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 hab. 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 (ToDo)

Treść testowa. *TexTit treść testowa.*. Treść testowa.

AKTUALNA WERSJA OPARTA NA: Project Brief

Context The customer of this project is the Dean of the Faculty of Computing Science at the Poznan University of Technology. The project is going to be developed for the University, which is a middle-sized, Polish higher education institution. Problems and Their Impact Problems: The University doesn't have statistics on its graduates' careers. It also doesn't have a feedback from the current and former students on their satisfaction of the services provided by the University. Impact of the problems: Because of the above problems, the University has a limited ability to measure and improve the quality of provided education services. If the quality is not satisfying, the University's prestige will decrease and fewer students will be willing to study there. This will cause the University to receive less funding from government and other organizations. Outline of the Solution The solution is to conduct surveys among the University students and graduates. In these surveys they will be asked to provide an opinion on the studies and to give some information on their professional life. There will be a web application developed which will be the platform used to carry out surveys. It will allow reaching various target groups (current students, former students, etc.), verifying respondent's identity, collecting, storing and analyzing answers, and generating reports. The solution will also provide incentives to encourage potential participants to take part in the surveys. Additionally, the application will allow to make anonymous surveys so in reports there will be no personal information about respondent. Moreover there will be some extra articles available only for those who complete the survey and kind of CMS to put articles on line. Business Constraints Application has to be done until the end of February/March 2013. It's a time when bachelor's degree students need to finish their project before final exam. They are our developers. Budget is not known yet. Preliminary Risk Assessment Priority will be high enough to get appropriate support from client-side. Also deadline seems to be suitable. The budget is still not known. Client did not specify a technology which has to be used in this project. There are some well-known technologies which may contribute our goals. Moreover, we hope that we will find easily a group of developers which can handle one of these technologies. Acceptance Criteria The most important quality criteria for the project are usability. Application should be useful for users and provide all functions at the easiest way to don't discourage end-user. System should allow as many users as possible to use it at same time. It is of high importance that the deadline cannot be met. Proposed Staff Jerzy Nawrocki – Jerzy.Nawrocki@cs.put.poznan.pl – Executive Sylwia Kopczyńska – Sylwia.Kopczynska@cs.put.poznan.pl – Senior Supplier Michał Witczak –

mich.witczak@gmail.com – Project Manager Błażej Matuszyk – blasoft@live.com – Architect / Quality Assurance Marcin Domański – marcaj13@gmail.com – Analyst Additional information Detailed problem description: University is currently using its own system to conduct surveys. Unfortunately, it has many disadvantages and students are not willing to use it (e.g. Users have problems with choosing the right tutor of their courses, sometimes they just cannot find them on the list). Moreover, that application doesn't provide features required by the customer. It doesn't support short surveys after lectures – surveys intended for graduates. Overall application should be more flexible to assist in carrying out many types of polls and surveys.

1.2 Cele projektu

Zbudowanie systemu umożliwiającego łatwe przeprowadzanie ankiet wśród studentów oraz absolwentów uczelni. System powinien:

- współpracować z innymi systemami funkcjonującymi na uczelni (np. Sokrates)
- oferować dużą elastyczność przy definiowaniu ankiet oraz grup respondentów
- oferować rozbudowane możliwości raportowania

Rozdział 2

Opis procesów biznesowych

2.1 Aktorzy

Aktorami zdefiniowanymi w Systemie są:

- System – opisywany System iQuest.
- Administrator – zarządza sprawami technicznymi, związanymi platformą Moodle. 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.

2.2 Obiekty biznesowe

2.2.1 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.

2.2.2 Ankieta

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

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

2.2.3 Grupa Docelowa

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

- Studenci/Absolwenci

2.2.4 Raport

Zebrane odpowiedzi z jednego lub z kilku badań. Może zawierać wykresy, zestawienia.

2.3 Biznesowe przypadki użycia

Poniżej przedstawione zostały biznesowe przypadki użycia.

2.3.1 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)

2.3.2 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)

