby móc udokumentować skąd pochodzą dane.	H	L

[1] ang. *Business Intelligence*

Ochrona użytkownika przed błędami	Dodatkowe potwierdzenie chęci wykonania operacji nieodwracalnych (nawet dla administratora), lub możliwość przywrócenia usuniętych danych przez jakiś czas.	H	M
Ochrona użytkownika przed błędami	Dla dużych ankiet, zatwierdzenie odesłania jej przez ankietowanego.	M	L
Ochrona użytkownika przed błędami	Potwierdzenie przed rozesłaniem ankiet.	M	L
Ochrona użytkownika przed błędami	Lista operacji wykonywanych w tle.	M	L
Odporność na wady	Gdy nastąpi awaria innych systemów np. *eKonto*, należy poinformować użytkownika o błędzie i uniemożliwić mu dalsze działanie w systemie.	H	L
Odtwarzalność	Odtwarzanie całego systemu w czasie nieprzekraczającym 3h.	M	L
Odtwarzalność	Kopia zapasowa bazy danych wykonywana z częstotliwością raz na dobę.	H	L
Odtwarzalność	Dostępność instrukcji odtwarzania.	H	L
Odtwarzalność	Wykonywanie operacji w transakcjach tam, gdzie to możliwe.	H	L
Poufność	Uwierzytelnianie i autoryzacja dla każdej operacji.	M	L
Współistnienie	Jedynym kryterium działania systemu ankiet (bez raportów) jest działająca przeglądarka.	H	L
Współistnienie	System generowania raportów może być osobną aplikacją, działającą na specjalnych: sprzęcie i oprogramowaniu.	L	L
Współistnienie	Wszystkie potrzebne dane z zewnętrznych systemów należy przechowywać lokalnie i synchronizować.	M	H
Współistnienie	Należy kolejkować zadania (np. e-maile) w systemie, na czas braku możliwości ich wysłania (np. awaria *ePoczty*).	H	M
Zużycie zasobów	System nie powinien wykorzystywać więcej serwerów *HTTP* niż jeden.	M	L
Zużycie zasobów	System powinien działać poprawnie przy nawet 10.000 użytkowników zalogowanych jednocześnie.	H	M
Zużycie zasobów	Zadania, które nie muszą zostać wykonane natychmiastowo, powinny być kolejkowane przez system.	M	M
Zużycie zasobów	Możliwość pracy przez ankietera/respondenta na średniej klasy laptopie (CPU 2GHz, RAM 4GB).	H	L

Tablica 4.1: Wymagania pozafunkcjonalne

Zważywszy, że projekt ten jest istotny dla Uczelni, dołożono wszelkich starań, aby spełnić wszystkie wymagania pozafunkcjonalne. Zadanie to zostało w sporej części wykonane dzięki zastosowaniu sprawdzonego i znanego narzędzia w roli podstawy dla całego systemu. Mowa o platformie *Moodle*[2]. Użycie jej pozwoliło m.in. na zapewnienie modułowości systemowi. Spełniona została także adaptatywność i przenośność systemu, oparta na uniwersalności języka *PHP*, jak też

[2]Więcej informacji w rozdziale 6, poświęconym implementacji i zastosowanym przy niej technologiom.

szerokiej gamie obsługiwanych przez *Moodle* systemów baz danych.

Zalety wykorzystania tak powszechnej technologii były – przynajmniej w teorii – znacznie szersze. *Moodle* posiada swój własny standard kodowania, którego przy realizowaniu projektu starano się w pełni przestrzegać. To sprawia, że w połączeniu z wytycznymi *DRO* (*Dział Rozwoju Oprogramowania Politechniki Poznańskiej*), dokumentacją *Moodle* i dokumentacją systemu *iQuest*, nakład pracy wymagany do wdrożenia się, w celu rozwoju projektu, został zmniejszony.

W trakcie prac nad projektem, niemały nacisk stawiano na prawidłowe zarządzanie uprawnieniami użytkowników. Wykorzystano w tym celu mechanizmy dostępne w platformie *Moodle*. Są one weryfikowane przy każdym działaniu podejmowanym przez użytkowników i zależą od przydzielonej im w systemie roli. Szczegóły tej kwestii swobodnie definiować może administracja. Ważnym jest, że osoby pełniące rolę ankietera mogą modyfikować i analizować jedynie te badania, które zostały utworzone przez siebie lub udostępnione im bezpośrednio. Podobnie działają uprawnienia przydzielone do roli respondenta – pozwalając dokładnie jeden raz odpowiedzieć na pytania w ankietach skierowanych do grup docelowych, do których należy użytkownik.

Rozdział 5

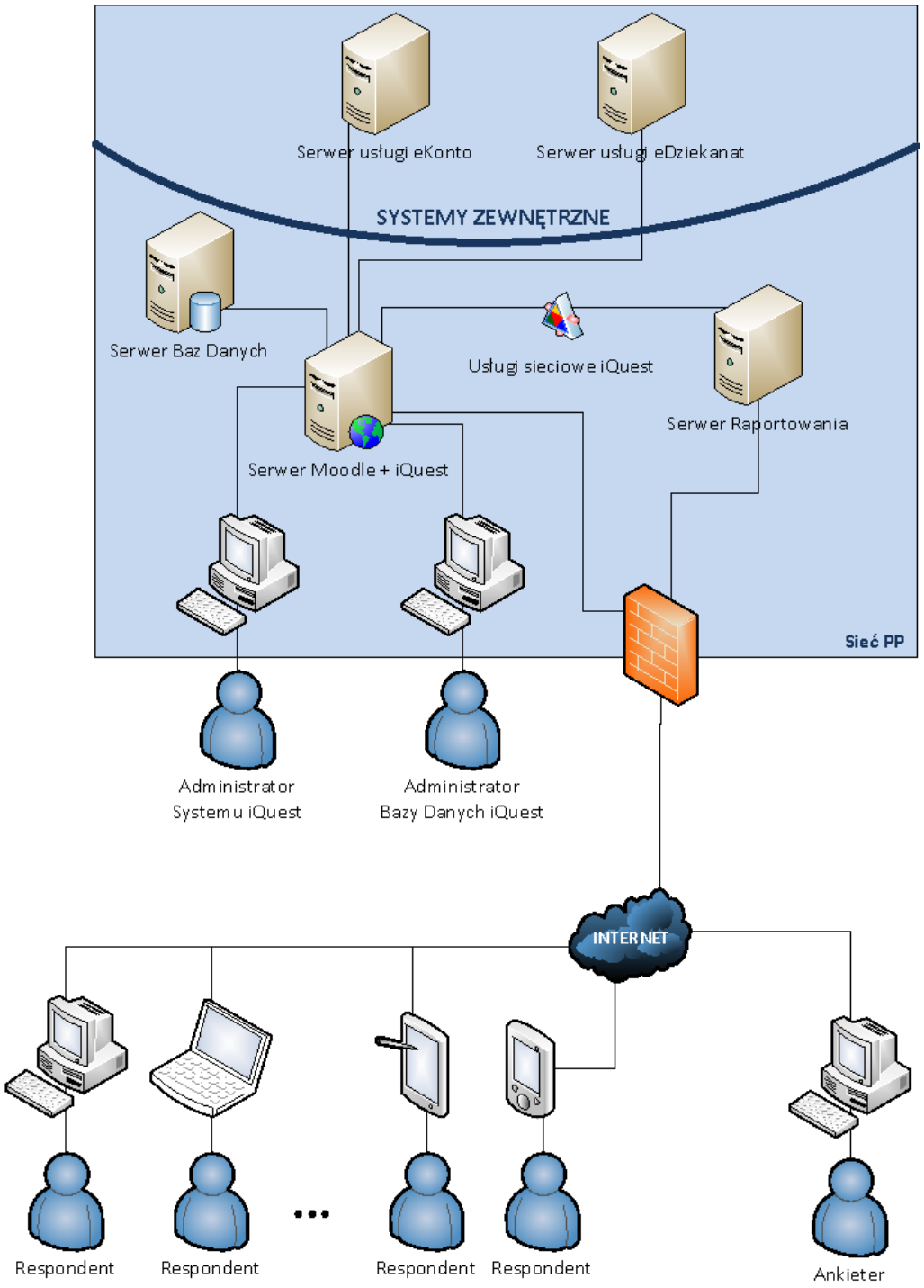
Architektura systemu

5.1 Wstęp

System *iQuest* został stworzony w oparciu o model architektury trójwarstwowej, w którym wyróżnione zostały warstwy: danych, logiki biznesowej oraz prezentacji. Dzięki takiemu podejściu, zadania związane z poszczególnymi warstwami można było bez większego problemu rozdzielić pomiędzy członków zespołu programistów, a w przypadku ewentualnej modyfikacji jednej z warstw nie występuje konieczność wprowadzania zmian w pozostałej części projektu.

5.2 Opis ogólny architektury – „Marketektura"

System *iQuest* to aplikacja internetowa w postaci zbioru rozszerzeń platformy e-learningowej *Moodle*. Całość (*Moodle* oraz rozszrzenia) zainstalowana jest na serwerze www, zlokalizowanym w sieci Politechniki Poznańskiej, łączącym się z osobnym serwerem baz danych oraz usługami *eKonto* i *eDziekanat*. Funkcje raportowania realizowane są w głównej mierze za pośrednictwem zewnętrznego systemu *BI*, pobierającego dane z *iQuest* za pośrednictwem usług sieciowych (ang. *webservices*). Administracja oraz obsługa systemu odbywa się za pośrednictwem przeglądarki internetowej, w ramach połączenia z platformą *Moodle*, lub serwerem raportowania. Respondenci mogą też uzyskać dostęp do systemu przy pomocy urządzeń mobilnych, takich jak tablety czy smartfony.



Rysunek 5.1: Diagram „Marketektury" (wyk. zespół zarządzający)

5.3 Analiza podejść/Analiza SWOT

W fazie inicjacji projektu, zespół zarządzający przeprowadził analizę SWOT (ang. *Strengths, Weaknesses, Opportunities, Threats* - mocne strony, słabe strony, szanse, zagrożenia). Poniżej zamieszczono tabele 5.1. oraz 5.2., zawierające jej wyniki. W oparciu o nie zespół zarządzający podjął decyzję o realizacji projektu z użyciem platformy *Moodle*.

—	**Pozytywne**	**Negatywne**
Wewnętrzne	**Mocne strony (S)** Środowisko *Moodle* jest dobrze udokumentowane[5]. Środowisko *Moodle* jest łatwe w rozwoju. Stosowanie podejścia MVC w środowisku *Moodle*. Już zaimplementowane mechanizmy, takie jak kontrola praw dostępu, CMS. Posiadanie bazy wiedzy, która ma zachęcać użytkowników.	**Słabe strony (W)** Konieczność trzymania się standardów *Moodle*. Konieczność wdrożenia się w nieznany kod.
Zewnętrzne	**Szanse (O)** Modułowość platformy *Moodle*. Łatwość tworzenia własnych wtyczek.	**Zagrożenia (T)** Środowisko może być nieznane programistom.

Tablica 5.1: Podejście pierwsze – w oparciu o platformę *Moodle*

—	**Pozytywne**	**Negatywne**
Wewnętrzne	**Mocne strony (S)** Dostępność platformy na wszystkich popularnych systemach operacyjnych.	**Słabe strony (W)** Konieczność implementacji dodatkowych funkcjonalności związanych z zachęcaniem do korzystania z systemu.
Zewnętrzne	**Szanse (O)** Dobra znajomość kodu przez programistów.	**Zagrożenia (T)** Konieczność tworzenia wszystkiego od podstaw. Nieznajomość technologii przez programistów.

Tablica 5.2: Podejście drugie – napisanie aplikacji od podstaw przy użyciu *Java EE*

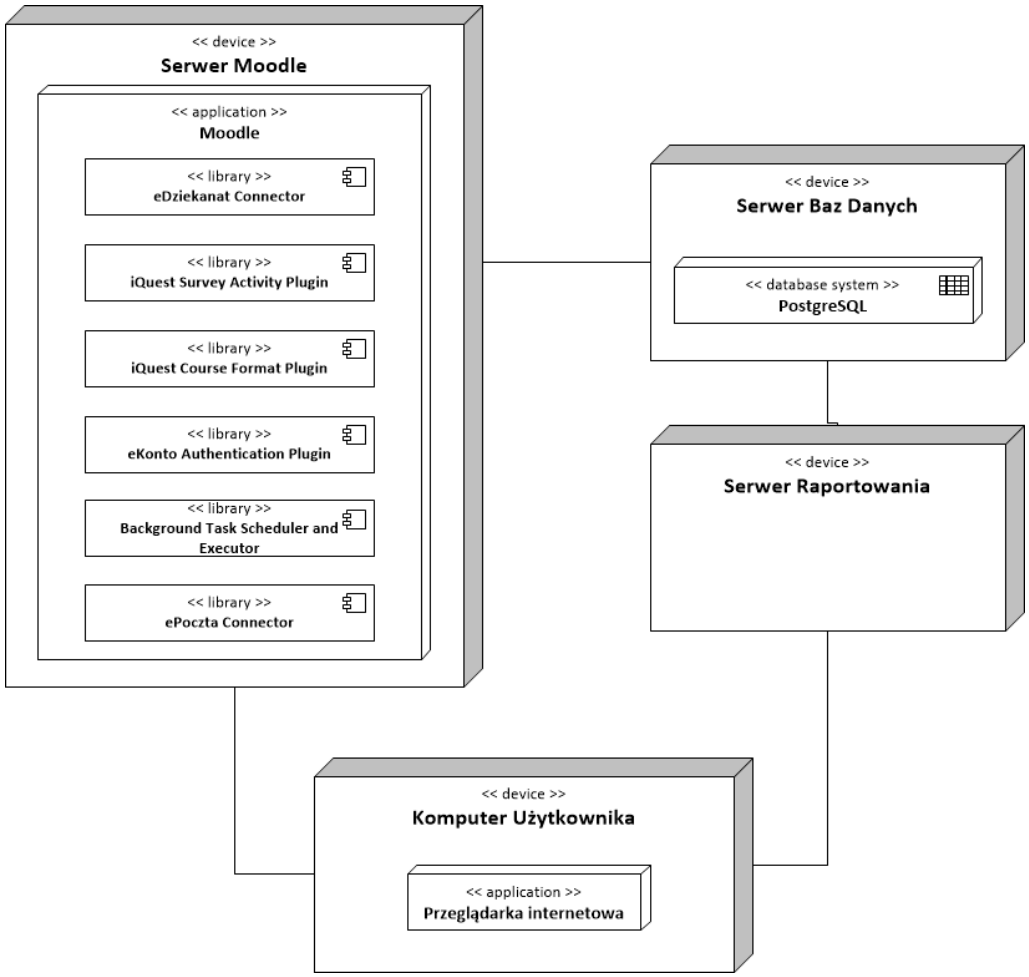
Rozpatrując powyższe tabele z perspektywy czasu, autorzy niniejszej pracy stwierdzili, iż analiza SWOT została zrealizowana nieprawidłowo, faworyzując rozwiązanie z użyciem systemu *Moodle*. Dla przykładu, dokumentacja *Moodle* okazała się w znacznym stopniu nieprecyzyjna i niekompletna, zaś w kodzie platformy nie stosuje się podejścia MVC. Prawidłowa, zdaniem zespołu programistów, analiza podejść dla pierwszego podejścia, została zamieszczona w tabeli 5.3. Niestety, nie została ona wzięta pod uwagę w trakcje projektowania systemu *iQuest*.

—	**Pozytywne**	**Negatywne**
Wewnętrzne	**Mocne strony (S)** Środowisko *Moodle* jest oparte na modułach, co częściowo ułatwia rozwój. Już zaimplementowane mechanizmy, takie jak kontrola praw dostępu, CMS.	**Słabe strony (W)** Dokumentacja platformy *Moodle*[5] jest niekompletna i nieprecyzyjna. Konieczność trzymania się standardów *Moodle*. Konieczność wdrożenia się w obcy kod.
Zewnętrzne	**Szanse (O)** *Moodle* jest znany w kręgach akademickich, w tym wśród przyszłych użytkowników systemu *iQuest*. Możliwość utworzenia bazy wiedzy, zachęcającej użytkowników do korzystania z funkcjonalności systemu *iQuest*.	**Zagrożenia (T)** Środowisko (w kategorii implementacyjnej) może być nieznane programistom.

Tablica 5.3: Podejście pierwsze – w oparciu o platformę *Moodle* – zdaniem zespołu programistów

5.4 Perspektywy architektoniczne

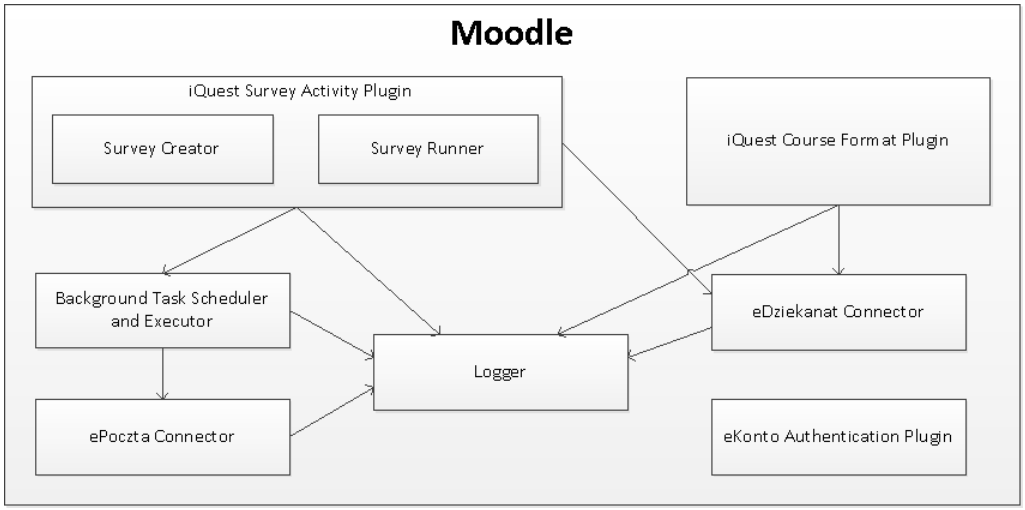
5.4.1 Perspektywa fizyczna



Rysunek 5.2: Diagram perspektywy fizycznej (wyk. zespół zarządzający)

Schemat 5.2 prezentuje perspektywę fizyczną projektu. Widać na nim dokładnie, opisaną wcześniej, opartą na rozszerzeniach dla platformy *Moodle* budowę logiki systemu *iQuest*. Za jej pośrednictwem dokonuje się połączeń z serwerem baz danych oraz serwerem raportowania. Użytkownik systemu, z wykorzystaniem przeglądarki internetowej, komunikuje się z platformą, lub systemem raportowania, korzystając przy tym z warstwy prezentacji.

5.4.2 Perspektywa logiczna



Rysunek 5.3: Diagram perspektywy logicznej (wyk. zespół zarządzający)

Schemat 5.3 prezentuje perspektywę logiczną systemu. Określa ona zależności między poszczególnymi komponentami „wszczepionymi" do platformy *Moodle*. Poniżej znajduje się opis wyszczególnionych na rysunku komponentów.

iQuest Survey Activity Plugin

Funkcjonalnością *Activity Plugin* jest udostępnianie możliwości dodania nowych rodzajów tzw. „aktywności" w ramach platformy *Moodle*. *iQuest Survey Activity Plugin* pozwala na dodanie nowego badania *iQuest*. Komponent ten składa się z dwóch subkomponentów:

- Survey Creator
- Survey Runner

Pierwszy z nich odpowiada za definiowanie ankiet, natomiast drugi za ich realizację.

iQuest Course Format Plugin

Course Format Plugin dla platformy *Moodle* odpowiada za obsługę interfejsu użytkownika. W przypadku systemu *iQuest*, zarządza kwestią wyświetlania użytkownikowi tylko tych składowych kursu związanego z systemem *iQuest*, które są dla niego dostępne. Przykładowo, Respondent uzyska dostęp do listy ankiet, które może wypełnić, podczas gdy ankieter uzyska dostęp do listy zarządzanych przez niego badań.

eDziekanat Connector oraz ePoczta Connector

Komponenty te odpowiadają za komunikację z usługami *eDziekanat* i *ePoczta*, pozwalające m.in. na pozyskanie danych o grupach docelowych (w oparciu o dane Grup Dziekańskich) oraz wysyłanie powiadomień za pośrednictwem poczty studenckiej.

