

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

1	Wprowadzenie	1
1.1	Opis problemu i koncepcja jego rozwiązania	1
1.1.1	Problem	1
1.1.2	Proponowane rozwiązanie	2
1.2	Ograniczenia i zagrożenia dla projektu	2
1.3	Cele projektu	3
1.4	Omówienie pracy	3
2	Opis procesów biznesowych	4
2.1	Aktorzy	4
2.2	Obiekty biznesowe	4
2.2.1	Badanie	4
2.2.2	Ankieta	5
2.2.3	Grupa Docelowa	5
2.2.4	Raport	5
2.3	Biznesowe przypadki użycia	6
2.3.1	BC01: Zbieranie informacji o Absolwentach	6
2.3.2	BC02: Zbieranie informacji o Studentach	6
2.3.3	BC03: Zarządzanie Grupami Docelowymi	7
3	Wymagania funkcjonalne	8
3.1	Diagram przypadków użycia	8
3.2	Ankieter	9
3.2.1	UC01: Stworzenie Ankiety	10
3.2.2	UC02: Edycja Ankiety	10
3.2.3	UC03: Wybór Grupy Docelowej	11
3.2.4	UC04: Uruchomienie Badania	11
3.2.5	UC06: Sprawdzenie wyników	12
3.3	Respondent	12
3.3.1	UC05: Udzielenie odpowiedzi	12
3.4	Administrator Bazy Danych	12
3.4.1	UC07: Tworzenie Grupy Docelowej	13
3.4.2	UC08: Edycja Grupy Docelowej	13
3.5	Wszyscy Użytkownicy	13
3.5.1	UC09: Logowanie do Systemu	14
4	Wymagania pozafunkcjonalne	15
4.1	Wstęp	15

4.2	Charakterystyki oprogramowania	15
4.3	Wymagania pozafunkcjonalne i ich weryfikacja	17
5	Architektura systemu	21
5.1	Wstęp	21
5.2	Opis ogólny architektury – Marketecture	22
5.3	Perspektywy architektoniczne	23
5.3.1	Perspektywa fizyczna	23
5.3.2	Perspektywa logiczna	24
	iQuest Survey Activity Plugin	24
	iQuest Course Format Plugin	24
	eDziekanat Connector oraz ePoczta Connector	25
	Background Task Scheduler and Executor	25
	eKonto Authentication Plugin	25
5.3.3	Perspektywa implemetacyjna	25
5.3.4	Perspektywa procesu (równoległości)	25
5.4	Decyzje projektowe	25
5.5	Wykorzystane technologie	26
5.6	Schemat bazy danych	26
6	Opis implementacji	27
6.1	Wstęp	27
6.2	Napotkane problemy i ich rozwiązania	27
6.3	Użyte technologie	27
6.3.1	Moodle	27
6.3.2	PHP	27
6.3.3	PHPUnit	27
6.3.4	Selenium	27
6.3.5	PostgreSQL	28
6.3.6	Eclipse IDE	28
6.3.7	SVN	28
6.3.8	Redmine	28
6.3.9	JasperReports	28
6.3.10	JavaScript	28
6.4	Ogólna struktura projektu	28
6.5	Interfejs	28
6.6	Logika (back-end)	29
6.6.1	Raporty	29
6.6.2	Moduły uwierzytelniania	30
6.6.3	Moduły dla serwisów zewnętrznych	30
6.7	Powiązanie back-endu z interfejsem	31
7	Zapewnianie jakości i konserwacja systemu	32
7.1	Testy i weryfikacja jakości oprogramowania	32
7.1.1	Testy jednostkowe	32
7.1.2	Testy integracyjne	32
7.1.3	Testy akceptacyjne	32

7.1.4	Inne metody zapewniania jakości	32
7.2	Sposób uruchomienia i działania systemu	32
8	Zebrane doświadczenia	33
8.1	Problemy i ich rozwiązania	33
8.1.1	Maciej Trojan	33
	Inicjalizacja bazy danych	33
	Instalacja modułu	33
8.1.2	Krzysztof Urbaniak	33
	Wprowadzenie	33
	Formularze	34
	Role	35
	Formater kursu	35
	Tworzenie badania	35
	Tworzenie ankiety	36
	Inwencja programistów	36
	Hierarchia CSS	37
8.1.3	Łukasz Wiczorek	38
	Mapowanie obiektowo-relacyjne	38
9	Zakończenie	40
9.1	Podsumowanie	40
9.2	Propozycja dalszych prac	40
A	Informacje uzupełniające	41
A.1	Wkład poszczególnych osób do przedsięwzięcia	41
A.2	Wykaz użytych narzędzi	42
A.3	Zawartość płyty CD	42
B	Wygląd aplikacji	43
C	Schemat bazy danych	44

Rozdział 1

Wprowadzenie

1.1 Opis problemu i koncepcja jego rozwiązania

1.1.1 Problem

Podstawą każdej działalności jest warunkująca ją potrzeba. Niemożliwą jest jednak analiza potrzeb, bez wcześniejszego ich zbadania. Najłatwiejszą i najbardziej rozpowszechnioną metodą pozyskiwania wiedzy na jakiś temat jest pytanie o to osób związanych ze sprawą. Kiedy pytań jest wiele, a liczba potencjalnych respondentów osiąga niewyobrażalny poziom, zadawanie pytań wymaga usystematyzowania, podobnie, jak metodyka ich dostarczania do odpowiadających, czy gromadzenia odpowiedzi. Takie rozumowanie prowadzi do niezwyklego narzędzia badań, pozwalającego na gromadzenie danych - ankiety.

Ankieta pozwala na nieinwazyjne, anonimowe i masowe pozyskiwanie opinii respondentów na zadany temat, a dzięki zastosowaniu różnych metod statystycznych, także na ocenę zdania całej populacji. W kontekście jednostki szkolnictwa wyższego, jaką jest Politechnika Poznańska, ankietowanie jest niezbędne do zapewnienia prawidłowej pracy. Bez informacji zwrotnej (ang. *feedback*) praktycznie nie ma możliwości sprawdzenia poprawności podejmowanych działań, co w efekcie zawsze będzie prowadzić do spiętrzającej się fali problemów. Przykładem może być sytuacja, gdy nie posiadając wiedzy o nieodpowiednim, zdaniem studentów i prowadzących zajęcia, wyposażeniu sali laboratoryjnej, władze uczelni ustalają harmonogram zajęć, zwiększający obciążenie tego pomieszczenia. Efektem takiego działania byłby spadek jakości kształcenia studentów korzystających z tego laboratorium.

