

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
1.2.1	Ograniczenia	2
1.2.2	Zagrożenia	3
1.2.3	Wpływ ograniczeń i zagrożeń na projekt	3
1.3	Cele projektu	3
1.4	Osoby realizujące projekt	4
1.5	Struktura pracy	4
2	Opis procesów biznesowych	5
2.1	Aktorzy	5
2.2	Obiekty biznesowe	5
2.2.1	Ankieta	5
2.2.2	Badanie	6
2.2.3	Grupa Docelowa	6
2.2.4	Katalog Ankiet	6
2.2.5	Pytanie	6
2.2.6	Raport	6
2.3	Biznesowe przypadki użycia	7
2.3.1	BC01: Zbieranie informacji o Absolwentach	7
2.3.2	BC02: Zbieranie informacji o Studentach	7
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	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	Opis wymagań	15
5	Architektura systemu	20
5.1	Wstęp	20
5.2	Opis ogólny architektury – Marketecture	21
5.3	Analiza podejść/Analiza SWOT	22
5.4	Perspektywy architektoniczne	23
5.4.1	Perspektywa fizyczna	23
5.4.2	Perspektywa logiczna	24
	iQuest Survey Activity Plugin	24
	iQuest Course Format Plugin	24
	eDziekanat Connector oraz ePoczta Connector	24
	Background Task Scheduler and Executor	25
	eKonto Authentication Plugin	25
5.4.3	Perspektywa implemetancyjna	25
5.4.4	Perspektywa procesu (równoległości)	26
5.5	Decyzje projektowe	26
5.5.1	Wstęp	26
5.5.2	Podjęte decyzje	26
5.5.3	Zależności między decyzjami	35
5.5.4	Alternatywne decyzje	36
5.6	Schemat bazy danych	38
6	Opis implementacji	39
6.1	Wstęp	39
6.2	Napotkane problemy i ich rozwiązania	39
6.2.1	Wstęp	39
6.2.2	Platforma Moodle	39
6.2.3	Inicjalizacja bazy danych	40
6.2.4	Instalacja modułu	40
6.2.5	Formularze	40
6.2.6	Role	41
6.2.7	Formater kursu	41
6.2.8	Tworzenie badania	42
6.2.9	Tworzenie ankiety	43
6.2.10	Hierarchia CSS	43
6.2.11	Testy jednostkowe i akceptacyjne	43
6.2.12	Mapowanie obiektowo-relacyjne	44
6.2.13	Inwencja programistów	46
6.3	Użyte technologie	47

6.3.1	Moodle	47
6.3.2	PHP	47
6.3.3	PHPUnit	47
6.3.4	Selenium	47
6.3.5	PostgreSQL	48
6.3.6	Eclipse IDE	48
6.3.7	SVN	48
6.3.8	Redmine	48
6.3.9	JasperReports	48
6.3.10	JetBrains PhpStorm	49
6.3.11	JavaScript	49
6.4	Ogólna struktura projektu	49
6.5	Interfejs	49
6.5.1	Bezpieczeństwo	49
6.6	Logika (back-end)	49
6.6.1	Raporty	52
6.6.2	Moduły uwierzytelniania	52
6.6.3	Moduły dla serwisów zewnętrznych	53
6.7	Powiązanie logiki z interfejsem	53
7	Zapewnianie jakości i konserwacja systemu	54
7.1	Testy i weryfikacja jakości oprogramowania	54
7.1.1	Wstęp	54
7.1.2	Testy jednostkowe	54
7.1.3	Testy akceptacyjne	57
	MAT	57
	AAT	63
7.1.4	Inne metody zapewniania jakości	63
8	Zebrane doświadczenia	65
9	Zakończenie	68
9.1	Podsumowanie	68
9.2	Propozycja dalszych prac	69
A	Informacje uzupełniające	70
A.1	Wkład poszczególnych osób do przedsięwzięcia	70
A.2	Wykaz użytych narzędzi i technologii	72
A.3	Zawartość płyty CD	73
B	Wygląd aplikacji	74
C	Schemat bazy danych	75

Rozdział 1

Wprowadzenie

1.1 Opis problemu i koncepcja jego rozwiązania

1.1.1 Problem

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 każdej jednostki, zrzeszającej większą grupę osób, jak zakład pracy, czy jednostka szkolnictwa wyższego, jaką jest Politechnika Poznańska, ankietowanie jest niezwykle potrzebne, celem zapewnienia prawidłowej pracy. Bez informacji zwrotnej (ang. *feedback*) praktycznie nie ma możliwości precyzyjnej analizy poprawności podejmowanych działań, co w konkretnych sytuacjach może sprzyjać pojawieniu się niespodziewanej fali problemów. Za przykład może posłużyć sytuacja, gdy nie posiadając wiedzy o nieodpowiednim (zdaniem studentów i prowadzących zajęcia) wyposażeniu sali laboratoryjnej, władze uczelni ustalają harmonogram zajęć, zwiększający obciążenie tego pomieszczenia, co byłoby w tej sytuacji niewskazane.

W ostatnich latach¹ Politechnika Poznańska posiadała szerokie spektrum narzędzi do pozyskiwania wiedzy o swoim działaniu i oferowanych usługach, funkcjonujących w zgodzie ze Statutem Politechniki Poznańskiej². Niestety, gros z nich wzajemnie się wykluczał. Przykładowy student na Wydziale Informatyki³ poddawany był ankietom: elektronicznym na poziomie Uczelni oraz Wydziału, „papierowym” na poziomie Uczelni oraz Samorządu Studentów, a ponadto, dodatkowej ankietyzacji w ramach niektórych zajęć dydaktycznych.

Żaden spośród tych „systemów” ankietyzacji nie posiadał odpowiednich zabezpieczeń, wymaganych do zapewnienia wymierności ich wyników (jak np. zagwarantowanie, 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 związana z pytaniem „kto tak naprawdę pozyskuje informacje”, działały tu na niekorzyść ankietujących.

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 brakuje

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

²Zapis dotyczący dokonywania oceny nauczyciela akademickiego z uwzględnieniem oceny studentów[?].

³Do 2010 roku – Wydziale Informatyki i Zarządzania

odpowiednich narzędzi do prowadzenia tego rodzaju analiz, w szczególności, że dotychczasową grupą docelową wszelkich badań i ankiet byli jedynie aktualni studenci. Nawiązując więc do wcześniejszych twierdzeń, w konsekwencji niespełnienia wymagań określonych w Ustawie i braku odpowiednich mechanizmów prowadzenia badań, mogłoby zaistnieć prawdopodobieństwo wystąpienia spadku jakości usług świadczonych przez uczelnię, a co za tym idzie, także jej prestiżu.

1.1.2 Proponowane rozwiązanie

Najłatwiejszym, a zarazem – 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 systemu iQuest rozwiązanie obejmuje więc też element zachęcający do korzystania z systemu, jakim jest udzielanie użytkownikom spoza uczelni dostępu do unikalnych materiałów dydaktyczno-naukowych.

Aby zapewnić, że pozyskane w badaniach dane są prawidłowe, w proponowanym rozwiązaniu znajdą się systemy autoryzacji osób ankietowanych. Jest to jednak jedyny element wymagający sprawdzenia tożsamości respondenta – same wyniki będą dla ankietera całkowicie anonimowe.

1.2 Ograniczenia i zagrożenia dla projektu

W zależności od punktu widzenia, z jakiego projekt iQuest podda się analizie, wskazać można różne zestawy ograniczeń i zagrożeń dla jego realizacji. Dla przykładu, rozpatrując tę kwestię ze strony klienta, ważna może być kwestia zachęty studentów i absolwentów Uczelni do korzystania z systemu. Dla zespołu programistów, którzy są jednocześnie autorami niniejszej pracy dyplomowej, będzie to sprawa poboczna, niezwiązana z realizacją wymagań funkcjonalnych⁴. Zdecydowano więc o analizie jedynie ograniczeń i zagrożeń związanych ze stroną implementacyjną projektu, przy uwzględnieniu następujących definicji:

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

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

1.2.1 Ograniczenia

System iQuest realizowany jest na zlecenie Dziekana Wydziału Informatyki. Z tego względu, musi on podlegać szerokiemu spektrum wymagań ze strony takich organów, jak DRO (*Dział Roz-*

⁴Rozdział 3. - Wymagania funkcjonalne.

woju Oprogramowania). Jednym z nich jest konieczność stosowania rozwiązań „otwartych” (ang. *open*) oraz „wolnych” (ang. *free*). Znacząco zawęża to możliwe do zastosowania technologie i zasoby.

Za poboczne ograniczenie można uznać z góry narzucony harmonogram prac. Zespół programistów realizował tego rodzaju projekt po raz pierwszy, stąd nie było dla niego łatwym zweryfikowanie możliwości jego wykonania.

1.2.2 Zagrożenia

Zagrożeniem dla projektu mógł być brak wcześniejszego doświadczenia zespołu przy realizacji tego rodzaju projektów. W początkowych fazach projektu sądzono więc, że nieznanomość technologii może przyczynić się do powstania ciężkich do przekroczenia barier. Jak się jednak okazało, powyższe zagrożenie okazało się mało istotne, gdyż zespół dynamicznie przystosowywał się do sytuacji, brak doświadczenia nadrabiając umiejętnością szybkiej nauki.

W nawiązaniu do powyższego zagrożenia, na drodze programistów stała jednak jedna poważna przeszkoda – niekompletność dokumentacji dotyczącej niektórych systemów i technologii zastosowanych w projekcie.

Najpoważniejszym problemem okazały się jednak niespodziewane zmiany w specyfikacji projektu oraz w wymaganiach klienta w trakcie tworzenia systemu..

1.2.3 Wpływ ograniczeń i zagrożeń na projekt

Decyzja o zastosowaniu konkretnych typów technologii zdecydowała o bardzo wielu aspektach projektu, poczynając od tego, czy realizowany będzie on od podstaw, czy na podstawie innego systemu. Było to dość istotne i miało swoje odzwierciedlenie na liście zagrożeń – o ile bowiem rozwiązania komercyjne są nierzadko doskonale udokumentowane przez ich twórców, o tyle te darmowe często opisuje tzw. „społeczność”, co wpływa na kompletność i jakość dostępnych treści. Wiążąc to z brakiem wcześniejszego doświadczenia, nawet pomimo umiejętności szybkiego uczenia się nowości, sprawiało to niemałe problemy – trudno nauczyć się czegoś, czego nigdzie nie opisano w sposób umożliwiający łatwą naukę. Często wymuszało to realizację zadań na zasadzie „prób i błędów”, co wpływało na ich czas realizacji.

Wspomniany w poprzedniej sekcji (1.2.2. - Zagrożenia) problem zmieniających się wymagań i specyfikacji sprawił jednak najwięcej problemów. Gdy bowiem rozwiązywane były kłopoty dotyczące nieprzystępności niektórych technologii, nierzadko trzeba było wracać do podstaw, by właśnie utworzoną funkcjonalność zmodyfikować, zapewniając zgodność z nowymi wytycznymi. Było to główną przyczyną powstawania opóźnień w realizacji projektu.

1.3 Cele projektu

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

- zapewnić spełnienie przez Uczelnię zapisów Ustawy „Prawo o Szkolnictwie Wyższym” dotyczących monitorowania rozwoju absolwentów Uczelni[?]
- ujednolicić uczelniany system pozyskiwania informacji

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

1.4 Osoby realizujące projekt

Projekt realizowany jest w ramach SDS (ang. *Software Development Studio*), służącego realizacji projektów informatycznych dla Wydziału Informatyki Politechniki Poznańskiej, gdzie w rolę zespołu zarządzającego wcielają się studenci studiów magisterskich (specjalizacja *Software Engineering*), zaś zespół programistów tworzą studenci ostatniego semestru studiów inżynierskich. Dla tych ostatnich, udział w SDS jest ściśle związany z pracą dyplomową inżynierską.