Background Task Scheduler and Executor

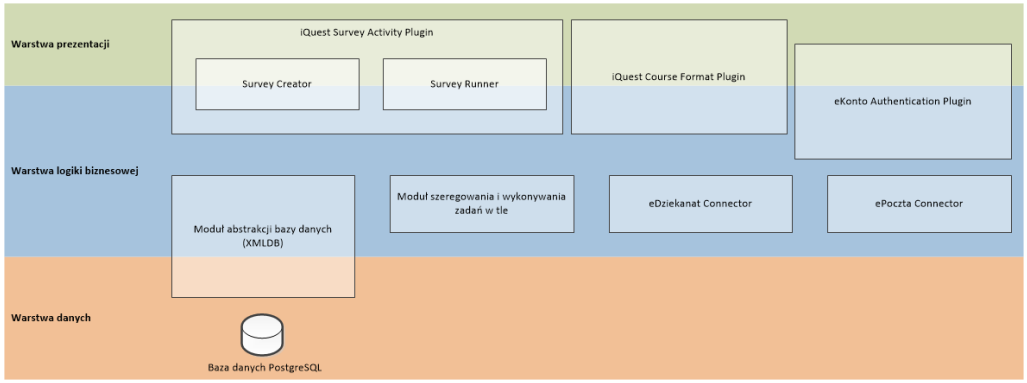
Dzięki temu komponentowi możliwe jest szeregowanie oraz wykonywanie zadań w tle. Jednym z jego zadań jest kolejkowanie i aktywowanie mechanizmów rozsyłania wiadomości e-mail z zaproszeniami do udziału w ankiecie.

eKonto Authentication Plugin

eKonto Authentication Plugin – to moduł uwierzytelniania (ang. *Authentication*), korzystający z systemu *eLogin* platformy *eKonto* do logowania się do platformy *Moodle*, obsługującej system *iQuest*. Korzystanie z tego systemu pozwala nie tylko na jednoznaczną weryfikację tożsamości użytkownika łączącego się z systemem, ale jest zarazem wygodne – dzięki jego zastosowaniu, nie ma potrzeby posiadania osobnego konta w systemie *iQuest*.

5.4.3 Perspektywa implemetancyjna

Diagram perspektywy implementacyjnej (zwanej też perspektywą warstw; rys. 5.4) przedstawia graficzną interpretację zależności między poszczególnymi elementami systemu *iQuest* w nawiązaniu do warstw zastosowanego modelu trójwarstwowego. Funkcje zaprezentowanych na nim modułów zostały wyjaśnione już wcześniej w niniejszym rozdziale.



Rysunek 5.4: Diagram perspektywy implementacyjnej (wyk. zespół zarządzający)

5.5 Decyzje projektowe

5.5.1 Wstęp

W procesie rozwoju systemu *iQuest* podjęto dokładnie 40 decyzji projektowych. Większość decyzji podejmował Architekt. W części przypadków rozpatrywano także zdanie zespołu programistów.

5.5.2 Podjęte decyzje

Identyfikator	D01
Nazwa	Postać systemu
Opis	System iQuest jest zestawem rozszerzeń środowiska *Moodle*
Uzasadnienie	Platforma *Moodle* jest już stosowana na uczelni, więc łatwiej jest dotrzeć z ankietami do studentów, jeżeli można je wypełniać bezpośrednio w *Moodle'u*.
Źródło	Architekt

Identyfikator	D02
Nazwa	Podział na komponenty
Opis	*iQuest* składa się z następujących głównych komponentów: *iQuest Course Format Plugin*, *iQuest Survey Activity Plugin*, *eDziekanat Connector*, *eKonto Authentication Plugin*, *Background Task Scheduler and Executor*, *Logger*
Uzasadnienie	Dzięki budowie modułowej łatwiej jest analizować, implementować i testować system.
Źródło	Architekt

Identyfikator	D03
Nazwa	*iQuest* jako kurs na *Moodle'u*
Opis	*iQuest* widoczny jest dla użytkowników platformy *Moodle* jako specjalny kurs.
Uzasadnienie	Jest to zgodne z ideą *Moodle'a* – treści umieszczane na tej platformie są pogrupowane w kursach.
Źródło	Architekt

Identyfikator	D04
Nazwa	Wtyczka *iQuest Course Format Plugin*
Opis	*iQuest Course Format Plugin* odpowiada za prezentowanie treści wewnątrz kursu, w ramach którego funkcjonuje system *iQuest*. Plugin dba o to, aby ankieterowi wyświetlała się lista tylko tych badań, do których ma on uprawnienia oraz żeby respondent widział listę tylko tych badań, w których ma on prawo wypełnić ankietę.
Uzasadnienie	Istnienie takiej wtyczki jest wymagane, ponieważ bez niej nie byłoby możliwe filtrowanie listy badań tak, aby użytkownik wydział tylko te do których ma uprawnienia.
Źródło	Architekt

Identyfikator	D05
Nazwa	*Wtyczka eKonto Authentication Plugin*
Opis	Logowanie użytkowników przy pomocy *eKonta* realizowane jest poprzez utworzenie rozszerzenia *eKonto Authentication Plugin* dla platformy *Moodle*.
Uzasadnienie	Platforma *Moodle* jest tak skonstruowana, że aby umożliwić uwierzytelnianie przy pomocy zewnętrznych mechanizmów, konieczne jest utworzenie specjalnej wtyczki.
Źródło	Architekt

Identyfikator	D06
Nazwa	Wtyczka *iQuest Survey Activity Plugin* - funkcjonalność dla ankieterów
Opis	*iQuest Survey Activity Plugin* pozwala ankieterowi na tworzenie, edytowanie i usuwanie ankiet, a także na tworzenie, edytowanie i usuwanie badań.
Uzasadnienie	Wtyczka typu *Activity Plugin* dla *Moodle'a* posiada wszystkie cechy potrzebne do zrealizowania tego typu funkcjonalności w sposób intuicyjny dla użytkownika.
Źródło	Architekt

Identyfikator	D07
Nazwa	Wtyczka *iQuest Survey Activity Plugin* - funkcjonalność dla respondentów
Opis	*iQuest Survey Activity Plugin* pozwala respondentowi na udzielenie odpowiedzi w ankiecie.
Uzasadnienie	Wtyczka typu *Activity Plugin* dla *Moodle'a* posiada wszystkie cechy potrzebne do zrealizowania tego typu funkcjonalności w sposób intuicyjny dla użytkownika.
Źródło	Architekt

Identyfikator	D08
Nazwa	Wykorzystanie protokołu HTTPS
Opis	*Moodle* (w ramach którego działa *iQuest*) będzie dostępny dla użytkowników poprzez protokół HTTPS.
Uzasadnienie	Zastosowanie protokołu HTTPS jest łatwym sposobem na dość skuteczne zabezpieczenie danych przesyłanych między klientem a serwerem.
Źródło	Architekt

Identyfikator	D09
Nazwa	Mechanizm kont i ról użytkowników
Opis	Wykorzystany jest istniejący w systemie *Moodle* mechanizm kont i ról użytkowników.
Uzasadnienie	Dzięki temu nie ma potrzeby implementowania osobnego mechanizmu. Ten istniejący w *Moodle'u* spełnia niezbędne wymagania.
Źródło	Architekt

Identyfikator	D10
Nazwa	Integracja *eKonta* z kontem na *Moodle'u*
Opis	W momencie pierwszego logowania danego użytkownika przy pomocy *eKonta*, jest mu tworzone profil w systemie *Moodle*. Za nazwę użytkownika przyjmuje się adres e-mail. Do profilu, dzięki możliwościom usługi *eKonto* kopiowane są także inne dane użytkownika (np. imię i nazwisko).
Uzasadnienie	Takie działanie jest wymagane, aby zainicjalizować konto użytkownika logującego się do *Moodle'a* przez *eKonto* po raz pierwszy.
Źródło	Architekt

Identyfikator	D11
Nazwa	Sposób zapamiętywania ustawień systemu *iQuest*
Opis	Ustawienia systemu *iQuest* przechowywane są w specjalnej tabeli bazy danych, w postaci identyfikator-wartość, które są ciągami znaków.
Uzasadnienie	Umożliwia to łatwy i wygodny dostęp do ustawień w różnych miejscach kodu źródłowego systemu.
Źródło	Architekt

Identyfikator	D12
Nazwa	Dziennik zdarzeń
Opis	Wszystkie ważniejsze zdarzenia w systemie *iQuest* są zapisywane w logu w bazie danych, w specjalnej tabeli. Dla każdego zdarzenia zapisane zostaną następujące dane: data i godzina, identyfikator użytkownika wykonującego operację (o ile dotyczy), tekstowy opis, typ (informacja, ostrzeżenie, błąd).
Uzasadnienie	Mechanizm logowania zdarzeń obecny w *Moodle'u* nie spełnia wymagań - nie pozwala na dzielenie wpisów na kategorie. Z tego powodu została podjęta decyzja o utworzeniu osobnego mechanizmu.
Źródło	Architekt

Identyfikator	D13
Nazwa	Listy ACL dla badań
Opis	Z każdą ankietą jest powiązana lista ACL zawierająca informacje o tym, którzy użytkownicy mają prawo wykorzystywać ankietę w swoim badaniu, którzy mogą ją edytować, a którzy mogą ją usuwać.
Uzasadnienie	Jest to elastyczny sposób zarządzania uprawnieniami do ankiet.
Źródło	Architekt

Identyfikator	D14
Nazwa	Wpisy na listach ACL dla badań - rodzaje podmiotów
Opis	Wpisy na listach ACL wspomnianych w decyzji D13 mogą dotyczyć konkretnych użytkowników, albo ról.
Uzasadnienie	Znacznie ułatwi to zarządzanie uprawnieniami do ankiet.
Źródło	Architekt

Identyfikator	D15
Nazwa	Wykonywanie zadań w tle
Opis	System *iQuest* posiada możliwość wykonywania złożonych zadań w tle. Poszczególne moduły systemu mogą dodawać zlecenia, z których każde składa się z jednego lub wielu zadań.
Uzasadnienie	Pozwala to na poprawienie responsywności systemu dla użytkownika (np. ankieter nie musi czekać, aż wyślą się e-maile z zaproszeniami dla respondentów). Podział zleceń na zadania umożliwia łatwiejszą obsługę sytuacji, w której w trakcie realizowania zlecenia następuje awaria - zadania już wykonane nie zostaną wykonane ponownie, a wykonane zostaną tylko zadania niewykonane wcześniej.
Źródło	Architekt

Identyfikator	D16
Nazwa	Kolejkowanie zadań wykonywanych w tle
Opis	Moduł wykonywana zadań w tle posiada mechanizm kolejkowania.
Uzasadnienie	Pozwala to na zwiększenie niezawodności systemu (np. e-maile do wysłania nie „przepadną" a zostaną zakolejkowane w przypadku awarii *ePoczty*).
Źródło	Architekt

Identyfikator	D17
Nazwa	Priorytety zadań kolejkowanych do wykonania w tle
Opis	Zlecenia do wykonania w tle posiadają priorytety. Mechanizm kolejkowania w pierwszej kolejności wybiera do wykonania zlecenia o najwyższym priorytecie.
Uzasadnienie	Pozwala to na wykonanie najważniejszych zadań w pierwszej kolejności.
Źródło	Architekt

Identyfikator	D18
Nazwa	Identyfikowanie modułów mających wykonać zadanie w tle
Opis	Każde zadanie do wykonania w tle posiada identyfikator modułu, jaki powinien je wykonać.
Uzasadnienie	Zadania do wykonania w tle mają różne rodzaje. Umożliwi to systemowi w łatwy sposób uruchomić kod odpowiedni do wykonania danego rodzaju zadania.
Źródło	Architekt

Identyfikator	D19
Nazwa	Sposób przechowywania danych niezbędnych do wykonania zadania w tle
Opis	Dane niezbędne do wykonania zadania przechowywane są w odpowiedniej kolumnie bazy danych w postaci łańcucha znakowego. Dane te mają postać obiektu zserializowanego przy pomocy technologii *JSON*.
Uzasadnienie	Zadania do wykonania w tle mają różne rodzaje. Umożliwi to systemowi w łatwy sposób uruchomić kod odpowiedni do wykonania danego rodzaju zadania.
Źródło	Architekt

Identyfikator	D20
Nazwa	Zewnętrzny system *BI*
Opis	Do generowania raportów wykorzystany jest system *BI* działający na osobnym serwerze.
Uzasadnienie	Dzięki temu nie trzeba implementować od początku mechanizmu raportowania, a można wykorzystać istniejące już oprogramowanie.
Źródło	Architekt

Identyfikator	D21
Nazwa	Wykonywanie zadań przez osobny proces
Opis	Po dodaniu zlecenia do wykonania w tle, na serwerze uruchamiany jest proces wykonujący zadania.
Uzasadnienie	Umożliwia to natychmiastowe rozpoczęcie wykonywania zadań. Utworzenie procesu pozwoli na przetwarzanie w tle.
Źródło	Architekt

Identyfikator	D22
Nazwa	Maksymalnie 1 proces wykonujący przetwarzanie w tle
Opis	Zadania w tle wykonywane są przez 1 proces na serwerze.
Uzasadnienie	Ułatwia to implementację - można uniknąć problemów związanych ze współbieżnością.
Źródło	Architekt

Identyfikator	D23
Nazwa	Oznaczanie zleceń jako „gotowe do wykonania”
Opis	Rozpoczęcie wykonywania w tle zlecenia może nastąpić tylko, jeżeli jest ono oznaczone jako „gotowe do wykonania”.
Uzasadnienie	Pozwala to zapobiec rozpoczęciu wykonywania zlecenia w momencie, kiedy cały czas do zlecenia dodawane są zadania.
Źródło	Architekt

Identyfikator	D24
Nazwa	Język programowania
Opis	Wykorzystany jest język programowania *PHP*.
Uzasadnienie	Jest to konieczne ze względu na to, że system *Moodle* jest wykonany w tej technologii.
Źródło	Architekt

Identyfikator	D25
Nazwa	*iQuest* jako aplikacja internetowa
Opis	System *iQuest* ma postać aplikacji internetowej.
Uzasadnienie	Dzięki temu po stronie klienta nie trzeba instalować dedykowanej aplikacji, a także nie jest zużywana duża ilość zasobów komputera.
Źródło	Architekt

Identyfikator	D26
Nazwa	Cache'owanie danych z usługi *eDziekanat*
Opis	Moduł *eDziekanat Connector* ma możliwość lokalnego cache'owania danych z usługi *eDziekanat*.
Uzasadnienie	Poprawia to wydajność oraz umożliwia dostęp do danych w przypadku awarii usługi *eDziekanat*.
Źródło	Architekt

Identyfikator	D27
Nazwa	Moduł *eDziekanat Connector*
Opis	Moduł *eDziekanat Connector* służy do pobierania danych z systemu *eDziekanat*.
Uzasadnienie	Wydzielenie osobnego modułu do pobierania danych poprzez usługę *eDziekanat* pozwala na umieszczenie kodu za to odpowiedzialnego „w jednym miejscu". W przypadku zmian w sposobie dostępu do usługi, wystarczy zmodyfikować tylko ten moduł.
Źródło	Architekt

Identyfikator	D28
Nazwa	Moduł *ePoczta Connector*
Opis	Moduł *ePoczta Connector* służy do wysyłania wiadomości e-mail przy pomocy usługi *ePoczta*.
Uzasadnienie	Wydzielenie osobnego modułu do interakcji z usługą *ePoczta* pozwala na umieszczenie kodu za to odpowiedzialnego „w jednym miejscu". W przypadku zmian w sposobie dostępu do usługi, wystarczy zmodyfikować tylko ten moduł. Możliwe jest też łatwe utworzenie „atrapy" modułu przydatnej przy testowaniu.
Źródło	Architekt

Identyfikator	D29
Nazwa	System zarządzania bazą danych
Opis	Przystosowanie systemu do pracy z urządzeniami mobilnymi polega na przygotowaniu odpowiedniej tzw. kompozycji dla systemu *Moodle*.
Uzasadnienie	Jest to łatwy sposób na zrealizowanie wymagania dotyczącego obsługi na urządzeniach przenośnych.
Źródło	Architekt

Identyfikator	D30
Nazwa	Przystosowanie systemu do pracy na urządzeniach mobilnych
Opis	Przystosowanie systemu do pracy z urządzeniami mobilnymi polega na przygotowaniu odpowiedniej tzw. kompozycji dla systemu *Moodle*.
Uzasadnienie	Jest to łatwy sposób na zrealizowanie wymagania dotyczącego obsługi na urządzeniach przenośnych.
Źródło	Architekt

Identyfikator	D31
Nazwa	Przechowywanie informacji o przerwie serwisowej
Opis	Informacja o przerwie serwisowej jest przechowywana w bazie danych w tabeli z ustawieniami systemu *iQuest*.
Uzasadnienie	W ten sposób wtyczki *iQuest Survey Activity Plugin* i *iQuest Course Format* będą mogły w łatwy sposób sprawdzić, czy system nie jest w stanie przerwy serwisowej.
Źródło	Architekt

Identyfikator	D32
Nazwa	Odtwarzanie po awarii
Opis	Odtwarzanie zawartości systemu po awarii jest wykonywane poprzez przywrócenie kopii zapasowej bazy danych.
Uzasadnienie	W ten sposób wtyczki *iQuest Survey Activity Plugin* i *iQuest Course Format* będą mogły w łatwy sposób sprawdzić, czy system nie jest w stanie przerwy serwisowej.
Źródło	Architekt

Identyfikator	D33
Nazwa	Kopie zapasowe bazy danych
Opis	Obowiązek wykonywania kopii zapasowej bazy danych spoczywa na administratorze serwera baz danych.
Uzasadnienie	Dzięki temu nie ma potrzeby implementowania dodatkowej funkcjonalności związanej z wykonywaniem kopii zapasowych. W mechanizm taki wyposażony jest system zarządzania bazą danych.
Źródło	Architekt

Identyfikator	D34
Nazwa	Moduł Logger
Opis	Za obsługę dziennika zdarzeń systemowych odpowiada specjalny moduł - *Logger*.
Uzasadnienie	Wydzielenie do tego celu osobnego modułu pozwala na umieszczenie kodu za to odpowiedzialnego „w jednym miejscu”.
Źródło	Architekt

Identyfikator	D35
Nazwa	Moduł *Background Task Scheduler and Executor*
Opis	Za obsługę zadań wykonywanych w tle odpowiada specjalny moduł - *Background Task Scheduler and Executor*.
Uzasadnienie	Wydzielenie do tego celu osobnego modułu pozwala na umieszczenie kodu za to odpowiedzialnego „w jednym miejscu”.
Źródło	Architekt

Identyfikator	D36
Nazwa	Sposób aktualizacji systemu
Opis	Do aktualizacji systemu wykorzystany jest mechanizm aktualizacji wtyczek platformy *Moodle*.
Uzasadnienie	Dzięki wykorzystaniu mechanizmu obecnego w *Moodle'u* nie ma potrzeby opracowywania własnego sposobu aktualizacji.
Źródło	Architekt