W ostatnich latach¹ Politechnika Poznańska posiadała szerokie spektrum narzędzi do pozyskiwania wiedzy o swoim działaniu i oferowanych usługach. Niestety, gros z nich wzajemnie się wykluczał. Przykładowy student na Wydziale Informatyki² poddawany był ankietom: elektronicznym na poziomie Uczelni, jak i Wydziału, „papierowym” na poziomie Uczelni oraz Samorządu Studentów, a ponadto dodatkowej ankietyzacji w ramach niektórych zajęć dydaktycznych. Co gorsza, żaden spośród tych „systemów” ankietyzacji nie posiadał odpowiednich zabezpieczeń, wymaganych do zapewnienia wymierności ich wyników (jak np. zapewnienie, aby nikt nie wziął w ankiecie udziału więcej niż jeden raz, czy uniemożliwienie wypełnienia ankiety przeznaczonej dla innego wydziału, kierunku). Brak wymiernych efektów udziału w tych badaniach, podobnie jak ich mnogość i dezorientacja, kto tak naprawdę pozyskuje informacje, działały tu na niekorzyść an-

¹Mowa o okresie 2008-2013, który jest znany autorom niniejszej Pracy Dyplomowej.

²Do 2010 roku – Wydziale Informatyki i Zarządzania

kietujących. Ważnym jest też fakt, że istniała tylko jedna grupa docelowa – studenci Politechniki Poznańskiej.

Ustawa o Szkolnictwie Wyższym wymaga od uczelni wyższych badania nie tylko aktualnej społeczności studenckiej, lecz także jej absolwentów. Sprawdzane powinny być takie czynniki, jak rozwój karier czy satysfakcja z zapewnianych usług. Niestety, na Politechnice Poznańskiej brak jest odpowiednich narzędzi do prowadzenia tego rodzaju badań. Nawiązując więc do wcześniejszych twierdzeń, potencjalna katastrofa (jak niezauważony spadek jakości usług, prowadzący do zmniejszenia satysfakcji odbiorców ze względu na brak podjęcia odpowiednich działań i w efekcie spadku prestiżu Uczelni) to jedynie kwestia czasu.

1.1.2 Proponowane rozwiązanie

Szczęśliwie, problemu sprawnego ankietowania nie można porównywać z węzłem gordyjskim. Najłatwiejszym i – w opinii autorów niniejszego dokumentu – najlepiej popartym logicznymi argumentami rozwiązaniem, jest stworzenie jednego, jednolitego, globalnego i prostego w obsłudze systemu prowadzenia badań i ankiet dla różnych grup docelowych, obejmujących wszystkich aktualnych i byłych studentów Politechniki Poznańskiej. Aby zapewnić dostępność oraz prostotę wdrożenia systemu, który na to pozwoli, zostanie on wykonany za pomocą technologii internetowych, co umożliwi też jego obsługę z użyciem dowolnej popularnej przeglądarki internetowej dostępnej na rynku.

W trakcie analizy problemu ustalono, że koniecznym będzie także zapewnienie swojego rodzaju „zachęty” dla potencjalnych Respondentów – o ile bowiem studentowi może zależeć na rozwoju jego uczelni, o tyle absolwent nie będzie czerpał z tego tytułu żadnych wymiernych korzyści. Wybrane dla iQuest rozwiązanie obejmuje więc też element zachęcający do korzystania z systemu, jakim jest dostęp dla Respondentów do unikalnych materiałów dydaktyczno-naukowych.

Aby zapewnić, że pozyskane dane są prawidłowe, w proponowanym rozwiązaniu znajdą się systemy autoryzacji osób uzyskujących dostęp do badań i ankiet. Jest to jednak jedyny element wymagający sprawdzenia tożsamości Respondenta – same wyniki będą całkowicie anonimowe.

1.2 Ograniczenia i zagrożenia dla projektu

Największym ograniczeniem, a zarazem zagrożeniem dla projektu jest kwestia zachęty studentów i absolwentów do korzystania z nowego systemu ankietowania. Popularność całego systemu zależy w głównej mierze właśnie od tego, jakie materiały będą za jego pośrednictwem udostępniane oraz jakie będą warunki uzyskania dostępu do nich. Kolejnym ograniczeniem jest wyznaczony z góry termin zakończenia prac. Na projektowanie, implementację i wdrożenie rozwiązania przewidziany jest jedynie okres od września 2012, do lutego 2013. Uzupełniając tę kwestię o fakt braku wcześniejszego doświadczenia zespołu przy realizacji projektów opartych na technologiach internetowych, spełnienie wszystkich wymagań związanych z projektem może okazać się utrudnione.

Ostatecznie jednak, w fazie tworzenia, kwestia zachęty dla Respondentów może być potraktowana jako poboczna. Kwestię ograniczonego czasu można nadrobić, stosując odpowiednie metody

zarządzania czasem, zaś brak doświadczenia – nadrobić w wolnych chwilach. Można więc uznać, że projekt nie posiada żadnych ważkich ograniczeń, czy zagrożeń, które mogłyby uniemożliwić jego realizację.

1.3 Cele projektu

Celem projektu *iQuest* jest zbudowanie systemu umożliwiającego łatwe przeprowadzanie ankiet wśród studentów i absolwentów uczelni. System ten powinien:

- oferować dużą elastyczność:
 - przy definiowaniu różnorodnych ankiet
 - przy tworzeniu i hierarchizacji grup respondentów
 - przy zachęcaniu do uczestnictwa w niej przez potencjalnych Respondentów
- oferować rozbudowane możliwości raportowania
- ujednolicić uczelniany system pozyskiwania informacji
- odciążyć pracowników uczelni oraz Samorząd Studentów z obowiązków związanych z przeprowadzaniem konwencjonalnych („papierowych”) ankiet

1.4 Omówienie pracy

Do uzupełnienia po ukończeniu reszty pracy

Rozdział 2

Opis procesów biznesowych

System *iQuest*, będący przedmiotem niniejszej Pracy Dyplomowej, jest nie tylko projektem edukacyjnym, lecz również pełnoprawnym zadaniem biznesowym. Wykonywany dla Dziekana 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.

2.1 Aktorzy

W systemie zdefiniowani są następujący aktorzy:

- 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

W ramach systemu iQuest, zdefiniowane są cztery obiekty biznesowe. Mowa o Badaniu, Ankiecie, Grupie Docelowej i Raporcie.

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. Obejmują one dwa główne zagadnienia: zbieranie informacji oraz zarządzanie Grupami Docelowymi.

