|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | |  | Wydział Informatyki i Zarządzania kierunek studiów: Informatyka Praca dyplomowa - inżynierskaSystem przeprowadzania testów z wielowyboremMaciej Struś słowa kluczowe:  1 linia  2 linia  3 linia  krótkie streszczenie:  1 linia  2 linia  3 linia  4 linia  5 linia   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | opiekun pracy  dyplomowej | .................................................. | | ....................... | | ....................... | | | *Tytuł/stopień naukowy/imię i nazwisko* | | *ocena* | | *podpis* | | | Ostateczna ocena za pracę dyplomową | | | | | | | | | Przewodniczący Komisji egzaminu dyplomowego | | | ..................................................  *Tytuł/stopień naukowy/imię i nazwisko* | | ....................... | | ....................... | | *ocena* | | *podpis* |   *Do celów archiwalnych pracę dyplomową zakwalifikowano do:\**   1. *kategorii A (akta wieczyste)* 2. *kategorii BE 50 (po 50 latach podlegające ekspertyzie)*   *\* niepotrzebne skreślić*   |  | | --- | | pieczątka wydziałowa | |
|  |  |  | Wrocław 2018 |

**Streszczenie**

**Spis treści**

1. Wstęp3

* 1. Wprowadzenie do tematyki, motywacja3
  2. Cel i zakres pracy4

2. Przegląd literatury w zakresie tematyki pracy 5

2.1. Przegląd istniejących rozwiązań5

2.1.1. Moodle5

2.1.2. Google Forms6

2.2. Przegląd technologii7

2.2.1. Vue.js7

2.2.2. Spring Boot7

2.2.3. H28

3. Specyfikacja wymagań9

3.1. Opis problemu, wizja systemu9

3.1.1. Użytkownicy9

3.1.2. Historyjki użytkowników10

3.2. Wymagania funkcjonalne i niefunkcjonalne10

3.3. Przypadki użycia11

3.3.1. Opis przypadków użycia11

3.3.2. Diagram przypadków użycia19

3.4. Prototypy interfejsu użytkownika19

3.5. Model domenowy, reguły biznesowe i ograniczenia dziedzinowe26

4. Projekt aplikacji 30

4.1. Architektura aplikacji30

4.1.1. Diagram pakietów30

4.1.2. Diagram klas32

4.2. Baza danych36

5. Implementacja37

5.1. Przykład implementacji funkcjonalności37

5.1.1. Implementacja komponentu37

5.1.2. Implementacja kontrolera39

5.1.3. Implementacja serwisu i DTO40

5.1.3. Implementacja repozytorium41

5.2. Interfejs użytkownika41

5.3. Narzędzia programistyczne48

6. Testowanie produktu49

6.1. Testy jednostkowe30

6.2. Testy integracyjne30

6.3. Testy systemowe30

7. Podsumowanie1

**1. Wstęp**

**1.1 Wprowadzenie do tematyki, motywacja**

Obecnie coraz częściej osoby prowadzące kursy na uczelniach wyższych oraz nauczyciele w szkołach rezygnują z tradycyjnego przeprowadzania sprawdzianów na papierze i decydują się na testy w formie elektronicznej. Dotyczy to nie tylko kursów prowadzonych w postaci e-learningów. Może to być spowodowane postępem technologicznym, rozwojem technologii webowych, który doprowadził do ułatwienia dostępu do interaktywnych, nieskomplikowanych w obsłudze aplikacji webowych.

Dostępność takich systemów i możliwość przeprowadzania testów w formie elektronicznej jest dogodnym ułatwieniem dla dydaktyków, ponieważ sprawdzanie stanu wiedzy wymaga dużego nakładu pracy oraz czasu. Skrócenie tego czasu oraz umożliwienie zmniejszenia nakładu pracy pozwali na zwiększenie efektywności dydaktyków. Umożliwia również prostsze konstruowanie testów składających się z pytań wielokrotnego wyboru.

Warto odpowiedzieć sobie na pytanie czym jest pytanie testowe wielokrotnego wyboru. Mianowicie jest to zestaw składający się z treści pytania, niepustego zbioru opcji będących odpowiedziami na pytanie, z czego co najmniej jedna odpowiedź jest prawidłowa. Osoba rozwiązująca test ma za zadanie przy każdym pytaniu wybrać te odpowiedzi, które jego zdaniem są prawidłowe.

Aby ocenić test składający się z takich pytań musimy ocenić każde pytanie z osobna. „O ile w klasycznych testach „kartkowych” ich twórcy dążą do maksymalnej prostoty systemu oceniania albowiem ułatwia to później sprawdzanie i wystawianie ocen, o tyle w testach komputerowych stopień komplikacji systemu oceniania nie ma znaczenia z punktu widzenia czasochłonności procesu wystawiania ocen.” [17] Ma znaczenie natomiast z punktu widzenia skomplikowania oraz możliwości realizacji żądanego sposobu oceniania. Z tego powodu (Przyłuski, 2014) [17] proponuje trzy systemy oceniania:

1. Dwuwartościowa ocena (DO)
2. Odpowiedzi częściowo prawidłowe bez żadnej nieprawidłowej odpowiedzi (CO)
3. Odpowiedzi częściowo poprawne i częściowo niepoprawne (TO) (totalna ocena)

Dwuwartościowa ocena polega na pozytywnym wyniku jedynie kiedy rozwiązujący test wybierze jedynie wszystkie prawidłowe odpowiedzi z danego pytania.

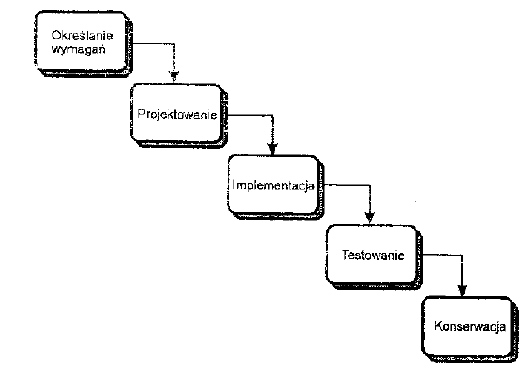
System oceniania (CO) pozwala oceniać pozytywnie również rozwiązania prawidłowe, natomiast niepełne. Za każdą prawidłową odpowiedź rozwiązujący test dostaje ocenę cząstkową za pytanie zależną od wagi danej opcji.

Totalna ocena daje możliwość indywidualnej oceny każdego rozwiązania pytania. Odpowiedzi prawidłowe są liczone pozytywnie do oceny za pytanie, natomiast odpowiedzi nieprawidłowe negatywnie. W tym systemie również istotne są wagi nadawane odpowiedziom.

Motywacją do podjęcia tematu pracy inżynierskiej umożliwienie przeprowadzanie testów z pytaniami wielokrotnego wyboru z możliwością wyboru systemu oceniania pytań z systemów DO, CO oraz TO. Oprócz tego umożliwienie wyboru czy przy korzystaniu z systemu TO punkty negatywne będą liczone do puli punktów z całego testu, czy będą traktowane jako rozwiązane źle i punkty za odpowiedź będą zerowane.

**1.2 Cel i zakres pracy**

Celem pracy jest dostarczenie oprogramowania spełniającego określone w procesie specyfikacji wymagań wymagania. Oprogramowanie skierowane jest do osób prowadzących kursy oraz do studentów, którzy te testy rozwiązują. Oprogramowanie ma umożliwiać tworzenie oraz przeprowadzanie testów z pytaniami wielokrotnego wyboru z danym systemem oceniania dla grupy studentów.   
 Zakres pracy obejmuje przegląd literatury zawierający przegląd rozwiązań dostępnych obecnie na rynku oraz przegląd literatury dotyczącej wykorzystanych w pracy technologii.  
Następnie w pracy znajdzie się opis procesu wytwarzania oprogramowania zgodny z klasycznym modelem kaskadowym z wykluczeniem etapu „Wdrożenie, pielęgnacja”.

  
Rys 1.1. Model kaskadowy cyklu życia oprogramowania   
Jaszkiewicz A. *Inżynieria oprogramowania. Helion, 1997*   
Tak jak zostało to przedstawione na Rys 1.1. jego model, przedstawia następujące po sobie kaskadowo etapy. Jest to model wytwarzania oprogramowania oparty na planowaniu, oznacza to, że wszystkie czynności zanim zostaną wykonane, muszą wcześniej zostać zaplanowane. [20] W ramach określania wymagań zostanie przygotowany opis problemu, na podstawie którego opracowana zostanie wizja systemu. Następnie określone zostaną wymagania na system. W etapie analizy wymagań opracowane zostaną: diagram przypadków użycia wraz z opisami przypadków użycia, prototypy interfejsu. Opracowany również zostanie diagram domenowy wraz z określonymi regułami biznesowymi i ograniczeniami dziedzinowymi.  
Podczas etapu projektowania aplikacji opracowana zostanie architektura aplikacji, w skład której wchodzić będzie diagram pakietów oraz diagram klas, oraz baza danych z jej modelem fizycznym. Etap implementacji składać będzie się z opisu rozwiązań programistycznych, opisu zaimplementowanego interfejsu użytkownika oraz wykorzystanych narzędzi programistycznych. Na etapie testowania opracowane zostaną testy jednostkowe, integracyjne oraz systemowe.**2. Przegląd literatury w zakresie tematyki pracy**

**2.1 Przegląd istniejących rozwiązań**

Obecnie są już dostępne na rynku systemu umożliwiające przeprowadzanie testów wielokrotnego wyboru. Są one szeroko wykorzystywane zarówno w celach edukacyjnych jak również w życiu codziennym. Nic dziwnego, ponieważ problem, który pozwalają rozwiązać nie jest zagadnieniem nowym w świecie technologii.   
 Przedstawione poniżej rozwiązania dotyczą tego samego problemu, jednak różnią się w wielu aspektach. Są kierowane do innej grupy użytkowników, proponują inne rozwiązania i funkcjonalności, a przede wszystkim tworzone są z myślą o innych zastosowaniach testów wielokrotnego wyboru.

2.1.1 Moodle

Moodle [12] jest darmową platformą e-learningową rozwijaną jako otwarte oprogramowanie. Wymaga on do działania uruchomienia własnej strony internetowej, na której działało będzie środowisko Moodle. Skutkuje to tym, że do utrzymywania tego serwisu jest potrzeba posiadania własnej infrastruktury, serwerów oraz administratorów. W związku z tym jeżeli nie posiadamy skonfigurowanej strony Moodle stworzenie prostego testu wielokrotnego wyboru może okazać się ciężkim i czasochłonnym wyzwaniem.   
 Moodle jednak oferuje znacznie większą liczbę funkcjonalności niż potrzebna jest do przeprowadzania testów wiedzy. Pozwala on na tworzenie kursów, dzielenie się materiałami, tworzenie forów aktualności, przesyłanie plików, przechowywanie pytań, dzielenie się pytaniami oraz wiele innych [13]. Również podczas tworzenia sprawdzianów Moodle oferuje znaczną ilość możliwych typów pytań.   
 Jednak najważniejszą funkcjonalnością Moodle w kontekście tematu pracy są pytania wielokrotnego wyboru.   
 Nauczyciel może wybrać “dozwolone jest wiele odpowiedzi” w rodzaju pytania wielokrotnego wyboru. Rodzaj pytania „wiele odpowiedzi” w quizie pozwalają na wybranie jednej lub więcej odpowiedzi. Każda odpowiedź może mieć pozytywną lub negatywną ocenę aby wybranie wszystkich opcji nie koniecznie prowadziło do uzyskania dobrej oceny. Jeżeli sumaryczna ocena jest ujemna, wtedy ostateczna ocena za to pytanie będzie wynosić zero. Dodatek umożliwiający pytania typu „Wszystko albo nic” jest przystosowany z istniejącego typu pytania wielokrotnego wyboru. Jeżeli wybrane opcje odpowiadają dokładnie odpowiedziom zdefiniowanym jako poprawne, odpowiadający uzyskuje 100% punktów. Jeżeli wybierze jakąkolwiek nieprawidłową opcję lub nie wybierze wszystkich poprawnych uzyskuje 0% punktów.[14]   
Oznacza to, że Moodle wykorzystuje system TO bez punktów negatywnych w ramach całego testu oraz po zainstalowaniu dodatku również system DO. Warto przypomnieć, że instalowanie dodatku musi zostać dokonane przez administratora, więc uzyskanie dodatkowego systemu oceniania nie jest łatwym procesem.

2.1.2 Formularze Google

Formularze Google [6] jest to system, którego główną funkcjonalnością jest tworzenie ankiet internetowych. Główną grupą użytkowników, do których skierowany jest system są ludzie, którzy potrzebują szybko i w prosty sposób utworzyć formularz do zbierania informacji. Jednak producent pomiędzy planowaniem wypadu pod namiot, rejestracją osób na szkolenie czy sondą wymienia również przeprowadzanie testu. [x] Formularze Google umożliwiają wybór różnych typów pytań od jednokrotnego wyboru przez wielokrotny aż po menu czy krótkie pytania otwarte. Formularze umożliwiają również graficzna reprezentację zebranych wyników.  
 Podobnie jak w przypadku Moodle w kontekście tematu pracy interesujące są pytania wielokrotnego wyboru oraz systemy oceniania. Formularze Google oferują funkcjonalność automatycznego oceniania odpowiedzi, natomiast jedynie w systemie CO. Chcąc uzyskać inny system oceniania, odpowiedzi należy oceniać ręcznie. Co prawda pozwala ocenianie grup odpowiedzi natomiast nie dzieje się to bez ingerencji osoby sprawdzającej, co może być przeszkodą w stosowaniu bardziej skomplikowanych systemów oceniania. Jedyną możliwością konfiguracji automatycznego oceniania jest przypisywanie punktów poszczególnym zadaniom. Co więcej Google w swoim produkcie nie pozwala na przenoszenie pytań do innych testów ani na jakiekolwiek ich zapisywanie.

****Rys 2.1. Widok tworzenia testu w aplikacji Formularze Google   
Zrzut ekranu ze strony: https://docs.google.com/forms/d/e/1FAIpQLSdFIFB5c2ZphgFoJnsf69URQ0lmZuccaYaTHiBCOOODnbThcg/viewscore?viewscore=AE0zAgAlA8Du2XTy8qyrQ-0qoIAwWhd\_AIWjGhrQcfoJTSYCgAE7VQ6uk739JQ  
 Znacznym atutem aplikacji jest jej wygląd zaprezentowany na Rys 2.1. Jest on zgodny z zasadami projektowania zaproponowanymi przez Google [9]. Jest to zamysł projektowania interfejsów użytkownika, z którym każdy użytkownik produktów Google jest zaznajomiony. Zamysł ten jest szeroko stosowany nie tylko w produktach Google. Uważany jest za intuicyjny i przyjazny użytkownikowi.

**2.2 Przegląd technologii**

W tym rozdziale przestawione zostaną technologie, które zostaną użyte aby zrealizować cel pracy. Na wybór technologii użytych w realizacji celów pracy miały wpływ następujące czynniki: dostępność technologii, możliwość wykorzystania ich do celów edukacyjnych bez obowiązku uiszczania opłat, czy technologie są rozwijane na zasadach otwartego oprogramowania, dostępność materiałów pozwalających na naukę oraz aktywność społeczności.

2.2.1. Vue.js

Vue.js [30] jest platformą programistyczną języka JavaScript umożliwiającą budowanie interfejsów użytkownika. Platforma przez firmę Vue Technology, której założycielem jest Evan You, były pracownik firmy Google. Vue.js rozwijane zgodnie z regułami otwartego oprogramowania, dzięki czemu posiada liczną i aktywną społeczność.   
 Jest on platformą, którą wykorzystać można zarówno w małych, statycznych stronach internetowych jak i w dużych, dynamicznych aplikacjach webowych. Aby tworzyć własną aplikację przy pomocy Vue.js wystarczy dodanie tagu <script>. Vue udostępnia też dedykowane rozwiązanie do **obsługi routingu**, dzięki któremu stronę można przekształcićw **SPA**. Takie podejście u**możliwia również naukę tej technologii wraz z rozwijaniem własnej aplikacji webowej.   
 Wspomniane „single file components”[30], czyli jedno plikowe komponenty są mechanizmem platformy pozwalającej na grupowanie kodu źródłowego aplikacji zgodnie z przyjętą konwencją. Komponent posiada unikalną w kontekście aplikacji nazwę. Składa się z szablonu zawierającego strukturę zdefiniowaną w języku znaczników HTML, części skryptowej, w której umieszczone są skrypty w języku JavaScript pozwalające na dynamiczne zarządzanie komponentem oraz części stylu, w której można zdefiniować styl komponentu za pomocą języka CSS. Taki podział odpowiedzialności pozwala na ustrukturyzowanie kodu na stronie, co poprawia czytelność, przyśpiesza proces implementacji oraz ułatwia utrzymanie i pielęgnację.  
 Wspomniane określenie „SPA” jest akronimem angielskiego wyrażenia „single-page application” oznaczającego aplikację jednostronicową. Jest to technologia, która rozwinęła się wraz z rozwojem nowoczesnych przeglądarek internetowych. W klasycznym podejściu do kreowania stron internetowych kolejne podstrony czy widoki były plikami wysyłanymi z serwera, podczas gdy klient zajmował się jedynie odpowiednim ich wyświetleniem. W podejściu aplikacji jednostronicowej aplikacja webowa posiada jeden niezmienny szablon, do którego zaczytywane są nowe dane zależne od kontekstu. Wszystkie te działania wykonywane są właśnie po stronie klienta. Również nawigowanie po stronie, trasowanie, nazywane „routingiem”, realizowane jest przez klienta.**

2.2.2. Spring Boot

Spring Boot [4] jest rozwiązaniem typu “konwencja ponad konfiguracją” [y] bazujące na platformie programistycznej Spring Framework. Oznacza to, że jest platformą, która upraszcza, jednocześnie nie ograniczając możliwości, konfigurację i uruchomienie projektu. Spring Boot jest rozwijany na zasadach otwartego oprogramowania przez firmę Pivotal Software, która odpowiedzialna jest również za sam Spring Framework. Spring Boot ułatwia również zarządzanie zależnościami w projekcie, dzięki czemu dodanie do projektu nowej biblioteki programistycznej jest proste i bezproblemowe.   
 Spring Framework natomiast jest platformą programistyczną języka Java również rozwijaną przez firmę Pivotal jako otwarte oprogramowanie. Powstał jako alternatywa i rozszerzenie J2EE.   
 Umożliwia on korzystanie z zaawansowanych programistycznych mechanizmów jak wstrzykiwanie zależności, które pozwala ograniczyć liczbę zależności w projekcie poprzez wstrzyknięcie instancji obiektu zamiast tworzenia obiektu. Dzięki wykorzystaniu tej platformy możliwe jest uporządkowanie kodu źródłowego oraz wykorzystanie zaawansowanych mechanizmów programistycznych w nieskomplikowany sposób.   
 Spring Framework wspomaga również zastosowanie stylu architektonicznego skierowanego na zasoby REST stosowanego w tworzeniu aplikacji webowych. REST jest akronimem od angielskiego wyrażenia „representational state transfer”. REST charakteryzuje się tym, że jest w łatwy sposób skalowalny oraz wysyłane żądania są czytelne. Główne założenia REST to:

* wszystko jest zasobem posiadającym URL
* zasoby, chociaż są tworami abstrakcyjnymi posiadają swoje reprezentacje, które są czytelnymi opisami ich stanu
* zasoby mogą posiadać wiele reprezentacji
* przesyłowi podlegają reprezentacje zasobów [18]

Dozwolone operacje ograniczają się do metod HTTP, z czego najważniejszymi są:

* GET – pobiera reprezentację zasobu
* HEAD – podobne do GET lecz zamiast zwracania reprezentacji zasobu zwraca tylko nagłówki (może służyć do sprawdzenia czy wersja zasobu w cache jest ciągle aktualna)
* POST – tworzy nowy zasób na podstawie dostarczonej reprezentacji
* PUT – podmienia stan zasobu na ten opisany w dostarczonej reprezentacji.
* DELETE – kasuje zasób [18]

2.2.3 H2

H2 [10] jest relacyjnym systemem zarządzania bazą danych napisanym w języku Java, rozwijanym jako otwarte oprogramowanie od 2004 roku. Oryginalnym twórcą H2 jest Thomas Mueller. Dzięki temu, że ten SZBD jest wytwarzany w języku Java zintegrowanie go z aplikacją napisaną w tym samym języku jest ułatwione. H2 nie jest duże, zajmuje zaledwie 1,5 MB. Dużą zaletą H2 jest możliwość szybkiego skonfigurowania czy będzie działać w trybie wbudowanym w aplikację, działając na tym samym JVM, czy w trybie serwera. Jest to funkcjonalność przydatna na etapie implementacji, ponieważ uruchomienie bazy danych w trybie wbudowanym znacznie przyśpiesza działanie.