Identyfikator	D37
Nazwa	Testowanie jednostkowe
Opis	Do obsługi testowania jednostkowego wykorzystana jest biblioteka *PHPUnit*.
Uzasadnienie	Biblioteka *PHPUnit* jest dość rozbudowana, a testowanie przy jej pomocy jest dodatkowo wspierane przez najnowszą wersję *Moodle'a*.
Źródło	Architekt

Identyfikator	D38
Nazwa	Dostęp absolwentów do systemu
Opis	Student po staniu się absolwentem ma dostęp do systemu przy pomocy „zwykłego" konta w systemie *Moodle*.
Uzasadnienie	Logowanie absolwentów przy pomocy *eKonta* nie jest na razie możliwe, a więc będą oni musieli korzystać ze „zwykłych" kont.
Źródło	Architekt

Identyfikator	D39
Nazwa	Przygotowanie do wielojęzyczności
Opis	Przygotowanie do wielojęzyczności polega na uwzględnieniu w bazie danych pól przeznaczonych na angielską treść pytań i odpowiedzi w pytaniach wielokrotnego/jednokrotnego wyboru.
Uzasadnienie	Tak proste rozwiązanie jest wystarczające, ponieważ w przewidywalnej przyszłości nie planuje się przygotowywania wielojęzycznych ankiet w językach innych niż polski i angielski.
Źródło	Architekt

Identyfikator	D40
Nazwa	Atrapy
Opis	Kod źródłowy systemu zawiera alternatywne wersje modułów *eDziekanat Connector* oraz *ePoczta Connector*. Nie łączą się one z systemami uczelnianymi, a służą jedynie jako „atrapy" wykorzystywane przy testowaniu.
Uzasadnienie	Takie rozwiązanie sprawia, że znaczną część prac związanych z testowaniem można wykonać bez konieczności połączenia z prawdziwymi usługami zewnętrznymi.
Źródło	Architekt

5.5.3 Zależności między decyzjami

Decyzja	Pozwala	Ogranicza
D01	D03, D04, D05, D06, D07, D09, D10, D30, D36, D37 i D38	D03, D04, D05, D06, D07, D09, D10, D30, D36 i D38
D02	D40	
D13	D14	
D15	D16, D17, D18, D19, D21, D22 oraz D23	
D24		D01
D25	D01	

Ponadto:

- decyzje D04, D05, D06, D07, D27, D28, D34 i D35 są powiązane z decyzją D02
- decyzja D32 jest powiązana z decyzją D33

5.5.4 Alternatywne decyzje

Identyfikator	AD1
Jest alternatywą do	D01
Nazwa	Postać systemu - alternatywa
Opis	System jest samodzielną aplikacją wykonaną w technologii *Java EE*.
Uzasadnienie	Tworzenie aplikacji całkowicie od początku okazałoby się bardziej pracochłonne, a także trudniej byłoby ją zintegrować z innymi aplikacjami używanymi przez studentów (np. *Moodle*).
Źródło	Architekt

Identyfikator	AD2
Jest alternatywą do	D11
Nazwa	Sposób zapamiętywania ustawień systemu *iQuest* - alternatywa
Opis	System zapamiętuje ustawienia bezpośrednio w pliku na serwerze.
Uzasadnienie	Rozwiązanie to wymagałoby wykonywania dodatkowych czynności w celu wykonania lub ewentualnego przywrócenia kopii zapasowej.
Źródło	Architekt

Identyfikator	AD3
Jest alternatywą do	D12
Nazwa	Dziennik zdarzeń - alternatywa
Opis	System w utrzymywania dziennika zdarzeń wykorzystuje istniejący w *Moodle'u* mechanizm logowania zdarzeń
Uzasadnienie	Standardowy mechanizm logowania zdarzeń w *Moodle'u* nie pozwala na przyporządkowanie wpisów do kategorii (np. ostrzeżenie, błąd, itp.), a więc nie byłby w stanie spełnić wymagania pozafunkcjonalnego obejmującego tę kwestię.
Źródło	Architekt

Identyfikator	AD4
Jest alternatywą do	D20
Nazwa	Stworzony od podstaw mechanizm raportowania
Opis	Częścią systemu jest stworzony od podstaw mechanizm definiowania, generowania i przechowywania raportów.
Uzasadnienie	Byłoby to zbyt pracochłonne. Lepiej wykorzystać istniejący system BI.
Źródło	Architekt

Identyfikator	AD5
Jest alternatywą do	D24
Nazwa	Język programowania - alternatywa
Opis	System jest napisany w języku *Java*.
Uzasadnienie	Nie pozwoliłoby to wykorzystanie *Moodle'a* jako platformy.
Źródło	Architekt

Identyfikator	AD6
Jest alternatywą do	D25
Nazwa	Klient systemu *iQuest* jako samodzielna aplikacja
Opis	Użytkownik korzysta z systemu za pośrednictwem specjalnej aplikacji klienckiej.
Uzasadnienie	Konieczność instalowania dodatkowej aplikacji w celu wypełnienia ankiety zniechęciłoby wielu potencjalnych respondentów. Ponadto, należałoby przygotować wersję dla różnych systemów operacyjnych, co wiązałoby się z dodatkowym nakładem pracy.
Źródło	Architekt

Identyfikator	AD7
Jest alternatywą do	D29
Nazwa	System zarządzania bazą danych - alternatywa
Opis	System korzysta z systemu zarządzania bazą danych *MySQL*.
Uzasadnienie	Byłoby to niezgodne z wymaganiami *DRO*.
Źródło	Architekt

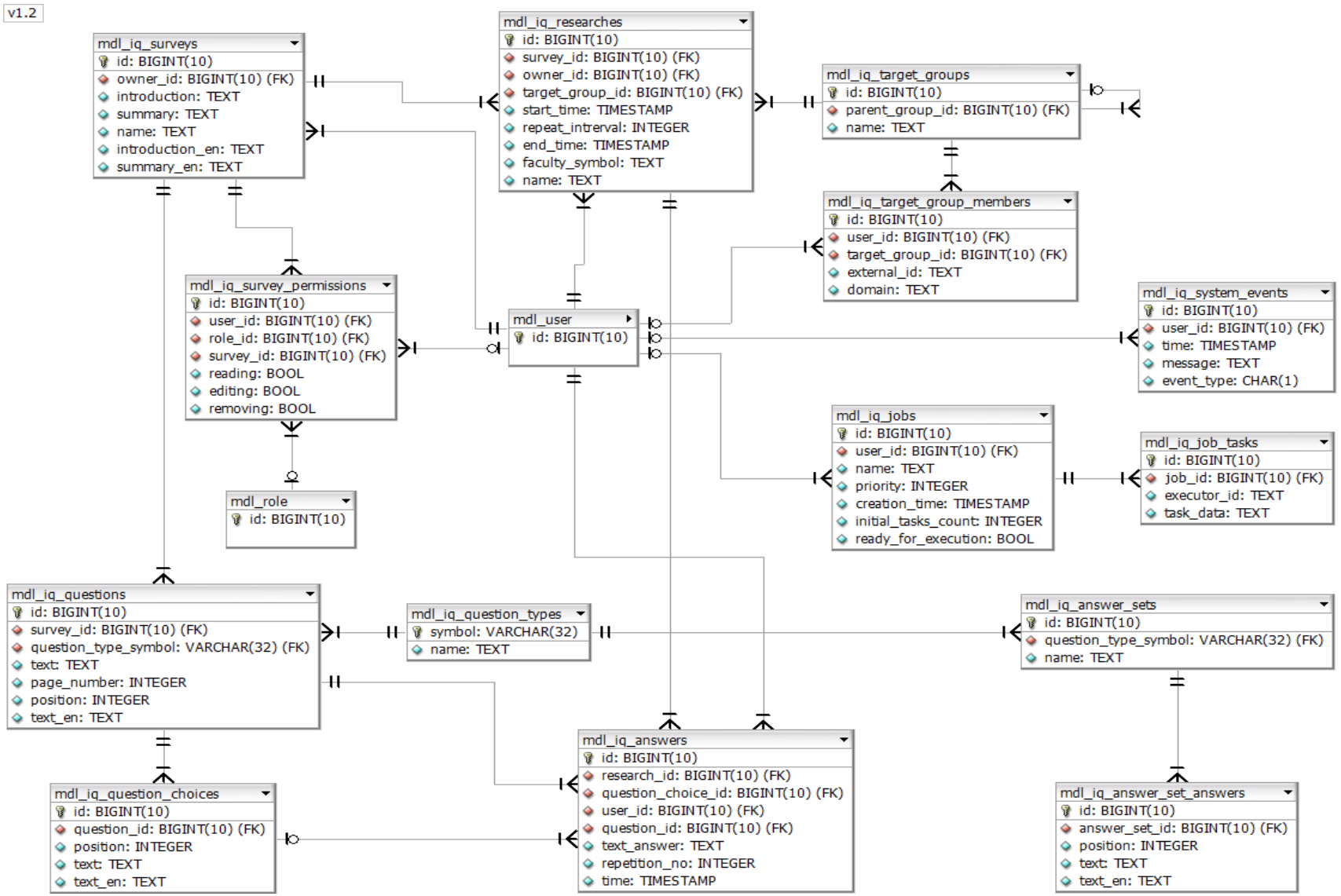
Identyfikator	AD8
Jest alternatywą do	D37
Nazwa	Testowanie jednostkowe - alternatywa
Opis	Do obsługi testowania jednostkowego wykorzystana jest biblioteka *SimpleTest*.
Uzasadnienie	Biblioteka *SimpleTest* jest mało rozbudowana. Bardziej funkcjonalna jest np. biblioteka PHPUnit.
Źródło	Architekt

Identyfikator	AD9
Jest alternatywą do	D38
Nazwa	Dostęp absolwentów do systemu - alternatywa
Opis	Student po staniu się absolwentem ma dostęp do systemu dzięki logowaniu się poprzez specjalne *eKonto* dla absolwentów.
Uzasadnienie	Na Uczelni nie są tworzone specjalne *eKonta* dla absolwentów i nie planuje się, aby w najbliższym czasie (zwłaszcza przed wdrożeniem systemu *iQuest*) to zmienić.
Źródło	Architekt

Identyfikator	AD10
Jest alternatywą do	D39
Nazwa	Przygotowanie do wielojęzyczności - alternatywa
Opis	W bazie danych istnieją tabele przeznaczone na przechowywanie różnych wersji językowych pytań i odpowiedzi.
Uzasadnienie	Jest bardzo mała szansa, że na Uczelni pojawi się potrzeba przeprowadzania ankiet w językach innych niż polski i angielski, więc niepotrzebnie zwiększyłoby to stopień skomplikowania systemu.
Źródło	Architekt

5.6 Schemat bazy danych

Ze względu na obszerność, schemat bazy danych przedstawiono na następnej stronie.



Rysunek 5.5: Diagram Bazy Danych (wyk. zespół zarządzający)

Rozdział 6

Opis implementacji

6.1 Wstęp

Realizacja projektu *iQuest* trwała 5 miesięcy. W tym czasie zrealizowano dwa kolejne wydania.

W pierwszym miesiącu, w trakcie praktyk studenckich odbywanych przez trzech spośród czterech członków zespołu programistów, utworzono pierwszą wtyczkę do platformy *Moodle*. Był to moduł logowania przez *eKonto*. Następne 3 miesiące trwało utworzenie pierwszego wydania, obejmującego podstawowe mechanizmy tworzenia badań i ankiet, oraz ich przeprowadzania. Ostatni miesiąc, przeznaczony na drugie wydanie, zaowocował powstaniem mechanizmów zarządzania grupami docelowymi, tworzenia predefiniowanych zestawów odpowiedzi do pytań oraz wieloma innymi dodatkami, bezpośrednio związanymi z wymaganiami funkcjonalnymi i pozafunkcjonalnymi, wyznaczonymi dla projektu.

W trakcie realizacji, zdarzały się dynamiczne zmiany podejścia do poszczególnych elementów systemu, co skutkowało zatwierdzaniem nowych decyzji projektowych i wymagało przebudowania gotowych już elementów. Dla zespołu programistów, było to uciążliwe i wymagało poświęcenia dodatkowych nakładów czasowych.

W dalszej części niniejszego rozdziału przedstawiono analizę systemu *iQuest* w kontekście implementacji. Opisano napotkane problemy, zastosowane rozwiązania i technologie, wraz z wyjaśnieniem kwestii związanych z implementacją interfejsu i logiki, a także mechanizmów wiążących je ze sobą.

6.2 Napotkane problemy i ich rozwiązania

6.2.1 Wstęp

Problemy oraz ich rozwiązania zostały posortowane chronologicznie, zgodnie z kolejnością, w jakiej pojawiały się w czasie realizacji projektu. W celu ułatwienia ich analizy, wcześniej zamieszczono krótki wstęp teoretyczny opisujący budowę platformy *Moodle*.

6.2.2 Platforma *Moodle*

Po zalogowaniu do systemu użytkownik musi wybrać *kurs*. Jest on największą częścią platformy *Moodle* i przeważnie kojarzony jest z „przedmiotem”. Na kurs składa się kilka, lub kilkanaście *sekcji*. Odpowiadają one najczęściej konkretnym zajęciom, wydarzeniom lub

np. tygodniom. Najmniejszą jednostką w *Moodle* jest *aktywność*, będąca podstawowym typem modułów rozszerzających funkcjonalność platformy. *Aktywnościami* są np. *Fora*, *Głosowania*, czy też *Czat*.

Istnieje również inny typ modułów: *zasoby*. Są to m.in. własne *strony internetowe*, *pliki*, *adresy URL*. Na potrzeby projektu została wyróżniona grupa „materiały". Zalicza się do niej wszystkie moduły inne niż *iQuest*, czyli inne niż badania. Moduły grupowane są w sekcje. Różnica pomiędzy tym, czy użytkownik znajduje się lub nie, w konkretnym elemencie platformy, jak moduł, czy kurs, nazywana jest *kontekstem*. O ułożeniu i wyświetlaniu elementów na stronie decyduje *formater*, definiując w ten sposób interfejs użytkownika.

Konieczność przystosowania interfejsu systemu *iQuest* do opisanej budowy *Moodle*, była dużym wyzwaniem dla zespołu programistów. Problem ten przewijał się przez cały czas implementacji.

6.2.3 Inicjalizacja bazy danych

Moduł *iQuest* do prawidłowego działania wymaga rozszerzenia istniejącej bazy danych platformy *Moodle* o dodatkowe tabele, przechowujące dane niezbędne do spełnienia założonej funkcjonalności.

Do zaimportowania bazy danych przygotowanej przez Architekta, wykorzystano narzędzie wbudowane w platformę Moodle: *XMLDB*. Gwarantuje ono bezobsługową instalację modułu w przyszłości. W trakcie pracy z tym narzędziem, znaleziony został błąd, uniemożliwiający zaimportowanie kluczy obcych do bazy. W efekcie, programiści musieli ręcznie utworzyć wszystkie klucze obce, przewidziane przez Architekta, co wymagało znaczących nakładów czasowych.

6.2.4 Instalacja modułu

Postanowiono, że wraz z instalacją modułu *iSurvey*, zawierającego logikę systemu *iQuest*, powinien automatycznie tworzyć się powiązany z nim kurs. Rozwiązanie to zmniejsza nakład czasu wymagany do przygotowania platformy do użytku, oraz zapobiega pomyłkom związanym z ręcznym tworzeniem i konfiguracją systemu.

Okazało się, że podejście to uniemożliwia instalację modułu jednocześnie z całą platformą *Moodle*, ponieważ w trakcie procesu instalacji platformy *Moodle*, dodatkowe moduły instalowane są przed mechanizmami pozwalającymi na tworzenie *kursu*. Z tego względu, moduł *iSurvey* należy dodawać do wcześniej zainstalowanej platformy.

6.2.5 Formularze

Projektując graficzny interfejs użytkownika, prędzej czy później, pojawia się potrzeba wyboru narzędzia do projektowania formularzy. Rozważano kilka możliwości. Pierwszym, najbardziej naturalnym odruchem, była idea zastosowania czystego języka *HTML*. Pod uwagę brane było także zastosowanie wbudowanych w *Moodle* interfejsów programowania (ang. *Application Programming Interface, API*) – *Form API* oraz *Output API* – jak też użycie zewnętrznych *API*, nie związanych bezpośrednio z platformą.

Zastosowanie *HTML* oraz zewnętrznych *API* zostało odrzucone. Decyzję tę podjęto ze względu na obawę, że pisanie interfejsu w całości od nowa okaże się zbyt pracochłonne. Z uwagi na dość mocno ograniczony czas, nie chciano wyważać otwartych już drzwi, tworząc coś, co już wcześniej zostało przez kogoś zrealizowane, nawet za cenę tego, że interfejs nie wyglądał dokładnie tak, jak go zaprojektowano – na pierwszym miejscu stawiano jego kompletność. Zastosowanie zewnętrznych *API* było natomiast niezgodne z założeniem, nakazującym stosowanie interfejsów *Moodle* wszędzie tam, gdzie to możliwe. Użytkownik powinien mieć uczucie, że programowana wtyczka jest integralną częścią platformy. Co więcej, różnice w stosunku do oryginalnego wyglądu platformy mogłyby sprawić, że interfejs oceniony zostałby jako nieintuicyjny. To zaważyło na decyzji odnośnie zastosowania *Output API* wbudowanego w *Moodle*.

Wyżej wspomniane *API* jest zestawem funkcji, wprowadzonych wraz wersją 2.0 Moodle. Umożliwiają one wstawianie na stronę standardowych elementów formularza, takich jak: etykiety, przyciski, linki, tabele, itp. Niestety, z przyczyn nieznanych dla osób tworzących ten dokument, to doskonale wyposażone i w pełni udokumentowane *API*, zostało usunięte z *Moodle* wraz z aktualizacją do wersji 2.2. Co jeszcze bardziej niezrozumiałe, część elementów można stosować w dalszym ciągu, lecz brak jest dokumentacji, określającej m.in. których z nich to dotyczy.

Ostatecznie, realizacja interfejsu musiała odbyć się z użyciem *Form API*. Ku rozczarowaniu zespołu programistów, ma on znacznie uboższą dokumentację. Zdarza się, że w funkcji są omówione np. tylko trzy pierwsze argumenty, podczas gdy reszta jest pominięta – tak, jakby kompletnie nie istniała. *Form API* różni się też od poprzednich *API* tym, że jest oparte na modelu obiektowym. To sprawia, że zawiera szereg zalet, jak np. fakt posiadania mechanizmu pozwalającego na weryfikację danych wprowadzanych przez użytkownika z użyciem *JavaScript*. Mechanizm ten waliduje dane po stronie klienta, pozwalając na przesłanie do serwera jedynie poprawnych informacji.

Podsumowując, mimo niekompletnej dokumentacji, jako narzędzie implementacji formularzy wybrano *Form API*. Choć zawiera ono sporo zalet, nie wszystkie potrzebne elementy udało się stworzyć korzystając jedynie z niego. Stosowano wówczas język *HTML*.

6.2.6 Role

Jedną z cech projektu, jest podział użytkowników na ankieterów i respondentów. W systemie *Moodle* istnieje mechanizm do zarządzania rolami, który wydawał się adekwatny do użycia w tym przypadku. Rola jest to zbiór *możliwości* (ang. *capability*), które można rozumieć, jako prawa do wykonania, fragmentu kodu określonego przez programistę. Zdecydowano więc o zastosowaniu tego gotowego rozwiązania.

6.2.7 Formater kursu

Jednym z problemów jakie napotkano, była konieczność wyświetlania respondentom i ankieterom tylko określonych modułów. Ankieterzy powinni zobaczyć tylko te badania, które utworzyli, lub które im udostępniono, wraz z innymi aktywnościami i zasobami. Respondenci powinni natomiast zobaczyć tylko te badania, w których mogą wziąć udział, a także materiały, do których pozyskali prawa do ich odczytu.