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
<ol style="list-style-type: none"> 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
<ol style="list-style-type: none"> 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
<ol style="list-style-type: none">1. Ankieter zgłasza potrzebę stworzenia Grupy Docelowej Administratorowi Bazy Danych2. Administrator Bazy Danych podaje nazwę Grupy Docelowej, którą zamierza utworzyć3. Administrator Bazy Danych dodaje/usuwa członków Grupy Docelowej4. Administrator Bazy Danych potwierdza chęć stworzenia Grupy Docelowej5. System tworzy Grupę Docelową6. Ankieter może korzystać z Grupy Docelowej

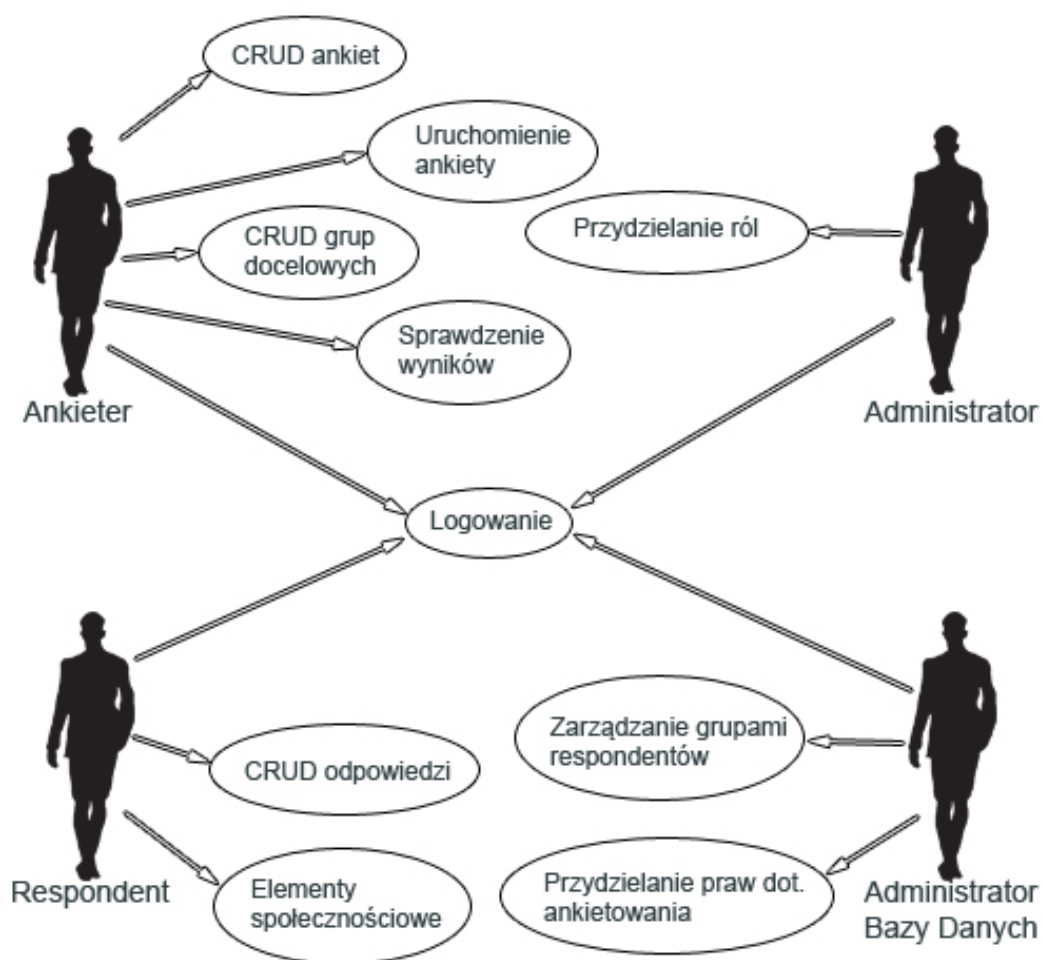
Rozdział 3

Wymagania funkcjonalne

Celem sprawnej realizacji celów warunkujących powstanie systemu iQuest, musi on spełniać szereg wymagań funkcjonalnych. Obejmuje to m.in. tworzenie i prowadzenie (zarówno ze strony Ankietera, jak i Respondenta) Badań i Ankiet, a także analizę 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. Koniecznością jest bowiem prawidłowa autoryzacja użytkowników systemu, jak też współpraca z innymi systemami Uczelni.

3.1 Diagram przypadków użycia

Na rysunku 3.1 (poniżej) 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 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.



RYSUNEK 3.1: Diagram przypadków użycia

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
<ol style="list-style-type: none"> 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
<ol style="list-style-type: none"> 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: <ol style="list-style-type: none"> 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
<ol style="list-style-type: none"> 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
<ol style="list-style-type: none"> 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 Ankietą 1.A.1 System informuje, że Ankietą 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 wysłać 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
<ol style="list-style-type: none">1. System wyświetla Użytkownikowi formularz logowania2. Użytkownik podaje login lub adres e-mail oraz hasło3. System uwierzytelnia Użytkownika
Rozszerzenia
<ol style="list-style-type: none">2.A Użytkownik chce się zalogować przy pomocy eKonta<ol style="list-style-type: none">2.A.1 Użytkownik wybiera opcję logowania przez eKonto2.A.2 System przekierowuje Użytkownika na stronę logowania przez eKonto (eLogin)2.A.3 Użytkownik wprowadza dane logowania do eKonta2.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. Dodatkowo, podjęta zostaje tu próba weryfikacji, które wymagania udało się spełnić i jakie są perspektywy dalszego rozwoju projektu.

4.2 Charakterystyki oprogramowania

Projektując system, pod uwagę brane były następujące charakterystyki:

- Analizowalność
- Autentyczność
- Bezpieczeństwo (wolność od ryzyka)
- Charakterystyka czasowa
- Dostępność personalna
- Dostępność techniczna
- Estetyka interfejsu Użytkownika
- Funkcjonalna poprawność
- Identyfikowalność
- Integralność
- Interoperacyjność
- Kompletność kontekstowa
- Łatwość adaptacji
- Łatwość instalacji
- Łatwość nauczania się
- Łatwość testowania
- Łatwość zamiany

- Łatwość zmiany
- Niezaprzeczalność
- Ochrona użytkownika przed błędami
- Odporność na wady
- Odtwarzalność
- Poufność
- Współlistnienie
- Zużycie zasobów

Wymienione powyżej charakterystyki były rozpatrywane w fazach projektowania i implementacji z zastosowaniem różnych priorytetów. Ciężko w tym miejscu wyznaczyć najważniejsze bądź najtrudniejsze do osiągnięcia wymagania z nimi związane, w szczególności w relacji wzajemnej między nimi. Tym niemniej, można dokonać pewnych obserwacji w klasyfikacji ogólnej.

Bardzo ważna okazała się interoperacyjność, czyli współpraca systemu iQuest z pozostałymi systemami Uczelni. Również odporność na wady oraz odtwarzalność – ze względu na ilość i wagę danych – okazały się znaczące dla projektu.

