

Politechnika Łódzka  
Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki  
Instytut Informatyki Stosowanej

## **PRACA DYPLOMOWA INŻYNIERSKA**

System do wzajemnej oceny kody programistycznego

A system for peer review of application source code

Autor: Adrian Pędziwiatr  
Number albumu: 180254

Opiekun pracy:  
dr inż. Radosław Adamus

Łódź, styczeń 2016



## Streszczenie

W ramach niniejszej pracy inżynierskiej autor opracował system do wzajemnej oceny kodu programistycznego. System ten nie jest kolejnym typowym systemem wspomagającym słownikowo rozumiany przegląd koleżeński. Zrealizowany system ma na celu dostosowanie wspomnianego przeglądu do realiów dydaktycznych.

Realizując specyficzne wymagania takie jak anonimowość przeglądu, czy nadzór osoby prowadzącej kurs (i powierzenie jej roli zlecniodawcy), wspomagany przegląd oddalił się od definicji tytułowego przeglądu wzajemnego. Autor jest głęboko przekonany, iż realizacja przeglądów prac podczas nauki, nie tylko przez prowadzącego kurs, ale także przez kolegów pozytywnie wpłynie na skuteczność usuwania defektów i postęp nauki. System ten ma to umożliwić.

System został zrealizowany jako aplikacja webowa. Jego budowanie przyświecała idea wielokrotnego użycia. W tym celu system głęboko integruje się z powszechnie wykorzystywanym systemem kontroli wersji Git, oraz usługą GitHub. Prowadzący kurs nie tylko nie musi rezygnować z wykorzystywanych już narzędzi, ale także bezproblemowo może poszerzyć warsztat narzędzi dydaktycznych.

**Słowa kluczowe:** *przegląd koleżeński, wielokrotne użycie*

## Abstract

Within this thesis the author has developed and implemented a system for peer review of application source code. This is not another system that just supporting peer review as understood in the dictionary. System has been adapted to the realities of learning process in schools.

By carrying out specific requirements such as „review must be anonymous” or „review is supervised by the person leading the course”, assisted review move away from the definition. The author is deeply convinced that performing peer review while learning, not only by course leader, but also by colleagues positively affect the efficiency of removal of defects and will speed up learning curve. This system is to create opportunities.

The system has been implemented as a Web application. Software reusable is very important in that system. It deeply integrates with one of commonly used version control system - Git and a GitHub service. Course leader can easily start using this system and does not have to give up already using tools.

**Keywords:** *peer review, reusability*



# Spis treści

<b>Streszczenie</b> . . . . .	3
<b>Abstract</b> . . . . .	3
<b>1. Wstęp</b> . . . . .	7
1.1. Cel pracy . . . . .	7
1.2. Rodzaje przeglądów . . . . .	7
1.3. Cel biznesowy . . . . .	8
<b>2. Wymagania</b> . . . . .	9
2.1. Cel systemu . . . . .	9
2.2. Aktorzy systemu . . . . .	9
2.3. Cechy systemu . . . . .	9
2.3.1. Zleceniodawca przeglądu . . . . .	10
2.3.2. Ocena całości . . . . .	10
2.3.3. Sposób oceny . . . . .	10
2.3.4. Kryteria oceny . . . . .	11
2.3.5. Anonimowość przeglądanej pracy . . . . .	11
<b>3. Potencjał do badań naukowych</b> . . . . .	12
<b>4. Sposób realizacji wymagań</b> . . . . .	13
4.1. Źródło ocenianych prac . . . . .	13
4.2. Sposób autoryzacji użytkownika . . . . .	13
4.3. Anonimowość przeglądanej pracy . . . . .	14
<b>5. Wykorzystane technologie</b> . . . . .	15
5.1. Spis użytych technologii . . . . .	15
5.2. Wielokrotne użycie . . . . .	19
5.2.1. Git i GitHub . . . . .	19
5.2.2. Sposób uwierzytelnienia - OAuth2 . . . . .	19
5.2.3. System formularzy . . . . .	21
<b>6. Architektura systemu</b> . . . . .	22
<b>7. Konfiguracja systemu</b> . . . . .	25
7.1. Plik konfiguracyjny . . . . .	25
7.2. Rejestracja aplikacji w usłudze GitHub . . . . .	27

<b>8. Projekt i implementacja systemu</b>	28
8.1. Logowanie do systemu	29
8.2. Kursy i ich uczestnicy	32
8.3. Prowadzący umieszcza zadanie na platformie GitHub	34
8.4. Kursant powiela repozytorium z zadaniem	35
8.5. Prowadzący zleca przegląd	36
8.5.1. Formularz oceny	36
8.5.2. Zlecenie przeglądu	39
8.5.3. Algorytm anonimizacji prac	43
8.6. Kursant dokonuje oceny	47
8.7. Prowadzący sprawdza odpowiedzi	48
8.8. Statystyki systemu i sprzątanie	49
<b>9. Baza danych</b>	50
9.1. Wstęp	50
9.2. Schemat bazy danych	50
9.2.1. Kursy	52
9.2.2. Formularze	52
9.2.3. Przeglądy	52
9.2.4. Odpowiedzi	53
<b>10. Napotkane problemy w implementacji</b>	54
10.1. Wyjątki związane z połączeniem do GitHub API	54
10.2. Pamięć podręczna dla połączeń z GitHub API	55
10.3. Relacja dziedziczenia po stronie bazy danych	57
<b>11. Możliwe kierunki rozwoju systemu</b>	59
11.1. Internacjonalizacja	59
11.2. Wielu prowadzących	59
11.3. Ręczny przydział przeglądów	60
11.4. Panel konfiguracyjny	60
11.5. Obsługa repozytoriów prywatnych	60
11.6. Zmniejszenie ilości wykonywanych kopii	61
<b>12. Podsumowanie</b>	62
<b>Spis rysunków</b>	63
<b>Bibliografia</b>	64

# 1. Wstęp

## 1.1. Cel pracy

Pomysłem przewodnim tej pracy inżynierskiej jest: studenci piszą programy - studenci je oceniają. Celem pracy jest opracowanie systemu, który umożliwia anonimowe przeglądanie kodu, przez kogoś równorzędnego do autora oraz dokonywanie oceny kodu na podstawie wytycznych zdefiniowanych przez zlecającego przegląd. Zakres pracy obejmuje projekt i implementację systemu, oraz opisanie rezultatów pracy.

## 1.2. Rodzaje przeglądów

Przegląd kodu (ang. review) można podzielić na dwie zasadnicze kategorie:

- przegląd formalny (ang. formal review) - wysoce sformalizowany przegląd, prowadzony i dokumentowany zgodnie z ustaloną procedurą;
- przegląd nieformalny (ang. informal review, ad-hoc review) - przegląd, którym nie kierują żadne formalne procedury.

Przeglądów nieformalnych tak na prawdę dokonujemy często - nie zastanawiając się na tym, a nawet no tym nie wiedząc. Najprostszym przykładem przeglądu nieformalnego jest poproszenie kolegi o pomoc w rozwiązaniu problemu. Jednym z typów przeglądu nieformalnego jest przegląd koleżeński. Adam Roman w swojej książce pojęcie to definiuje następująco:

„przegląd koleżeński (ang. peer review) - przegląd produktów powstałych podczas wytwarzania oprogramowania prowadzony przez kolegów ich twórcy mający na celu wskazanie defektów i możliwości poprawek” [14]

Zwyczajowo przeglądy wykonuje się w pracy zawodowej, starając się zapewnić jak najwyższą jakość tworzonego oprogramowania. Działający produkt to nie wszystko. Każde oprogramowanie dostarczające rozwiązanie niebanalnego problemu posiada błędy. Ważne jest, aby te błędy wykrywać jak najszybciej. Im szybciej wykryta i naprawiona zostanie usterka, tym mniejszy koszt wygeneruje. Przeglądy są jednym z typów testowania statycznego, których celem jest odnajdywanie defektów na jak najwcześniejszym etapie. Mediana skuteczności usuwania defektów przy wykonywaniu przeglądu wynosi

około 60% dla przeglądów formalnych, i około 35% dla przeglądów nieformalnych [14]. Statystyki te zachęcają do przyjrzenia się bliżej przeglądom i wykorzystaniu ich potencjału nie tylko w czasie pracy zawodowej.

### **1.3. Cel biznesowy**

Przegląd koleżeński jest tym typem przeglądu na którym skupiono się w tej pracy. Motywacją, która przyświeca tej pracy jest uwzględnienie takiej czynności w procesie nauki. Nie da się nauczyć tworzyć dobre oprogramowanie kończąc na realizacji algorytmu. Refaktoryzacja działającego kodu jest ważnym etapem. O wiele prościej jest ją wykonać, gdy ktoś da nam wskazówki, przedstawi swoje uwagi. Te natomiast łatwo zdobyć wykonując przegląd koleżeński.

System zaprojektowano z myślą o zajęciach studenckich, na których prowadzący laboratoria może zadać studentom jako formę ćwiczenia wykonanie przeglądu prac swoich kolegów. Przegląd taki wykonywany jest według kryteriów ustalonych przez prowadzącego i jest anonimowy (oceniający nie zna autora pracy). Na potrzeby procesu zbierania wymagań, a później projektowania systemu, zajęcia studenckie zostały uogólnione do postaci kursu, a studenci do kursantów. W kontekście systemu laboratoria studenckie nie wyróżnia nic, co by nakazywało użyć formy bardziej szczegółowej. Ponadto takie określenia zaznaczają, że system jest uniwersalny i może być użyty nie tylko w ramach zajęć studenckich, lecz w trakcie jakichkolwiek kursów związanych z nauką programowania.



## **2. Wymagania**

### **2.1. Cel systemu**

Wymagania stawiane systemowi do wzajemnej oceny kodu, wykorzystywanemu w procesie nauki są inne niż przypadku pracy zawodowej. Celem systemu jest spowodowanie, aby kursant-recenzent przyjrzał się pracy wykonywanej przez innych kursantów. Pozwoli mu to spojrzeć na kod z innej perspektywy. Możliwe jest, że wtedy zauważy inne elementy, rozważy nowe sposoby rozwiązania, nauczy się czegoś. System, który spełni postawione wymagania potencjalnie może wspierać wiele scenariuszy. Osoba oceniającą może być zarówno „poprowadzona za rękę” poprzez przekazanie jej podczas procesu oceny szczegółowych instrukcji, jak i pozostawiona sama sobie posiadając jedynie ogólne wskazówki i pytania.

### **2.2. Aktorzy systemu**

System musi uwzględniać następujące role:

- prowadzący kurs (ang. course master),
- zleceniodawca przeglądu (ang. review supervisor),
- uczestnik kursu (ang. course participant),
- autor pracy, osoba oceniana (ang. assessed person),
- recenzent, osoba oceniająca (ang. reviewer, assessor).

### **2.3. Cechy systemu**

System wykonany w ramach niniejszej pracy inżynierskiej, w porównaniu do tradycyjnych systemów wspomagających proces przeglądów nieformalnych, wyróżnia się w następujących elementach:

- przegląd zleca osoba nadrzędna (zleceniodawca),
- oceniana jest całość, nie pojedyncza zamiana,
- ocena następuje poprzez wypełnienie ankiety,
- ocena następuje według określonych kryteriów,
- ocena jest dokonywana anonimowo.

Różnice wynikają z innej grupy docelowej, która ma system wykorzystywać, i celom, którym ma służyć. Jest to jednocześnie spis wymagań postanowionych w stosunku do tworzonego oprogramowania, które zostały przewidziane i zrealizowane.

### **2.3.1. Zleceniodawca przeglądu**

Wykorzystując system wspomagający przegląd koleżeński w zakładzie pracy zleceniodawcą przeglądu może być zarówno kierownik, jak i pracownik (choć z definicji należało by mówić tylko o koledze - pracowniku). Wybór osoby oceniającej także leży w gestii obydwu tych osób. Wszystko zależy od tego, jakie są przyjęte w firmie (projekcie) zasady, i jaka jest motywacja przeglądu. Może tu chodzić zarówno o sprawdzenie kodu osoby mniej doświadczonej, przez bardziej doświadczoną - gdzie głównym celem jest poprawa jakości kodu, jak i odwrotnie - gdzie mimo świeżego poglądu prawdopodobnie większy zaobserwowany zysk będzie w aspekcie edukacyjnym (zobaczeniu jak pewne problemy rozwiązują starsi, bardziej doświadczeni koledzy).

W opracowanym systemie przegląd jest rodzajem ćwiczenia, w szczególnym przypadku pracy domowej. Wykonanie przeglądu zawsze zleca osoba nadrzędna - prowadząca kurs. W chwili obecnej nie przewidziano możliwości wskazania kto ma czytać pracę ocenić - jako element nieistotny przydział został zrealizowany losowo, jednak z uwzględnieniem tego, że powinien być równomierny - tj. każda praca oceniona tyle samo razy.

### **2.3.2. Ocena całości**

W typowym systemie często nie chodzi o przegląd całej aplikacji - gdyż jest to często niemożliwe z uwagi stopień jej skomplikowania. Osoba nr 1 implementując nową funkcjonalność, lub modyfikując istniejącą potencjalnie eliminuje, lub wprowadza nowe błędy w systemie. Motywacją przeglądu kodu przez inną osobę (nr 2) jest między innymi jest poprawienie jakości tego kodu, eliminacja defektów, skupiając się na tym, co zostało w nim zmodyfikowane. Historia zmian jest istotną kwestią w tej motywacji.

W opracowanym systemie historia zmian nie jest istotna. Z założenia program nie jest długi, a osoba oceniająca nie musi znać historii, wiedzieć jak wyglądał rozwój. Ważny jest efekt końcowy - i to on powinien zostać oceniony, w całości.

### **2.3.3. Sposób oceny**

Typowy system zwykle umożliwia napisanie komentarza: do linii kodu, do funkcji, do pliku, do całej ocenianej zmiany. Nie ma tu jednak żadnej standaryzacji. Każdy wpisuje swoje uwagi w miejscu, które uważa za najodpowiedniejsze. Zwykle wybór ten zależy

od tego, jak bardzo szczegółowa uwaga zostaje zanotowana. Trudno więc porównać otrzymane uwagi, w przypadku gdy ten sam fragment przegląda wiele osób.

W opracowanym systemie recenzent nie wpisuje oceny bezpośrednio komentując kod. Prowadzący może określić zarówno szczegółowe kryteria jak i dać jedynie ogólne wskazówki/pytania. Ważne jest, by możliwe było łatwe porównanie opinii różnych uczestników kursu na daną realizację zadania (w przypadku, gdy pracę oceniało wiele osób). Ustalonym sposobem oceny spełniających to wymaganie jest wypełnienie przez recenzenta odpowiednio przygotowanej ankiety. Ankietę przygotowuje prowadzący uwzględniając co było w danym zadaniu ważne, jaki scenariusz chciałby zrealizować, oraz jaki cel chciałby osiągnąć w tym ćwiczeniu. Ankieta jako sposób oceny daje szeroką gamę możliwości i czyni system bardziej wszechstronnym.

#### **2.3.4. Kryteria oceny**

W typowym systemie kryteria oceny mogą dotyczyć zgodności napisanego kodu z zasadami przyjętymi w firmie - dla przykładu: stosowane formatowanie kodu, unikanie niezalecanych funkcji. Każdy przegląd jest jednak inny, dotyczy czegoś innego - brak jest formalnej procedury.

W opracowanym także można przygotować ogólny zestaw pytań, pasujący do każdego zadania. Typowy przebieg jest w nim jednak inny. Tutaj proces oceny dotyczy wcześniej przygotowanego, dokładnie przemyślanego zadania. W zamyśle prowadzącego kurs było ułożenie tego zadania tak, by np. zasadnym było wykorzystanie konkretnego wzorca programistycznego. Łatwo jest przygotować wyspecjalizowane kryteria oceny, którymi powinien podążać recenzent. Punkt ten naturalnie związany jest ze sposobem oceny i uzupełnia wymóg dostosowania sposobu oceny.

#### **2.3.5. Anonimowość przeglądanej pracy**

Typowa ocena jest imienna - to znaczy nie ma powodu, by ukrywać informację kto jest autorem ocenianej pracy. Realizowany system zakłada jednak, że uczestnik kursu nie wie czyją pracę ocenia. Anonimowe przedstawienie pracy zapewnia, że recenzent nie będzie się sugerował faktem, iż ocenia osobę, której wyniki są zwykle powyżej lub poniżej przeciętnej. Taka autosugestia może negatywnie wpłynąć na bezstronność oceny, szczególnie na etapie nauki. Dlatego też system zapewniać musi, iż recenzent nie zostanie poinformowany o personaliach autora kodu.

### **3. Potencjał do badań naukowych**

Wymagania postawione przed systemem, w celu zrealizowania określonego celu biznesowego, jasno ukazują, że opracowany system odbiega od definicji „przeglądu nieformalnego”, jakim jest tytułowy „przegląd koleżeński”. Odstępstwa, takie jak wprowadzenia procedury oceny, czy wymuszenie tej oceny przez wydanie polecenia osoby nadrzędnej, skłaniają ku klasyfikowaniu go jako "przeglądu formalnego". Trudno jednak wskazać bez wątpliwości jego klasyfikację. System ten jest swego rodzaju hybrydą rozwiązań opisanych już w książkach, i zbadanych pod kątem skuteczności.

Wniosek ten zachęca do przeprowadzania badań nad skutecznością takiego mieszanego rozwiązania. Nie tylko w kwestii skuteczności wykrywania defektów, ale także w kontekście zyskiwanych umiejętności. Warto sprawdzić, czy istnieje zależność pomiędzy tym, że kursanci regularnie oceniali w ten sposób prace kolegów, a ich wynikami w kursie. Potencjalnym zyskiem może być też spłaszczenie krzywej nauki, szybsze nabieranie właściwych nawyków czy staranniejsze wykonywanie zadań, z uwagi na sam fakt bycia ocenianym nie tylko przez prowadzącego kurs, ale także przez kolegów.

Przeprowadzanie tego typu badań wykracza jednak poza zdefiniowany zakres pracy. Praca skupia się nad jak najlepszym zaprojektowaniem systemu, implementacją zgodną ze sztuką, oraz opisaniem zrealizowanego systemu.

## **4. Sposób realizacji wymagań**

Wymagania nie wymuszają sposobu ich realizacji. W czasie projektowania systemu autor przemyślał kilka rozwiązań i ocenił pod kątem ich przydatności w realizowanym systemie, trudności implementacji i atrakcyjności biznesowej.

### **4.1. Źródło ocenianych prac**

Pierwszą ważną kwestią jest źródło ocenianych prac. Prowadzący kurs mógłby na przykład udostępniać poprzez system zadanie do realizacji, a kursanci po jego wykonaniu wgrywać rozwiązanie do systemu za pomocą odpowiedniego formularza. Takie rozwiązanie nie uzależnia od żadnych zewnętrznych usług. Problemy rodzi dopiero możliwość aktualizacji zadania, notowanie postępów w czasie. Nie jest to także atrakcyjne wyjście z biznesowego punktu widzenia.

Wzorując się jednak na istniejących narzędziach zauważyć należy, że wykorzystują one do działania systemy kontroli wersji (ang. revision control system). Wykorzystanie takiego systemu nie koniecznie ułatwia implementację systemu. Znacząco usprawnia jednak prowadzenie kursów, zwłaszcza gdy dodatkowo wykorzystany zostanie usługa pozwalająca przechowywać repozytorium (ang. repository) systemu kontroli wersji (pliki, i historię ich zmian) online. Kontrola wersji jest aktualnie nieodłącznym elementem programowania. Dlatego projektowany system w znacznej części opiera się o kontrolę wersji, i usługi z tym związane. Wybrany źródłem ocenianych prac jest usługa przechowująca historię wersji w Internecie. Naturalnym wydaje się próba integracji z już wykorzystywanymi usługami, ma to też uzasadnienie biznesowe.