**3. Specyfikacja wymagań**

Specyfikacja wymagań [20] jest wynikiem uzgodnienia wymagań użytkownika oraz wymagań na system. Wymagania powinny być czytelne, jednoznaczne, łatwe do zrozumienia, kompletne oraz spójne. Powinny być zrozumiałe dla osoby, która nie posiada technicznego wykształcenia, więc nie powinny zawierać szczegółów projektu aplikacji, a opisywać jedynie zachowanie systemu. Dokument zawierający wymagań powinien korzystać z języka naturalnego jak i nieskomplikowanych diagramów.   
 Jednak przed przejściem do procesu specyfikacji wymagań potrzebne jest wcześniejsze zebranie wymagań.

**3.1. Opis problemu, wizja systemu**

Produkt skierowany jest do osób prowadzących zajęcia na uczelniach, w szkołach lub prowadzących szkolenia, dotyczy on również osób rozwiązujących testy, studentów. W celu ułatwienia dostępności, niezależnie od systemu operacyjnego, realizowany jest jako aplikacja webowa.

Problemem, który tworzony system ma na celu rozwiązanie, są trudności z tworzeniem testów wielokrotnego wyboru oraz brak możliwości konfiguracji sposobu oceniania zadań. Dotychczasowe rozwiązania udostępniają takie funkcjonalności, natomiast żadne z dostępnych nie jest całkowicie zorientowane na tworzeniu oraz przeprowadzaniu testów wielokrotnego wyboru. Są to rozwiązania, w których testy wielokrotnego wyboru są jedną z wielu proponowanych funkcjonalności, co nie jest zaletą dla osoby, której jedynym celem jest jedynie przeprowadzenie takiego testu. Dodatkowo rozwiązania te nie oferują wielu możliwości sposobów oceniania pytań wielokrotnego wyboru.

3.1.1. Użytkownicy

Użytkownik, czyli najważniejsza część systemu, jest osobą, do której skierowany jest nasz system. Z potrzeb, wymagań użytkownika wynikają wymagania na system. Z tego powodu ważnym punktem jest zdefiniowanie użytkowników systemu.

W tabeli 3.1. zdefiniowani zostali użytkownicy systemu.

Tabela 3.1. Użytkownicy systemu

| **Nazwa** | **Opis** | **Odpowiedzialności** |
| --- | --- | --- |
| Gość | Osoba nie posiadająca konta w systemie. | Zakładanie konta. |
| Niezalogowany użytkownik | Osoba posiadająca konto w systemie, nie będąca obecnie zalogowana. | Logowanie się do aplikacji. |
| Nauczyciel | Osoba, która chce przeprowadzić test wielokrotnego wyboru, posiadająca konto w systemie. Jest osobą prowadzącą kurs. | Tworzenie pytań testowych, grupowanie ich w przedmioty, działy. Tworzenie testów, w tym przypisywanie studentów. Przeglądanie wyników testów. Edytowanie własnego profilu |
| Student | Osoba uczestnicząca w kursie, rozwiązująca testy, posiadająca konto w systemie. | Przeglądanie własnych testów, rozwiązywanie testów, edytowanie własnego profilu |

3.1.2. Historyjki użytkowników

Historyjka użytkownika [25] jest krótką historią opisującą potrzebę użytkownika, której celem jest zebranie wymagania od użytkownika. Nie zagłębia się ona w jaki sposób system ma realizować żądaną przez użytkownika funkcjonalność. Skupia się ona jedynie na potrzebie. Zakres historyjki powinien być ograniczony do jednej potrzeby, a jeżeli dana potrzeba jest zbyt rozległa, historyjka powinna zostać rozbita. Zaletą historyjek jest to, że są krótkie, zrozumiałe dla każdego niezależnie od wykształcenia czy ekspertyzy.

W tabeli 3.2. wypisane zostały historyjki użytkowników systemu.

Tabela 3.2. Historyjki użytkowników

|  |
| --- |
| **Historyjka użytkownika** |
| Jako *Niezalogowany* użytkownik potrzebuje możliwości zalogowania się na swoje konto aby móc korzystać z systemu. |
| Jako *Gość* potrzebuję możliwości zakładania konta. |
| Jako *Student* potrzebuję możliwości rozwiązywania testu. |
| Jako *Student* potrzebuję informacji o swoim wyniku z testu. |
| Jako *Nauczyciel* potrzebuję możliwości tworzenia zadań i przypisywania odpowiedziom punktów aby mieć wykorzystać możliwości systemów oceniania. |
| Jako *Nauczyciel* potrzebuję możliwości grupowania zadań aby ułatwić ich przeszukiwanie. |
| Jako *Nauczyciel* potrzebuję możliwości przeprowadzania testów aby sprawdzać wiedzę swoich kursantów. |
| Jako *Nauczyciel* potrzebuję możliwości konfigurowania sposobu oceniania testów, wyboru z systemu DO, CO, TO z punktami ujemnymi oraz TO bez punktów ujemnych aby stosować różne techniki dydaktyczne. |
| Jako *Nauczyciel* potrzebuje możliwości konfigurowania sposobu nawigowania po rozwiązywanym teście. |
| Jako *Nauczyciel* potrzebuje dostępu do wyników testów aby móc wystawić oceny kursantom. |

**3.2. Wymagania funkcjonalne i niefunkcjonalne**

Wymagania funkcjonalne są stwierdzeniami o żądanych możliwościach systemu. Wynikają one bezpośrednio z wymagań użytkowników, natomiast nie dotyczą tego co może zrobić użytkownik, ale jakie funkcjonalności udostępni system. [25]   
 Wymagania funkcjonalne zostały przedstawione w tabeli 3.3.

Tabela 3.3. Wymagania funkcjonalne

|  |
| --- |
| **Wymagania funkcjonalne** |
| Umożliwia rejestrację do systemu. |
| Umożliwia logowanie się do systemu. |
| Umożliwia odczytanie oraz modyfikację danych własnego profilu. |
| Umożliwia dodawanie, odczytanie, modyfikacje oraz usuwanie przedmiotów. |
| Umożliwia dodawanie, odczytanie, modyfikacje oraz usuwanie działów przedmiotów. |
| Umożliwia dodawanie, odczytanie, modyfikacje oraz usuwanie pytań testowych. |
| Umożliwia dodawanie, odczytanie, modyfikacje oraz usuwanie testów. |
| Umożliwia konfigurowanie sposobu oceniania sposób oceniania testów oraz sposobu nawigowania po rozwiązywanym teście. |
| Umożliwia dostęp do wyników testu. |
| Pozwala rozwiązywać testy, do których został przyznany dostęp. |

Wymagania niefunkcjonalne są ograniczeniami nakładanymi na funkcjonowanie systemu. Często dotyczą raczej całości systemu niż poszczególnych funkcjonalności systemu. [20] Wymagania niefunkcjonalne dla systemu zostały przedstawione w tabeli 3.4.

Tabela 3.4. Wymagania niefunkcjonalne

| **Wymagania niefunkcjonalne** |
| --- |
| System jest dostępny przez Internet. |
| Możliwe jest korzystanie z systemu na poniżej wymienionych przeglądarkach internetowych:   * Mozilla Firefox w wersji 63.0.1 lub wyższej * Google Chrome w wersji 70.0.3538.77 lub wyższej |
| Funkcjonowanie systemu nie powinno zależeć od systemu operacyjnego, na którym jest uruchamiany. |

**3.3. Przypadki użycia**

Przypadek użycia jest opisem interakcji pomiędzy aktorami a systemem. Aktorzy mogą być użytkownikami systemu lub zewnętrznymi systemami. Przypadek użycia posiada szerszy zakres niż historyjki użytkownika. W przypadku użycia posiadamy już informacje o tym w jaki sposób system będzie się zachowywał, a nie jedynie co i z jakiego powodu oczekuje od systemu użytkownik. Przypadki użycia są również bardziej ustrukturyzowane niż historyjki użytkownika. Zazwyczaj opis scenariusza przypadku użycia składa się z tytułu przypadku użycia, głównego scenariusza, który jest scenariuszem najczęściej wykonywanym oraz rozszerzeniami, czyli scenariuszami alternatywnymi, które mogą być również innymi przypadkami użycia. [25]

3.3.1. Opis przypadków użycia

Poniżej w tabelach 3.5. – 3.22. przedstawione zostały opisy przypadków użycia. Przedstawione zostały w postaci tabel w celu poprawienia czytelności. W pierwszym wierszu tabeli znajduje się tytuł przypadku użycia. W drugim wierszu znajduje się identyfikator postaci PUXXX, gdzie XXX reprezentuje kolejne numery. W trzecim wierszu znajduje się aktor inicjujący przypadek użycia. W kolejnym wierszu opisany jest główny scenariusz przypadku użycia. W ostatnim wierszu tabeli znajdują się scenariusze alternatywne, wraz z odnośnikami do rozszerzających przypadków użycia.

Tabela 3.5. PU001

|  |  |
| --- | --- |
| Rejestracja | |
| Identyfikator | PU001 |
| Aktor | Gość |
| Cel | Założenie konta w systemie |
| Scenariusz:   1. Użytkownik chce się zarejestrować. 2. System wyświetla formularz rejestracji. Do wpisu adres e-mail, hasło oraz powtórzenie hasła. Do wybrania jest również żądana rola: Nauczyciel lub Student. 3. Użytkownik wprowadza dane. 4. System stwierdza poprawność danych i przekierowuje na stronę logowania. | |
| Scenariusz alternatywny – wejście punkt 4:   1. System nie stwierdza poprawności danych i wyświetla komunikat o błędnym wprowadzeniu danych. | |

Tabela 3.6. PU002

|  |  |
| --- | --- |
| Logowanie | |
| Identyfikator | PU002 |
| Aktor | Niezalogowany użytkownik |
| Cel | Zalogowanie się do systemu |
| Scenariusz:   1. Użytkownik chce się zalogować. 2. System wyświetla formularz logowania. Do wpisu adres e-mail oraz hasło. 3. Użytkownik wprowadza dane. 4. System stwierdza poprawność danych i przekierowuje na stronę główną aplikacji. | |
| Scenariusz alternatywny – wejście punkt 4:   1. System nie stwierdza poprawności danych i wyświetla komunikat o błędnym wprowadzeniu danych. | |

Tabela 3.7. PU003

|  |  |
| --- | --- |
| Edytowanie profilu | |
| Identyfikator | PU003 |
| Aktor | Zalogowany użytkownik |
| Cel | Zmiana danych do logowania |
| Scenariusz:   1. Użytkownik chce edytować swój profil. 2. System wyświetla aktualne informacje o profilu użytkownika w formie edytowalnej. 3. Użytkownik zmienia dane, które chce zmodyfikować ( hasło ). 4. Użytkownik zatwierdza zmianę danych. 5. System stwierdza poprawność danych i wyświetla "Profil zaktualizowano". | |

Tabela 3.8. PU003

|  |  |
| --- | --- |
| Przeglądanie kursów. | |
| Identyfikator | PU004 |
| Aktor | Nauczyciel |
| Cel | Utrzymanie kursów dostępnych w systemie. |
| Scenariusz:   1. Nauczyciel chce przeglądać kursy. 2. System wyświetla listę kursów. 3. Nauczyciel chce zakończyć przeglądanie kursów. 4. System zamyka listę kursów. | |
| Scenariusze alternatywne   1. Wejście punkt 3: PU004 – Dodawanie kursu 2. Wejście punkt 3: PU005 – Modyfikowanie kursu 3. Wejście punkt 3: PU006 – Przeglądanie działów kursu 4. Wejście punkt 3: PU009 – Przeglądanie pytań | |

Tabela 3.9. PU005

|  |  |
| --- | --- |
| Dodawanie kursu | |
| Identyfikator | PU005 |
| Aktor | Nauczyciel |
| Cel | Dodanie kursu do systemu |
| Scenariusz:   1. Nauczyciel chce dodać kurs. 2. System wyświetla formularz dodawania kursu. Do uzupełnienia nazwa kursu. 3. Nauczyciel wprowadza dane i potwierdza operację. 4. System stwierdza poprawność danych i przekierowuję do listy kursów. | |
| Scenariusz alternatywny – wejście punkt 3   1. System nie stwierdza poprawności danych i wyświetla komunikat o błędnym wprowadzeniu danych | |

Tabela 3.10. PU006

|  |  |
| --- | --- |
| Modyfikowanie kursu | |
| Identyfikator | PU006 |
| Aktor | Nauczyciel |
| Cel | Zmiana nazwy kursu. |
| Scenariusz:   1. Nauczyciel chce zmodyfikować kurs. 2. System wyświetla formularz modyfikowania kursu. Do modyfikacji nazwa kursu. 3. Nauczyciel modyfikuje nazwę kursu i potwierdza operację. 4. System stwierdza poprawność danych i przekierowuje do listy kursów. | |
| Scenariusz alternatywny – wejście punkt 4   1. System nie stwierdza poprawności danych i wyświetla komunikat o błędnym wprowadzeniu danych   Scenariusz alternatywny – wejście punkt 1   1. Nauczyciel chce usunąć kurs. 2. System wyświetla ekran żądający potwierdzenia operacji. 3. Nauczyciel potwierdza operację. 4. System usuwa kurs i przekierowuje do listy kursów.   Scenariusz alternatywny:  3a. Nauczyciel anuluje żądanie | |

Tabela 3.11. PU007

|  |  |
| --- | --- |
| Przeglądanie działów kursu. | |
| Identyfikator | PU007 |
| Aktor | Nauczyciel |
| Cel | Utrzymanie działów danego kursu. |
| Scenariusz:   1. Nauczyciel chce przeglądać działy kursu. 2. System wyświetla listę działów kursu. 3. Nauczyciel chce zakończyć przeglądanie działów kursu. 4. System zamyka listę działów kursu. | |
| Scenariusze alternatywne   1. Wejście punkt 3: PU007 – Dodawanie kursu 2. Wejście punkt 3: PU008 – Modyfikowanie kursu 3. Wejście punkt 3: PU009 – Przeglądanie pytań 4. Wejście punkt 3: PU009 – Przeglądanie pytań – poprzez wybranie szczególnego działu kursu. | |

Tabela 3.12. PU008

|  |  |
| --- | --- |
| Dodawanie działu kursu | |
| Identyfikator | PU008 |
| Aktor | Nauczyciel |
| Cel | Dodanie nowego działu do kursu. |
| Scenariusz:   1. Nauczyciel chce dodać dział kursu. 2. System wyświetla formularz dodawania działu kursu. Do uzupełnienia nazwa działu. 3. Nauczyciel wprowadza dane i potwierdza operację. 4. System stwierdza poprawność danych i przekierowuję do listy działów kursu. | |
| Scenariusz alternatywny – wejście punkt 3   1. System nie stwierdza poprawności danych i wyświetla komunikat o błędnym wprowadzeniu danych | |

Tabela 3.13. PU009

|  |  |
| --- | --- |
| Modyfikowanie działu kursu | |
| Identyfikator | PU009 |
| Aktor | Nauczyciel |
| Cel | Zmiana wybranego działu kursu. |
| Scenariusz:   1. Nauczyciel chce zmodyfikować dział kursu. 2. System wyświetla formularz modyfikowania działu kursu. Do modyfikacji nazwa działu. 3. Nauczyciel modyfikuje nazwę działu i potwierdza operację. 4. System stwierdza poprawność danych i przekierowuje do listy działów kursu. | |
| Scenariusz alternatywny – wejście punkt 4   1. System nie stwierdza poprawności danych i wyświetla komunikat o błędnym wprowadzeniu danych   Scenariusz alternatywny – wejście punkt 1   1. Nauczyciel chce usunąć dział. 2. System wyświetla ekran żądający potwierdzenia operacji. 3. Nauczyciel potwierdza operację. 4. System usuwa dział i przekierowuje do listy działów.   Scenariusz alternatywny:  3a. Nauczyciel anuluje żądanie | |

Tabela 3.14. PU010

|  |  |
| --- | --- |
| Przeglądanie pytań. | |
| Identyfikator | PU010 |
| Aktor | Nauczyciel |
| Cel | Utrzymanie pytań dostępnych w systemie. |
| Scenariusz:   1. Nauczyciel chce przeglądać pytania. 2. System wyświetla listę pytań ( jeżeli wejście było z PU003 wyświetlane są wszystkie pytania, jeżeli z PU006 ze scenariusza alternatywnego 3 wyświetlane są pytania z wybranego kursu, jeżeli z PUU006 ze scenariusza alternatywnego 4 wyświetlane są pytania z wybranego działu kursu ). 3. Nauczyciel chce zakończyć przeglądanie pytań. 4. System zamyka listę pytań. | |
| Scenariusze alternatywne   1. Wejście punkt 3: PU010 – Dodawanie pytania 2. Wejście punkt 3: PU011 – Modyfikowanie pytania 3. Wejście punkt 3: 4. Nauczyciel chce usunąć pytanie wybrane z listy. 5. System wyświetla ekran żądający potwierdzenia operacji. 6. Nauczyciel potwierdza operację. 7. System usuwa pytanie.   Scenariusz alternatywny:  3a. Nauczyciel anuluje operacje. | |

Tabela 3.15. PU011

|  |  |
| --- | --- |
| Dodawanie pytania | |
| Identyfikator | PU011 |
| Aktor | Nauczyciel |
| Cel | Dodanie nowego pytania do systemu. |
| Scenariusz:   1. Nauczyciel chce dodać pytanie. 2. System wyświetla formularz dodawania pytania. Do uzupełnienia kurs, dział kursu, treść pytania, odpowiedzi na pytanie wraz z punktami. 3. Nauczyciel wprowadza dane i potwierdza operację. 4. System stwierdza poprawność danych i przekierowuję do listy pytań. | |
| Scenariusz alternatywny – wejście punkt 3   1. System nie stwierdza poprawności danych i wyświetla komunikat o błędnym wprowadzeniu danych | |