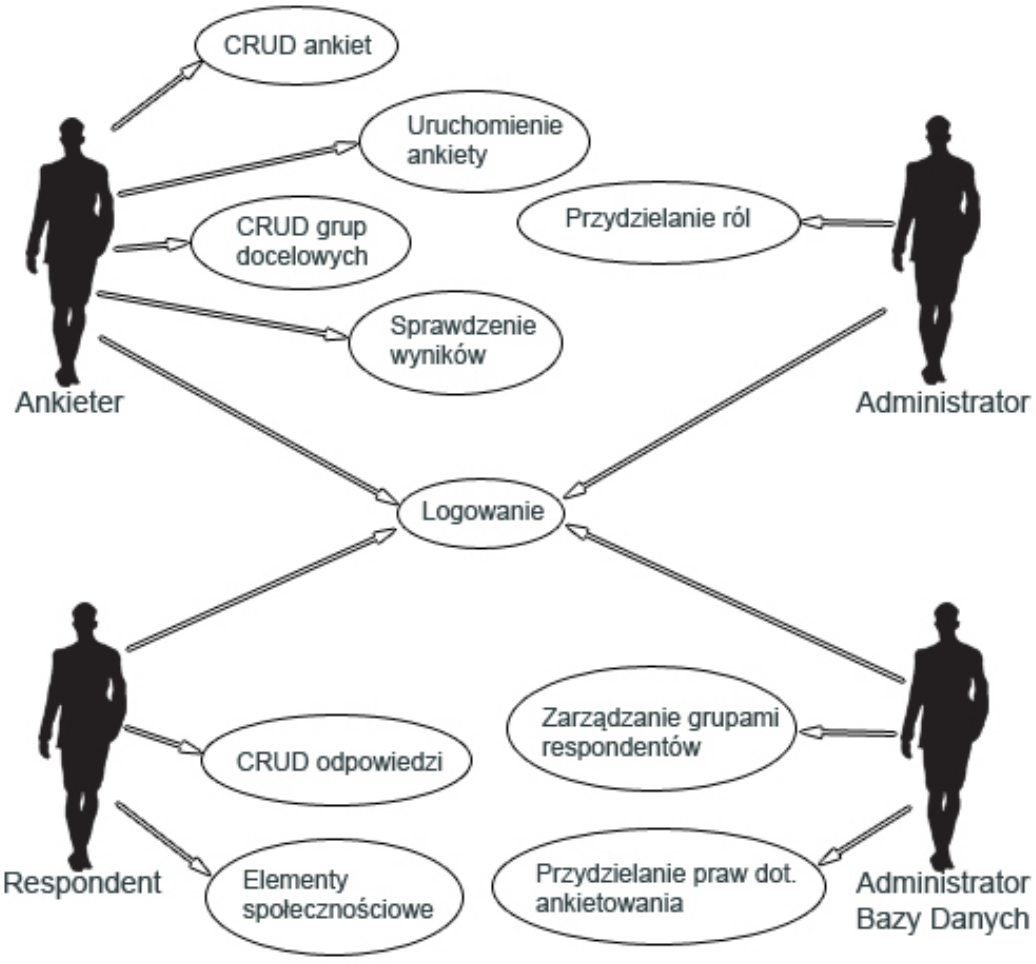
2.3.3 BC03: Zarządzanie Grupami Docelowymi

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

Rozdział 3

Wymagania funkcjonalne

3.1 Wstęp – diagram przypadków użycia



Rysunek 3.1: 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, Anki-

eter może tworzyć Badania i analizować ich Statystyki oraz Raporty, podczas gdy Respondent może odpowiadać w Ankietach w ramach skierowanych do niego Badań; Administrator zarządza przydzielaniem ról; Administrator Bazy Danych zajmuje zarządza grupami Respondentów i przydzielaniem praw i zezwoleń.

3.2 Ankieter

Poniżej przedstawiono przypadki użycia dotyczące Ankietera.

3.2.1 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 wpisuje atrybuty Ankiety: nazwę Ankiety, wstęp, podsumowanie 2. System prezentuje stronę umożliwiającą dodawanie pytań 3. Ankieter wybiera typ pytania 4. Ankieter wpisuje treść pytania 5. Ankieter podaje możliwe odpowiedzi 6. System prezentuje podsumowanie ankiety 7. Ankieter akceptuje ankietę 8. System zapisuje ankietę w Katalogu Ankiet Ankietera
Rozszerzenia
4.A Typ pytania: pytanie otwarte 4.A.1 Ankieter pomija krok 5. 5.A Ankieter chce dodać kolejne pytanie 5.A.1 Powrót do kroku 3.

3.2.2 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 wybiera pytanie do edycji/usunięcia 4. System prezentuje pytanie z możliwością edycji 5. Ankieter edytuje/usuwa pytanie 6. Ankieter potwierdza chęć zapisu zmienionej Ankiety 7. System zapisuje zmienioną Ankietę
Rozszerzenia
5.A. Edycja możliwych odpowiedzi do pytań 5.A.1 Ankieter edytuje możliwe odpowiedzi do pytań

3.2.3 UC03: Wybór Grupy Docelowej

Przypadek użycia: UC03: Wybór Grupy Docelowej
Aktorzy: Ankieter, Administrator Bazy Danych
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 przycisk „Utwórz badanie” 2. System prezentuje listę Grup Docelowych, do których Ankieter ma uprawnienia 3. Ankieter wybiera Grupy Docelowe dla danej Ankiety 4. Ankieter wybiera typ powiadamiania Respondentów 5. 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 Bazy Danych, 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.