### **4.2. Sposób autoryzacji użytkownika**

Użytkownik musi mieć konto w systemie, aby mógł zobaczyć zlecenia przeglądów, które zostały mu zadane, oraz na nie odpowiedzieć. Wymóg logowania zapewnia także, że nikt nie wypełni jego pracy. Klasycznym sposobem autoryzacji użytkowników jest założenie mu konta w serwisie i przekazanie danych autoryzujących, lub pozwolenie na samodzielne założenie konta (ewentualnie połączone z automatycznym lub manualnym mechanizmem potwierdzania prawidłowości wprowadzonych danych).

Należy się jednak zastanowić, czy jest konieczne, aby kursant musiał mieć kolejne konto. W związku z założeniem, iż projekt opiera się na usługach związanych kontrolą wersji autor rozważył pogłębienie integracji. Usługi takie często oferują mechanizmy dostępu do zasobów prywatnych innym usługom, bez przekazywania danych autoryzujących. Przykładem takich sposobów autoryzacji są SAML, OpenID, OAuth1, OAuth2 [12]. Taki mechanizm łatwo wykorzystać jako sposób logowania za pomocą konta w aplikacji trzeciej. Wystarczy, że wymagamy od kursanta posiadania konta w usłudze związanej z systemem kontroli wersji - nie ma potrzeby wymagać od niego jeszcze jednego konta. Rozwiązanie takie daje jeszcze jedną korzyść - brak konieczności dodatkowego łączenia pomiędzy kontem w serwisie a kontem w usłudze kontroli wersji.

### **4.3. Anonimowość przeglądanej pracy**

Istotnym wymaganiem jest anonimowość przeglądanej pracy. Recenzent nie może wiedzieć, czyją pracę ocenia. Nie można więc po prostu mu przekazać adresu, gdzie autor umieścił swoje rozwiązanie. Przekazujemy w ten sposób wszystkie meta-dane tam zawarte, w tym autora pracy. Problem ten autor rozważył uwzględniając dodatkowe założenie integracji z usługą kontroli wersji. Autor nie dokonał analizy technik możliwych do zastosowania w przypadku innego źródła ocenianych prac.

Jednym z pomysłów rozwiązania tego problemu jest założenie kursantom kont prywatnych w usłudze przechowywania repozytorium historii wersji. Każde takie konto może mieć nazwę, która nie musi zdradzać personaliów kursanta. Prywatne konta jednak częściowo nie spełniają wymagania, mówiącego, że ewentualne usługi trzecie muszą być darmowe. W przypadku poziomu akademickiego często można poprosić o takie konta, dla potrzeb uczelni. Chcąc jednak uogólnić system nie można przyjąć tego rozwiązania jako satysfakcjonującego.

Innym, wykorzystanym, sposobem jest skopiowanie danej pracy, i przekazanie recenzentowi kopii. Kopia taka, odpowiednio przygotowana, dokonana przy użyciu specjalnie do tego przeznaczonego konta, spełnia wymóg anonimowości. Autor widniejący pod przekazanym adresem nie odzwierciedla wtedy prawdziwej informacji, za każdym razem wskazuje na to samo bezosobowe konto.

## 5. Wykorzystane technologie

### 5.1. Spis użytych technologii

System został napisany jako aplikacja webowa, w języku Java, w wersji 1.8. Java to obiektowy, silnie typowany, wieloplatformowy język programowania, obecnie rozwijany przez firmę Oracle Corporation. Ponadto do realizacji systemu wykorzystano takie istniejące rozwiązania jak:

- Git - rozproszony system kontroli wersji (ang. version control system). System kontroli wersji pozwala śledzić zmiany w dokumencie. Zapisuje on kto, kiedy i jakie modyfikacje w dokumencie dokonał. Programowanie jest dziedziną, gdzie systemy kontroli wersji wykorzystuje się powszechnie. Ułatwiają one pracę wielu osób nad jednym dokumentem, wycofywanie nieprawidłowych zmian, znalezienie winnego błędnej modyfikacji, czy proces przeglądu kodu [10]. W programowaniu Git to jeden z obecnie (2014) najpopularniejszych systemów kontroli wersji, zyskuje on na znaczeniu w ostatnich latach (2011-2014) [4];
- GitHub - usługa internetowa, pozwalająca na przechowywanie repozytoriów Git. Oferuje całą bazową funkcjonalność systemu Git, a także ją dodatkowo ją rozszerza. Z usługi tej korzysta obecnie 12,4 mln osób, którzy rozwijają projekty w ramach 31,7 mln repozytoriów systemu kontroli wersji [8]. Darmowe konto pozwala przechowywać nieograniczoną liczbę repozytoriów publicznych, do których można zaprosić nieograniczoną liczbę współautorów. Za możliwość utworzenia repozytorium prywatnego trzeba zapłacić;
- Gradle - system zarządzania zależnościami dla języka Java. Gradle jest alternatywą dla znacznie popularniejszych systemów takich jak Maven, czy Ant [4]. Autor dokonał wyboru systemu zarządzania zależnościami pomiędzy systemami Maven i Gradle - systemy te są wspierane przez wybraną bazową platformę programistyczną (Spring Framework). Ostateczny wybór jest kwestią indywidualnych upodobań autora. Autor uważa, że składnia używana przez Gradle jest znacznie czytelniejsza niż w Maven - w szczególności lista zależności. System zarządzania zależnościami odpowiada za proces budowania do wybranego wspieranego formatu wynikowego, a także za pobranie pakietów (plików .jar) zależności przed procesem budowania

oprogramowania. Dzięki temu nie ma potrzeby dostarczania z kodem źródłowym systemu także pakietów zależności - te zostaną pobrane automatycznie. System zarządzania zależnościami, we współpracy ze zintegrowanym środowiskiem programistycznym (ang. Integrated Development Environment, IDE), potrafi także pobierać dokumentacje i kody źródłowe wykorzystywanych zależności - co znacząco podnosi komfort programowania;

- Spring Framework - platforma programistyczna (ang. framework) dla języka Java, dostarcza takie komponenty jak: szkielet wzorca architektonicznego Model-Widok-Kontroler (ang. Model-view-controller, MVC), mechanizm wstrzykiwanie zależności, mechanizm zarządzania transakcjami, mechanizm zarządzania dostępem do danych i inne. Platforma ta jest podstawą systemu. Definiuje ona główną architekturę systemu. Zastosowanie platformy z reguły poprawia jakość pisanego kodu, podnosi efektywność procesu programowania, a także zwiększa niezawodność systemu [3]. W zamian podnosi złożoność systemu, oraz może obniżyć jego wydajność [3];
- Spring Boot - biblioteka dostarczająca strategię „konwencja, nie konfiguracja” (ang. convention over configuration) dla platformy Spring. Jest to dodatkowa warstwa, która wykorzystuje jako zależność oryginalną platformę Spring Framework. Jej celem jest minimalizacja liczby konfiguracji wymaganej do uruchomienia strony internetowej z wykorzystaniem platformy Spring. Biblioteka ta została wydana w kwietniu 2014 roku, jako „recepta” na zbyt skomplikowanie bazowej platformy Spring Framework, która wymaga dużej liczby konfiguracji i szerokiej wiedzy na temat sposobu jej działania [16]. Oficjalnym „załącznikiem” tej biblioteki jest zgłoszenie, w systemie zgłaszania i śledzenia błędów (ang. issue tracker) platformy Spring Framework, w 2012 roku propozycji „improve support for 'containerless' web application architectures”, gdzie Mike Youngstrom opisał wady istniejącego rozwiązania i zaproponował zmianę koncepcji [16];
- Jetty - serwer stron internetowych i serwletów języka Java, napisany w całości w tym języku. Może działać w trybie wbudowanym (ang. embedded mode). Jeden z dwóch, obok Tomcat, serwerów wspieranych przez Spring Boot. Autor wybrał ten serwer na podstawie własnych preferencji, kierując się głównie historią wersji obydwu serwerów, która wskazuje iż rozwój Jetty jest aktywniejszy, i szybciej podąża za nowymi technologiami;
- H2 - system zarządzania relacyjną bazą danych (ang. relational database management system, RDBMS). Jest napisany w języku Java, wspiera standard SQL, charakteryzuje go wysoka wydajność w trybie wbudowanym i obsługa transakcji bazodanowych [13].



Autor w ramach pracy zdecydował się użyć bazy danych w trybie wbudowanym. Pod względem wydajności wybrana baza z sukcesem konkuruje z konkurencją, także tą, która wymaga do działania serwera, tzn. działających w trybie klient-serwer (ang. client-server mode) [13]. Autor nie rozważał możliwości użycia baz nierelacyjnych (typu NoSQL), ani baz obiektowych (ang. object database);

- Hibernate ORM - biblioteka dla języka Java dostarczająca mechanizm mapowania z relacyjnego systemu bazodanowego na obiekty w języku Java. Implementuje ona oficjalny standard mapowania relacyjno-obiektowego (ang. object-Relational Mapping, ORM) dla języka Java: Java Persistence API (JPA). Mechanizm mapowania relacyjno-obiektowego pozwala odwoływać się do danych, zapisanych w bazie w sposób relacyjny, w sposób obiektowy z poziomu języka programowania. Klasa reprezentująca tabelę w bazie danych nazywa się encja (ang. entity). Za operację typu CRUD (ang. create, read, update, delete; pol. utwórz, odczytaj, aktualizuj, usuń) na warszawie bazy danych odpowiada menadżer encji (ang. entity manager). Wykorzystanie ORM odciąża programistę od potrzeby pisania zapytań SQL (operowanie na encjach jest także bardziej naturalne dla programowania obiektowego), oraz zabezpiecza przed atakami typu „SQL injection”;
- Thymeleaf - biblioteka dla języka Java dostarczająca silnik szablonów do przetwarzania i generowania dokumentów HTML, XML, plików JavaScript, CSS. Pisany system jest dynamiczny. Informacje wyświetlane użytkownikowi zależą od aktualnego stanu systemu. Wykorzystanie silnika szablonów zapewnia wydzielenie warstwy widoku, zgodnie z wykorzystanym wzorem model-widok-kontroler. Widok zawiera referencję do modelu, otrzymaną od kontrolera skąd może pobrać za każdym razem aktualne dane. Wybrany silnik szablonów posiada wsparcie dla najnowszej wersji standardu HTML (HTML5), internacjonalizacji oraz automatycznie kontroluje poprawność składniową generowanego dokumentu;
- OAuth2 - otwarty standard autoryzacji użytkowników. Określa on przebieg procesu dostępu do zasobów aplikacji trzeciej, bez udostępniania danych autoryzujących. Użytkownik w systemie musi być autoryzowany. Ustalonym sposobem autoryzacji jest OAuth2, gdyż to właśnie ten sposób jest wspierany przez usługę GitHub;
- pac4j: Spring Web MVC / Spring Boot - biblioteka dla języka Java dostarczająca mechanizmy pozwalające zabezpieczyć systemy webowe przed dostępem osób nieautoryzowanych. Budowa tego mechanizmu jest modułowa. Dołączając odpowiedni można przy jej użyciu zrealizować właściwie każdy sposób uwierzytelnienia (ang. authentication) i autoryzacji (ang. authorization) użytkownika;

- pac4j-oauth: moduł biblioteki pac4j pozwalający zrealizować system autoryzacji z użyciem standardu OAuth2. W projektowanej aplikacji rolę serwera autoryzacji (ang. authorization server) i serwera zasobów (ang. resource server) pełni usługa GitHub;
- GitHub API v3 (GitHub API) - interfejs programistyczny pozwalający w sposób programowy komunikować się z usługą GitHub w celu uzyskania lub modyfikacji zasobów, które one przechowuje. Interfejs programistyczny udostępniony przez GitHub jest bardzo zaawansowany. Można przy jego użyciu wykonać właściwie wszystkie akcje dostępne w serwisie, w tym te najbardziej interesujące w kontekście budowanego systemu: odczyt podstawowych informacji na temat repozytorium (lista powieści, lista gałęzi, nazwa gałęzi głównej, adres do pobrania repozytorium, możliwość utworzenia nowego);
- com.squareup: OkHttp - biblioteka dla języka Java, implementacja klienta http z możliwością zapamiętywania odpowiedzi w pamięci podręcznej (ang. cache);
- org.kohsuke: github-api - biblioteka dla języka Java będąca implementacją komunikacji z GitHub API w tym języku. Udostępniona dokumentacja API opisuje istniejące zasoby i sposoby odwołania do nich. Nie istnieje jednak oficjalna implementacja dostępu do wspomnianego API w jakimś konkretnym języku programowania. Wybrana biblioteka udostępnia obiektowo-zorientowaną reprezentację komunikacji ze wspomnianym API. Autor biblioteki deklaruje iż pokrywa ona „znaczną część” API. Autor sprawdził, iż pokrycie jest wystarczające do zrealizowania systemu;
- Eclipse JGit - biblioteka dla języka Java pozwalająca zarządzać repozytorium Git programowo, z poziomu tego języka. W opinii autora akcje takiej jak: zatwierdzanie zmian, synchronizacja repozytorium lokalnego ze zdalnym za pomocą GitHub API są mało intuicyjne i stosunkowo trudne. Autor zdecydował się użyć innej biblioteki do tych zadań. Wybrana biblioteka JGit realizuje te zadania wykorzystując system plików - działa podobnie jak program „git”. Jest to biblioteka generyczna - znajduje zastosowanie zarówno dla repozytoriów lokalnych, jak i dla dowolnej usługi przechowującej repozytorium Git zdalnie;
- Apache POI - biblioteka dla języka Java, umożliwia generowanie dokumentów pakietu Microsoft Office (w tym arkuszy Excel) programowo, z poziomu tego języka. Podsumowania przeglądów system generuje jako arkusz MS Excel. Zastosowana biblioteka potrafi generować zarówno starsze pliki pakietu Office (przed wersją 2007), jak i nowsze (od wersji 2007, z dodatkową literą „x” w rozszerzeniu);

## **5.2. Wielokrotne użycie**

Wymyślanie koła na nowo nie zawsze jest dobrym rozwiązaniem. Pisaniu aplikacji sprzyjała idea wielokrotnego użycia. Wykorzystano jak najwięcej istniejących rozwiązań, w większości znanych i dojrzałych. Istotnym założeniem projektowania była integracja z istniejącymi i powszechnie wykorzystanymi usługami.

### **5.2.1. Git i GitHub**

System kontroli wersji Git, i usługa GitHub odgrywają w projekcie kluczową rolę. Oparte na tym zostało wiele elementów: system logowania, system anonimizacji ocenianych prac, prezentacja ocenianej pracy.

Wyborowi systemu kontroli wersji Git sprzyjała głównie jego rosnąca popularność [4], jak i dostępność darmowych usług pozwalających przechowywać historię, zapisaną w tym systemie, w sieci internet.

Na wybór usługi GitHub jako miejsca składowania kodu wpływ miał głównie fakt, iż udostępnia ona programistyczne API pozwalające sprawnie sprawdzić zasoby tego serwisu z poziomu kodu, oraz może pełnić rolę serwera autoryzacji OAuth2. Pod uwagę autor wziął także dwa inne darmowe serwisy. Konkurencyjna usługa GitLab wykazuje poważne braki w tych wymaganiach - udostępnia API, lecz nie wszystkie informacje o zasobach, istotne z punktu tworzonego systemu, są dostępne w ten sposób [9]. Inną konkurencyjną usługą jest Bitbucket, ten jednak nie jest chętnie używany z uwagi na mało intuicyjny interfejs [2].

### **5.2.2. Sposób uwierzytelnienia - OAuth2**

Użytkownik nie posiada oddzielnego konta w systemie. Jego uwierzytelnienie następuje następuje z użyciem usługi GitHub. Rozwiązanie takie nie tylko ma sens biznesowy (co autor udowodnił podczas opisu sposobu realizacji wymagań), ale także odciąża programistę i administratora systemu od obowiązku zapewnienia przechowywania haseł w sposób bezpieczny. Co prawda system wciąż przechowuje dane osobowe (imię i nazwisko, powiązanie z kontem GitHub), jednak poziom wrażliwości danych ulega obniżeniu.

Zastosowanym sposobem autoryzacji jest protokół OAuth2. Kluczowym elementem mający wpływ na wybór technologii autoryzacji było wsparcie jej przez usługę GitHub. Protokół OAuth2 określa sposób dostępu do zasobów usługi trzeciej (w tym wypadku usługi GitHub) bez udostępniania danych autoryzujących (w tym przypadku hasła w usłudze GitHub). Uzyskane zezwolenie dostępu do zasób łatwo wykorzystać jako sposób uwierzytelnienia w systemie. W tym celu jedynym zasobem wykorzystywanym w systemie jest nazwa użytkownika, który dostępu udzielił.

Protokół OAuth2 definiuje następujące role:

- właściciel zasobu (ang. resource owner) - użytkownik będący właścicielem zasobu. W tym przypadku zasobem jest nazwa użytkownika, a uwierzytelniany użytkownik jej właścicielem;
- klient (ang. client) - serwis, który chce poznać zasób. W tym przypadku projektowany system chce poznać nazwę użytkownika w usłudze serwera zasobów;
- serwer zasobów, dostawca API (ang. resource server) - usługa, do której dostęp należy uzyskać, aby zapoznać się z zasobem. W tym przypadku jest to usługa GitHub;
- serwer autoryzacji, dostawca tożsamości (ang. authorization server) - usługa, która udziela klucze dostępu do zasobów. W tym przypadku nie ma wydzielonej takiej roli. Za czynność tą odpowiada sama usługa GitHub.

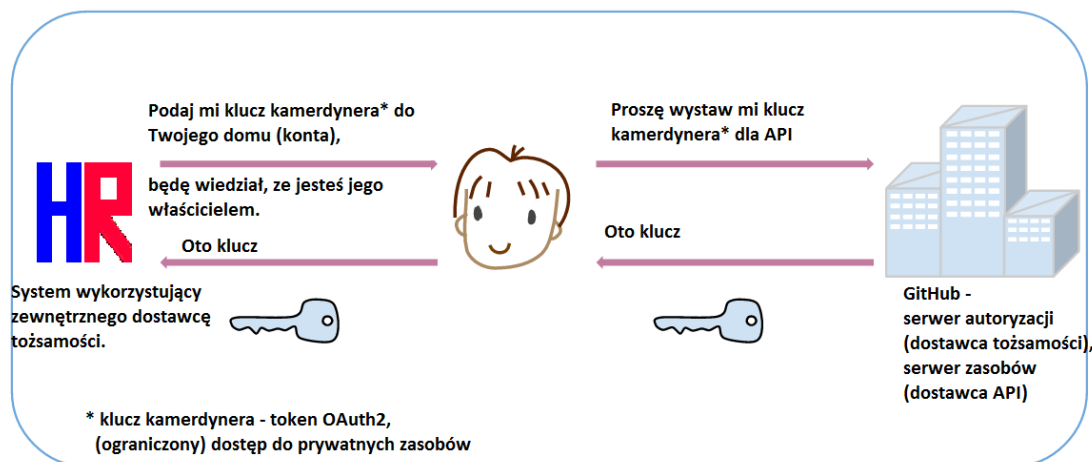
Ponadto w omawianym protokole występują dwa ważne pojęcia:

- zgoda na dostęp (ang. authorisation grant) - potwierdzenie właściciela zasobu, że udziela klientowi dostępu do niego.
- token (ang. access token) - klucz, za pomocą którego klient może przeczytać zasób u dostawcy zasobu.