Tabela 3.16. PU012

|  |  |
| --- | --- |
| Modyfikowanie pytania | |
| Identyfikator | PU012 |
| Aktor | Nauczyciel |
| Cel | Zmiana danych wybranego pytania. |
| Scenariusz:   1. Nauczyciel chce zmodyfikować pytanie. 2. System wyświetla formularz modyfikowania pytania. Do modyfikacji kurs, dział kursu, treść pytania, odpowiedzi na pytanie wraz z punktami. 3. Nauczyciel modyfikuje dane i potwierdza operację. 4. System stwierdza poprawność danych i przekierowuję do listy pytań. | |
| Scenariusz alternatywny – wejście punkt 3   1. System nie stwierdza poprawności danych i wyświetla komunikat o błędnym wprowadzeniu danych | |

Tabela 3.17. PU013

|  |  |
| --- | --- |
| Przeglądanie testów. | |
| Identyfikator | PU013 |
| Aktor | Nauczyciel |
| Cel | Utrzymanie testów dostępnych w systemie. |
| Scenariusz:   1. Nauczyciel chce przeglądać testy. 2. System wyświetla listę testów. 3. Nauczyciel chce zakończyć przeglądanie testów. 4. System zamyka listę testów. | |
| Scenariusze alternatywne   1. Wejście punkt 3: PU013 – Dodawanie testu 2. Wejście punkt 3: PU014 – Modyfikowanie testu 3. Wejście punkt 3: PU015 – Przeglądanie wyników testu 4. Wejście punkt 3: 5. Nauczyciel chce usunąć test wybrany z listy. 6. System wyświetla ekran żądający potwierdzenia operacji. 7. Nauczyciel potwierdza operację 8. System usuwa pytanie.   Scenariusz alternatywny:  3a. Nauczyciel anuluje operację. | |

Tabela 3.18. PU014

|  |  |
| --- | --- |
| Dodawanie testu | |
| Identyfikator | PU014 |
| Aktor | Nauczyciel |
| Cel | Dodanie nowego testu do systemu. |
| Scenariusz:   1. Nauczyciel chce dodać test. 2. System wyświetla formularz dodawania testu. Do uzupełnienia: kurs, data i czas początku testu, data i czas końca testu, metoda oceniania testu, nawigowalność testu, próg zaliczenia testu, hasło do testu. Do wybrania z listy również są pytania oraz studenci. 3. Nauczyciel wprowadza dane i potwierdza operację. 4. System stwierdza poprawność danych i przekierowuję do listy testów. | |
| Scenariusz alternatywny – wejście punkt 3   1. System nie stwierdza poprawności danych i wyświetla komunikat o błędnym wprowadzeniu danych | |

Tabela 3.19. PU015

|  |  |
| --- | --- |
| Modyfikowanie testu | |
| Identyfikator | PU015 |
| Aktor | Nauczyciel |
| Cel | Zmiana danych wybranego testu. |
| Scenariusz:   1. Nauczyciel chce dodać test. 2. System wyświetla formularz modyfikowania testu. Do modyfikacji: kurs, data i czas początku testu, data i czas końca testu, metoda oceniania testu, nawigowalność testu, próg zaliczenia testu, hasło do testu. Do wybrania z listy również są pytania oraz studenci. 3. Nauczyciel wprowadza dane i potwierdza operację. 4. System stwierdza poprawność danych i przekierowuję do listy testów. | |
| Scenariusz alternatywny – wejście punkt 3   1. System nie stwierdza poprawności danych i wyświetla komunikat o błędnym wprowadzeniu danych | |

Tabela 3.20. PU016

|  |  |
| --- | --- |
| Przeglądanie wyników testu | |
| Identyfikator | PU016 |
| Aktor | Nauczyciel |
| Cel | Otrzymanie informacji o wynikach uzyskanych przez studentów w ramach wybranego testu. |
| Scenariusz:   1. Nauczyciel chce przeglądać wyniki testu. 2. System wyświetla ekran z wynikami testu. Wyświetlone są: średnia zdobyta liczba punktów, wszystkie przesłane rozwiązania. 3. Nauczyciel chce zakończyć przeglądanie wyników testu. 4. System zamyka widok wyników testu. | |
| Scenariusz alternatywny – wejście punkt 3   1. Nauczyciel chce zobaczyć rozwiązanie wybrane z listy. 2. System wyświetla żądane rozwiązanie: pytania, wybrane odpowiedzi oraz uzyskany wynik. 3. Nauczyciel chce zakończyć przeglądanie rozwiązania. 4. System zamyka widok i przekierowuje do widoku wyników testu. | |

Tabela 3.21. PU017

|  |  |
| --- | --- |
| Przeglądanie własnych testów. | |
| Identyfikator | PU017 |
| Aktor | Student |
| Cel | Otrzymanie informacji o testach możliwych do rozwiązania bądź o testach posiadających wyniki do przeglądnięcia. |
| Scenariusz:   1. Student chce przeglądać testy, do których jest przypisany. 2. System wyświetla listę testów. 3. Student chce zakończyć przeglądanie testów. 4. System zamyka listę testów. | |
| Scenariusze alternatywne   1. Wejście punkt 3: PU016 – Rozwiązywanie testu | |

Tabela 3.22. PU018

|  |  |
| --- | --- |
| Rozwiązywanie testu | |
| Identyfikator | PU018 |
| Aktor | Student |
| Cel | Rozwiązanie testu. |
| Scenariusz:   1. Student chce rozwiązywać test, do którego jest przypisany. 2. System wyświetla ekran proszący o hasło do testu. 3. Student wprowadza hasło. 4. System stwierdza poprawność wprowadzonego hasła i przekierowuje na widok rozwiązywania testu. 5. Dla każdego dostępnego pytania Student wybiera i zapisuje odpowiedzi. 6. Student chce zakończyć test. 7. System przesyła i ocenia rozwiązanie i przekierowuje na stronę z wynikiem testu. | |

3.3.2. Diagram przypadków użycia

Diagram przypadków użycia został wykonany za pomocą UML. UML, czyli Unified Modeling Language, jest językiem modelowania wizualnego używanym do specyfikacji, wizualizacji, konstruowania oraz dokumentowania artefaktów systemu informatycznego. Pozwala on zamodelować zarówno statyczną strukturę systemu jak i jego dynamiczne zachowanie.[19]  
 Na Rys 3.1. widocznych jest 5 aktorów, przedstawionych jako uproszczona postać człowieka, z czego 4 jest zdefiniowanymi wcześniej użytkownikami systemu. Oprócz tego widoczny jest abstrakcyjny aktor „Zalogowany użytkownik”, który połączony jest asocjacją, nieprzerwaną linią, z przypadkiem użycia PU003 Edytowanie profilu. Aktorzy „Student” oraz „Nauczyciel” połączeni są z „Zalogowanym użytkownikiem” relacją generalizacji, co oznacza, że oboje są „Zalogowanym użytkownikiem”. Przypadki użycia przedstawione są jako elipsy. Oprócz asocjacji, którymi połączone są z aktorami, połączone są również relacją „<<extend>>” z innymi przypadkami użycia. Relacja ta jak sama nazwa wskazuje pozwala „rozszerzać” przypadki użycia, czyli zgodnie ze scenariuszami z poprzedniego rozdziału, pozwala realizować scenariusze alternatywne.

 Rys 3.1. Diagram przypadków użycia

**3.4. Prototypy interfejsu użytkownika**

Prototypy interfejsu użytkownika są nieinteraktywnymi prototypami wyglądu aplikacji. Opracowywanie ich ma na celu bardziej realistyczne przedstawienie jak będzie wyglądać skończona aplikacja niż jest to pokazane w przypadku historyjek użytkownika czy przypadków użycia. Ich opracowanie pozwala również na zweryfikowanie zebranych wymagań i ich dopracowanie.

Projektując interfejs warto kierować się pewnymi zasadami aby użytkownik, który będzie używał systemu nie miał problemu z korzystaniem z aplikacji. Dobrze zaprojektowany interfejs usprawni pracę użytkownika, a przede wszystkim nie będzie mu zawadzał. W tym celu warto wesprzeć się znanymi 10 heurystykami Nielsena [16]:

1. Pokazuj status systemu.
2. Zachowaj zgodność pomiędzy systemem a rzeczywistością.
3. Daj użytkownikowi pełną kontrolę.
4. Trzymaj się standardów i zachowaj spójność.
5. Zapobiegaj błędom.
6. Pozwalaj wybierać zamiast zmuszać do pamiętania.
7. Zapewnij elastyczność i efektywność.
8. Dbaj o estetykę i umiar.
9. Zapewnij skuteczną obsługę błędów.
10. Zadbaj o pomoc i dokumentację.

Kierując się tymi heurystykami opracowane zostały poniższe prototypy interfejsów użytkownika Rys 3.2. – Rys 3.12.

 Rys 3.2. Spis przedmiotów



Rys 3.3. Dodawanie przedmiotu

 Rys 3.4. Spis działów przedmiotu

 Rys 3.5. Dodawanie działu przedmiotu  Rys 3.6. Spis pytań

 Rys 3.7. Dodawanie nowego pytania  Rys 3.8. Spis testów

 Rys 3.9. Dodawanie testu

 Rys 3.10. Rozwiązywanie testu.

 Rys 3.11. Wynik testu

 Rys 3.12. Edytowanie profilu

**3.5. Model domenowy, reguły biznesowe, ograniczenia dziedzinowe** Model domenowy jest modelem, który przedstawia warstwę zakresu biznesowego, którego dotyczy system. Przedstawia on zarówno dane biznesowe jak i reguły biznesowe, które wykorzystane są w systemie. Dane oraz procesy są przedstawione w jednym modelu. Na pierwszy rzut oka model domenowy może wydawać się podobny do modelu bazy danych, natomiast model domenowy miesza ze sobą dane i procesy oraz wykorzystuje mechaniki, które nie są dozwolone przy modelowaniu bazy danych. [7] Model domenowy przedstawiany jest za pomocą statycznego diagramu klas opracowanego za pomocą UML. Jest on łącznikiem pomiędzy specyfikacją wymagań a architekturą aplikacji. Wynika on bezpośrednio z zebranych wymagań i jednocześnie definiuje on strukturę, reguły oraz ograniczenia architektury aplikacji. Model domenowy aplikacji przedstawiony został na Rys 3.13. następnie opracowane zostały reguły biznesowe i ograniczenia dziedzinowe.

**** Rys 3.13 Model domenowy

**Reguły biznesowe**

1. *Dział* jest częścią *Kursu*, może odpowiadać wykładowi bądź grupie wykładów, może odpowiadać też innej strukturze zdefiniowanej przez *Nauczyciela*. Identyfikowany jest przez numer identyfikacyjny. Opisywany jest przez swoją nazwę.
2. *Kurs* jest przedmiotem akademickim. Składa się on z różnych *działów*. Identyfikowany jest prze numer identyfikacyjny. Opisywany jest przez swoją nazwę.
3. *Metoda Oceniania* jest określoną metodą oceniania testów.
4. *Nauczyciel* jest osobą zatrudnioną na uczelni. Jest autorem *pytań* oraz *testów*. Jest identyfikowany poprzez numer identyfikujący. Autoryzowany jest poprzez email oraz hasło.
5. *Odpowiedź* jest jedną z wielu opcji *Pytania,* którą *Student* może wybrać podczas odpowiadania na *Pytanie*. Jest identyfikowana przez numer identyfikujący. Opisywana jest poprzez swoją zawartość, liczbę punktów oraz informacje czy jest poprawna.
6. *Pytanie* jest pytaniem testowym wielokrotnego wyboru. Jest identyfikowane przez numer identyfikujący. Opisywane jest przez swoją zawartość. Pytanie odnosi się do *Kursu* oraz *Działu*. Posiada *Odpowiedzi*.
7. *Rozwiązanie* jest zbiorem *Odpowiedzi* na *Pytania* z *Testu*, którego dotyczy. Przesyłane jest przez *Studenta*. Identyfikowane jest poprzez numer identyfikujący. Opisywane jest poprzez informację czy zostało już przesłane oraz poprzez ilość uzyskanych punktów.
8. *Student* jest osobą biorącą udział w *Kursie*. Przesyła on *Rozwiązania* do *Testów*, w których bierze udział. Jest identyfikowany poprzez numer identyfikujący. Autoryzowany jest poprzez email oraz hasło.
9. *Test* jest testem wielokrotnego wyboru. Identyfikowany jest przez numer identyfikujący. Opisywany jest przez datę i czas początku testu, datę i czas końca testu, liczbę możliwych do uzyskania punktów, wybraną metodę oceniania oraz informację czy test będzie nawigowalny. Nawigowalność oznacza określenie czy *Student* po zapisaniu rozwiązania na dane *Pytanie* będzie mógł do niego wrócić i swoje rozwiązanie skorygować.
10. Liczba możliwych do uzyskania punktów w Teście wyliczana jest na podstawie sumy możliwych do uzyskania punktów z każdego Pytania, które zawarte jest w Teście. Suma możliwych punktów do uzyskania z Pytania w kontekście danego Testu wyliczana jest na podstawie Metody Oceniania Testu i punktów przypisanym Odpowiedziom do danego Pytania.
11. *Student* może przesłać wiele *Rozwiązań*.
12. *Student* może nie przesłać żadnego *Rozwiązania*.
13. *Rozwiązanie* musi być przesłane przez *Studenta*.
14. *Rozwiązanie* może być przesłane przez co najwyżej jednego *Studenta*.
15. *Student* może brać udział w wielu *Testach*.
16. *Student* nie musi brać udziału w *Teście*.
17. *Test* może mieć przypisanych wielu *Studentów*.
18. *Test* nie musi mieć przypisanego *Studenta*.
19. *Rozwiązanie* może zawierać wiele *Odpowiedzi*.
20. *Rozwiązanie* nie musi zawierać *Odpowiedzi*.
21. *Odpowiedź* może się znajdywać w wielu *Rozwiązaniach*.
22. *Odpowiedź* nie musi znajdywać się w *Rozwiązaniu*.
23. *Rozwiązanie* musi rozwiązywać *Test*.
24. *Rozwiązanie* może rozwiązywać co najwyżej jeden *Test*.
25. *Test* nie musi być rozwiązywany przez *Rozwiązanie*.
26. *Test* może być rozwiązywany przez wiele R*ozwiązań*.
27. *Nauczyciel* może być autorem wielu *Testów*.
28. *Nauczyciel* nie musi być autorem *Testu*.
29. *Test* musi posiadać *Nauczyciela* jako autora.
30. *Test* może posiadać co najwyżej jednego *Nauczyciela* jako autora.
31. *Test* może zawierać wiele *Pytań*.
32. *Test* musi zawierać co najmniej jedno *Pytanie*.
33. *Pytanie* może znajdywać się w wielu *Testach*.
34. *Pytanie* nie musi znajdywać się w *Teście*.
35. *Test* przeprowadzany jest z jednego *Kursu*.
36. *Test* musi być przeprowadzany z *Kursu*
37. W ramach *Kursu* przeprowadzane może być wiele *Testów*.
38. W ramach *Kursu* nie musi być przeprowadzany *Test*.
39. *Pytanie* może posiadać wiele *Odpowiedzi*.
40. *Pytanie* może nie posiadać *Odpowiedzi*.
41. *Odpowiedź* może odpowiadać na tylko jedno *Pytanie*.
42. *Odpowiedź* musi odpowiadać na *Pytanie*.
43. *Pytanie* może odnosić się tylko do jednego *Kursu*.
44. *Pytanie* musi odnosić się do *Kursu*.
45. *Pytanie* może odnosić się tylko do jednego *Działu*.
46. *Pytanie* nie musi odnosić się do *Działu*.
47. *Kurs* może zawierać wiele *Działów*.
48. *Kurs* nie musi zawierać *Działu*.
49. *Dział* może być częścią tylko jednego *Kursu*.
50. *Dział* musi być częścią *Kursu*.
51. *Pytania* tworzone są przez *Nauczyciela*.
52. *Kursy* tworzone są przez *Nauczyciela*.
53. *Działy* tworzone są przez *Nauczyciela*.
54. *Rozwiązanie* może być nadsyłane jedynie w czasie pomiędzy początkiem a końcem *Testu*.
55. *Test* jest oceniany poprzez sumowanie wyników z każdego *Pytania*.
56. *Metoda Oceniania* „Dwuwartościowa Ocena” polega na pozytywnym wyniku jedynie kiedy rozwiązujący test wybierze jedynie wszystkie prawidłowe odpowiedzi z danego pytania.
57. *Metoda Oceniania* „Częściowa Ocena” pozwala oceniać pozytywnie również rozwiązania prawidłowe, natomiast niepełne. Za każdą prawidłową odpowiedź rozwiązujący test dostaje punkty cząstkowe za pytanie zależną od liczby punktów wybranych odpowiedzi.
58. *Metoda Oceniania* „Totalna Ocena z punktami ujemnymi” daje możliwość indywidualnej oceny każdego rozwiązania pytania. Odpowiedzi prawidłowe są liczone pozytywnie do oceny za pytanie, natomiast odpowiedzi nieprawidłowe negatywnie.
59. *Metoda Oceniania* „Totalna Ocena bez punktów ujemnych” funkcjonuje tak jak metoda oceniania „Totalna Ocena z punktami ujemnymi”. Różnicą jest liczenie punktów w ramach całego testu. W metodzie „Totalna Ocena bez punktów ujemnych” pozwala na liczenie oceny za pytanie, za które suma punktów wyszła ujemna, jako odpowiedź z liczbą punktów zero w ramach całego testu.