3.2.4 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 Ankiety 2. Ankieter potwierdza chęć uruchomienia Badania 3. System rozsyła Ankietę do Respondentów 4. System powiadamia Respondentów o Ankiecie
Rozszerzenia
1.A. Ankieter chce prowadzić ankietę wielokrotną 1.A.1 Ankieter ustawia częstotliwość ankiety

3.2.5 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

3.3 Respondent

Poniżej przedstawiono przypadki użycia dotyczące Respondenta.

3.3.1 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. System prezentuje Ankietę Respondentowi 2. Respondent udziela odpowiedzi na pytania 3. System prezentuje podsumowanie odpowiedzi 4. Respondent zatwierdza wypełnioną Ankietę 5. System zapisuje odpowiedzi
Rozszerzenia
1.A. Przedawniona Ankieta 1.A.1 System informuje, że Ankieta już się zakończyła 5.A. Brak odpowiedzi na niektóre pytania 5.A.1 System informuje, że pozostały pytania bez odpowiedzi 5.A.2 Powrót do kroku 2.

3.4 Administrator Bazy Danych

Poniżej przedstawiono przypadki użycia dotyczące Administratora Bazy Danych.

3.4.1 UC07: Tworzenie Grupy Docelowej

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

3.4.2 UC08: Edycja Grupy Docelowej

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

3.5 Wszyscy Użytkownicy

Poniżej przedstawiono przypadki użycia dotyczące wszystkich Użytkowników.

3.5.1 UC09: Logowanie do Systemu

Przypadek użycia: UC09: Logowanie do Systemu
Aktorzy: Użytkownik (Ankieter, Administrator, Administrator Bazy Danych, 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 charakterystyki oraz wymagania pozafunkcjonalne obowiązujące dla systemu. Ponadto zostanie podjęta próba weryfikacji, które wymagania udało się spełnić i jakie są perspektywy rozwoju

Rozdział 5

Architektura systemu

5.1 Wstęp

Wprowadzenie, bardzo ogólny opis.

5.2 Opis ogólny architektury – Marketecture

Tutaj rysunek i opis marketecture.

5.3 Perspektywy architektoniczne

5.3.1 Perspektywa fizyczna

Rysunek wraz z opisem.

5.3.2 Perspektywa logiczna

Rysunek wraz z opisem.

5.3.3 Perspektywa implemetancyjna

Rysunek wraz z opisem. Można tutaj też umieścić perspektywę kodu.

5.3.4 Perspektywa procesu (równoległości)

Rysunek wraz z opisem.

5.4 Decyzje projektowe

Tutaj piszemy o decyzjach projektowych, związkach pomiędzy nimi oraz innych związanych sprawach.

5.5 Wykorzystane technologie

Opis wykorzystywanych technologii (COTS).

5.6 Schemat bazy danych

Schemat bazy danych, ze względu na objętość, można umieścić w którymś dodatku.

Rozdział 6

Opis implementacji

6.1 Wstęp

6.2 Użyte technologie

W poniższym rozdziale postaram się wyjaśnić wybór odpowiednich technologii użytych w pracy.

PHP

PHP to język w którym napisany jest system *Moodle*, dlatego też jest językiem większości naszych rozszerzeń.

PHPUnit

Ze względu na fakt, iż programiści *Moodle'a* testują jednostkowo kod tejże platformy z użyciem *PHPUnit* zdecydowaliśmy się na to samo. *Moodle* udostępnia dwie klasy do testowania, tj. *basic_testcase* i *advanced_testcase*, przy czym ta druga służy do testów, które wchodzą w interakcję z bazą danych.

Selenium

Selenium to szybko rozwijające się narzędzie do testów akceptacyjnych. Był to naturalny wybór zwłaszcza, że zostało ono nam przybliżone już na zajęciach z Inżynierii Oprogramowania na 6. semestrze studiów.

PostgreSQL

Ze względu na wymaganie pozafunkcjonalne wybraliśmy system zarządzania bazą danych *PostgreSQL*.

Eclipse

Ze względu na darmową dostępność jako zintegrowane środowisko programowania wybraliśmy *Eclipse* z dodatkiem *PHP Development Tools*.

SVN

Ze względu na wymaganie pozafunkcjonalne wybraliśmy system zarządzania wersjami Subversion.