Proces uwierzytelnienia w wykorzystanym protokole wygląda następująco:

1. użytkownik otwiera system. W tym momencie nie jest zalogowany. Klikając odpowiedni przycisk stwierdza chęć autoryzacji z użyciem określonego serwisu;
2. użytkownik opuszcza system - zostanie przekierowany do serwera autoryzacji. Jedynym dostępnym serwisem uwierzytelniającym w systemie jest usługa GitHub. W tym momencie system przyjmuje rolę klienta, a usługa GitHub serwera autoryzacji;
3. użytkownik dowiaduje się od serwera autoryzacji, jaki klient prosi o zgodę na dostęp i do jakiego zasobu. Użytkownik potwierdza wydając zgodę na dostęp;
4. użytkownik opuszcza serwer autoryzacji - zostaje przekierowany z powrotem do systemu. W sposób dla niego transparentny przekazuje do systemu token, który otrzymał od serwera autoryzacji. W praktyce potwierdzając dostęp wysłał on formularz, kierujący do systemu, gdzie w ukrytym polu przekazywany jest wspomniany token;
5. system odwołuje się do serwera zasobów, gdzie za pomocą tokena uzyskuje dostęp do zasobu, o który prosił. System poznał w tym momencie nazwę użytkownika i posiada wszystkie dane by wykonać autoryzację i określić uprawnienia użytkownika, który właśnie się zalogował. Użytkownikowi zostaje wyświetlony widok zgodny z jego rolą i uprawnieniami.

Ilustracja technicznego przebiegu uwierzytelnienia z użyciem protokołu OAuth2 została załączona jako rysunek 5.1.



Rysunek 5.1. Przebieg autoryzacji z użyciem protokołu OAuth2. (Tłumaczenie własne zasobów Wikimedia Commons. Plik udostępniony na licencji Creative Commons CC0 1.0 Uniwersalna Licencja Domeny Publicznej. Pierwotni autorzy: Saqibali, Amada44, Perhelion.)

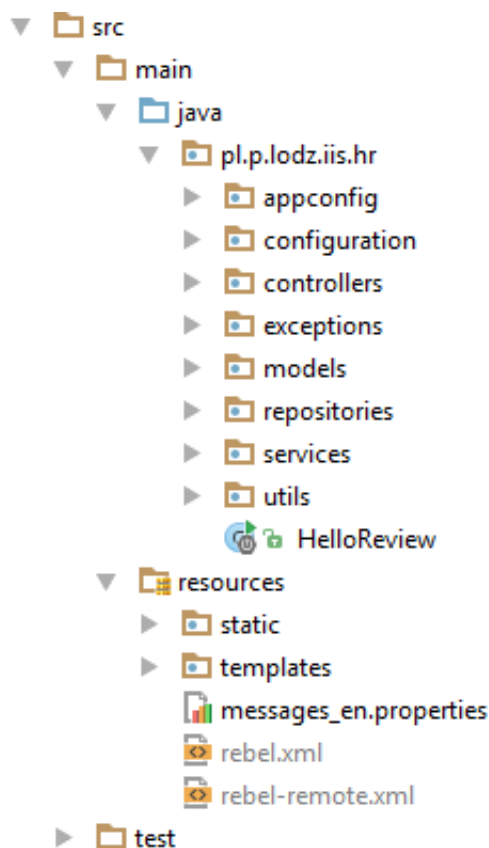
### 5.2.3. System formularzy

System formularzy autor napisał na nowo, mimo iż istnieje wiele systemów tego typu. Podstawowym wymaganiem w stosunku do systemu formularzu był dostęp do programistycznego API, który pozwoli założyć, kasować i odczytywać historię odpowiedzi z poziomu kodu. Usługa musiała być też darmowa, lub udostępniać darmowy dostęp do celów akademickich. Przetestowano wiele rozwiązań, jednak żadna z nich nie była satysfakcjonująca dla potrzeb tego projektu. Z tego autor powodu zdecydował się napisać prosty autorski system formularzy, dopasowany na miarę do realizowanego systemu.

## 6. Architektura systemu

System został napisany jako aplikacja webowa z użyciem wzorca architektonicznego Model-Widok-Kontroler (ang. Model-View-Controller, MVC). Wzorzec ten zakłada podział aplikacji na 3 części:

1. model (ang. model) - który odpowiada za dane i logikę biznesową;
2. widok (ang. view) - który odpowiada na interfejs, za pomocą którego użytkownik komunikuje się z aplikacją;
3. kontroler (ang. controller) - który odpowiada za interakcję z użytkownikiem, przyjmuje od niego dane wejściowe (przekazane za pomocą widoku), przetwarza informacje a następnie aktualizuje model i/lub widok;

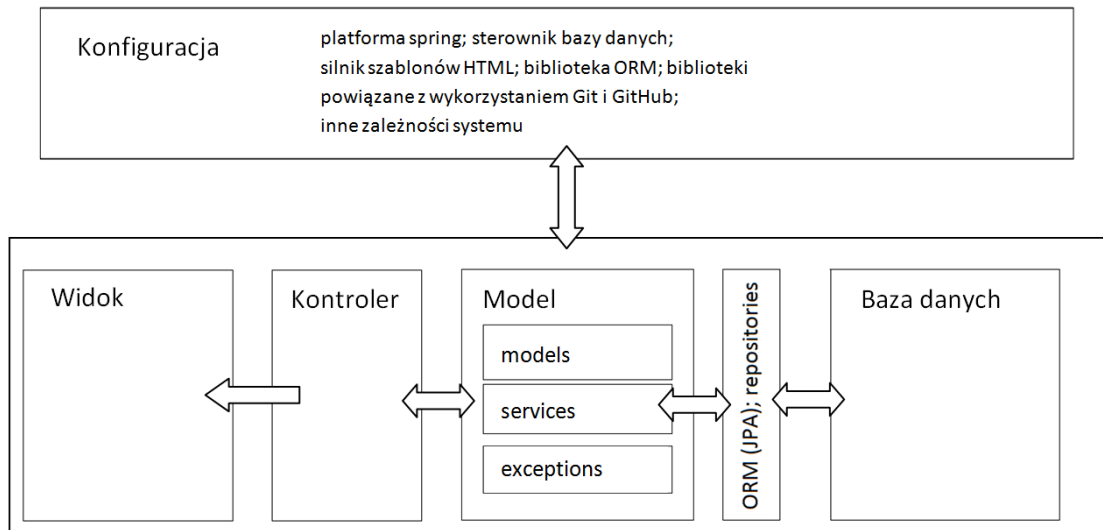


Rysunek 6.1. Architektura systemu - struktura plików

Rysunek 6.1 przedstawia strukturę plików w projekcie. System składa kolejno z takich pakietów jak:

- appconfig - zawiera klasy odpowiedzialne za odczyt z pliku konfiguracji systemu;
- configuration - zawiera klasy odpowiedzialne za skonfigurowanie systemu, to znaczy skonfigurowanie bazowej platformy Spring Boot, połączenie z nią używanych zależności (innych technologii), konfigurację serwera http;
- controllers - klasy wchodzące w skład warstwy kontroler. Odbierają żądania http, przekazują komunikaty do modelu, i renderują odpowiedni widok zawierający odpowiedź;
- exceptions - wydzielona część warstwy kontroler. Zawiera deklaracje klas odpowiedzialnych za obsługę sytuacji wyjątkowych;
- models - część warstwy model. Zawiera encje - klasy odpowiadające za składowanie danych;
- repositories - część warstwy model. Warstwa pośrednia pomiędzy sterownikiem dostępu do bazy danych, a encjami. Zawiera, zgodnie ze wzorem repozytorium (ang. repository pattern), klasy umożliwiające operacje typu CRUD (utwórz, odczytaj, zaktualizuj, usuń; ang: create, read, update, delete), oraz operacje selekcji warunkowych w systemie bazodanowym;
- serves - część warstwy model. Zawiera usługi operujące na encjach, w praktyce ten pakiet zawiera logikę biznesową;
- utils - dodatkowe klasy narzędziowe, nie związane bezpośrednio z aplikacją, lecz użyteczne przy jej tworzeniu;
- resources/static - część warstwy view. Zawiera wszystkie pliki statyczne, które nie zależą od kontekstu wykonania, takie jak: obrazki, pliki skryptów JavaScript, arkusze stylu CSS;
- resources/templates - część warstwy view. Zawiera szablony generowane dynamicznie, zależne od kontekstu, i danych przekazanych do nich poprzez kontroler.

Zastosowana struktura pakietów trochę rozdrabnia ogólny podział, który jest zdefiniowany przez wzorzec MVC. Wynika to z naturalnej tendencji do separacji klas, które łączy wspólna cecha oraz chęci tworzenia systemu warstwowego, o prostej strukturze zależności. Szkic założonej architektury zawiera rysunek 6.2.



Rysunek 6.2. Architektura systemu - zależność pakietów



## 7. Konfiguracja systemu

### 7.1. Plik konfiguracyjny

Rozdział ten został poświęcony omówieniu sposobu konfiguracji systemu. Opis ten jest wstępem do szczegółowego omówienia projektu i implementacji systemu. Konfiguracja systemu odbywa się poprzez plik typu XML, którego nazwa ustalona została przez autora na „HelloReviewConfig.xml”. Plik ten musi znajdować się folderze, z którego zostaje uruchomiony system. Jego obecność, jak i wszystkich składowych znaczników jest obligatoryjna do poprawnego uruchomienia i działania systemu. Przykład konfiguracji ilustruje rysunek 7.1, który przedstawia zrzut zawartości przykładowego pliku konfiguracyjnego.



Rysunek 7.1. Konfiguracja systemu - przykładowy plik

Sekcja „generalConfig” zawiera zgrupowane ogólne luźno powiązane parametry ogólnego przeznaczenia:

- url - adres URL do systemu, wykorzystywany jest on podczas autoryzacji przy użyciu protokołu OAuth2;
- tempdir - względna lub bezwzględna ścieżka do katalogu, który będzie wykorzystywany jako folder dla plików tymczasowych podczas anonimizacji prac;
- cachedir - względna lub bezwzględna ścieżka do katalogu, który będzie wykorzystywany jako pamięć podręczna podczas wykonywania połączeń do GitHub API.

Sekcja „github” zawiera konfiguracje związane z usługą GitHub.

Podsekcja „masters” zawiera listę nazw kont w usłudze GitHub, należące do osób, które w systemie posiadają uprawnienia prowadzących kurs.

Podsekcja „courserepos” zawiera listę nazw kont w usłudze GitHub, które są wykorzystywane podczas kursów. Konta te wykorzystywane są jako źródło kursów. Na ich podstawie tworzona jest lista wyboru kursu podczas tworzenia nowego zlecenia przeglądu.

Podsekcja „dummy” zawiera token aplikacyjny usługi GitHub dla konta, które będzie wykorzystywane do anonimizacji prac. Na tym koncie przechowywane będą anonimowe kopie. Ponad to w tej podsekcji ustawiony jest domyślny tekst dla akcji „commit” w systemie kontroli wersji Git. Tekst ten zostanie użyty podczas procesu anonimizacji prac.

Podsekcja „application” definiuje dane autoryzujące aplikację w usłudze GitHub.

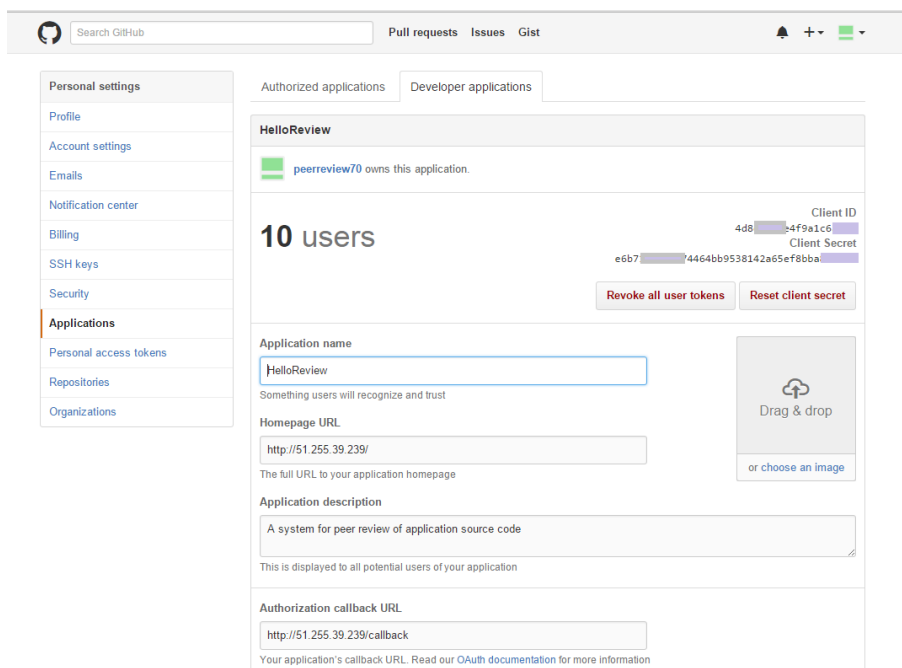
Znaczenie parametrów konfiguracji zostanie ściślej określone w kolejnych rozdziałach, przy okazji opisu miejsc, gdzie są one wykorzystane.

## 7.2. Rejestracja aplikacji w usłudze GitHub

Z uwagi na dużą skalę integracji z usługą GitHub istotna część konfiguracji systemu jest związana właśnie z tym serwisem. Autor zaleca rejestrację specjalnego konta przeznaczonego do użycia wyłączenie dla celów systemu. Jest to głównie związane z tym, że składowane na nim będą anonimowe kopie ocenianych prac. Łatwo policzyć, że jeżeli na kurs chodzi 20 uczestników i w danym momencie zarejestrujemy tylko jeden przegląd, gdzie każdy będzie musiał ocenić tylko jedną pracę to już jest 20 repozytoriów, które nie wyglądają profesjonalnie na koncie, i odwracają uwagę od wartościowej zawartości. Ich ilość może być na tyle przytłaczająca, że inne treści zostaną zupełnie niezauważone.

Rysunek nr 7.2 zawiera zrzut ekranowy z przykładem poprawnie zarejestrowanej aplikacji w serwisie GitHub. Nazwę („Application name”) i opis („Application description”) można dowolnie zmieniać. Istotne są natomiast adresy URL („Homepage URL” i „Authorization callback URL”). Ten pierwszy wskazuje na stronę domową, ten drugi natomiast musi wskazywać na podstronę „/callback”. To pod ten adres usługa GitHub przekieruje użytkownika po procesie logowania. Jest to jedyny poprawny adres, pod którym zostanie obsłużona informacja zwrotna.

Ilustrowana strona zawiera także ciągi znaków „Client ID” i „Client Secret”, które należy wprowadzić w pliku konfiguracyjnym systemu. Istotne jest, aby ciąg „Client Secret” nie upubliczniać. Jest to ciąg związany z mechanizmem autoryzacji „tożsamości” rejestrowanej aplikacji.



Rysunek 7.2. Zrzut ekranowy: zarejestrowana aplikacja w usłudze GitHub

## 8. Projekt i implementacja systemu

Autor nazwał system HelloReview (HR). Informacja ta ma charakter porządkowy, nazwa ta, lub jej skrót, występuje na załączonych rysunkach, może także wystąpić w tekście.

Przepływ danych w systemie wygląda następująco:

1. Użytkownik loguje się do systemu następuje przy użyciu konta GitHub;
2. Prowadzący zakłada kursy, do których zapisuje kursantów;
3. Prowadzący tworzy projekt z zadaniem na platformie GitHub;
4. Kursant rozwiązując zadanie powielają odpowiedni projekt na platformie GitHub;
5. Prowadzący zleca wzajemną ocenę postępów pracy dla danego projektu;
6. Kursant ocenia pracę innego studenta wypełniając ankietę;
7. Prowadzący sprawdza odpowiedzi w systemie.

Taki przebieg wydarzeń uwzględnia wszystkie wymagania w stosunku do projektowanego systemu. W rozdziale tym przedstawione zostaną szczegóły projektowe i implementacyjne na bazie przedstawionych punktów. Przedstawienie projektu za zasadzie chronologicznie ułożonych wydarzeń ma na celu ułatwienie odbioru pracy, oraz zrozumienia budowy i zasady działania systemu.

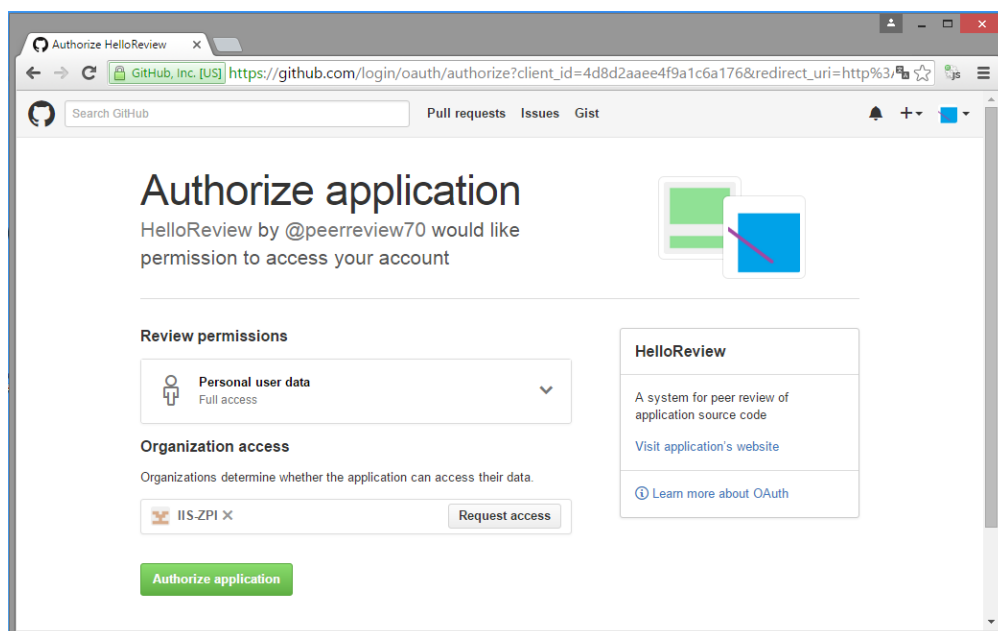
## 8.1. Logowanie do systemu

Niealogowany użytkownik po uruchomieniu strony internetowej nie posiada żadnych uprawnień. Zastaje on widok jak na zrzucie ekranowym na rysunku 8.1. Jedyną przewidzianą akcją w tym momencie jest logowanie do systemu.



Rysunek 8.1. Zrzut ekranowy: strona główna dla niealogowanego użytkownika

Logowanie do systemu odbywa się przy użyciu usługi GitHub jako serwera autoryzacji. Autoryzacja następuje przy użyciu protokołu OAuth2. Użytkownik po kliknięciu w przycisk zostaje przekierowany do platformy GitHub, gdzie potwierdza chęć udostępnienia danych systemowi, który go skierował. Wygląd potwierdzenia autoryzacji dla aplikacji przedstawia zrzut ekranowy na rysunku 8.2.



Rysunek 8.2. Zrzut ekranowy: autoryzacja aplikacji w serwisie GitHub

Jeżeli użytkownik potwierdził autoryzację system otrzymuje token autoryzacyjny, za pomocą którego może odczytać podstawowe dane na temat użytkownika. Najważniejszą informacją udostępnianą przez serwis GitHub jest nazwa użytkownika. To na jej podstawie system określa uprawnienia. Jeżeli nazwa zalogowanego użytkownika znajduje się w pliku konfiguracyjnym, na liście w sekcji „masters” to uzyskuje on dostęp z prawami prowadzącego. W przeciwnym razie ma on uprawnienia kursanta. Użytkownik, który nie zgodził się na udostępnienie danych nie może korzystać z systemu. Wyrażenie braku zgody polega na zamknięciu zakładki z akcją potwierdzenia, a więc przerwaniu procesu potwierdzenia i opuszczeniu systemu.