Do rozwiązania problemu zdecydowano się użyć formatera kursu. Narzędzie to, jako integralna część *Moodle*, wydawało się najlepszym rozwiązaniem spośród dostępnych. W krótkim czasie okazało się jednak, że niekompletność dokumentacji, czy nawet merytorycznych dyskusji na ten temat w Internecie, znacząco utrudnia wykonanie zadania. Całą pracę wejścia, polegającą na poznaniu narzędzia, wykonano studiując kod źródłowy domyślnych formaterów dostępnych w *Moodle*.

Pierwotnie zakładano wyświetlanie użytkownikowi dwóch sekcji. Jednej z odpowiednimi badaniami, drugiej z materiałami. Należało także ograniczyć ankieterowi możliwość dodawania w pierwszej sekcji modułów innych niż *iQuest*, oraz dokładnie odwrotnego działania w drugiej z nich. Okazało się to nieosiągalne bez ingerowania w wewnętrzny kod platformy.

Przyczyną były uaktualnienia zastosowane w *Moodle*. Kod *PHP* wyświetlania typów modułów jest nadpisywany przez *JavaScript*. W ten sposób, z poziomu funkcji *PHP* odpowiedzialnych za wyświetlanie listy modułów w danej sekcji, nie da się kontrolować, które moduły zostaną wyświetlane, a które nie. Mówiąc prościej, programista może jedynie wybrać, jakie moduły będą wyświetlane we wszystkich sekcjach w danym kursie, nie mając wpływu na to, co można wykonywać w każdej sekcji z osobna.

Rozwiązaniem było umieszczenie listy badań oraz listy materiałów w jednej sekcji. Można w niej dodać jakikolwiek moduł. Dopiero przy wyświetlaniu moduły dzielone są na dwie listy: listę badań i listę materiałów. Dzięki temu cel został osiągnięty – użytkownik zobaczy tylko te moduły, które ma prawo wyświetlać. Co więcej, będą one odpowiednio posegregowane, aby użytkownik szybko mógł znaleźć to, czego szuka.

6.2.8 Tworzenie badania

Kolejną trudnością w projekcie było połączenie utworzonej dla systemu *iQuest* wtyczki z platformą *Moodle*. Głównie sprowadzało się to do wykorzystania interfejsu graficznego *Moodle* w sposób niwelujący uczucie zmiany systemu u użytkownika. Zarówno wygląd, jak i sposób wykorzystywania funkcjonalności, powinny być zgodne ze standardem *Moodle*. Dzięki takiemu podejściu, osoba korzystająca wcześniej z platformy, a pragnąca używać wtyczki *iQuest*, nie będzie musiała zmieniać swoich przyzwyczajeń. Co więcej, w projekcie duży nacisk postawiono na zachęcanie respondentów do wypełnienia ankiety, co było dodatkową motywacją do zaprojektowania przyjaznego użytkownikom interfejsu.

Wstępna wersja interfejsu, zaprojektowana przez Architekta, działała wedle następującego schematu: ankieter wyrażał chęć utworzenia nowego badania poprzez kliknięcie odpowiedniego przycisku. Wówczas mógł dodać do badania ankietę z katalogu, ewentualnie utworzyć nową. W kolejnych krokach, użytkownik definiował szczegóły badania, takie jak: nazwa, grupa docelowa, czas rozpoczęcia i zakończenia itp. Niestety, realizacja takiego rozwiązania okazała się niemożliwa.

Problem stwarzało dodawanie ankiety w trakcie procesu tworzenia badania, jeszcze przed jego zakończeniem. Zaczynając generowanie badania od zdefiniowania ankiety, nie można było jej od razu do niego dodać – badanie to bowiem jeszcze nie istniało. W takim wypadku należałoby przechowywać informację, że po utworzeniu badania ma dodać się do niego ankieta[1]. Dodatkowo,

[1]Przykładowo można w tym celu wykorzystać dodatkowy parametr w adresie *URL*, choć stwarzałoby to po-

w Moodle, przy kreowaniu nowego modułu, użytkownikowi wyświetlany jest domyślny formularz, w którym podaje się parametry potrzebne do zbudowania instancji tego modułu. Przyjmując, że badanie jest kojarzone z modułem, nie ma możliwości, aby przed zakończeniem tworzenia badania wstawić wewnątrz dodatkowy formularz.

Z tego względu, zamieniona została kolejność tworzenia badania i ankiety. Najpierw użytkownik tworzy badanie, czyli moduł realizujący ankietę. Dopiero wówczas ma możliwość załączenia do niego ankiety. Podejście to ma kilka zalet: jest to zgodne z procedurami charakterystycznymi dla *Moodle*, a co za tym idzie, bardziej intuicyjne dla użytkownika obeznanego z platformą, a jednocześnie pozwala na łatwe dodanie ankiety do badania. Ponadto ankieter może zrezygnować z komponowania ankiety przy kreowaniu badania, odkładając to, znacznie bardziej czasochłonne, zadanie na później.

6.2.9 Tworzenie ankiety

Przy tworzeniu ankiety pojawił się dość specyficzny problem implementacyjny. Wynikał on z faktu, że ankietę definiować można zarówno z poziomu kursu, jak i z poziomu badania. Powstało pytanie: Jak przetwarzać dane pochodzące z różnych, niezależnych od siebie kontekstów?

Standardowo, we wtyczkach *Moodle*, elementy odpowiedzialne za wyświetlanie informacji na ekranie znajdują się w pliku *view.php*. Pojawił się pomysł, aby rozszerzyć strukturę o dwa dodatkowe pliki: *mod.php* oraz *course.php*. Do *mod.php* trafiać miały dane z kontekstu modułu. Drugi plik zajmować miałby się przetwarzaniem danych z kontekstu kursu. Taki podział gwarantował większy porządek w kodzie źródłowym. Porządek był ważny, ponieważ *Moodle* nie jest zgodny ze wzorcem Model-View-Controller. W związku z tym istotne jest aby efektywnie zarządzać kodem źródłowym, żeby mała jego zmiana nie wymagała zmiany wielu elementów.

Niestety wprowadzone zmiany okazały się niewystarczające. Występowało niepotrzebne powielanie kodu. Wydzielono jeszcze jeden plik, w którym przetwarzano dane otrzymane z formularzy i zapisywano je do bazy danych. Później zwracano sterowanie do wyżej wspomnianych plików, w zależności od kontekstu.

Dzięki utworzeniu trzech dodatkowych plików, kod źródłowy stał się bardziej przejrzysty. Wartość takiego rozwiązania można zauważyć dopiero, gdy zachodzi konieczność znalezienia błędu lub wprowadzenia modyfikacji do kodu. Przy dobrym zarządzaniu kodem mała zmiana wymaga nieznacznych tylko poprawek.

6.2.10 Hierarchia CSS

Na wielu poziomach serwisu borykano się z problemem hierarchii plików *CSS*. Twórcy platformy *Moodle* po, jak zapewniają, gruntownym przemyśleniu sprawy i rozważeniu wszystkich możliwości, ustalili następującą hierarchię kaskadowych arkuszy stylów:

- Najważniejsze są pliki umieszczone w katalogu *theme*, odnoszące się do całej platformy.
- Następnie uwzględniane są reguły z pliku *styles.css*, umieszczonego w katalogu konkretnej wtyczki.

tencjalny problem dotyczący kwestii liczby przekierowań, przez które musiałby on być przekazywany.

Główną wadą tego podejścia jest fakt, że nie można we wtyczce nadpisać właściwości, która została zdefiniowana w katalogu *theme*. Aby zmienić choćby jedną właściwość z tego katalogu należy utworzyć nowy *wygląd*, kopiując oryginalny i zmieniając tę jedną właściwość. Następnie administrator platformy musi ustawić ten wygląd w swoim systemie (co wiąże się z dodatkową operacją, jeśli poprzedni wygląd był wyglądem domyślnym). Ostatecznie jednak, problem ten udało się rozwiązać, stosując w tym celu atrybut *!important*.

6.2.11 Testy jednostkowe i akceptacyjne

Realizacja wszystkich testów została pierwotnie powierzona jednemu z członków zespołu programistów. Ze względu na brak precyzyjnej architektury logiki we wczesnej fazie projektu, początkowe utrzymanie testów okazało się być bardzo czasochłonne.

Przyczyną takiego stanu był także stały rozwój systemu. Realizowany bez zastosowania metody rozwoju w oparciu o testowanie, *iQuest* ze znaczną szybkością ewoluował niezależnie od testów. Nowe parametry, nowe wartości wyjściowe oraz zmiana dostępności poszczególnych funkcji sprawiały, że utrzymywanie testów zajmowało nawet kilkudziesięciokrotnie więcej czasu, niż ich utworzenie od nowa.

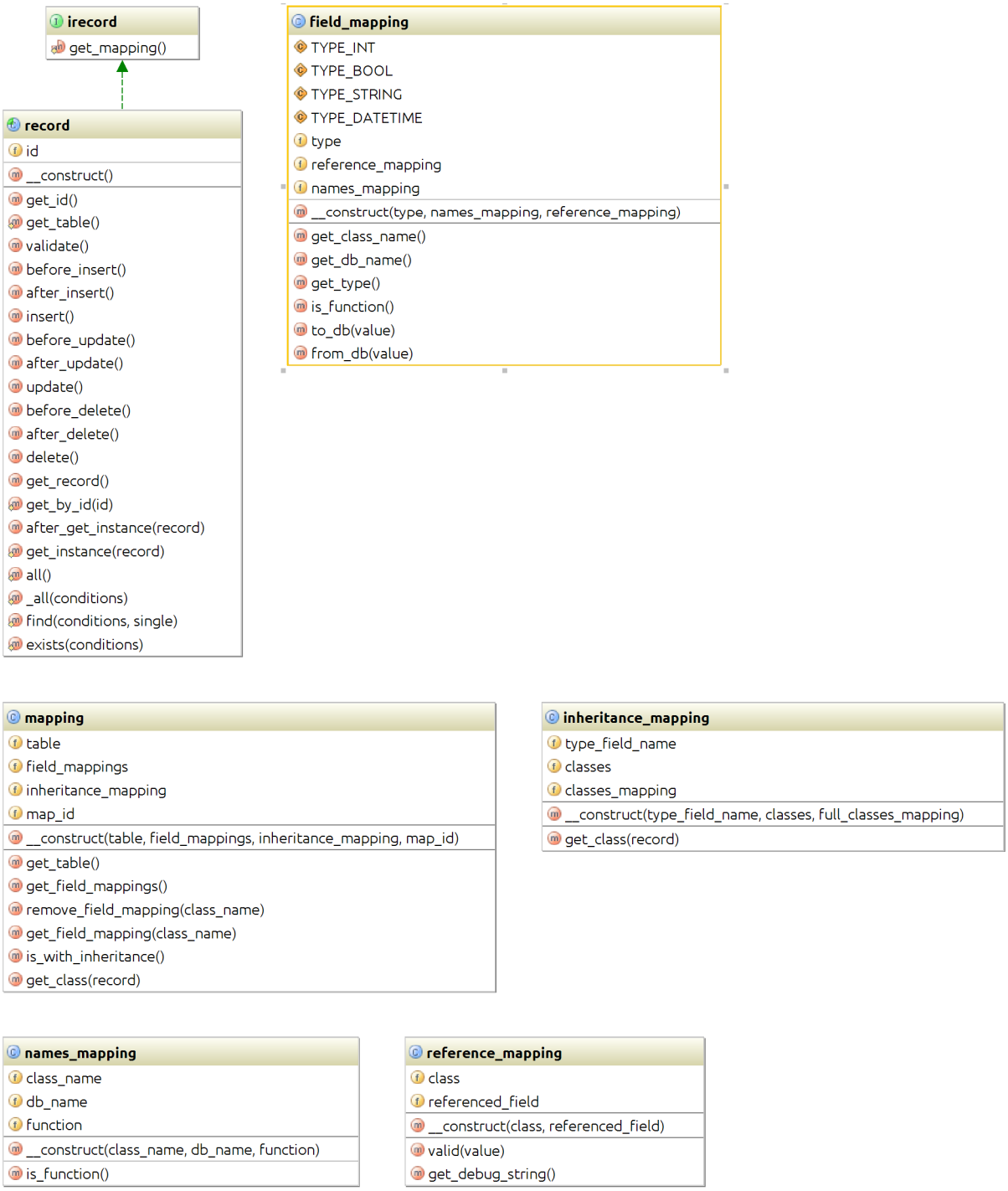
Problem ten występował jednak jedynie w trakcie pierwszego wydania projektu. Przy drugim wydaniu, działalność związaną z testami jednostkowymi w całości przejął programista logiki aplikacji, co znacząco zmniejszyło czasochłonność ich realizacji.

Większy problem dotyczył testów akceptacyjnych. Już na początku realizacji projektu, wyszło na jaw, że eksport z *Selenium IDE* (standard *HTML*) do *Eclipse IDE* (standard *Java*) nie jest zadaniem prostym. Ze względu na jego czasochłonność, w szczególności przy utrzymywaniu testów, zdecydowano o pominięciu tego kroku.

6.2.12 Mapowanie obiektowo-relacyjne

Mapowanie obiektowo-relacyjne pozwala uprościć operacje na danych przechowywanych w bazie danych poprzez udostępnienie ich programiście w postaci obiektowej. System *iQuest* operuje na klasach takich jak: ankieta, badanie, grupa docelowa, członek grupy docelowej, uprawnienie dostępu, pytanie (i potomne), odpowiedź, zadanie, praca w tle, etc. Początkowo architekt stworzył diagram klas, na którym każda klasa miała wyróżnione publiczne metody *insert, update, delete*. Niestety, takie rozwiązanie spowodowało powielenie dużej ilości kodu związanego z interakcją z bazą danych. W ramach refaktoryzacji podjęto się zadania stworzenia klas, które wzorem nowoczesnych systemów *ORM*, uproszczą projektowanie nowych klas reprezentujących dane. Ze względu na silną integrację istniejącego już kodu z mechanizmami *Moodle*, w grę nie wchodziły gotowe rozwiązania. Autorskie rozwiązanie korzysta z *Moodle Data manipulation API* oraz mechanizmu refleksji języka *PHP*, by pozwolić programiście korzystającemu z tego rozwiązania na proste pobieranie i manipulację obiektami przechowywanymi w bazie danych. Diagram UML przedstawia się następująco:

Rysunek 6.1: iQuest ORM (wyk. Łukasz Wieczorek)



Klasa danych dziedziczy po klasie *record* oraz implementuje statyczną metodę *get_mapping* interfejsu *irecord*, by uzyskać dostęp do metod komunikacji z bazą danych. Metoda *get_mapping* pozwala zdefiniować mapowanie danej klasy na odpowiednią relację w bazie danych. Należy przy tym podać nazwę tabeli, mapowanie dla pól klasy składające się z mapowania nazw (klasa *names_mapping*), tj. nazwy w klasie i bazie danych oraz typu, który zadecyduje o metodzie pobrania/zapisania danej (typem może być także klasa potomna klasy *record*). Dodatkowo można uwzględnić istnienie kluczy obcych, których poprawność będzie sprawdzana, jeżeli utworzymy obiekt klasy *reference_mapping*. W przypadku dziedziczenia wystarczy zdefiniować przy mapowaniu sposób jego obsługi (m.in. jakie pole określa typ klasy). Najważniejszy kod znajduje się w metodzie *get_instance*, która pobiera konstruktor danej klasy, poprzez refleksję tworzy obiekt i na podstawie pobranych z bazy danych informacji, ustawia resztę pól. Metody *insert, update, delete* pobierają reprezentację obiektu oczekiwanego przez metody *Data manipulation API* oraz wykonują żądane operacje.

Zastosowane rozwiązanie znacząco poprawiło czytelność kodu poprzez zastosowanie zasady DRY (ang. *Don't repeat yourself*). Projektowano je, mając na uwadze rozwiązanie, z którym programista logiki miał już wcześniej styczność, tj. implementację *ActiveRecord* z *Ruby on Rails*. W trakcie pracy nad projektem doceniono stosowanie konwencji nazewniczych, których obecność znacząco upraszcza projektowanie klas mapujących dane.

6.2.13 Inwencja programistów

W trakcie rozwoju oprogramowania pojawiło się kilka niejasności, które należało rozwiązać. Wykazano również inicjatywę i zaproponowano rozwiązania, które stały się ostatecznie częścią projektu.

Pierwszą ideą było zagospodarowanie przestrzeni w widoku badania. Po utworzeniu badania i dodaniu do niego ankiety, ankieterowi ukazuje się widok badania. Architekt nie zaproponował jak ma on wyglądać. Dał tylko pewne wskazówki. Zaznaczył, że z tego widoku, ankieter ma mieć możliwość usunięcia ankiety z badania oraz edytowania jej. Żeby spełnić wymagania, na stronie wystarczyło pokazać odnośniki: „edytuj" i „usuń z badania". Praktycznie cała strona pozostawała pusta. Sytuacja taka jest niedopuszczalna, gdyż zawsze istnieją przydatne informacje, które można w takim miejscu wyświetlić. Ustalono, że najbardziej naturalnym będzie zamieszczenie w tym miejscu podstawowych statystyk badania.

W pierwszej wersji zaimplementowano tylko proste parametry, takie jak: ile czasu zostało do zakończenia badania, ile osób liczy grupa docelowa oraz ile osób już odpowiedziało. Wraz z rozwojem systemu, dodano kolejną tabelę z danymi. Wyświetla się ona, gdy choć jedna osoba odpowie na któreś pytanie. Można na jej podstawie przeanalizować, jak kształtowały się odpowiedzi w pytaniach zamkniętych. Nie zdecydowano się wyświetlać odpowiedzi na pytania otwarte, ze względu na ich różnorodność – negatywnie wpływałoby to na czytelność strony. Ideą tabeli było pokazanie skróconych informacji o badaniu. Dokładny, rozbudowany raport, można wygenerować z użyciem systemu *JasperReport*.

Pozyskanie i podliczenie odpowiedzi dla danego pytania wiązało się z zastosowaniem odpowiedniego algorytmu. Teoretycznie najprostszym rozwiązaniem byłoby sprawdzanie liczby krotek związanych z danym pytaniem w tabeli *answers*. To rozwiązanie jest jednak nieoptymalne,

gdyż wiąże się z wielokrotnymi odwołaniami do bazy danych. Lepszym wyborem było użycie wbudowanych mechanizmów systemu zarządzania bazą danych w celu optymalizacji odwołania do tablic. Przy użyciu wyrażenia „GROUP BY" opracowano zapytanie, które od razu zwraca liczbę odpowiedzi respondentów w dany sposób, co pozytywnie wpłynęło na szybkość działania algorytmu.

Kolejna kwestia dotyczy odnośników, zwiększających intuicyjność pracy z systemem. Zarówno w katalogu, jak i widoku badania, umieszczono przycisk „pokaż". Służy on do wyświetlenia ankiety w taki sam sposób, w jaki widzi ją respondent. Dzięki temu, że ankieter może zobaczyć układ pytań, łatwiej mu zdecydować np. o podziale na strony. W widoku katalogu pojawił się także przycisk pozwalający na dodanie nowej ankiety. Znajduje się on zarówno w dolnej, jak i górnej części tabeli katalogu, co zwalnia użytkownika z konieczności mozolnego przewijania strony.

W założeniach projektu ustalono, że raz udzielona odpowiedź na pytanie jest nieedytowalna. Z tego względu, dodano udoskonalenie, które polega na tym, że respondent nie musi od razu wypełnić całej ankiety. Może to robić stopniowo – z każdym kolejnym razem zostaną mu jednak wyświetlone tylko te pytania, na które jeszcze nie udzielił odpowiedzi. Pozwala to także uniknąć sytuacji, w której respondent przeoczy jakieś pytania. Jeśli respondent nie wypełni całej ankiety, to badanie nie zniknie z widoku kursu. Dopiero po wypełnieniu całej ankiety, badanie już więcej nie pojawi się w kursie.