Szczegóły odnośnie osób realizujących projekt znajdują się w dodatku A. Poniżej przedstawiono wykaz kompetencji poszczególnych zespołów:

Zespół zarządzający – obejmuje dwóch studentów, pełniących role *kierownika projektu* oraz *architekta*. Kompetencje:

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

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

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

1.5 Struktura pracy

Cała praca podzielona została umownie (w trakcie tworzenia) na trzy uzupełniające się części, zawierające po kilka rozdziałów. Część pierwsza, zawierająca rozdziały 1., 8. i 9. oraz sekcję 6.2. rozdziału 6., odnosi się do całości projektu z punktu widzenia osób go realizujących. Opisane są w niej założenia, napotkane problemy i wyciągnięte z pracy wnioski. Część druga, zawierająca rozdziały od 2. do 5., dotyczy charakterystyki projektu oraz jego architektury. Oparto ją na danych pozyskanych od zespołu zarządzającego projektem[?], dostępnych dla zespołu w ramach platformy *Redmine*. Część trzecia, zawierająca pozostałe rozdziały, tj. 6. i 7, przedstawia projekt od strony implementacji. Zawiera szczegółowe informacje odnośnie budowy systemu iQuest.

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

2.2 Obiekty biznesowe

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

2.2.1 Ankieta

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

- Nazwa Ankiety,

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

- Wstęp,
- Podsumowanie,
- Przypisane Pytania.

2.2.2 Badanie

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

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

2.2.3 Grupa Docelowa

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

- Studenci/Absolwenci

2.2.4 Katalog Ankiet

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

2.2.5 Pytanie

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

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

2.2.6 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
Pre: Ankieter chce ankietować Studentów
Post: Ankieta, Raport
Scenariusz Główny
<ol style="list-style-type: none"> 1. Ankieter zgłasza potrzebę stworzenia Grupy Docelowej Administratorowi 2. Administrator podaje nazwę Grupy Docelowej, którą zamierza utworzyć 3. Administrator dodaje/usuwa członków Grupy Docelowej 4. Administrator potwierdza chęć stworzenia Grupy Docelowej 5. System tworzy Grupę Docelową 6. Ankieter może korzystać z Grupy Docelowej

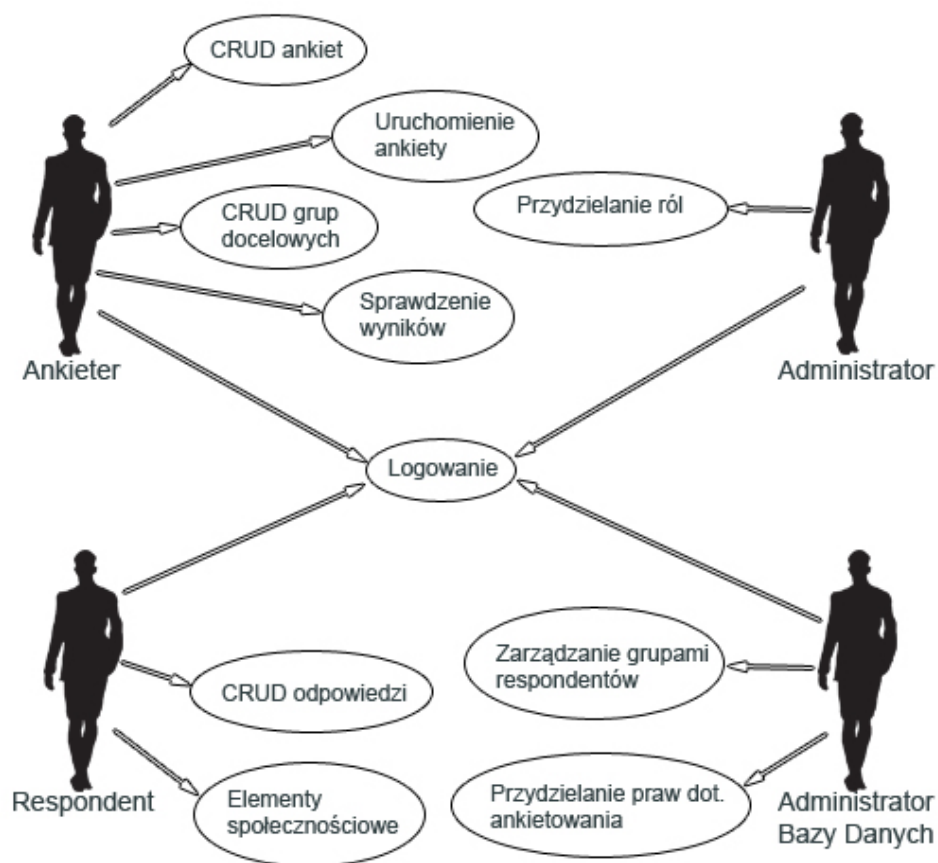
Rozdział 3

Wymagania funkcjonalne

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 przedstawiono diagram przypadków użycia. W ramach systemu udostępniane są różne funkcje, możliwe do wykonania przez różnych aktorów. Dla przykładu, Ankieter może stworzyć 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 Ankiętę
Post:
Scenariusz Główny
<ol style="list-style-type: none"> 1. Ankieter uruchamia interfejs tworzenia Ankiety. Podaje atrybuty Ankiety: nazwę Ankiety, wstęp, podsumowanie 2. System prezentuje stronę umożliwiającą dodawanie pytań 3. Ankieter wybiera typ pytania 4. Ankieter podaje treść pytania 5. Ankieter podaje możliwe odpowiedzi 6. Ankieter akceptuje Ankiętę 7. System prezentuje podsumowanie ankiety i zapisuje ją w Katalogu Ankiet ankietera
Rozszerzenia
<ol style="list-style-type: none"> 4.A Typ pytania: pytanie otwarte 4.A.1 Ankieter pomija krok 5. 6.A Ankieter chce dodać kolejne pytanie 6.A.1 Powrót do kroku 3.

3.2.2 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 Ankiętę
Post:
Scenariusz Główny
<ol style="list-style-type: none"> 1. Ankieter wybiera Ankiętę do modyfikacji 2. System prezentuje wskazaną Ankiętę 3. Ankieter modyfikuje lub usuwa dostępne pytania 4. Ankieter potwierdza chęć zapisu zmienionej Ankiety 5. System zapisuje zmienioną Ankiętę
Rozszerzenia
<ol style="list-style-type: none"> 3.A. Edycja możliwych odpowiedzi do pytań 3.A.1 Ankieter edytuje możliwe odpowiedzi do pytań

3.2.3 UC03: Wybór Grupy Docelowej

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

3.2.4 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 Badania oraz grupę docelową (UC3) 2. Ankieter potwierdza chęć uruchomienia Badania 3. System udostępnia Ankietę Respondentom 4. System powiadamia Respondentów o Ankiecie
Rozszerzenia
1.A. Ankieter chce prowadzić cykliczną ankietyzację 1.A.1 Ankieter ustawia częstotliwość powtarzania Badania

3.2.5 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. Respondent wybiera Ankietę, w której chce wziąć udział 2. System prezentuje wybraną Ankietę Respondentowi 3. Respondent udziela odpowiedzi na pytania 4. Respondent zatwierdza wypełnioną Ankietę 5. System zapisuje odpowiedzi
Rozszerzenia
2.A. Przedawniona Ankieta 2.A.1 System informuje, że Ankieta już się zakończyła 5.A. Brak odpowiedzi na niektóre pytania 5.A.1 System pozwala powrócić do pytań, na które nie udzielono odpowiedzi 5.A.2 Powrót do kroku 1.

3.4 Administrator

Poniżej przedstawiono przypadki użycia dotyczące Administratora.

3.4.1 UC07: Tworzenie Grupy Docelowej

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

3.4.2 UC08: Edycja Grupy Docelowej

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

3.5 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, 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 wymagania pozafunkcjonalne dotyczące systemu *iQuest*. Dodatkowo, zostały one przydzielone do wybranych przez zespół zarządzający charakterystyk oprogramowania.

4.2 Opis wymagań

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

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

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

Charakterystyka	Wymaganie	W	TI
Analizowalność	Komentarze w kodzie źródłowym powinny być w języku angielskim.	M	L
Analizowalność	Kod źródłowy systemu powinien być utworzony zgodnie ze standardami <i>Moodle</i> .	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 <i>eKonta</i> .	H	M
Bezpieczeństwo (wolność od ryzyka)	Dane (opinie respondentów) przechowywane w systemie muszą być uzyskiwane poprzez wbudowane mechanizmy ankietowania.	H	L
Bezpieczeństwo (wolność od ryzyka)	Projekt systemu należy poddać analizie z wykorzystaniem metody ATAM.	M	M
Charakterystyka czasowa	Wyświetlenie ankiety powinno trwać nie dłużej niż 4 sekundy.	H	L
Charakterystyka czasowa	Generowanie raportów powinno odbywać się ze średnio nie dłużej niż 1 godzinę.	M	M
Charakterystyka czasowa	Należy zdefiniować klasy operacji, w zależności od czasu ich trwania. Klasy: <ul style="list-style-type: none"> • bez komunikatu potwierdzającego wykonanie • z potwierdzeniem wykonania • wykonywane na serwerze w tle 	M	M
Dostępność personalna	Przewidzieć, na poziomie architektury, możliwość rozbudowy np. o interfejs dla osób niedowidzących.	L	M
Dostępność techniczna	System może mieć przerwę serwisową, lecz musi wówczas prezentować specjalny ekran informujący o czasie jej trwania.	H	L
Estetyka Interfejsu Użytkownika	Środowisko ma być przyjazne i czytelne dla użytkownika końcowego.	H	M
Funkcjonalna poprawność	Wszystkie wartości mają być prezentowane z dokładnością do 2 miejsc po przecinku.	H	L
Identyfikowalność	System ma umożliwiać identyfikowanie podmiotów (osobno: administratorów, ankietatorów, respondentów), podejmujących konkretne działania: tworzenie ankiet, odpowiadanie w ankietach, itp.	H	M
Integralność	Baza danych powinna być chroniona przed nieuprawnionym dostępem [modyfikacją/usunięciem] w następujący sposób: logowanie za pomocą loginu i hasła.	H	L

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

¹ang. *Business Intelligence*

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

Tablica 4.1: Wymagania pozafunkcyjne

Zważywszy, że projekt ten jest istotny dla Uczelni, dołożono wszelkich starań, aby spełnić wszystkie wymagania pozafunkcyjne. Zadanie to zostało w sporej części wykonane dzięki zastosowaniu sprawdzonego i znanego narzędzia w roli podstawy dla całego systemu. Mowa o platformie *Moodle*². Użycie jej pozwoliło m.in. na zapewnienie pełnej modułowości systemowi, dzięki czemu wprowadzanie jakichkolwiek zmian w dowolnej jego części jest uproszczone i nie wpływa na resztę kodu. Spełniona została także adaptatywność i przenośność systemu, oparta na uniwersalności

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

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 systemu *iQuest*, nakład pracy wymagany do wdrożenia się, celem rozwoju projektu, został zminimalizowany.

Warto zaznaczyć, że w trakcie prac nad projektem, bardzo silny nacisk stawiano na prawidłowe zarządzanie uprawnieniami użytkowników. Są one weryfikowane przy każdym działaniu podejmowanym przez użytkowników. Zależą one od przydzielonej im w systemie roli. 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 bezpośrednio, 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.

W trakcie testów na urządzeniu przenośnym typu netbook³ pełniącym rolę serwera HTTP, serwera bazy danych oraz klienta systemu jednocześnie, cały proces instalacji nie przekraczał 40 minut. Zaobserwowanym głównym czynnikiem ograniczającym wydajność procesu była szybkość środowiska wirtualnego oraz zastosowanego dysku twardego.

³ *Asus Automobili Lamborghini VX6*, CPU: Intel Atom D525 1,7GHz, RAM: 4GB, HDD: 320GB 5400 RPM, Środowisko: korzystający z *WUBI* system *Ubuntu 12.04 LTS*.