Charakter systemu powoduje, że nie ma tu stopniowania uprawnień. Uprawnienia są rozłączone, co ułatwiło implementację kontroli uprawnień. Wszystkie podstrony pasujące do wzorca „/m/\*\*” są przeznaczone dla prowadzącego, wzorzec „/p/\*\*” odpowiada podstronom dedykowanym kursantom. Inne zasoby są ogólnodostępne, ich zawartość może być pobrana przez każdego - są to pliki statyczne, strona główna, oraz strony błędów.

Kontrola uprawnień została zaimplementowana wykorzystując mechanizm „przechwytywania” (ang. intercept) żądań. Przed przejściem do właściwego kontrolera Spring uruchamia zdefiniowane mechanizmy wykonujące autoryzację. Rysunek numer 8.3 na stronie 31 zawiera listing kodu z klasą „InterceptorRegistryConfig”. Odpowiada ona za zarejestrowanie klas typu „interceptor”. Dwie z zarejestrowanych klas są odpowiedzialne za kontrolę uprawnień. „InterceptorHasRoleMaster” sprawdza uprawnienia do podstron dla prowadzącego, oraz odpowiednio „InterceptorHasRoleMaster” sprawdza uprawnienia dla podstron dla kursantów.

Rysunek numer 8.4 na stronie 31 zawiera listing kodu z klasą „InterceptorHasRoleMaster”. Autoryzacja polega na odpytaniu biblioteki pac4j, czy aktualnie użytkownik jest zalogowany, oraz sprawdzenie, czy jego nazwa użytkownika widnieje w konfiguracji jako poprawna nazwa konta prowadzącego. Do komunikacji z biblioteką pac4j autor napisał dodatkowo klasę użytkową „GHPac4jSecurityHelper”, która eksponuje jedynie rzeczy istotne dla systemu. Jeżeli użytkownik jest niezalogowany zostaje przekierowany do strony głównej. W przypadku wykrycia próby uzyskania dostępu do zasobów przez zalogowanego użytkownika, do których nie ma uprawnień wysyłana jest odpowiedź ze stroną błędu o kodzie „401 Unauthorized”. Autoryzacja uprawnień do podstron dla kursanta wygląda w sposób odpowiadający - za kursanta system uważa każdego, kto nie został rozpoznany jako prowadzący.

```

1 package pl.p.lodz.iis.hr.configuration;
2
3 import ...
4
10
11 @Configuration
12 @DependsOn({"pac4jConfig", "appConfig"})
13 class InterceptorRegistryConfig extends WebMvcConfigurerAdapter {
14
15     @Autowired private GitHubClient gitHubClient;
16     @Autowired private AppConfig appConfig;
17
18     @Override
19     public void addInterceptors(InterceptorRegistry registry) {
20         registry.addInterceptor(new InterceptorHasRoleMaster(gitHubClient, appConfig)).addPathPatterns("/m/**");
21         registry.addInterceptor(new InterceptorHasRolePeer(gitHubClient, appConfig)).addPathPatterns("/p/**");
22         registry.addInterceptor(new InterceptorUserInfoAppender(gitHubClient, appConfig)).addPathPatterns("/**");
23     }
24 }
25

```

Rysunek 8.3. Listing kodu źródłowego: klasa InterceptorRegistryConfig

```

1 package pl.p.lodz.iis.hr.configuration;
2
3 import ...
4
10
11 /**
12  * Interceptor to check if logged in user has "master" role.
13  */
14 class InterceptorHasRoleMaster extends HandlerInterceptorAdapter {
15
16     private final GitHubClient gitHubClient;
17     private final AppConfig appConfig;
18
19     InterceptorHasRoleMaster(GitHubClient gitHubClient, AppConfig appConfig) {
20         this.gitHubClient = gitHubClient;
21         this.appConfig = appConfig;
22     }
23
24     @Override
25     public boolean preHandle(HttpServletRequest request,
26                             HttpServletResponse response,
27                             Object handler)
28         throws IOException {
29
30         GHPac4jSecurityHelper ghSecurityHelper = new GHPac4jSecurityHelper(gitHubClient, request, response);
31
32         if (ghSecurityHelper.isAuthenticated()) {
33             if (ghSecurityHelper.isMaster(appConfig)) {
34                 return true;
35             } else {
36                 response.sendError(HttpServletResponse.SC_UNAUTHORIZED);
37                 return false;
38             }
39         } else {
40             response.sendRedirect("/");
41             return false;
42         }
43     }
44 }
45
46

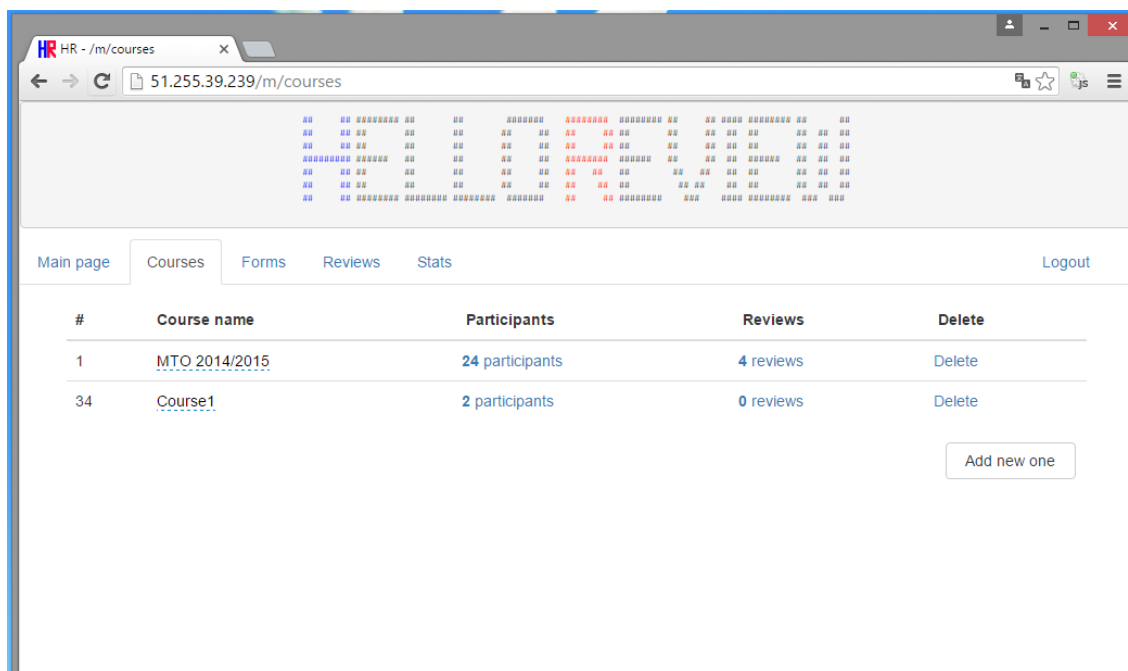
```

Rysunek 8.4. Listing kodu źródłowego: klasa InterceptorHasRoleMaster

## 8.2. Kursy i ich uczestnicy

Korzystanie z systemu należy rozpocząć od zarejestrowania kursu i przypisania do nich uczestników. System został tak zorganizowany, że na każdej podstronie widnieje lista znajdujących się zasobów danego typu (w postaci tabelarycznej), a pod nią przycisk dodania kolejnego. Obrazki 8.5, 8.6, 8.7 zawierają kolejno zrzuty ekranowe przedstawiające: przykładową listę kursów, przykładową listę kursantów, i przykład próby dodania kursu zakończonego niepowodzeniem, z powodu niepowodzenia walidacji wprowadzonych danych.

Kurs identyfikuje jedynie jego nazwa (ang. course name). W przypadku kursanta należy podać dwie wartości: personalia (zwykle imię i nazwisko) (ang. participant name) i nazwę konta w systemie GitHub (ang. GitHub nick, GitHub name). Poszukiwanie informacji dla danego kursanta, gdy ten się zaloguje do systemu, polega na porównaniu nazwy konta, z którego się zalogował z istniejącymi wpisami na temat uczestników w bazie danych. Jest to jedyny warunek kontroli uprawnień do zasobów kursanta. Tożsamość została już wcześniej potwierdzona i powiązana z nazwą konta. Na tym etapie jest to warunek konieczny i zupełnie wystarczający.

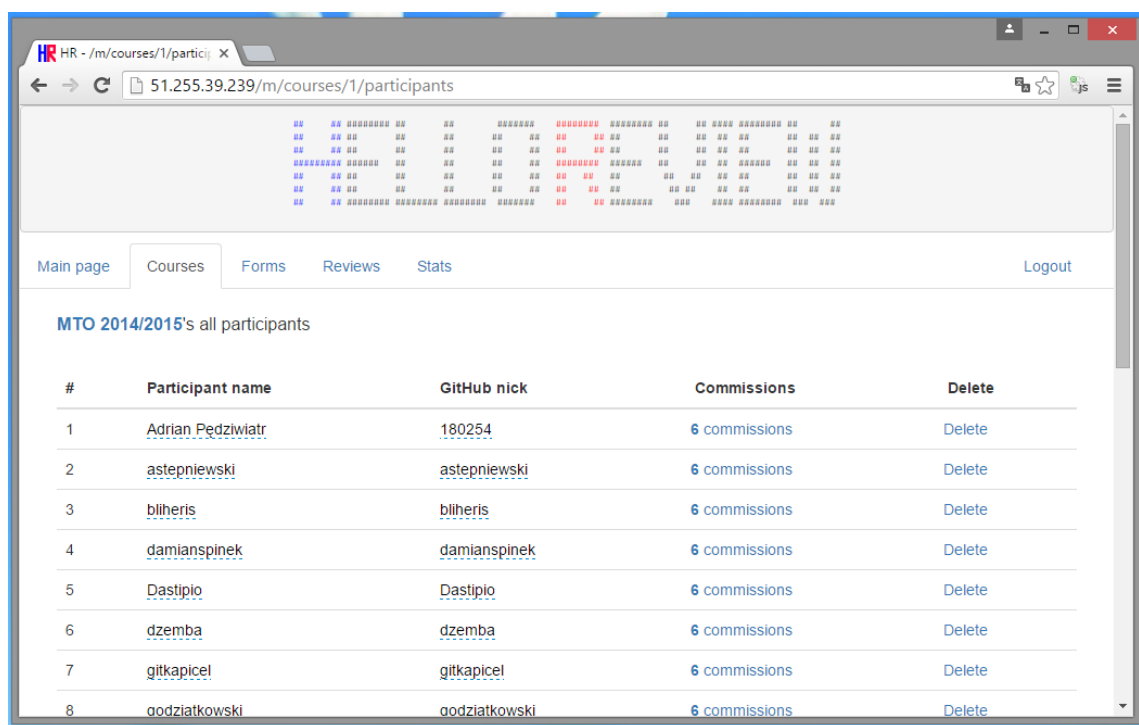


#	Course name	Participants	Reviews	Delete
1	<a href="#">MTO 2014/2015</a>	24 participants	4 reviews	<a href="#">Delete</a>
34	<a href="#">Course1</a>	2 participants	0 reviews	<a href="#">Delete</a>

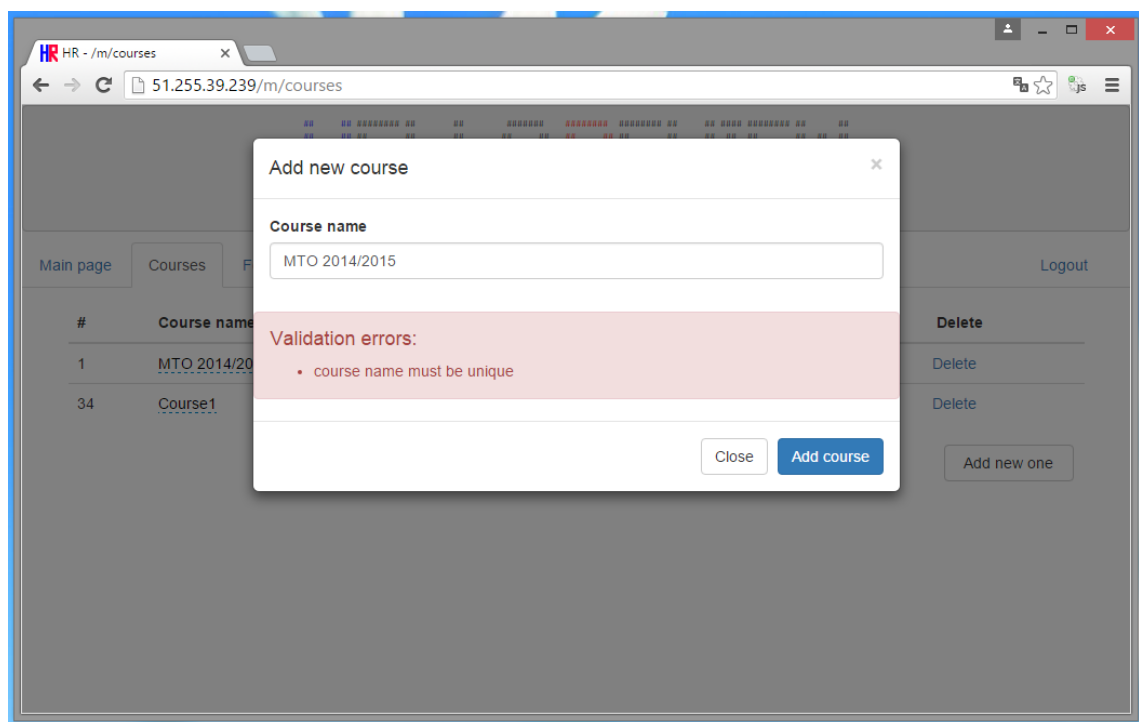
[Add new one](#)

Rysunek 8.5. Zrzut ekranowy: lista kursów





Rysunek 8.6. Zrzut ekranowy: lista kursantów



Rysunek 8.7. Zrzut ekranowy: formularz dodania kursu

### **8.3. Prowadzący umieszcza zadanie na platformie GitHub**

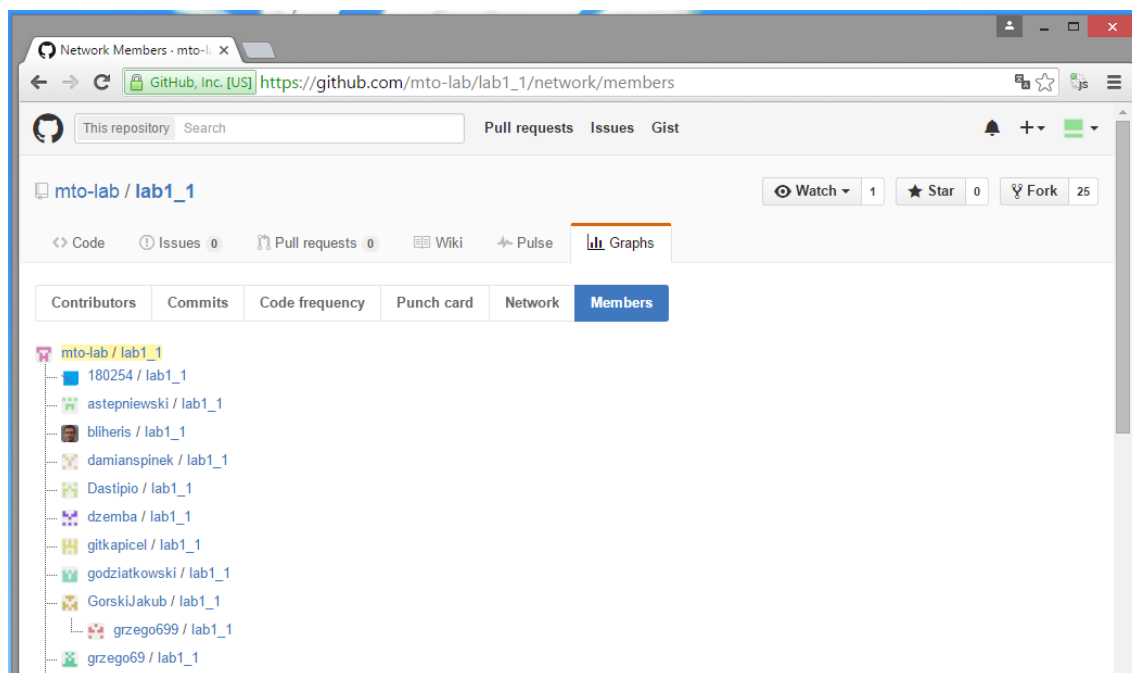
Prowadzący umieszcza dla każdego zadania nowe repozytorium Git na platformie GitHub. Przy czym nie chodzi tutaj o umieszczenie w nim samej treści zadania, a o projekt zawierający kod startowy. Założenie było wzorowane na laboratoriach, gdzie taki projekt bazowy zawierał kod, który należało rozbudować, poprawić lub uzupełnić o przypadki testowe. Nic nie stoi na przeszkodzie jednak, by repozytorium takie było podstawowym ogólnym szablonem i/lub zawierało treść polecenia. Istotne jest jednak, że dla każdego zadania należy utworzyć oddzielne repozytorium.

Założeniem jest jedno repozytorium = jeden przegląd, jedna ocena. Zgrupowanie zadań powoduje więc jednocześnie, że możliwość zlecenia przeglądu także istnieć będzie jedynie grupowo. Istnieje możliwość sprostowania, np. poprzez napisanie komentarza do zlecenia przeglądu, lecz użytkownik dostanie kopię całego repozytorium. Próba grupowania wydaje się więc być zbędną kombinacją, mimo wszystko możliwą jednak do realizacji.

Dla systemu nie są istotne nazwy repozytoriów. Przykładową konwencją może być np. „lab\_X-Y”, gdzie X to kolejny numer zajęć, natomiast Y to numer zadania w ramach danych zajęć.

## 8.4. Kursant powiela repozytorium z zadaniem

Kursant rozwiązanie zadania rozpoczyna od powielenia (ang. fork) repozytorium na platformie GitHub. W ten sposób system rozpoznaje, że kursant rozpoczął proces wykonania zadania, i znajduje jego rozwiązanie. Z uwagi na specyfikę dostępu do zasobów przy użyciu GitHub API istotne jest, aby kursant powielił projekt źródłowy. Błędem jest wielokrotne powielenie. Wielokrotne powielenie należy rozumieć jako powielenie repozytorium z konta „X”, które jest powieleniem ze źródłowego konta „Y”. Należy mieć na uwadze opisane ograniczenie i poinformować o nim kursantów. Nieprawidłowe powielenie skutkować będzie nie wykryciem rozwiązania kursanta przez system, co może budzić różne konsekwencje, w zależności od woli prowadzącego kurs. Rysunek numer 8.8 zawiera zrzut ekranowy z platformy GitHub, na którym widać, iż projekt został wielokrotnie powielony, lecz kursant o nazwie konta „grzego699” wykonał to nieprawidłowo (w rozumieniu ograniczeń systemu).



Rysunek 8.8. Zrzut ekranowy: powielenia repozytorium na platformie GitHub

## 8.5. Prowadzący zleca przegląd

### 8.5.1. Formularz oceny

Jedną ze składowych przeglądu jest formularz, za pomocą którego będzie dokonana ocena. Przed przejściem do właściwej części zlecenia przeglądu należy zastanowić się, czy zastosowanie ma jakiś z istniejących już w systemie formularzy, czy też należy dodać następny.