Redmine

Już od początku pracy korzystaliśmy z systemu zarządzania projektami *Redmine*. To narzędzie bardzo przydatne w wymianie informacji pomiędzy członkami zespołu, integrujące się m.in. z repozytorium kodu.

JasperReports

Ze względu na wymaganie pozafunkcjonalne zdecydowaliśmy się skorzystać z mechanizmów raportowania oferowanych przez *JasperReports*.

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 użytkownika zapobiega frustracji, związanej z częstym przeładowywaniem całej strony.

6.3 Ogólna struktura projektu

Sekcja druga.

6.4 Interfejs

6.4.1 Wprowadzenie

Jedną z części pracy było zaprojektowanie graficznego interfejsu użytkownika. Głównym problemem jaki się pojawił, był wybór odpowiedniego narzędzia. Celem jaki postawiono, była maksymalna zgodność projektowanych elementów z różnymi wersjami *Moodle* – zarówno wcześniejszymi, jak i późniejszymi. Zdecydowano, aby starać się korzystać z gotowych interfejsów programowania aplikacji *(API)* dostarczonych przez *Moodle*, tj. *Page API*, *Form API*, oraz *Access API*. Wszystkie interfejsy są napisane przy użyciu języka PHP – są wykonywane po stronie serwera. Konieczne okazało się też wykonanie niektórych skryptów po stronie klienta. Dlatego w projekcie wykorzystano również język skryptowy *Java Script*.

6.5 Logika (back-end)

Jednym z zadań w ramach pracy było zaprogramowanie odpowiedniej logiki biznesowej rozwiązującej zadania stawiane przed zaprojektowanym systemem. Najważniejszym zadaniem z perspektywy back-end'u jest interakcja z bazą danych. Poza tym system posiada: procesor zadań wykonywanych w tle oparty na *cron*; moduł odpowiadający za komunikację z systemem uczelanianym *ePoczta*; moduł logowania zdarzeń. W trakcie implementacji zdecydowaliśmy się nie tworzyć osobnego mechanizmu do przechowywania ustawień w bazie danych i skorzystaliśmy z istniejącego już w *Moodle*. Jednym z wymagań pozafunkcjonalnych było wykorzystanie bazy danych *PostgreSQL*. Platforma *Moodle* korzysta z mechanizmu *XMLDB*, co pozwala na ominięcie wielu problemów pojawiających się przy migracjach pomiędzy różnymi systemami baz danych. Niestety kosztem wykorzystania tego mechanizmu jest konieczność pracy z interfejsami programowania aplikacji dostarczanymi przez platformę *Moodle*, m.in. *Data manipulation API*.

6.6 Powiązanie back-endu z interfejsem

Dalsze opisy.

Rozdział 7

Zapewnianie jakości i konserwacja systemu

7.1 Testy i weryfikacja jakości oprogramowania

7.1.1 Testy jednostkowe

7.1.2 Testy integracyjne

7.1.3 Testy akceptacyjne

7.1.4 Inne metody zapewniania jakości

7.2 Sposób uruchomienia i działania systemu

Rozdział 8

Zebrane doświadczenia

8.1 Problemy i ich rozwiązania

8.1.1 Maciej Trojan

Inicjalizacja bazy danych

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

Do zaimportowania bazy danych przygotowanej przez architekta wykorzystano narzędzie, wbudowane w platformę *Moodle*, *XMLDB*, które gwarantuje bezobsługową instalację modułu w przyszłości. Jak się później okazało narzędzie to posiada błąd, który uniemożliwia zaimportowania kluczy obcych. Wymagało to od programistów ręcznego utworzenia wszystkich kluczy obcych, przewidzianych przez architekta.

Instalacja modułu

Postanowiono, że wraz z instalacją modułu powinień automatycznie tworzyć się odpowiedni *kurs* związany jedynie z modułem *iQuest*. Zmniejsza to potrzebny czas na przygotowanie platformy do użytku, oraz zapobiega pomyłkom związanych z ręcznym tworzeniem i konfiguracji *kursu*.

Niestety okazało się, że podejście to uniemożliwia instalację modułu jednocześnie z całą platformą *Moodle*, ponieważ dodatkowe moduły instalowane są przed mechanizmami pozwalającymi na tworzenie *kursu*. Doinstalowanie modułu do zainstalowanej platformy nie powoduje żadnych komplikacji.

8.1.2 Krzysztof Urbaniak

Wprowadzenie

Na początku tego rozdziału należy przypomnieć, główne pojęcia związane z korzystaniem z platformy *Moodle*.