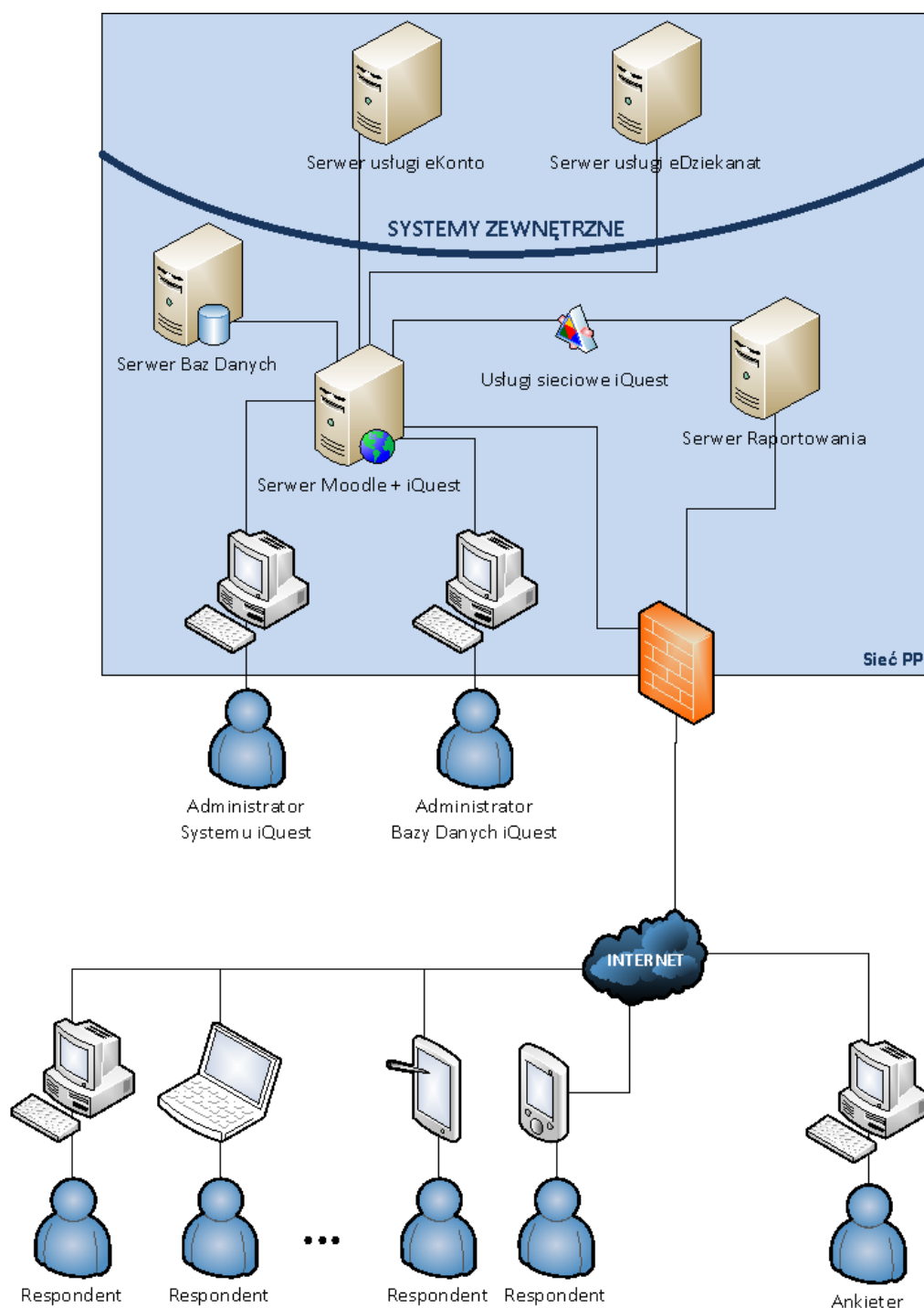
Rozdział 5

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 „Marketecture”

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 ra-

mach 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 Analiza podejść/Analiza SWOT

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

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

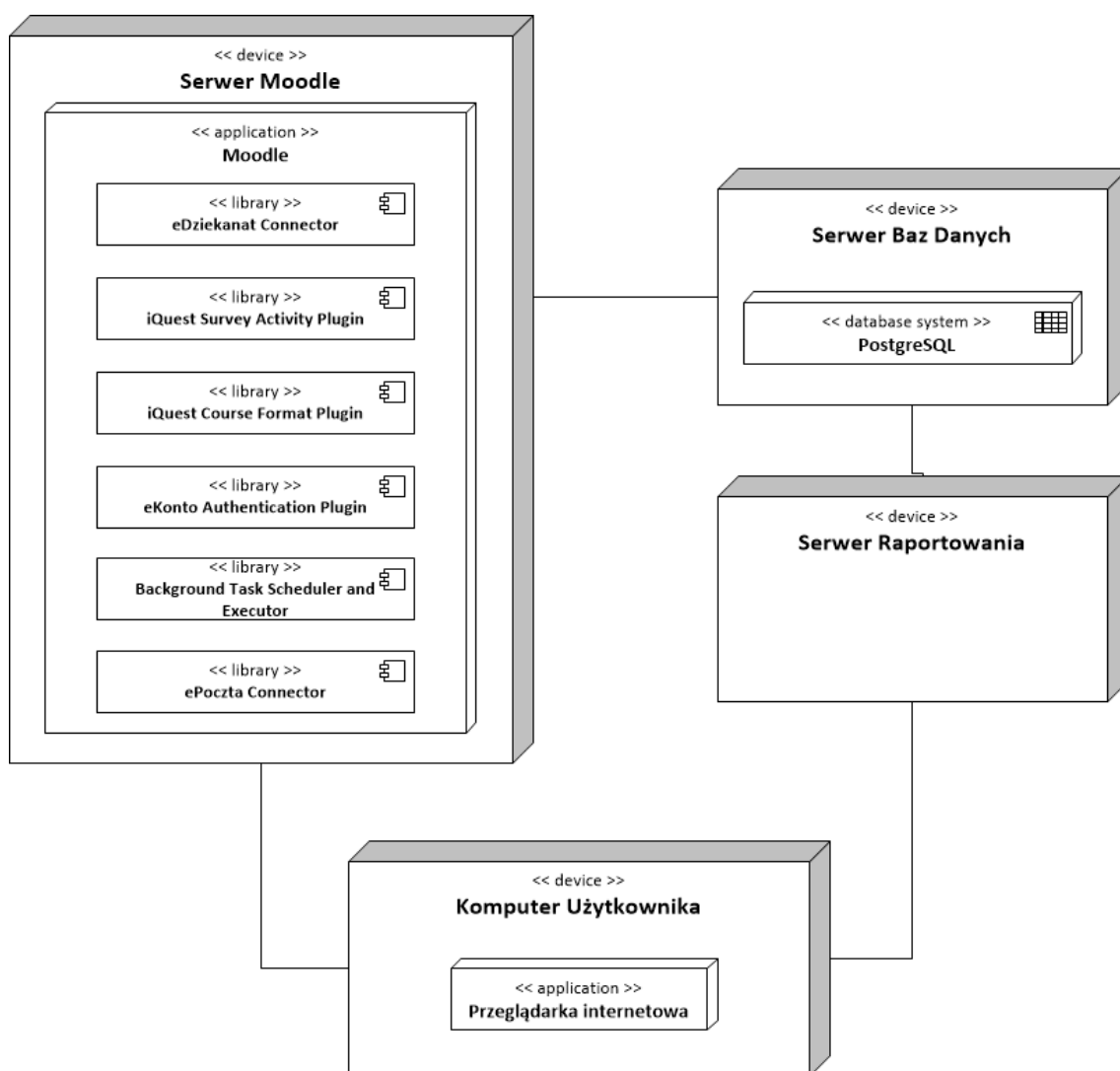
—	Pozytywne	Negatywne
Wewnętrzne	Mocne strony (S) Dostępność platformy na wszystkich popularnych systemach operacyjnych.	Słabe strony (W) Konieczność implementacji dodatkowych funkcjonalności związanych z zachęcaniem do korzystania z systemu.
Zewnętrzne	Szanse (O) Dobra znajomość kodu przez programistów.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

TABLICA 5.2: Podejście drugie – napisanie aplikacji od podstaw przy użyciu *JAVA EE*

W oparciu o analizy SWOT przedstawione w tabelach 5.1 i 5.2, zespół zarządzający zarządził o realizacji projektu z użyciem platformy *Moodle*.

5.4 Perspektywy architektoniczne

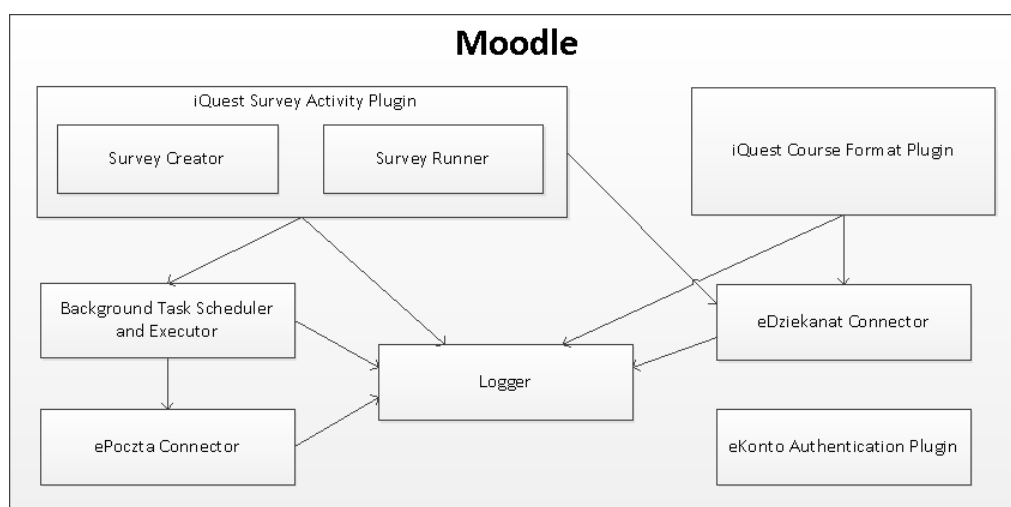
5.4.1 Perspektywa fizyczna



RYSUNEK 5.2: Diagram perspektywy fizycznej

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

5.4.2 Perspektywa logiczna



RYSUNEK 5.3: Diagram perspektywy logicznej

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

iQuest Survey Activity Plugin

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

- Survey Creator
- Survey Runner

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

iQuest Course Format Plugin

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

eDziekanat Connector oraz ePoczta Connector

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

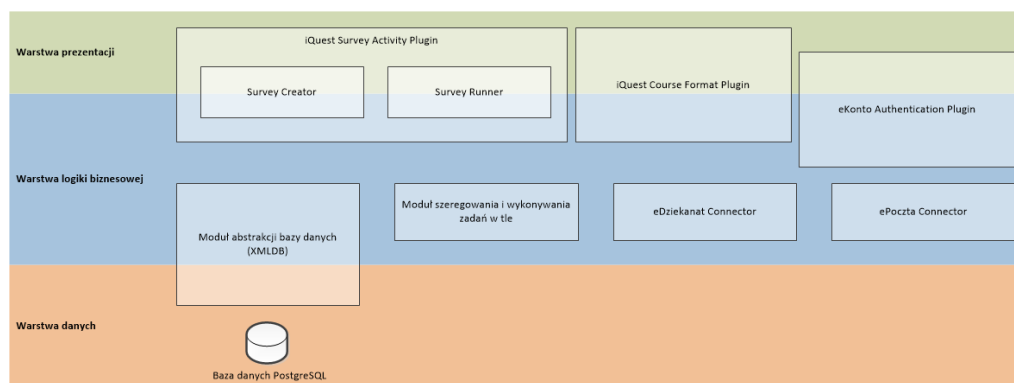
Background Task Scheduler and Executor

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

eKonto Authentication Plugin

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

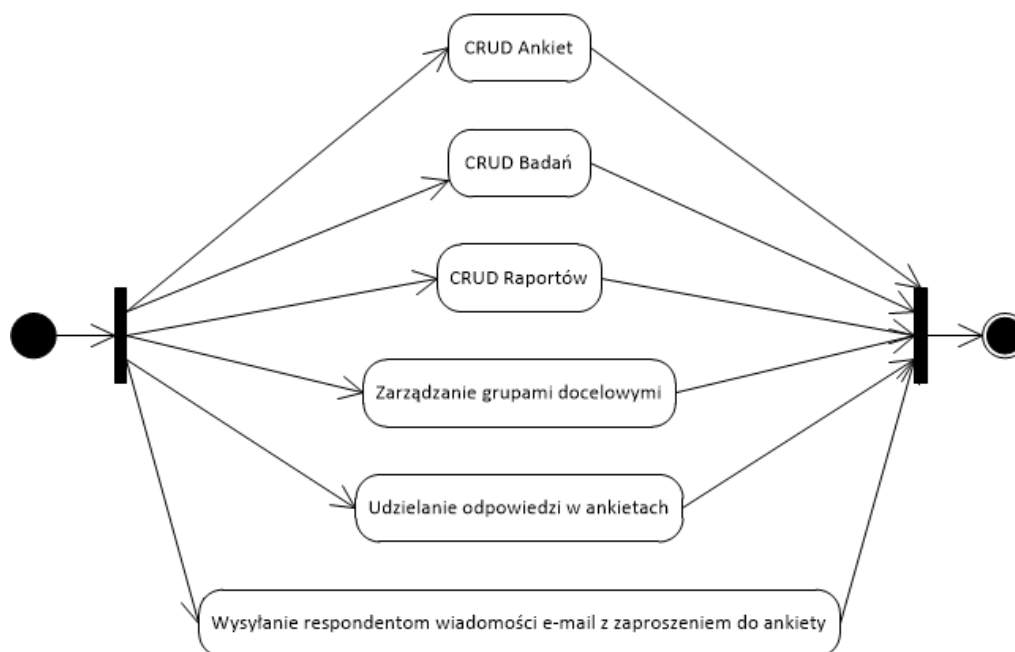
5.4.3 Perspektywa implemetacyjna



RYSUNEK 5.4: Diagram perspektywy implementacyjnej

Diagram perspektywy implementacyjnej (rys. 5.4) 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.

