Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki

Praca dyplomowa inżynierska

LINKEDINGRADS: OTWARTY SYSTEM MONITOROWANIA KARIER ZAWODOWYCH ABSOLWENTÓW

Adam Butkiewicz, 98938
Dominika Stempniewicz, 89827
Dawid Wengrzik, 100191
Joanna Zakrzewska, 99167

Promotor
dr hab. inż. Mikołaj Morzy

Poznań, 2014 r.

Spis treści

Rozdział 1

Wprowadzenie

1.1 Opis problemu i koncepcja jego rozwiązania

Zgodnie z rozporządzeniem Ministerstwa Nauki i Szkolnictwa Wyższego [1,2] każda uczelnia wyższa w Polsce jest zobowiązana do monitorowania karier swoich absolwentów, w celu pozyskania informacji o ich aktualnej sytuacji zawodowej. Oprócz spełnienia wymogów formalnych, systematyczne gromadzenie i analizowanie danych o zatrudnieniu absolwentów umożliwia uczelniom weryfikację jakości i efektywności kształcenia na poszczególnych wydziałach i kierunkach. Zebrane informacje stanowią cenną wskazówkę w ciągłym procesie doskonalenia oferty dydaktycznej uczelni, pomagając w dostosowywaniu kierunków studiów i programów kształcenia do potrzeb rynku pracy. Prowadzenie rzetelnych badań na temat losów zawodowych absolwentów i prezentowanie statystyk zatrudnienia jest również istotne z punktu widzenia wizerunku uczelni.

Ministerstwo Nauki i Szkolnictwa Wyższego nie narzuca uczelniom sposobu realizacji procesu monitorowania karier absolwentów. Większość uczelni wyższych, w tym Politechnika Poznańska, wywiązuje się z tego obowiązku za pomocą badania ankietowego. Opracowane ankiety są rozsyłane do absolwentów w formie elektronicznych lub drukowanych formularzy bądź przeprowadzane za pośrednictwem rozmów telefonicznych. Rozwiązania te są nie tylko kosztowne i czasochłonne, lecz charakteryzują się także niewielką efektywnością. Z szacunków Centrum Praktyk i Karier Politechniki Poznańskiej wynika„ że z możliwości dobrowolnego wypełnienia ankiety absolwenckiej korzysta poniżej 10

W związku z wymienionymi wadami dotychczasowych metod zaproponowano stworzenie systemu informatycznego, w postaci aplikacji internetowej, który monitorowałby kariery zawodowe absolwentów wykorzystując dane udostępniane przez nich w serwisie LinkedIn. Serwis LinkedIn stanowi jedną z największych sieci zawodowych, łącząc ponad 250 mln. użytkowników w 200 krajach i terytoriach na całym świecie. Stworzony system powinien w założeniach zautomatyzować i uskutecznić proces monitorowania, pobierać dane o zatrudnieniu absolwentów z serwisu LinkedIn oraz prezentować je w formie raportów o z góry określonej strukturze. Należy przewidzieć również możliwość integracji z innymi serwisami mogącymi posłużyć jako źródło danych o sytuacji zawodowej absolwentów.

System został zrealizowany na Wydziale Informatyki Politechniki Poznanskiej w ramach zajęć Studia Rozwoju Oprogramowania. Wykonanie systemu zostało zlecone przez rzeczywistego klienta, w postaci przedstawicieli władz wydziału i uczelni. Prace były prowadzone według przyjętej metodyki i harmonogramu.

1.2 Omówienie pracy

Niniejsza praca opisuje otwarty system monitorowania karier zawodowych absolwentów LinkedInGrads (ang. LinkedInGrads: graduate career tracking system), zwany dalej Systemem, realizujący koncepcję przedstawioną w punkcie 1.1. Praca stanowi dokumentację techniczną systemu, a także wyjaśnia idee stojące za poszczególnymi decyzjami projektowymi. Powinna być przydatna zarówno dla użytkowników końcowych systemu, jak i dla osób, które zamierzają go wdrożyć, utrzymywać bądź rozwijać. Jako praca dyplomowa inżynierska jest również skierowana do członków komisji egzaminacyjnej.

W rozdziale 2. przedstawiono aktorów, obiekty biznesowe oraz przypadki użycia występujące w systemie. W rozdziale 3. opisano wymagania funkcjonalne, a w rozdziale 4. wymagania pozafunkcjonalne, wraz z informacją, które z nich zostały zrealizowane. W rozdziale 5. omówiono ogólną architekturę systemu. Rozdział 6. zawiera szczegóły implementacji systemu oraz opis wykorzystanych koncepcji i technologii. W rozdziale 7. przedstawiono metody i narzędzia wspomagające zapewnienie jakości systemu. W rozdziale 8. opisano zebrane wnioski i doświadczenia. Rozdział 9. zawiera podsumowanie całości projektu oraz propozycje dalszego rozwoju systemu. W skład dokumentu wchodzi również bibliografia pracy oraz dodatki, obejmujące informacje uzupełniające, prezentację wyglądu aplikacji, instrukcję instalacji oraz scenariusze manualnych testów akceptacyjnych.

Rozdział 2

Opis procesów biznesowych

Niniejszy rozdział przedstawia otoczenie systemu LinkedInGrads. Wyszczególnieni zostali aktorzy: użytkownicy oraz zewnętrzne systemy. Zaprezentowano obiekty biznesowe będące rzeczywistością, w jakiej porusza się użytkownik systemu. Każdy obiekt opatrzono krótkim opisem oraz wykazem atrybutów. Ostatni podrozdział prezentuje biznesowe przypadki użycia.

2.1 Aktorzy i zewnętrzne systemy

- Administrator - osoba odpowiedzialna za aktualizację danych w systemie: dodawanie nowych absolwentów, znajdowanie adresów profili absolwentów.
- Pracownik dziekanatu - główny użytkownik systemu, grupuje pojęcia dotyczące absolwentów, generuje raporty.
- eLogin - zewnętrzny system Politechniki Poznańskiej służący do uwierzytelniania użytkowników.
- LinkedIn - sieć społecznościowa zrzeszająca specjalistów, służąca nawiązywaniu kontaktów i rozwojowi kariery.

2.2 Obiekty biznesowe

2.2.1 Raport

Raport jest obiektem biznesowym reprezentującym wycinek danych przetworzonych przez system. Dodatkowo każdy z raportów uwzględnia nagłówek zawierający podstawowe informacje na temat raportu oraz zestawienie stosunku ilości danych pochodzących z różnych źródeł. Raport może również zawierać uzasadnienie prezentowanych danych prezentując dodatkowo listę absolwentów wraz z wartością użytych do wygenerowania raportu.

- Analiza pracodawców - prezentuje ilość absolwentów zatrudnionych w danych firmach
- Analiza zamieszkania - prezentuje miejsca zamieszkania absolwentów
- Analiza umiejętności - prezentuje umiejętności nabyte przez studentów
- Analiza zatrudnienia - prezentuje stosunek osób zatrudnionych do bezrobotnych
- Analiza stanowisk - prezentuje ilość studentów na danych stanowiskach, które mogą zostać zgrupowane do bardziej uniwersalnego nazewnictwa. Dodatkowo uwzględnia historyczne stanowiska absolwentów.

Atrybuty:

- Typ raportu,
- Rok ukończenia studiów,
- Uzasadnienie danych.

2.2.2 Grupa

Grupa jest obiektem biznesowym reprezentującym alias dla nazewnictwa użytego w danych pobranych z zewnętrznych źródeł. Obiekt ten wykorzystywany jest wewnątrz raportu w celu unifikacji nazewnictwa użytego w zewnętrznych źródłach.

Atrybuty:

- zgrupowana nazwa,
- nazwa pierwotna.

2.3 Biznesowe przypadki użycia

2.3.1 Generowanie raportu o absolwentach

Przypadek użycia: BUC1: Generowanie raportu o absolwentach
Aktorzy: Pracownik dziekanatu
Pre: Istnieją dane użytkowników pobrane z zewnętrznych źródeł i wzorce raportów
Post: Raport
Scenariusz Główny
1. Pracownik wybiera typ raportu. 2. Pracownik wybiera filtry raportu. 3. Pracownik analizuje otrzymany raport.

2.3.2 Łączenie pojęć w grupy

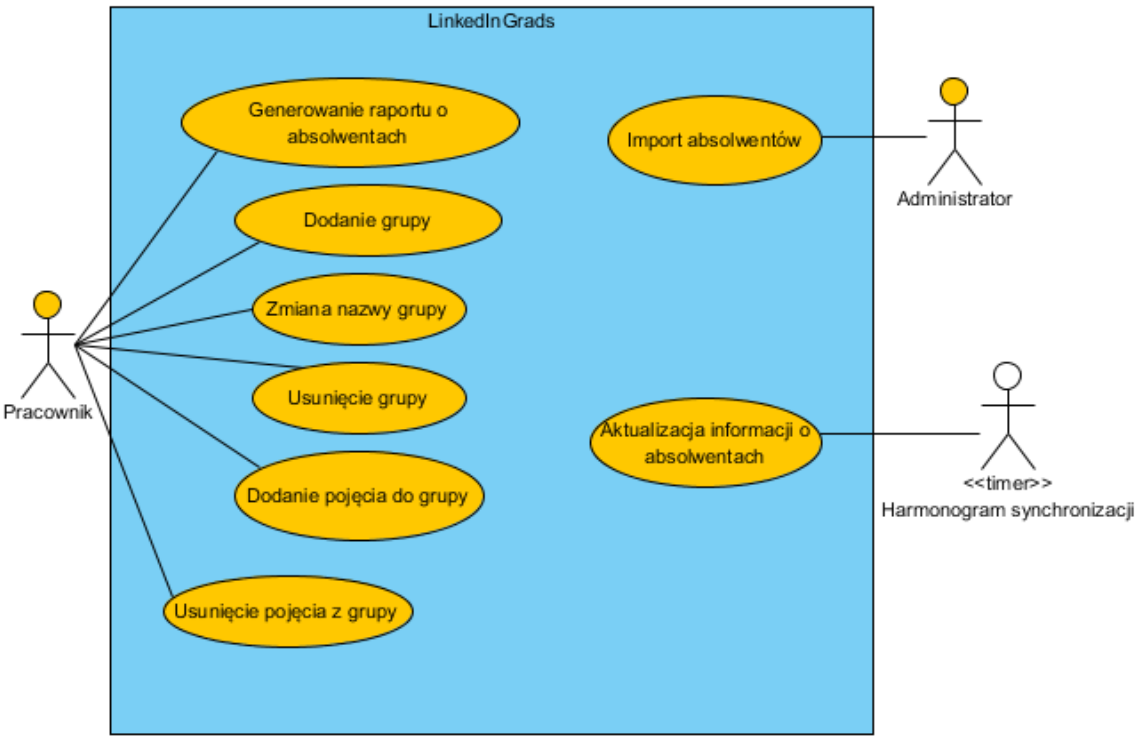
Przypadek użycia: BUC2: Łączenie pojęć w grupy
Aktorzy: Pracownik dziekanatu
Pre: Istnieją dane użytkowników pobrane z zewnętrznych źródeł
Post: Grupa
Scenariusz Główny
1. Pracownik tworzy nową grupę. 2. Pracownik przypisuje pojęcia do grupy.

Rozdział 3

Wymagania funkcjonalne

Wymagania funkcjonalne (ang. functional requirements) określają co system powinien oferować użytkownikowi, to jest jakie operacje można na nim wykonać. Ważne jest, by utrzymać kompletny zbiór poprawnie zdefiniowanych wymagań, pomaga to w zrozumieniu w jaki sposób powinien działać system, nawet dla osób które nie mają wiedzy technicznej oraz pomaga podczas jego projektowania.

Niniejszy rozdział przedstawia wymagania funkcjonalne za pomocą dwóch najpopularniejszych sposobów ich opisu, są to: przypadki użycia (ang. use cases) oraz opowieści użytkownika (ang. user stories). Pierwsza z metod polega na określeniu listy kroków. Reprezentuje ona interakcję między aktorem a systemem. Wykonanie kroków skutkuje osiągnięciem celu, który ma zapewnić system. Druga metoda - Opowieści użytkownika przedstawia w kilku zdaniach potrzebę użytkownika, którą ma realizować system.