W skład formularza wchodzi następujące składowe:

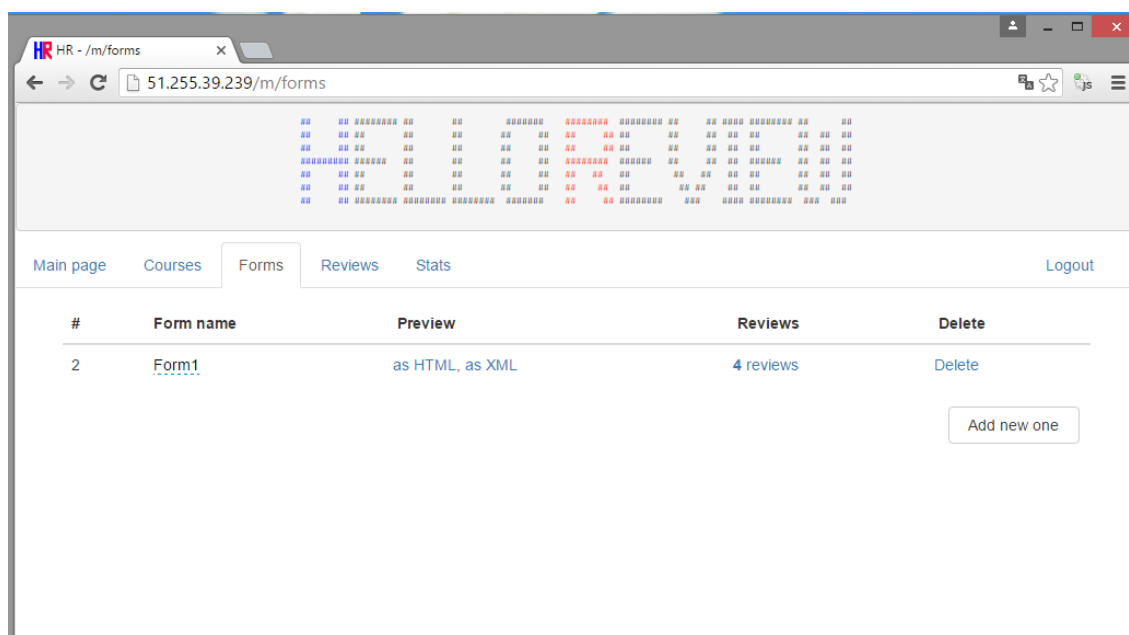
- opis (ang. description) - ogólny informacja dla studenta na temat zadania, które ma aktualnie wykonać (przegląd i ocena). Opis powinien zawierać dwa znaczniki specjalne {project} i {url}. Pierwszy z nich zostanie w ostatecznej wersji zastąpiony nazwą projektu (tj. repozytorium), którego ocena następuje. Drugi natomiast zostanie zamieniony na adres do anonimowej kopii, gdzie kursant może się zapoznać z ocenianą pracą;
- pytania (ang. questions) - lista pytań, na które kursant musi odpowiedzieć.

Pytanie w formularzu składa się z następujących składowych:

- tekst pytania (ang. question text) - treść pytania;
- dodatkowe wskazówki (ang. additional tips) - dodatkowa treść umieszczana zaraz pod treścią główną pytania. Jej przeznaczeniem jest przekazanie kursantowi dodatkowych wskazówek (np. na co powinien zwrócić szczególną uwagę), czy też dowolna inna treść uzupełniająca pytanie. Element ten nie jest obowiązkowy i nie musi być w pytaniu wykorzystywany;
- kontrolki (ang. inputs) - pola, które służą do wprowadzenia odpowiedzi na pytanie przed kursanta. System obecnie obsługuje dwa rodzaje kontrolek. Kontrolka typu tekst (ang. text) pozwala na wprowadzenie dowolnej treści tekstowej do 10 000 znaków. Kontrolka typu skala (ang. scale) pozwala wyrazić ocenę w skali liczbowej. Zakres skali, oraz jej znaczenie semantyczne jest bez znaczenia w sensie technicznym. Pytanie może składać się z wielu kontrolek - brak jest ograniczeń w tym względzie. Podczas generowania formularza ich kolejność zostanie zachowana. Kontrolka może zostać oznaczona jako obowiązkowa do wypełnienia lub nie. Z powodu ograniczeń technicznych systemu parametr ten ma zastosowanie tylko dla kontrolki typu tekst. Wypełnienie kontrolki typu skala zawsze jest obowiązkowe.

Formularz do systemu wprowadza się poprzez plik XML o odpowiedniej strukturze. Struktura ta jest sprawdzana pod kątem obecności wszystkich obowiązkowych składowych i poprawności składniowej. System pozwala na wyświetlenie struktury XML już zdefiniowanych formularzy. Możliwy jest także podgląd wyglądu formularza po jego wygenerowaniu. Wyświetlony formularz jest identyczny z tym, który zostanie wyświetlony kursantowi. Rysunki o numerach 8.9-8.11 zawierają zrzuty ekranowe z procesu dodawania formularza do systemu.

Plik XML podany przez użytkownika jest przetwarzany na klasę „Form”, będąca jednocześnie encją. Za tą formę „deserializacji” odpowiada biblioteka „Jackson”. Następnie system sprawdza, czy przekazany formularz jest prawidłowy. Kontrola ta polega na upewnieniu się, że wszystkie wymagane pola zostały uzupełnione i nie są puste. Za proces walidacji odpowiada zastosowana biblioteka „Hibernate”, a dokładnie moduł który nazywa się „Hibernate Validator”. W przypadku niepowodzenia walidacji prowadzący otrzymuje odpowiednie szczegóły, niezbędne do jego poprawienia. Jeżeli kod formularza okazał się być prawidłowy encja zostaje zapisana w bazie danych i podgląd formularza może zostać wyświetlony użytkownikowi.



Rysunek 8.9. Zrzut ekranowy: lista formularzy w systemie

Proszę ocenić pracę z projektu **user/repo1\_1**  
Praca jest dostępna pod adresem <https://github.com/user1/84cc4672-c4fd-46dd-ab8a-378323cfce19>

Odpowiedź na wszystkie pytania jest obowiązkowa.  
Obowiązkowe pola w pytaniu są oznaczone znakiem \*.

**Czy poprawnie sformatowano kod?**

Zwróć uwagę, czy zgadzają się wcięcia.  
Func ullamcorper odio non leo bibendum, vitae aliquet lectus varius.Praesent viverra leo tellus, in eleifend enim elementum eget. Nunc ac euismod elit.In ornare quam nisi, sed elementum risus dapibus quis. Vestibulum vel quam condimentum,elementum risus et, euismod arcu.

Oceń w skali \*

Źle

Prawidłowo

Dodatkowy komentarz

0 / 1000

**Czy zastosowano wzorzec fabryki w klasie SomeClass?**

Oceń w skali \*

Nie

Tak

Dodatkowy komentarz \*

0 / 1000

Submit

Rysunek 8.10. Zrzut ekranowy: podgląd formularza po jego wygenerowaniu

```

▼<form>
  ▼<description>
    Proszę ocenić pracę z projektu <strong>{project}</strong><br/> Praca jest dostępna pod adresem <strong style="font-size: 0.99em"><a
href={url}>{url}</a></strong><br/> <br/> Odpowiedź na wszystkie pytania jest obowiązkowa.<br/> Obowiązkowe pola w pytaniu są oznaczone
znakiem *.
  </description>
  ▼<question>
    <questionText>Czy poprawnie sformatowano kod?</questionText>
    ▼<additionalTips>
      Zwróć uwagę, czy zgadzają się wcięcia.<br/> Func ullamcorper odio non leo bibendum, vitae aliquet lectus varius.Praesent viverra leo
tellus, in eleifend enim elementum eget. Nunc ac euismod elit.In ornare quam nisi, sed elementum risus dapibus quis. Vestibulum vel
quam condimentum,elementum risus et, euismod arcu.
    </additionalTips>
    ▼<input type="scale">
      <required>true</required>
      <label>Oceń w skali</label>
      <fromLabel>Źle</fromLabel>
      <fromS>0</fromS>
      <toLabel>Prawidłowo</toLabel>
      <toS>10</toS>
    </input>
    ▼<input type="text">
      <required>false</required>
      <label>Dodatkowy komentarz</label>
    </input>
  </question>
  ▼<question>
    ▼<questionText>
      Czy zastosowano wzorzec fabryki w klasie SomeClass?
    </questionText>
    <additionalTips/>
    ▼<input type="scale">
      <required>true</required>
      <label>Oceń w skali</label>
      <fromLabel>Nie</fromLabel>
      <fromS>0</fromS>
      <toLabel>Tak</toLabel>
      <toS>1</toS>
    </input>
    ▼<input type="text">
      <required>true</required>
      <label>Dodatkowy komentarz</label>
    </input>
  </question>
</form>

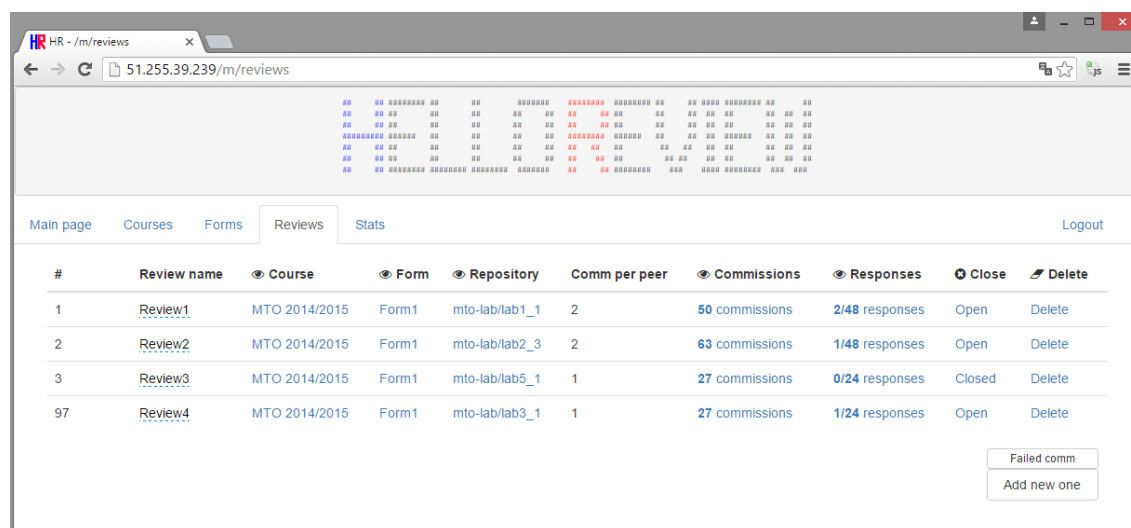
```

Rysunek 8.11. Zrzut ekranowy: przykładowy plik XML z definicją formularza

### 8.5.2. Zlecenie przeglądu

Rysunek numer 8.12 zawiera zrzut ekranowy z listą przeglądów w systemie. Lista jest przedstawiona w sposób tabelaryczny i zawiera następujące informacje:

- nazwa przeglądu (ang. review name);
- odnośnik do kursu (ang. course) związanego z przeglądem;
- odnośnik do formularza (ang. form) związanego z przeglądem;
- odnośnik ocenianego projektu (repozytorium; ang. repository) związanego z przeglądem;
- liczba zleceń oceny na osobę (ang. commissions per peer);
- odnośnik do statusu zleceń (ang. commissions);
- odnośnik do odpowiedzi (ang. responses);
- status zamknięcia (ang. close state).



#	Review name	Course	Form	Repository	Comm per peer	Commissions	Responses	Close	Delete
1	Review1	MTO 2014/2015	Form1	mto-lab/lab1_1	2	50 commissions	2/48 responses	Open	Delete
2	Review2	MTO 2014/2015	Form1	mto-lab/lab2_3	2	63 commissions	1/48 responses	Open	Delete
3	Review3	MTO 2014/2015	Form1	mto-lab/lab5_1	1	27 commissions	0/24 responses	Closed	Delete
97	Review4	MTO 2014/2015	Form1	mto-lab/lab3_1	1	27 commissions	1/24 responses	Open	Delete

Failed comm  
Add new one

Rysunek 8.12. Zrzut ekranowy: lista przeglądów w systemie

Pod listą stworzonych już przeglądów, zgodnie z zastosowaną konwencją, znajduje się przycisk do formularza, za pomocą którego możliwe jest dodanie nowego przeglądu. Wygląd formularza przedstawia zrzut ekranowy na rysunku 8.13.

Formularz dodawania przeglądu wymaga podania następujących informacji:

- nazwa przeglądu (ang. review name) - unikatowa nazwa nowo tworzonego przeglądu;
- liczba zleceń na kursanta (ang. number of commissions per participant) - określa ile prac ma ocenić każdy z kursantów. Przydział prac do oceniania jest losowy. Nie ma jednak możliwości, by kursant dostał dwa tą samą pracę do ocenienia. Nie ma też możliwości ręcznego dokonania przydziału;
- kurs związany z przeglądem (ang. course) - należy wskazać z listy kurs, dla którego zostanie utworzony przegląd;

- formularz związany z przeglądem (ang. form) - należy wskazać z listy formularz, który zostanie użyty do oceny;
- repozytorium projektu związanego z przeglądem (ang. repository) - należy wskazać repozytorium projektu, którego rozwiązania kursanci mają dokonać oceny.

Przegląd jest pojęciem ogólnym i dotyczy zadania w ramach grupy - prowadzący zlecił dokonanie przeglądu danego zadania, w danej grupie. Przegląd składa się ze zleceń. Zlecenie jest to pojedyncze zadanie ocenienia pracy autorstwa A przez osobę B.

Po potwierdzeniu chęci wykonania przeglądu poprzez przycisk „stwórz przegląd” (ang. create review) system dokona sprawdzenia, czy wszyscy kursanci prawidłowo wykonali powielenie projektu. Policzy także efektywną liczbę zleceń na kursanta. Jeżeli podana w formularzu liczba nie jest możliwa do osiągnięcia, zostanie ona obniżona do najbliższej możliwej wartości. Zrzut ekranowy na rysunku 8.13 przedstawia widok z wyświetlonymi informacjami wstępnymi. Jeżeli proces tworzenia przeglądu zostanie potwierdzony, system wykona tą czynność, a następnie uruchomi algorytm anonimizacji prac.

System umożliwia zamknięcie przeglądu. Zamknięcie przeglądu powoduje, że użytkownik nie będzie mógł już udzielić odpowiedzi. Anonimowe kopie związane z przeglądem zostają oznaczone jako niepotrzebne, i usunięte w trakcie najbliższego procesu czyszczenia.

Rysunek 8.13. Zrzut ekranowy: formularz dodawania przeglądu



Rysunki numer 8.14 oraz 8.15 zawierają listing kodu źródłowego. Są to dwie części kodu z klasy „GHReviewCreator”, która odpowiada za proces przygotowania przeglądu i zarejestrowania wszystkich zleceń do wykonania. Pierwszy z nich zawiera fragment do momentu sprawdzenia, ile osób nie wywiązało się z zadania i wyświetlenia tej informacji prowadzącemu, celem uzyskania potwierdzenia chęci kontynuacji. Po potwierdzeniu zastosowanie znajduje kod zamieszczony na drugim listingu. Funkcję, której listing zamieszczono wywołuje kontroler, odpowiedzialny za podstronę zlecenia przeglądów. Przekazywany do niej zostaje wypełniony formularz (obiekt typu „GHReviewAddForm”), repozytorium z zadaniem (obiekt typu „GHRepository”) oraz obiekt umożliwiający zmianę stanu wysyłanej odpowiedzi HTTP („HttpServletResponse”). Funkcja zwraca numer nowo utworzonego przeglądu.

```

74  /**
75  * Create review.
76  *
77  * @param ghReviewAddForm GHReviewAddForm ModelAttribute, must be already validated
78  * @param ghRepository reviewed repository
79  * @param response HttpServletResponse from mapping
80  * @return singleton list with id number of created review
81  * @throws LocalizableErrorException if any processing error occurred
82  */
83  public long createReview(GHReviewAddForm ghReviewAddForm,
84                          GHRepository ghRepository,
85                          HttpServletResponse response) throws LocalizableErrorException {
86
87      try {
88          // get course, list of participants in course, and used form in review
89          Course course = repositoryProvider.course().getOne(ghReviewAddForm.getCourseIDLong());
90          List<Participant> participants = course.getParticipants();
91          Form form = repositoryProvider.form().getOne(ghReviewAddForm.getFormIDLong());
92
93          // get map of forks (GitHub username -> GHRepository)
94          List<GHRepository> forks = GHExecutor.ex() -> ghRepository.listForks().asList();
95          Map<String, GHRepository> forksMap = new TreeMap<>(String.CASE_INSENSITIVE_ORDER);
96          forks.forEach(fork -> forksMap.put(fork.getOwnerName(), fork));
97
98          // get list of participants who forked reviewed project
99          List<Participant> partiWhoForked = participants.stream()
100              .filter(p -> forksMap.containsKey(p.getGitHubName()))
101              .collect(Collectors.toList());
102
103          // get list of participants who not forked reviewed project
104          List<Participant> partiWhoNotForked = participants.stream()
105              .filter(p -> !forksMap.containsKey(p.getGitHubName()))
106              .collect(Collectors.toList());
107
108          // calculate effective number of "responses per peer" - check if given number is proper
109          // responses per peer is number of review task for each peer
110          long respPerPeerVsForks = Math.min((long) partiWhoForked.size() - 1L, ghReviewAddForm.getRespPerPeerLong());
111          long respPerPeer2 = Math.max(respPerPeerVsForks, 0L);
112
113          // if it is first step of review creation do not create review
114          // first return info what will be really done
115          // if information was confirmed continue
116          if (ghReviewAddForm.getIgnoreWarning() == 0L) {
117              response.setStatus(HttpStatus.PRECONDITION_FAILED.value());
118
119              List<String> warning = new ArrayList<>(10);
120              warning.add(String.valueOf(respPerPeer2));
121              warning.add(String.valueOf(partiWhoNotForked.size()));
122              warning.add(String.valueOf(participants.size()));
123              partiWhoNotForked.forEach(p -> warning.add(p.getName()));
124              return warning;
125          }
126      }

```

Rysunek 8.14. Listing kodu źródłowego: tworzenie przeglądu, klasa GHReviewCreator (część 1)

```

127 // get list of "assessed" participants
128 // it is list of participants who forked project, multiplied by number of "responses per peer"
129 // and shuffled, as every participant should got random review task (commission)
130 List<Participant> mulParticipants = new ArrayList<>(10);
131 long expectedMulParSize = (participants.size() * respPerPeer2);
132 while (mulParticipants.size() < expectedMulParSize) {
133     mulParticipants.addAll(partiWhoForked);
134 }
135 Collections.shuffle(mulParticipants);
136 mulParticipants.addAll(partiWhoForked); // to be sure, that is enough
137
138 // create review model
139 Review review = new Review(
140     ghReviewAddForm.getName(), respPerPeer2, course, form, ghReviewAddForm.getRepositoryFullName()
141 );
142 Collection<Commission> responses = new ArrayList<>(10);
143
144 // add to review model info about participants who not forked project
145 for (Participant participant : partiWhoNotForked) {
146     Commission rResponse = new Commission(review, participant, null, (String) null);
147     rResponse.setStatus(CommissionStatus.NOT_FORKED);
148     responses.add(rResponse);
149 }
150
151 // list of already assessed participants,
152 // to be ensure that one assessor will not have two same review task (commission)
153 Collection<Participant> assessedCollection = new LinkedList<>();
154
155 // add commissions for each participant
156 for (Participant assessor : course.getParticipants()) {
157     assessedCollection.clear();
158
159     // each participant got "respPerPeer2" number of commissions
160     for (long lo = 0L; lo < respPerPeer2; lo++) {
161
162         Participant assessed = popUnique(mulParticipants, assessor, assessedCollection);
163         assessedCollection.add(assessed);
164
165         GHRepository assessedRepo = forksMap.get(assessed.getGitHubName());
166
167         Commission rResponse = new Commission(review, assessed, assessor, assessedRepo.getHtmlUrl());
168         responses.add(rResponse);
169     }
170 }
171
172 // finally save review to database
173 repositoryProvider.review().save(review);
174 repositoryProvider.commission().save(responses);
175
176 // and register github clone task, to provide anonymous copy
177 responses.stream().filter(r -> r.getStatus() != CommissionStatus.NOT_FORKED)
178     .forEach(ghTaskScheduler::registerClone);
179
180 return review.getId();
181
182 } catch (GHCommunicationException e) {
183     throw (LocalizableErrorRestException)
184         new LocalizableErrorRestException("NoGitHub", e.toString()).initCause(e);
185 }
186
187 private <T> T popUnique(Collection<T> collection, T meExclude, Collection<T> excludeCollection) {
188     for (T collElement : collection) {
189         if (!meExclude.equals(collElement) && !excludeCollection.contains(collElement)) {
190             collection.remove(collElement);
191             return collElement;
192         }
193     }
194
195     throw new InternalServerErrorException();
196 }
197
198 }
199
200 }