5.4.4 Perspektywa procesu (równoległości)



RYSUNEK 5.5: Diagram perspektywy procesu

Jak widać na diagramie 5.5, istnieje możliwość jednoczesnego wysyłania e-maili i wykonywania w systemie operacji na ankietach, badaniach, raportach oraz grupach docelowych przez różnych użytkowników w tym samym czasie.

5.5 Decyzje projektowe

5.5.1 Wstęp

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

5.5.2 Podjęte decyzje

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.5.3 Zależności między decyzjami

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

Ponadto:

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

5.5.4 Alternatywne decyzje

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

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

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

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

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

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

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

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

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

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

5.6 Schemat bazy danych

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

Rozdział 6

Opis implementacji

6.1 Wstęp

Realizacja projektu iQuest rozciągała się na okres trwający 5 miesięcy. W tym czasie zrealizowano dwa kolejne wydania.

W pierwszym miesiącu – w trakcie praktyk studenckich odbywanych przez 3 spośród 4 członków zespołu programistów – utworzono pierwszą wtyczkę do platformy Moodle. Był to moduł logowania przez eKonto. Następne 3 miesiące trwało utworzenie pierwszego wydania, obejmującego podstawowe mechanizmy tworzenia badań i ankiet, oraz ich przeprowadzania. Ostatni miesiąc poświęcony został na dopracowanie wcześniej wspomnianych modułów o dodatkowe elementy, bezpośrednio związane z wymaganiami funkcjonalnymi i pozafunkcjonalnymi wyznaczonymi dla projektu.

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

W dalszej części niniejszego rozdziału znajduje się analiza systemu iQuest w kontekście implementacji. Rozpoczęty opisem napotkanych problemów i zastosowanych rozwiązań wywód, kontynuowany będzie opisem zastosowanych technologii. Wówczas nastąpi przejście do wyjaśnienia kwestii związanych z implementacją interfejsu i logiki, a także mechanizmów wiążących je ze sobą.

6.2 Napotkane problemy i ich rozwiązania

6.2.1 Wstęp

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

6.2.2 Platforma Moodle

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

Odpowiadają one najczęściej konkretnym zajęciom, wydarzeniom lub np. tygodniom. Najmniejszą jednostką w Moodle jest *aktywność*, będąca podstawowym typem modułów rozszerzających funkcjonalność platformy. Aktywnościami są np. *Fora*, *Głosowania*, czy *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 i związane z nimi ankiety. Moduły grupowane są w sekcje. Różnica pomiędzy tym, czy użytkownik znajduje się lub nie, w konkretnym elemencie platformy, jak moduł, czy kurs, nazywana jest *kontekstem*. O ułożeniu i wyświetlaniu elementów na stronie decyduje *formater*, definiując w ten sposób interfejs użytkownika.

6.2.3 Inicjalizacja bazy danych

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

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

6.2.4 Instalacja modułu

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

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

6.2.5 Formularze

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

Zastosowanie *HTML* oraz zewnętrznych API zostało odrzucone. Decyzję tę podjęto ze względu na obawę, że pisanie interfejsu w całości od nowa okaże się zbyt pracochłonne. Z uwagi na dość mocno ograniczony czas, nie chciano wyważać otwartych już drzwi, tworząc coś, co już wcześniej zostało przez kogoś zrealizowane, nawet kosztem tego, że interfejs nie wyglądał dokładnie tak, jak

go zaprojektowano – na pierwszym miejscu stawiano jego kompletność. Zastosowanie zewnętrznych API było natomiast niezgodne z założeniem, mówiącym o korzystaniu z interfejsów Moodle wszędzie tam, gdzie to możliwe. Niepożądanym było, aby użytkownik poczuł, że programowana wtyczka nie jest integralną częścią platformy. Co więcej, różnice w stosunku do oryginalnego wyglądu platformy mogłyby sprawić, że interfejs oceniony zostałby jako nieintuicyjny. To zaważyło na decyzji odnośnie zastosowania *Output API* wbudowanego w Moodle.

Wyżej wspomniane API jest zestawem funkcji, wprowadzonych wraz wersją 2.0 Moodle. Umożliwiają one wstawianie na stronę standardowych elementów formularza, takich jak: etykiety, przyciski, linki, tabele, itp. Niestety, z nieznanym dla osób tworzącym ten dokument przyczyn, to doskonale wyposażone i w pełni udokumentowane API zostało usunięte z Moodle wraz z aktualizacją do wersji 2.2. Co bardziej niezrozumiałe, niektóre elementy można nadal stosować, lecz niemożliwym okazało się odnalezienie dotyczącej tego dokumentacji.

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

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

6.2.6 Role

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

6.2.7 Formater kursu

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

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

Pierwotnie zakładano wyświetlanie użytkownikowi dwóch sekcji. Jednej z odpowiednimi badaniami, drugiej z materiałami. Należało także ograniczyć ankieterowi możliwość dodawania w

pierwszej sekcji modułów innych niż iQuest, oraz dokładnie odwrotnego działania w drugiej z nich. Okazało się to nieosiągalne bez ingerowania w wewnętrzny kod platformy.

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

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

6.2.8 Tworzenie badania

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

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

Problem stwarzało dodawanie ankiety w trakcie procesu tworzenia badania, jeszcze przed jego zakończeniem. Zaczynając generowanie badania od zdefiniowania ankiety, nie można było jej od razu do niego dodać – badanie to bowiem jeszcze nie istniało. W takim wypadku należałoby przechowywać informację, że po utworzeniu badania ma dodać się do niego ankietę¹. Dodatkowo, w Moodle, przy kreowaniu nowego modułu, użytkownikowi wyświetlany jest domyślny formularz, w którym podaje się parametry potrzebne do zbudowania instancji tego modułu. Przyjmując, że badanie jest kojarzone z modułem, nie ma możliwości, aby przed zakończeniem tworzenia badania wstawić wewnątrz dodatkowy formularz.

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

¹Przykładowo można w tym celu wykorzystać dodatkowy parametr w adresie *URL*, choć stwarzałoby to potencjalny problem dotyczący kwestii liczby przekierowań, przez które musiałby on być przekazywany.

6.2.9 Tworzenie ankiety

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

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

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

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

6.2.10 Hierarchia CSS

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

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

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

Główną wadą tego podejścia jest fakt, że nie można we wtyczce nadpisać właściwości, która została zdefiniowana w katalogu *theme*. Aby zmienić choćby jedną właściwość z tego katalogu należy utworzyć nowy *wygląd*, kopiując oryginalny i zmieniając tę jedną właściwość. Następnie administrator platformy musi ustawić ten wygląd w swoim systemie (co wiąże się z dodatkową operacją, jeśli poprzedni wygląd był wyglądem domyślnym). Problem ten nie mógł zostać rozwiązany i przewijał się przez cały czas implementacji systemu iQuest.

Fragment niezredagowany

6.2.11 Testy jednostkowe i akceptacyjne

Realizacja wszystkich testów została pierwotnie powierzona jednemu z członków zespołu programistów. Zadanie to okazało się posiadać bardzo wysoki stopień złożoności, sprawiając, że napisanie pojedynczej linii kodu zajmowało średnio bardzo dużo czasu.

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

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

Większy problem dotyczył się testów akceptacyjnych. Już na początku realizacji projektu, wyszło na jaw, że eksport z Selenium IDE (standard HTML) do Eclipse IDE (standard Java) nie jest zadaniem prostym. Dla przykładu, część metod odwołania do poszczególnych obszarów na stronie nie może być przenoszony do Javy, ze względu na znane problemy.

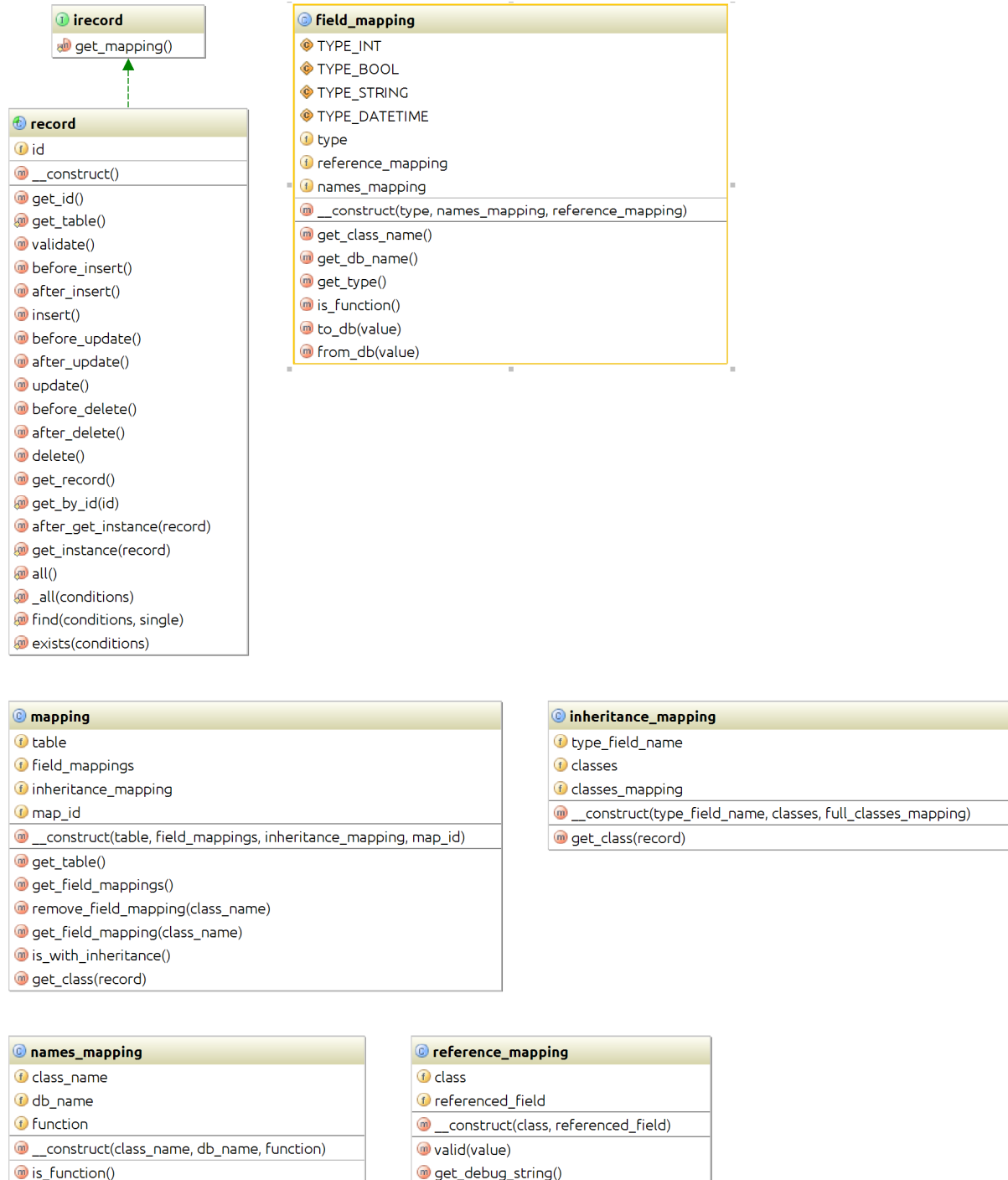
Na podstawie powyższej analizy stwierdzono, że testy akceptacyjne mogą być wykonywane za pomocą rozszerzenia Selenium IDE for Firefox, szczególnie, że narzędzia w nim dostępne umożliwiają spory stopień automatyzacji.