Rysunek 3.1: Diagram przypadków użycia

3.0.3 Generowanie raportu o absolwentach

Przypadek użycia: UC1: Generowanie raportu o absolwentach
Aktorzy: Pracownik dziekanatu, System
Scenariusz Główny
1. Pracownik wybiera opcję generowania raportu. 2. System prezentuje typy raportów do wyboru. 3. Pracownik wybiera typ raportu. 4. System prezentuje rok ukończenia studiów do wyboru. 5. Pracownik wybiera rok ukończenia studiów. 6. Pracownik inicjuje generowanie raportu. 7. System prosi o wybranie ścieżki zapisu raportu. 8. Pracownik wybiera ścieżkę zapisu.

3.0.4 Dodanie grupy

Przypadek użycia: UC2: Dodanie grupy
Aktorzy: Pracownik dziekanatu, System
Scenariusz Główny
1. Pracownik wybiera opcję dodania grupy. 2. System prosi o podanie nazwy grupy. 3. Pracownik podaje nazwę grupy. 4. System informuje o pomyślnym dodaniu grupy.
Rozszerzenia
3.A. Grupa o podanej nazwie już istnieje. 3.A.1. System informuje o błędzie. 3.A.2. Powrót do kroku 3.

3.0.5 Zmiana nazwy grupy

Przypadek użycia: UC3: Zmiana nazwy grupy
Aktorzy: Pracownik dziekanatu, System
Scenariusz Główny
1. Pracownik wybiera opcję zmiany nazwy grupy. 2. System prezentuje listę istniejących grup. 3. Pracownik wybiera grupę. 4. System prosi o podanie nowej nazwy grupy. 5. Pracownik podaje nową nazwę grupy. 6. System informuje o pomyślnej zmianie nazwy grupy.
Rozszerzenia
5.A. Grupa o podanej nazwie już istnieje. 5.A.1. System informuje o błędzie. 5.A.2. Powrót do kroku 3.

3.0.6 Usunięcie grupy

Przypadek użycia: UC4: Usunięcie grupy
Aktorzy: Pracownik dziekanatu, System
Scenariusz Główny
1. Pracownik wybiera opcję usunięcia grupy. 2. System prezentuje listę istniejących grup. 3. Pracownik wybiera grupę do usunięcia. 4. System informuje o pomyślnym usunięciu grupy.

3.0.7 Dodanie pojęcia do grupy

Przypadek użycia: UC5: Dodanie pojęcia do grupy
Aktorzy: Pracownik dziekanatu, System
Scenariusz Główny
1. Pracownik wybiera opcję grupowania pojęć. 2. System prezentuje listę istniejących grup. 3. Pracownik wybiera grupę. 4. System prezentuje listę pojęć niezgrupowanych. 5. Pracownik wybiera pojęcie do dodania. 6. System informuje o pomyślnym dodaniu pojęcia do grupy.
Rozszerzenia
3.A. Grupa została w międzyczasie usunięta. 3.A.1. System informuje o błędzie. 3.A.2. Powrót do kroku 2. 5.A. Grupa została w międzyczasie usunięta. 5.A.1. System informuje o błędzie. 5.A.2. Powrót do kroku 2.

3.0.8 Usunięcie pojęcia z grupy

Przypadek użycia: UC6: Usunięcie pojęcia z grupy
Aktorzy: Pracownik dziekanatu, System
Scenariusz Główny
1. Pracownik wybiera opcję grupowania pojęć. 2. System prezentuje listę istniejących grup. 3. Pracownik wybiera grupę. 4. System prezentuje listę pojęć przypisanych do grupy. 5. Pracownik wybiera pojęcie do usunięcia. 6. System informuje o pomyślnym usunięciu pojęcia z grupy.
Rozszerzenia
3.A. Grupa została w międzyczasie usunięta. 3.A.1. System informuje o błędzie. 3.A.2. Powrót do kroku 2.

3.0.9 Aktualizacja informacji o absolwentach

Opowieść użytkownika: US1
Opis: Przebieg aktualizacji informacji o absolwentach. Odbywa się automatycznie, w tle działającego systemu.
Treść: Jako Pracownik chcę aby System, co ustalony czas pobierał informacje o absolwentach z portalu LinkedIn, a następnie aktualizował swoją bazę danych.

3.0.10 Import absolwentów

Przypadek użycia: UC7: Import absolwentów
Aktorzy: Administrator, System
Scenariusz Główny
1. Pracownik wywołuje skrypt importu absolwentów. 2. System prezentuje listę dodanych absolwentów.
Rozszerzenia
1.A. Niepoprawne dane wejściowe. 1.A.1. System przerywa import i informuje o błędzie. 1.A.2. Powrót do 1.

3.0.11 Grupowanie pojęć

Opowieść użytkownika: US2
Opis: Generowanie raportu z użyciem mechanizmu grupowania pojęć.
Treść: Jako Pracownik chcę aby pojęcia (umiejętności, stanowiska, organizacje) w wygenerowanym raporcie były pogrupowane (np. za pomocą metody LDA) w bardziej ogólne zbiory.

Rozdział 4

Wymagania pozafunkcjonalne

Wymagania pozafunkcjonalne (ang. non-functional requirements) określają jakość sposobu realizacji funkcji przez system. Mimo, że wymagania funkcjonalne uda się spełnić, to nie zawsze realizacja celu jest w stanie usatysfakcjonować odbiorcę systemu. Dobrze sprecyzowane wymagania pozafunkcjonalne pozwalają na uniknięcie niskiej użyteczności systemu oraz mogą mieć wpływ na sam sposób realizacji projektu. Stworzenie i weryfikacja tych wymagań jest ważna zarówno dla zlecającego projekt jak i wykonawcy projektu, gdyż doprecyzowują one użyteczność korzystania z systemu, co pozwala na uniknięcie sporów podczas odbioru projektu, w przypadku niskiej jego jakości. Jakość uzyskanego systemu ma również wpływ na jego późniejsze utrzymanie po stronie klienta, stąd tak ważne rozsądne opisanie wymagań pozafunkcjonalnych, które nie zawsze potrafią być oczywiste dla użytkowników końcowych.

4.1 Standard ISO/IEC FDIS 25010

Model jakości oprogramowania standardu ISO/IEC FDIS 25010 [3] wyróżnia następujące charakterystyki i podcharakterystyki jakości oprogramowania:

- Funkcjonalne dopasowanie (ang. Functional suitability)
  - Funkcjonalna kompletnosc (ang. Functional completeness)
  - Funkcjonalna odpowiedniość (ang. Functional correctness)
  - Funkcjonalna poprawność (ang. Functional appropriateness)
- Wydajność (ang. Performance efficiency)
  - Charakterystyka czasowa (ang. Time behaviour)
  - Zużycie zasobów (ang. Resource utilization)
  - Oczekiwana wydajność (ang. Capacity)
- Kompatybilność (ang. Compatibility)
  - Współistnienie (ang. Co-existence)
  - Interoperacyjność (ang. Interoperability)
- Użyteczność (ang. Usability)
  - Rozpoznawalność zastosowania (ang. Appropriateness recognizability)
  - Łatwość nauczenia się (ang. Learnability)

  - Łatwość operowania (ang. Operability)
  - Ochrona użytkownika przed błędami (ang. User error protection)
  - Estetyka interfejsu użytkownika (ang. User interface aesthetics)
  - Dostępność personalna (ang. Accessibility)
- Niezawodność (ang. Reliability)
  - Dojrzałość (ang. Maturity)
  - Tolerancja uszkodzeń (ang. Availability)
  - Odporność na wady (ang. Fault tolerance)
  - Odtwarzalność (ang. Recoverability)
- Bezpieczeństwo (ang. Security)
  - Poufność (ang. Confidentiality)
  - Integralność (ang. Integrity)
  - Niezaprzeczalność (ang. Non-repudiation)
  - Identyfikowalność (ang. Accountability)
  - Autentyczność (ang. Authenticity)
- Łatwość utrzymania (ang. Maintainability)
  - Modułowość (ang. Modularity)
  - Łatwość ponownego wykorzystania (ang. Reusability)
  - Łatwość analizy (ang. Analysability)
  - Łatwość zmiany (ang. Modifiability)
  - Łatwość testowania (ang. Testability)
- Przenośność (ang. Portability)
  - Łatwość adaptacji (ang. Adaptability)
  - Łatwość instalacji (ang. Installability)
  - Łatwość zamiany (ang. Replaceability)

Wymienione podcharakterystyki posłużyły jako kategorie wymagań pozafunkcjonalnych w projekcie. Ze względu na specyfikę systemu - aplikacji internetowej niektóre kategorie tego standardu nie zostały wykorzystane podczas prac nad wymaganiami pozafunkcjonalnymi.

4.2 Wymagania pozafunkcjonalne i ich weryfikacja

W kolejnych tablicach przedstawiono wymagania pozafunkcjonalne systemu LinkedInGrads, określonych przy pomocy standardu ISO/IEC FDIS 25010. Priorytet wymagań określono za pomocą notacji:

- Would - System może spełniać dane wymaganie,
- Should - System powinien spełniać dane wymaganie,

- Must - System musi spełniać dane wymaganie. Dodatkowo określono złożoność każdego z wymagań wykorzystując następującą notację:
- Low - Niski stopień złożoności wymagania pozafunkcjonalnego,
- Medium - Średni stopień złożoności wymagania pozafunkcjonalnego,
- High - Wysoki stopień złożoności wymagania pozafunkcjonalnego. Metryki te pozwalały podczas prac nad projektem na realizację najważniejszych wymagań, uwzględniając ich złożoność czasową skonfrontowaną z dostępnym czasem na realizację projektu. Ostatnia kolumna tabeli określa czy w projekcie LinkedInGrads udało się spełnić dane wymaganie pozafunkcjonalne.

4.2.1 Funkcjonalne dopasowanie: Funkcjonalna kompletność

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Współpraca z przeglądarką Firefox (dla dwóch najnowszych wersji).	Must	Low	Zrealizowano

Tablica 4.1: Funkcjonalne dopasowanie: Funkcjonalna kompletność

System był tworzony wykorzystując do testów najnowszą wersję przeglądarki Mozilla Firefox, oraz dodatkowo Google Chrome. Zapewniło to kompatybilność systemu z tymi przeglądarkami, a funkcje użyte wewnątrz systemu nie wykraczają poza zakres możliwości poprzednich kilku wersji tych przeglądarek.

4.2.2 Wydajność: Charakterystyka Czasowa

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Dla 20000 absolwentów aktualizacja danych z LinkedIn nie powinna trwać dłużej niż tydzień.	Must	Medium	Zrealizowano
System powinien estymować czas aktualizacji danych.	Would	High	Pominęto
Generowanie raportu nie może trwać dłużej niż 10 sekund - w przeciwnym wypadku informacja o czasie.	Should	Medium	Zrealizowno

Tablica 4.2: Wydajność: Charakterystyka Czasowa

Wymagania tej kategorii udało się spełnić, a dzięki aktualizacji danych w tle, estymacja czasu zakończenia aktualizacji okazała się zbędna w finalnej wersji produktu, co również spowodowało niski czas generowania raportu.

4.2.3 Kompatybilność: Współistnienie

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Na jednym serwerze może być uruchomionych wiele instancji aplikacji.	Would	Medium	Zrealizowano

Tablica 4.3: Kompatybilność: Współistnienie

Wymaganie to zostało zrealizowane na poziomie architektury poprzez wykorzystanie aplikacji internetowej uruchamianej w środowisku serwera WWW.

4.2.4 Kompatybilność: Interoperacyjność

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
System ma udostępniać dane absolwenta w trybie odczytu w formacie JSON przez webservice.	Should	High	Pominięto
System ma generować raporty w formacie CSV.	Should	Low	Pominęto
System ma generować raporty w formacie XLS.	Must	Low	Zrealizowano

Tablica 4.4: Kompatybilność: Interoperacyjność

Ze względu na duży narzut czasowy wymagań pozafunkcjonalnych z tej kategorii udało się zrealizować tylko generowanie raportu w formacie XLS. Pominięte wymagania funkcjonalne nie były kluczowe dla systemu, co spowodowało, że ich realizacja nie została przydzielona do żadnego ze sprintów.