Nie mniej istotna okazała się estetyka interfejsu użytkownika, która ma przyciągnąć potencjalnych respondentów i umożliwić im wygodę przy podejmowaniu ankiet. Jego intuicyjność sprawia, że użytkownik końcowy nie potrzebuje wiele czasu na naukę użytkowania systemu, nawet, jeśli zrezygnuje z zapoznania się z instrukcją użytkownika końcowego – która również została przygotowana. Znacząco wspomaga on obsługę systemu i pozwala w łatwy sposób tworzyć, prowadzić i wypełniać badania i ankiety.

Zważywszy, że projekt ten jest istotny dla Uczelni, dołożono wszelkich starań, aby spełnić wszystkie wymagania pozafunkcjonalne – nie tylko te opisane powyżej jako ważne. Zadanie to zostało w sporej części wykonane dzięki zastosowaniu sprawdzonego i znanego narzędzia w roli podstawy dla całego systemu. Mowa o platformie Moodle¹. Zastosowanie jej pozwoliło m.in. na zapewnienie pełnej modułowości systemowi, dzięki czemu wprowadzanie jakichkolwiek zmian w dowolnej jego części jest proste i nie wpływa na resztę kodu. Spełniona została także adaptatywność i przenośność systemu, oparta na uniwersalności języka PHP, jak też szerokiej gamie obsługiwanych przez Moodle systemów baz danych.

Zalety wykorzystania tak powszechnej technologii są jednak 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 projektu, nakład pracy wymagany do wdrożenia się celem rozwoju systemu iQuest został zminimalizowany.

W trakcie prac nad projektem, bardzo silny nacisk stawiano na prawidłowe zarządzanie uprawnieniami użytkowników. Są one weryfikowane przy każdym działaniu. Zależą one od przydzielonej

¹ Więcej informacji w rozdziale 6, poświęconym implementacji i zastosowanym w niej technologiom.

im w systemie roli użytkownika. Szczegóły tej kwestii swobodnie definiować może administracja. Ważnym jest, że osoby o roli ankietera mogą modyfikować i analizować jedynie utworzone przez siebie, lub udostępnione im badania. Podobnie działają uprawnienia przydzielone do roli respondenta – pozwalając dokładnie jeden raz odpowiedzieć na pytania w ankietach skierowanych do grup docelowych, do których należy użytkownik o takiej roli.

Charakterystyką, której od samego początku nie traktowano priorytetowo, była szybkość działania systemu. W ogólnym sensie, wyświetlanie danych realizowane jest w relatywnie krótkim czasie, zależy to jednak od tego, gdzie są one przechowywane, czy ich pobranie wymaga połączenia z systemem zewnętrznym oraz czy istnieją one już w bazie danych systemu, czy dopiero należy je tam zamieścić. Tym niemniej, wszelkie związane z tą kwestią wymagania zostały spełnione.

Na sam koniec, warto wspomnieć o kwestii instalacji systemu. Realizowana jest ona trójstopniowo. Pierwszym krokiem jest umieszczenie na serwerze docelowym plików platformy Moodle. W tak przygotowanym środowisku, umieszczane są następnie dane systemu iQuest. Ostatnim krokiem jest wykonanie standardowej instalacji platformy Moodle oraz jej konfiguracja. Aktualizacja systemu przebiega bardzo podobnie, i polega jedynie na zmianie odpowiednich plików na serwerze, w ramach którego działa system, i przeprowadzenia procesu aktualizacji zgodnie ze standardowymi procedurami obowiązującymi dla Moodle. W trakcie testów na urządzeniu przenośnym z procesorem *Intel Atom D525*, 4 GB pamięci operacyjnej oraz małowydajnym dyskiem twardym, pod kontrolą zwirtualizowanego systemu *Ubuntu 12.04 LTE*, cały proces instalacji nie przekraczał 40 minut. Głównym czynnikiem ograniczającym wydajność procesu była szybkość środowiska wirtualnego oraz zastosowanego dysku twardego.

4.3 Wymagania pozafunkcjonalne i ich weryfikacja

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

Charakterystyka	Wymaganie	W	TI
Analizowalność	Komentarze w kodzie źródłowym powinny być w języku angielskim.	M	L
Analizowalność	Kod źródłowy systemu 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 trzymane w systemie muszą być tymi, uzyskanymi w ankietach.	H	L
Bezpieczeństwo (wolność od ryzyka)	Projekt systemu należy poddać analizie z wykorzystaniem metody ATAM.	M	M
Charakterystyka czasowa	Wyświetlanie ankiety powinno odbywać się z maksymalną prędkością nie większą niż 4 sekundy.	H	L
Charakterystyka czasowa	Generowanie raportów powinno odbywać się ze średnią prędkością nie większą niż 1 godzina.	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	Ma zachęcać, nie zniechęcać. Środowisko ma być przyjazne i czytelne.	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.	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 przysyłać 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 przysyłać wyniki ankiet do systemu raportowania.	H	M
Kompletność kontekstowa	System ma działać dla: 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	System powinien współpracować z systemem BI (raportowanie).	H	H
Łatwość adaptacji	Należy utworzyć raport z łatwości adaptacji. Gdzie znajdują się adaptery.	M	L
Łatwość instalacji	System umożliwiać łatwą aktualizację. Zakładamy, że wersja Moodle'a pozostaje bez zmian.	H	H
Łatwość nauczania się	Interfejs użytkownika (dla ankietowanych) powinien być całkowicie intuicyjny.	H	M
Łatwość nauczania się	Interfejs użytkownika (dla ankietera) może wymagać drobnego szkolenia.	M	L
Łatwość testowania	„Atrapy” (ang. <i>mock</i>) systemów zewnętrznych (m.in. eKonto).	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, 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
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. eKonta, należy poinformować użytkownika o błędzie i uniemożliwić mu dalsze działanie w systemie.	H	L
Odtwarzalność	Odtwarzanie całego systemu w czasie do 3h.	M	L
Odtwarzalność	Kopia zapasowa bazy danych 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
Współlistnienie	Jedynym kryterium działania systemu ankiet (bez raportów) jest działająca przeglądarka.	H	L
Współlistnienie	System generowania raportów może być osobną aplikacją, działającą na specjalnych: sprzęcie i oprogramowaniu.	L	L
Współlistnienie	Wszystkie potrzebne dane z zewnętrznych systemów należy przechowywać lokalnie i synchronizować.	M	H
Współlistnienie	Należy kolejkować zadania (e-maile) w systemie, na czas braku możliwości ich wysłania (awaria ePoczty).	H	M
Zużycie zasobów	System nie powinien wykorzystywać więcej serwerów 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 natychmiastowe, 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

Uzupełnić o problemy związane ze spełnieniem wymagań pozafunkcjonalnych - weryfikacja.