6.3 Użyte technologie

6.3.1 *Moodle*

Moodle (ang. *Modular Object-Oriented Dynamic Learning Environment*) – stanowi podstawę systemu *iQuest*. Jest to popularna (ponad 63 miliony użytkowników) platforma e-learningowa o otwartym kodzie źródłowym, napisana w języku *PHP*. Wyboru dokonano ze względu na kilka czynników:

- Propozycję Kierownika Projektu oraz następującą po niej decyzję Architekta, wynikającą z faktu, iż *Moodle* posiada już implementację wielu wymaganych w *iQuest* mechanizmów, jak np. konta użytkowników, system ról i uprawnień.
- Oczekiwania Klienta, wynikające z popularności platformy *Moodle* wśród systemów uczelnianych.
- Modułowość *Moodle*, umożliwiającą pisanie rozszerzeń.

6.3.2 *PHP*

PHP – platforma *Moodle* jest napisana właśnie w tym języku programowania. Z tego względu, jest to technologia zastosowana w większości rozszerzeń utworzonych przez zespół *iQuest*, korzystających z interfejsów programowania aplikacji tej platformy. Ponadto *PHP* jest jednym z najpopularniejszych języków programowania aplikacji internetowych, posiada doskonałą dokumentację[2] oraz jest wciąż rozwijany.

[2] [7]

Dodatkowo, język *PHP* jest dość przyjazny dla programisty, gdy chodzi o komunikację z bazą danych. Przy realizacji zadań z tym związanych, odnoszono się zarówno do wspomnianej wyżej dokumentacji, jak też do literatury fachowej[3].

6.3.3 *PHPUnit*

Ze względu na fakt, iż programiści *Moodle'a* wykonują testy jednostkowe kodu wykorzystując do tego celu *PHPUnit*, zdecydowano się skorzystać z przygotowanego przez nich oprogramowania. *Moodle* udostępnia dwie klasy do testowania – *basic_testcase* i *advanced_testcase*, przy czym druga wymieniona służy do testów, które wchodzą w interakcję z bazą danych. Korzystanie z tych klas dodatkowo uprasza fakt istnienia świetnej dokumentacji technicznej[4].

6.3.4 *Selenium*

Selenium – szybko rozwijający się zestaw narzędzi do testów akceptacyjnych. Był to naturalny wybór zwłaszcza, że zostało ono przybliżone programistom na zajęciach z Inżynierii Oprogramowania w trakcie toku studiów. *Selenium* składa się m.in. z następującego oprogramowania[5]:

- *Selenium IDE* – zintegrowane środowisko programistyczne dla skryptów *Selenium* – zaimplementowane jako rozszerzenie dla przeglądarki internetowej *Firefox*. Pozwala na: nagrywanie i odtwarzanie sekwencji kroków, wykonywanych podczas pracy z przeglądarką, eksport skryptów do kodu języków programowania (np. *Java*).
- *Selenium Client Drivers* (*Java*) – sterownik klienta dla języka *Java*, pozwalający na wykonywanie skryptów *Selenium* z poziomu języka *Java*,
- *HtmlUnit Driver* – Implementacja klasy *WebDriver*, która emuluje zachowanie przeglądarki. Pozwala na uruchamianie skryptów *Selenium* bez korzystania z przeglądarki internetowej.

6.3.5 *PostgreSQL*

System zarządzania bazą danych *PostgreSQL* został wybrany ze względu na wymaganie pozafunkcjonalne – pracownicy DRO korzystają z tej właśnie bazy danych. Jest to baza danych o otwartym kodzie źródłowym, zgodna ze standardami, ciągle rozwijana, wysoce konfigurowalna.

6.3.6 *Eclipse IDE*

Wybór *Eclipse IDE* jako stosowanego dla projektu *iQuest* zintegrowanego środowiska programistycznego wynika z faktu, iż oprogramowanie to jest dostępne za darmo. Dodatkową zaletą *Eclipse* jest modularność tego rozwiązania, dzięki czemu dostępny jest w nim dodatek *PHP Development Tools*, upraszczający pracę z technologią *PHP*. Udostępnia m.in. narzędzia do analizy poprawności składniowej pisanego kodu, formatery kodu, wyszukiwanie fraz w wielu plikach, kontekstowe podpowiedzi i nawigację.

6.3.7 *SVN*

Subversion został wybrany jako podstawowy system kontroli wersji ze względu na wymagania pozafunkcjonalne. Zespół eksploatacji, który docelowo przejmie zarządzanie artefaktami

[3] m.in. „PHP i MySQL - Księga przykładów" autorstwa E. Quigley oraz M. Gargenta[11]
[4] [8]
[5] [9]

związanymi z projektem, wykorzystuje właśnie *SVN*. Główne funkcjonalności tego systemu to: atomowe publikowanie zmian, historia operacji na plikach (zmiana nazwy, skopiowanie, przeniesienie, modyfikacja, usunięcie), wersjonowanie plików i folderów, łatwy dostęp do informacji o zmianach.

6.3.8 *Redmine*

Systemu zarządzania projektami *Redmine* wykorzystywany był od samego początku istnienia projektu. Jest to narzędzie bardzo przydatne w wymianie informacji pomiędzy członkami zespołu, integrujące się m.in. z repozytorium kodu, bazą wiedzy o projekcie, listą zagadnień, forum. Technologia ta została narzucona, ze względu na sposób organizacji pracy w *Software Development Studio* na Politechnice Poznańskiej.

6.3.9 *JasperReports*

Ze względu na wymagania pozafunkcjonalne, zdecydowano się skorzystać z mechanizmów raportowania oferowanych przez *JasperReports*. Jest to najbardziej popularny silnik raportowania o otwartym kodzie źródłowym (wersja *Community*). Pozwala na generację raportów, których treść jest określona z dokładnością co do piksela. Generowane raporty można eksportować do popularnych formatów dokumentów, np. *HTML*, *PDF*, *Excel*, *Word*.

6.3.10 *JetBrains PhpStorm*

PhpStorm jest komercyjnym *IDE* dla języka *PHP* tworzonym przez firmę *JetBrains*, dostępnym dla studentów na licencji edukacyjnej. Dla celów niniejszej pracy dyplomowej, wykorzystano funkcjonalność tworzenia diagramów UML z kodu źródłowego. Wygenerowane w trybie *Organic* diagramy można zobaczyć w dalszej części pracy (schematy 6.2., 6.3., 7.1., 7.2., zamieszczone w rozdziałach 6. i 7.).

6.3.11 *HTML* oraz *CSS*

Po stronie użytkownika, system *iQuest* prezentowany jest za pośrednictwem interpretowanego przez jego przeglądarkę internetową kodu w języku *HTML*. Jego wygląd, wraz z umiejscowieniem elementów, określa natomiast arkusz stylów *CSS*. Realizując zadania związane z tymi technologiami, odnoszono się do profesjonalnej literatury branżowej[6].

6.3.12 *JavaScript*

Formularze wymagające częstej interakcji z klientem, np. formularz umożliwiający tworzenie nowej ankiety oraz funkcje związane z walidacją pól uzupełnianych przez klienta zostały napisane w *JavaScript*. Obsługa strony po stronie klienta jest dla użytkownika znacznie wygodniejsza, gdyż nie wymaga częstego przeładowywania całej strony. Dodatkowo, ogranicza to obciążenie łącza zarówno w lokalizacji serwera, jak i klienta, co jest korzystne dla obu stron.

6.4 Ogólna struktura projektu

Zgodnie z ideą trójwarstwowej architektury, opisanej w rozdziale 5., system *iQuest* podzielono na trzy warstwy, w tym: prezentacji i logiki biznesowej. Obie z nich są ze sobą ściśle powiązane. Szczegóły z tym związane przedstawiają rozdziały 6.5-6.7.

[6] M.in. „Wstępu do CSS3 i HTML5" autorstwa Bartosza Danowskiego[10]

6.5 Interfejs

W trakcie projektowania graficznego interfejsu użytkownika, głównym problemem okazał się wybór odpowiedniego narzędzia. Celem, jaki postawiono, była maksymalna zgodność tworzonych elementów z różnymi wersjami *Moodle* – zarówno wcześniejszymi, jak i późniejszymi. Wymagano stosowania gotowych interfejsów programowania aplikacji (*API*) dostarczonych przez *Moodle*, m.in. *Form API*. Wszystkie interfejsy zostały napisane przy użyciu języka *PHP* (są więc wykonywane po stronie serwera). Konieczne okazało się też wykonanie niektórych skryptów po stronie klienta. Z tego względu w projekcie wykorzystano również język skryptowy *JavaScript*.

6.5.1 Bezpieczeństwo

Potencjalnie, z systemu może korzystać wielu użytkowników, zarówno studentów jak i pracowników. Każda z tych osób może być uprawniona do wykonywania innych czynności w systemie, niezależnie od siebie. Wiąże się to z jednej strony z uwierzytelnianiem użytkowników, z drugiej – z ich autoryzacją. Odbywa się to za pomocą wbudowanego w *Moodle* mechanizmu ról.

Zapewnienie bezpieczeństwa w systemie wymagane jest, aby użytkownik przez pomyłkę nie wykonał czynności, do których nie został uprawniony. Sprowadza się to do ograniczenia mu dostępnych opcji do tych, których może używać. W efekcie, użytkownik może zobaczyć i użyć jedynie tych odnośników, które prowadzą do zasobów, do których posiada uprawnienia.

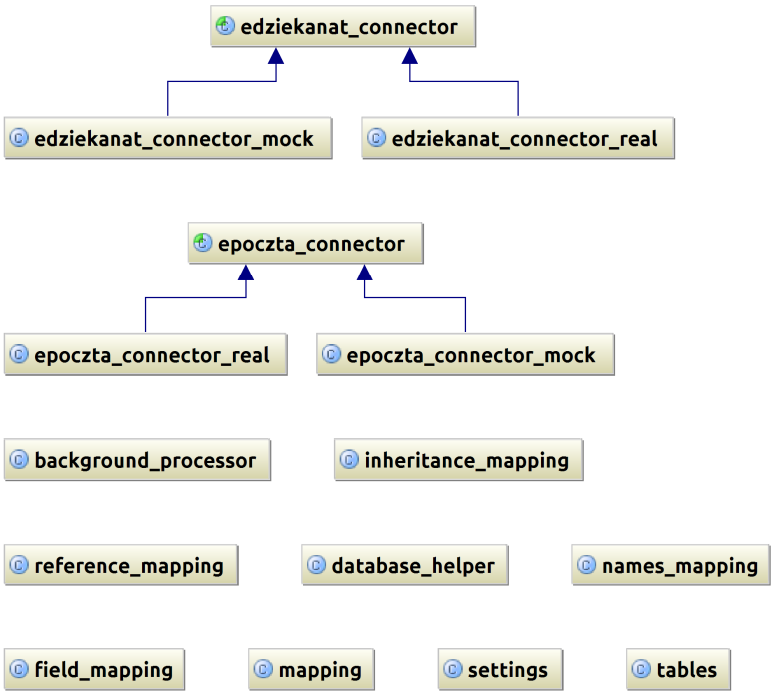
Zabezpieczenie nie polega jedynie na ograniczeniu dostępności opcji. Należy założyć, że użytkownik może przypadkowo, lub z intencją, podjąć próbę pozyskania zasobów bez autoryzacji. Z tego względu zapewnienie bezpieczeństwa wiąże się także z odmową dostępu do zabronionych zasobów. W tym celu sprawdza się dane, które przychodzą do serwera, m.in. tworząc zmienne wiązane w zapytaniach *SQL*, co pozwala na obronę przed atakami typu *SQL-injection*. Ostatecznie, bezpośrednio przed wyświetleniem treści użytkownikowi, sprawdzane jest, czy jest on uprawniony do ich odczytu. Dzięki temu, niepowołany użytkownik nie uzyska dostępu do niedozwolonych treści, nawet przez wykorzystanie adresu prowadzącego bezpośrednio do żądanego zasobu.

6.6 Logika (back-end)

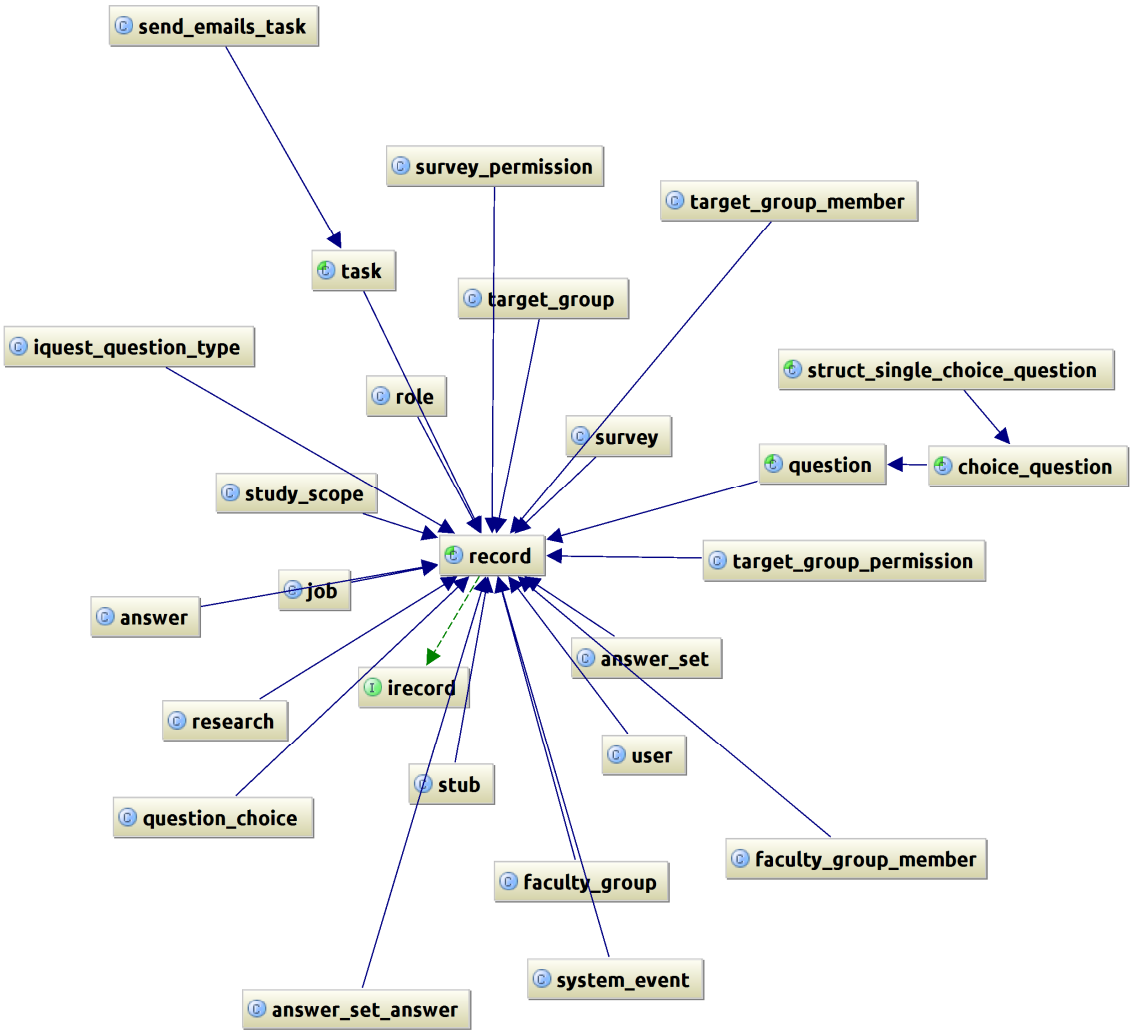
System *iQuest* do prawidłowej pracy wymagał zaprogramowania odpowiedniej logiki biznesowej, rozwiązującej stawiane przed nim zadania. Najważniejszym zagadnieniem w kategorii logiki systemu jest interakcja z bazą danych. Poza nią, system posiada: procesor zadań wykonywanych w tle, oparty na *CRON*; moduły odpowiadające za komunikację z systemami uczelnianymi (m.in. *ePoczta* i *eDziekanat*); moduł logowania zdarzeń. W trakcie implementacji, zdecydowano się nie tworzyć osobnego mechanizmu do przechowywania ustawień w bazie danych, wykorzystując do tego celu funkcjonalność dostępną już w *Moodle*.

Jedno z wymagań pozafunkcjonalnych dotyczyło zastosowania bazy danych *PostgreSQL*. Platforma *Moodle* korzysta z mechanizmu *XMLDB*, co pozwala na uniknięcie wielu problemów pojawiających się przy migracjach pomiędzy różnymi systemami baz danych. Niestety ceną zastosowania tego mechanizmu jest konieczność pracy z *API* dostarczanym wraz z platformą *Moodle – Data manipulation API*. Na diagramach 6.2. oraz 6.3. przedstawiających wyżej opisaną

logikę systemu *iQuest*, znajdują się także klasy przechowujące stałe: *tables* oraz *settings*.



Rysunek 6.2: Struktura back-endu (1) (wyk. Łukasz Wieczorek)



Rysunek 6.3: Struktura back-endu (2) (wyk. Łukasz Wieczorek)

6.6.1 Raporty

Raporty wykonano z użyciem platformy *JasperReports*, wykorzystując następujące produkty firmy *Jaspersoft*:

JasperReports Server (wersja 5.0)
: Serwer usług raportowania, na którym przechowywane są przygotowane przez zespół artefakty, służące umożliwieniu generacji raportów osobom dysponującym odpowiednimi uprawnieniami. Wykorzystano następujące funkcjonalności: definiowania źródła danych, raportu, ładowania plików z projektem raportu, zasobami oraz z generacji raportu.

Jaspersoft Studio (wersja 1.3.2)
: Oparte na Eclipse narzędzie, służące projektowaniu raportów. Z jego użyciem przygotowano projekty raportów w formacie *JRXML*.

Kody źródłowe pakietu *JasperReports* są pisane w języku *Java*. Źródłem danych dla raportu, jest przygotowana przez zespół implementacja interfejsu *ReportDataSourceService* z *API JasperServer*. Źródło danych łączy się z usługami zdalnymi udostępnianymi przez wskazaną instancję systemu *iQuest*, z których otrzymuje informacje o przeprowadzanych badaniach, korzystając do tego celu z protokołu SOAP. Struktura raportu zależy od typu pytania (otwarte/ zamknięte). W przypadku pytań otwartych, prezentowana jest lista odpowiedzi. Dla pytań zamkniętych, na podstawie pobranych danych, generowane są statystyki, przekazywane następnie do podraportu w postaci obiektu klasy *JRBeanCollectionDataSource*. W generacji statystyk z danych badań wykorzystano bibliotekę *JoSQL*.

Definicja projektu raportu składa się z czterech plików, odpowiadających trzem poziomom:

researches.jrxml – raport główny dla badań.

questions.jrxml – podraport dla pytań.

answers_closed.jrxml – podraport odpowiedzi zamkniętych.

answers_open.jrxml – podraport dla odpowiedzi otwartych.

Do generacji namiastek obiektów zdalnych (ang. *stub*) wykorzystano *Apache Axis*. Wygenerowane klasy dostosowano tak, by akceptowały obiekty z zadanej instancji *iQuest* oraz dla zmiennej przestrzeni nazw (ang. *namespace*).

W trakcie generacji *stub'ów* okazało się, iż definicja usług dla protokołu *SOAP* w języku *WSDL* (ang. *Web Service Description Language*) generowana przez *Moodle* jest niepoprawna. Skorzystano zatem z poprawionej wersji z zewnętrznego źródła[7].