6.2.12 Mapowanie obiektowo-relacyjne

Mapowanie obiektowo-relacyjne pozwala uprościć operacje na danych przechowywanych w bazie danych poprzez udostępnienie ich programiście w postaci obiektowej.

System *iQuest* operuje na klasach takich jak: ankieta, badanie, grupa docelowa, członek grupy docelowej, uprawnienie dostępu, pytanie (i potomne), odpowiedź, zadanie, praca w tle, etc. Początkowo architekt stworzył diagram klas na którym każda klasa miała wyróżnione publiczne metody *insert*, *update*, *delete*. Niestety takie rozwiązanie spowodowało powielenie dużej ilości kodu związanego z interakcją z bazą danych. W ramach refaktoryzacji podjęto 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:

RYSUNEK 6.1: iQuest ORM



Klasa danych dziedziczy po klasie *record* oraz implementuje statyczną metodę *get_mapping* interfejsu *irecord*, by uzyskać dostęp do metod komunikacji z bazą danych. Metoda *get_mapping* pozwala zdefiniować mapowanie danej klasy na odpowiednią relację w bazie danych. Należy przy tym podać nazwę tabeli, mapowanie dla pól klasy składające się z mapowania nazw (klasa *names_mapping*), tj. nazwy w klasie i bazie danych oraz typu, który zadecyduje o metodzie pobrania/zapisania danej (typem może być także klasa potomna klasy *record*). Dodatkowo można uwzględnić istnienie kluczy obcych, których poprawność będzie sprawdzana, jeżeli utworzymy obiekt klasy *reference_mapping*. W przypadku dziedziczenia wystarczy zdefiniować przy mapowaniu sposób jego obsługi (m.in. jakie pole określa typ klasy). Najważniejszy kod znajduje się w metodzie *get_instance*, która pobiera konstruktor danej klasy, poprzez refleksję tworzy obiekt z wyłuskanyymi 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*). Projektowano je, mając na uwadze rozwiązanie, z którym programista logiki miał już wcześniej styczność, tj. implementację *ActiveRecord* z *Ruby on Rails*. W trakcie pracy nad projektem doceniono stosowanie konwencji nazewniczych, których obecność znacząco upraszcza projektowanie klas mapujących dane.

6.2.13 Inwencja programistów

W trakcie rozwoju oprogramowania pojawiło się kilka 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, ankietowemu ukazuje się widok badania. Architekt nie zaproponował jak ma on wyglądać. Dał tylko pewne wskazówki. Zazaczył, 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”. Praktycznie cała strona pozostawała pusta. Sytuacja taka jest niedopuszczalna, bo na pewno istniały jakieś przydatne informacje, które można było w tym miejscu wyświetlić. Wykonocypowano, 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. Znajduje 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.

6.3 Użyte technologie

6.3.1 Moodle

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

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

6.3.2 PHP

PHP – platforma *Moodle* jest napisana właśnie w tym języku programowania. Z tego względu, jest to technologia zastosowana w większości rozszerzeń utworzonych przez zespół /textitiQuest, 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 druga wymieniona 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. Projekt ten składa się m.in. z następującego oprogramowania:

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

6.3.5 PostgreSQL

System zarządzania bazą danych *PostgreSQL* został wybrany ze względu na wymaganie 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. Udostępnia m.in. narzędzia do analizy poprawności składniowej pisanego kodu, formatery kodu, wyszukiwanie fraz w wielu plikach, kontekstowe podpowiedzi i nawigację.

6.3.7 SVN

Subversion został wybrany jako podstawowy system kontroli wersji ze względu na wymagania pozafunkcyjne. Zespół eksploatacji, który docelowo przejmie zarządzanie artefaktami związanymi z projektem, wykorzystuje właśnie *SVN*. Główne funkcjonalności tego systemu to: atomowe publikowanie zmian, historia operacji na plikach (zmiana nazwy, skopiowanie, przeniesienie, modyfikacja, usunięcie), wersjonowanie plików i folderów, łatwy dostęp do informacji o zmianach.

6.3.8 Redmine

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

6.3.9 JasperReports

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

6.3.10 JetBrains PhpStorm

PhpStorm jest komercyjnym *IDE* dla języka *PHP* tworzonym przez firmę *JetBrains*. Wykorzystano funkcjonalność odpowiedzialną za generację diagramów *UML* z kodu. Wygenerowane w trybie *Organic* diagramy można zobaczyć w dalszej części pracy.

6.3.11 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

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*.

6.5.1 Bezpieczeństwo

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

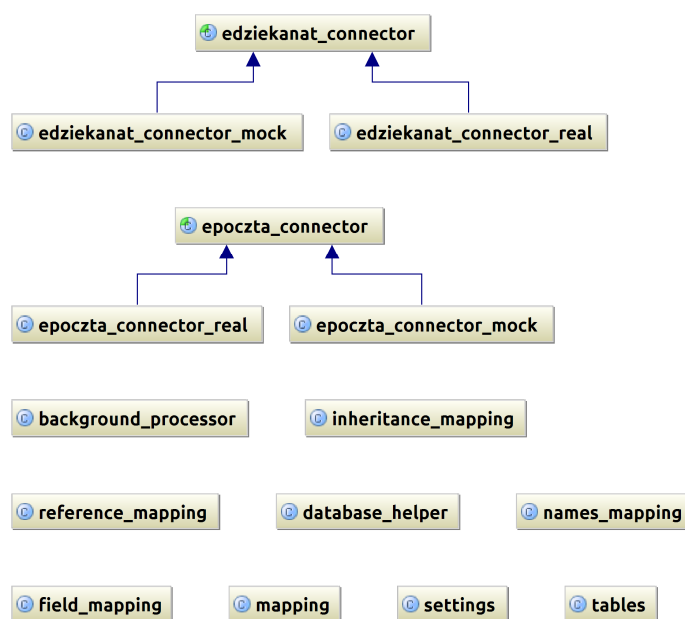
Zapewnienie bezpieczeństwa w systemie polega po pierwsze na tym, aby użytkownik przez pomyłkę nie wykonał czynności, do których nie został uprawniony. Sprowadzało się to do tego, aby ograniczyć użytkownikowi wyświetlane opcje tylko do tych, które może używać. Np. tylko tych odnośników, które prowadzą do dozwolonych dla użytkownika zasobów.

Po drugie, zapewnienie bezpieczeństwa wiąże się z odmową dostępu do zabronionych zasobów użytkownikom, których intencją było dostanie się do nich. Dlatego sprawdzano dane, które przychodzą do serwera. Np. utworzono zmienne związane w zapytaniu *SQL* w celu obrony przed atakiem *SQL injection*. Co więcej przed wyświetleniem treści użytkownikowi sprawdzano, czy użytkownik może je przeczytać. Dzięki temu niepowołany użytkownik nie przeczyta niedozwolonych treści wpisując bezpośrednio adres do żądanej podstrony w pasku adresu przeglądarki.

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

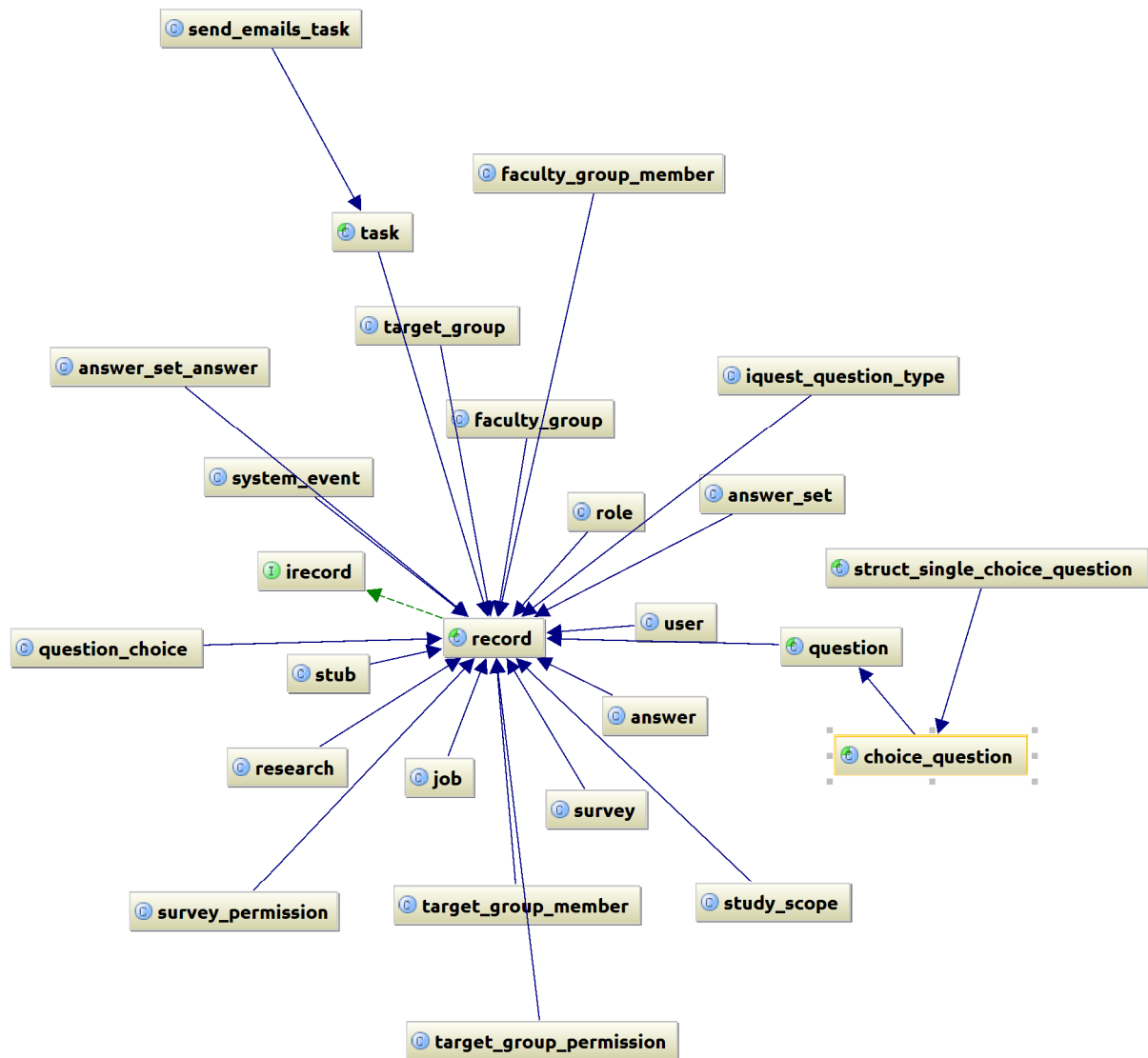
wykonywanych w tle oparty na *cron*; moduły odpowiadające za komunikację z systemami uczelnianymi *ePoczta* i *eDziekanat*; moduł logowania zdarzeń. W trakcie implementacji zdecydowano się nie tworzyć osobnego mechanizmu do przechowywania ustawień w bazie danych i skorzystaliśmy z istniejącego już w *Moodle*. Jednym z wymagań pozafunkcyjnych 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 dostarczanym przez platformę *Moodle* – *Data manipulation API*. Na diagramie poniżej znajdują się także klasy przechowujące stałe: (*tables*, *settings*).



Powered by yFiles

RYSUNEK 6.2: Struktura back-end'u (1)

RYSUNEK 6.3: Struktura back-end'u (2)



6.6.1 Raporty

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

- *JasperReports Server* (wersja 5.0) – serwer usług raportowania, na którym przechowywane są przygotowane przez zespół artefakty, w celu umożliwienia generacji raportu osobom dysponującym odpowiednimi uprawnieniami. Wykorzystano następujące funkcjonalności: definiowania źródła danych, raportu, ładowania plików z projektem raportu, zasobami oraz z generacji raportu.
- *Jaspersoft Studio* (wersja 1.3.2) – bazowane na Eclipse narzędzie do projektowania raportów. Posłużyło zespołowi do przygotowania projektów raportów w formacie *JRXML*.