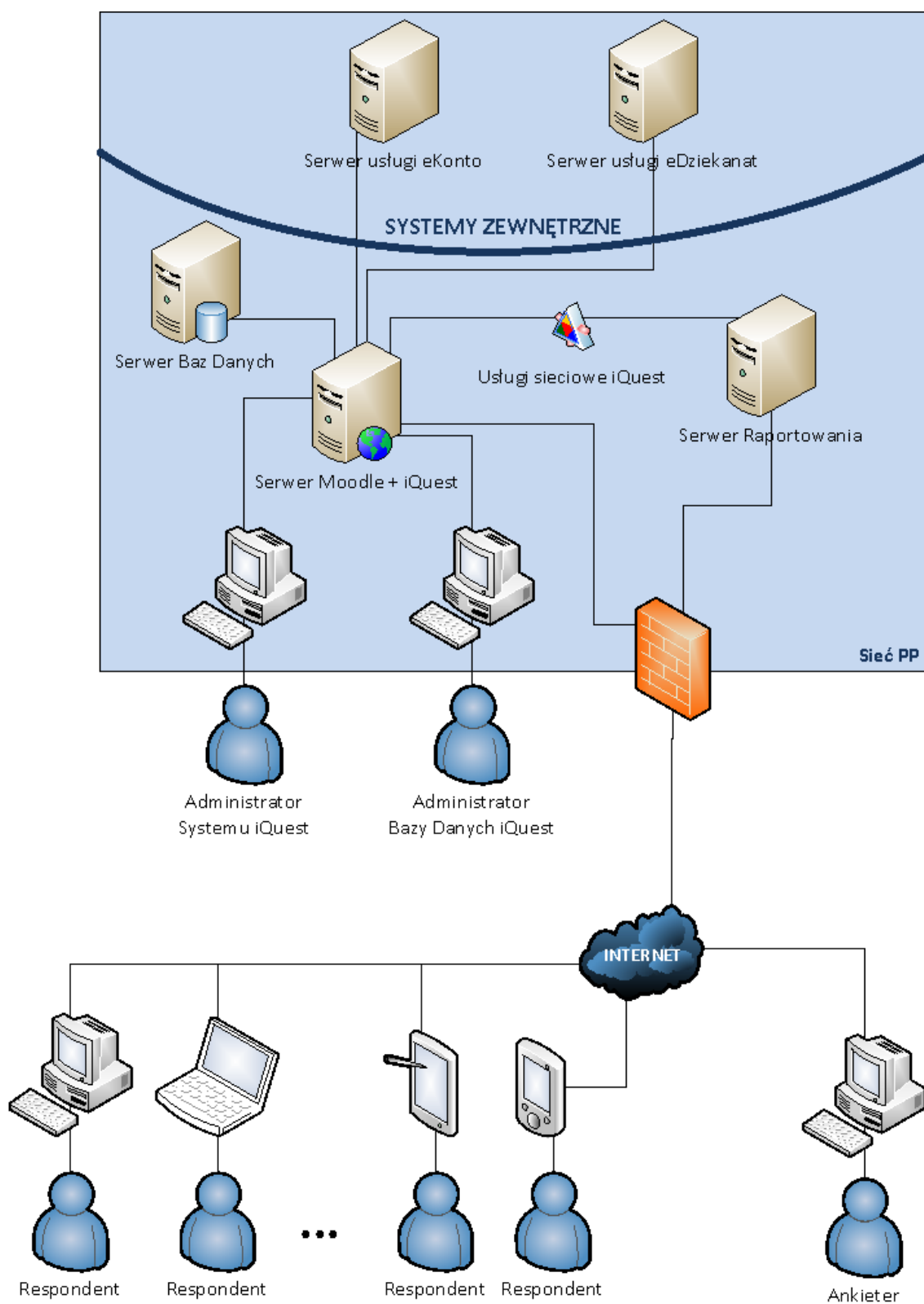
Rozdział 5

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 między członków zespołu programistów, a w przypadku ewentualnej modyfikacji jednej z warstw nie występuje konieczność wprowadzania zmian w reszcie projektu.