```

Rysunek 8.15. Listing kodu źródłowego: tworzenie przeglądu, klasa GHReviewCreator (część 2)

### 8.5.3. Algorytm anonimizacji prac

Z momentem utworzenia przeglądu system uruchomił algorytm odpowiedzialny za anonimizację prac. W tym momencie lista zleceń jest już znana, zostały ustalone adresy repozytoriów źródłowych - było to niezbędne już na wcześniejszym etapie. Dla każdego zlecenia system dokonuje kopiowania pracy. Za skopiowanie pracy jest odpowiedzialna klasa GHTaskClone. Proces kopiowania przebiega zgodnie ze schematem:

1. system zgłasza zlecenie: Kursant1 ma ocenić pracę Kursant2;
2. system nadaje zleceniu indywidualny unikalny ID, zgodny ze standardem UUID4;
3. system zakłada repozytorium zdalne na koncie „manekin” (ang. dummy), którego token został podany w pliku konfiguracyjnym, o nazwie takiej, jak nadany ciąg ID;
4. system sprawdza listę gałęzi (ang. branch) w kopiowanym repozytorium;
5. system kopiuje każdą ze znalezionych gałęzi:
  - a) system pobiera do folderu tymczasowego (zdefiniowanego znacznikiem „tempdir” w pliku konfiguracyjnym) wybraną gałąź z repozytorium źródłowego, na tym folderze wykonuje kolejne akcje;
  - b) system usuwa ukryty folder „.git” - projekt przestaje być repozytorium Git;
  - c) system inicjalizuje nowe puste repozytorium lokalne;
  - d) system konfiguruje nowe repozytorium - nazwa użytkownika i adres e-mail zostają ustawione na takie, które widnieją w konfiguracji konta „dummy” na platformie GitHub;
  - e) system zapisuje zmiany (w tym przypadku: wszystkie istniejące pliki) w repozytorium lokalnym (akcja git: commit), treścią używaną do opisu zmian jest ciąg „commitmsg” ustalony w pliku konfiguracyjnym;
  - f) system wysyła skopiowaną lokalnie gałąź do repozytorium zdalnego (git push);
6. system sprawdza jaką gałąź w repozytorium źródłowym jest ustawiona jako domyślna (ang. default), takiego samego ustawienia dokonuje na kopii;
7. system zapisuje adres utworzonej kopii w bazie danych - to ten adres zostanie przekazany recenzentowi.

Jeżeli proces kopiowania nie powiedzie się, na jakimkolwiek etapie, zmiany są wycofywane. W bazie danych zostaje zapisana odpowiednia informacja o porażce. Niepowodzenie może być spowodowane wieloma czynnikami. Najprostszą przyczyną jest niedostępność jednej z usług platformy GitHub. Inną możliwą przyczyną jest niedoskonałość implementacji algorytmu, który zawiódł z bliżej nieokreślonej przyczyny - podczas testów przypadek taki został zanotowany dla około pół procent zleceń. Autor po konsultacji z opiekunem pracy ustalił, że zbadana niezawodność jest wystarczająca. Proces

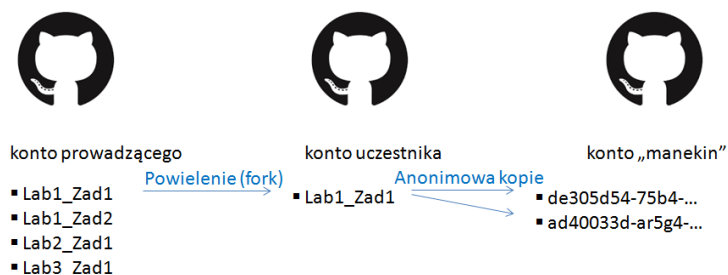
ponowienia próby kopiowania nie jest dokonywany automatycznie, należy go uruchomić manualnie. System przewiduje odpowiednią opcję dostępną w podglądzie stanów zleceń.

„Esencję” algorytmu anonimizacji zawiera obrazek 8.17 na stronie 45. Jest to listing funkcji „run” z klasy „GHTaskClone”. Wspominana klasa reprezentuje zadanie anonimizacji prac i wykonuje kolejne czynności zgodnie z zaprezentowanym schematem.

Jak wspomniano, w systemie istnieje możliwość podglądu stanów zleceń oceny w ramach przeglądu. Podgląd ten został zaprezentowany na zrzucie ekranowym na obrazku numer 8.18 na stronie 46. Widniejący tam status może przyjąć następujące wartości:

- projekt nie powielony (ang. project not forked) - informacja dodatkowa, wpis taki jest tworzony dla każdego kursanta, którego rozwiązanie nie zostało znalezione. W sposób naturalny praca takiego kursanta nie może zostać oceniona. Sam kursant jednak oczywiście dostaje do oceny inne prace;
- przetwarzanie zakończone porażką (ang. processing failed) - proces kopiowania zakończył się niepowodzeniem. W przypadku niepowodzenia obok zlecenia staje się dostępny przycisk pozwalający ponowić próbę (ang. retry) anonimizacji dla tej pracy;
- w trakcie przetwarzania (ang. processing) - anonimizacja dla tego zlecenia została zaplanowana, lub jest w trakcie przetwarzania. System kolejkuje zlecenia, wykonując jednocześnie nie więcej niż trzy kopie.
- ocena nie wypełniona (ang. unfilled) - proces anonimizacji zakończył się powodzeniem i formularz oceny jest dostępny dla kursanta. Kursant nie dokonał jeszcze oceny;
- ocena wypełniona (ang. filled) - kursant dokonał oceny.

Jak już zostało wyjaśnione kopie są składowane na specjalnie do tego przeznaczonym koncie. Obrazek 8.16 przedstawia schematycznie obieg repozytorium. Zrzut ekranowy na obrazku 8.19 na stronie 46 przedstawia natomiast przykładową listę repozytoriów widoczną na takim koncie.



Rysunek 8.16. Repozytorium z zadaniem i jego powielenia nad kontach w usłudze GitHub

```

102  @Override
103  @Transactional
104  public void run() {
105      LOGGER.debug("{} Repo cloning started.", uuid);
106
107      try {
108          init();
109
110          deleteDirForCloneIfExist(false);
111          deleteTargetRepoIfExist();
112          getListOfBranchesOfAssessedRepo();
113          createTargetRepo();
114
115          for (String branch : assessedRepoBranches) {
116              LOGGER.debug("{} Processing branch: {}", uuid, branch);
117
118              createDirForClone();
119              cloneAssessedRepo(branch);
120              removeGitSubdirInClone();
121
122              LOGGER.debug("{} Init-ing clone dir as new repo.", uuid);
123
124              try (Git gitTarget = Git.init().setDirectory(directoryForClone).call()) {
125
126                  LOGGER.debug("{} Init-ed clone dir as new repo.", uuid);
127
128                  addAllFilesToCommit(gitTarget);
129                  setRepoConfiguration(gitTarget);
130                  commitFiles(gitTarget);
131                  createBranchIfRequired(gitTarget, branch);
132                  pushChangesIntoTargetRepo(gitTarget);
133                  awaitUntilPushIsCompleted(branch);
134              }
135
136              deleteDirForCloneIfExist(true);
137          }
138
139          setDefaultBranchSameAsInAssessed();
140
141          LOGGER.debug("{} Repo cloning done. Updating commision status.", uuid);
142
143          commission.setStatus(CommissionStatus.UNFILLED);
144          commission.setGhUrl(targetRepo.getHtmlUrl().toString());
145          commissionRepository.save(commission);
146
147          LOGGER.debug("{} Repo cloning done. Updated commision status.", uuid);
148          LOGGER.info("{} Done.", uuid);
149
150      } catch (GitAPIException | GHCommunicationException | IOException | RuntimeException e) {
151          LOGGER.info("{} Repo cloning failed.", uuid, e);
152
153          LOGGER.debug("{} Cleaning. Updating commision status.", uuid);
154          commission.setStatus(CommissionStatus.PROCESSING_FAILED);
155          commissionRepository.save(commission);
156
157          LOGGER.debug("{} Cleaning. Deleting target repo if exist.", uuid);
158          ExceptionUtils.ignoreException2(() ->
159              GHExecutor.ex(() -> ghWait.getRepository(targetRepoFullName).delete())
160          );
161
162          LOGGER.debug("{} Cleaning. Deleting clone dir if exist.", uuid);
163          ExceptionUtils.ignoreException2(() ->
164              deleteDirForCloneIfExist(false)
165          );
166
167          LOGGER.debug("{} Cleaning done. ", uuid);
168          LOGGER.info("{} Done.", uuid);
169      }
170  }
171

```

Rysunek 8.17. Listing kodu źródłowego: anonimizacja prac, kopiowanie repozytorium

#	Review	Assessor	Assessed	Status	Retry
257	Review 21		grzego699	Project not forked	
258	Review 21		180132	Project not forked	
259	Review 21	Adrian Pędziwiatr	astepniewski	Unfilled	
260	Review 21	Adrian Pędziwiatr	Dastipio	Unfilled	
261	Review 21	Adrian Pędziwiatr	dzemba	Unfilled	
262	Review 21	astepniewski	jarkos	Processing by server	
263	Review 21	astepniewski	Hirikiawashi	Processing by server	
264	Review 21	astepniewski	grzego69	Processing by server	
265	Review 21	bilheris	matihuf	Processing by server	
266	Review 21	bilheris	jarkos	Processing by server	
267	Review 21	bilheris	Adrian Pędziwiatr	Processing by server	
268	Review 21	damianspinek	bilheris	Processing by server	

Rysunek 8.18. Zrzut ekranowy: status zleceń w przeglądarce

peerreview70 (Peer Review) X

GitHub, Inc. [US] https://github.com/peerreview70?tab=repositories

Search GitHub Pull requests Issues Gist

Contributions Repositories Public activity Edit profile

Find a repository... Search All Public Private Sources Forks Mirrors New

**Peer Review 70**  
peerreview70

Lodz University of Technology  
Lodz  
http://weeia.p.lodz.pl  
Joined on 9 Oct 2015

0 Followers 0 Starred 0 Following

**3e117e17-4255-4407-aa81-fac3b3629f05** Java ★ 0 0  
clone for anonymous peer review purposes  
Updated 43 minutes ago

**b77e6575-a88d-4683-a844-e00ff5e0b621** Java ★ 0 0  
clone for anonymous peer review purposes  
Updated 43 minutes ago

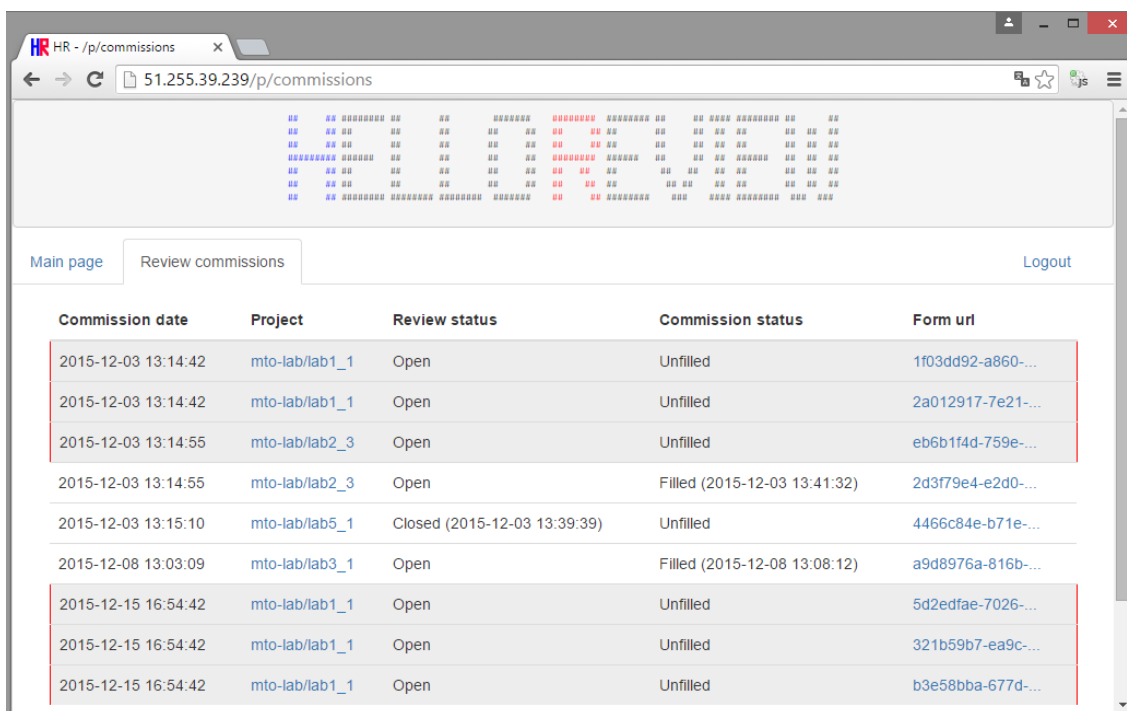
**8a6c33b0-6850-4db5-b55c-0bd13961554d** Java ★ 0 0  
clone for anonymous peer review purposes  
Updated 43 minutes ago

**989c3576-0f9f-416f-a664-f26f4cce9c5f** Java ★ 0 0  
clone for anonymous peer review purposes  
Updated 43 minutes ago

Rysunek 8.19. Zrzut ekranowy: anonimowe kopie zleceń w usłudze GitHub

## 8.6. Kursant dokonuje oceny

Kursant po zalogowaniu do systemu widzi listę zleceń. Rysunek ze zrzutem ekranowym numer 8.20 przedstawia przykładową listę. Jeżeli zlecenie zostało już poprawnie przetworzone przez algorytm anonimizujący, przegląd nie jest zamknięty a odpowiedź jeszcze nie udzielona użytkownik może wykonać przydzielone mu zadanie - dokonać przeglądu i oceny pracy. Zlecenia takie zostały wyróżnione na liście szarym kolorem, oraz czerwoną lewą i prawą krawędzią tabeli. Adres do formularza oceny znajduje się jako ostatnia pozycja na liście. Adres do formularza, sam formularz, ani żadna inna informacja nie ujawnia autora ocenianego rozwiązania. Kursant zna tylko indywidualny numer zlecenia, który nie zdradza tej informacji, ani nie daje się z niczym powiązać (między innymi z uwagi na zastosowaną losowość ID).



Commission date	Project	Review status	Commission status	Form url
2015-12-03 13:14:42	mto-lab/lab1_1	Open	Unfilled	1f03dd92-a860-...
2015-12-03 13:14:42	mto-lab/lab1_1	Open	Unfilled	2a012917-7e21-...
2015-12-03 13:14:55	mto-lab/lab2_3	Open	Unfilled	eb6b1f4d-759e-...
2015-12-03 13:14:55	mto-lab/lab2_3	Open	Filled (2015-12-03 13:41:32)	2d3f79e4-e2d0-...
2015-12-03 13:15:10	mto-lab/lab5_1	Closed (2015-12-03 13:39:39)	Unfilled	4466c84e-b71e-...
2015-12-08 13:03:09	mto-lab/lab3_1	Open	Filled (2015-12-08 13:08:12)	a9d8976a-816b-...
2015-12-15 16:54:42	mto-lab/lab1_1	Open	Unfilled	5d2edfae-7026-...
2015-12-15 16:54:42	mto-lab/lab1_1	Open	Unfilled	321b59b7-ea9c-...
2015-12-15 16:54:42	mto-lab/lab1_1	Open	Unfilled	b3e58bba-677d-...

Rysunek 8.20. Zrzut ekranowy: lista zleceń na koncie kursanta

Formularz wyświetlony użytkownikowi, jest identyczny z tym, który prowadzący widział na podglądzie. Obowiązkowe kontrolki zostały oznaczone znakiem gwiazdki. Zrzut ekranowy na rysunku numer 8.10 na stronie 38 przedstawia przykładowy formularz. Wyświetlenie formularza, na który już dokonano odpowiedzi skutkuje wyświetleniem formularza, na którym wszystkie kontrolki są w trybie tylko do odczytu - użytkownik może się zapoznać z dokonanymi odpowiedziami, ale nie ma już możliwości ich modyfikacji. Próba wyświetlenia formularza, związanego z jeszcze nie przetworzonym zleceniem powoduje wyświetlenie komunikatu ze stosowną informacją - „Sorry, this form is not

ready yet, it is not fully processed by server. Please wait 30 minutes, then contact the course master”.

Kursant ma czas na dokonanie do czasu zamknięcia przeglądu. Zgodnie z zastosowanym założeniem, po jego zamknięciu może on wyświetlić dokonane odpowiedzi, ale adresy do anonimowych kopii mogą być już nieaktywne, gdyż jako niepotrzebne mogły zostać już usunięte.

## 8.7. Prowadzący sprawdza odpowiedzi

Prowadzący posiada dwa źródła informacji o stanie odpowiedzi. Pierwszym jest status zleceń, przedstawiony już na rysunku 8.18 na stronie 46. W tym miejscu jest jednak tylko ogólna informacja kto dokonał oceny, a kto jeszcze nie.

Ponad to lista przeglądów (rysunek nr 8.12 na stronie 39) podsumowuje tę informację liczbowo. Wspominana informacja liczbowo jest odnośnikiem do szczegółów odpowiedzi. Nie ma możliwości podejrzenia ich poprzez podstronę systemu. Odnośnik ten prowadzi do generowanego dynamicznie arkusza Microsoft Excel gdzie znajdują się: stempel czasowy wygenerowania przeglądu, podsumowanie przeglądu, oraz wszystkie udzielone odpowiedzi. Prezentacja odpowiedzi w arkuszu Excel pozwala w łatwy sposób je filtrować i przetwarzać. Przykład wygenerowanego arkusza zawiera rysunek nr 8.21.

Prowadzący jest jedyną osobą, która zna odpowiedzi. W jego gestii leży sposób ich wykorzystania. Może je uwzględnić wystawiając ocenę, przekazać grupie ogólne podsumowanie, każdemu z osobna osobną informację lub w wykorzystać w jakikolwiek inny sposób, który uzna za słuszny.