**Ograniczenia dziedzinowe**

1. Numer identyfikacyjny *Działu* jest typu UUID.
2. Numer identyfikacyjny *Działu* jest unikalny.
3. Numer identyfikacyjny *Działu* jest obowiązkowy
4. Nazwa *Działu* jest niepustym łańcuchem znaków.
5. Nazwa *Działu* jest obowiązkowa.
6. Numer identyfikacyjny *Kursu* jest typu UUID.
7. Numer identyfikacyjny *Kursu* jest unikalny.
8. Numer identyfikacyjny *Kursu* jest obowiązkowy
9. Nazwa *Kursu* jest niepustym łańcuchem znaków.
10. Nazwa *Kursu* jest obowiązkowa.
11. *Metoda Oceniania* jest typu wyliczeniowego.
12. Numer identyfikujący *Nauczyciela* jest typu UUID.
13. Numer identyfikacyjny *Nauczyciela* jest unikalny.
14. Numer identyfikacyjny *Nauczyciela* jest obowiązkowy
15. Email *Nauczyciela* jest niepustym łańcuchem znaków.
16. Email *Nauczyciela* jest poprawnym adresem email.
17. Email *Nauczyciela* jest unikalny.
18. Email *Nauczyciela* jest obowiązkowy.
19. Hasło *Nauczyciela* jest niepustym łańcuchem znaków.
20. Hasło *Nauczyciela* jest obowiązkowe.
21. Hasło *Nauczyciela* jest szyfrowane.
22. Numer identyfikacyjny *Odpowiedzi* jest typu UUID.
23. Numer identyfikacyjny *Odpowiedzi* jest unikalny.
24. Numer identyfikacyjny *Odpowiedzi* jest obowiązkowy.
25. Zawartość *Odpowiedzi* jest niepustym łańcuchem znaków.
26. Zawartość *Odpowiedzi* jest obowiązkowa.
27. Liczba punktów *Odpowiedzi* jest liczbą rzeczywistą.
28. Informacja czy *Odpowiedź* jest poprawna jest typu logicznego.
29. Informacja czy *Odpowiedź* jest poprawna jest obowiązkowa.
30. Numer identyfikacyjny *Pytania* jest typu UUID.
31. Numer identyfikacyjny *Pytania* jest unikalny.
32. Numer identyfikacyjny *Pytania* jest obowiązkowy.
33. Zawartość *Pytania* jest niepustym łańcuchem znaków.
34. Zawartość *Pytania* jest obowiązkowa.
35. Numer identyfikacyjny *Rozwiązania* jest typu UUID.
36. Numer identyfikacyjny *Rozwiązania* jest unikalny.
37. Numer identyfikacyjny *Rozwiązania* jest obowiązkowy.
38. Informacja czy *Rozwiązanie* zostało przesłane jest typu logicznego.
39. Informacja czy *Rozwiązanie* zostało przesłane jest obowiązkowa.
40. Liczba punktów *Rozwiązania* jest liczbą rzeczywistą.
41. Numer identyfikujący *Studenta* jest typu UUID.
42. Numer identyfikacyjny *Studenta* jest unikalny.
43. Numer identyfikacyjny *Studenta* jest obowiązkowy
44. Email *Studenta* jest niepustym łańcuchem znaków.
45. Email *Studenta* jest poprawnym adresem email.
46. Email *Studenta* jest unikalny.
47. Email *Studenta* jest obowiązkowy.
48. Hasło *Studenta* jest niepustym łańcuchem znaków.
49. Hasło *Studenta* jest obowiązkowe.
50. Hasło *Studenta* jest szyfrowane.
51. Numer identyfikujący *Testu* jest typu UUID.
52. Numer identyfikacyjny *Testu* jest unikalny.
53. Numer identyfikacyjny *Testu* jest obowiązkowy
54. Moment rozpoczęcia *Testu* jest typu Datetime.
55. Moment rozpoczęcia *Testu* jest obowiązkowy.
56. Moment zakończenia *Testu* jest typu Datetime.
57. Moment zakończenia *Testu* jest obowiązkowy.
58. Liczba punktów *Testu* jest liczbą rzeczywistą.
59. Metoda Oceniania *Testu* jest typu Metoda Oceniania.
60. Metoda Oceniania *Testu* jest obowiązkowa.
61. Informacja czy *Test* jest nawigowalny jest typu logicznego.
62. Informacja czy *Test* jest nawigowalny jest obowiązkowa.

**4. Projekt aplikacji** Proces implementacji aplikacji polega na przetłumaczeniu wyspecyfikowanych wymagań na działający program, aby było to możliwe implementacja musi zostać on poprzedzony procesem projektowania. W ramach projektu przygotowywane są dokumenty dotyczące struktury oprogramowania jak i struktury danych. [20]

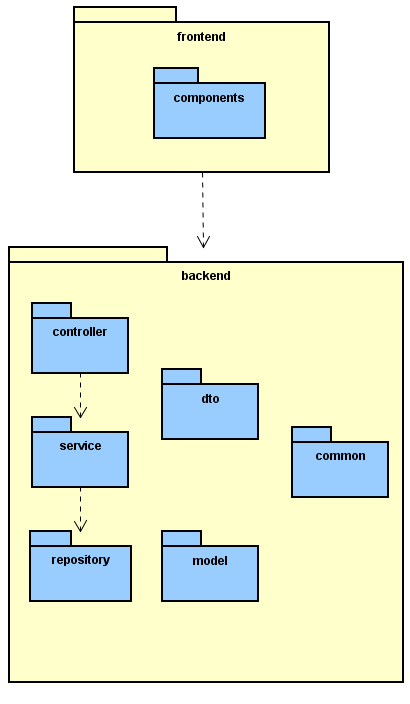
**4.1. Architektura aplikacji**

W ramach projektu aplikacji przygotowana została architektura aplikacji, czyli statyczna struktura opracowywanego oprogramowania. Architektura oprogramowania jest elementem szczególnie ważnym, ponieważ zależy od niej to jak system będzie realizował wymagania funkcjonalne, jak będzie skalowalny, utrzymywalny i niezawodny. Przyjętym w projekcie wzorcem architektonicznym jest architektura wielowarstwowa. W architekturze wielowarstwowej rozdziela się warstwy systemu w zależności od ich odpowiedzialności. Pozwala to uzyskać możliwości skalowalności aplikacji, daje możliwość podmiany poszczególnych segmentów systemu bez przymusu przebudowy całej aplikacji.[20]

4.1.1. Diagram pakietów   
 Diagram pakietów jest przedstawiony za pomocą notacji UML. Reprezentuje on strukturę pakietów w aplikacji. Zdefiniowane zostały następujące warstwy, widoczne na Rys 4.1.:

* Warstwa prezentacji – pakiet „frontend” zawierający strukturę plików Vue.js, w którym znajduje się pakiet „components”, zwierający komponenty SPA, które komunikują się z serwerem backend
* Warstwa pośrednicząca – pakiet „backend.controller”, w którym znajdują się kontrolery REST, nie jest związany ze wzorcem MVC. Warstwa ta funkcjonuje jako fasada dla warstwy prezentacji. Pośredniczy pomiędzy warstwą prezentacji a warstwą logiki biznesowej.
* Warstwa logiki biznesowej – pakiet „backend.service”, w którym znajdują się serwisy, obsługujące logikę biznesową aplikacji.
* Warstwa dostępu do danych – pakiet „repository”. W tym pakiecie znajdują się repozytoria, które obsługują operacje wykonywane na danych.

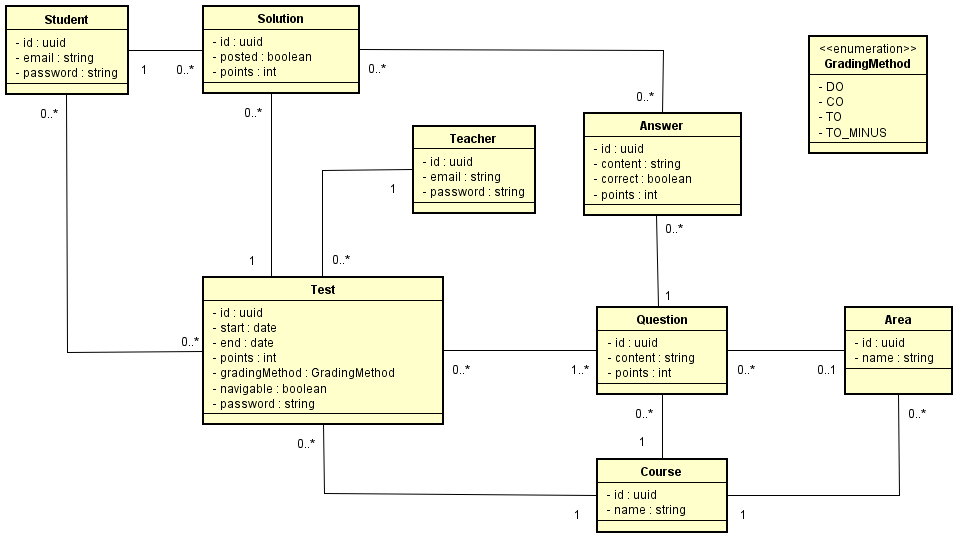
Oprócz widocznych warstw systemu na Rys 4.1. przedstawiony został również pakiet „backend.model”, w którym znajduje się zdefiniowana struktura obiektów biznesowych wykorzystywanych w systemie.   
 Przedstawiony został również pakiet „backend.dto”, w tym pakiecie znajdują się klasy definiujące strukturę używanych w aplikacji DTO. DTO jest akronimem od „Data Transfer Object”, czyli obiekt do transportowania danych. Chociaż zazwyczaj używany jest do grupowania więcej niż jednej encji aby ograniczyć ruch na serwerze, można go również wykorzystać do ograniczenia widoczności atrybutów w przesyłanym obiekcie. Umożliwia również zdefiniowanie struktury obiektów, które serwer będzie przyjmował.[7]   
 Ostatnim widocznym pakietem jest pakiet „backend.common”, w którym znajduje się zarówno konfiguracja systemu jak i infrastruktura wykorzystywana przez wszystkie warstwy znajdujące się w pakiecie „backend”.

  
Rys 4.1. Diagram pakietów

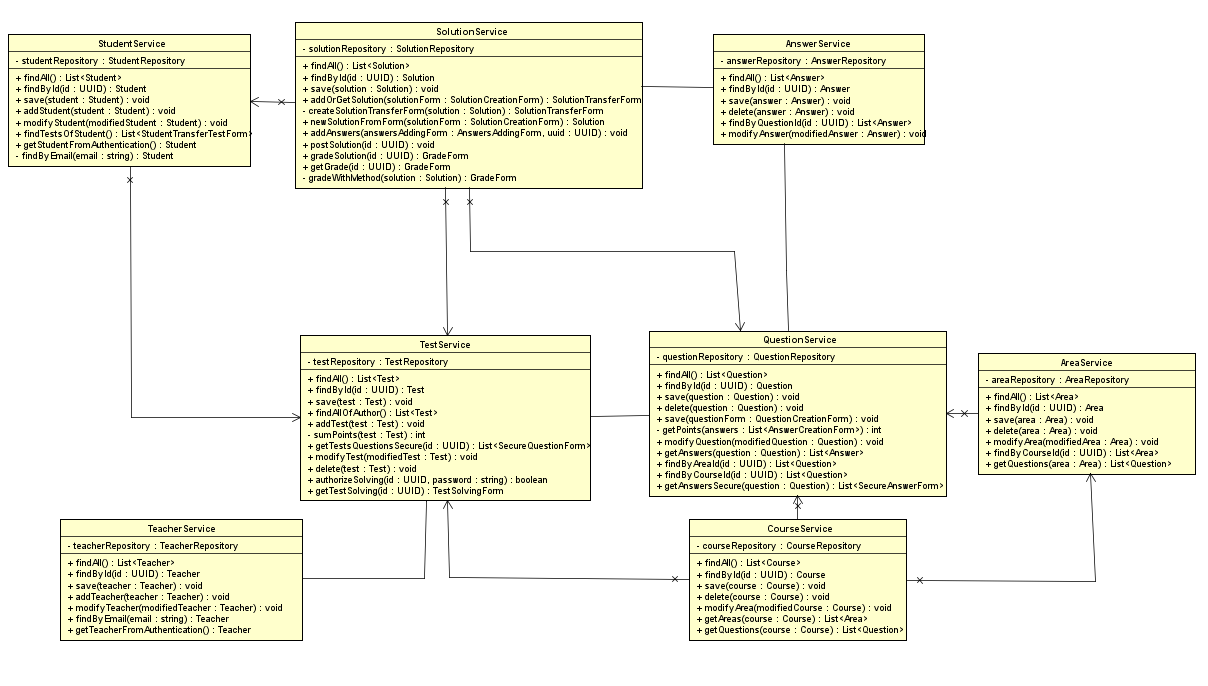
4.1.2. Diagram klas

Diagramy klas wykorzystywane są do modelowania oprogramowania zorientowanego obiektowo. Na celu ma przedstawienie klas obecnych w systemie oraz powiązań między nimi, asocjacji. Jest on bardziej szczegółowy niż diagram pakietów, który przedstawia jedynie wysokopoziomową strukturę systemu.

W pakiecie „model” znajdują się klasy modelujące obiekty reprezentujące byty znane z rzeczywistości opisanej w procesie specyfikacji wymagań. Klasy te przedstawione zostały na Rys 4.2.. W dużej mierze wywodzi się on z modelu domenowego, natomiast dla utrzymania spójności z językiem Java nazwy klas oraz atrybuty przetłumaczone zostały na język angielski. W klasach zostały pominięte akcesory pól, gettery oraz settery.

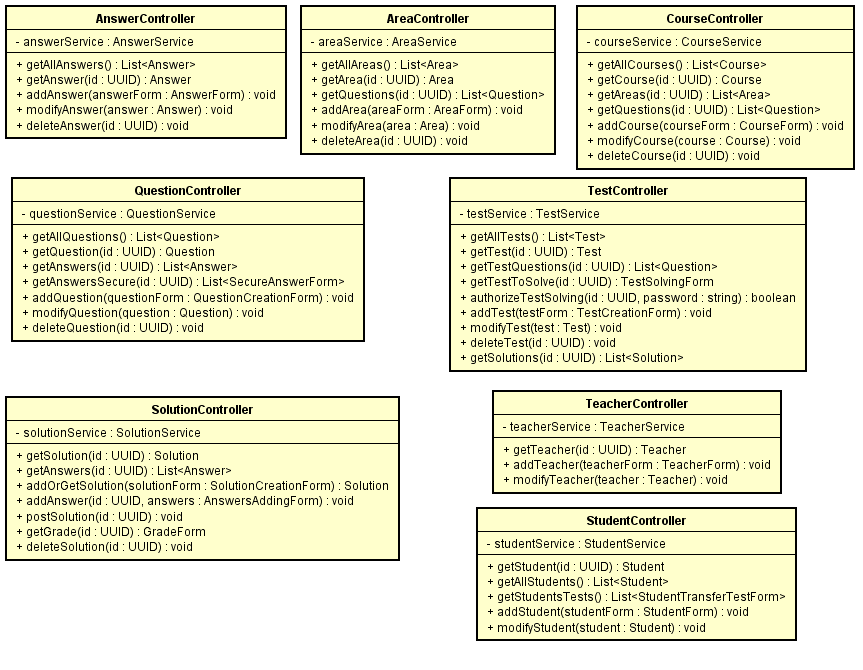
 Rys 4.2. Diagram klas pakietu model

W pakiecie „service”, na Rys 4.3. opracowane zostały klasy zajmujące się obsługą logiki biznesowej. Pakiet „service” korzysta z pakietu „repository”, którego uszczegółowienie nie zostało przedstawione na diagramie klas, ponieważ znajdują się w nim jedynie interfejsy pozwalające na operacje na danych znajdujących się w bazie danych. Klasy w pakiecie „service” są klasami łączącymi klasy, które pozwalają na dokonywanie operacji na danych z klasami, które przedstawiają operacje, które jest w stanie wykonać pakiet „backend”

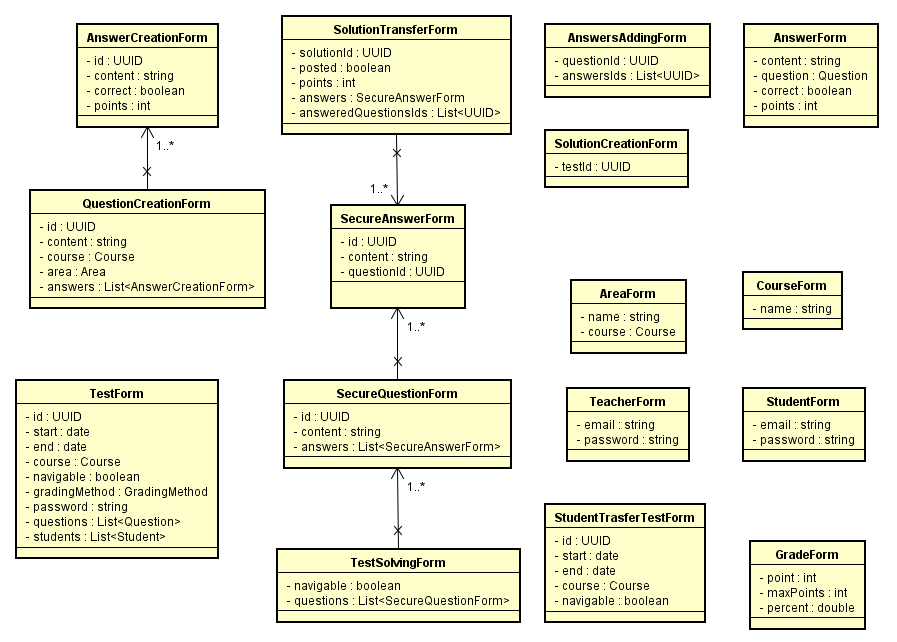


Rys 4.3. Diagram klas pakietu service

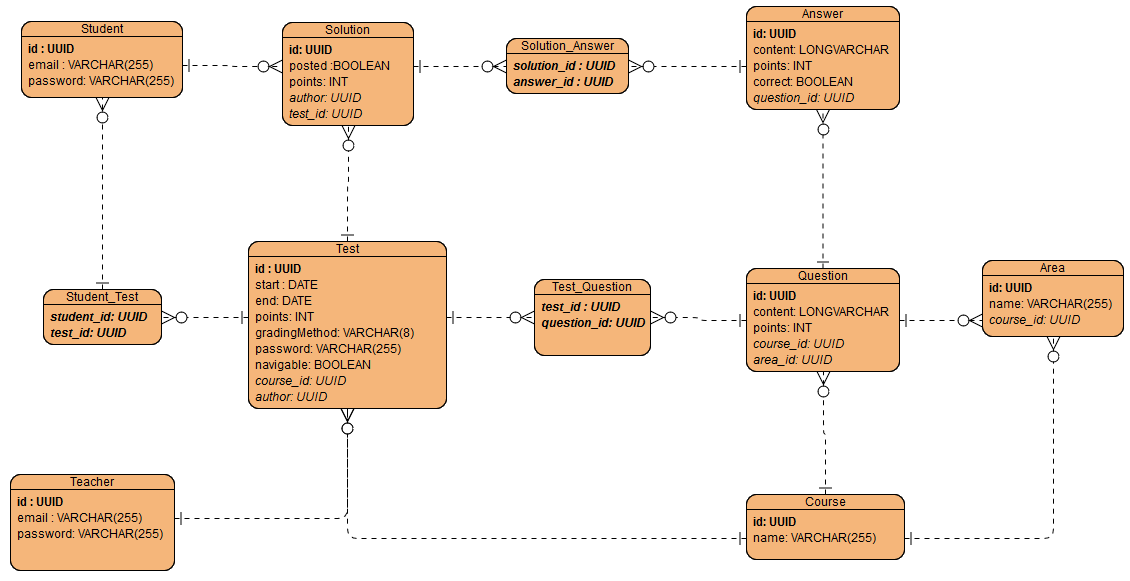
Właśnie w pakiecie „controller” (Rys 4.4.) znajdują się klasy, które definiują API, czyli „application programming interface”, co oznacza interfejs programowania aplikacji. Definiuje on to jakie operacje pozwala wykonać nasza aplikacja. [18] Jest to interfejs, za pomocą którego komponenty widoku z pakietu „frontend” komunikować będą się z aplikacją znajdującą się w pakiecie „backend”. Dzięki wykorzystaniu API, do skorzystania z usług oferowanych przez opracowywaną aplikację nie będzie wymagana znajomość klas, które oferują dane usługi, a jedynie adresu URL na serwerze oraz metody HTTP. Zapewni to przezroczystość dla klienta.

 Rys 4.4. Diagram klas pakietu controller

Na ostatnim (Rys 4.5.) diagramie klas przedstawiony jest pakiet „dto”, który modeluje klasy definiujące strukturę obiektów używanych w komunikacji za pomocą API. Przede wszystkim są to klasy, które ukrywają część atrybutów, natomiast w użyciu są również klasy definiujące strukturę obiektów, które posiadają mniejszą liczbę atrybutów, pomijając atrybut „id”, który jest automatycznie generowany. Zapobiega to tworzeniu obiektów o jednakowych identyfikatorach.