Dostęp do usług zdalnych definiowanych w *Moodle* zabezpieczono korzystając z mechanizmu generacji tokenu dla wybranego użytkownika. Użytkownik, który z poziomu serwera *Jasper Server* zamierza wygenerować raport, musi znać adres systemu oraz posiadać token dostępu do usługi.

[7] [2]

6.6.2 Moduły uwierzytelniania

Korzystając z mechanizmów rozszerzeń *Moodle* zaimplementowano dwa moduły uwierzytelniania, tj.:

eKontoAuthenticationPlugin – integruje logowanie przez *eKonto* z systemem *iQuest*,

emailgraduate – pozwala absolwentom uczelni na rejestrację z użyciem adresu e-mail.

W celu spełnienia wymagań DRO odnośnie wygaszania sesji użytkownika *eKonto* po zadanym czasie (np. 15 minut), zmodyfikowano pliki źródłowe *Moodle* – dla kodu odnoszącego się do sesji użytkownika *Moodle* nie została przewidziana możliwość rejestracji rozszerzeń. Relacja *user* została rozszerzona o opcjonalne pola związane z *eKontem*, a jako że kod odpowiedzialny za manipulację schematem bazy danych nie jest wykonywany podczas instalacji modułu uwierzytelniania, umieszczono go w osobnym module (*ekontodb*). *eKontoAuthenticationPlugin* może być instalowany bez konieczności instalacji *iSurvey*. Podczas jego implementacji korzystano z dokumentu udostępnionego przez DRO[8].

Podczas rejestracji z użyciem modułów systemu *iQuest* niezwiązanych z *eKontem*, użytkownik jest przydzielany do grupy docelowej „Absolwenci", nadawana jest mu też rola respondenta w kontekście *kursu iQuest*. Utworzenie modułu wiązało się się z przygotowaniem klasy dziedziczącej z *auth_plugin_base*, formularza ustawień, pliku lokalizacji oraz wersji.

6.6.3 Moduły dla serwisów zewnętrznych

ePocztaConnector – służy do wysyłania e-maili z serwera Politechniki Poznańskiej,

eDziekanatConnector – pobiera i aktualizuje lokalne informacje o grupach dziekańskich, zakresach tematycznych tychże grup oraz ich studentach.

Dział Rozwoju Oprogramowania udostępnia klienty *eUsług* dla różnych języków, w tym dla *PHP*. Komunikacja z usługami zdalnymi uczelni odbywa się poprzez protokół *SOAP*[9]. Wyżej wymienione moduły zaimplementowano z wykorzystaniem fabryki obiektów, która w zależności od trybu (testowy/produkcyjny) zwraca obiekt odpowiedniej klasy. Zadania związane z oba modułami są zlecane procesorowi zadań w tle.

6.7 Powiązanie logiki z interfejsem

Moodle jako paradygmat programowania stosuje podejście proceduralne, natomiast logika zaprogramowanej wtyczki – podejście obiektowe. Zaprojektowano więc mechanizm łączący te dwa sposoby programowania.

Mechanizm łączący stosuje podejście proceduralne. Zaimplementowano szereg dodatkowych funkcji, które operują na danych zwracanych przez formularze. Zamiast bezpośredniego zapisu do bazy danych, tworzone są najpierw obiekty, które dopiero później zapisywane są do bazy. Cały proces odbywa się po stronie serwera i jest zapisany w języku *PHP*.

Mimo większej złożoności, zastosowanie tej metody pozwoliło zmaksymalizować część projektu wykonaną z użyciem programowania obiektowego, pozwalającego na lepszą organizację kodu, a co za tym idzie, także szybszego wykrycie ewentualnych błędów oraz minimalizację powielania kodu. Oczywiście są to tylko niektóre z zalet programowania obiektowego.

[8] „Centralne uwierzytelnianie i wymiana danych. Wersja 1.2 (2010.07.06)"[14]
[9] [15]

6.7.1 Instalacja *iQuest*

Instalacja systemu realizowana jest trójstopniowo. Pierwszym krokiem jest umieszczenie na serwerze docelowym plików platformy *Moodle* i ich instalacja oraz wstępna konfiguracja. W tak przygotowanym środowisku, umieszczane są następnie pliki systemu *iQuest*, które – dzięki wbudowanym mechanizmom platformy *Moodle* – administrator może w bardzo prosty sposób zainstalować i aktywować. Aktualizacja systemu przebiega bardzo podobnie. Polega jedynie na przeprowadzeniu standardowego dla *Moodle* procesu aktualizacji, poprzedzonego zmianą odpowiednich plików na serwerze, w ramach którego działa system.

Rozdział 7

Testy i weryfikacja jakości oprogramowania

7.1 Wstęp

Testy i weryfikacja jakości oprogramowania realizowana była na trzech poziomach: testów jednostkowych (dla logiki) oraz automatycznych i manualnych testów akceptacyjnych. Te ostatnie realizowane były nie tylko w zgodzie z dokumentem *MAT*[1], ale też intuicyjnie, poprzez normalne korzystanie z systemu.

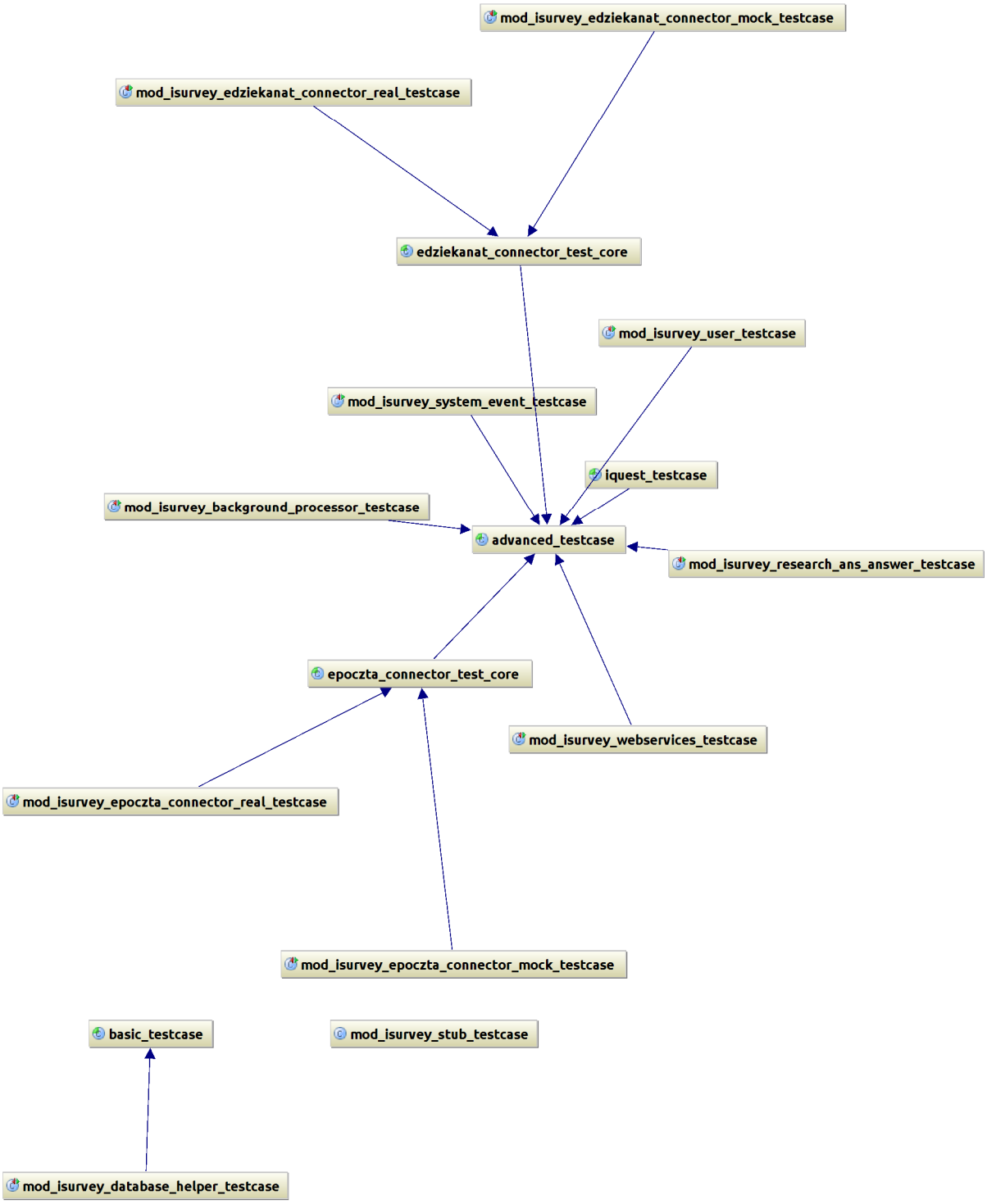
7.2 Testy jednostkowe

Testy jednostkowe zostały wykonane jako pierwsze i traktowane były z wysokim priorytetem. Realizowane były z użyciem klas *PHPUnit*, stosowanych powszechnie m.in. przy testowaniu wtyczek do platformy *Moodle*. Testy te były kluczowe dla rozwoju logiki systemu iQuest. Przygotowane zostały też konfigurowalne skrypty automatyzujące proces testowania w języku *BASH*. Część testów operuje na systemie w trybie produkcyjnym, część na trybie testowym, obsługującym tzw. „atrapy" (ang. *mock*), imitujące działanie systemów zewnętrznych poprzez zwracanie przykładowych danych.

Przy realizacji pierwszego wydania, za testy jednostkowe w pełni odpowiadał jeden z członków zespołu programistów. W wydaniu drugim, rolę tę przejął programista realizujący logikę systemu. Planowano stosować technikę *Test-Driven Development* (TDU) - rozwój w oparciu o testy), jednakże szybko zrezygnowano z tego pomysłu. Przyczyną był brak doświadczenia programistów w tej materii. Więcej informacji o problemach i rozwiązaniach przyjętych przy wykonywaniu testów znaleźć można w rozdziale 6.2.11.

[1] [12]

Rysunek 7.1: Struktura klas testujących (1) (wyk. Łukasz Wieczorek)



Rysunek 7.2: Struktura klas testujących (2) (wyk. Łukasz Wieczorek)

Na diagramach widoczne są trzy klasy służące do testowania, z których dziedziczą wszystkie inne:

basic_testcase – podstawowe testy jednostkowe

advanced_testcase – testy z użyciem bazy danych

iquest_testcase – rozszerzenie *advanced_testcase* na potrzeby testowania klas dziedziczących z *record*

7.3 Testy akceptacyjne

Na początku projektu, przygotowano program w języku *Java*, uruchamiający zestaw testów akceptacyjnych. W drugim wydaniu korzystano jednak wyłącznie z *Selenium IDE*, ze względu na możliwość szybszego rozpoznania problemów z poziomu przeglądarki, w przeciwieństwie do terminala.

Testy akceptacyjne rozpatrywane są na dwóch poziomach: automatycznym i manualnym. Różnica polega jedynie na tym, kto (lub co) wykonuje test - komputer z odpowiednim oprogramowaniem, czy człowiek.

7.3.1 MAT

Poniżej przedstawiono wybrane Manualne Testy Akceptacyjne:

TC1 : Logowanie do systemu przez *eKonto*		
Warunki początkowe		
• **Użytkownik jest niezalogowany** • **Użytkownik posiada *eKonto*** • **Połączenie z Internetem**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	Użytkownik wybiera opcję „zaloguj się”	Strona logowania do system *iQuest*
2.	Użytkownik naciska przycisk „Zaloguj przez eKonto”	Strona logowania *eLogin*
3.	Użytkownik wpisuje dane logowania	
4.	Użytkownik naciska przycisk „Zaloguj”	Przekierowanie na stronę systemu *iQuest*, wyświetlenie strony głównej z zalogowanym użytkownikiem.
Uwagi		

TC2 : Stworzenie Ankiety		
Warunki początkowe		
• **Zalogowany użytkownik z prawem do tworzenia ankiet** • **Użytkownik w trakcie przeglądania swojego katalogu ankiet**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	Użytkownik wybiera przycisk „Stwórz ankietę”	Strona umożliwiająca tworzenie ankiet
2.	Użytkownik podaje nazwę ankiety	
3.	Użytkownik podaje wstęp i podsumowanie ankiety	
4.	Użytkownik wybiera opcję definiowania pytań	Interfejs dodawania pytań
5.	Użytkownik dodaje pytanie jednokrotnego wyboru	Pojawia się pole na wpisanie treści pytania
6.	Użytkownik wpisuje treść pytania	
7.	Użytkownik naciska przycisk „Dodaj odpowiedź” dwukrotnie	Pojawiają się dwa pola do wpisania możliwych odpowiedzi
8.	Użytkownik podaje treści możliwych odpowiedzi	
9.	Użytkownik dodaje stronę wciskając przycisk „Dodaj stronę”	Wyświetla się nowa strona na dodawanie pytań
10.	Użytkownik dodaje pytanie otwarte	Wyświetla się pole na wpisanie treści pytania
11.	Użytkownik wpisuje treść pytania	
12.	Użytkownik wybiera przycisk „Zapisz zmiany”	Komunikat o pomyślnym stworzeniu ankiety
Uwagi		

Krok	**Dane**
2	„Ankieta testowa”
3	„Wstęp” oraz „Podsumowanie”
5	„Pytania jednokrotnego wyboru działają?”
7	„tak” oraz „nie”
10	„Pytania otwarte działają?”

Tablica 7.1: Poprawne dane dla scenariusza TC2

TC2.2 : Stworzenie Ankiety - brak pytań		
Warunki początkowe		
• **Zalogowany użytkownik z prawem do tworzenia ankiet** • **Użytkownik w trakcie przeglądania swojego katalogu ankiet**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	Użytkownik wybiera przycisk „Stwórz ankietę”	Strona umożliwiająca tworzenie ankiet
2.	Użytkownik podaje nazwę ankiety	
3.	Użytkownik podaje wstęp i podsumowanie ankiety	
4.	Użytkownik wybiera przycisk „Zapisz zmiany”	Komunikat o braku pytań w ankiecie
Uwagi		

TC3 : Edycja ankiety		
Warunki początkowe		
• **Zalogowany użytkownik z prawem do tworzenia ankiet** • **Użytkownik posiada prawo do edycji ankiety „Ankieta testowa”** • **Użytkownik w trakcie przeglądania swojego katalogu ankiet**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	Użytkownik wybiera przycisk „edytuj” przy „Ankiecie Testowej”	Strona umożliwiająca edycję „Ankiety Testowej”
2.	Użytkownik wciska przycisk „usuń” przy pytaniu drugim	Pytanie drugie znika
3.	Użytkownik naciska przycisk „Dodaj odpowiedź” przy pytaniu pierwszym	Pojawia się pole do wpisania możliwej odpowiedzi
4.	Użytkownik wpisuje możliwą odpowiedź	
5.	Użytkownik wybiera przycisk „Zapisz zmiany”	Strona wyświetla komunikat potwierdzający zapisanie zmian w ankiecie.
Uwagi		
Edytowane mogą być jedynie ankiety, na które nie udzielono jeszcze żadnej odpowiedzi		

Krok	**Dane**
5	„nie wiem”

Tablica 7.2: Poprawne dane dla scenariusza TC3

TC3 : Edycja ankiety - usunięcie wszystkich pytań		
Warunki początkowe		
• **Zalogowany użytkownik z prawem do tworzenia ankiet** • **Użytkownik posiada prawo do edycji ankiety „Ankiety testowej"** • **Użytkownik w trakcie przeglądania swojego katalogu ankiet**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	Użytkownik wybiera przycisk „edytuj" przy „Ankiecie Testowej"	Strona umożliwiająca edycję „Ankiety Testowej"
2.	Użytkownik wciska przycisk „usuń" przy każdym z pytań	Pytania znikają
3.	Użytkownik wybiera przycisk „Zapisz zmiany"	Strona wyświetla komunikat, że ankieta nie posiada pytań i prosi o ich dodanie.
Uwagi		
Edytowane mogą być jedynie ankiety, na które nie udzielono jeszcze żadnej odpowiedzi		

TC4 : Wybranie grupy docelowej		
Warunki początkowe		
• **Zalogowany użytkownik z prawami do tworzenia ankiet** • **Użytkownik posiada prawa do ankiety „Ankieta Testowa" oraz badania „Badanie Testowe"** • **Użytkownik posiada prawo do ankietowania grupy docelowej „test"**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	Użytkownik wybiera przycisk „Włącz tryb edycji"	Interfejs edycji *Moodle*
2.	Użytkownik wybiera przycisk „edytuj" przy „Badaniu Testowym"	Strona umożliwiająca edycję „Ankiety Testowej"
3.	Użytkownik wybiera grupę docelową „test"	
4.	Użytkownik wybiera przycisk „Zapisz zmiany"	Strona wyświetla komunikat, że ankieta została zaktualizowana pomyślnie.
5.	Respondent otrzymuje e-mail z powiadomieniem o ankiecie	
Uwagi		

TC5 : Udzielanie odpowiedzi		
Warunki początkowe		
• **Zalogowany użytkownik** • **Użytkownik znajduje się w grupie docelowej ankiety „testowa”** • **Użytkownik nie odpowiadał udzielał odpowiedzi na ankietę „testowa”**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	System prezentuje ankiety na które użytkownik jeszcze nie odpowiedział	
2.	Użytkownik wybiera ankietę „testowa”	System prezentuje ankietę
3.	Użytkownik zaznacza odpowiedź na pytanie 1 jako „tak”	
4.	Użytkownik podaje odpowiedź na pytanie drugie jako „tak”	
5.	Użytkownik potwierdza wypełnienie ankiety przyciskiem „Wyślij”	System prezentuje ankiety na które użytkownik jeszcze nie odpowiedział oraz komunikat o pomyślnym przesłaniu odpowiedzi
6.	Respondent otrzymuje e-mail z powiadomieniem o ankiecie	
Uwagi		

TC6 : Sprawdzenie wyników		
Warunki początkowe		
• **Zalogowany użytkownik z prawem do oglądania wyników ankiety „Testowa”**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	Użytkownik wybiera badanie, którego podstawowe wyniki chce sprawdzić	System prezentuje podsumowanie ankiety *Moodle*
Uwagi		
Dotyczy podstawowych wyników. Wyniki zaawansowane obsługuje zewnętrzny serwer *BI*		

TC7 : Dodawanie grupy docelowej		
Warunki początkowe		
• **Zalogowany administrator z prawami do tworzenia grup docelowych** • **Brak w systemie grupy docelowej „Grupa 1”**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	Administrator wybiera przycisk „Zarządzaj grupami docelowymi”	System prezentuje dostępne grupy docelowe w systemie
2.	Administrator wybiera przycisk „dodaj”	System prezentuje interfejs dodawania grupy docelowej
3.	Administrator podaję nazwę nowej grupy „Grupa 1” i wskazuje jej członków	
4.	Administrator wybiera grupę nadrzędną dla nowej grupy docelowej	System automatycznie zapisują zmiany
Uwagi		
Test przygotowany na podstawie makiety systemu *iQuest*		

TC8 : Edycja grupy docelowej		
Warunki początkowe		
• **Zalogowany administrator z prawami do tworzenia grup docelowych** • **Grupa docelowa „Grupa 1” istnieje w systemie**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	Administrator wybiera przycisk „Zarządzaj grupami docelowymi”	System prezentuje dostępne grupy docelowe w systemie
2.	Administrator wybiera przycisk „zmień nazwę”	System prezentuje interfejs zmiany nazwy grupy docelowej
3.	Administrator pozostawia nazwę niezmienioną i zatwierdza zmiany	System prezentuje dostępne grupy docelowe w systemie
4.	Administrator wybiera grupę nadrzędną dla grupy docelowej i dodaje nowego członka do grupy	System automatycznie zapisują zmiany
Uwagi		
Test przygotowany na podstawie makiety systemu *iQuest*		

TC9 : Logowanie bez użycia *eKonta*		
Warunki początkowe		
• **Użytkownik jest niezalogowany** • **Istnieje konto użytkownika w systemie**		
Krok	**Akcja**	**Oczekiwana odpowiedź**
1.	Użytkownik wpisuje adres systemu	Strona główna *Moodle* z kursem zawierającym system *iQuest*
2.	Użytkownik naciska przycisk „Zaloguj się”	Strona logowania
3.	Użytkownik wpisuje dane logowania	
4.	Użytkownik naciska przycisk „Zaloguj się”	Przekierowanie na główną stronę *Moodle* z kursem zawierającym system *iQuest*. Wyświetla się napis: „Jesteś zalogowany(a) jako...”
Uwagi		

Dodatkowo, poniżej znajduje się wykaz mapujący przypadki testowe do przypadków użycia:

- TC1.X – UC09 Logowanie do systemu.
- TC2.X – UC01 Stworzenie ankiety
- TC3.X – UC02 Edycja ankiety
- TC4.X – UC03 UC04 Wybranie grupy docelowej, uruchomienie ankiety
- TC5.X – UC05 Udzielenie odpowiedzi
- TC6.X – UC06 Sprawdzenie wyników
- TC7.X – UC07 Tworzenie grupy docelowej
- TC8.X – UC08 Edycja grupy docelowej
- TC9.X – UC09 Logowanie do systemu

7.3.2 AAT

Automatyczne testy akceptacyjne realizowano w zgodzie z testami manualnymi, operując na tych samych wytycznych. Nagrywanie testów odbywało się za pomocą oprogramowania *Selenium IDE*, udostępnianego w formie rozszerzenia dla przeglądarki *Mozilla Firefox*. Pierwotnie, testy były konwertowane do języka *Java*, w celu uruchamiania ich za pomocą jego środowiska, oferującego sporą swobodę przy projektowaniu warunków początkowych i końcowych dla testów. Problemy, jakie wynikały z takiego działania, opisane zostały w rozdziale 6. Na ich podstawie zdecydowano o pozostaniu w obrębie *Selenium IDE*, które samo w sobie również umożliwia automatyzację. Dla dodatkowego ułatwienia zadania, przygotowano skrypt ustawiający bazę danych w stan początkowy dla realizacji testów.