Po zalogowaniu do systemu użytkownik musi wybrać *kurs*. Kurs jest największą częścią *Moodle* i przeważnie kojarzony jest z przedmiotem. Na kurs składa się kilka lub kilkanaście *sekcji*. Sekcja odpowiada najczęściej konkretnym zajęciom, jest związana z jakimś wydarzeniem lub tygodniem. Najmniejszą jednostką w *Moodle* jest *aktywność*. Aktywność to podstawowy typ modułów rozszerzających funkcjonalność *Moodle*. Aktywnościami są np. *Forum*, *Głosowanie*, *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.

Formater kursu decyduje o ułożeniu i wyświetlaniu elementów na stronie.

Różnica pomiędzy tym, czy użytkownik znajduje się lub nie, w module lub w kursie, to *kontekst*.

Jako, że rozdział ten tyczy się grafiki, warto wspomnieć, że elementy na stronie ułożone są w następujący sposób:

Na ekranie wyświetlana jest lista sekcji.

Wewnątrz każdej sekcji znajduje się lista badań. Pod nią drukowana jest lista materiałów.

Formularze

Projektując graficzny interfejs użytkownika, prędzej, czy później pojawia się potrzeba wyboru narzędzia do projektowania formularzy. Rozważano kilka możliwości. Po pierwsze, użycie po prostu czystego języka *HTML*. Drugą opcją było użycie wbudowanych w *Moodle API – Form API* lub *Output API*. Jako trzecią możliwość rozważano użycie zewnętrznych *API* – nie związanych z *Moodle*.

Odrzucono pomysł pierwszy oraz trzeci. Pomysł pierwszy wydał się zbyt pracochłonny. Z uwagi na dość mocno ograniczony czas, niemądrze byłoby programować coś od początku samemu, skoro ktoś inny już coś podobnego napisał. Nawet kosztem tego, że interfejs nie wyglądał dokładnie tak, jak go zaprojektowano. Ważne, aby był kompletny. Wariant trzeci był nie zgodny z założeniem mówiącym o korzystaniu z interfejsów *Moodle* tam, gdzie to możliwe. Nie chciano, żeby użytkownik poczuł, że programowana wtyczka nie jest częścią systemu *Moodle*. Co więcej, interfejs odbiegałby od wyglądu *Moodle* i mógłby zostać oceniony jako nieintuicyjny. Zdecydowano się korzystać z *Output API*.

Output API to zestaw funkcji, wprowadzonych od wersji *Moodle* 2.0. Umożliwiają one wstawianie na stronę standardowych elementów formularza, takich jak: etykiety, przyciski, linki, tabele etc. Niestety, z nieznanych powodów, to wspaniale wyposażone, z pełną dokumentacją i w pełni przydatne *API* zostało usunięte z *Moodle* już w wersji 2.2. Co ciekawe niektórych elementów można nadal używać, lecz nie znaleziono dokumentu, który mówiłby o tym jakie dokładnie części tego *API* można używać, a które zostały wycofane.

Następcą *Output API* został *Form API*. Ku rozczarowaniu zespołu *Form API* ma nieco uboższą dokumentację. Zdarza się, że w funkcji są omówione np. tylko trzy pierwsze argumenty, natomiast reszta jest pominięta, tak jakby nie istniała. *Form API* różni się też od poprzednich *API* tym, że jest oparte na modelu obiektowym. Posiada także mechanizm pozwalający na weryfikację danych, wprowadzanych przez użytkownika. Mechanizm ten napisany jest z użyciem języka *JavaScript*, oznacza to, że walidacja przeprowadzana jest po stronie klienta, a dopiero poprawne dane przesyłane są do serwera.

Podsumowując, mimo niekompletnej dokumentacji, jako narzędzie implementacji formularzy, zostało wybrane *Form API*. Posiada ono szereg przydatnych funkcji, pozwalających na definiowanie standardowych elementów każdego formularza. Nie bez znaczenia są też funkcje pozwalające na walidację wprowadzanych danych. Oczywiście nie wszystkie potrzebne elementy udało się napisać używając jedynie *API*. Korzystano wtedy z czystego języka *HTML*.

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*. Możliwość można rozumieć jako prawo do wykonania, określonego przez programistę, fragmentu kodu.

Formater kursu

Jednym z problemów jakie napotkano, była konieczność wyświetlania respondentom i ankieterom tylko określonych modułów. Ankieterzy powinni zobaczyć tylko te badania, które utworzyli oraz inne aktywności, zasoby. Respondenci powinni zobaczyć tylko te badania, w których mogą wypełnić ankietę, a także materiały, które mają prawo wyświetlać.

Do rozwiązania problemu zdecydowano się użyć formatera kursu. Narzędzie to, jako integralna część *Moodle*, było najlepszym rozwiązaniem z możliwych. Niestety wkrótce okazało się, że brak dokumentacji, a także dyskusji na temat tego narzędzia w internecie utrudnia wykonanie zadania. Całą pracę wejścia, polegającą na poznaniu narzędzia, wykonano studiując kod źródłowy domyślnych formaterów *Moodle*.