 Rys 4.5. Diagram klas pakietu dto

**4.2. Baza danych** Baza danych, wykorzystana w systemie jest bazą danych relacyjną, oznacza to, że na etapie projektowania należy sporządzić jej relacyjny model. Model relacyjnej bazy danych nazywany jest diagramem ERD od angielskiego akronimy „Entity Relation Diagram”, który na język polski tłumaczone jest jako diagram związków encji. Przedstawione na nim są encje posiadające swoją nazwę oraz atrybuty wraz z ich typami oraz związki pomiędzy nimi. [16] Przedstawiony na Rys 4.6. diagram wykonany jest na podstawie modelu domenowego, który był wynikiem przeprowadzonej specyfikacji wymagań. Przedstawione reguły biznesowe i ograniczenia dziedzinowe dotyczą również diagramu ERD bazy danych. Natomiast na diagramie ERD pojawiają się również klucze obce, których obowiązkowość definiowana jest przez obowiązkowość udziału encji w związku. Co więcej na diagramie ERD przedstawione zostały encje pośredniczące w związkach N:N.

 Rys 4.6. Diagram ERD

**5. Implementacja**

Proces implementacji aplikacji polega na przetłumaczeniu wyspecyfikowanych wymagań na działający program. [20] Podczas tego procesu realizuje się modele przedstawione w architekturze aplikacji. Wynikiem tego procesu ma być działający, spełniający wymagania program. Jest to realizacyjna faza procesu wytwarzania oprogramowania.

**5.1. Przykład implementacji funkcjonalności** Poniżej przedstawiona zostanie implementacja pojedynczej funkcjonalności systemu. Jest to funkcjonalność pozwalająca studentowi na przeglądanie przez *Studenta* testów, do których został przypisany.

5.1.1. Implementacja komponentu

Szkielet programistyczny Vue.js pozwala na realizowanie aplikacji webowej w postaci SPA. Oznacza to, że posiadamy zdefiniowaną strukturę strony, a jej zawartość podczas nawigacji zmienia się dynamicznie. Przedstawiony na Rys 5.1. kod źródłowy aplikacji przedstawia właśnie zdefiniowaną strukturę strony. Widoczny jest wykorzystany komponent „AppHeader”, który definiuje wygląd i zachowanie nagłówka aplikacji. Komponent „Navigation” dotyczy paska nawigacji aplikacji. Ostatnim widocznym elementem jest element „<router-view/>”. Właśnie ten element pozwala na dynamiczną zamianę zawartości aplikacji, zależnie w jakim miejscu w aplikacji znajduje się użytkownik, ten element będzie odpowiednio podmieniony. [30]

<**template**>  
 <**div id="app"**>  
 <**AppHeader**></**AppHeader**>  
 <**Navigation**></**Navigation**>  
 <**router-view**/>  
 </**div**>  
</**template**>

Rys 5.1. Struktura strony

Jednym z takich komponentów jest komponent „StudentsTests”, który przedstawia testy, do których przypisany jest aktualnie zalogowany *Student*. Plik zawierający komponent Vue.js składa się z trzech elementów. W znaczniku <template> znajduje się kod źródłowy przedstawiający strukturę widoku napisany za pomocą języka HTML. W zakresie znacznika <script> znajduje się logika widoku napisana w języku JavaScript, natomiast w znaczniku <style> znajdują się arkusze styli dotyczące tego widoku.[30]   
 W zakresie znacznika <template> znajduje się element <md-table>, który jest tabelą dostępną z biblioteki programistycznej vue-material [15]. Jest to biblioteka programistyczna udostępniająca gotowe komponenty możliwe do wykorzystania w plikach .vue. Tabela ta za pomocą atrybutu v-model pozwala na splątanie jej zawartości z zawartością tablicy zadeklarowanej w zawartości danych (data) znajdujących się w zakresie znacznika <script>. W skład wspomnianej tabeli wchodzą elementy „<md-table-toolbar>”, „<md-table-row>” oraz <md-table-cell>”, które odpowiednio odpowiadają nagłówkowi, wierszowi oraz komórce tabeli. W komórkach tabeli znajdują się splątane elementy zamodelowanej tablicy. Część z nich zostaje przetworzona przez funkcje z sekcji skryptowej pliku. W jednej z komórek znajdują się przyciski, których widoczność, określona atrybutem v-if, jest splątana z wynikiem funkcji określającej czy obecny czas jest już po czasie końca testu. Jeżeli nie, widoczny jest przycisk „Rozwiąż”, który jest nieaktywny (disabled), jeżeli obecny czas jest wcześniejszy niż czas rozwiązywania testu. Jeżeli tak, widoczny jest przycisk „Wynik”, który pozwala zobaczyć wynik z danego testu. [30]

W zakresie znacznika <script> znajduje się logika sterująca widokiem, zdefiniowana za pomocą skryptowego języka Javascript. Komponent ten korzysta z dwóch bibliotek programistycznych. Biblioteka „axios” [29] pozwala na tworzenie zapytań HTTP oraz obsługę otrzymanych odpowiedzi, natomiast biblioteka moment udostępnia narzędzia ułatwiające obsługiwanie dat. Komponent ten wykorzystuje również „router”, który umożliwia mu dokonywanie przekierowań. Jak wspomniane zostało wcześniej „data” przechowuje dane obsługiwane przez widok. W sekcji „mounted”, czyli „zamontowane” znajdują się wywołania funkcji wywoływanych w momencie przekierowania do tego widoku. Kolejno w sekcji „methods”, czyli metody znajdują się funkcje dostępne w danym komponencie. [30] Funkcja „fetchStudentsTests”, czyli pobierz testy studenta, pobiera dane z adresu „api/students/tests/mine” oraz podaje nagłówek z tokenem autoryzacyjnym użytkownika. Za pomocą tych zapytań przebiega komunikacja pakietu „frontend” z pakietem „backend”. Kolejno ustawia tablicę testów dostępną w „data” na posortowane dane odpowiedzi otrzymanej na zapytanie.

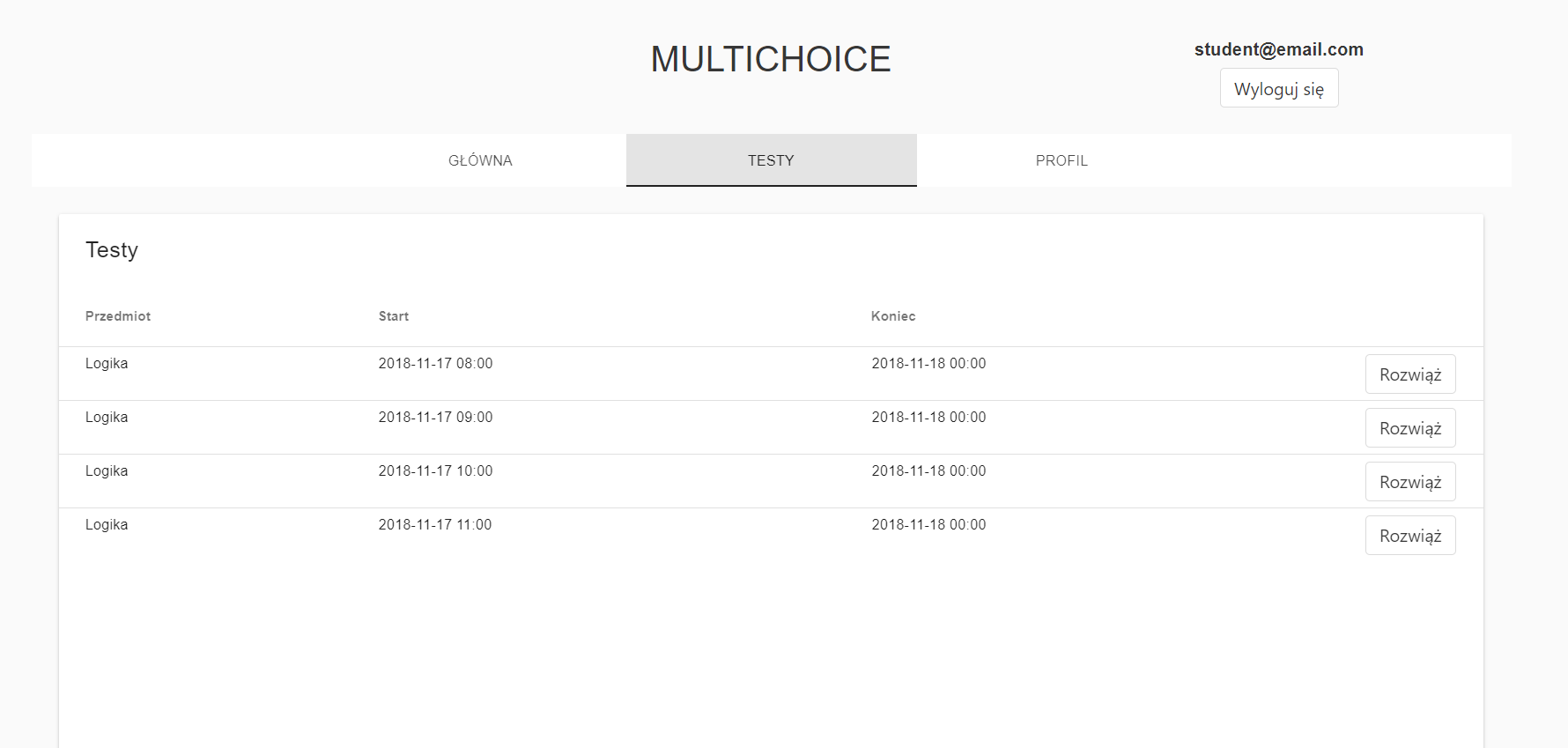
W zakresie znacznika <style> znajdują się arkusze styli wykorzystane w danym komponencie. Znajduje się tam jedynie jedna, prosta klasa CSS, jednak pomimo to uzyskany widok, widoczny na Rys 5.3. nie wygląda jakby nie posiadał żadnego stylu. Spowodowane jest to wykorzystaniem biblioteki programistycznej „bulma” [27], która definiuje zestaw gotowych klas języka CSS do wykorzystania programiście. Jej wykorzystanie można zobaczyć w użyciu w zakresie znacznika <template> w atrybutach „class” znaczników. Klasa „containter”, czyli kontener wyśrodkowuje zawartość znacznika i nadaje mu boczne marginesy. Klasa „button” nadaje elementowi wygląd inny niż ten generowany przez przeglądarki dla przycisku nie posiadającego żadnego stylu.

<**template**>  
 <**div class="container padding-class"**>  
 <**md-table v-model="tests" md-card md-fixed-header**>  
 <**md-table-toolbar**>  
 <**h1 class="md-title md-toolbar-section-start"**>Testy</**h1**>  
 </**md-table-toolbar**>  
 <**md-table-row slot="md-table-row" slot-scope="{ item }"**>  
 <**md-table-cell md-label="Przedmiot"**>{{ ***item***.**course**.**name** }}</**md-table-cell**>  
 <**md-table-cell md-label="Start"**>{{convertDate(***item***.**start**)}}</**md-table-cell**>  
 <**md-table-cell md-label="Koniec"**>{{convertDate(***item***.**end**)}}</**md-table-cell**>  
 <**md-table-cell width="50"**>  
 <**button class="button" v-if="**!isAfterSolvingTime(***item***)**"  
 :disabled="**!isSolvingTime(***item***)**" v-on:click="**routeToTestSolving(***item***)**"**>  
 Rozwiąż  
 </**button**>  
 <**button class="button" v-if="**isAfterSolvingTime(***item***)**"**

**v-on:click="**routeToTestSolving(***item***)**"**>  
 Wynik  
 </**button**>  
 </**md-table-cell**>  
 </**md-table-row**>  
 </**md-table**>  
 </**div**>  
</**template**>  
  
<**script**>  
**import** axios **from 'axios'  
import** \* **as** moment **from 'moment'  
import** {router} **from '../router/index'  
  
export default** {  
 **name**: **'StudentsTests'**,  
 data () {  
 **return** {  
 **tests**: []  
 }  
 },  
 mounted () {  
 **this**.fetchStudentsTests()  
 },  
 **methods**: {  
 fetchStudentsTests () {  
 axios  
 .get(**'/api/students/tests/mine'**, {**headers**: {**'Authorization'**: ***sessionStorage***.getItem(**'user-token'**)}})  
 .then(response => (**this**.sortTests(response.**data**)))  
 },  
 sortTests (tests) {  
 **this**.**tests** = tests.sort((a, b) =>

(a.**start** < b.**start**) ? 1 : ((b.**start** > a.**start**) ? -1 : 0))  
 },  
 convertDate (dateJavaFormat) {  
 *// 2018-10-19T10:00:00.628+0000* **return** *moment*(dateJavaFormat).format(**'YYYY-MM-DD HH:mm'**)  
 },  
 isSolvingTime (test) {  
 **return** *moment*().isBetween(test.**start**, test.**end**)  
 },  
 routeToTestSolving (test) {  
 router.push(**'/test/solving/'** + test.**id**)  
 },  
 isAfterSolvingTime (test) {  
 **return** *moment*().isAfter(test.**end**)  
 }  
 }  
}  
</**script**>  
  
<**style scoped**>  
 .**padding-class** {  
 **padding**: 25**px**;  
 }  
</**style**>

Rys. 5.2. Implementacja komponentu

 Rys 5.3. Wynik implementacji komponentu

5.1.2. Implementacja kontrolera

Kontroler, czyli klasa, za pomocą której pakiet „frontend” komunikuje się z pakietem „backend”, opisana jest adnotacją ze szkieletu programistycznego Spring „@RestController”. Adnotacja ta oznacza, że dana klasa jest oznaczona jak kontroler oraz, że każda z metod znajdujących się w niej oznaczona adnotacją „@RequestMapping” zwraca obiekt odpowiedzi HTTP. [24] Adnotacja ta, oznacza, że dana klasa jest traktowana jako komponent szkieletu Spring, więc może zostać wstrzyknięta za pomocą adnotacji „@Autowired”. [31]

Fragmenty klasy kontrolera „StudentController” , przedstawione na Rys 5.4. oznaczona jest również adnotacją „@RequestMapping”, odpowiadającą za mapowanie żądań przychodzących do aplikacji na podany w argumencie adres URL właśnie w zakres działania tej klasy. [31] W zakresie tej klasy widoczne jest również pole „studentService”, którego zależność jest wstrzykiwana przez szkielet Spring, dzięki czemu programista nie musi brać na siebie odpowiedzialności tworzenia obiektu danej klasy.

Metoda, której wykonanie zostało zmapowane przez żądanie HTTP przychodzące na adres URL „/api/students/tests/mine” z metodą GET, oznaczona jest dodatkowo adnotacją „@PreAuthorize”, do której podany jest argument sprawdzający autoryzację wywołania. Adnotacja ta odpowiada za dostęp do metody przed jej wywołaniem. [31]

Sama metoda odpowiadająca adresowi i metodzie żądania odpowiedzialna jest jedynie za wywołanie odpowiedniej metody z warstwy logiki biznesowej oraz dodania odpowiedniego statusu HTTP do odpowiedzi na żądanie.

@RestController  
@RequestMapping(**"/api/students"**)  
**public class** StudentController {  
  
 @Autowired  
 StudentService **studentService**;  
  
...  
 @PreAuthorize(**"hasAuthority('STUDENT')"**)  
 @GetMapping(**"/tests/mine"**)  
 **public** ResponseEntity getStudentsTests(){  
 List<StudentTransferTestForm> tests = **studentService**.findTestsOfStudent();  
 **if** (tests == **null**) {  
 **return new** ResponseEntity(HttpStatus.***NOT\_FOUND***);  
 }  
 **return** ResponseEntity.*ok*(tests);  
 }  
...  
 }  
}

Rys 5.4. Implementacja kontrolera   
  
5.1.3. Implementacja serwisu i DTO

Serwis, czyli klasa, która implementuje logikę biznesowa aplikacji oznaczona jest adnotacją „@Service”, która podobne jak adnotacja „@RestController” również rejestruje daną klasę jako komponent szkieletu Spring. Oprócz tego jedynie wskazuje ona, że dana klasa oferuje usługi, które można wykorzystać.

Fragmenty klasy implementującej serwis wykorzystywany przez kontroler z poprzedniego podrozdziały przedstawione są na Rys 5.5. Do świadczenia usługi wywoływanej przez kontroler serwis wykorzystuje wstrzyknięte zależności do repozytorium oraz do innego serwisu. Metoda „findTestsOfStudent”, która jest wywoływana przez kontroler pobiera przez serwis zajmujący się testami, którego fragment przedstawiony jest na Rys 5.6. Serwis ten aby pobrać wszystkie testy korzysta z metody udostępnionej przez repozytorium. Otrzymane testy następnie są filtrowane aby uzyskać jedynie te testy, do których przypisany jest student, który znaleziony został na podstawie danych autentykacyjnych. W kolejnym kroku uzyskane testy mapowane są na DTO. Implementacja DTO przedstawiona jest na Rys 5.7. Klasa definiująca strukturę DTO oznaczona jest adnotacją „@Data”, jest to adnotacja pochodząca z biblioteki programistycznej Lombok. Adnotacja ta pozwala na automatyczne generowanie getterów, setterów, metody equals, toString oraz hashCode. [26] Oprócz tego klasa ta posiada konstruktor przyjmujący obiekt testu. Klasa ta zastosowana jest w celu ukrycia części informacji dostępnych w obiektach klasy Test, które nie powinny być dostępne dla studenta.