Kody źródłowe pakietu *JasperReports* są pisane w języku Java. Źródłem danych dla raportu, jest przygotowana przez zespół implementacja interfejsu *ReportDataSourceService* z *API JasperServer*. Źródło danych łączy się z 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 prezentowana jest lista odpowiedzi. Dla pytań zamkniętych na podstawie pobranych danych generowane są statystyki, które przekazywane są do podraportu w postaci obiektu klasy *JRBeanCollectionDataSource*. W generacji statystyk z danych badań wykorzystano bibliotekę *JoSQL*. Definicja projektu raportu składa się z czterech plików, odpowiadających trzem 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*) wykorzystano Apache Axis. Wygenerowane klasy dostosowano tak, by akceptowały obiekty z zadanej instancji *iQuest* oraz dla zmiennej przestrzeni nazw (ang. *namespace*).

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

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

6.6.2 Moduły uwierzytelniania

Korzystając z mechanizmów rozszerzeń *Moodle* zaimplementowano 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 zadany czasie (e.g. 15 min.) zmodyfikowano pliki źródłowe *Moodle*, gdyż dla kodu odnoszącego się do sesji użytkownika *Moodle* nie została przewidziana możliwość rejestracji rozszerzeń. Relacja *user* została rozszerzona o opcjonalne pola związane z *eKonto*, jako że kod odpowiedzialny za manipulację schematem bazy danych nie jest wykonywany podczas instalacji modułu uwierzytelniania, umieszczono go w osobnym module (*ekontodb*). *eKontoAuthenticationPlugin* może być instalowany bez konieczności instalacji *iSurvey*. Podczas jego implementacji 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 respondent 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 ich studentach.

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

6.7 Powiązanie logiki z interfejsem

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

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

Mimo większej złożoności, zastosowano ten sposób, aby w jak największej części projektu użyć programowania obiektowego, które pozwala na lepszą organizację kodu, a przez to np. na szybsze wykrycie ewentualnych błędów oraz minimalizację powielania kodu. Oczywiście są to tylko niektóre z zalet programowania obiektowego.

Koniec fragmentu niezredagowanego

Rozdział 7

Zapewnianie jakości i konserwacja systemu

7.1 Testy i weryfikacja jakości oprogramowania

7.1.1 Wstęp

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

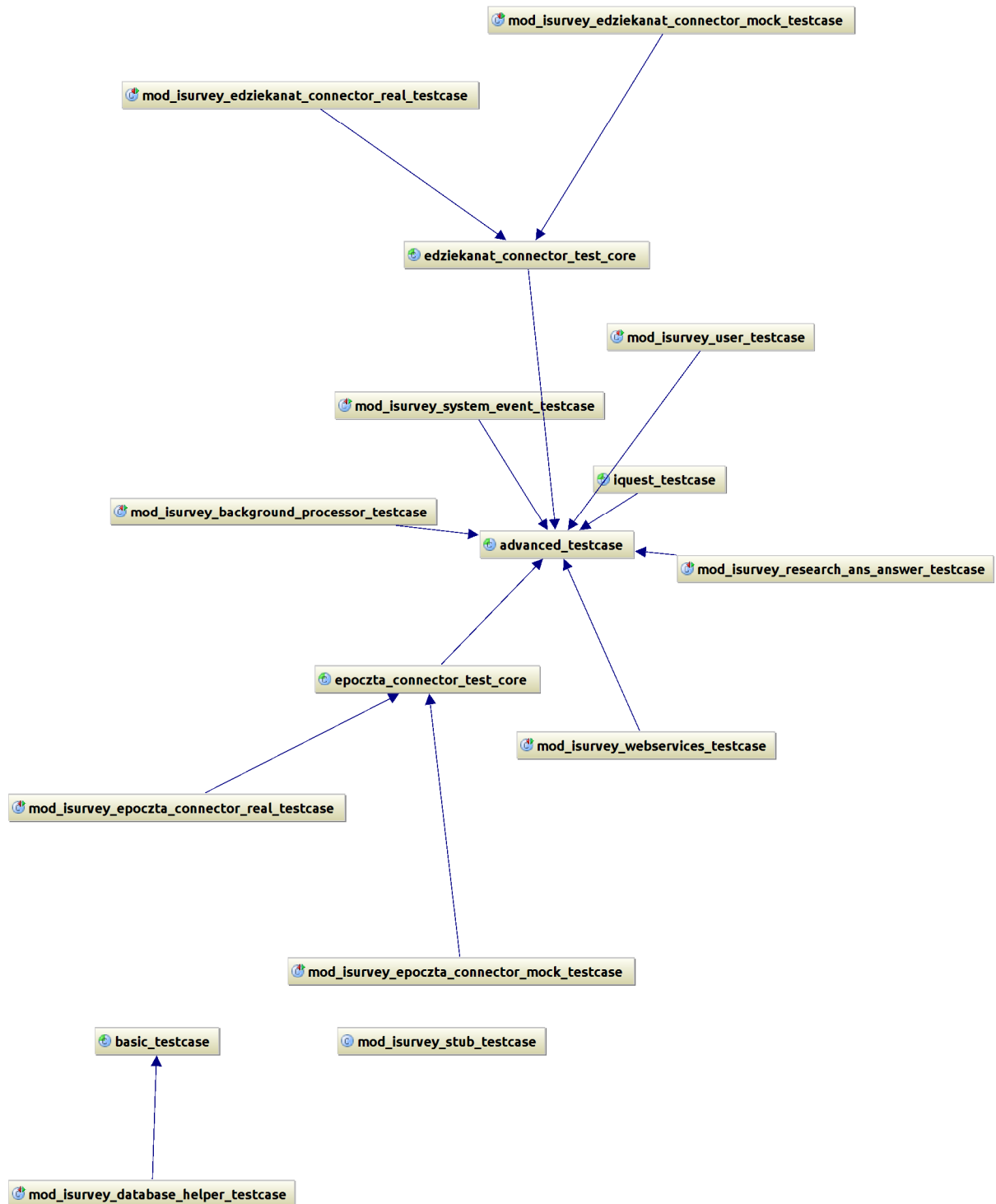
7.1.2 Testy jednostkowe

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

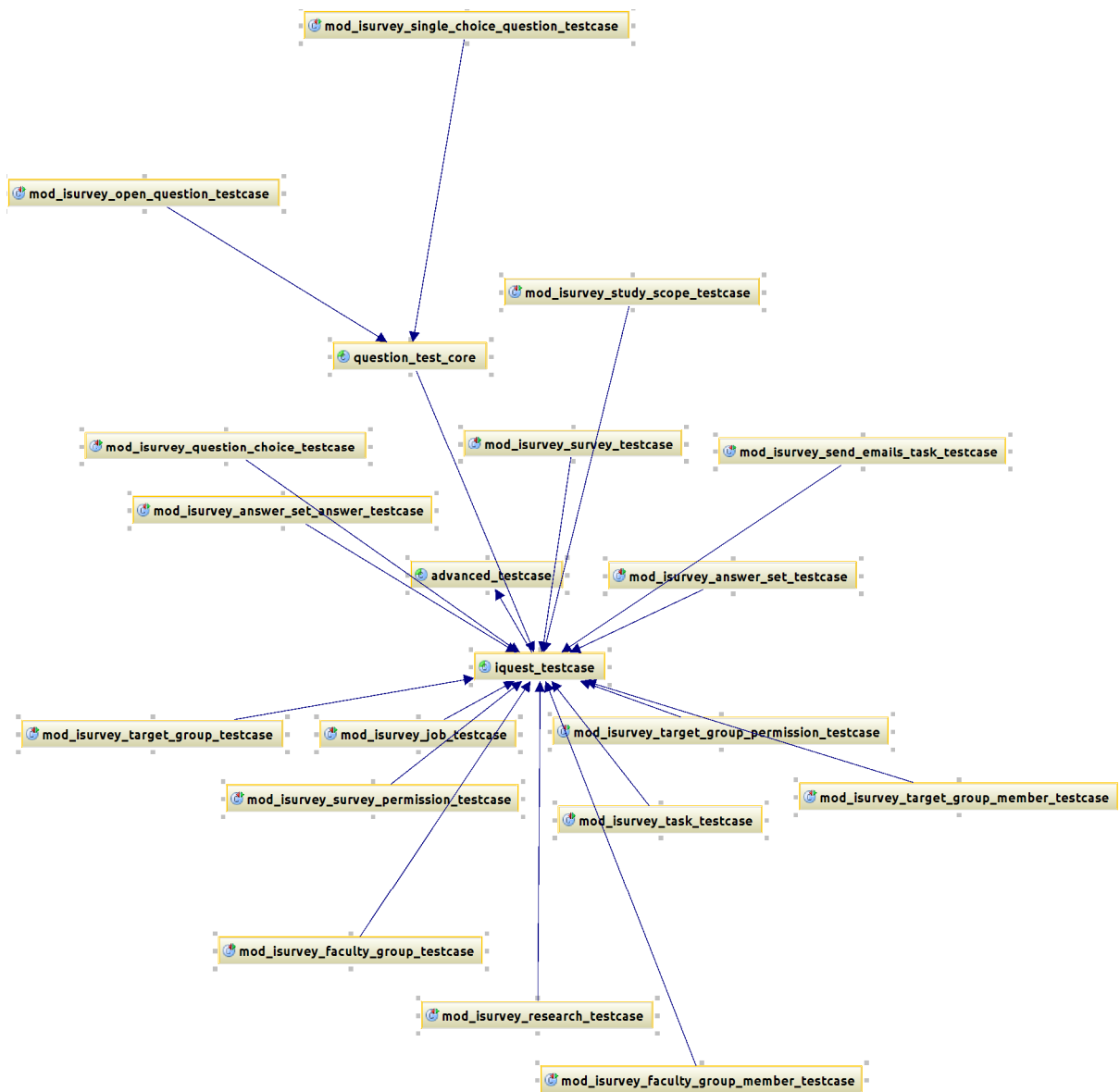
Przy realizacji pierwszego wydania, za testy jednostkowe w pełni odpowiadał jeden z członków zespołu programistów. W wydaniu drugim, rolę tę przejął programista realizujący logikę systemu. Z początku próbowaliśmy działać w oparciu o TDU (ang. *Test-Driven Development* - rozwój w oparciu o testy), jednakże szybko porzuciliśmy tą metodykę, ze względu na brak doświadczenia programistów.

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

RYSUNEK 7.1: Struktura klas testujących (1)



RYSUNEK 7.2: Struktura klas testujących (2)



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

- *basic_testcase* – podstawowe testy jednostkowe
- *advanced_testcase* – testy z użyciem bazy danych
- *iquest_testcase* – rozszerzenie *advanced_testcase* na potrzeby testowania klas dziedziczących z *record*

7.1.3 Testy akceptacyjne

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

MAT

Testy akceptacyjne wymagają poprawienia – dane od zespołu zarządzającego były nieaktualne!!! Poniżej przedstawiono Manualne Testy Akceptacyjne:

TC1 : Logowanie do systemu przez eKonto		
Warunki początkowe		
<ul style="list-style-type: none"> • Użytkownik jest niezalogowany • Użytkownik posiada eKonto • Połączenie z Internetem 		
Krok	Akcja	Oczekiwana odpowiedź
1.	Użytkownik wpisuje adres systemu	Strona logowania do system iQuest
2.	Użytkownik naciska przycisk „Zaloguj przez eKonto”	Strona logowania eLogin
3.	Użytkownik wpisuje dane logowania	
4.	Użytkownik naciska przycisk „Zaloguj”	Przekierowanie na stronę systemu iQuest, wyświetlenie strony głównej z zalogowanym użytkownikiem.
Uwagi		

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

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

TABLICA 7.1: Poprawne dane dla scenariusza TC2

TC2.2 : Stworzenie Ankiety - brak pytań		
Warunki początkowe		
<ul style="list-style-type: none"> • Zalogowany użytkownik z prawem do tworzenia ankiet 		
Krok	Akcja	Oczekiwana odpowiedź
1.	Użytkownik wybiera przycisk „Stwórz ankietę”	Strona umożliwiająca tworzenie ankiet
2.	Użytkownik podaje nazwę ankiety	
3.	Użytkownik podaje wstęp i podsumowanie ankiety	
4.	Użytkownik wybiera przycisk „Zapisz zmiany”	Komunikat o braku pytań w ankiecie
Uwagi		