7.4 Inne metody zapewniania jakości

Konieczność zapewnienia jak najwyższej jakości oprogramowania, wymusiła testowanie w kontrolowanych warunkach na różnych urządzeniach. Co prawda, lokalne serwery developerskie pracowały zawsze w oparciu o system *Ubuntu 12.04 LTS*[2], z serwerem *Apache* i systemem zarządzania bazą danych *PostgreSQL*, jednak maszyny klienckie były już znacznie bardziej różnorodne.

Testy klienckie odbyły się na komputerach stacjonarnych klasy PC oraz równoważnych laptopach, z użyciem zarówno systemów rodziny *Windows* (wersje od *XP* do *Win8*), jak i *Linux* (wspomniana wcześniej wersja *Ubuntu*) oraz *Mac OS X*. Ponadto, przetestowano trzy główne platformy mobilne (Android, iOS @ iPhone 4S, WP7.x @ Nokia Lumia 710) poprzez dostęp do systemu z poziomu telefonów komórkowych. Komputery użyte do testów charakteryzowały się posiadaniem co najmniej jednordzeniowego procesora ze zbiorem 32-bitowych instrukcji i taktowaniem zegara nie mniejszym niż 1,5 GHz, oraz 4 GB pamięci RAM.

Na podstawie testów, utworzono dwa raporty: „wygląd i działanie systemu *iQuest* na platformach mobilnych" oraz „wygląd i działanie systemu *iQuest* w różnych konfiguracjach system-przeglądarka", udostępnione w ramach systemu zarządzania projektem[3]. Wynika z nich, że system *iQuest* jest przenośny.

[2] [6]
[3] [12]

Rozdział 8

Zebrane doświadczenia i wnioski

8.1 Doświadczenia związane z zastosowanymi technologiami

Podczas pracy zebrano następujące doświadczenia, bezpośrednio związane z zastosowanymi technologiami:

1. Implementowanie testów jednostkowych z użyciem PHPUnit – w celu kontroli poprawności logiki pisanej w języku *PHP*.
2. Implementowanie logiki aplikacji w języku *PHP* na podstawie UML – na podstawie diagramów UML oraz rozmów z architektem zaimplementowano kod back-endu w języku *PHP*.
3. Implementowanie usług internetowych (protokół *SOAP*) – dla komunikacji z usługami uczelni wykorzystano klienty dostarczone przez DRO, natomiast dla komunikacji z serwerem raportowania wykorzystano API *external services* platformy *Moodle*, po stronie systemu *iQuest* oraz *Apache Axis* (Java) po stronie serwera *JasperReports*.
4. Implementowanie schematu bazy danych w formacie *XMLDB* na podstawie UML – wykorzystano interfejs do zarządzania *XMLDB* dostarczany przez *Moodle*; drobne poprawki wprowadzano ręcznie w pliku *XML*, definiującym schemat bazy danych.
5. Implementowanie modułów uwierzytelniania systemu *Moodle* – na podstawie przykładowych oraz istniejących modułów uwierzytelniania stworzono moduł uwierzytelniania przez *eKonto* oraz moduł dla absolwentów.
6. Rozszerzanie funkcjonalności oprogramowania o otwartym kodzie źródłowym – w celu spełnienia wymagań projektowych dodano m.in. okresowe sprawdzanie ważności sesji *eKonto* do mechanizmu zarządzania sesją *Moodle*, zmodyfikowano wygląd strony logowania i panelu użytkownika.
7. Konfigurowanie systemów operacyjnych *Ubuntu* – w celu zdalnej konfiguracji wykorzystano protokół *SSH*. Należało zainstalować i skonfigurować wymagane oprogramowanie (w tym *Apache*, *CRON*, *PostgreSQL*, *PHP*, *Check Point's Linux SNX*), przygotować katalogi repozytoriów kodu.
8. Programowanie z użyciem *Eclipse PDT* – wykorzystano zintegrowane środowisko programistyczne w celu zwiększenia produktywności programistów.

9. Konfigurowanie *JasperServer* – dodano dialekt zapytań *JoSQL*, własne źródło danych (pomocne okazały się informacje z projektu *Business Intelligence Server* dostępnego na *Redmine*[1]),
10. Projektowanie raportów *JasperReports* – zaprojektowano raporty w *JasperReports Studio*.
11. Korzystanie z klientów *VPN* (firmy *CheckPoint*) – w celu uzyskania dostępu do sieci wewnętrznej, zobligowano zespół do zestawiania połączenia *VPN*.
12. Korzystanie z systemu zarządzania projektami *Redmine* – zarządzanie zagadnieniami, bazą wiedzy, repozytorium, plikami, korzystanie z dostępnych metod komunikacji: komunikaty, forum, komentarze.
13. Korzystanie z systemów kontroli wersji *SVN* i *Git* – *SVN* wykorzystano jako repozytorium kodu, *Git* natomiast posłużyło zespołowi podczas tworzenia niniejszej pracy.
14. Pisanie dokumentacji technicznej oraz dokumentacji użytkownika.
15. Implementowanie modułów aktywności systemu *Moodle*.
16. Projektowanie testów akceptacyjnych z użyciem *Selenium IDE*.
17. Implementowanie formularzy z wykorzystaniem *JavaScript*.

8.2 Wnioski z udziału w realizacji projektu

Bardzo istotna cześć doświadczeń związanych z projektem wiąże się z pracą zespołową. Członek zespołu musi posiadać następujące cechy: sumienność, punktualność, odpowiedzialność, dokładność, terminowość, prawdomówność, uczciwość, asertywność, zdolność kreatywnego myślenia, kultura osobista oraz komunikatywność. Praca w grupie nad dużym projektem wymaga dobrej koordynacji oraz odpowiedniego podziału zadań, które przydzielano w zależności od: umiejętności, doświadczeń i zainteresowań poszczególnych członków zespołu. Bardzo ważna była dostępność architekta i kierownika projektu.

8.2.1 Wnioski indywidualne

Krzysztof Marian Borowiak:

- Praca zespołowa znacząco upraszcza realizowanie różnych projektów – gdy jeden z członków zespołu czegoś nie wie lub uzyskuje inne efekty niż oczekiwane, może to szybko skonsultować z kolegami. Jest to znacznie szybsze i efektywniejsze, niż wyszukiwanie informacji samodzielnie.
- Dokumentacja wykonywana przez „społeczność" nie jest tak dobra, jak próbują przekonywać zwolennicy rozwiązań „otwartych" i „darmowych" – jest niekompletna i nie można się w pełni na niej oprzeć.
- Wykonywanie testów jednostkowych we wstępnej fazie projektu, zwłaszcza, gdy szczegóły architektoniczne logiki oprogramowania nie są jeszcze kompletne, jest niezwykle uciążliwe i czasochłonne. Wyjątkiem jest zastosowanie techniki TDU, przy czym wymaga ona niemałego doświadczenia ze strony programistów.

[1] [12]

- Wraz z projektem, powinna być rozwijana baza wiedzy. Każdy problem, z którym dowolny członek zespołu się spotkał, powinien zostać zanotowany i opisany na wspólnej platformie, aby pozostali członkowie zespołu, mogli się z nim zapoznać i uniknąć jego powielania.
- Zespół zarządzający powinien dbać o to, aby zadania były przydzielane bezpośrednio poszczególnym członkom zespołu programistów i egzekwować ich wykonanie w wyznaczonym terminie.
- Należy oddzielać pracę od spraw osobistych.
- Zaufanie i lojalność to podstawa dobrej współpracy zespołu – każda poruszana kwestia powinna znaleźć konstruktywne rozwiązanie.
- Nie każdy student kierunku Informatyka wiąże swoją przyszłość z rolą programisty. Jest wiele specjalizacji, w których może się on rozwijać. SDS, skupiające się wokół wytwarzania oprogramowania pozwala na wykonanie pracy dyplomowej tylko w tym zakresie.

Maciej Trojan

- Niewątpliwą zaletą pracy nad dużym projektem było zrozumienie jak ważną role odgrywa dobre przygotowanie projektu w fazie planowania przed przystąpieniem do programowania.
- Praca w zespole z określonym podziałem na role umożliwia szybkie rozwiązywanie problemów napotkanych w trakcie wytwarzania oprogramowania. Ponadto podział ten wymusza zagłębienie się w dziedzinach powiązanych z pełnioną rolą, co znacząco wpływa na rozwijanie umiejętności.
- Aby projekt został zrealizowany w wyznaczonym terminie, powstające zagadnienia należy przydzielać do konkretnych osób oraz egzekwować ich wykonanie w terminie.

Krzysztof Urbaniak:

- Kluczem do sprawnej pracy jest dobra organizacja zasobów. System *Redmine* bardzo pomagał w odpowiednim przydziale zadań do osób, w taki sposób, aby projekt ukończyć w określonym czasie.
- Wspólna praca zespołu w jednym pokoju wpływa na większą świadomość jego członków o postępie prac. Przekłada się to na szybsze znajdowanie błędów, czy grupowe rozwiązywanie większych problemów (np. w formie burzy mózgów).
- Rodzina systemów operacyjnych *Linux* jest o wiele przyjaźniejsza programistom, niż *Windows*. Góruje ona chociażby w kwestii dostępności aplikacji operujących na plikach tekstowych, a także łatwości pisania skryptów. Programista pracujący z użyciem *Linuxa* ma poczucie, że wiele procesów można zautomatyzować, w przeciwieństwie do pracy w środowisku *Windows*.
- Wybór rozwiązań, o których ma się szczątkowe pojęcie, nie jest dobrym pomysłem. Lepiej jest wybierać rozwiązania bardziej znane, nawet jeśli wydaje się, że wymagają więcej zaangażowania.
- Wybór pracy inżynierskiej pod nadzorem zespołu SDS był dobrym pomysłem. Wsparcie architekta oraz kierownika było bardzo istotne, m.in. w kategorii ukończenia prac w terminie.

Łukasz Wieczorek:

- Zmiany zatwierdza się (ang. *commit*) dopiero, gdy funkcjonalność, która miała być zaimplementowana, jest kompletna i przetestowana.
- Logika, z której będą korzystać programiści interfejsu, powinna zostać przygotowana wcześniej, by nie opóźniać ich pracy.
- Jeśli nie wiadomo, jak zaimplementować daną funkcjonalność, warto skierować się do architekta – zwykle dysponuje on większą wiedzą ogólną w takich kategoriach.
- Werbalizacja problemu bardzo często pomaga w jego rozwiązaniu.
- Wielokrotnie powtarzane fragmenty kodu należy jak najszybciej zrefaktoryzować, by uchronić się od poprawiania podobnych konstrukcji składniowych. Ponadto, wcześniejsza refaktoryzacja znacznie ułatwia dalszą pracę nad kodem.
- Testy jednostkowe ułatwiają refaktoryzację kodu – nawet jeśli ich utrzymywanie jest kosztowne, warto je realizować.
- „Nie należy mnożyć bytów ponad potrzebę".
- Podstawą efektywnej pracy jest kontrola czasu poświęcanego na realizację przydzielonych zdań.
- Dobra komunikacja w zespole to podstawa sukcesu.

8.2.2 Wnioski zbiorowe

Rozwijanie istniejącego oprogramowania wymaga dużo większego nakładu pracy niż konstrukcja oprogramowania od podstaw. Wiele czasu poświęca się na analizę rozwiązań zastosowanych przez twórców rozwiązania bazowego. W trakcie implementacji pojawiają się problemy, na które długimi godzinami szuka się rozwiązań. Niektórych nie udaje się w ogóle rozwiązać, co prowadzi do konieczności modyfikacji wcześniej utworzonego oprogramowania (ang. *hacking*). Praca nad oprogramowaniem, którego tworzenie zaczęło się przed ponad dziesięcioma laty, wymaga pracy z rozwiązaniami architektonicznymi, które dawno zostały już zarzucone (np. *transaction script* porzucono na rzecz *Model View Controller*). Dodatkową trudnością są zmieniające się lub niewspierane już interfejsy programowania aplikacji. Wymienione problemy dotyczą szczególnie obszernego oprogramowania, właśnie takiego z jakim przyszło nam pracować – *Moodle* to wg programu *CLOC* ponad 2 miliony linii kodu. Uważamy, że decyzja zespołu zarządzającego odnośnie rozwijania systemu *Moodle* była błędna i nieprzemyślana, zwłaszcza w kontekście braku doświadczenia całego zespołu w tej technologii. Potwierdzenie naszych wniosków widzimy chociażby porównując wydajność pracy z innymi zespołami realizującymi projekty w ramach *Software Development Studio*. Jednakże, mimo wszelkich trudności, dużym nakładem pracy, udało nam się zakończyć projekt w terminie.

Możliwość współpracy z SDS była niewątpliwie zaletą. Po pierwsze, projekt, który należało wykonać był bardzo złożony. Powiększenie zespołu projektowego o dwie dodatkowe osoby, dało możliwość odciążenia pozostałych od części pracy, związanej z projektowaniem systemu, a co za tym idzie, pozostali mogli zająć się samą implementacją. Po drugie, wszystkie ewentualnie spory można było szybko rozwiązać, pytając o zdanie architekta, nie tracąc tym samym czasu na niepotrzebne spory o sposób realizacji. Po trzecie, udostępniony system *Redmine* znacząco polepsza organizację pracy. Dzięki „śledzeniu zagadnień" można lepiej rozporządzać czasem,

co więcej, każdy wie, co ma robić. Nie bez znaczenia jest też możliwość korzystania z repozytorium *SVN*. Wyposażony pokój jest udogodnieniem, które wpływa na lepszą komunikację w zespole. Korzyścią wynikającą ze sposobu organizacji pracy jest konieczność trzymania się terminów, dzięki której nasz ostatni semestr studiów inżynierskich był dobrze zaplanowany pod względem równomiernego rozkładu pracochłonności.

Rozdział 9

Zakończenie

9.1 Podsumowanie

Realizacja projektu obejmującego utworzenie systemu *iQuest* została zakończona sukcesem. Platforma *Moodle*, w połączeniu z autorskimi wtyczkami składającymi się na system *iQuest* zapewnia pełnię zleconej funkcjonalności, spełniając zarazem wymagania przedstawione w niniejszym dokumencie.

Dzięki wdrożeniu systemu *iQuest*, Politechnika Poznańska uzyska dostęp do niezbędnego narzędzia prowadzenia badań wśród swoich absolwentów w rozumieniu ustawy „Prawo o Szkolnictwie Wyższym". Jest to znaczący krok w stronę lepszego poznania potrzeb rynku, zarówno pracodawców, jak i samych studentów, pozwalający usprawnić mechanizmy zapewniania jakości kształcenia funkcjonujące na uczelni.

Udział w tak dużym i znaczącym dla Politechniki Poznańskiej projekcie przyczynił się do znaczącego rozwoju jego uczestników. Pozyskali oni wiele cennych doświadczeń i wyciągnęli znaczną liczbę najróżniejszych wniosków. Autorzy niniejszej pracy dyplomowej wyrażają więc nadzieję, że utworzony przez nich system zostanie pozytywnie przyjęty przez studentów, absolwentów oraz prowadzących.

9.2 Propozycja dalszych prac

Podczas spotkania z reprezentantami Działu Rozwoju Oprogramowania poruszony został temat etykietowania (ang. *tag*) badań. Proponowany mechanizm z pewnością usprawniłby proces wyszukiwania badań. W obecnej wersji systemu zachętą dla studentów do wypełniania ankiet są materiały publikowane przez wykładowców uczelni, do których dostęp przyznawany jest użytkownikom, którzy w określonym czasie udzielili odpowiedzi w dowolnym badaniu. W przyszłości mechanizm ten można zastąpić możliwością subskrypcji (wykupu) dostępu do publikowanych materiałów z wykorzystaniem wirtualnej waluty (np. punktów za udział w badaniach).

Dodatek A

Informacje uzupełniające

A.1 Wkład poszczególnych osób w przedsięwzięcie

Skład zespołu pracującego nad projektem został przedstawiony w tablicy A.1.