4.2.5 Użyteczność: Łatwość nauczenia się

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Interfejs użytkownika powinien być skonsultowany ze wszystkimi potencjalnymi użytkownikami.	Must	Low	Zrealizowano
Instrukcja użytkownika powinna być zorientowana na cele poszczególnych aktorów.	Must	Low	Zrealizowano

Tablica 4.5: Użyteczność: Łatwość nauczenia się

Zespół był wysoce zorientowany na potrzeby użytkownika końcowego, co już w samej fazie projektowania interfejsów użytkownika zapewniło wysoką jakość systemu. Po skonsultowaniu z użytkownikami, wdrożono udoskonalenia co pozwoliło na zrealizowanie wymagania pozafunkcjonalnego.

4.2.6 Użyteczność: Ochrona użytkownika przed błędami

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
System musi być przygotowany na rozszerzenie list atrybutów.	Must	Low	Zrealizowano
System powinien logować błedy (wyjątki) do logu systemowego.	Must	Low	Zrealizowano
System powinien informować o błędnej konfiguracji natychmiast po jej wprowadzeniu.	Would	Medium	Pominięto
System ma potwierdzać powiązanie/odwiązanie użytkownika.	Would	Low	Pominięto
System powinien zapewnić kontrolę nieprzewidzianych wyjątków.	Must	Low	Zrealizowano częściowo.
Po wykonaniu importu danych system informuje ilu absolwentów zaimportowano, a ile jest konfliktów.	Must	Low	Pominięto

Tablica 4.6: Użyteczność: Ochrona użytkownika przed błędami

System został przygotowany mając od początku na uwadze możliwość rozszerzenia go o dodatkowe dane jak i dodatkowe źródła danych, co zapewniło realizację pierwszego z wymagań. Wszystkie błędy które pojawią się w aplikacji automatycznie zapisywane są do logów, co częściowo realizuje również wymaganie dotyczące kontroli nieprzewidzianych wyjątków, które zostają w momencie wystąpienia ukryte przed użytkownikiem końcowym. Podczas prac nad systemem zrezygnowano z możliwości konfiguracji parametrów jego pracy, ze względu na brak zasadności jakichkolwiek zmian. Wymaganie to powstało mając na uwadze możliwe ograniczenia narzucone na system pochodzące z zewnętrznych źródeł danych. Również w trakcie prac zrezygnowano z automatycznego wyszukiwania użytkowników wewnątrz sieci społecznościowej LinkedIn i wiązania znalezionych kont z danymi osób dostarczonymi z dziekanatu. Wyeliminowało to potrzebę potwierdzania powiązania/odwiązania użytkownika, gdyż dane te wprowadzane są manualnie przez użytkownika. Ostatnie z wymagań nie zostało zrealizowane ze względu na dołączenie go do wymagań pozafunkcjonalnych w późnej fazie projektu i nie uwzględnienie jego realizacji w żadnym ze sprintów.

4.2.7 Użyteczność: Estetyka interfejsu użytkownika

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Interfejs powinien zawierać logo Politechniki Poznańskiej oraz opis przeznaczenia systemu.	Must	Low	Zrealizowano

Tablica 4.7: Użyteczność: Estetyka interfejsu użytkownika

4.2.8 Użyteczność: Dostępność personalna

Wymaganie	Priorytet	Złożoność	Realizacja
System dostępny tylko w jednej wersji językowej - Polskiej	Must	Low	Zrealizowano

Tablica 4.8: Użyteczność: Dostępność personalna

4.2.9 Niezawodność: Tolerancja uszkodzeń

Wymaganie	Priorytet	Złożoność	Realizacja
Przerwy serwisowe możliwe są w godzinach 18-6 lub po wcześniejszym ustaleniu.	Should	Low	Pominięto
Niekontrolowana awaria może trwać maksymalnie 2 dni.	Should	Low	Pominięto

Tablica 4.9: Niezawodność: Tolerancja uszkodzeń

Wymagania należące do tej kategorii nie zostały zrealizowane. Wynika to z faktu, że dotyczą one utrzymania systemu po jego wdrożeniu, więc weryfikacja tych wymagań była niemożliwa do przeprowadzenia.

4.2.10 Niezawodność: Odporność na wady

Wymaganie	Priorytet	Złożoność	Realizacja
System należy poddać analizie z wykorzystaniem metody ATAM.	Must	Low	Zrealizowano
Wszystkie funkcje głównego scenariusza powinny być możliwe do przetestowania w sposób automatyczny.	Should	Low	Zrealizowano
W przypadku problemów z połączeniem, proces aktualizacji danych powinien być wznawiany (od stanu w którym nastąpiło przerwanie).	Would	Low	Zrealizowano
System powinien posiadać testy jednostkowe oraz testy akceptacyjne.	Must	High	Zrealizowano
W przypadku problemów z połączeniem proces aktualizacji danych powinien być ponawiany.	Must	Low	Zrealizowano

Tablica 4.10: Niezawodność: Odporność na wady

4.2.11 Niezawodność: Odtwarzalność

Wymaganie	Priorytet	Złożoność	Realizacja
System powinien tworzyć kopie zapasowe danych codziennie.	Must	Low	Zrealizowano

Tablica 4.11: Niezawodność: Odtwarzalność

4.2.12 Bezpieczeństwo: Poufność

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
System zapewnia szyfrowane połączenie z użytkownikiem (nie wymaga zaufanego certyfikatu).	Should	Low	Zrealizowano
System powinien korzystać z systemu eLogin do uwierzytelniania użytkowników.	Must	Medium	Zrealizowano

Tablica 4.12: Bezpieczeństwo: Poufność

4.2.13 Bezpieczeństwo: Integralność

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Dane przechowywane w systemie powinny być chronione przed nieuprawnionym odczytem, modyfikacją, usunięciem.	Must	Low	Zrealizowano
System powinien być odporny na SQL injection.	Must	Low	Zrealizowano

Tablica 4.13: Bezpieczeństwo: Integralność

Wymagania zawarte w tej kategorii zostały zrealizowane na poziomie architektury, poprzez wykorzystanie przygotowanych zapytań (ang. prepared statements), oraz wymaganie uwierzytelnienia użytkownika.

4.2.14 Bezpieczeństwo: Niezaprzeczalność

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Import danych powinien mieć stempel czasowy – w raporcie powinna się znaleźć data aktualizacji danych na podstawie których będzie generowany raport.	Must	Low	Zrealizowano

Tablica 4.14: Bezpieczeństwo: Niezaprzeczalność

4.2.15 Łatwość utrzymania: Łatwość analizy

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Należy zapewnić dokumentację zgodną z wymaganiami DRO.	Must	Medium	Zrealizowano
Kod źródłowy systemu musi przestrzegać konwencji wymaganej przez DRO.	Must	Medium	Zrealizowano

Tablica 4.15: Łatwość utrzymania: Łatwość analizy

4.2.16 Łatwość utrzymania: Łatwość zmiany

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Architektura powinna zapewnić rozdzielenie warstwy backendowej od frontendowej.	Must	Low	Zrealizowano
System musi być przygotowany na zmianę generowania raportów i udostępniania danych.	Must	Low	Zrealizowano
Model danych systemu powinien być przygotowany na zmiany: integrację z nowymi systemami, adaptację na potrzeby innych uczelni.	Must	Low	Zrealizowano

Tablica 4.16: Łatwość utrzymania: Łatwość zmiany

Rozdzielenie warstwy frontendowej od backendowej zostało zrealizowane z pomocą frameworka Ruby on Rails implementującego wzorzec projektowy MVC (Model-view-controller). System zapewnia również możliwość tworzenia nowych raportów, oraz integrację z innymi systemami ze względu na modularną architekturę systemu.

4.2.17 Łatwość utrzymania: Łatwość testowania

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Przewidzenie dwóch trybów - produkcyjny i testowy.	Should	Low	Zrealizowano

Tablica 4.17: Łatwość utrzymania: Łatwość testowania

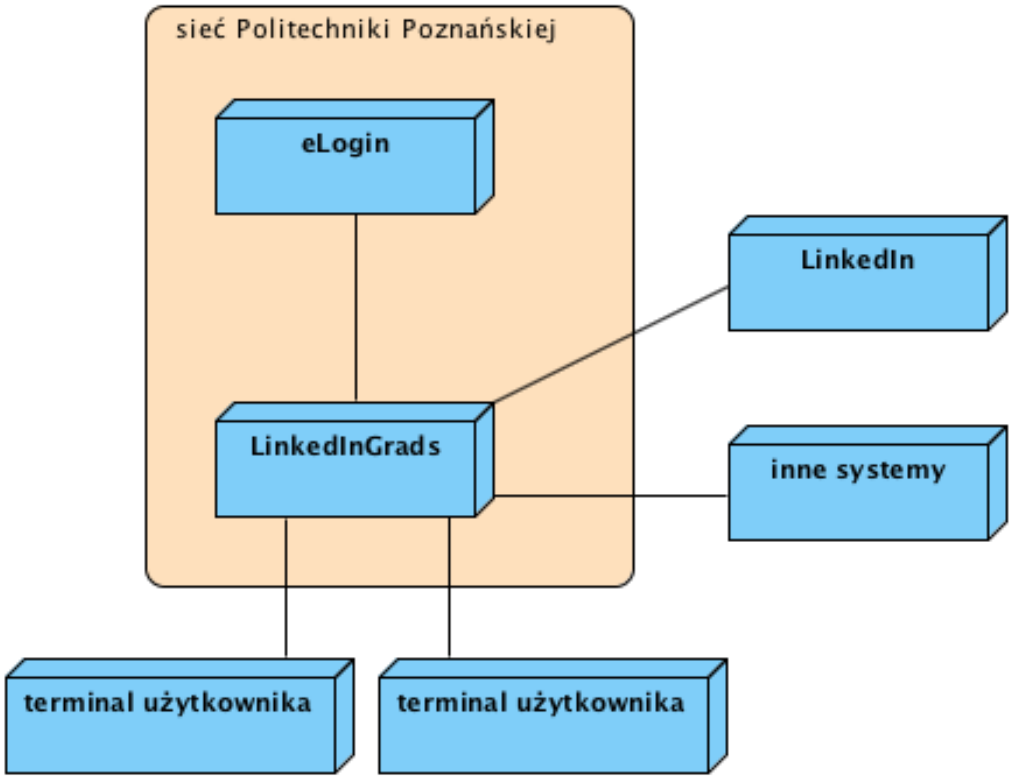
4.2.18 Przenośność: Łatwość instalacji

Wymaganie	**Priorytet**	**Złożoność**	**Realizacja**
Należy dostarczyć szczegółową instrukcję instalacji oraz konfiguracji środowiska wykonawczego.	Must	Medium	Zrealizowano

Tablica 4.18: Przenośność: Łatwość instalacji

Rozdział 5

Architektura systemu



Rysunek 5.1: Architektura systemu LinkedInGrads

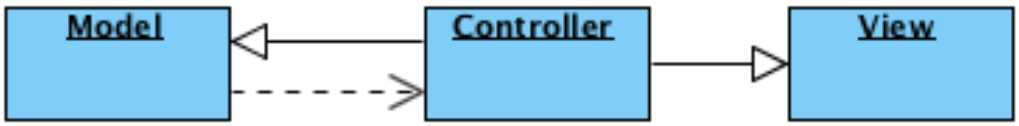
W rozdziale przedstawione zostaną zagadnienia związane z architekturą systemu LinkedInGrads. Ukazane zostaną zastosowane podejścia architektoniczne, podjęte decyzje (razem z ich uzasadnieniami oraz zależnościami), zobrazowany będzie przyjęty schemat bazy danych. W celu lepszego przedstawienia tematu architektury, ze względu na jego złożoność, w podrozdziale 5.4. Perspektywy architektoniczne wykorzystano model 4+1 Views.

5.1 Zastosowane podejście architektoniczne

Rozdział dotyczy wzorca projektowego, w oparciu o który zaprojektowano architekturę systemu.

5.1.1 Architektura MVC

Wzorzec projektowy, który wykorzystano w systemie to MVC (ang. Model View Controller). Pozwala on na organizację aplikacji posiadających interfejs użytkownika dzieląc kod na trzy główne części przedstawione na diagramie poniżej. Dzięki takimi rozwiązaniu można odseparować wewnętrzną reprezentację informacji od sposobu jej prezentacji użytkownikowi końcowemu.