@Service  
**public class** StudentService {  
   
 @Autowired  
 StudentRepository **studentRepository**;

...   
 @Autowired  
 TestService **testService**;  
  
...  
 **public** List<StudentTransferTestForm> findTestsOfStudent() {  
 Student student = getStudentFromAuthentication();  
 List<StudentTransferTestForm> studentsTests = **testService**.findAll()  
 .stream()  
 .filter(test -> test.getStudents().contains(student))  
 .map(StudentTransferTestForm::**new**)  
 .collect(Collectors.*toList*());  
 **return** studentsTests;  
 }  
   
 **public** Student getStudentFromAuthentication(){  
 Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  
 Student student = **studentRepository**.findByEmail(authentication.getName());  
 **return** student;  
 }  
}

Rys 5.5. Implementacja serwisu studenta

@Service  
**public class** TestService {  
  
 @Autowired  
 TestRepository **testRepository**;  
  
...  
 **public** List<Test> findAll(){  
 **return testRepository**.findAll();  
 }

...  
}

Rys 5.6. Implementacja serwisu testu

@Data  
**public class** StudentTransferTestForm {  
 @NotNull  
 **private** UUID **id**;  
  
 @NotNull  
 **private** Date **start**;  
  
 @NotNull  
 **private** Date **end**;  
  
 @NotNull  
 **private** Course **course**;  
  
 @NotNull  
 **private boolean navigable**;  
  
 **public** StudentTransferTestForm(Test test) {  
 **this**.setId(test.getId());  
 **this**.setStart(test.getStart());  
 **this**.setEnd(test.getEnd());  
 **this**.setCourse(test.getCourse());  
 **this**.setNavigable(test.isNavigable());  
 }  
}

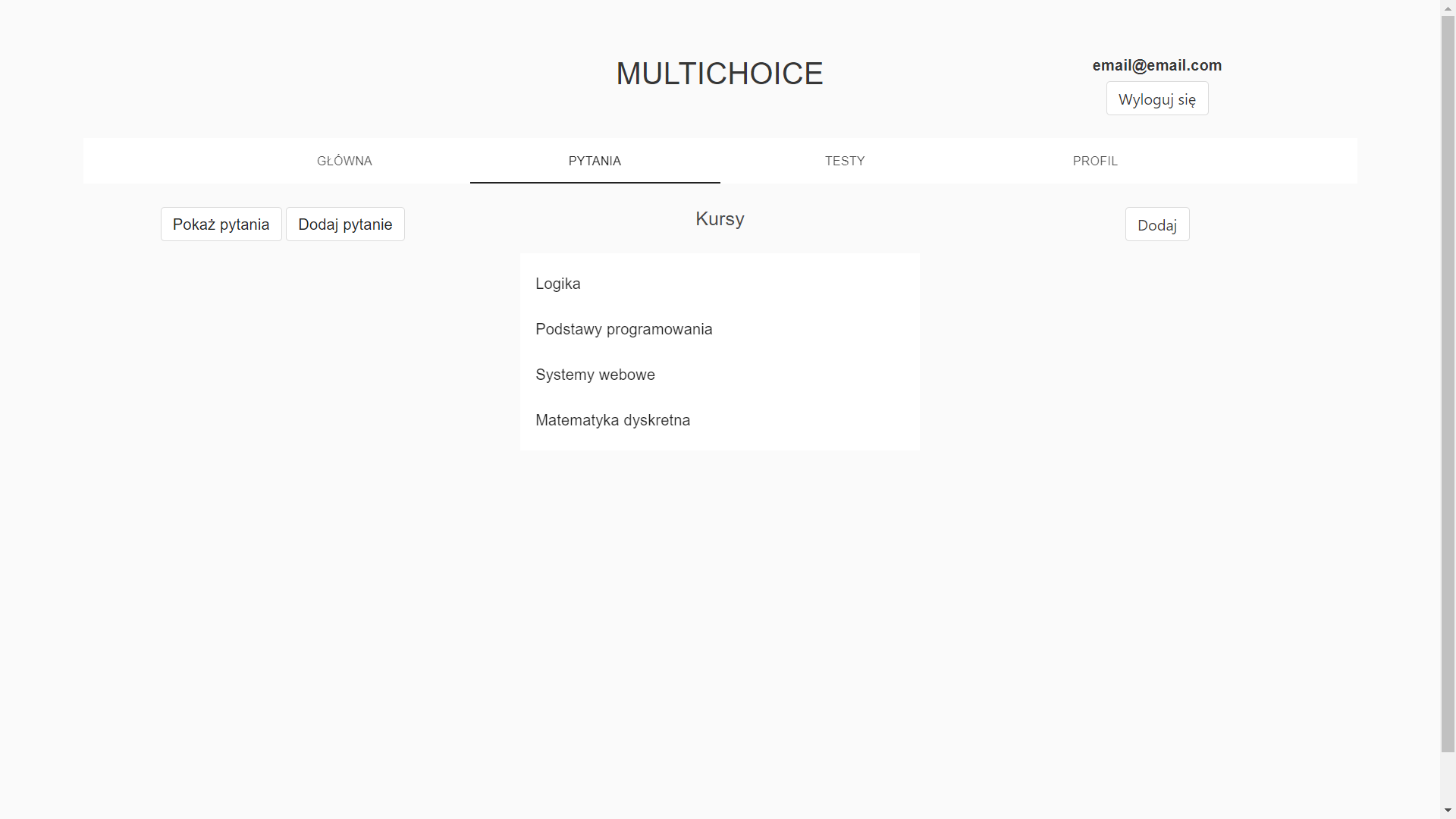
Rys 5.7. Implementacja DTO   
  
5.1.4. Implementacja repozytorium   
 Repozytorium oznaczone jest adnotacją „@Repository”, która również oznacza daną klasę jako komponent szkieletu Spring. Oprócz tego adnotacja ta ma informować o tym, że dana klasa jest odpowiedzialna za operacje na danych. [31] Oprócz tego interfejs ten rozszerza interfejs „JpaRepository”, który jest interfejsem pozwalającym na dostęp do danych oraz udostępnia proste operacje na danych za pomocą Java Persistence API. [23] Java Persistence API jest szkieletem programistycznym pozwalającym na wykorzystanie mapowania obiektowo-relacyjnego, czyli mechanizmu, który pozwala na automatyczne mapowanie obiektów wykorzystywanych w aplikacji na wiersze w tabelach bazy danych. [31]

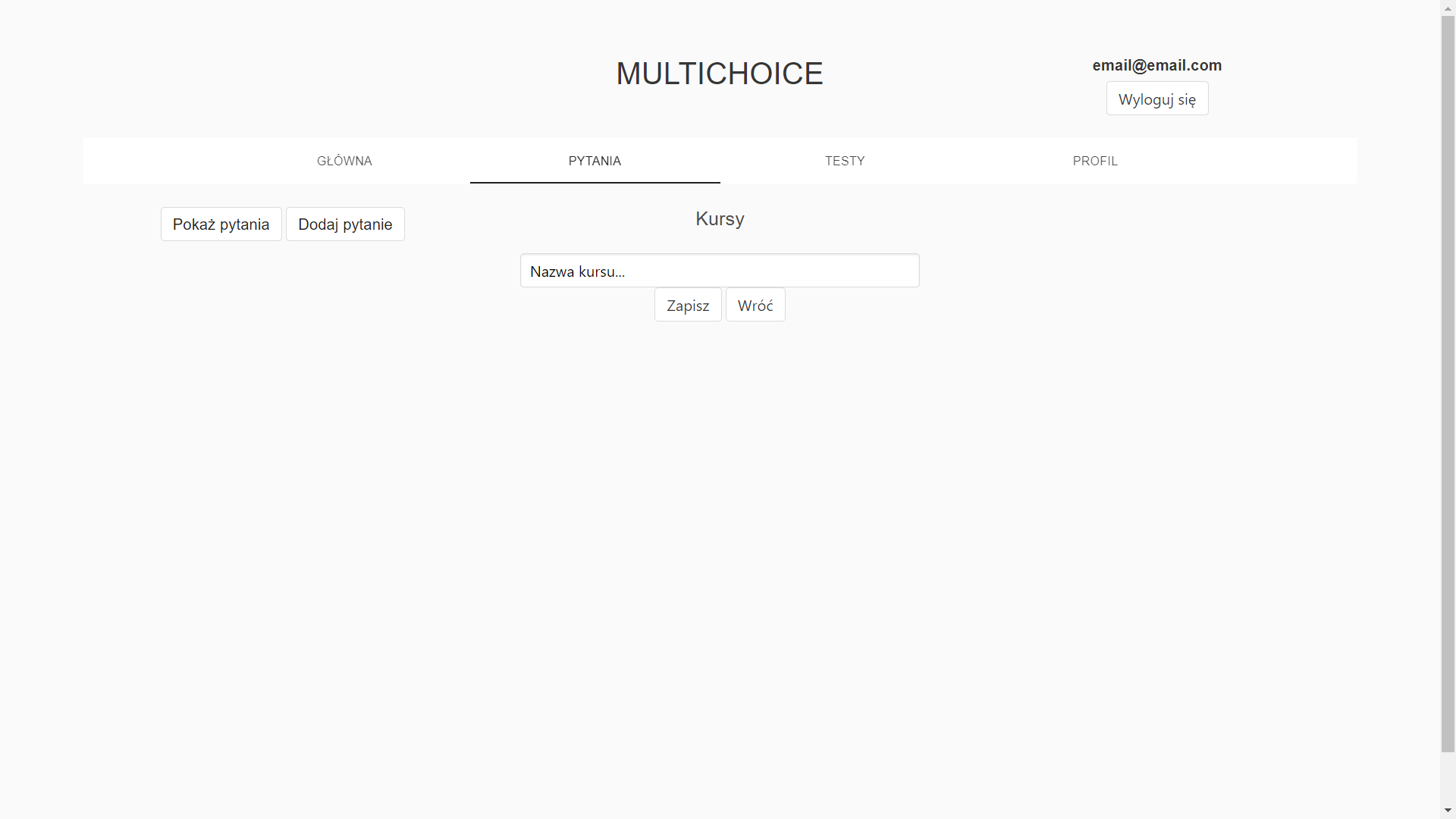
@Repository  
**public interface** TestRepository **extends** JpaRepository<Test, Long> {  
 Test findById(UUID uuid);  
}

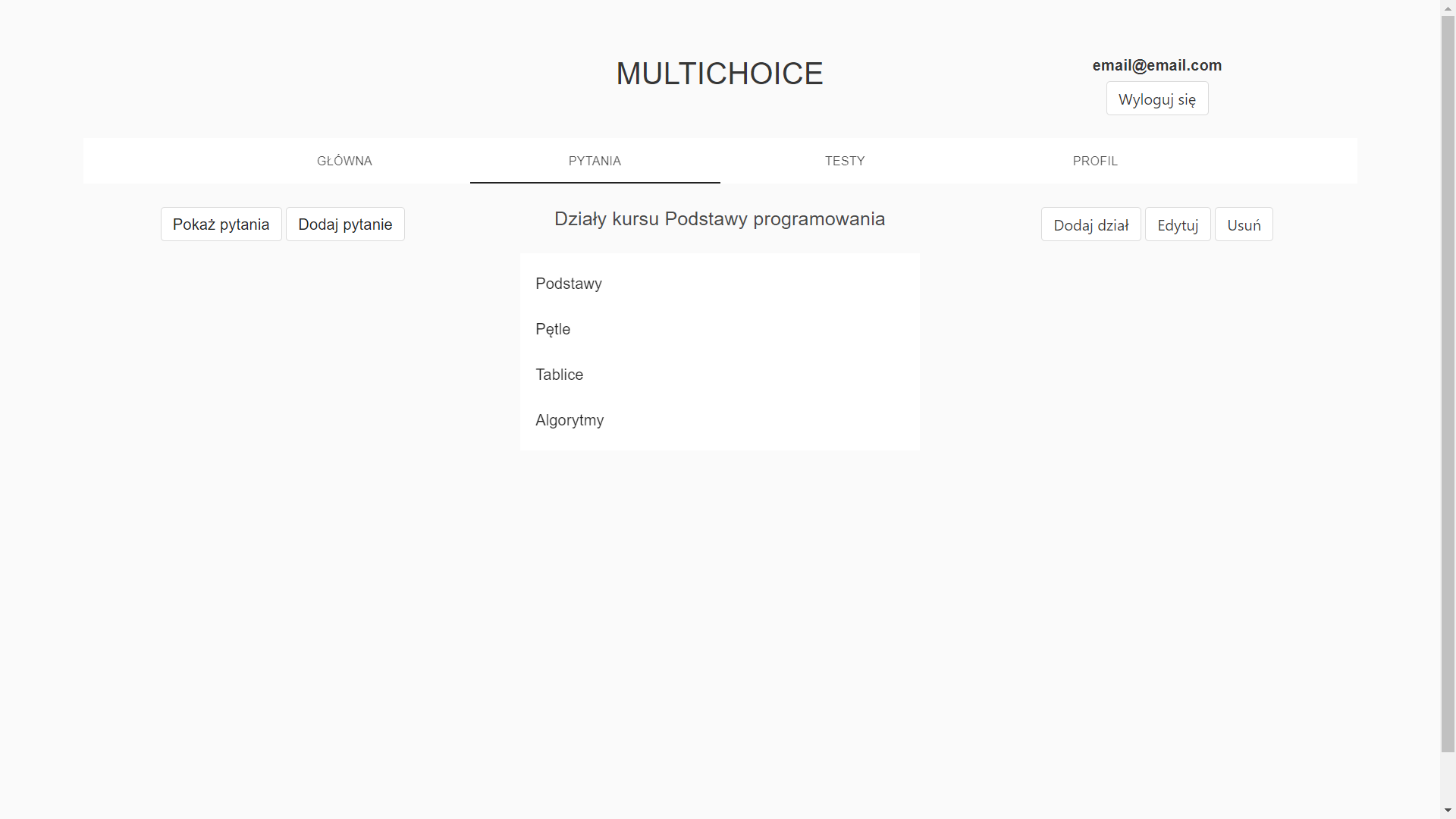
Rys 5.8. Implementacja repozytorium

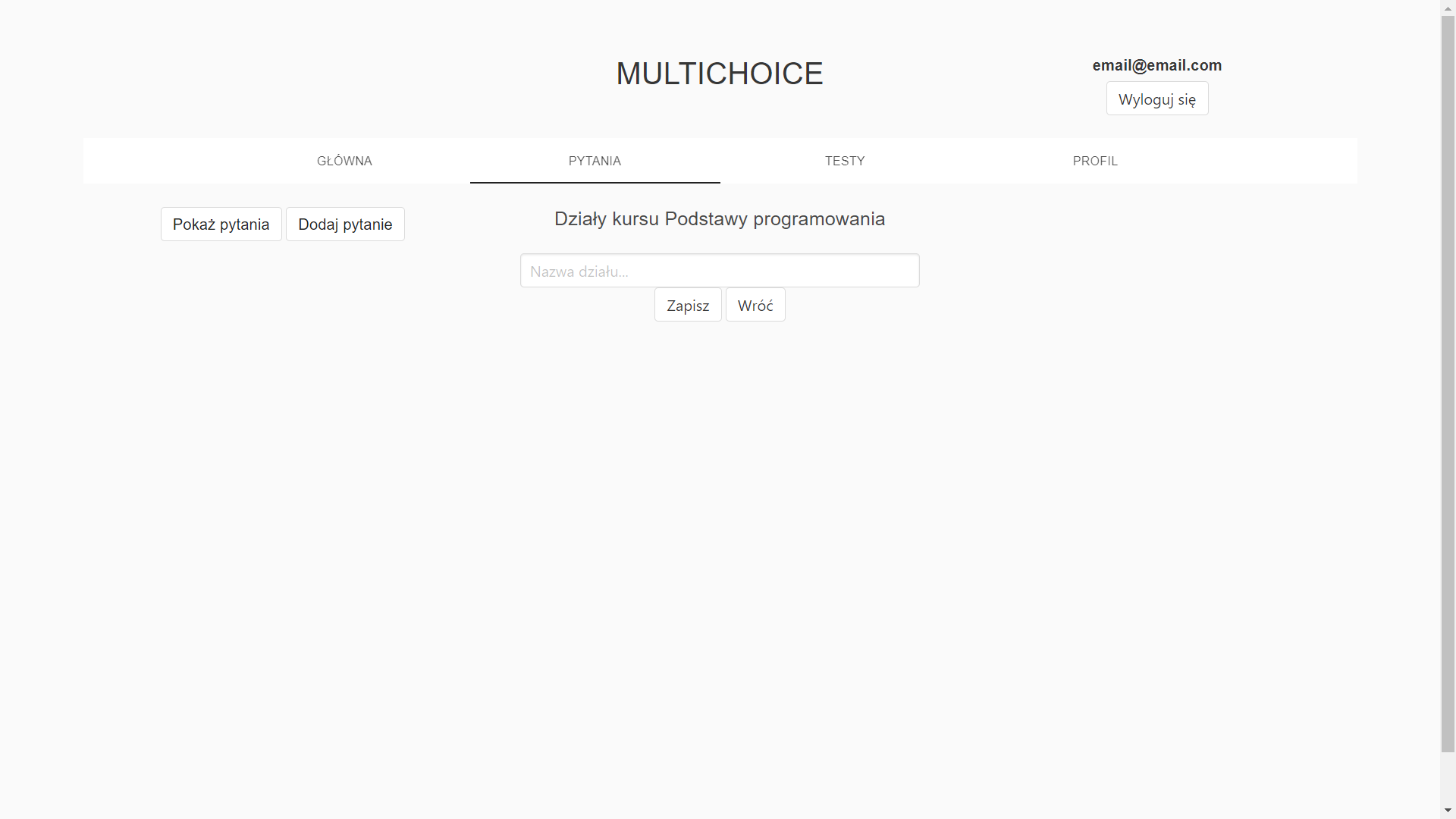
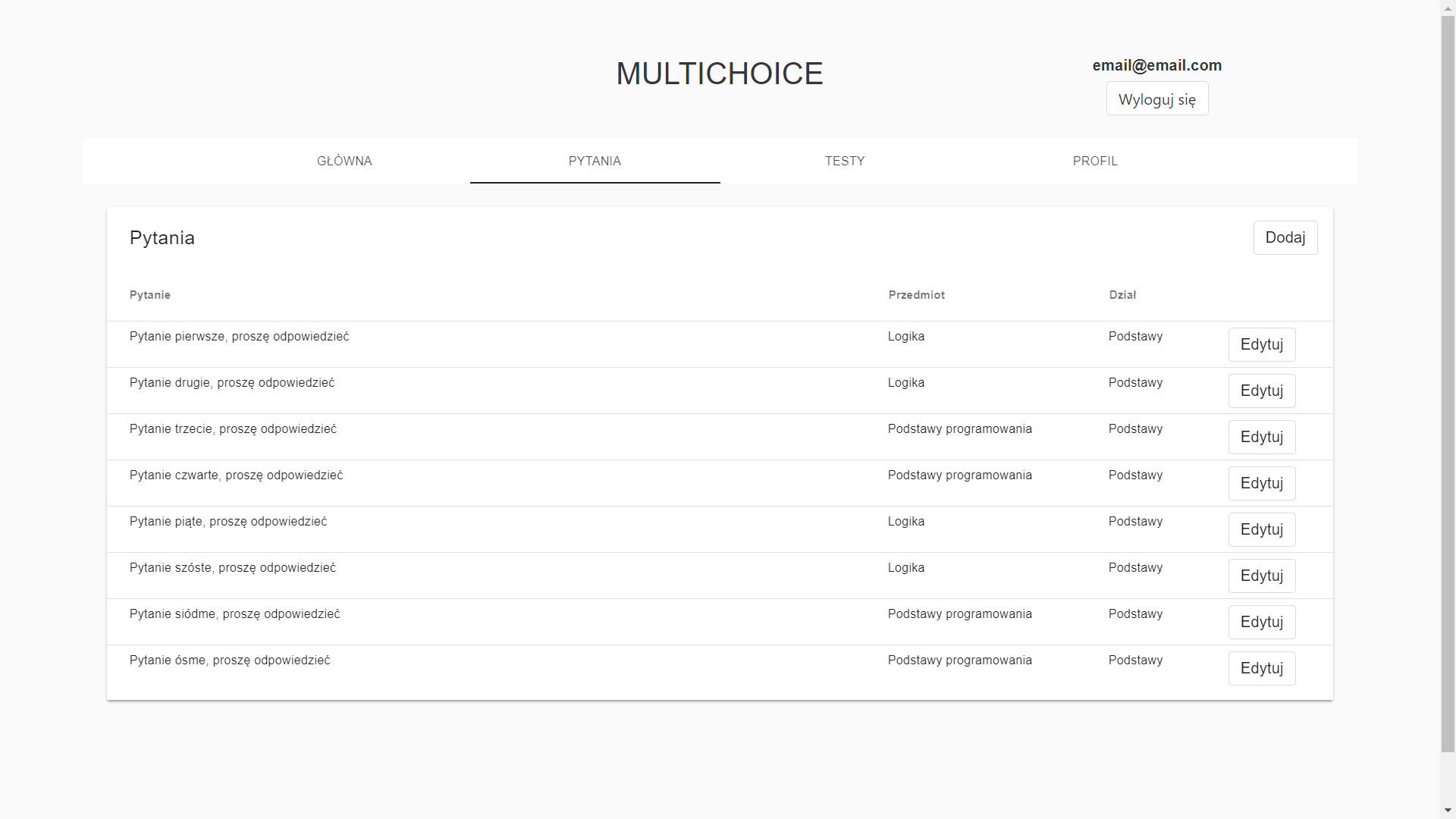
**5.2 Interfejs użytkownika**