Pierwszy plan zakładał wyświetlanie użytkownikowi dwóch sekcji. Jedną z odpowiednimi badaniami, drugą z materiałami. Należało także ograniczyć ankieterowi możliwość dodawania w pierwszej sekcji modułów innych niż iQuest, a w drugiej sekcji uniemożliwić dodanie tej aktywności. Niestety wyszło na jaw, że jest to nieosiągalne bez ingerowania w wewnętrzny kod platformy *Moodle*.

Zostało to spowodowane uaktualnieniami w *Moodle*. Kod *PHP* wyświetlania typów modułów, zostaje nadpisywany przez *JavaScript*. W taki 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, a nie może decydować o tym, co można dodać 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 może wyświetlić. Co więcej, będą one odpowiednio posegregowane, aby użytkownik szybko mógł znaleźć to, czym jest zainteresowany.

Tworzenie badania

Kolejną trudnością w projekcie, było połączenie go z systemem *Moodle*. Głównie sprowadzało się to do tego, aby tak wykorzystać interfejs graficzny oferowany przez *Moodle*, żeby użytkownik nie miał poczucia zmiany systemu. Zarówno wygląd, jak i dodawanie różnych funkcji, powinny być zgodne ze standardem *Moodle*. Dzięki takiemu podejściu, osoba, korzystająca wcześniej z *Moodle*, a pragnąca używać wtyczki *iQuest*, nie musiała zmieniać swoich przyzwyczajeń. Co więcej, w projekcie duży nacisk został postawiony na nie zniechęcanie respondentów do wypełnienia ankiety, co było dodatkową motywacją do zaprojektowania przyjaznego użytkownikom interfejsu.

Wstępna wersja interfejsu, zaprojektowana przez architekta, wyglądała następująco. Najpierw ankieter wyrażał chęć utworzenia nowego badania, klikając odpowiedni przycisk. Następnie mógł dodać do badania ankietę z katalogu, ewentualnie utworzyć nową. Na kolejnej stronie ankieter definiował szczegóły badania, takie jak: nazwa, grupa docelowa, czas rozpoczęcia i zakończenia etc. Nie było możliwe, aby zaimplementować to w powyższy sposób.

Problem stwarzało dodawanie ankiety wewnątrz tworzenia badania. Gdy tworzenie badania zaczyna się od zdefiniowania ankiety, to nie można od razu dodać jej do odpowiedniego badania, ponieważ to badanie jeszcze nie istnieje. W takim wypadku należałoby przechowywać gdzieś

informację, że po utworzeniu badania ma dodać się do niego ankieta. Na przykład można w tym celu użyć dodatkowy parametr w adresie *URL*. Dodatkowo, w *Moodle*, przy kreowaniu nowego modułu, użytkownikowi wyświetlany jest domyślny formularz, w którym podaje się parametry potrzebne do zbudowania instancji modułu. Przyjmując, że badanie jest kojarzone z modułem, nie ma możliwości, aby przed zakończeniem tworzenia badania wstawić dodatkowy formularz.

Rozwiązaniem jest prosta zamiana kolejności tworzenia badania oraz ankiety. Najpierw użytkownik tworzy badanie, czyli moduł. Dopiero wówczas ma możliwość sporządzenia ankiety. Podejście to ma kilka zalet: jest to zgodne z procedurą *Moodle*, a co za tym idzie, bardziej intuicyjne dla użytkownika obeznanego z *Moodle* oraz pozwala na łatwe dodanie ankiety do badania. Ponadto ankieter może zrezygnować z komponowania ankiety przy kreowaniu badania. Nie sposób nie zgodzić się z tezą, że jest to jedyny słuszny sposób dodawania nowego modułu *iQuest*.

Tworzenie ankiety

Przy tworzeniu ankiety powstał 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 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 pliku *mod.php* trafiały dane z kontekstu modułu. Drugi plik zajmował się przetwarzaniem danych z kontekstu kursu. Taki podział gwarantował większy porządek w kodzie źródłowym. Porządek był ważny, ponieważ w *Moodle* nie jest zgodny z wzorcem Model-View-Controller. W związku z tym ważne 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, zapisywano je do bazy danych. Później zwracano sterowanie do plików *mod.php* albo *course.php* 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 zmian. Przy dobrym zarządzaniu kodem mała zmiana wymaga małych zmian.