Rysunek 5.2: Zależności warstw w architekturze MVC

Zgodnie z rysunkiem 5.2. występuje podział aplikacji na:

- Model - warstwa reprezentująca logikę biznesową aplikacji, odpowiada również za stan aplikacji oraz utrzymywanie informacji,
- Controller - warstwa odpowiedzialna za komunikację i sterowanie, przyjmuje żądania użytkownika, reagując na nie aktualizacją modeli, prezentuje informacje modeli używając do tego celu widoków,
- View - warstwa prezentująca dane i interfejs użytkownikowi, nie zajmuje się przetwarzaniem informacji.

Powyższy wzorzec posiada wiele zalet. Jedną z nich może być bardzo czytelny podział odpowiedzialności poszczególnych części aplikacji, który znacznie obniża barierę wejścia w razie dołączenia nowego członka do projektu. Główne korzyści płyną jednak z faktu odseparowania modeli, czyli logiki biznesowej od pozostałych warstw: widoków oraz kontrolerów. Wykorzystanie samodzielnych modeli:

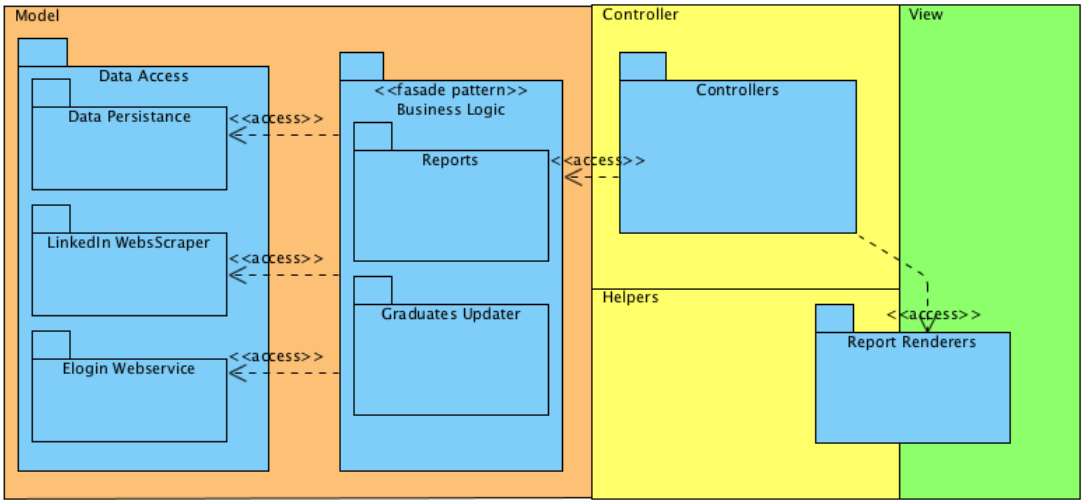
- sprawia, że mogą być użyte ponownie w wielu miejscach, niezależnie od interfejsów, Przykładem może być prezentowanie danych z tego samego modelu w różnych formatach, wykorzystując do tego różne widoki
- upraszcza testowanie kodu logiki biznesowej, która nie jest powiązane z warstwami wymuszającymi np. wytwarzanie żądania http
- ułatwia rozbudowę interfejsów, przy solidnie zbudowanej logice, najczęściej wykonywane zmiany: dodawanie nowych i modyfikowanie istniejących widoków nie wpływa wcale na główną część systemu

- minimalizuje sposobność naruszenia reguł biznesowych, ponieważ wyłącznie model może modyfikować dane

Wadą tego wzorca są kosztowne zmiany interfejsów już istniejących modeli. W przypadku powiązania wielu widoków z danym modelem, należy zadbać o stosowne zmiany w każdym z nich. Dodatkowo, testowanie złożonych widoków, korzystających z wielu modeli może okazać się bardzo trudne, ze względu na liczbę powiązanych różnych obiektów.

5.1.2 Podział warstwy Model oraz ReportRenderes

Na rysunku 5.3 przedstawiono podział warstw wzorca Model View Controller oraz wprowadzone modyfikacje:



Rysunek 5.3: Architektura MVC w systemie LinkedInGrads

Dokonano podziału warstwy odpowiadającej za logikę - model - na dwie dodatkowe warstwy:

- Business Logic - odpowiada za implementację właściwych reguł biznesowych oraz obiektów,
- Data Access - odpowiada za komunikację z bazą danych: odczyt oraz zapis, interakcję z zewnętrznymi źródłami danych, co jest wynikiem komunikacji z klasami warstw Business Logic.

Rozwiązanie takie oprócz zwiększenia czytelności podziału odpowiedzialności powoduje zwiększenie bezpieczeństwa zachowania reguł biznesowych - nic poza warstwą Business Logic nie powinno wykonywać zleceń do warstwy Data Access i tym samym mieć możliwości modyfikowania danych. Odizolowanie logiki od warstwy odpowiadającej za komunikację z bazą danych sprawia, że ewentualna zmiana silnika bazy danych na inny, powinna przebiegać szybko i nie spowoduje olbrzymiego nakładu czasowego.

W obrębie klasy Business Logic zastosowano fasadę, ułatwiającą korzystanie z Raportów. Zgodnie z diagramem utworzono warstwę ReportRenderers, odpowiedzialną za generowanie raportów w oczekiwanych formatach - wykazuje więc cechy warstwy View, dostarcza odpowiednio zaprezentowane dane. ReportRenderes inspirowana była wzorcami Visitor oraz Builder, jest wykorzystywana

przez kontrolery i wizytuje obiekty Report. Rozwiązanie takie pozwala na elastyczne zarządzanie dostępnymi formatami raportów.

5.1.3 Rozbudowa oprogramowania

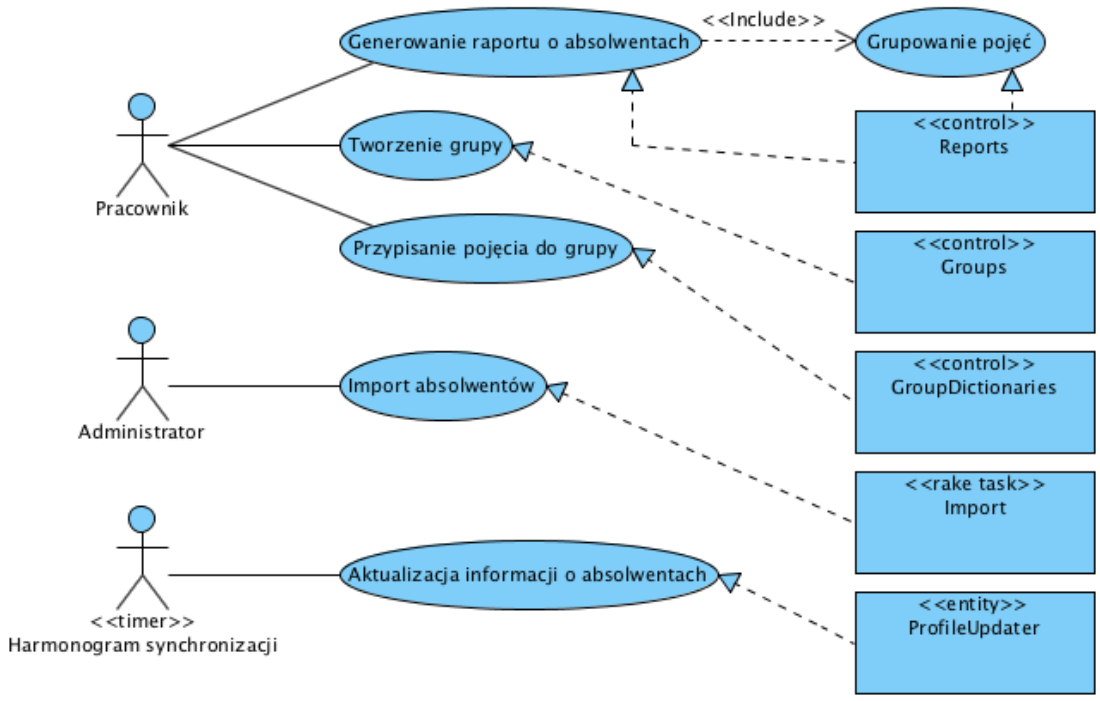
Opisane wcześniej rozwiązania sprawiają, że zwiększa się stopień odizolowania właściwej logiki biznesowej. Pozwala to na używanie warstwy Model, a nawet samej Business Logic przez inne zewnętrzne moduły, które stają się w stosunku do niej klientami. Dodatkowo organizacja generowania raportów oparta o znane wzorce projektowe, sprawia, że rozbudowa o nowe raporty czy też rozszerzenia jest bardzo elastyczna, a ogólna rozbudowa nie będzie kosztowna.

5.2 Perspektywy architektoniczne

Podstawa organizacyjna każdego systemu reprezentowana jest przez elementy strukturalne, zachowanie elementów i kompozycję zarówno jednych jak i drugich. Podzbiory te są dyktowane przez szczególne cechy np. wymagania. Architektura oznacza więc różne zapotrzebowania różnych interesariuszy. Model 4+1, zaprezentowany w kolejnych podrozdziałach wychodzi temu naprzeciw organizując architekturę względem indywidualnych potrzeb grup osób zaangażowanych.

5.2.1 Perspektywa logiczna

Perspektywa logiczna zajmuje się funkcjonalnościami systemu, jakie ma do zaoferowania użytkownikom końcowym. Koncentruje się na operacjach możliwych do zrealizowania i danych.

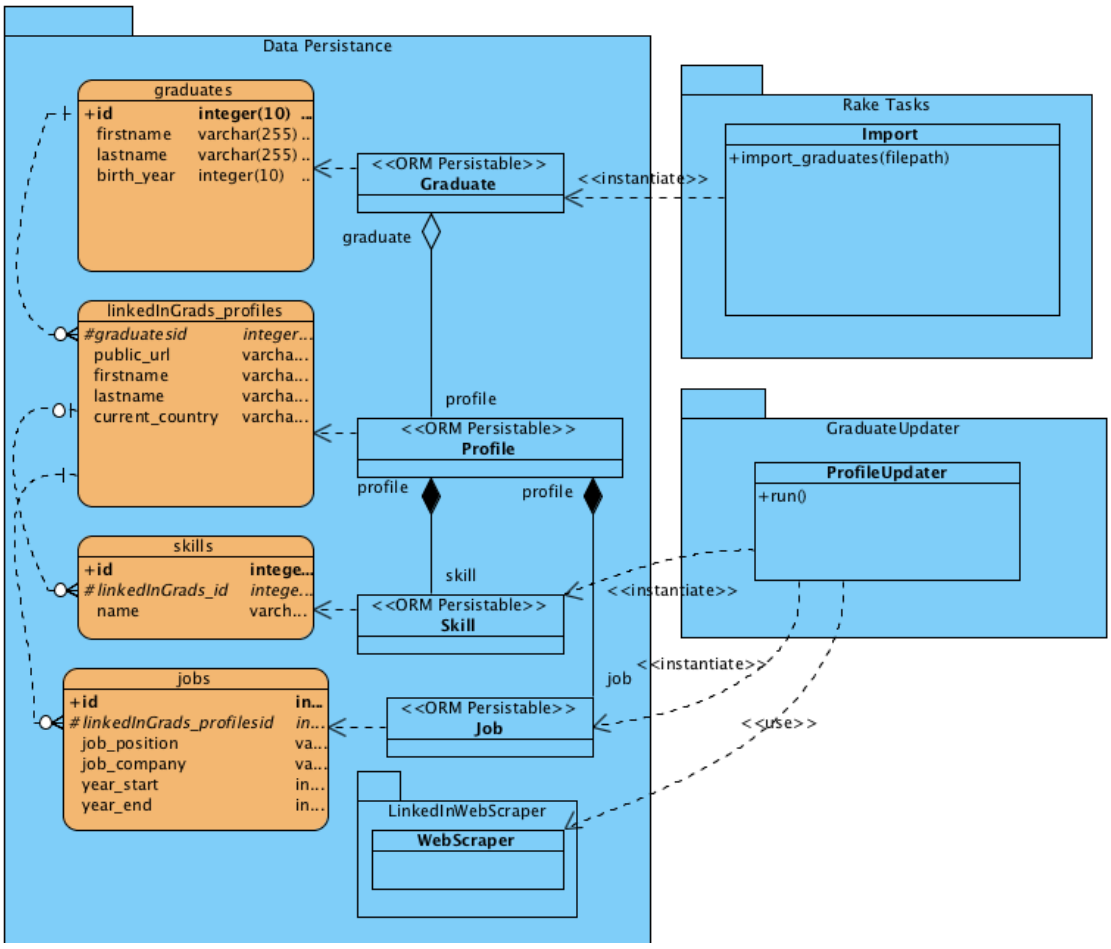


Rysunek 5.4: Perspektywa logiczna

Na rysunku 5.4. zaprezentowano realizację przypadków użycia przez poszczególne pogrupowane odpowiednio jednostki kodu. Z uwagi na to, że system oparty jest o wzorzec MVC większa część modułów realizujących przypadki użycia to kontrolery. Inaczej jest w przypadku "Importu absolwentów", który wykonywany jest przez administratora nie przez interfejs aplikacji internetowej, a także "Aktualizacji informacji o absolwentach" wykonywanego w tle, przez sam system. Dane przedstawione są w rozdziale 5.7. za pomocą schematu związków-encji.