TC3 : Edycja ankiety		
Warunki początkowe		
<ul style="list-style-type: none"> • Zalogowany użytkownik z prawem do tworzenia ankiet • Użytkownik posiada prawo do edycji ankiety „Ankiety testowej” 		
Krok	Akcja	Oczekiwana odpowiedź
1.	Użytkownik wybiera przycisk „Katalog Ankiet”	Strona z listą ankiet do których prawo ma użytkownik.
2.	Użytkownik wybiera przycisk „edytuj” przy „Ankiecie Testowej”	Strona umożliwiająca edycję „Ankiety Testowej”
3.	Użytkownik wciska przycisk „usuń” przy pytaniu drugim	Pytanie drugie znika
4.	Użytkownik naciska przycisk „Dodaj odpowiedź” przy pytaniu pierwszym	Pojawia się pole do wpisania możliwej odpowiedzi
5.	Użytkownik wpisuje możliwą odpowiedź	
6.	Użytkownik wybiera przycisk „Zapisz zmiany”	Strona wyświetla komunikat potwierdzający zapisanie zmian w ankiecie.
Uwagi		

Krok	Dane
5	„nie wiem”

TABLICA 7.2: Poprawne dane dla scenariusza TC3

TC3 : Edycja ankiety - usunięcie wszystkich pytań		
Warunki początkowe		
<ul style="list-style-type: none"> • Zalogowany użytkownik z prawem do tworzenia ankiet • Użytkownik posiada prawo do edycji ankiety „Ankiety testowej” 		
Krok	Akcja	Oczekiwana odpowiedź
1.	Użytkownik wybiera przycisk „Katalog Ankiet”	Strona z listą ankiet do których prawo ma użytkownik.
2.	Użytkownik wybiera przycisk „edytuj” przy „Ankiecie Testowej”	Strona umożliwiająca edycję „Ankiety Testowej”
3.	Użytkownik wciska przycisk „usuń” przy każdym z pytań	Pytania znikają
4.	Użytkownik wybiera przycisk „Zapisz zmiany”	Strona wyświetla komunikat, że ankieta nie posiada pytań i prosi o ich dodanie.
Uwagi		

TC4 : Wybranie grupy docelowej		
Warunki początkowe		
<ul style="list-style-type: none"> • Zalogowany użytkownik z prawami do tworzenia ankiet • Użytkownik posiada prawa do ankiety „Ankieta Testowa” • Użytkownik posiada prawo do ankietowania grupy docelowej „test” 		
Krok	Akcja	Oczekiwana odpowiedź
1.	Użytkownik wybiera przycisk „Włącz tryb edycji”	Interfejs edycji Moodle
2.	Użytkownik wybiera przycisk „edytuj” przy „Ankiecie Testowej”	Strona umożliwiająca edycję „Ankiety Testowej”
3.	Użytkownik wybiera grupę docelową „test”	
4.	Użytkownik wybiera przycisk „Zapisz zmiany”	Strona wyświetla komunikat, że ankieta została zaktualizowana pomyślnie.
5.	Respondent otrzymuje email z powiadomieniem o ankiecie	
Uwagi		

TC5 : Udzielanie odpowiedzi		
Warunki początkowe		
<ul style="list-style-type: none"> • Zalogowany użytkownik • Użytkownik znajduje się w grupie docelowej ankiety „testowa” • Użytkownik nie odpowiadał udzielał odpowiedzi na ankietę „testowa” 		
Krok	Akcja	Oczekiwana odpowiedź
1.	System prezentuje ankiety na które użytkownik jeszcze nie odpowiedział	
2.	Użytkownik wybiera ankietę „testowa”	System prezentuje ankietę
3.	Użytkownik zaznacza odpowiedź na pytanie 1 jako „tak”	
4.	Użytkownik podaje odpowiedź na pytanie drugie jako „tak”	
5.	Użytkownik potwierdza wypełnienie ankiety przyciskiem „Wyślij”	System prezentuje ankiety na które użytkownik jeszcze nie odpowiedział oraz komunikat o pomyślnym przesłaniu odpowiedzi
6.	Respondent otrzymuje email z powiadomieniem o ankiecie	
Uwagi		

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

TC7 : Dodawanie grupy docelowej		
Warunki początkowe		
<ul style="list-style-type: none"> • Zalogowany administrator z prawami do tworzenia grup docelowych • Brak w systemie grupy docelowej „Grupa 1” 		
Krok	Akcja	Oczekiwana odpowiedź
1.	Administrator wybiera przycisk „Zarządzaj grupami docelowymi”	System prezentuje dostępne grupy docelowe w systemie
2.	Administrator wybiera przycisk „dodaj”	System prezentuje interfejs dodawania grupy docelowej
3.	Administrator podaje nazwę nowej grupy „Grupa 1” i wskazuje jej członków	
4.	Administrator wybiera grupę nadrzędną dla nowej grupy docelowej	System automatycznie zapisuje zmiany
Uwagi		
Test przygotowany na podstawie makiety systemu iQuest		

TC8 : Edycja grupy docelowej		
Warunki początkowe		
<ul style="list-style-type: none"> • Zalogowany administrator z prawami do tworzenia grup docelowych • Grupa docelowa „Grupa 1” istnieje w systemie 		
Krok	Akcja	Oczekiwana odpowiedź
1.	Administrator wybiera przycisk „Zarządzaj grupami docelowymi”	System prezentuje dostępne grupy docelowe w systemie
2.	Administrator wybiera przycisk „zmień nazwę”	System prezentuje interfejs zmiany nazwy grupy docelowej
3.	Administrator pozostawia nazwę niezmienną i zatwierdza zmiany	System prezentuje dostępne grupy docelowe w systemie
4.	Administrator wybiera grupę nadrzędną dla grupy docelowej i dodaje nowego członka do grupy	System automatycznie zapisuje zmiany
Uwagi		
Test przygotowany na podstawie makiety systemu iQuest		

TC9 : Edycja grupy docelowej		
Warunki początkowe		
<ul style="list-style-type: none"> • Użytkownik jest niezalogowany • Istnieje konto użytkownika w systemie 		
Krok	Akcja	Oczekiwana odpowiedź
1.	Użytkownik wpisuje adres systemu	Strona główna moodla z kursem iQuest
2.	Użytkownik naciska przycisk „Zaloguj się”	Strona logowania
3.	Użytkownik wpisuje dane logowania	
4.	Użytkownik naciska przycisk „Zaloguj się”	Przekierowanie na główną stronę moodla z kursem iQuest. Wyświetla się napis: „Jesteś zalogowany(a) jako...”
Uwagi		

Niedodane jeszcze TC: Logowanie bez eKonta,

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

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

AAT

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

7.1.4 Inne metody zapewniania jakości

Celem zapewnienia jak najwyższej jakości oprogramowania, było ono testowane – w kontrolowanych warunkach – na różnorodnych maszynach. Co prawda, lokalne serwery developerskie pracowały

w oparciu o system Ubuntu 12.04 LTS, z serwerem Apache i systemem zarządzania bazą danych PostgreSQL, jednak maszyny klienckie były już znacznie bardziej różnorodne.

System przetestowano na zbiorze komputerów stacjonarnych klasy PC oraz równoważnych laptopów, z użyciem systemów Windows i Linux. Realizowano je także na laptopach i netbookach klasy PC oraz Macintosh, również pod kontrolą systemów Windows i Linux, oraz - w przypadku tej ostatniej klasy - MacOS. Ponadto, przetestowano trzy główne platformy mobilne (Android, iOS, WP7.x) poprzez dostęp do systemu z poziomu telefonów komórkowych.

Wspomniane wyżej maszyny pracowały pod kontrolą następujących systemów operacyjnych:

- Windows XP Professional SP3 32-bit + Internet Explorer 7
- Windows Vista Business SP2 32-bit + Internet Explorer 8
- Windows 7 Professional SP1 64-bit + Mozilla Firefox + Google Chrome
- Windows 8 Professional 64-bit + Internet Explorer 10 + Google Chrome
- Ubuntu 12.04 64-bit + Mozilla Firefox + Google Chrome
- Mac OS X + Apple Safari
- Google Android 2.2, 2.3, 4.0, 4.1
- iOS (iPhone 4S)
- Windows Phone 7.5 (Nokia Lumia 710)

Na podstawie powyższych testów, utworzono dwa raporty: „wygląd i działanie systemu iQuest na platformach mobilnych” oraz „wygląd i działanie systemu iQuest w różnych konfiguracjach system-przeglądarka”, udostępnione w ramach systemu zarządzania projektem[?]. Wynika z nich, że system iQuest posiada bardzo wysoki współczynnik przenośności.

Rozdział 8

Zebrane doświadczenia

Podczas pracy zebraliśmy następujące doświadczenia bezpośrednio związane z zastosowanymi technologiami:

- Implementowanie testów jednostkowych z użyciem PHPUnit – w celu kontroli poprawności pisanego kodu back-end’u w języku PHP,
- Implementowanie logiki aplikacji w języku PHP na podstawie UML – na podstawie diagramów UML oraz rozmów z architektem zaimplementowano kod back-end’u w języku PHP,
- Implementowanie usług internetowych (protokół *SOAP*) – dla komunikacji z usługami uczelni wykorzystano klienty dostarczone przez Dział Rozwoju Oprogramowania w języku PHP, dla komunikacji z serwerem raportowania wykorzystano API *external services* platformy Moodle (PHP) po stronie systemu iQuest oraz Apache Axis (Java) po stronie serwera JasperReports,
- Implementowanie schematu bazy danych w formacie XMLDB na podstawie UML – wykorzystano interfejs do zarządzania XMLDB dostarczany przez Moodle; drobne poprawki wprowadzano ręcznie w pliku XML definiującym schemat bazy danych,
- Implementowanie modułów uwierzytelniania systemu Moodle – na podstawie przykładowych oraz istniejących modułów uwierzytelniania stworzono moduł uwierzytelniania przez eKonto oraz moduł dla absolwentów,
- Rozszerzanie funkcjonalności oprogramowania o otwartym kodzie źródłowym – w celu spełnienia wymagań projektowych dodano m.in. okresowe sprawdzanie ważności sesji eKonto do mechanizmu zarządzania sesją Moodle, zmodyfikowano wygląd strony logowania, panelu użytkownika,
- Konfigurowanie systemów operacyjnych Ubuntu i OpenSUSE – W celu zdalnej konfiguracji wykorzystywano protokół SSH; Należało zainstalować i skonfigurować wymagane oprogramowanie (w tym Apache, cron, PostgreSQL, PHP, Check Point’s Linux SNX), przygotować katalogi repozytoriów kodu,
- Programowanie z użyciem Eclipse PDT – wykorzystano zintegrowane środowisko programistyczne w celu zwiększenia produktywności programistów,
- Konfigurowanie JasperServer – dodano dialekt zapytań JoSQL, własne źródło danych (pomocne okazały się informacje z projektu *Business Intelligence Server* dostępnego na Redmine),

- Projektowanie raportów JasperReports – obejmowało zaprojektowanie raportów w JasperReports Studio,
- Korzystanie z klientów VPN (firmy CheckPoint) – w celu uzyskania dostępu do sieci wewnętrznej zobligowano zespół do zestawiania połączenia VPN,
- Korzystanie z systemu zarządzania projektami Redmine – obejmuje zarządzanie zagadnieniami, bazą wiedzy, repozytorium, plikami, korzystanie z dostępnych metod komunikacji: komunikaty, forum, komentarze,
- Korzystanie z systemów kontroli wersji SVN i Git – SVN wykorzystano jak repozytorium kodu, repozytorium Git służyło zespołowi podczas pracy nad niniejszą pracą,
- Pisanie dokumentacji technicznej oraz użytkownika,
- Implementowanie modułów aktywności systemu Moodle,
- Projektowanie testów akceptacyjnych z użyciem Selenium IDE,
- Implementowanie formularzy z wykorzystaniem JavaScript,

Bardzo istotna część doświadczeń związanych z projektem wiąże się z pracą zespołową. Nauczyliśmy się, że dobrego członka zespołu wyróżnia: sumienność, punktualność, odpowiedzialność, dokładność, terminowość, prawdomówność, uczciwość, asertywność, zdolność kreatywnego myślenia, kultura bycia oraz komunikatywność. Praca w grupie nad dużym projektem wymaga dobrej koordynacji prac. Dostępność architekta i kierownika projektu znacząco ułatwiła pracę nad projektem. Dzięki podziałowi zadań, każdy członek zespołu mógł pogłębić swoje zainteresowania. Wraz z upływem każdy członek zespołu znalazł sobie zbiór typów zagadnień, które potrafił zrealizować szybciej i lepiej ze względu na wcześniej pozyskane doświadczenie. Uzyskane doświadczenia z pewnością przydadzą się nam w pracy zawodowej.

Wnioski:

Krzysztof Borowiak:

- ...

Maciej Trojan

- ...