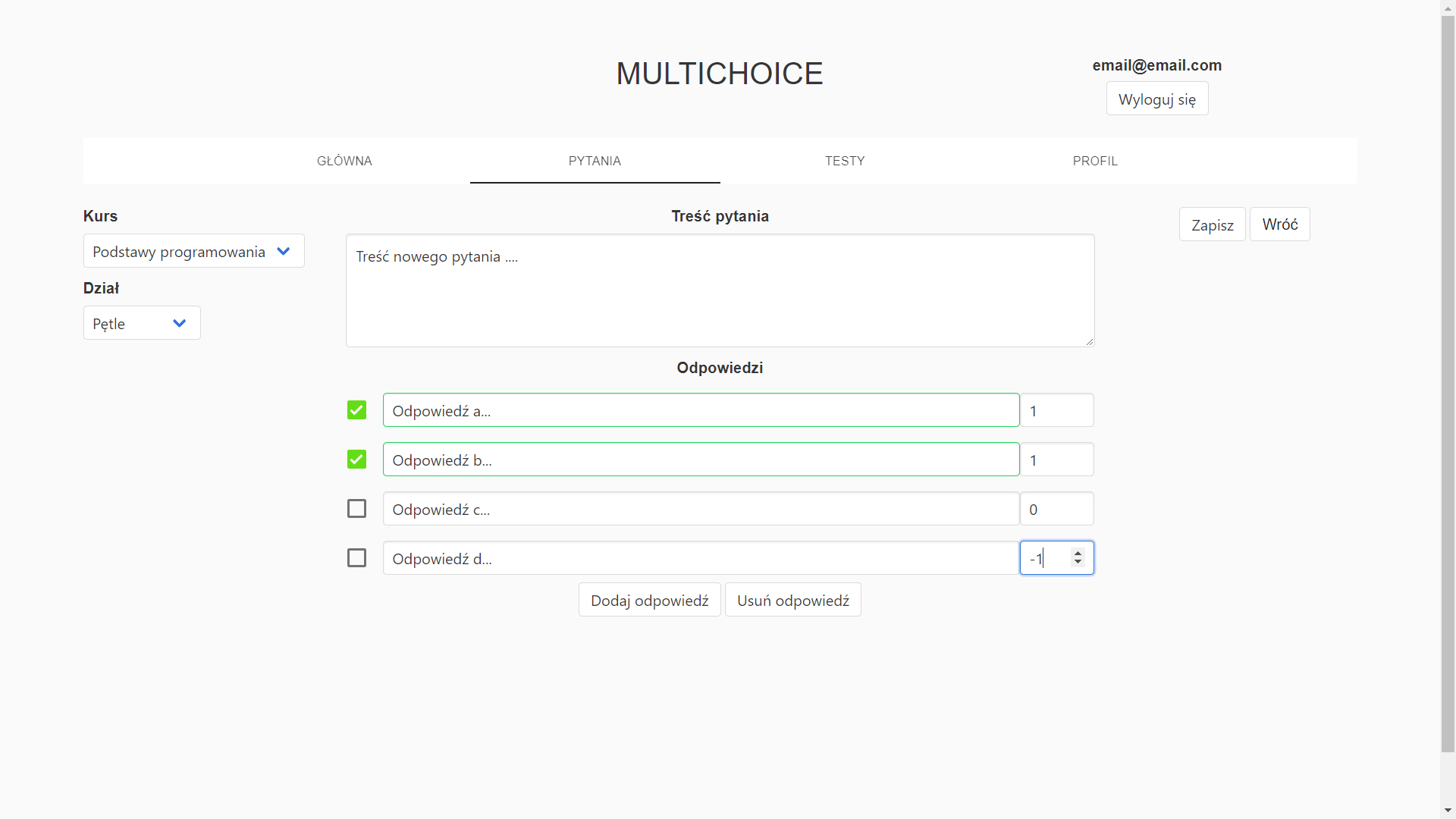
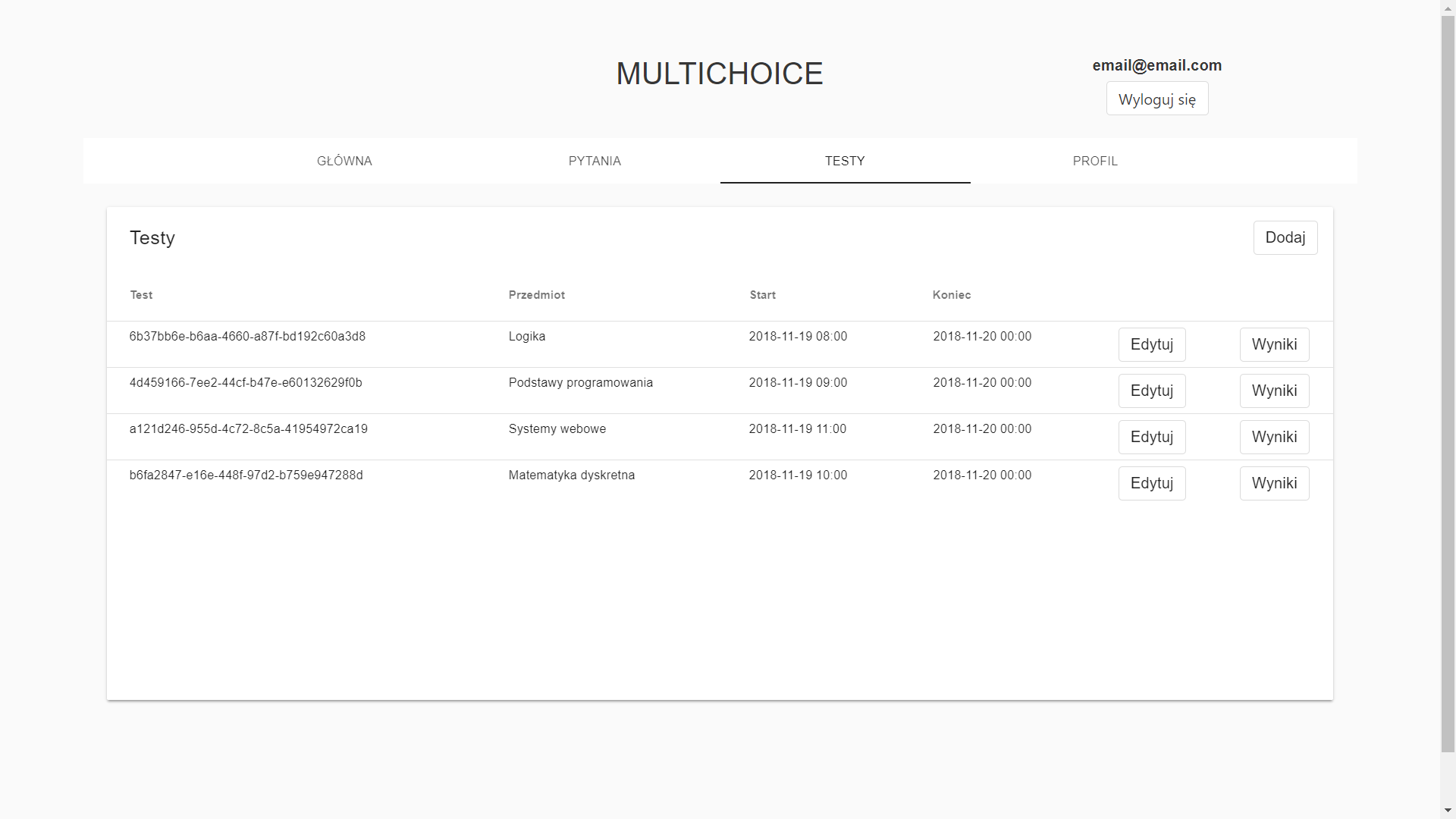
Interfejs użytkownika powstał w oparciu o prototypy interfejsów opracowane przy specyfikacji wymagań opisanej w rozdziale 3. Zaimplementowane widoki zaprezentowane zostały na Rys 5.9. – Rys 5.19.