Stanowisko	**Osoba**
Założyciel projektu, klient	prof. Jerzy Nawrocki
Główny użytkownik	prof. Jerzy Nawrocki
Główny dostawca	Tomasz Sawicki
Dostawca od strony DRO	Tomasz Sawicki
Starszy konsultant	Sylwia Kopczyńska
Konsultant	Sylwia Kopczyńska
Kierownik projektu	inż. Marcin Domański
Analityk/Architekt	inż. Błażej Matuszczyk
Programiści	Krzysztof Marian Borowiak Maciej Trojan Krzysztof Urbaniak Łukasz Wieczorek

Tablica A.1: Osoby związane z przedsięwzięciem

Odpowiedzialność za utworzenie treści niniejszej pracy dyplomowej została przedstawiona poniżej:

Krzysztof Marian Borowiak

- Edycja i dostosowanie szablonu pracy w środowisku LaTeX
- Redakcja całej pracy, włącznie z częściami współautorów
- Pozyskanie, przetworzenie i zamieszczenie materiałów zewnętrznych
- Pozyskanie, przetworzenie i zamieszczenie materiałów pochodzących od zespołu zarządzającego
- Rozdział 1 – Wprowadzenie
- Rozdział 7 – Zapewnianie jakości i konserwacja systemu
- Rozdział 6.2.11 – Testy jednostkowe i akceptacyjne
- Rozdział 8 – Wnioski - część własna
- Rozdział 9 – Zakończenie

- Dodatki
- Zrzuty ekranowe

Maciej Trojan

- Rozdział 6.2.3; 6.2.4 – Napotkane problemy i ich rozwiązania – Inicjalizacja bazy danych; Inicjalizacja modułu
- Rozdział 6.3.11 – Użyte technologie – JavaScript
- Rozdział 8 – Wnioski – część własna

Krzysztof Urbaniak

- Rozdział 6.2.5-10 – Napotkane problemy i ich rozwiązania - Formularze; Role; Formater kursu; Tworzenie badania; Tworzenie ankiety; Hierarchia CSS
- Rozdział 6.5 – Interfejs
- Rozdział 6.7 – Powiązanie logiki z interfejsem
- Rozdział 8 – Wnioski - część własna

Łukasz Wieczorek

- Rozdział 6.2.12 – Mapowanie obiektowo-relacyjne
- Rozdział 6.3.1-10 – Użyte technologie - Moodle, PHP, PHPUnit, Selenium, PostgreSQL,Eclipse IDE, SVN, Redmine, JasperReports, JetBrains PhpStorm
- Rozdział 6.6 – Logika (back-end)
- Rozdział 7.1.2 – Testy jednostkowe
- Rozdział 8 – Wnioski - część własna
- Część grafik (z odpowiednim odniesieniem w etykiecie)

Zespół zarządzający projektem (Marcin Domański, Błażej Matuszyk)

- Materiały[12], zastosowane jako podstawa dla rozdziałów 1-5
- Część grafik (z odpowiednim odniesieniem w etykiecie)

Odpowiedzialność za część implementacyjną systemu została przedstawiona poniżej:

Krzysztof Marian Borowiak

- Testy jednostkowe i akceptacyjne
- Dokumentacja dla Użytkownika Końcowego (Administratora, Użytkownika)
- Dokumentacja techniczna (raporty dot. funkcjonowania na platformach mobilnych oraz w różnych środowiskach)

Maciej Trojan

- Interfejs użytkownika
- Utworzenie Bazy Danych

Krzysztof Urbaniak

- Interfejs użytkownika
- Powiązanie interfejsu z logiką

Łukasz Wieczorek

- Logika
- Testy jednostkowe
- System raportowania

Autorzy niniejszej pracy dyplomowej inżynierskiej składają serdeczne podziękowania Promotorowi, dr. inż. Bartoszowi Walterowi, który wytrwale wspierał ich przy realizacji zadań związanych z projektem, oraz dr. inż. Grzegorzowi Pawlakowi, prowadzącemu przedmiot „Pracownia inżynierska", za aktywne motywowanie ich do wytężonej pracy.

Podziękowania należą się także zespołowi zarządzającemu za przygotowanie wymaganych materiałów źródłowych, oraz opiekunom *SDS*, w tym w szczególności mgr inż. Sylwii Kopczyńskiej, za niewyczerpaną wiarę w możliwości zespołu programistów.

A.2 Wykaz użytych narzędzi i technologii

Numery w nawiasach w poniższej liście oznaczają numer wersji.

- Apache (2.2.22)
- BASH (4.2.37)
- Check Point's Linux SNX (800007027)
- Chrome (24.0)
- CLOC (1.56)
- CRON (3.0)
- Eclipse IDE (3.7.2)
- FastStone Capture (5.3)
- Git (1:1.7.10.4)
- JasperReports Studio (1.3.2)
- JasperReports Server (5.0.1)
- Java (1.6.0_38)
- JavaScript
- JetBrains PhpStorm (5.0.4)
- Kazam Screencaster (1.0.6)

- Meld (1.6.0)
- Moodle (2.3.1)
- Mozilla Firefox (18.0.1)
- MySQL (14.14)
- PHP ($>$ 5.3)
- PHPUnit (3.6.10)
- psql (9.1.7)
- PostgreSQL (9.1)
- recordMyDesktop (0.3.8.1)
- Redmine
- Selenium IDE (1.10.0)
- SSH (1:6.0)
- SVN (1.7.5)
- TexLive (20120611)
- TexMaker (3.4)
- VIM (7.3)
- Zend PHP Developer Tools for Eclipse IDE (3.0.2)

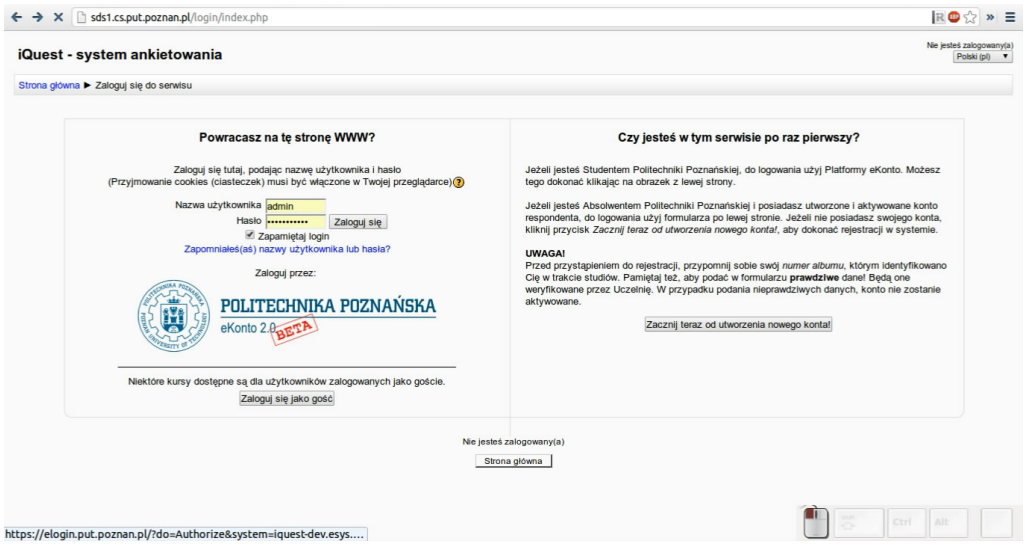
A.3 Zawartość płyty CD

Do dokumentu załączono płytę CD o następującej zawartości:

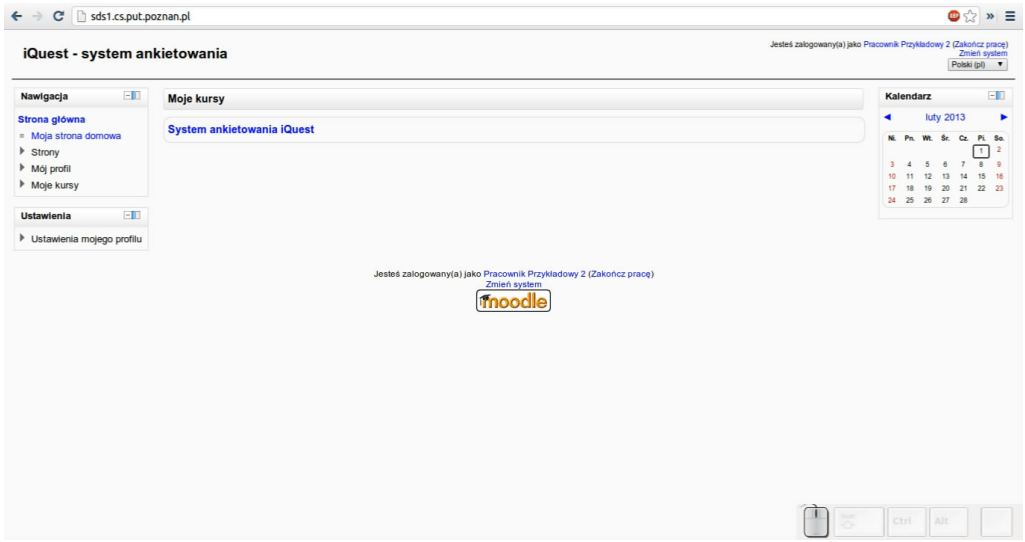
- Dokumentacja systemu iQuest
- Niniejszy dokument w formacie PDF
- Pliki źródłowe systemu iQuest
- Pliki źródłowe wykorzystywanej wersji Moodle

Dodatek B

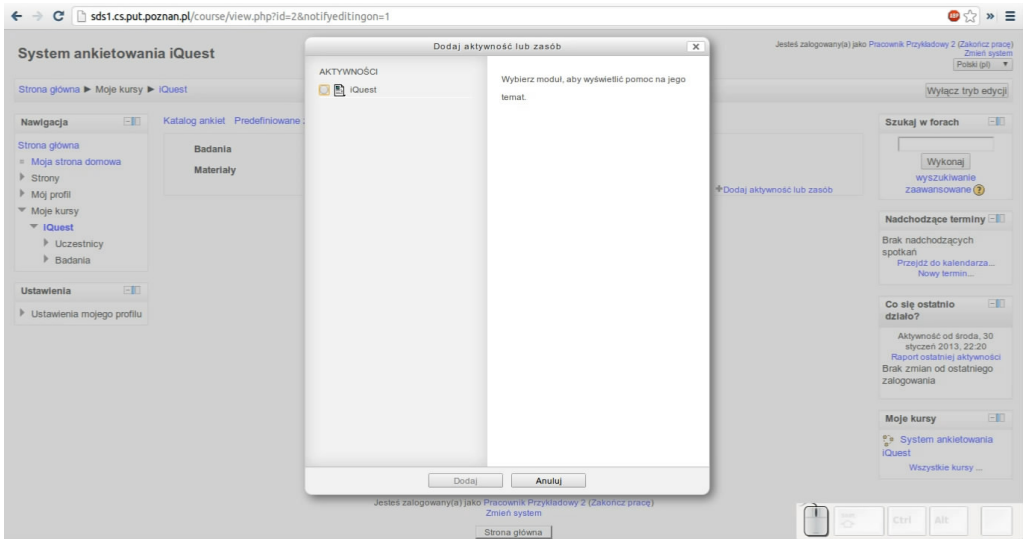
Wygląd aplikacji



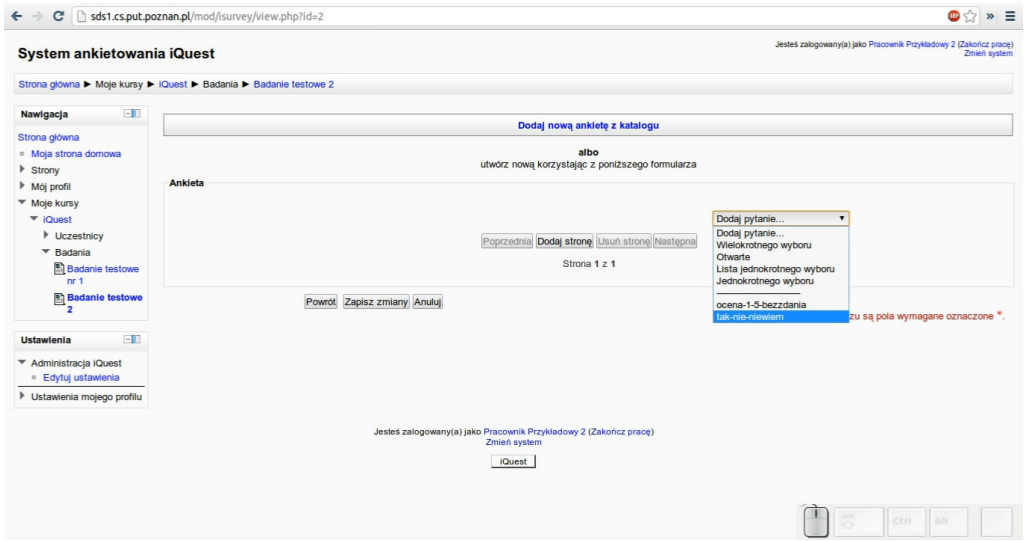
Rysunek B.1: Logowanie do systemu *iQuest* (wyk. Krzysztof Marian Borowiak)



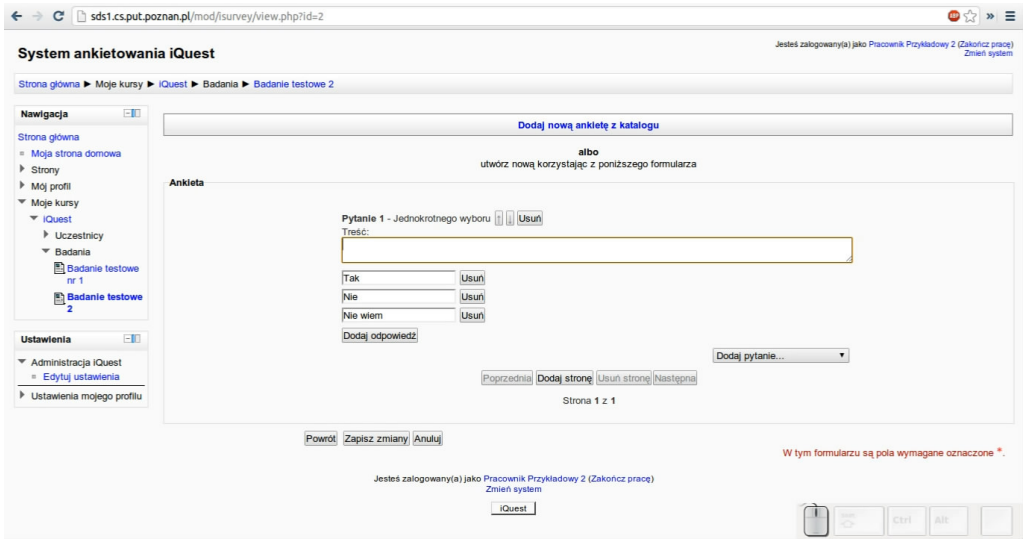
Rysunek B.2: Strona główna systemu *iQuest* – widok ankietera (wyk. Krzysztof Marian Borowiak)



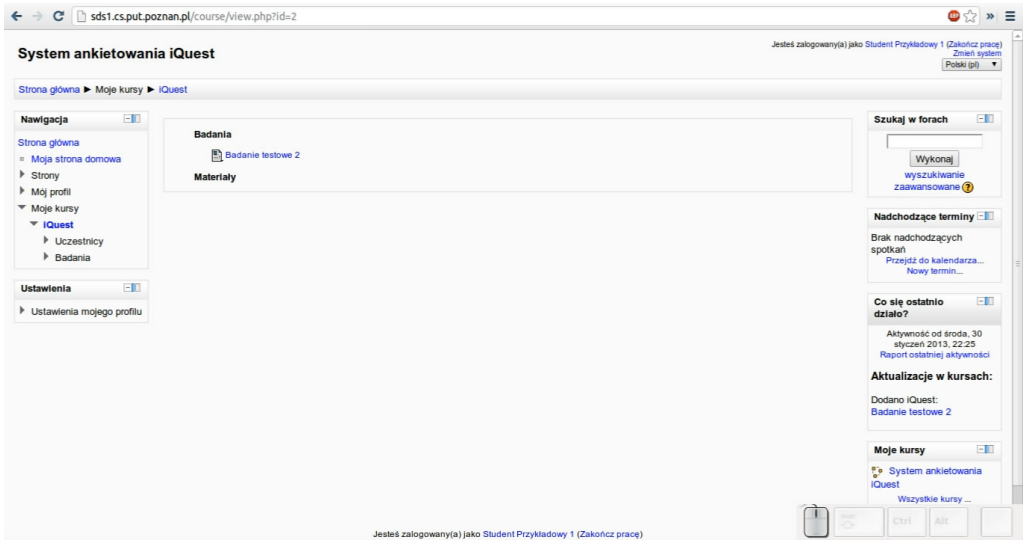
Rysunek B.3: Dodawanie badania w systemie *iQuest* (wyk. Krzysztof Marian Borowiak)



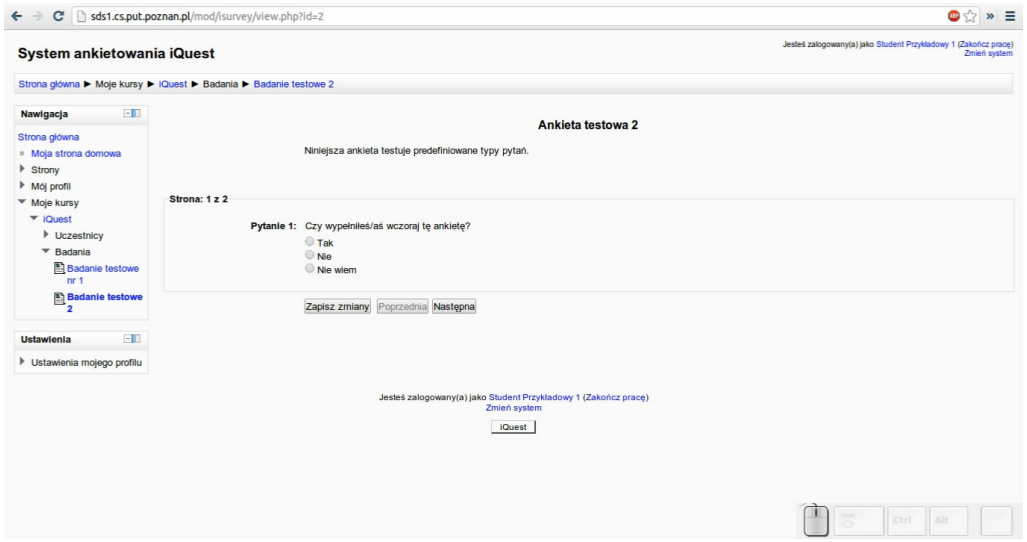
Rysunek B.4: Tworzenie ankiety w systemie *iQuest* – wybór typu pytania (wyk. Krzysztof Marian Borowiak)



Rysunek B.5: Tworzenie ankiety w systemie *iQuest* – modyfikacja pytania (wyk. Krzysztof Marian Borowiak)



Rysunek B.6: Lista badań dostępna dla respondenta (wyk. Krzysztof Marian Borowiak)



Rysunek B.7: Interfejs ankiety dla respondenta (wyk. Krzysztof Marian Borowiak)

Literatura

[1] Badania ankietowe. [on-line] `http://pl.wikipedia.org/wiki/Badania_ankietowe`.

[2] Soap full documented array plugin for moodle. [on-line] `https://github.com/ghigio/moodle-webservice_soapfda`.

[3] Ustawa z dnia 27 lipca 2005 r. – prawo o szkolnictwie wyższym, 2005. Dz. U. Nr 164, poz. 1365, z późn. zm., Art. 13a.

[4] Statut politechniki poznańskiej, 2011. Uchwała Nr 154, §83, ust. 5.

[5] *Moodle documnentation*, 2012-2013.

[6] *Official Ubuntu Documentation*, 2012-2013.

[7] *PHP Manual*, 2012-2013.

[8] *PHPUnit Manual*, 2012-2013.

[9] *Selenium Documentation*, 2012-2013.

[10] Bartosz Danowski. *Wstęp do CSS3 i HTML5*. Helion, 2011.

[11] Marko Gargenta Ellie Quigley. *PHP i MySQL - Księga przykładów*. Helion, 2007.

[12] B. Matuszyk M. Domański. Dokumentacja systemu „iquest". Materiały zamieszczone w ramach platformy Redmine w domenie `http://conaiten.put.poznan.pl`, 2012-2013.

[13] Iwona Pilchowska. Ankieta w badaniach ilościowych. [on-line] `http://www.badaniamarketingowe.org.pl/system/attachments/75/original/Zasady_tworzenia_ankiety_badania_ilosciowego.pdf?1288570397!`

[14] Dział Rozwoju Oprogramowania Politechniki Poznańskiej. Centralne uwierzytelnianie i wymiana danych., 2010. Wersja 1.2 (2010.07.06).

[15] World Wide Web Consortium (W3C). Soap version 1.2 part 1: Messaging framework (second edition). [on-line] `http://www.w3.org/TR/soap12-part1/`.

Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@mastersthesis{ key,
    author = "Krzysztof Marian Borowiak \and Maciej Trojan \and Krzysztof Urbaniak \and Łukasz
Wieczorek",
    title = "{iQuest - system rozszerzonych ankiet studenckich}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2013",
}
```