5.2 Opis ogólnej architektury – Marketecture

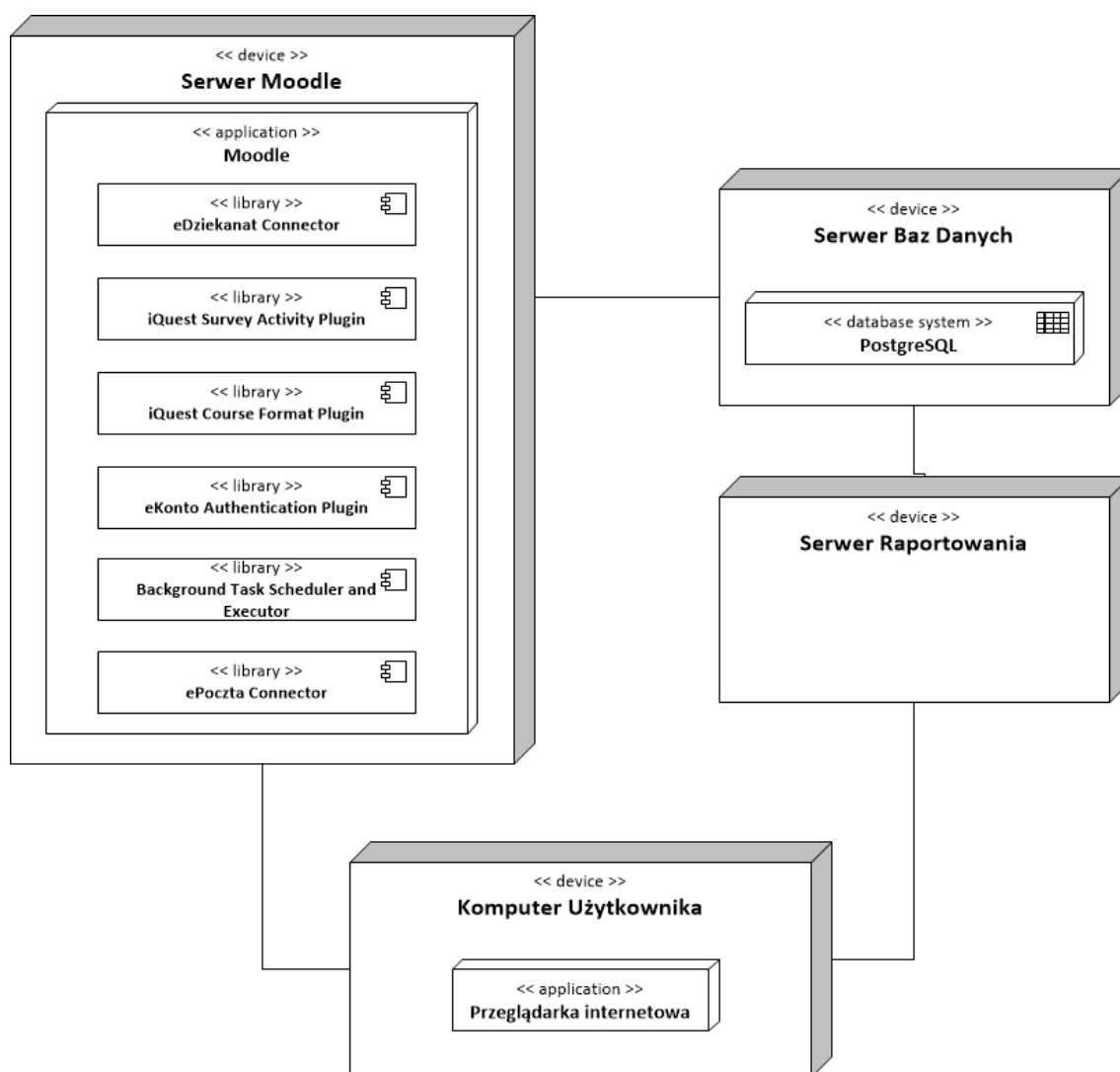


RYSUNEK 5.1: Diagram „Marketektury”

System iQuest to aplikacja internetowa w postaci zbioru rozszerzeń platformy e-learningowej Moodle. Całość (Moodle oraz rozszerzenia) 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 (en. *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.

5.3 Perspektywy architektoniczne

5.3.1 Perspektywa fizyczna

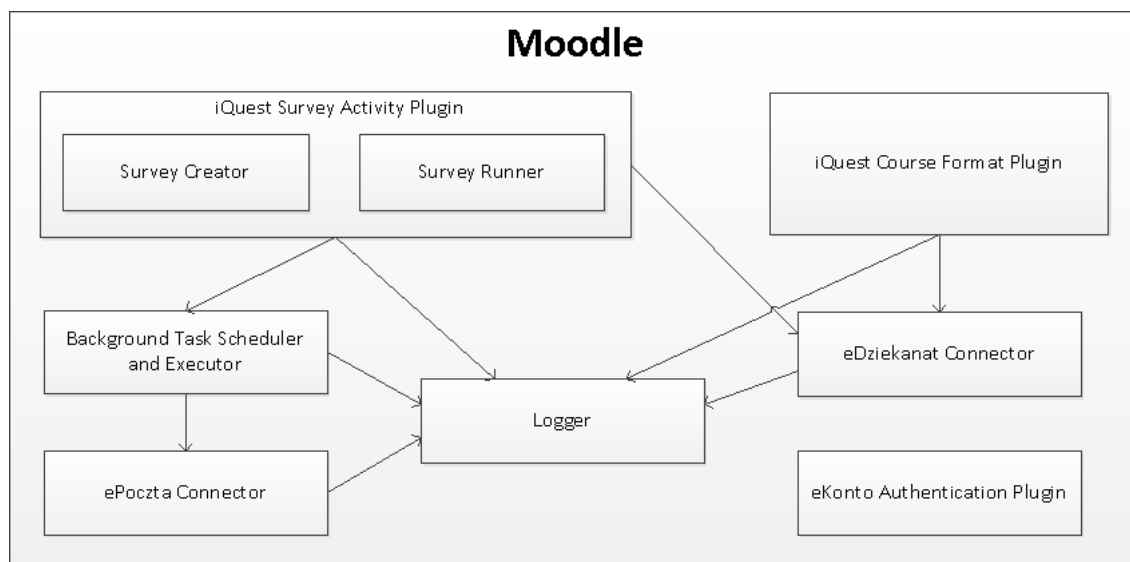


RYSUNEK 5.2: Diagram perspektywy fizycznej

Powyższy schemat prezentuje perspektywę fizyczną projektu. Widać na nim dokładnie opisaną wcześniej budowę systemu iQuest, opartą na rozszerzeniach dla platformy Moodle, w ramach których wykonywana jest cała logika aplikacji. Za jej pośrednictwem dokonuje on połączeń z

serwerem baz danych oraz serwerem raportowania. Użytkownik systemu, z wykorzystaniem przeglądarki internetowej, komunikuje się z platformą, lub systemem raportowania, uzyskując w ten sposób dostęp do warstwy prezentacji.

5.3.2 Perspektywa logiczna



RYSUNEK 5.3: Diagram perspektywy logicznej

Przedstawiony powyżej schemat 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 Pluginów jest udostępnianie możliwości dodania nowych rodzajów Aktywności w ramach platformy Moodle. Pozwala on na dodanie nowego Badania iQuest w ramach kursu iQuest. Komponent ten składa się z dwóch subkomponentów:

- Survey Creator,
- Survey Runner.

Pierwszy z nich odpowiada za definiowanie ankiet, podczas gdy drugi za ich przeprowadzanie.

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 "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.

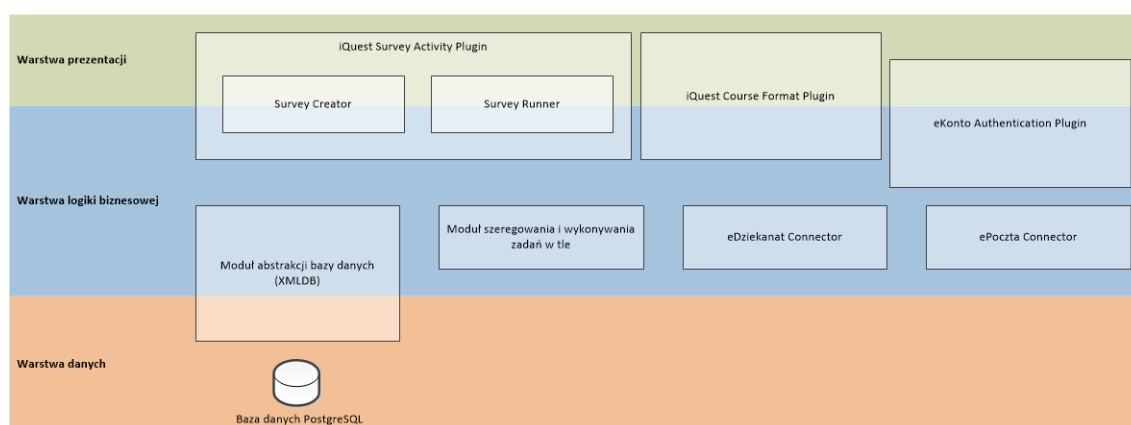
Background Task Scheduler and Executor

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.

5.3.3 Perspektywa implementacyjna



RYSUNEK 5.4: Diagram perspektywy implementacyjnej

Diagram perspektywy implementacyjnej pozwala na analizę przeplatania się elementów systemu iQuest w ramach poszczególnych warstw zastosowanego modelu trójwarstwowego. Funkcje zaprezentowanych na nim modułów zostały wyjaśnione już wcześniej w niniejszym dokumencie.