Krzysztof Urbaniak:

- Kluczem do sprawnej pracy jest dobra organizacja zasobów. System *Redmine* bardzo pomagał w odpowiednim przydzieleniu zadań do osób tak, aby projekt skończyć w określonym czasie.
- Wspólna praca zespołu w jednym pokoju wpływa na większą świadomość jego członków w postępie prac. Przekłada się to na przykład na szybsze znajdowanie błędów, czy zbiorowe rozwiązywanie większych problemów.
- Linux jest o wiele lepszym systemem operacyjnym do programowania niż Windows. Linux góruje ilością programów operujących na plikach tekstowych i łatwością pisania skryptów. Programista pracujący na systemie Linux ma poczucie, że wiele procesów można zautomatyzować, w przeciwieństwie do pracy na systemie Windows.

- Nie jest dobrym pomysłem wybieranie rozwiązań, o których ma się szcątkowe pojęcie. Lepiej jest wybrać pewne rozwiązania, nawet jeżeli wydaje się, że wymagają więcej zaangażowania.
- Wybór pracy inżynierskiej pod nadzorem zespołu SDS był dobrym pomysłem. Wsparcie architekta oraz analityka było bardzo przydatne, a nawet niezbędne do ukończenia pracy w terminie.

Łukasz Wieczorek:

- Zmiany zatwierdza się (ang. *commit*) dopiero, gdy funkcjonalność, którą należało zaimplementować, jest kompletna i przetestowana jednostkowo oraz akceptacyjnie,
- Warto przygotować wcześniej logikę, z której będą korzystać programiści interfejsu, by nie opóźniać ich pracy,
- W przypadku braku wiedzy odnośnie sposobu implementacji funkcjonalności najlepiej skonsultuj się z architektem – zwykle dysponuje on większą wiedzą ogólną,
- Werbalizacja problemu bardzo często pomaga w jego rozwiązaniu,
- Wcześniejsza refaktoryzacja znacznie ułatwia dalszą pracę nad kodem,
- Wielokrotnie powtarzane fragmenty kodu należy jak najszybciej zrefaktoryzować, by uchronić się od poprawiania podobnych konstrukcji składniowych,
- „Nie należy mnożyć bytów ponad potrzebę”,
- Testy jednostkowe ułatwiają refaktoryzację kodu – nawet jeśli ich utrzymywanie jest kosztowne, warto je pisać,
- Podstawą efektywnej pracy jest kontrola czasu poświęcanego na realizację przydzielonych zadań,
- Dobra komunikacja w zespole to podstawa sukcesu,

Rozdział 9

Zakończenie

9.1 Podsumowanie

Do uzupełnienia pod koniec prac. Rozwijanie istniejącego oprogramowania wymaga dużo większego nakładu pracy niż konstrukcja oprogramowania od podstaw. Wiele czasu poświęca się na analizę rozwiązań zastosowanych przez twórców rozszerzanego rozwiązania. W trakcie implementacji pojawiają się problemy, których rozwiązania szuka się długimi godzinami; niektórych nie udaje się w ogóle rozwiązać, co z kolei prowadzi do modyfikacji czyjegoś oprogramowania (ang. *hacking*). Praca nad oprogramowaniem, którego tworzenie zaczęło się przed ponad dziesięcioma laty, wymaga pracy z rozwiązaniami architektonicznymi, które dawno zostały już porzucone (np. *transaction script* na rzecz *Model View Controller*). Dodatkową trudnością są zmieniające się lub porzucane interfejsy programowania aplikacji. Wymienione problemy dotyczą szczególnie dużego oprogramowania, właśnie takiego z jakim przyszło nam pracować – Moodle to wg programu CLOC ponad 2 miliony linii kodu. Uważamy, że decyzja odnośnie rozwijania systemu Moodle była błędna i nieprzemyślana, zwłaszcza w kontekście niedoświadczenia programistów. Potwierdzenie naszych wniosków widzimy chociażby porównując wydajność pracy zespołu w porównaniu z innymi zespołami realizującymi projekty w ramach Software Development Studio. Błędem było oparcie projektu o rozwiązanie nieznane dobrze architektowi i programistom. Czas poświęcony na pracę wejścia okazał się być znacznie wyższy niż oszacowany przez kierownika projektu.

Możliwość współpracy z Software Development Studio była niewątpliwie zaletą. Po pierwsze, projekt, który należało wykonać był z naszej perspektywy bardzo złożony. Dzięki kierownikowi projektu oraz architektowi część pracy związana z projektowaniem i dokumentowaniem systemu nie obciążała programistów, pozwalając nam na koncentrację sił na implementacji systemu. Po drugie, wszystkie ewentualnie nieдомówienia projektowe, będące przyczyną sporów programistów, były szybko rozwiązywane w ramach konsultacji z architektem. Po trzecie, udostępniony system Redmine znacząco ułatwia organizację pracy. Dzięki „śledzeniu zagadnień” można lepiej organizować czas i przydział zadań. Pokój, w którym przyszło nam pracować był udogodnieniem, które dobrze wpłynęło na komunikację w zespole. Korzyścią wynikającą z zadanej nam organizacji pracy jest konieczność trzymania się terminów, dzięki której nasz ostatni semestr studiów inżynierskich był dobrze zaplanowany pod względem równomiernego rozkładu pracochłonności.

Mamy nadzieję, że powstały system zostanie ciepło przyjęty zarówno przez studentów, absolwentów oraz prowadzących.

9.2 Propozycja dalszych prac

Do uzupełnienia pod koniec prac. Podczas spotkania z reprezentantami Działu Rozwoju Oprogramowania poruszony został temat etykietowania (ang. *tag*) badań. Proponowany mechanizm z pewnością usprawniłby proces wyszukiwania badań. W obecnej wersji systemu zachętą dla studentów do wypełniania ankiet są materiały publikowane przez wykładowców uczelni, do których dostęp przyznawany jest użytkownikom, którzy w określonym czasie udzielili odpowiedzi w dowolnym badaniu. W przyszłości mechanizm ten można by rozbudować o wirtualną walutę służącą do „wykupowania” dostępu do publikowanych materiałów.

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 Wiczorek

TABLICA A.1: Osoby związane z przedsięwzięciem

Odpowiedzialność za utworzenie treści niniejszej pracy dyplomowej została przedstawiona poniżej:

Krzysztof Marian Borowiak

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

- Dodatki

Maciej Trojan

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

Krzysztof Urbaniak

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

Łukasz Wieczorek

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

Zespół zarządzający projektem (Marcin Domański, Błażej Matuszyk)

- Materiały wskazane w bibliografii[?], zastosowane jako baza dla rozdziałów 1-5
- Część grafik (z odpowiednim odniesieniem w etykiecie)

Odpowiedzialność za część implementacyjną systemu została przedstawiona poniżej:

Krzysztof Marian Borowiak

- Testy jednostkowe (wydanie 1)
- Testy akceptacyjne
- Dokumentacja dla Użytkownika Końcowego (Administratora, Użytkownika)
- Raport funkcjonowania na różnych platformach mobilnych
- Raport funkcjonowania w różnych przeglądarkach internetowych

Maciej Trojan

- Interfejs użytkownika
- Obsługa Bazy Danych

Krzysztof Urbaniak

- Interfejs użytkownika
- Powiązanie interfejsu z logiką
- Obsługa Bazy Danych

Łukasz Wieczorek

- Logika
- Testy jednostkowe (wydanie 2)
- System raportowania
- Obsługa Bazy Danych

Serdeczne podziękowania należą się zespołowi zarządzającemu, przygotowane przez który materiały, choć wymagające znaczącej redakcji, pomogły w utworzeniu niniejszej pracy dyplomowej.

A.2 Wykaz użytych narzędzi i technologii

Numery w nawiasach w poniższej liście oznaczają numer wersji.

- Apache (2.2.22)
- BASH (4.2.37)
- Check Point's Linux SNX (800007027)
- Chrome (24.0)
- CLOC (1.56)
- CRON (3.0)
- Eclipse IDE (3.7.2)
- FastStone Capture (5.3)
- Git (1:1.7.10.4)
- JasperReports Studio (1.3.2)
- JasperReports Server (5.0.1)
- Java (1.6.0_38)
- JavaScript
- JetBrains PhpStorm (5.0.4)
- Kazam Screencaster (1.0.6)
- Meld (1.6.0)
- Moodle (2.3.1)
- Mozilla Firefox (18.0.1)

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

A.3 Zawartość płyty CD

Do dokumentu załączono płytę CD o następującej zawartości:

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

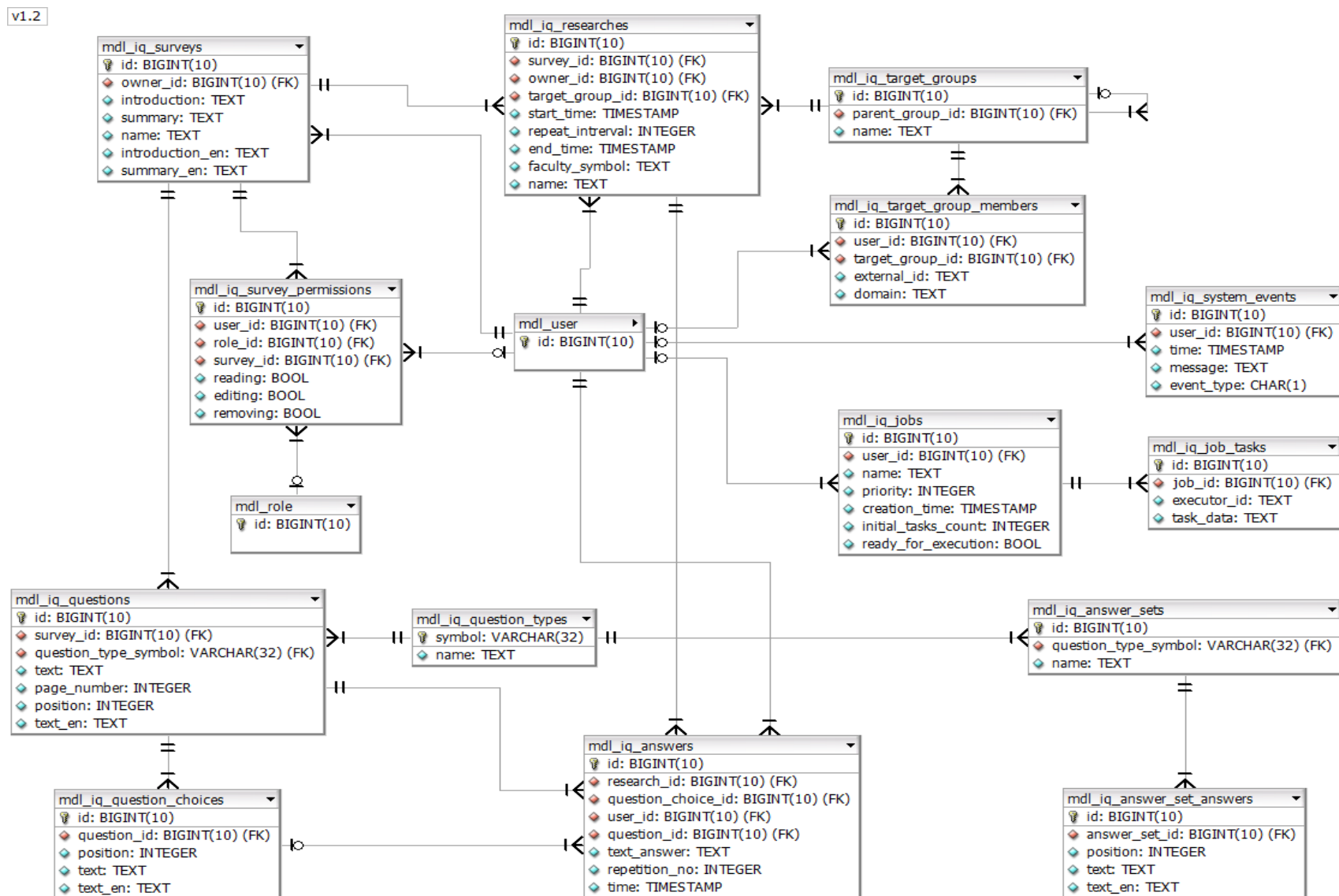
Dodatek B

Wygląd aplikacji

Tu będą zrzuty ekranowe z systemu iQuest, zamieszczone w instrukcji. Na chwilę obecną, zrzuty są dopiero wykonywane (trwają prace nad systemem).

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",  
}
```