	A	B	C	D	E	F	G	H
1	Generation timestamp	2015-12-15 20:41:56						
2								
3	Name	Review 22						
4	Created	2015-12-15 18:09:04						
5	Closed							
6	Course	Course1						
7	Form	Form1						
8	Repository	mto-lab/lab1_1						
9	Resp per peer	1						
10								
11	Timestamp	Assessor	Assessed	Assessed GH url	Status	Czy poprawnie sformatowano kod? Oceń w skali (0 - 10)	Czy poprawnie sformatowano kod? Dodatkowy komentarz	Czy zastosow wzorzec fabryki SomeClass? Oce (0 - 1)
12		Stępniewski Arkadiusz	Pędziwiatr Adrian	https://github.com/1802	UNFILLED			
13	2015-12-15 18:27:32	Pędziwiatr Adrian	Stępniewski Arkadiusz	https://github.com/astep	FILLED		8 Kod co do zasady sformat	
14								
15								
16								
17								
18								
19								
20								
21								
22								

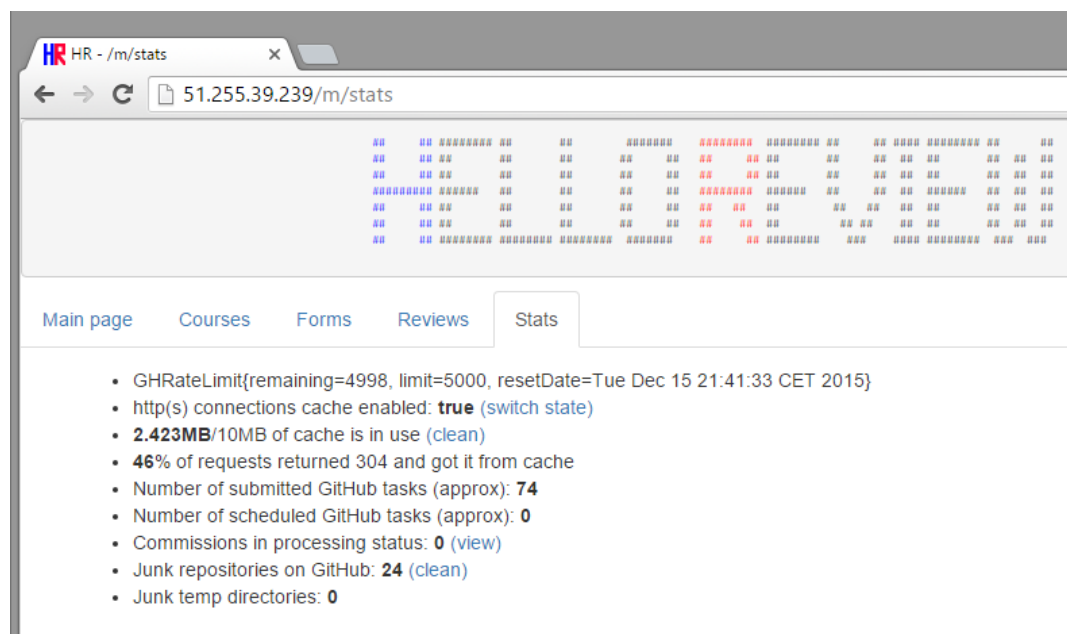
Rysunek 8.21. Zrzut ekranowy: arkusz MS Excel z podsumowaniem dokonanych ocen



## 8.8. Statystyki systemu i sprzątanie

System posiada jeszcze jedną podstronę kontrolno-sprzątającą zaprezentowaną na rysunku 8.22. Znajdują się na niej informacje statystyczne i techniczne (ściśle związane ze sposobem implementacji, mogą być pomocne przy rozwiązywaniu problemów). Wśród wypisywanych informacji są takie dane jak:

- wykorzystanie aktualnego limitu połączeń z GitHub API;
- stan pamięci podręcznej - czy jest ona włączona, oraz możliwość zmiany stanu;
- stan pamięci podręcznej - ilość wykorzystanego miejsca, oraz możliwość skasowania zapamiętanych informacji;
- procentowy wskaźnik udziału pamięci podręcznej przy wykonywaniu żądań http;
- przybliżona liczba zgłoszonych zadań związanych z anonimizacją prac (dana techniczna);
- przybliżona liczba zadań, związanych z anonimizacją prac, oczekujących w kolejce do wykonania (dana techniczna);
- liczba zleceń, które mają w systemie status przetwarzanych;
- liczba repozytoriów (anonimowych kopii) na koncie „manekin” (dummy), które nie mają pokrycia w bazie danych i nie są potrzebne. Do tej liczby zaliczają się także kopie związane z zamkniętymi przeglądami. W tym miejscu istnieje możliwość usunięcia ich z konta;
- liczba folderów w katalogu tymczasowym (powinna być równa zero, jeżeli brak jest zadań w kolejce) (dana techniczna).



Rysunek 8.22. Zrzut ekranowy: statystyki systemowe

## 9. Baza danych

### 9.1. Wstęp

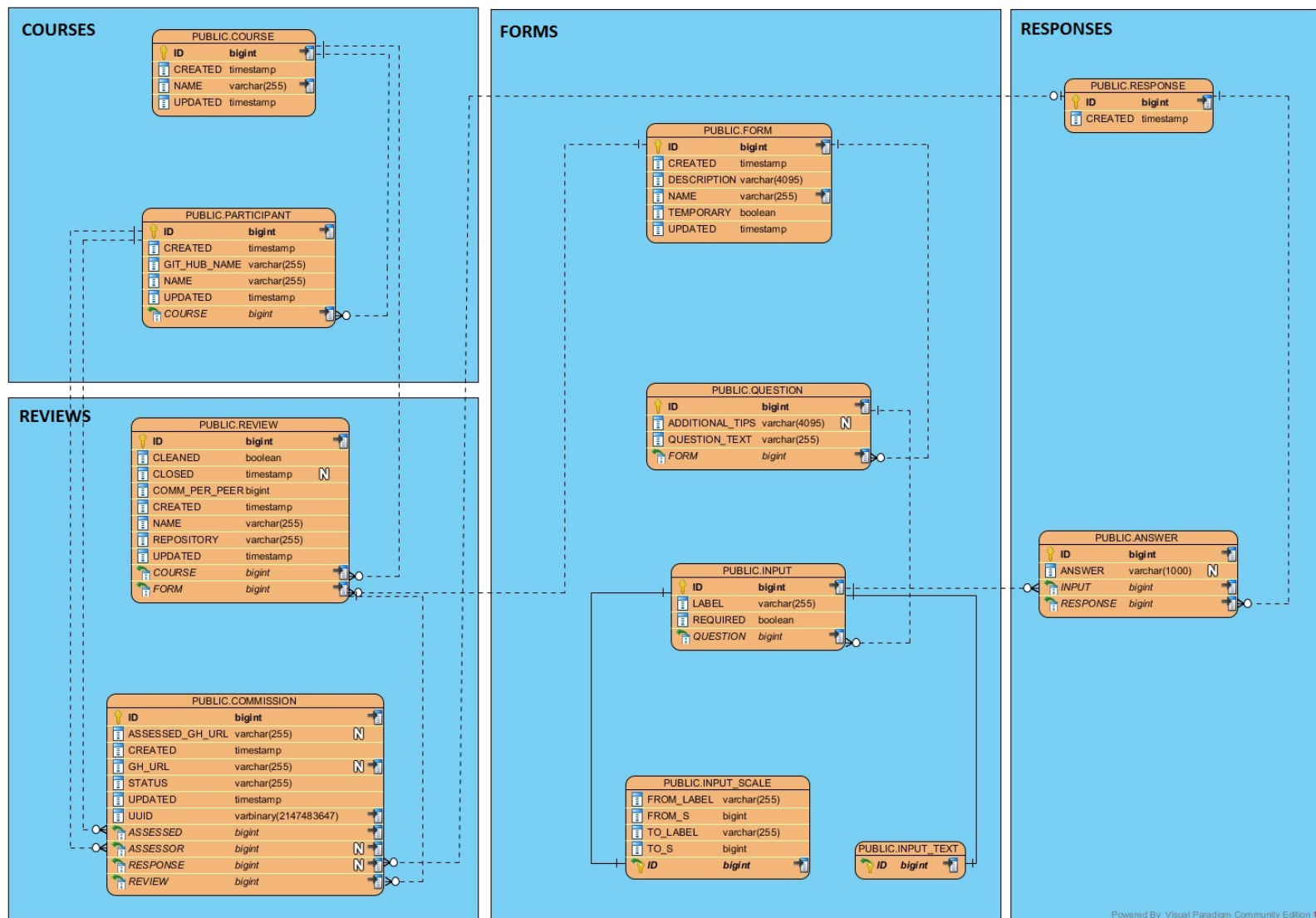
Dane wytwarzane przez system muszą być nieulotne. Z tego powodu istnieje potrzeba składowania ich na dysku twardym. System zapisuje dane w relacyjnej bazie danych zgodnej ze standardem SQL.

Wykorzystanym (i skonfigurowanym) systemem zarządzania relacyjną bazą danych (ang. Relational Database Management System, RDBMS) jest H2 w trybie wbudowanym (ang. embedded mode). Autor nie widzi potrzeby wykorzystania na potrzeby pracy inżynierskiej bazy danych w trybie klient-server (ang. client-server mode). Zarówno wykorzystywany system bazy danych, jak i tryb w którym jest ona uruchamiana są konfigurowalne poprzez plik „application.properties”. Jeżeli zaistnieje potrzeba zmiany systemu zarządzania relacyjną bazą danych można to zrobić. Jedynym wymogiem jest, by była to baza relacyjna zgodna ze standardem SQL.

Autor projektując bazę danych zastosował podejście „najpierw kod” (ang. code-first). Oznacza to, że autor zaprojektował bazę danych, jako zbiór encji (ang. entities) - klas języka Java zgodnych ze standardem mapowania relacyjno-obiektowego w tym języku (tj. Java Persistence API, JPA). Za utworzenie odpowiednich tabel bazodanych, na podstawie istniejących encji, odpowiedzialny jest zastosowany silnik mapowania relacyjno-obiektowego - implementacja JPA - Hibernate. Dla każdej encji wspomniany silnik utworzył odpowiadającą jej tabelę w bazie danych.

### 9.2. Schemat bazy danych

Rysunek 9.1 na stronie 51 przedstawia zastosowany w systemie schemat bazy danych. Rysunek ten zawiera tabele i istniejące pomiędzy nimi powiązania.



Rysunek 9.1. Schemat bazy danych

Autor podzielił model bazy danych na cztery części wzajemnie ze sobą powiązane:

1. kursy (ang. courses) - encje odpowiedzialne za kursy i uczestników;
2. formularze (ang. forms) - encje odpowiedzialne za system formularzy;
3. przeglądy (ang. review) - encje odpowiedzialne za przeglądy i zlecenia;
4. odpowiedzi (ang. responses) - encje odpowiedzialne za odpowiedzi na zlecenia przeglądów.

Cechą wspólną wszystkich encji jest to, że system zapisuje w atrybutach czas ich utworzenia („created”). Większość z nich zawiera także czas ostatniej aktualizacji („updated”). Czas aktualizacji nie jest notowany dla encji, które z założenia są niemodyfikowalne (ang. immutable) i nie będą zmieniane.

### **9.2.1. Kursy**

Kursy zostały opisane przy użyciu dwóch encji. Pierwsza z nich „course” odpowiada za przechowywanie informacji na temat utworzonych kursów - nadanej nazwy („name”),. Druga z encji to „participant” i przechowuje informacje o uczestnikach zapisanych do danego kursu - imię („name”), nazwa konta w usłudze GitHub („githubname”).

### **9.2.2. Formularze**

Każdy formularz posiada odpowiedni wpis w encji „form”. Formularz charakteryzuje się nazwą („name”). Zawiera także opis („description”), listę pytań („question”) i znacznikiem, czy jest to wpis tymczasowy („temporary”). Wpis tymczasowy to atrybut związany z zastosowaną implementacją podglądu formularza podczas jego tworzenia. Wpis oznaczony przez system jako tymczasowy nie może zostać użyty na innych etapach i zostanie automatycznie usunięty przy najbliższej okazji.

Pytania, które wchodzi w skład formularza posiadają takie atrybuty jak: treść pytania („question\_text”), opcjonalne informacje dodatkowe („additional\_tips”). Pytanie zawiera także listę kontroltek („input”), które mogą być typu skala („input\_scale”) lub tekst („input\_text”).

### **9.2.3. Przeglądy**

Encje związane z przeglądami to przegląd („review”) i zlecenie („commission”). Przegląd jest ogólnym pojęciem i jest związany z grupą. Prowadzący zleca przegląd w grupie. Przegląd może składać się z wielu zleceń. Zlecenie jest to pojedyncze zadanie oceny pracy autorstwa kursanta A przez innego kursanta B.

Przegląd zawiera takie atrybuty jak:

- nazwa („name”);
- nazwa źródłowego repozytorium z ocenianym zadaniem („repository”);
- liczba zleceń w ramach przeglądu przypadająca na jednego kursanta („comm\_per\_peer”);
- znacznik, czy przegląd został już zamknięty („closed”);
- odnośnik do zastosowanego formularza („form”);
- odnośnik do kursu dla którego przegląd został zlecony („course”).

Zlecenie zawiera następujące atrybuty:

- odnośnik do przeglądu, do którego zlecenie należy („review”);
- odnośnik do kursanta, którego praca jest oceniana („assessed”);
- odnośnik do kursanta, który ma dokonać oceny („assessor”);
- adres URL do repozytorium z ocenianą pracą („assessed\_gh\_url”);
- indywidualny numer przeglądu, zgodny ze standardem UUID4 („uuid”);
- status przeglądu („status”);
- odnośnik do udzielonej odpowiedzi („response”).

#### **9.2.4. Odpowiedzi**

Każde zlecenie zawiera odnośnik do udzielonej odpowiedzi. Za przechowywanie odpowiedzi odpowiadają encje „response” i „answer”. Pierwsza z nich nie zawiera szczególnych atrybutów - jest to jedynie informacja, że odpowiedź została udzielona. Jedynym atrybutem jest czas udzielenia odpowiedzi („created”).

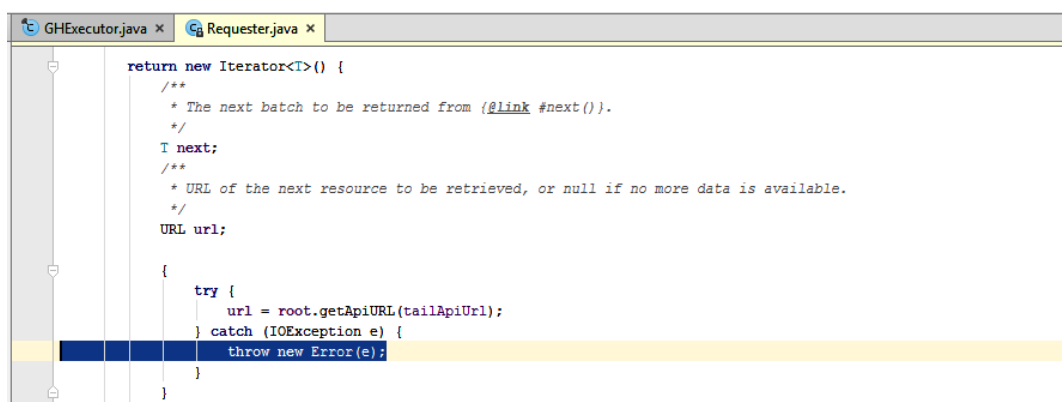
Dla każdej kontrolki, która była w formularzu przypisanym do zlecenia, utworzony zostaje wpis typu „answer”. Encja taka zawiera odnośnik do kontrolki, której dotyczy („input”), odpowiedź udzieloną przez kursanta („answer”), oraz odnośnik do encji „response”, która identyfikuje zlecenie.

## 10. Napotkane problemy w implementacji

Implementacja nigdy nie przebiega bezproblemowo. Rozdział ten przeznaczony został na opisanie najciekawszych napotkanych przez autora problemów w trakcie realizacji tytułowego systemu, wraz z genezą ich powstania i sposobem dotarcia do przyczyny lub zastępczego rozwiązania.

### 10.1. Wyjątki związane z połączeniem do GitHub API

Autor użył do komunikacji z GitHub API biblioteki autostwa Kohsuke Kawaguchi o nazwie `org.kohsuke:github-api`. W opinii autora reprezentacja obiektowa została w niej przystępnie zaprojektowana i jest ona prosta w obsłudze. Okazało się jednak, że dla żądań listujących (np. lista powieści repozytorium) w przypadku niepowodzenia rzuca ona błędem typu „Error”, zamiast wyjątkiem typu „Exception” lub „RuntimeException”. Sytuację taką przedstawia listing na rysunku 10.1. W ocenie autora użycie takiego typu błędu nie jest odpowiednim rozwiązaniem oraz jest niepożądane dla działania systemu. Autor jako rozwiązanie napisał dodatkową klasę pomocniczą „GHExecutor”, która przetwarza wyjątki rzucane przez tę bibliotekę, na własny „GHCommunicationException” i jednocześnie „zamienia” nieporządkane błędy w wyjątki. Listing napisanej klasy zawiera rysunek numer 10.2 na stronie 55. Wszystkie wywołania używanej biblioteki, które powodują połączenie z usługą GitHub zostały „opakowane” napisaną klasą użytkową.



```
return new Iterator<T>() {  
    /**  
     * The next batch to be returned from {@link #next()}.  
     */  
    T next;  
    /**  
     * URL of the next resource to be retrieved, or null if no more data is available.  
     */  
    URL url;  
  
    {  
        try {  
            url = root.getApiURL(tailApiUrl);  
        } catch (IOException e) {  
            throw new Error(e);  
        }  
    }  
}
```

Rysunek 10.1. Listing kodu źródłowego: nieoczekiwany błąd (Error) podczas połączeń z GitHub API

```

GHExecutor.java x  Requester.java x
/**
 * Additional wrapper for getting information from GitHub api using org.kohsuke.github.GitHub.<br/>
 * Unfortunately, it not only throws IOException, as documented.<br/>
 * Experimentally found out that IOException may be wrapper in Error.<br/>
 * This class mainly to unwrap it.<br/>
 * Additionally, exceptions are converted to internal GHCommunicationException exception.
 */
public final class GHExecutor {
    private GHExecutor() {
    }

    public static <T> T ex(GHRunner1<T> runner) throws GHCommunicationException {
        try {
            return runner.execute();
        } catch (IOException e) {
            throw new GHCommunicationException(e.getMessage(), e);

            // oh, it may be wrapped with Error ;/
        } catch (Error e) {
            if (e.getCause() instanceof IOException) {
                throw new GHCommunicationException(e.getMessage(), e);
            } else {
                throw new Error(e);
            }
        }
    }

    public static void ex(GHRunner2 runner) throws GHCommunicationException {
        GHExecutor.<.>ex(() -> {
            runner.execute();
            return null;
        });
    }
}

```

Rysunek 10.2. Listing kodu źródłowego: klasa GHCommunicationException

## 10.2. Pamięć podręczna dla połączeń z GitHub API

Połączeń do GitHub API nie można wykonywać dowolnie dużo. GitHub limituje liczbę połączeń na godzinę. Limit ten dla połączeń anonimowych wynosi 60, natomiast dla połączeń używających danych autoryzacyjnych 5000 [7]. Limit jest więc bardzo duży - wystarczy się przedstawić, a można wysyłać zapytania o zasoby częściej niż co sekundę. Teoretycznie można się nim nie przejmować limitem, i nie wykonywać żadnych optymalizacji w tym kierunku.

Warto jednak pamiętać, że wykorzystujemy darmową usługę, i chociażby w ramach podziękowania zastosować się do wskazówek w oficjalnej dokumentacji GitHub API. Znajduje się tam informacja, że jeżeli zapytania są wysyłane z użyciem mechanizmów pamięci podręcznej, to w ramach podziękowania w pewnych przypadkach wykonanie zapytania nie obniża limitu. W wielu odpowiedziach wysyłane są nagłówki HTTP „ETag” i/lub „Last-Modified”. Informacje te pozwalają wysłać żądanie warunkowe - „Podaj mi proszę zasób, tylko jeżeli się zmienił. W przypadku braku zmian użyję odpowiedzi zapisanej w pamięci podręcznej”. Jeżeli odpowiedzią na takie zapytanie warunkowe jest kod „304 Not Modified”, czyli „brak zmian”, to limit nie jest obniżany.