5.2.2 Perspektywa implementacyjna

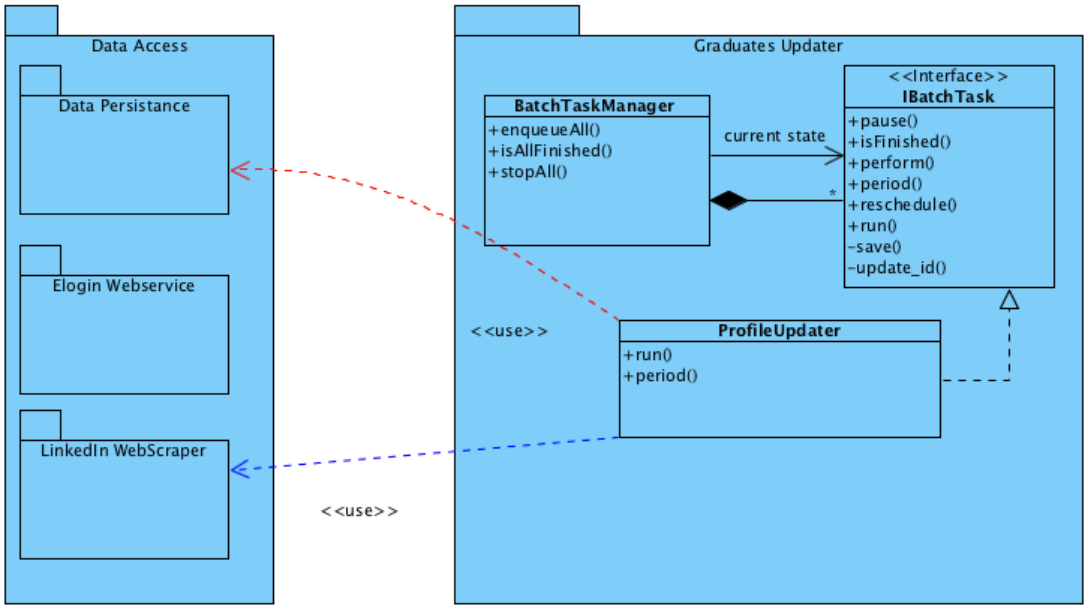
Perspektywa implementacyjna przedstawia komponenty składające się na system, skupiając się na organizacji modułów i zależnościach między nimi. O ile perspektywa logiczna jest na poziomie pojęciowym, ta perspektywa przedstawia rzeczywiste jednostki, odzwierciedlone w kodzie źródłowym. Przydatna może być podczas konfiguracji, rozwijania oraz konserwacji oprogramowania przez programistów.



Rysunek 5.5: Perspektywa implementacyjna

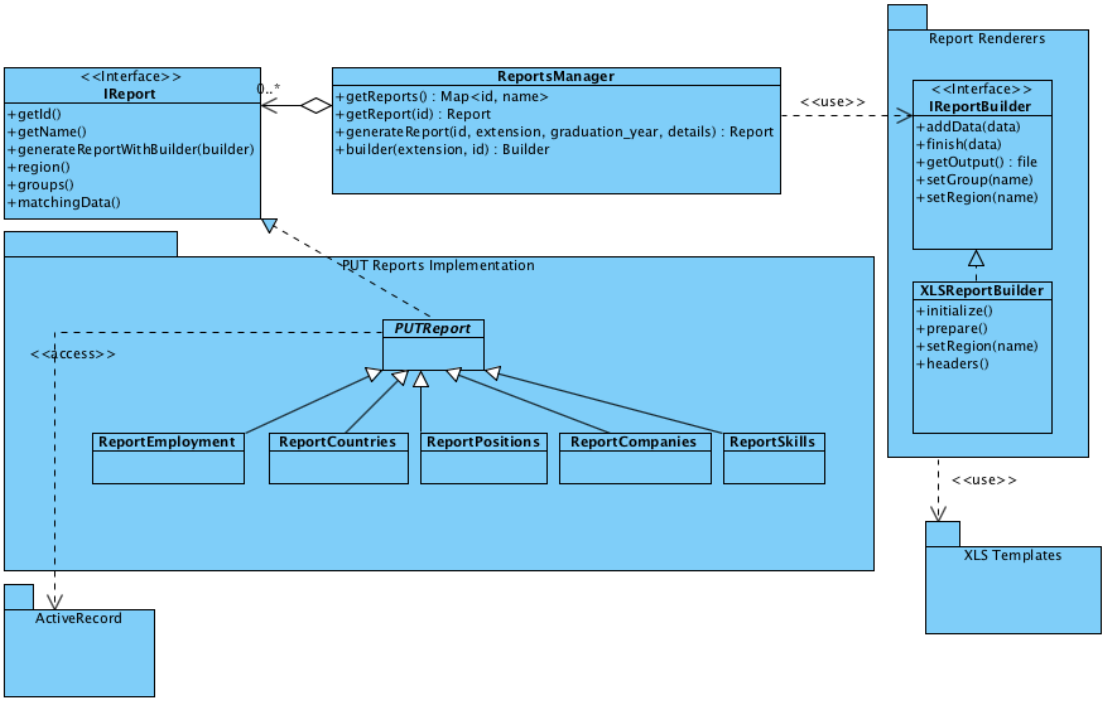
Rysunek 5.5. opisuje możliwe sposoby wypełniania bazy danych danymi. Źródłem danych początkowych jest wykonanie Importu, co powoduje dodanie nowych absolwentów reprezentowanych przez model Graduate. Użycie klasy ProfileUpdater natomiast jest źródłem danych z serwisów zewnętrznych - używa w tym celu LinkedinWebScrapera i inicjalizuje modele Skill oraz Job. Na

diagramie pokazano także podstawowe relacje dotyczące raportów oraz ich mapowanie na odpowiadające klasy ORM.



Rysunek 5.6: Diagram klas - menadżer zadań

Rysunek 5.6. przedstawia realizację menadżera zadań, za którego pomocą istnieje możliwość tworzenia okresowych zadań do wykonania. Zadania te można wstrzymywać oraz wznawiać w dowolnym okresie. Możliwe jest kolejkowanie kilku obiektów klas implementujących interfejs IBatchTask. Procesy mogą być kontrolowane przez menadżera. Wykorzystanie komponentu Batch skupia się wokół aktualizacji danych z serwisów zewnętrznych, które może być długotrwałe, dlatego korzystne jest wykonywanie tego typu zadań w tle.



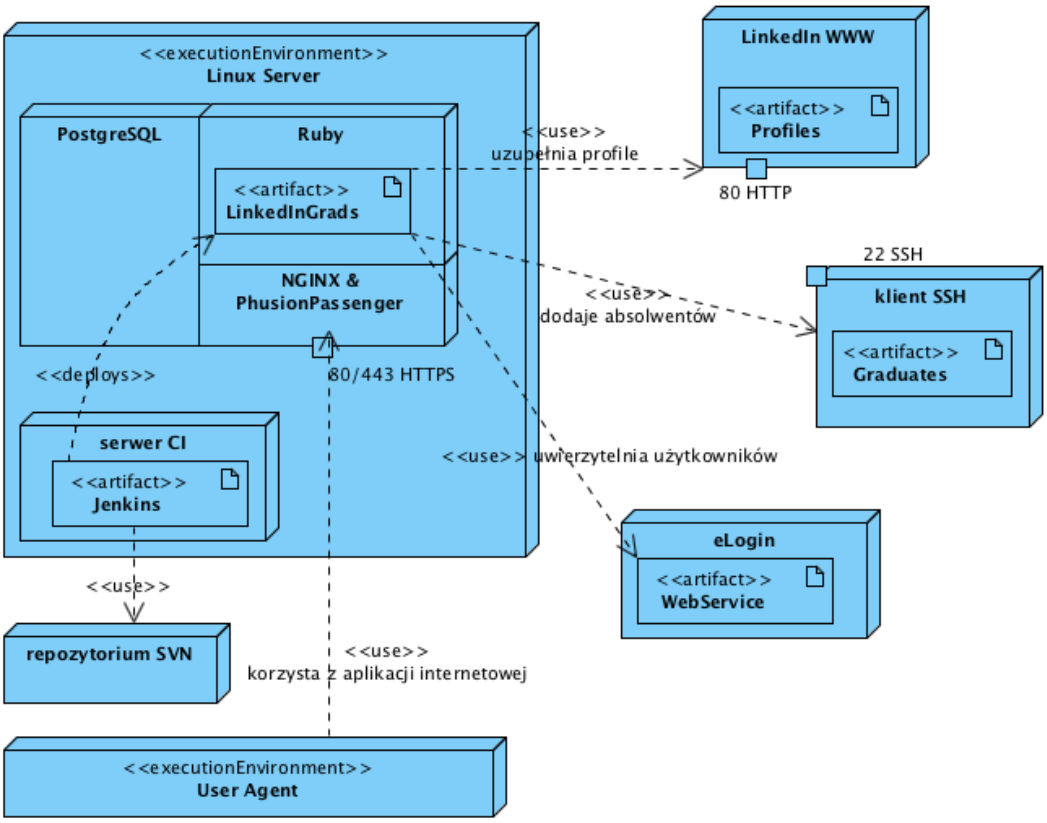
Rysunek 5.7: Diagram klas - generowanie raportów

Rysunek 5.7. obrazuje logikę biznesową związaną z generowaniem raportów. Podejście inspirowane jest wzorcami projektowymi Visitor oraz Builder. Klasy odpowiedzialne za właściwe dostarczenie danych i generację raportu to klasy implementujące interfejs IReport. Generacja przebiega za pomocą wizytującego obiektu IReportBuilder. Klasy implementujące ten interfejs odpowiedzialne są za różne typy formatów wyjściowych.

5.2.3 Perspektywa fizyczna

Perspektywa fizyczna skupia się na topologii sprzętowej systemu informatycznego służącego do uruchomienia aplikacji. Procesy i zadania mapowane są do elementów na których się wykonują. Informacje mogą być przydatne w szczególności dla osób odpowiedzialnych za wdrożenie systemu lub administratorów.

Rysunek 5.8. przedstawia rozmieszczenie fizycznych elementów systemu. Mimo tego, że serwer ciągłej integracji oraz baza danych rozlokowane są na jednej maszynie nie ma takiej konieczności, można je umieścić na osobnych serwerach, jeżeli zaistnieje taka potrzeba. Technologie widoczne na rysunku zostały opisane w rozdziale 5.5.