Inwencja programistów

W trakcie rozwoju oprogramowania pojawiło się kilka małych niejasności, które programiści musieli rozwiązać sami. Kilka razy wykazali również inicjatywę i zaproponowali rozwiązania, które stały się ostatecznie częścią projektu.

Pierwszą ideą było zagospodarowanie przestrzeni w widoku badania. Po utworzeniu badania i dodania 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. I tu pojawił się problem. Żeby spełnić wymagania, na stronie wystarczyło pokazać odnośniki: „edytuj" i „usuń z badania". Praktycznie cała stron pozostawała pusta. Sytuacja taka jest niedopuszczalna, bo na pewno istniały jakieś przydatne informacje, które można było w tym miejscu wyświetlić. Wykoncypowano, że najbardziej naturalnie będzie pokazać w tym miejscu statystyki dla badania.

W pierwszej wersji zaimplementowano tylko proste statystyki. Można się z nich dowiedzieć: ile czasu zostało do zakończenia badania, ile osób liczy grupa docelowa oraz ile osób już odpowiedziało i poznać wartość procentową. Następnie dodano kolejną tabelkę ze statystykami. Wyświetla

się gdy choć jedna osoba odpowie na któreś pytanie. Możemy w niej zobaczyć jak kształtowały się odpowiedzi, w pytaniach zamkniętych, na które odpowiedziano. Nie zdecydowano się wyświetlać odpowiedzi na pytania otwarte ze względu na ich różnorodność, a więc ilość miejsca, które zajmowały. Ideą tabelki było pokazanie skróconych informacji o badaniu. Cały, dokładny, rozbudowany raport można wygenerować z użyciem systemu *Jasper Report*.

Pozyskanie i podliczenie odpowiedzi na dane pytanie wiązało się z wymyśleniem algorytmu. Teoretycznie najprostszym rozwiązaniem byłoby, dla każdej dozwolonej odpowiedzi na pytanie zamknięte, sprawdzenie liczności krotek w tabeli *answers*. To rozwiązanie jest jednak nieoptymalne, bo wiąże się z wielokrotnym odwoływaniem się do bazy danych. Lepiej pozwolić bazie danych samej zoptymalizować odwołania do tablic. Tak postąpiono w tym przypadku. Przy użyciu wyrażenia „GROUP BY" opracowano zapytanie, które od razu zwracało liczbę odpowiedzi respondentów na możliwą odpowiedź. Takie podejście gwarantuje szybsze wykonanie algorytmu.

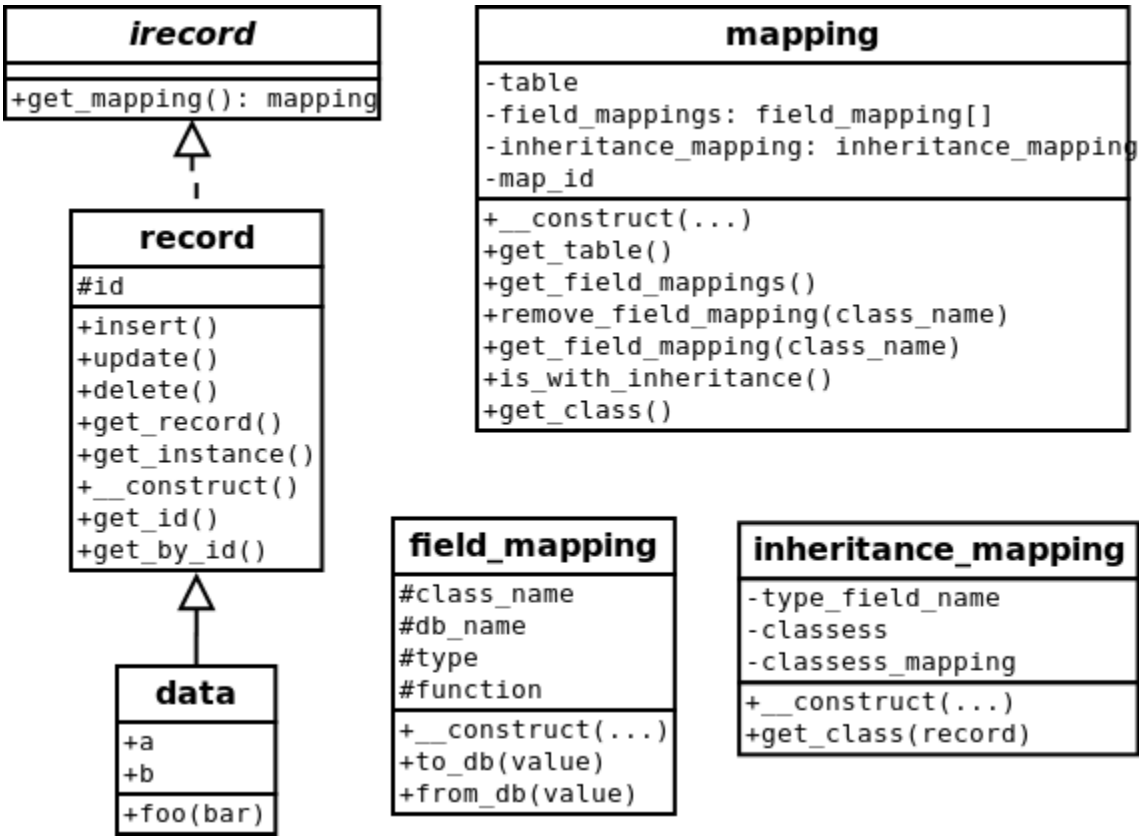
Kolejnym pomysłem programistów było dodanie kilku przycisków. Zarówno w katalogu jak i widoku badania umieszczono przycisk „pokaż". Służy on do wyświetlenia ankiety tak samo jak widzi ją respondent. Ankieter może zobaczyć, układ pytań, łatwiej mu zdecydować o dodaniu kolejnej strony. Pojawił się także przycisk pozwalający na dodanie nowej ankiety, będąc w widoku katalogu. Znajduję się on zarówno na dole jak i na górze tabelki, aby nie było konieczności przewijania ekranu.