Podążając wskazówkami autor postanowił dostosować swój kod, tak, by z takiej pamięci podręcznej korzystał. W dokumentacji biblioteki wykorzystywanej implementacji dostępu do GitHub API w języku java - tj. „org.kohsuke: github-api”, znajduje się rekomendacja zmiany domyślnego agenta http, na bibliotekę „OkHttp”, która obsługuje mechanizm pamięci podręcznej. Tak też zrobiono. Włączenie mechanizmu pamięci podręcznej spowodowało jednak, że algorytm kopiowania przestał działać poprawnie - nagle bardzo spowolnił, ilość zapytań paradoksalnie nie zmalała, a wzrosła.

Zapisywane informacje z przebiegu działania algorytmu nie wskazywały jasno przyczyny. Ich analiza wykazała jednak, że przestój następował w momencie odpytywania GitHub API o informację, czy wysyłana gałąź w repozytorium już jest widoczna jako zasób, co terminuje iż jej wysyłanie zakończyło się. Autor wysnuł teorię, że interfejs programistyczny usługi GitHub się myli, i odpowiada niezgodnie z rzeczywistością w przypadku wysyłania zapytań warunkowych. Teoria ta znalazła potwierdzenie w trakcie niezależnych prób z użyciem ręcznie wysyłanych zapytań narzędziem cURL - okazało się, że zapytanie z użyciem nagłówka „ETag” działa prawidłowo, ale dla „Last-Modified” informacja już jest błędna. Autor zgłosił błąd, a programista GitHub potwierdził jego istnienie i utworzenie w wewnętrznym systemie zgłoszenia. Poinformował także, że nagłówek ten nie powinien być w ogóle w tym miejscu obsługiwany.

Nie chcąc rezygnować z pamięci podręcznej, w tym momencie wydawało się jasne, że należy zmienić konfigurację biblioteki „OkHttp”, tak by dostosować priorytet używania nagłówków. Analiza dokumentacji i kodu źródłowego tej biblioteki wykazała jednak, że obsługa nagłówka „ETag” ma priorytet przed nagłówkiem „Last-Modified”, więc wszystko powinno być w porządku. Debugowanie aplikacji przy użyciu aplikacji Charles Proxy, tj. podsłuchując wykonywane połączenia HTTP w systemie, wykazało, że prawdopodobnie GitHub API nie odpowiada nieprawidłowo. Właściwie to w ogóle nie odpowiada, gdyż nie może - żadne zapytanie http nie zostało do niego wysłane.

Ostatecznie okazało się, że GitHub API wysyła w odpowiedzi nagłówki „Cache-Control: private, max-age=60, s-maxage=60”. Nagłówek ten informuje agenta http, że uzyskana odpowiedź pozostaje aktualna przez następne 60 sekund, i nie ma powodu, by w tym czasie wysyłać to zapytanie jeszcze raz. Używany agent http - OkHttp - prawidłowo obsługuje ten nagłówek, i dostając żądanie o ten sam zasób w ciągu 60 sekund od jego zapamiętania, w ogóle nie wykonuje połączenia, tylko od razu pobiera informację z pamięci podręcznej. Dla algorytmu kopiującego informacja sprzed kilku sekund nie jest świeża, należy bezwzględnie zapytać o aktualność zasobu dostawcę, tj. usługę GitHub. Nadpisanie tego nagłówka własną wartością rozwiązało problem.



Debugując ten problem autor zupełnie przez przypadek znalazł błąd w interfejsie programistycznym usługi GitHub. Błąd, chociaż istnieje, nie miał wpływu na faktyczną przyczynę spowolnienia algorytmu. Jest to ciekawe doświadczenie ukazujące, że nawet starannie wykonany zapis przebiegu działania algorytmu nie zawsze dostarcza poszukiwanej informacji. Nie oznacza to jednak, że takich zapisów nie należy dokonywać - bardzo ułatwiają one debugowanie aplikacji, a przetoczoną historię należy traktować w kategorii wyjątku potwierdzającego regułę.

### **10.3. Relacja dziedziczenia po stronie bazy danych**

Inny wartę zanotowania problem autor znalazł w trakcie implementacji systemu formularzy. Jest to błąd z kategorii heisenbug (za Wikipedia: „rodzaj błędu programu komputerowego, który wymyka się próbom wyizolowania warunków jego występowania, na przykład nie występuje lub daje inne rezultaty w trakcie próby powtórzenia go w tych samych warunkach” [11]).

Moduł odpowiedzialny z formularze operuje między innymi na takich encjach jak: Question, Input, InputText, InputScale. Question jest encją odpowiedzialną za przechowywanie szczegółów danego pytania, pozostaje ona w relacji jeden do wielu z encją Input. Encja Input natomiast jest typem abstrakcyjnym - przechowuje informacje wspólne dla wszystkich typów dostępnych form odpowiedzi. Encje InputText i InputScale uszczegóławiają ten typ abstrakcyjny, i są odpowiedzialne za konkretne typy formy odpowiedzi: odpowiednio komentarz tekstowy, i ocena w skali.

Problem wynika z połączenia wczytywania leniwego (ang. lazy) i relacji dziedziczenia po stronie bazy danych. Z jakiegoś powodu w jednym z miejsc otrzymywany typ Input wyrażał jedynie abstrakcyjną encję Input, nie był natomiast żadnym z typów szczegółowych. Próba debugowania tego błędu była niemożliwa - ustawienie punktu stopu naprawiało problem - otrzymywany Input był typu szczegółowego. Autor nie znalazł przyczyny takiego zachowania, nie wiadomo dlaczego w trakcie próby debugowania błąd nie istniał. Nie może być to błąd typu race condition, gdyż problem dotyczy otrzymanego typu otrzymanego obiektu.

Pierwszą dokonaną próbą naprawy sytuacji było wyłączenie odczytu leniwego z bazy, i zastąpienie go chciwym (ang. eager). Mimo, że jest to bardzo skuteczne rozwiązanie, to wyjątkowo słabe. Ilość dokonywanych zapytań do bazy danych diametralnie wzrosła. Taki spadek wydajności nie był do zaakceptowania.

Ostatecznie problem został ominięty poprzez wykonanie inicjalizacji obiektu i dokonanie jego konwersji na typ „prawdziwy”, zamiast operować na obiekcie typu

„proxy” dostarczanego przez silnik dokonujący mapowania relacyjno-obiektowego. Zmiana ta została dodana do akcesora listy encji typu Input w encji Question. Po tej zmianie system działa prawidłowo, i relacja dziedziczenia nie dostarcza problemów. Nie jest to jednak wyjaśnienie dlaczego bez tego występują problemy, i gdzie faktycznie znajduje się błąd. Próba powtórzenia na prostszym, minimalistycznym projekcie się nie powiodła. Autor zrezygnował z poszukiwania źródeł problemu z ramach wykonywania pracy inżynierskiej, zastosowane rozwiązanie uznał za zadowalające.

## **11. Możliwe kierunki rozwoju systemu**

Przygotowany system jest kompletny. Wszystkie założone wymagania znajdują pokrycie w zrealizowanej funkcjonalności. Nie oznacza to jednak, że nie można go rozbudowywać i rozszerzać dostępnych możliwości. Poniżej wymieniono kilka wartych rozważenia opcji wraz z próbą przybliżenia stopnia trudności ich realizacji.

### **11.1. Internacjonalizacja**

System komunikuje się z użytkownikiem w języku angielskim. Umożliwienie komunikacji w innym języku jest relatywnie proste. Należy przetłumaczyć komunikaty, które zostały już zebrane w jednym pliku, oraz zaprogramować funkcję zmiany języka komunikatów w interfejsie (front-end). Funkcjonalność taka została przewidziana na serwerze (back-end), i odpowiednie funkcje po tej stronie są gotowe do użycia.

Inaczej jest jednak z systemem formularzy, za pomocą którego recenzent udziela odpowiedzi. Ten system nie został zaprojektowany z myślą o wyświetlaniu komunikatów w wielu językach. Nie stanowi to problemu, jeżeli kurs składa się w całości z osób komunikujących się jednym wspólnym językiem. Formularz pod konkretne zadanie można utworzyć wielokrotnie - za każdym razem wpisując komunikaty w języku odpowiednim dla danego kursu. Możliwość tworzenia wielonarodowościowych kursów nie została przewidziana i wymaga przebudowy systemu formularzy.

### **11.2. Wielu prowadzących**

System umożliwia ustawienie uprawnień master (prowadzący) wielu użytkownikom. Wszyscy jednak mają tę samą listę kursów, formularzy, zleceń i recenzji. Elementy te nie mają przypisanego właściciela i wymagają wspólnego zaufania pomiędzy osobami z uprawnieniami prowadzącego. Podczas projektowania nie została przewidziana taka możliwość, więc logika z tym związana nie istnieje. Należy ją dopisać, co spowoduje także wymóg zmiany struktury bazodanowej.

### **11.3. Ręczny przydział przeglądów**

Prowadzący przy zlecaniu przeglądu może określić ile prac każdy z kursantów powinien ocenić. Jest to obecnie realizowane losowo, z uwzględnieniem równomierności przydziału, i braku możliwości oceny swojej pracy. Nie ma jednak możliwości ręcznego przydziału kto czyją pracę powinien ocenić. Realizacja takiej opcji nie powinna sprawiać problemów komplementacyjnych, choć wymaga dostosowania algorytmu do takiej możliwości.

### **11.4. Panel konfiguracyjny**

System obecnie jest konfigurowany przy użyciu pliku XML. Konfiguracja jest ładowana raz - przy uruchomieniu serwera. Nie ma możliwości zmiany konfiguracji podczas działania systemu. Zmiana zawartości pliku konfiguracyjnego wymaga przeładowania całego systemu. System można usprawnić umożliwiając zmianę konfiguracji podczas pracy systemu. Sposobów na realizację tego są co najmniej dwa. Jednym z nich jest zrealizowanie opcji ponownego odczytu konfiguracji, i aktualizacji elementów zależnych od tej konfiguracji podczas działania systemu - przy czym nadal konfiguracja znajduje się w pliku. Drugim sposobem jest przebudowanie systemu konfiguracji na nowo - wprowadzenie panelu konfiguracyjnego, i przeniesienie wpisów konfiguracyjnych z pliku do systemu bazodanowego.

Stopień skomplikowania zależy od przyjętych nowych założeń, które elementy konfiguracji mogą być zmienione bez potrzeby ponownego uruchomienia serwera. Niektóre z nich nie sprawią żadnych trudności, i nie wymagają dodatkowych działań. Do nich zaliczyć można np. listę osób z uprawnieniami prowadzących, czy listę kont, które są źródłem zadań w usłudze Github. Są jednak też takie, których zmiana może znacznie skomplikować logikę. Do tych zaliczyć należy zmianę folderu, który służy jako cache połączeń, oraz zmianę sekretnych danych OAuth2 identyfikujących aplikację w usłudze GitHub.

### **11.5. Obsługa repozytoriów prywatnych**

System zrealizowano bazując na obsłudze kont darmowych - w których repozytoria kontroli wersji są publicznie dostępne. Wykorzystana usługa GitHub umożliwia jednak także tworzenie organizacji w ramach których repozytoria są prywatne. Konta takie zwykle są płatne, lecz w tym konkretnym przypadku można poprosić o licencję akademicką, do celów naukowych. Aktualnie takie repozytoria nie są obsługiwane.

GitHub API repozytorium, i repozytorium w ramach organizacji uznaje za inny rodzaj zasobu. Ten drugi nie jest w tym momencie w ogóle używany. Algorytmy należy uogólnić, tak by odpytywał o prawidłowy zasób. Należy też rozbudować konfigurację systemu i formularz tworzenia zleceń przeglądów.

## **11.6. Zmniejszenie ilości wykonywanych kopii**

Zastosowany algorytm anonimizacji można zoptymalizować pod kątem czasowym, ilości wykonywanych operacji, i zapytań do GitHub API. Miejscem, w którym można znaleźć dość oczywisty słaby punkt jest ilość wykonywanych kopii. W tym momencie, jeżeli w ramach przeglądu prowadzący kurs zleci, by każdy ocenił pracę 3 innych kolegów, to każda z prac będzie skopiowana co najmniej 3 razy - dla każdej oceny.

Usunięcie wykonywania zbędnych kopii wymaga kilku zmian. Po pierwsze, najbardziej oczywiste - poprawa algorytmu kopiującego. Po drugie zmiana struktury bazodanowej - która aktualnie przewiduje oddzielny link dla każdej kopii. Do takiej zmiany dostosować należało by także algorytmy oczyszczające - kasujące kopie, tak by nie próbowały wielokrotnie usuwać tego samego repozytorium.

## 12. Podsumowanie

W ocenie autora realizacja systemu zakończyła się sukcesem. Udało się zaprojektować, zaimplementować i opisać działający system, który spełnia wszystkie założone wymagania. System może być wdrożony i używany. Dzięki wysokiej integracji z repozytoriami Git i usługą GitHub system nie jest zamkniętym ekosystemem, którego wykorzystanie wymaga zmiany przyzwyczajeń prowadzących. Studenci mogą oceniać wykonanie zadania przez kolegów, na zlecenie prowadzącego, nie sugerując się autorem rozwiązania. Poszerzą w ten sposób swoją wiedzę oraz będą pisać kod coraz wyższej jakości. Autor głęboko wierzy w biznesową użyteczność stworzonego narzędzia i korzyści jakie może przynieść jego wykorzystanie.

## Spis rysunków

5.1.	Przebieg uwierzytelnienia z użyciem protokołu OAuth2 . . . . .	21
6.1.	Architektura systemu - struktura plików . . . . .	22
6.2.	Architektura systemu - zależność pakietów . . . . .	24
7.1.	Konfiguracja systemu - przykładowy plik . . . . .	25
7.2.	Zrzut ekranowy: zarejestrowana aplikacja w usłudze GitHub . . . . .	27
8.1.	Zrzut ekranowy: strona główna dla niezalogowanego użytkownika . . . . .	29
8.2.	Zrzut ekranowy: autoryzacja aplikacji w serwisie GitHub . . . . .	29
8.3.	Listing kodu źródłowego: klasa InterceptorRegistryConfig . . . . .	31
8.4.	Listing kodu źródłowego: klasa InterceptorHasRoleMaster . . . . .	31
8.5.	Zrzut ekranowy: lista kursów . . . . .	32
8.6.	Zrzut ekranowy: lista kursantów . . . . .	33
8.7.	Zrzut ekranowy: formularz dodania kursu . . . . .	33
8.8.	Zrzut ekranowy: powielenia repozytorium na platformie GitHub . . . . .	35
8.9.	Zrzut ekranowy: lista formularzy w systemie . . . . .	37
8.10.	Zrzut ekranowy: podgląd formularza po jego wygenerowaniu . . . . .	38
8.11.	Zrzut ekranowy: przykładowy plik XML z definicją formularza . . . . .	38
8.12.	Zrzut ekranowy: lista przeglądów w systemie . . . . .	39
8.13.	Zrzut ekranowy: formularz dodawania przeglądu . . . . .	40
8.14.	Listing kodu źródłowego: tworzenie przeglądu, klasa GHReviewCreator (część 1) . .	41
8.15.	Listing kodu źródłowego: tworzenie przeglądu, klasa GHReviewCreator (część 2) . .	42
8.16.	Repozytorium z zadaniem i jego powielenia nad kontach w usłudze GitHub . . . . .	44
8.17.	Listing kodu źródłowego: anonimizacja prac, kopiowanie repozytorium . . . . .	45
8.18.	Zrzut ekranowy: status zleceń w przeglądzie . . . . .	46
8.19.	Zrzut ekranowy: anonimowe kopie zleceń w usłudze GitHub . . . . .	46
8.20.	Zrzut ekranowy: lista zleceń na koncie kursanta . . . . .	47
8.21.	Zrzut ekranowy: arkusz MS Excel z podsumowaniem dokonanych ocen . . . . .	48
8.22.	Zrzut ekranowy: statystyki systemowe . . . . .	49
9.1.	Schemat bazy danych . . . . .	51
10.1.	Listing kodu źródłowego: nieoczekiwany błąd (Error) podczas połączeń z GitHub API	54
10.2.	Listing kodu źródłowego: klasa GHCommunicationException . . . . .	55

## Bibliografia

- [1] Katarzyna Biernat. *Po co robimy code review?* 2012. URL: <http://xlab.pl/po-co-robimy-code-review/> (term. wiz. 15. 12. 2015).
- [2] Viktor Farcic. *GitHub vs GitLab vs BitBucket Server (Formerly Stash)*. 2015. URL: <http://technologyconversations.com/2015/10/16/github-vs-gitlabs-vs-bitbucket-server-formerly-stash/> (term. wiz. 12. 12. 2015).
- [3] Trzmiel, Filemon. *Definicja słowa Framework w Wikipedia*. 2015. URL: <https://pl.wikipedia.org/wiki/Framework> (term. wiz. 14. 01. 2016).
- [4] Eclipse Foundation. *Eclipse Community Survey 2014*. 2014. URL: <http://www.slideshare.net/IanSkerrett/eclipse-community-survey-2014> (term. wiz. 08. 12. 2015).
- [5] Mateusz Harasymczuk. „Development to nie tylko kod”. W: *Programista* 13 (2013), s. 134–135.
- [6] Chris Piech, Jonathan Huang. *Tuned Models of Peer Assessment in MOOCs*. 2013. URL: <http://arxiv.org/pdf/1307.2579.pdf> (term. wiz. 14. 01. 2016).
- [7] GitHub Inc. *Dokumentacja GitHub API v3*. 2015. URL: <https://developer.github.com/v3/> (term. wiz. 01. 11. 2015).
- [8] GitHub Inc. *Informacje prasowe usługi GitHub*. 2016. URL: <https://github.com/about/press> (term. wiz. 14. 01. 2016).
- [9] GitLab Inc. *Dokumentacja GitLab API*. 2015. URL: <http://doc.gitlab.com/ee/api> (term. wiz. 12. 12. 2015).
- [10] Erudis Process Management. *Po co nam kontrola wersji kodu źródłowego?!* URL: <http://www.erudis.pl/pl/node/104> (term. wiz. 14. 01. 2016).
- [11] Eric Raymond, Margos. *Definicja słowa Heisenbug w Wikipedia*. 2015. URL: <https://pl.wikipedia.org/wiki/Heisenbug> (term. wiz. 13. 12. 2015).
- [12] Lohit Mehta. *SAML, OAuth, OpenID*. 2014. URL: <http://resources.infosecinstitute.com/saml-oauth-openid/> (term. wiz. 28. 12. 2015).
- [13] Thomas Mueller. *Porównanie wydajności systemów bazodanowych w dokumentacji systemu H2*. URL: <http://www.h2database.com/html/performance.html> (term. wiz. 08. 12. 2015).
- [14] Adam Roman. *Testowanie i jakość oprogramowania. Modele, techniki i narzędzia*. Wydawnictwo Naukowe PWN S.A., 2015, s. 107–127. ISBN: 978-73-01-18160-4.



- [15] Phillip Webb, Dave Syer. *Dokumentacja Spring Boot - "Spring Boot Reference Guide"*. 2015. URL: <http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/> (term. wiz. 01.10.2015).
- [16] Mike Youngstrom. *Spring Framework issue: Improved support for 'containerless' web application architectures*. 2012. URL: <https://jira.spring.io/browse/SPR-9888> (term. wiz. 14.01.2016).