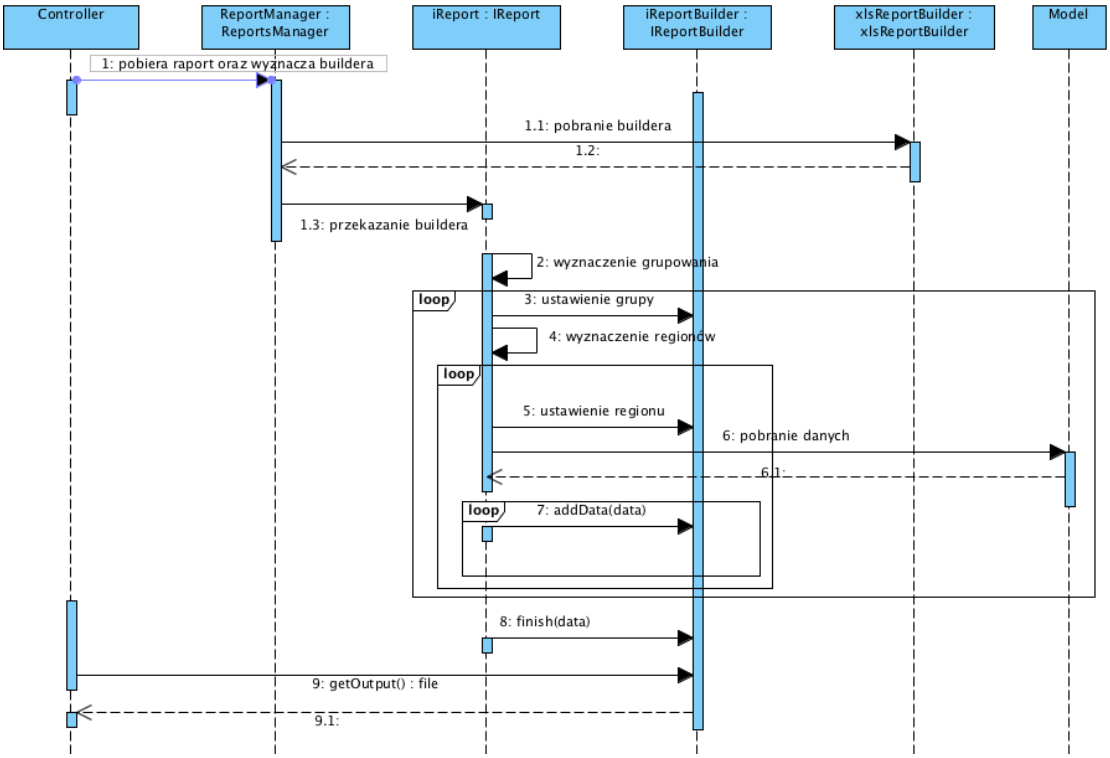
Rysunek 5.8: Perspektywa fizyczna

5.2.4 Perspektywa procesu

Perspektywa procesu zajmuje się dynamicznym aspektem systemu, wyjaśnia komunikację procesów systemu. Adresuje współbieżność, integrację, wydajność i skalowalność.

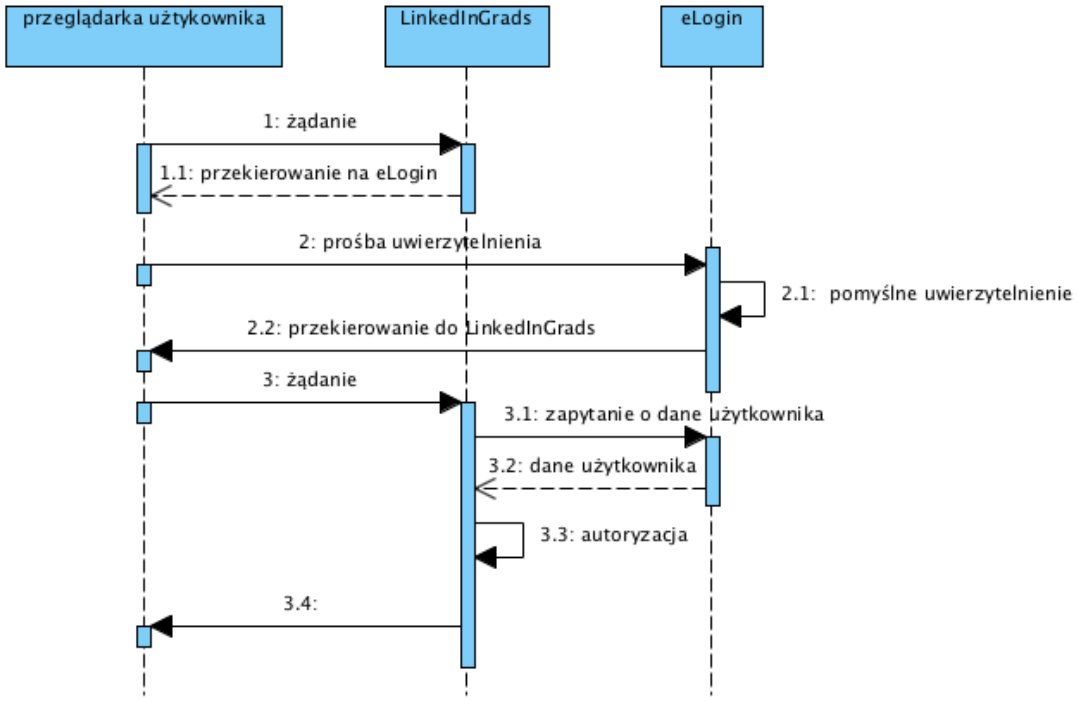
Aplikacja LinkedInGrads będąc aplikacją internetową korzysta z protokołu request-response jakim jest HTTP. Cała komunikacja odbywa się więc przez kolejne wiadomości żądań oraz odpowiedzi, co oznacza, że serwer nie wymaga utrzymywania połączeń z klientami.

Rysunek 5.9. obrazuje dynamikę generowania raportów z użyciem pakietu Reports, o którym była mowa także w rozdziale 5.4.2.



Rysunek 5.9: Perspektywa procesu

Rysunek 5.10. wyjaśnia sposób przeprowadzenia komunikacji dotyczącej logowania do aplikacji, korzystając z zewnętrznego web service'u udostępnionego przez Politechnikę Poznańską. Przy próbie logowania użytkownik zostaje przekierowany na stronę Politechniki Poznańskiej, gdzie ma możliwość użycia danych logowania jednakowych dla wszystkich systemów Politechniki Poznańskiej. Po pomyślnym logowaniu otrzymuje token dostępu, wykorzystywany później do autoryzacji.



Rysunek 5.10: Przepływ sterowania podczas uwierzytelnienia z systemem eLogin

5.3 Decyzje projektowe i wybór technologii

5.3.1 DT1 Wybór platformy serwerowej Ruby on Rails

Uzasadnienie: Platforma Ruby on Rails jest darmowa i dodatkowo posiada duże wsparcie społeczności, co skutkuje olbrzymią liczbą gotowych bibliotek, które można również za darmo wykorzystać. Ze względu na stosowane w niej ‘convention over configuration‘ przyjmuje rozsądne założenia dotyczące początkowej konfiguracji, co powoduje brak konieczności spędzania godzin na konfigurowaniu i przyspiesza rozpoczęcie właściwych prac. Framework ułatwia tworzenie aplikacji wykorzystujących wzorzec Model-View-Controller. Dodatkowo członkowie zespołu posiadają doświadczenie w tej technologii, co jest niewątpliwą zaletą.

5.3.2 DT2 Wybór serwera HTTP Nginx oraz Phusion Passenger

Uzasadnienie: Najpopularniejsze połączenie serwerowe używane przy serwowaniu aplikacji RubyOnRails, testowane przez setki użytkowników. Cechuje się dobrym wsparciem, łatwą i błyskawiczną konfiguracją oraz bardzo dobrą skalowalnością.

5.3.3 DT3 Wybór systemu bazy danych PostgreSQL

Uzasadnienie: System bazy danych wybrany został ze względu na zalecenia Działu Rozwoju Oprogramowania Politechniki Poznańskiej.

5.3.4 DT4 Wybór technologii frontendowej Javascript oraz biblioteka jQuery

Uzasadnienie: Wszechobecna technologia do tworzenia dynamicznych skryptów po stronie klienta, jest prosta w nauce i powszechnie znana przez programistów. Użycie biblioteki jQuery zdecydowanie upraszcza kod Javascript ułatwiając i przyspieszając m.in. manipulacje DOM.

5.3.5 DT5 Wybór technologii backendowej gem pg oraz ActiveRecord

Uzasadnienie: Biblioteka pg jest interfejsem relacyjnego systemu bazy danych PostgreSQL dla języka Ruby. Jest obecnie najchętniej używanym adapterem bazy danych, dlatego posiada bardzo dobre wsparcie jak i dokumentację.

ActiveRecord jest implementacją wzorca object-relational-mapping (ORM) i służy do mapowania relacyjnych baz danych na obiekty w języku Ruby. Minimalizuje to ilość kodu, która pozwala na stworzenie wiernego modelu obiektowego. Tak jak poprzednie rozwiązania, jest jednym z najpopularniejszych i wyznaje zasadę konwencji ponad konfiguracji. Charakteryzuje się m.in.:

- wygodnym sposobem modyfikacji schematu bazy danych przez migracje,
- prostym tworzeniem (a także) rozszerzaniem walidacji oraz automatycznym wywoływaniem metod podczas różnych etapów życia obiektu,
- banalną deklaracją relacji za pomocą zwykłych metod,
- abstrakcją bazy danych poprzez użycie adapterów (np. wykorzystywany przez nas gem pg).

5.3.6 DT6 Wybór technologii backendowej gem Savon

Uzasadnienie: Biblioteka umożliwia połączenie z WebService'ami za pomocą protokołu SOAP. Najpowszechniej używany klient w języku ruby, znacząco upraszcza autoryzację użytkowników w systemie eLogin.

5.3.7 DT7 Wybór technologii frontendowej LESS

Uzasadnienie: Rozszerza kaskadowe arkusze styli o bardzo przydatne funkcje takie jak:

- możliwość używania zmiennych, definiowania funkcji i wykonywania operacji
- tworzenie 'mixins', które pozwalają na wielokrotne proste używanie zgrupowanych właściwości - klas styli - w różnych miejscach
- wygodnej zagnieżdżonej składni, która zmniejsza objętość kodu i zwiększa jego czytelność

Kompilowaniem LESS do CSS zajmuje się framework Rails, pozwala na to prosta konfiguracja gemem less-rails.

5.3.8 DT8 Wybór technologii frontendowej HAML

Uzasadnienie: HAML - HTML abstraction markup language, silnik szablonów dla HTML. HAML wykorzystany został zamiast standardowego silnika szablonów jakim jest ERB. Charakteryzuje się m.in:

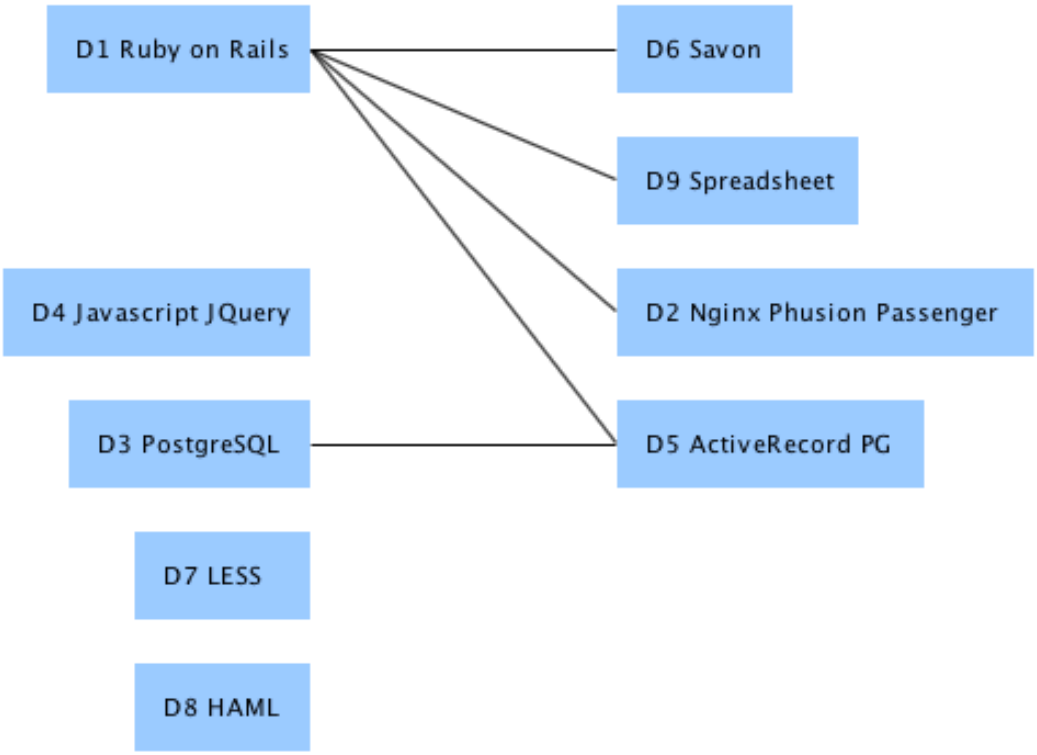
- prostotą, skrótowymi formami tworzenia tagów HTML - np. dokument HAML wykorzystuje wcięcia i rezygnuje z tagów zamykających,
- wygodnym sposobem osadzania fragmentów kodu w języku Ruby w dokumencie HAML. Powyższe powodują, że objętość kodu zmniejsza się drastycznie i dodatkowo zwiększa się jego czytelność ułatwiając tym samym wprowadzanie zmian.