5.3.4 Perspektywa procesu (równoległości)

Rysunek wraz z opisem.

5.4 Decyzje projektowe

Tutaj będzie o decyzjach projektowych, związkach pomiędzy nimi oraz innych związanych sprawach. Jest tego dość sporo i wymaga specyficznego formatowania.

5.5 Wykorzystane technologie

Opis wykorzystywanych technologii (COTS). Sprawdzić czy ten rozdział to nie dubel!

5.6 Schemat bazy danych

Schemat bazy danych, ze względu na objętość, zamieszczono w dodatku C.

Rozdział 6

Opis implementacji

6.1 Wstęp

Wersja bez redakcji.

6.2 Napotkane problemy i ich rozwiązania

6.3 Użyte technologie

6.3.1 Moodle

Moodle (roz. *Modular Object-Oriented Dynamic Learning Environment*) – stanowi podstawę systemu *iQuest*. Wyboru dokonano ze względu na kilka czynników:

- Propozycję 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.

6.3.2 PHP

PHP – platforma Moodle opiera się właśnie na tym języku programowania. Z tego względu, jest to też technologia zastosowana w większości rozszerzeń utworzonych przez zespół /textiti-Quest, korzystających z interfejsów programowania aplikacji tej platformy. Ponadto PHP jest jednym z najpopularniejszych języków programowania aplikacji internetowych, posiada doskonałą dokumentację oraz jest cały czas rozwijany.

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 ta druga służy do testów, które wchodzą w interakcję z bazą danych.

6.3.4 Selenium

Selenium – szybko rozwijające się narzędzie do testów akceptacyjnych. Był to naturalny wybór zwłaszcza, że zostało ono przybliżone programistom na zajęciach z Inżynierii Oprogramowania w trakcie toku studiów.

6.3.5 PostgreSQL

System zarządzania bazą danych *PostgreSQL* został wybrany ze względu na wymaganie pozafunkcyjne – pracownicy *Działu Rozwoju Oprogramowania Politechniki Poznańskiej* 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.

6.3.7 SVN

Subversion został wybrany jako podstawowy system kontroli wersji ze względu na wymagania pozafunkcyjne. Zespół eksploatacji, który docelowo przejmie zarządzanie artefaktami związanymi z projektem, wykorzystuje właśnie *SVN*.

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. 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 pozafunkcyjne, zdecydowano się skorzystać z mechanizmów raportowania oferowanych przez *JasperReports*.

6.3.10 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, co ogranicza zarówno obciążenie łącza po stronie serwera, jak i po stronie klienta.

6.4 Ogólna struktura projektu

Sekcja druga.

6.5 Interfejs

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*.

6.6 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 uczelnianym *ePocztą*; moduł logowania zdarzeń. W trakcie implementacji zdecydowano 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 dostarczonymi przez platformę *Moodle*, m.in. *Data manipulation API* (zarządzanie danymi), *Access API* (zarządzanie dostępem, rolami, prawami).

6.6.1 Raporty

Raporty wykonaliśmy korzystając z platformy *JasperReports*, wykorzystując następujące produkty firmy *Jaspersoft*:

- *JasperReports Server* (wersja 5.0) – serwer usług raportowania, na którym przechowywane są przygotowane przez nas artefakty, w celu umożliwienia generacji raportu osobom dysponującym odpowiednimi uprawnieniami. Korzystaliśmy z funkcjonalności: definiowania źródła danych, raportu, ładowania plików z projektem raportu, zasobami oraz z generacji raportu.
- *Jaspersoft Studio* (wersja 1.3.2) – bazowane na Eclipse narzędzie do projektowania raportów. Posłużyło nam do przygotowania projektów raportów w formacie *JRXML*. Tak przygotowany raport zawiera dokładną informację o pozycji tworzących go elementów z dokładnością do piksela.

Kody źródłowe pakietu *JasperReports* są pisane w języku Java. Źródłem danych dla raportu, jest przygotowana przez nas implementacja interfejsu *ReportDataSourceService* z *API JasperServer*. Nasze źródło danych łączy się z udostępnianymi przez wskazaną instancję systemu *iQuest* usługami zdalnymi, z których otrzymuje informacje o przeprowadzanych badaniach poprzez protokół SOAP. Struktura raportu zależy od typu pytania (otwarte/zamknięte). W przypadku pytań otwartych prezentujemy listę odpowiedzi. Dla pytań zamkniętych na podstawie pobranych danych

generujemy statystyki, które przekazywane są do podraportu w postaci obiektu klasy *JRBeanCollectionDataSource*. W generacji statystyk z danych badań pomogła nam biblioteka *JoSQL*. Definicja projektu raportu składa się z czterech plików, odpowiadających trzem kolejnym poziomom:

1. *researches.jrxml* – raport główny dla badań,
2. *questions.jrxml* – podraport pytań,
3. *answers_closed.jrxml* oraz *answers_open.jrxml* – podraporty odpowiedzi. zamkniętych i otwartych.

Do generacji namiastek obiektów zdalnych (ang. *stub*) wykorzystaliśmy Apache Axis. Wygenerowane klasy przerobiliśmy, by akceptowały obiekty z zadanej instancji *iQuest* – zmienna przestrzeń nazw (ang. *namespace*).

W trakcie generacji *stub*’ów odkryliśmy, iż definicja usług dla protokołu *SOAP* w języku *WSDL* (*Web Service Description Language*) generowana przez *Moodle* jest niepoprawna. Skoryzaliśmy zatem z poprawionej wersji z zewnętrznego źródła (https://github.com/ghigio/moodle-webservice_soapfda).

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

6.6.2 Moduły uwierzytelniania

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

- *eKontoAuthenticationPlugin* – integruje logowanie przez *eKonto* z naszym systemem,
- *emailgraduate* – pozwala absolwentom uczelni na rejestrację z użyciem adresu e-mail.

W celu spełnienia wymagań Działu Rozwoju Oprogramowania Politechniki Poznańskiej odnośnie wygaszania sesji użytkownika *eKonto* po zadanim czasie (e.g. 15 min.) musieliśmy zmodyfikować pliki źródłowe *Moodle*, gdyż dla kodu odnoszącego się do sesji użytkownika *Moodle* nie została przewidziana możliwość rejestracji rozszerzeń. Relacja *user* została przez nas rozszerzona o opcjonalne pola związane z *eKonto*, jako że kod odpowiedzialny za manipulację schematem bazy danych nie jest wykonywany podczas instalacji modułu uwierzytelniania, umieściliśmy go w osobnym module (*ekontodb*). *eKontoAuthenticationPlugin* może być instalowany bez konieczności instalacji *iSurvey*. Podczas jego implementacji korzystaliśmy z dokumentu *Centralne uwierzytelnianie i wymiana danych. Wersja 1.2 (2010.07.06)*.