W założeniach projektu ustalono, że odpowiedź jest nieedytowalna. Dodano udoskonalenie, które polegało na tym, że respondent nie musi od razu wypełnić całej ankiet. Może to robić stopniowo. Za każdym razem jednak zostaną mu wyświetlone tylko te pytania, na które jeszcze nie odpowiedział. 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 nie pokaże się w kursie.

8.1.3 Łukasz Wieczorek

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 podjąłem się zadania stworzenia klas, które wzorem nowoczesnych systemów *ORM* uproszczą projektowanie nowych klas reprezentujących dane. Ze względu na silną integrację z mechanizmami *Moodle* w grę nie wchodziły gotowe rozwiązania. Autorskie rozwiązanie korzysta z mechanizmów *Moodle Data manipulation API* oraz mechanizmu refleksji języka PHP, by pozwolić programiście korzystającego z tego rozwiązania na proste pobieranie i manipulację obiektami przechowywanymi w bazie danych. Diagram UML przedstawia się następująco: Klasa danych dziedziczy po klasie *record* oraz implementuje 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 mapowania dla atrybutów, tj. nazwy w klasie i bazie danych oraz typ, który zadecyduje o metodzie pobrania/zapisania danej (typem może być także klasa potomna klasy *record*). W przypadku, gdy mamy do czynienia z dziedziczeniem wystarczy zdefiniować przy mapowaniu sposób obsługi dziedziczenia (m.in. jakie pole określa typ klasy). Najważniejszy kod znajduje się



Rysunek 8.1: iQuest ORM

w metodzie *get_instance*, która pobiera konstruktor danej klasy, poprzez refleksję tworzy obiekt z wyłuskanymi z bazy danych parametrami wymaganymi przy jego tworzeniu oraz ustawia resztę pól pobranych z bazy danych. 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 (*Don't repeat yourself*). Projektowałem je, mając na uwadze rozwiązanie, z którym miałem wcześniej styczność, tj. implementację *ActiveRecord* z *Ruby on Rails*. W trakcie pracy nad projektem doceniłem stosowanie konwencji nazewniczych, których obecność znacząco upraszcza projektowanie klas mapujących dane.

Rozdział 9

Zakończenie

9.1 Podsumowanie

9.2 Propozycja dalszych prac

Dodatek A

Informacje uzupełniające

A.1 Wkład poszczególnych osób do przedsięwzięcia

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 część implementacyjną systemu została przedstawiona poniżej:

Krzysztof Marian Borowiak

- Testy jednostkowe
- Testy akceptacyjne
- Dokumentacja dla Użytkownika Końcowego

Maciej Trojan

- Interfejs użytkownika
- Powiązanie interfejsu z back-endem
- Obsługa Bazy Danych

Krzysztof Urbaniak

- Interfejs użytkownika
- Powiązanie interfejsu z back-endem

- Obsługa Bazy Danych

Łukasz Wieczorek

- Logika (back-end)
- Powiązanie interfejsu z back-endem

A.2 Wykaz użytych narzędzi

A.3 Zawartość płyty CD

Do dokumentu załączono płytę CD o następującej zawartości:

- Zawartość 1
- Zawartość 2
- Zawartość 3
- ...

Dodatek B

Wygląd aplikacji

Dodatek C

Schemat bazy danych

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",
}
```