5.3.9 DT9 Wybór technologii frontendowej gem Spreadsheet

Uzasadnienie: Biblioteka służąca do generowania plików formatu '.xls'

5.4 Zależności między decyzjami projektowymi

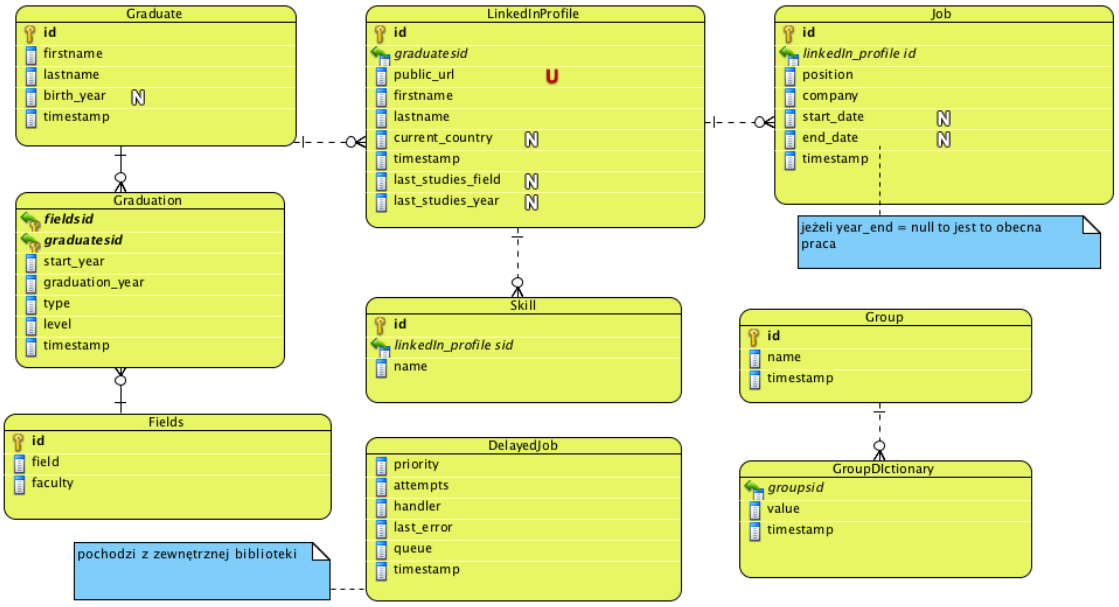
Rysunek 5.11. prezentuje zależności między decyzjami projektowymi. Linie obrazują możliwość podjęcia decyzji.



Rysunek 5.11: Zależności decyzji projektowych

5.5 Schemat bazy danych

Rysunek 5.12. obrazuje relacje bazy danych w systemie LinkedInGrads.



Rysunek 5.12: Schemat bazy danych

Zastosowanie poszczególnych tabel:

- Graduate - tabela przechowująca informacje o absolwentach
- Graduation - tabela przechowująca informacja o ukończonym profilu studiów
- Field - tabela przechowująca metrykę profilu studiów
- Skill - tabela przechowująca informacje o umiejętnościach absolwentów, pobieranych z serwisów zewnętrznych
- LinkedInProfile - tabela przechowująca informacje o profilu absolwenta w serwisie zewnętrznym LinkedIn
- Job - tabel przechowująca informacje o posadach obecnych i historycznych absolwentów, pobieranych z serwisów zewnętrznych
- Group - tabela realizująca obiekt biznesowy Grupa opisany w podrozdziale 2.2.2
- GroupDictionary - tabela
- DelayedJob - tabela tworzona przez zewnętrzną bibliotekę, odpowiadającą za wykonywanie zadań w tle

Rozdział 6

Opis implementacji

W powodzeniu realizacji systemu informatycznego istotny jest wybór narzędzi i technologii, które wspomogą zespół programistów i zwiększą jakość rozwiązania. W niniejszym rozdziale opisane zostaną narzędzia wykorzystane podczas wykonywania pracy inżynierskiej oraz struktura projektu wraz z podstawowymi plikami konfiguracji.

6.1 Narzędzia

6.1.1 RubyMine

Zintegrowane środowisko programistyczne (IDE) dedykowane dla programistów Ruby i Ruby on Rails, podstawowa aplikacja wspierająca pracę programisty. Zespół podjął decyzję o wyborze narzędzia RubyMine. Oprócz podstawowej funkcjonalności jak formatowanie kodu, refaktoryzacja, nawigacja po projekcie i łatwe debuggowanie, zapewniło stabilność oraz integrację z systemem kontroli wersji SVN oraz z systemem zarządzania bazą danych PostgreSQL.

6.1.2 SVN

Scentralizowany system kontroli wersji. Umożliwia śledzenie zmian w projekcie, co ułatwia pracę jednoczesną pracę wielu członków zespołu. Kolejne wersje systemu mogą być oznaczane, więc gdy zajdzie potrzeba można natychmiastowo przywrócić system do poprzednich wersji.

6.1.3 Jenkins

Podczas procesu implementacji projektu wykorzystano serwer ciągłej integracji Jenkins. Został on zintegrowany w ten sposób by wraz z każdą nową wersją w repozytorium uruchamiane były testy. W przypadku pomyślnego zbudowania aplikacji zmiany są automatycznie wprowadzane na serwer produkcyjny.

6.1.4 Nginx oraz Phusion Passenger

Konfiguracja serwerów HTTP użyta do uruchomienia sytemu LinkedInGrads w środowisku produkcyjnym. Takie połączenie ma dwie zalety:

- Bezpieczeństwo - serwery jak Apache i Nginx są bardzo dojrzałe. Użycie ich jako serwery pośredniczące zwiększają bezpieczeństwo uruchomionych na nich aplikacji.
- Wydajność - niektóre serwery lepiej radzą sobie z obsługą treści statycznych.

Jako serwery bazowy wybrano Nginx. Jedyną realną konkurencją dla niego mógłby być Apache, lecz ze względu na łatwiejszą konfigurację oraz bardzo dobrą skalowalność wybrano ten pierwszy. By móc uruchomić na nim aplikację napisaną w języku Ruby trzeba zintegrować go z innym serwerem HTTP. Podjęto decyzję o wyborze Phusion Passenger. Na jego korzyść przemawia duża popularność - korzysta z niego wiele znanych serwisów o bardzo dużym ruchu sieciowym takich jak: salesforce.com czy Symantec. Cechuje się wysoką stabilnością, dobrą wydajnością oraz łatwą konfiguracją. Dodatkowych atutem jest jego interoperacyjność - może również zostać uruchomiony razem z serwerem Apache.

6.1.5 Redmine

Oprogramowanie wspomagające zarządzanie projektem. W przypadku projektu LinkedInGrads pomagał w podziale zadań oraz śledzeniu błędów. Służył również za miejsce wymiany wiedzy o projekcie (wiki).

6.1.6 Ruby on Rails

Framework Open Source służący do szybkiego tworzenia aplikacji internetowych. Wspiera kilka znanych zasad tworzenia oprogramowania:

- wzorzec MVC – dzieli strukturę aplikacji na trzy płaszczyzny: model zawiera logikę biznesową oraz umożliwia utrwalenie danych, kontroler – odpowiada na żądania użytkownika wykorzystując model, widok – określa w jaki sposób wyświetlić odpowiedź dla użytkownika.
- konwencja ponad konfiguracją (ang. Convention Over Configuration) – określa domyślne rozwiązania częstych problemów napotkanych podczas tworzenia aplikacji. Przykładowo we framework'u Ruby on Rails tabela reprezentująca model posiada nazwę taką samą jak on.
- reguła DRY (ang. Don't Repeat Yourself) – Unikanie powielenia kodu; abstrahowanie powtarzających się operacji do metod pomocniczych.

W ramach framework'a Ruby on Rails w systemie LinkedInGrads wykorzystano następujące komponenty:

- Active Record – domyślna biblioteka służąca do mapowania obiektowo-relacyjnego. Udostępnia generator modeli, który znacznie ułatwia tworzenie modeli i ich odwzorowania w relacyjnym schemacie bazy danych. Dodatkowo udostępnia język zapytań do bazy danych, dzięki temu aplikacja nie jest powiązana z konkretnym systemem zarządzania bazą danych.
- ActionPack – odpowiada za tworzenie widoków i kontrolerów.
- ActionSupport – rozszerza możliwości biblioteki standardowej języka Ruby o częste operacje wykorzystywane podczas implementacji aplikacji internetowych.

6.2 Struktura projektu

W poniższej sekcji opisana zostanie struktura systemu LinkedInGrads, która nie odbiega znacząco od domyślnej struktury proponowanej przez framework Ruby on Rails.

- app – zawiera podstawowe komponenty aplikacji: widoki, kontrolery, modele oraz klasy pomocnicze (ang. helpers),

- bin – zawiera skrypty odpowiedzialne za wdrażanie i uruchamianie aplikacji,
- config – zawiera pliki konfiguracyjne aplikacji,
- db – zawiera pliki odpowiedzialne za tworzenie i zarządzanie zmianami w schemacie bazy danych,
- docs – zawiera dokumentację projektu,
- lib – zawiera wykorzystywane biblioteki,
- log – zawiera logi,
- public – zawiera pliki statyczne przesyłane do klienta,
- spec – zawiera testy oraz związane z testami pliki pomocnicze,
- tmp – zawiera pliki tymczasowe,
- vendor – zawiera biblioteki zewnętrznych dostawców.

6.3 Pliki konfiguracji

W katalogu config znajdują się następujące pliki konfiguracji: applicaton.rb, boot.rb i environment.rb zawierają one ustawienia aplikacji wspólne dla jej wszystkich środowisk (test, development, production), database.yml zawiera dane potrzebne do połączenia z serwerem bazy danych. Dla każdego środowiska może być ustawiona inny serwer bazy danych, routes.rb łączy akcje kontrolerów z adresami URL. Oprócz tego w katalogu znajdują się dwa podkatalogi: initializers i environemnts. Pierwszy zawiera skrypty inicjalizujące poszczególne części aplikacji. Drugi zawiera konfigurację dla poszczególnych środowisk.

Rozdział 7

Zapewnianie jakości i konserwacja systemu

Podstawą zapewniania jakości każdego systemu informatycznego jest wprowadzenie i pielęgnacja mechanizmów umożliwiających jego systematyczną weryfikację i walidację. Obecność takich mechanizmów w projekcie pozwala na możliwie wczesne wykrywanie i korygowanie błędów związanych z niezgodności systemu ze specyfikacją wymagań bądź z potrzebami i oczekiwaniami klienta. W systemie LinkedIGrads głównymi mechanizmami zapewniania jakości było stworzenie zestawu testów automatycznych i manualnych, wykorzystanie systemu do ciągłej integracji, przestrzeganie przyjętych standardów kodowania oraz realizacja projektu zgodnie z modelem przyrostowym.

7.1 Testy i weryfikacja jakości oprogramowania

7.1.1 Środowisko testowe

Do napisania testów automatycznych systemu wykorzystano następujące narzędzia:

- RSpec Rails - framework do tworzenia i wykonywania testów aplikacji Ruby on Rails, w myśli idei programowania sterowanego zachowaniami (ang. Behaviour Driven Development),
- Shoulda matchers - biblioteka udostępniana w postaci gemu, rozszerzająca zestaw dostępnych asercji i funkcji pomocniczych,
- FactoryGirl Rails - biblioteka udostępniana w postaci gemu, pozwalająca na znaczne uproszczenie procesu tworzenia i zapisywania w bazie danych obiektów testowych,
- Faker - biblioteka udostępniana w postaci gemu, służąca do automatycznego generowania wartości pól tworzonych obiektów testowych,
- niezależna testowa baza danych PostgreSQL.