Podczas rejestracji z użyciem naszych modułów użytkownik jest przydzielany do grupy docelowej „Absolwenci”, nadawana jest mu też rola respondenta w kontekście *kursu iQuest*. Utworzenie modułu wiąże 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 studentach.

Dział Rozwoju Oprogramowania udostępnia klienty *eUsług* dla różnych języków, w tym dla *PHP*. Komunikacja z usługami zdalnymi uczelni odbywa się poprzez protokół SOAP. Wyżej wymienione moduły zaimplementowaliśmy 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 back-endu z interfejsem

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

Wersja bez redakcji; zostanie zmieniony sposób podziału na chronologiczny; zostaną rozdzielone problemy i ich rozwiązania od doświadczeń i wniosków, oraz przeniesione do działu implementacji/architektury.

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 powinien 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*. Aktywności 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 dotyczy 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, nie chciano programować czegoś od początku samemu, skoro wcześniej zostało to już przez kogoś napisane. Nawet kosztem tego, że interfejs nie wyglądał dokładnie tak, jak go zaprojektowano. Na pierwszym miejscu stawiano jego kompletność. 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. W pierwszej kolejności 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 wspaniałe wyposażenie, 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.

Po *Output API* wybrano *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*. Mimo wielu jego zalet, 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 ankierów 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 ankierom tylko określonych modułów. Ankierzy 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 ankierowi 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 wtyczki 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 ankier wyrażał chęć utworzenia nowego badania, klikając odpowiedni przycisk. Następnie mógł dodać do badania ankietę z katalogu, ewentualnie utworzyć nową. Na kolejnej stronie ankier

definiował szczegóły badania, takie jak: nazwa, grupa docelowa, czas rozpoczęcia i zakończenia etc. Niestety, 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 informację, że po utworzeniu badania ma dodać się do niego ankieta. (Na przykład można w tym celu użyć dodatkowy parametr w adresie *URL*, ale stwarza to potencjalny problem ile razy trzeba go będzie przesłać.) 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 modulem, 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 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, 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 należało rozwiązać. Kilka razy wykazano również inicjatywę i zaproponowano rozwiązania, które stały się ostatecznie częścią projektu.

Pierwszą ideą było zagospodarowanie przestrzeni w widoku badania. Po utworzeniu badania i dodania do niego ankiety, ankietarowi ukazuje się widok badania. Architekt nie zaproponował jak ma on wyglądać. Dał tylko pewne wskazówki. Zaznaczył, że z tego widoku, ankietar 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”. Prak-

tycznie cała strona 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 odpowiadano. 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 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. Dzięki temu, że 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 długiego przewijania ekranu.

W założeniach projektu ustalono, że odpowiedź jest nieedytowalna. Dodano udoskonalenie, które polegało na tym, że respondent nie musi od razu wypełnić całej ankiety. 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.

Hierarchia CSS

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

Najważniejsze są pliki umieszczone w katalogu *theme* odnoszące się do całej platformy *Moodle*.

Następnie uwzględniane są reguły z pliku *styles.css* umieszczonego w katalogu konkretnej wtyczki.

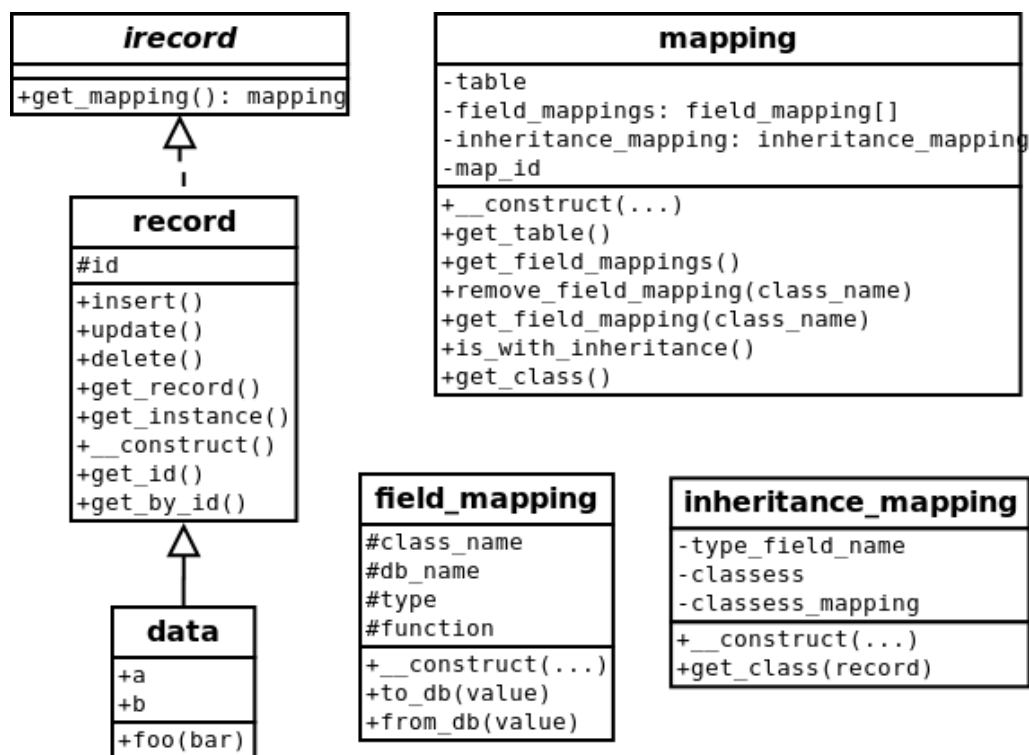
Główną wadą tego podejścia jest to, że nie można we wtyczce nadpisać właściwości, która została zdefiniowana w katalogu *theme*. Aby zmienić choćby jedną właściwość z tego katalogu należy utworzyć nowy wygląd, kopiując stary i zmieniając tę jedyną właściwość. Następnie administrator musi ustawić ten wygląd w swoim systemie (co wiąże się z dodatkową operacją jeśli poprzedni wygląd był wyglądem domyślnym).

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 korzystającemu z tego rozwiązania na proste pobieranie i manipulację obiektami przechowywanymi w bazie danych. Diagram UML przedstawia się następująco:



RYСУNEK 8.1: iQuest ORM

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 w metodzie *get_instance*, która pobiera konstruktor danej klasy, poprzez refleksję tworzy obiekt z

wyłuskany 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

Do uzupełnienia pod koniec prac.

9.2 Propozycja dalszych prac

Do uzupełnienia pod koniec 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
Analitik/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



© 2013 Krzysztof Marian Borowiak, Maciej Trojan, Krzysztof Urbaniak, Łukasz Wieczorek

Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu \LaTeX .

Bib \TeX :

```
@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",  
}
```