7.1.2 Testy jednostkowe

Testy jednostkowe pozwalają na weryfikację poprawności działania pojedynczych metod określonej klasy, a zatem umożliwiają precyzyjną lokalizację przyczyny wystąpienia ewentualnych błędów wykonania. W systemie LinkedInGrads stanowią podstawowy rodzaj testów i są wykorzystywane głównie do sprawdzania powiązań i reguł walidacji modeli oraz publicznych akcji kontrolerów. Wprowadzenie do projektu testów jednostkowych pozwoliło na wykrywanie drobnych błędów w możliwie krótkim czasie od ich pojawienia się w systemie.

7.1.3 Testy integracyjne

Testy integracyjne pozwalają na weryfikację poprawności współdziałania kilku jednostek systemu, takich jak klasy, moduły czy zasoby. Z uwagi na silne zorientowanie tworzonego systemu na dane, stanowią dla niego bardzo istotny rodzaj testów i są wykorzystywane głównie do sprawdzania wyników złożonych zapytań do bazy danych. Wprowadzenie do projektu testów integracyjnych pozwoliło na upewnienie się, że generowane przez system raporty o absolwentach zawierają właściwe informacje.

7.1.4 Testy akceptacyjne

Testy akceptacyjne pozwalają na walidację systemu, a zatem potwierdzenie jego zgodności z oczekiwaniami klienta. Z uwagi na prostotę interfejsu systemu LinkedInGrads oraz niewielką liczbę kroków głównego scenariusza, a także istotny koszt czasowy konfiguracji odpowiednich narzędzi, zrezygnowano z wprowadzania do projektu automatycznych testów tego rodzaju na rzecz systematycznie przeprowadzanych testów manualnych. W dodatku D przedstawiono scenariusze manualnych testów akceptacyjnych, opracowane na podstawie przypadkow uzycia przedstawionych w rozdziale 3.

7.1.5 Inne metody zapewniania jakości

7.1.6 Metodyka pracy i model przyrostowy

W trakcie realizacji projektu zespół programistyczny pracował zgodnie ze zwinną metodyką Scrum. Metodyka ta opiera się na podejściu iteracyjnym, tj. wprowadzeniu podziału procesu rozwijania produktu na krótkie iteracje (tzw. sprinty, przebiegi bądź przyrosty). Po każdej iteracji zespół dostarczał klientowi działającą wersję produktu, wzbogaconą w stosunku do poprzedniej o pewien zakres funkcji niosących wartość biznesową. Podczas spotkania będącego przeglądem sprintu (ang. Sprint Review) klient miał możliwość zapoznania się z uaktualnionym systemem i przekazania zespołowi informacji zwrotnej na jego temat. Prowadzenie prac implementacyjnych zgodnie z modelem przyrostowym oraz utrzymywanie stałego kontaktu z klientem umożliwiło systematyczną walidację systemu i stanowiło istotną metodę zapewniania jakości rozumianej jako zgodność z oczekiwaniami klienta.

7.1.7 Standardy kodowania

Standardy kodowania stanowią zestaw zaleceń dotyczących konwencji nazewniczych i reguł formatowania kodu źródłowego. Dzięki stosowaniu standardów systemy współtworzone przez wielu programistów zachowują czytelność i jednolitość. Służy to przede wszystkim przyszłym administratorom systemu i osobom, które zamierzają go rozwijać. W trakcie prac nad systemem LinkedInGrads zastosowano standardy kodowania zdefiniowane w The Ruby Style Guide [5]. Weryfikacji zgodności z przyjętymi standardami dokonano za pomocą narzędzia do statycznej analizy kodu źródłowego RuboCop.

7.1.8 Ciągła integracja

Ciągła integracja (ang. continuous integration) jest uznaną praktyką programistyczną, zakładającą regularne integrowanie zmian wprowadzanych do kodu programu przez różnych programistów. W celu realizacji tego założenia, w ramach prac nad systemem LinkedInGrads, na serwerze dewe-

loperskim został zainstalowany serwer ciągłej integracji Jenkins, który po każdej wykrytej zmianie w systemie kontroli wersji samoczynnie wywoływał następujące akcje:

- pobranie najnowszej wersji kodu z repozytorium SVN,
- pobranie bibliotek potrzebnych do prawidłowego działania systemu,
- aktualizacja schematu produkcyjnej i testowej bazy danych,
- scalenie i minifikacja statycznych plików CSS i JS,
- wykonanie automatycznych testów jednostkowych i integracyjnych,
- wykonanie automatycznej analizy kodu pod kątem zgodności z przyjętymi standardami,
- zatrzymanie i ponowne uruchomienie serwera aplikacji. Wykorzystanie tak skonfigurowanego narzędzia do ciągłej integracji zagwarantowało regularne i automatyczne przeprowadzanie testów oraz umożliwiło monitorowanie bieżącego stanu systemu. Istotnemu zmniejszeniu uległ również koszt czasowy wdrażania kolejnych wersji systemu na serwer produkcyjny.

Rozdział 8

Zebrane doświadczenia

Realizacja systemu LinkedInGrads pozwoliła członkom zespołu programistycznego na zdobycie wartościowego doświadczenia zawodowego w pracy na rzecz rzeczywistego klienta. Poniżej przedstawione zostały wnioski zebrane w trakcie prowadzenia prac nad projektem:

- W początkowej fazie projektu bardzo ważne jest zdefiniowanie głównego użytkownika i klienta oraz nadrzędnego celu działania systemu. Pierwotnie rolę klienta miało pełnić Centrum Praktyk i Karier Politechniki Poznańskiej, późniejsza zmiana pociągnęła za sobą konieczność ponownego zebrania wymagań i aktualizacji istotnej części dokumentacji projektu.
- Dla powodzenia całości projektu kluczowy jest przemyślany dobór narzędzi. Pożądane jest, aby programiści mieli solidne podstawy teoretyczne, a najlepiej praktyczne doświadczenie w pracy z wybranymi technologiami. Bardzo dobra znajomość języka Ruby przez część zespołu programistycznego oraz stosunkowo niski próg wejścia zapewniły sprawny przebieg prac programistycznych.
- Realizacja projektu zgodnie z metodyką Scrum i związane z nią podejście iteracyjne zostały dobrze przyjęte przez zespół programistyczny. Szczególnie istotnymi praktykami okazały się:
  - przeglądy sprintu (ang. Sprint Review), pozwalające na systematyczną walidację systemu i dostosowywanie go do oczekiwań klienta,
  - retrospektywy (ang. Sprint Retrospective), pozwalające na wyciągnięcie wniosków z dotychczasowych doświadczeń i ciągłe ulepszanie metod pracy,
  - codzienne spotkania (ang. Daily Scrum), pozwalające na zidentyfikowanie bieżących wyzwań i problemów oraz wspierające przepływ informacji w zespole.
- Rozmowy i spotkania związane z realizacją projektu zazwyczaj trwały dłużej niż planowano. Warto przed każdym spotkaniem narzucić zespołowi górne ograniczenie czasu i egzekwować jego przestrzeganie. Dyscypliną i stanowczością w tym zakresie powinien się wykazać prowadzący spotkanie "Mistrz Młyna"(ang. Scrum Master).
- Korzystanie z narzędzi do ciągłej integracji umożliwia automatyzację wykonywania testów i znacznie skraca czas wdrożenia. Wymienione korzyści rosną wraz z rozmiarem systemu i czasem trwania projektu. W przypadku systemu LinkedInGrads narzędzia do ciągłej integracji nie zostały niestety skonfigurowane na początku projektu. W konsekwencji jeden ze sprintów został zakończony z opóźnieniem spowodowanym problemami z wdrożeniem systemu na serwer. Po wprowadzeniu systemu Jenkins problemy już się nie pojawiły.

- Tworzenie i utrzymywanie obszernej dokumentacji projektowej jest utrudnione w projektach, w których dynamicznie zmieniają się wymagania. Prowadzi to do sytuacji, w których dokumenty prezentują stan niezgodny z rzeczywistością, a ich aktualizacja nie jest przeprowadzania z uwagi na zbyt duże obciążenie czasowe i niski priorytet. W przypadku pracy w niewielkim, lokalnym zespole nad ograniczonym czasowo projektem słuszne okazało się założenie metodyk zwinnych o wyższości działającego oprogramowanie nad formalną dokumentacją.

Rozdział 9

Zakończenie

9.1 Podsumowanie

Podsumowanie powstałego systemu, czy przedsięwzięcie się udało, czy to się nadaje do czegokolwiek. Taki ładny epilog na koniec pracy inżynierskiej.

9.2 Propozycja dalszych prac

Być może system wymaga jakichś prac w przyszłości lub są jakieś propozycje rozszerzenia funkcjonalności. Dotyczy to zarówno tego, co być może sami będziecie dalej robić (jeśli Wam oczywiście zapłacą) lub mają po Was przejąć inne osoby (z następnymi rocznikami włącznie).

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	Tytuł Imię Nazwisko
Główny użytkownik	Tytuł Imię Nazwisko
Główny dostawca	Tytuł Imię Nazwisko
Dostawca od strony DRO	Tytuł Imię Nazwisko
Starszy konsultant	Tytuł Imię Nazwisko
Konsultant	Tytuł Imię Nazwisko
Kierownik projektu	inż. Imię Nazwisko
Analityk/Architekt	inż. Imię Nazwisko
Programiści	Imię Nazwisko Imię Nazwisko Imię Nazwisko Imię Nazwisko

Tablica A.1: Osoby związane z przedsięwzięciem

Teraz bardzo ważna rzecz – w tym miejscu piszecie, co kto przygotowywał w tekście pracy inżynierskiej. Prawdopodobnie tutaj będziecie musieli wymienić, które rozdziały został dla Was przygotowane przez kierownika projektu, analityka, architekta lub inną osobę. Oczywiście, piszecie też, za które części dokumentu Wy jesteście odpowiedzialni. To jest ważna, aby ta część była tutaj precyzyjnie przygotowana – takie są wymogi uczelni oraz też uwzględnienia pracy innych osób.

Odpowiedzialność za część implementacyjną systemu została przedstawiona poniżej:

Imię i nazwisko pierwszego programisty

- Odpowiedzialność 1
- Odpowiedzialność 2
- Odpowiedzialność 3
- ...

Imię i nazwisko drugiego programisty

- Odpowiedzialność 1

- Odpowiedzialność 2
- Odpowiedzialność 3
- ...

Imię i nazwisko trzeciego programisty

- Odpowiedzialność 1
- Odpowiedzialność 2
- Odpowiedzialność 3
- ...

Imię i nazwisko czwartego programisty

- Odpowiedzialność 1
- Odpowiedzialność 2
- Odpowiedzialność 3
- ...

Ewentualne podziękowania dla innych osób, które Wam pomagały, mowy dziękczynne, itd.

A.2 Wykaz użytych narzędzi

Wprawdzie jest odpowiedni podrozdział w rozdziale 5, ale tutaj można wymienić nawet małe narzędzia i biblioteki, które wykorzystywaliście (np. narzędzie do robienia makiet interfejsu) i które warto wymienić, także dla przyszłych roczników (można też dać linki).

A.3 Zawartość płyty CD

Do dokumentu załączono płytę CD o następującej zawartości:

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

Dodatek B

Wygląd aplikacji

Dodatek opcjonalny, tutaj można zamieścić jakieś zrzuty ekranu czy inne materiały dotyczące interfejsu. Jeżeli nie chcecie tego dodatku, zakomentujecie załączenie tego pliku w `thesis-bachelor-polski.tex`.

Dodatek C

Schemat bazy danych

Dodatek opcjonalny, tutaj można zamieścić schematy bazy danych, jeśli nie zmieścił się w rozdziale o architekturze. Może być też tak, że będziecie mieli legen-czekaj-darny schemat o formacie A3 i będziecie go osobno drukować i wklejać w tym miejscu. Jeżeli nie chcecie tego dodatku, zakomentujecie załączenie tego pliku w `thesis-bachelor-polski.tex`.

Literatura

Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@mastersthesis{ key,
    author = "Adam Butkiewicz \and Dominika Stempniewicz \and Dawid Wengrzik \and Joanna
Zakrzewska",
    title = "{LinkedInGrads:  Otwarty System Monitorowania Karier Zawodowych Absolwentów}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2014",
}
```