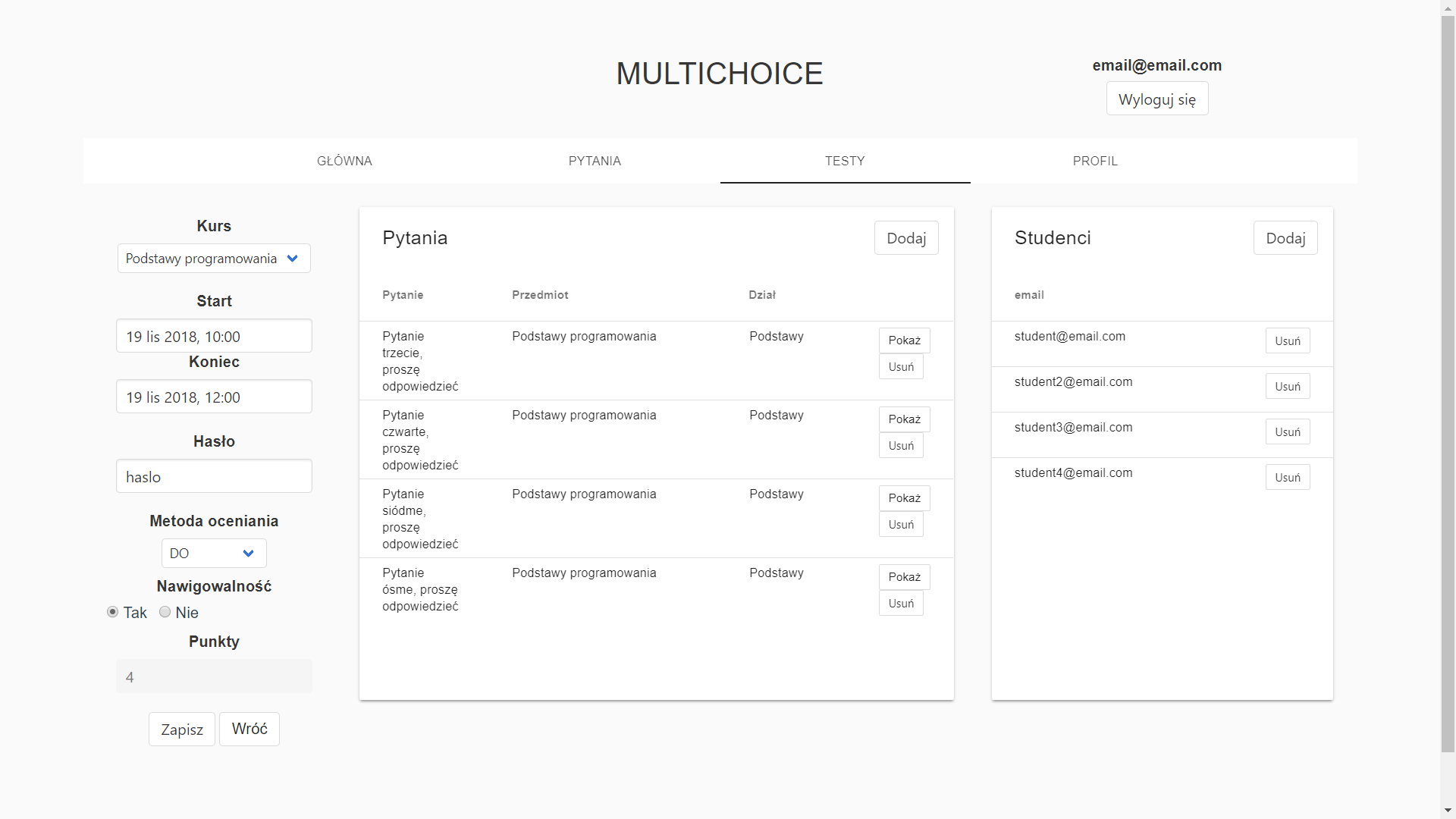
 Rys 5.9. Spis przedmiotów

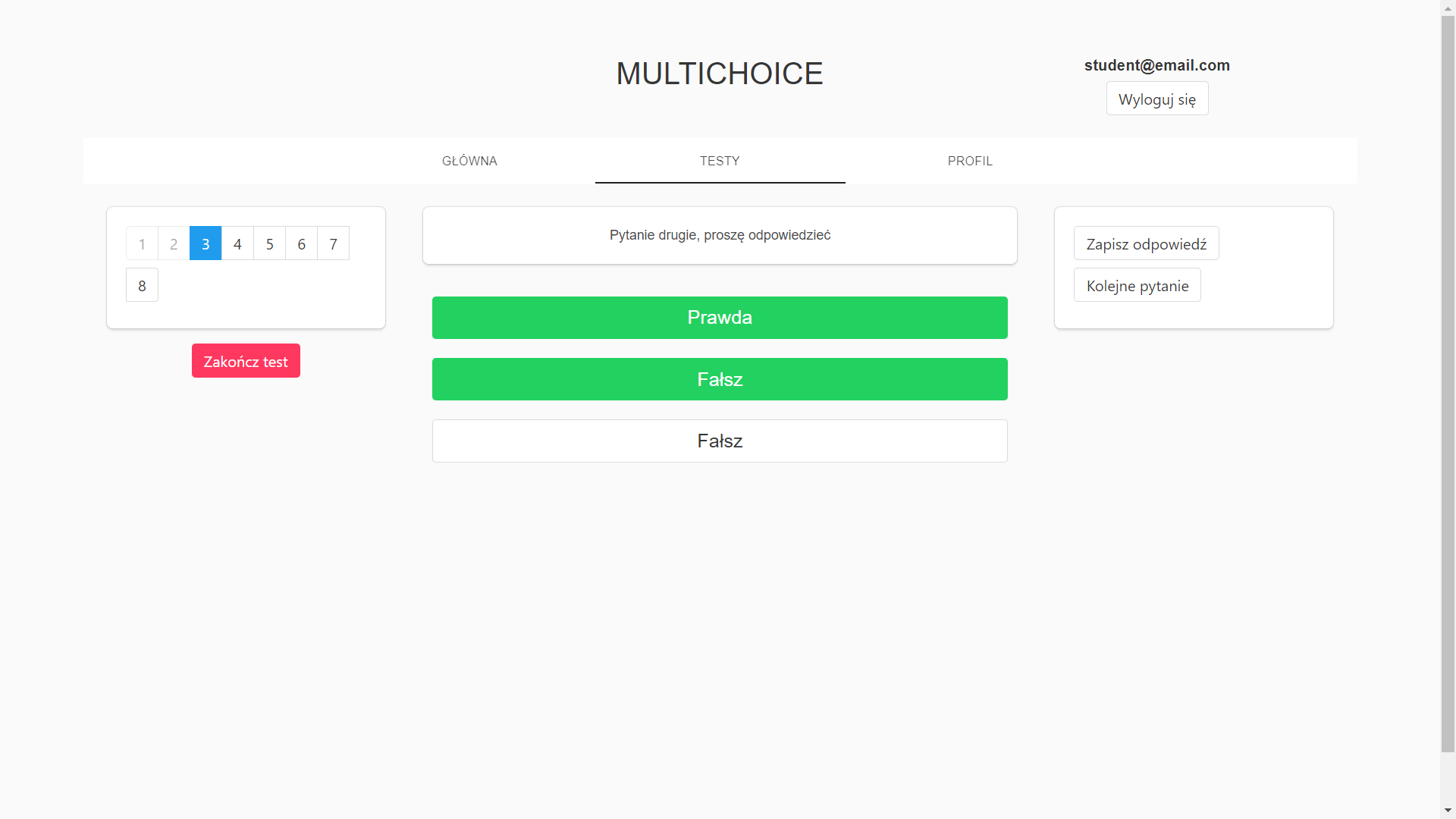
 Rys 5.10. Dodawanie przedmiotu

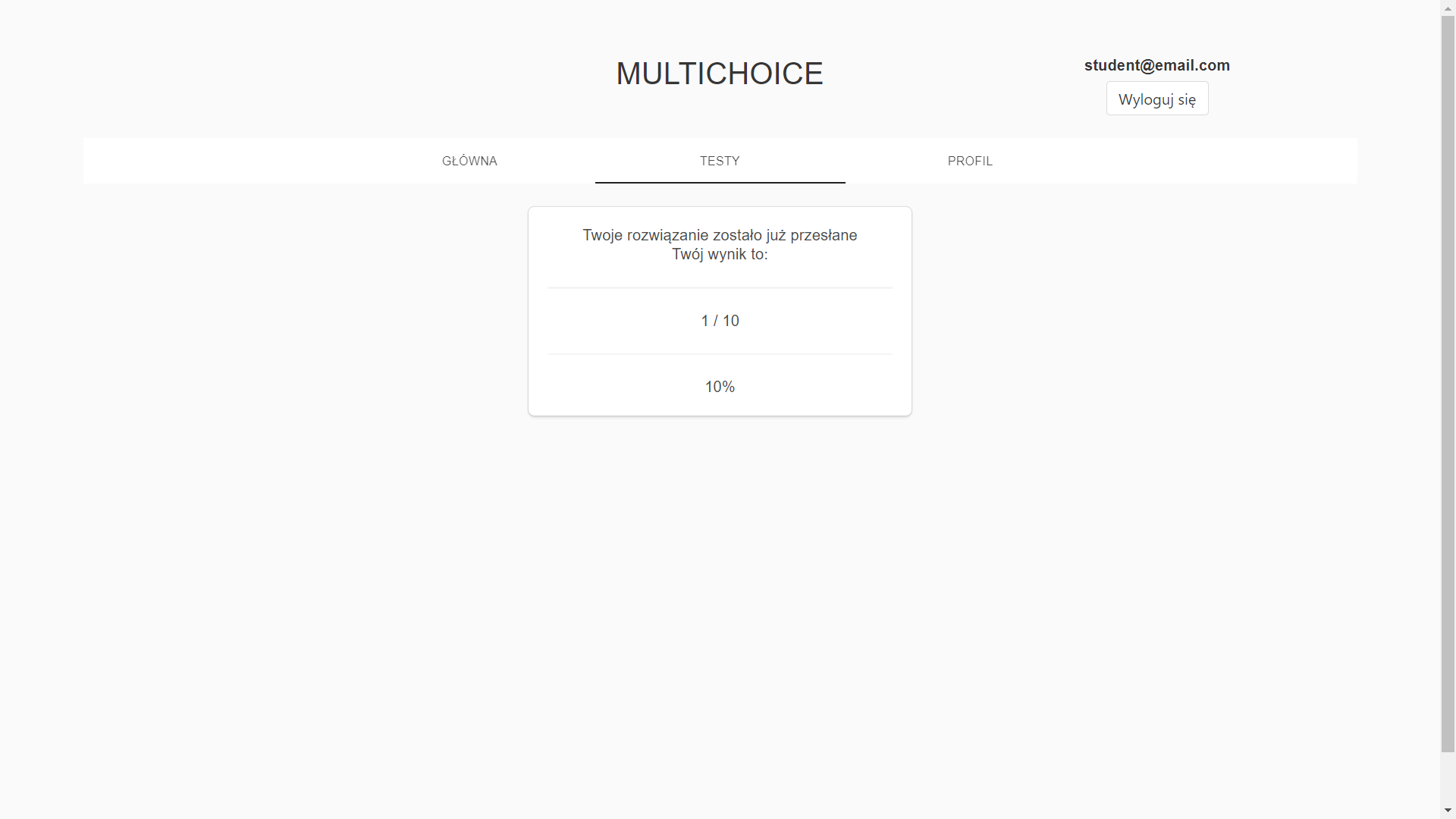
 Rys 5.11. Spis działów przedmiotu

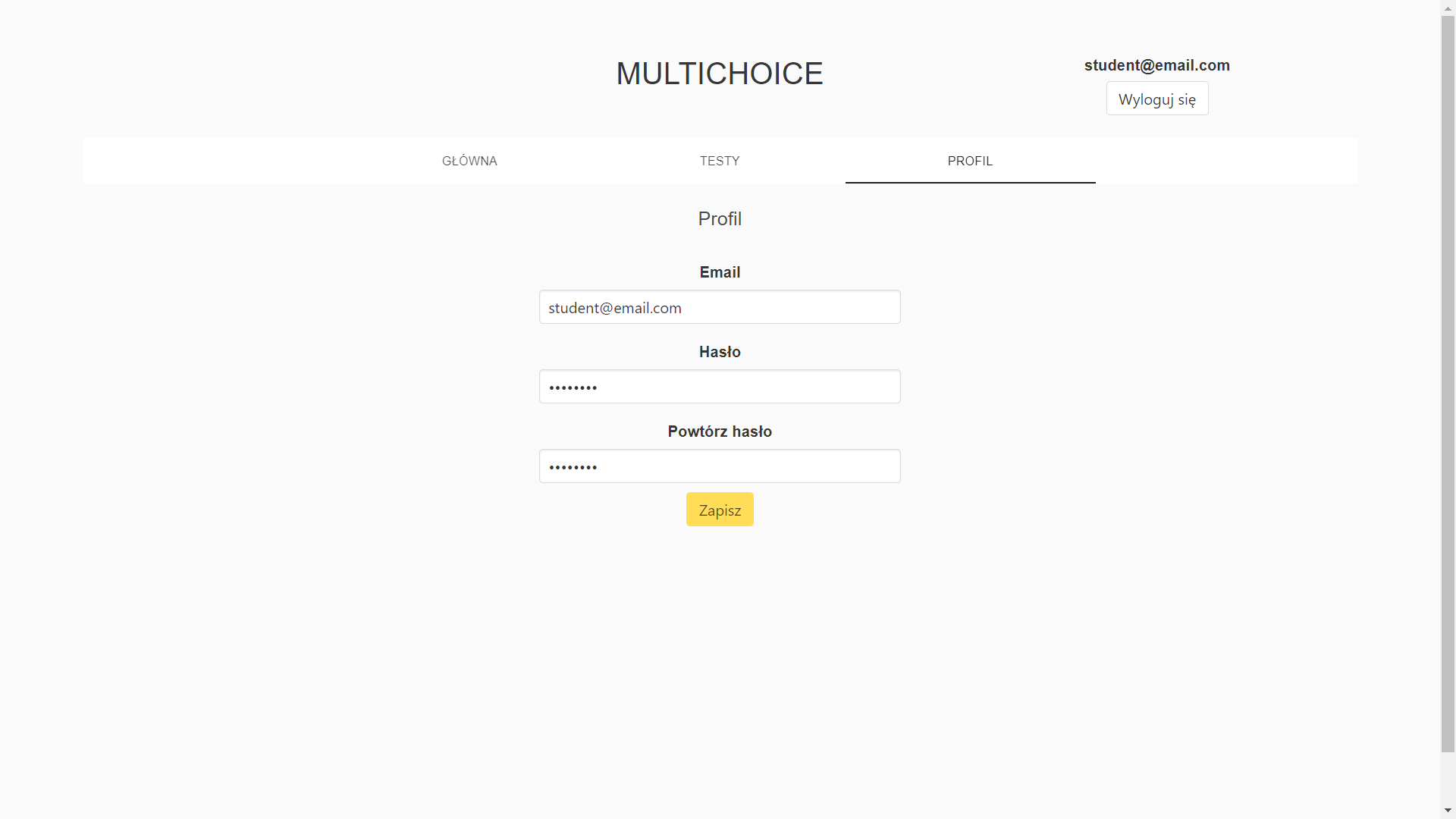
 Rys 5.12. Dodawanie działu przedmiotu  Rys 5.13. Spis pytań

 Rys 5.14. Dodawanie nowego pytania  Rys 5.15. Spis testów

 Rys 5.16. Dodawanie testu

 Rys 5.17. Rozwiązywanie testu.

 Rys 5.18. Wynik testu

 Rys 5.19. Edytowanie profilu

**5.3. Wykorzystane narzędzia programistyczne**

Istotnym etapem realizacji projektu jest odpowiednie dobranie wykorzystywanych narzędzi. Podjęcie tych decyzji ma znaczący wpływ na to w jakim tempie oraz jak komfortowo i efektywnie przebiegać będą prace nad implementacją oprogramowania. Czynnikiem, który może pomóc podjąć decyzję jest to jak dużą społeczność internetową dane narzędzie posiada. Kolejnymi takimi czynnikami mogę być dostępność produktu oraz prostota użytkowania. Kierując się tymi przesłankami wybrane zostały następujące technologie.

5.3.1. Apache Maven

Maven jest narzędziem służącym do zarządzania projektem napisanym w języku Java. Pozwala on na automatyzowanie procesu budowania aplikacji oraz zarządzanie zależnościami. Dzięki wykorzystaniu struktury XML jest czytelny oraz nieskomplikowany w obsłudze. [1]

5.4.1. Git

Git jest zdecentralizowanym systemem kontroli wersji. Jego opracowywanie zaczął Linus Torvalds, autor jądra Linux. Obecnie Git jest rozwijany na zasadach wolnego oprogramowania. Głównym założeniem Gita jest ułatwienie pracy nad projektem, w którym bierze udział wielu programistów. [8] Obecnie jest najszerzej stosowanym systemem kontroli wersji, co spowodowało, że społeczność internetowa jest liczna. Dostępne też jest wiele stron oferujących darmowe repozytoria kodu.

5.4.2. IntelliJ IDEA

IntelliJ jest zintegrowanym środowiskiem programistycznym opracowanym dla języka Java, jednak obecnie wspiera znacznie szerszy zakres języków programowania. Jest flagowym produktem firmy JetBrains, rozwijany jest na zasadach wolnego oprogramowania. [11] Jego istotną zaletą jest wbudowane wsparcie dla narzędzi Maven i Git oraz mechanizmy pozwalające na debugowanie kodu źródłowego mechanizmów Javy 8 oraz jej nowszych wersji. Obsługuje on również mechanizmy szkieletu Spring bez potrzeby instalowania dodatkowych wtyczek.

5.4.2. WebStorm

WebStorm jest zintegrowanym środowiskiem programistycznym zorientowanym na technologie webowe, w szczególności JavaScript. Umożliwia on również obsługę kodu źródłowego HTML oraz CSS. Podobnie jak IntelliJ jest produktem firmy JetBrains. [32] Tak jak swój odpowiednik do języka Java, udostępnia on wsparcie dla systemu kontroli wersji Git. Pozwala on też na debugowanie oraz uzupełnianie kodu JavaScript. Wspiera on szkielet programistyczny Vue.js bez potrzeby pobierania wtyczek.

**6. Testowanie produktu**

Proces implementacji systemu niestety nie przebiega bezbłędnie, a im bardziej skomplikowany system jest dostarczany tym więcej błędów popełnić może programista. Aby weryfikować pracę wykonaną przez programistów potrzebne jest testowanie produktu. Testowanie jest akcją mającą potwierdzić, że system działa poprawnie oraz odkryć luki w oprogramowaniu zanim trafi ono do użytku. Proces testowania ma dwa wyszczególnione cele:

1. Zademonstrowanie, że program spełnia wyspecyfikowane wymagania.
2. Odnalezienie sytuacji, w których oprogramowanie nie zachowuje się odpowiednio.

Aby osiągnąć te cele stosuje się różne techniki testowania na różnych poziomach ziarnistości oprogramowania. [20]

**6.1. Testy jednostkowe**

Podczas testowania jednostkowego, testowane są pojedyncze jednostki oprogramowania. Jednostką oprogramowania nazywa się pojedyncze obiekty lub metody. Kiedy testowaniu poddawany jest obiekt należy przetestować wszystkie możliwe operacje na tym obiekcie oraz sprawdzić każdy możliwy stan obiektu. Aby sprawnie korzystać z możliwości jakie dają testy jednostkowe powinno się je automatyzować. Funkcjonalność automatyzacji testów zapewniają szkielety programistyczne zorientowane na testy. [20] Szkieletem użytym w projekcie jest Spock. Spock jest szkieletem do automatyzacji testów dla języka Java, opartym na skryptowym języku Groovy. [22] Na Rys. 6.1. przedstawiona została przykładowa klasa testowa zawierająca testy jednostkowe serwisu działu kursu oraz przykład implementacji pojedynczego testu. W metodzie „setup”, czyli ustawienie, jak sama nazwa wskazuje ustawiane są pola klasy testowej. Aby odciąć zależność od innych klas aby możliwe było testowanie jednostek, zależności od nich są modelowane (mock). Sam test opracowywany w szkielecie Spock oparty jest na strukturze given, when, then. Given, czyli dane, jest zakresem, w którym definiowane są dodatkowe dane, które wykorzystane są w danym teście. When, czyli kiedy, definiuje akcję, która jest testowana. Oprócz tego w zakresie when definiowane są zachowania modelowanych obiektów. W zakresie then, znajdują się oczekiwane wyniki akcji wyspecyfikowanej w zakresie then w kontekście danych z zakresu given. W takiej notacji wykonany jest test jednostkowy z Rys 6.1. W zakresie given definiowany jest dział kursu, który jest zmodyfikowanym działem, który tworzony jest w metodzie setup. W zakresie given zdefiniowane jest zachowanie zamodelowanego repozytorium, które z wywołania metody znajdującej dział po jego identyfikatorze z argumentem będącym identyfikatorem działu stworzonego w metodzie setup, zwraca właśnie ten dział. Następnie na wywołanie metody z serwisu mającej zmodyfikować dział w zakresie then oczekujemy, że dział z metody setup będzie posiadał nazwę oraz kurs działu zmodyfikowanego oraz zostanie jednokrotnie wywołana metoda z repozytorium zapisująca obiekt.

**class** AreaServiceTest **extends** Specification {  
  
 **private** AreaService **areaService**;  
 **private** AreaRepository **areaRepository**;  
 **private** Area **area**;  
  
 **void** setup() {  
 **areaService** = **new** AreaService()  
 **areaRepository** = Mock(AreaRepository.**class**)  
 **areaService**.**areaRepository** = **areaRepository  
  
 area** = **new** Area()  
 **area**.id = UUID.*randomUUID*()  
 **area**.name = **"Test Area"  
 areaRepository**.save(**area**)  
 }  
...  
 **def "Modify area"**(){  
 **given**:  
 Area modifiedArea = **new** Area(

**area**.id, **"Modified Area"**,

**new** Course(UUID.*randomUUID*(),**"Test Course"**)) **when**:  
 **areaRepository**.findById(modifiedArea.id) >> **area  
 areaService**.modifyArea(modifiedArea)  
 **then**:  
 **area**.name.is(modifiedArea.name)  
 **area**.course.is(modifiedArea.course)  
 1 \* **areaRepository**.save(**area**)  
  
 }

...  
}

Rys. 6.1. Implementacja testu jednostkowego

**6.2. Testy integracyjne**

Testowanie integracyjne, zwane też komponentowym polega na testowaniu działania kilku zintegrowanych, współpracujących ze sobą jednostek. Testowanie integracyjne powinno skupiać się na interfejsach komponentów kodu. [20] Takim interfejsem można nazwać każdy z kontrolerów znajdujących się w warstwie pośredniczącej. Szkielet Spock pozwala na testowanie API za pomocą zapytań HTTP, więc został on również wykorzystany do opracowywania testów integracyjnych. Test integracyjny na Rys. 6.2. również korzysta ze struktury given, when, then. W zakresie given tworzony jest obiekt, który będzie zawartością żądania HTTP, w zakresie when jest on przesyłany jako obiekt JSON. W zakresie when tworzone oraz wysyłane jest żądanie HTTP. W zakresie then sprawdzane jest czy status otrzymanej odpowiedzi potwierdza poprawne wykonanie operacji.

**class** AreaControllerTest **extends** Specification {  
  
 **private** AreaController **areaController  
 private** AreaService **areaService  
 private** Area **area** MockMvc **mockMvc  
  
 void** setup() {  
 **areaController** = **new** AreaController()  
 **areaService** = Mock(AreaService.**class**)  
 **areaController**.**areaService** = **areaService  
 mockMvc** = MockMvcBuilders.*standaloneSetup*(**areaController**).build()  
  
 **area** = **new** Area()  
 **area**.id = UUID.*randomUUID*()  
 **area**.name = **"Test Area"  
 areaService**.save(**area**)  
 }

...  
 **def "put"**(){  
 **given**:  
 Map request = [  
 **id** : **area**.id,  
 **name** : **"Modified Area"** ]  
 **when**:  
 **def** response = **mockMvc**.perform(*put*(**"/api/areas"**)  
 .contentType(MediaType.***APPLICATION\_JSON***)  
 .content(*toJson*(request)))  
 .andReturn().response  
 **then**:  
 response.status == ***OK***.value()  
 }  
}

Rys. 6.2. Implementacja testu integracyjnego

**6.3. Testy systemowe**

Testowanie systemowe, czyli kiedy część lub wszystkie komponenty są ze sobą zintegrowane i współpracują jako całość. Testom poddawany jest wtedy cały system. Przeprowadzane są na zasadzie interakcji z systemem. Testowanie systemowe zazwyczaj wykonywane jest na zasadzie czarnej skrzynki. Oznacza to, że osoba wykonująca test nie musi być świadoma jak zaimplementowany jest system, natomiast ważne jest to jakie wyjścia system dostarcza na podstawie podanych mu wejść. [20] Istotnym w testach systemowych jest to jaki jest oczekiwany wynik na daną akcję użytkownika. W odróżnieniu od testów jednostkowych i integracyjnych testy systemowe przeprowadzane są zazwyczaj jako testy manualne lub zautomatyzowane testy interfejsu użytkownika. Aby nadać testom systemowym strukturę należy przygotować instrukcje przeprowadzania tych systemów. Jako, że testy polegać mają na sprawdzaniu czy system reaguje poprawnie na akcje użytkownika, instrukcja powinna zawierać zdefiniowane akcje użytkownika i oczekiwane wyniki.

|  |  |  |
| --- | --- | --- |
| Numer kroku | Akcja | Oczekiwane wyniki |
| 1. |  |  |
| 2. |  |  |

**Bibliografia**

[1] *Apache Maven Project*. (2018). Pobrano z lokalizacji Maven: https://maven.apache.org/

do usuniecia [2] Bałaszow, B., Królikowski, T., Susłow, W. i Szatkiewicz, T. (brak daty). *Portal wspomagania nauczania Politechniki Koszalińskiej StudiaNET.pl*. Koszalin, Polska.

do usuniecia [3] Bielski, K. (2012). *Implementacja REST w Spring MVC 3.0*. Pobrano z lokalizacji Krzysztof Bielski Blog: https://kbielski.wordpress.com/tag/rest-web-service/

[4] *Building an Application with Spring Boot*. (2018). Pobrano z lokalizacji Spring: https://spring.io/guides/gs/spring-boot/

[y] Chen, N. (2006). Convention over Configuration.

do usuniecia [5] Dąbrowski, W. (brak daty). *Materiały edukacyjne Polsko-Japońskiej Akademii Technik Komputerowych*. Pobrano z lokalizacji http://edu.pjwstk.edu.pl/wyklady/byt/scb/wyklady/2\_2.html

[6] *Edytory dokumentów - Pomoc*. (2018). Pobrano z lokalizacji Google: https://support.google.com/docs/answer/7032287?hl=pl&ref\_topic=6063584&fbclid=IwAR3n1tdG0EaZzcOWA4MPSGMdgVdWqVFILofZjpbxJpkmH\_YHHbn7\_QruRyY

[x] *Formularze Google*. (2018). Pobrano z lokalizacji Google: https://www.google.pl/intl/pl/forms/about/#start

[7] Fowler, M. (2002). *Patterns of Enterprise Application Architecture.* Pearson Education.

[8] *Git*. (2018). Pobrano z lokalizacji Git: https://git-scm.com/

[9] *Google Design*. (2018). Pobrano z lokalizacji Google: https://design.google/

[10] *H2*. (brak daty). Pobrano z lokalizacji H2 Database Engine: http://www.h2database.com/html/main.html

[11] *IntelliJ IDEA*. (2018). Pobrano z lokalizacji JetBrains: https://www.jetbrains.com/idea/

[12] *moodledocs - Features*. (2014). Pobrano z lokalizacji https://docs.moodle.org/2x/pl/Features

[13] *moodledocs - Multiple Choice question type*. (2014). Pobrano z lokalizacji https://docs.moodle.org/2x/pl/Multiple\_Choice\_question\_type#Multiple-answer\_questions

[14] *moodledocs*. (2014). Pobrano z lokalizacji https://docs.moodle.org/2x/pl/Strona\_g%C5%82%C3%B3wna

[15] Moura, M. (2018). *Vue Material - Table*. Pobrano z lokalizacji Vue Material: https://vuematerial.io/components/table

[16] Nielsen, J. (1990). *Hypertext and Hypermedia.* Boston: Academic Press.

[17] Przyłuski, W. (2014). Wielokrotny wybór czyli rozważania o pewnych pytaniach testowych. *ELEKTRONIKA - KONSTRUKCJE, TECHNOLOGIE, ZASTOSOWANIA*.

[18] Richardson, L. i Amundsen, M. (2013). *RESTful Web APIs.* O'Reilly Media Inc.

[19] Rumbaugh, J., Jacobson, I. i Booch, G. (2004). *The Unified Modeling Language Reference Manual Second Edition.* Pearson Higher Education.

[20] Sommerville, I. (2011). *Software Engineering, 9th Edition.* Boston, Massachusetts: Pearson.

[21] Speelpenning, J., Daux, P. i Gallus, J. (2001). *Data Modeling and Relational Database Design.* Oracle.

[22] *Spock*. (2018). Pobrano z lokalizacji Spock: http://spockframework.org/

[23] *Spring Data JPA 2.1.2.RELEASE API*. (2018). Pobrano z lokalizacji https://docs.spring.io/spring-data/jpa/docs/current/api/overview-summary.html

[24] *Spring Framework 5.1.2.RELEASE API*. (2018). Pobrano z lokalizacji https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html

[25] Stephens, R. (2015). *Beginning Software Engineering.* Indianapolis: John Wiley & Sons, Inc.

[26] The Project Lombok Authors. (2018). *@Data*. Pobrano z lokalizacji Lombok Project: https://projectlombok.org/features/Data

[27] Thomas, J. (2018). *Bulma Documentation*. Pobrano z lokalizacji Bulma: https://bulma.io/documentation/

do usuniecia [28] Urbański, W. (2018). *Wprowadzenie do kursu Vue.js od podstaw*. Pobrano z lokalizacji type<of>web: https://typeofweb.com/2018/06/08/wprowadzenie-do-kursu-vue-js-od-podstaw/

[29] *Using Axios to Consume APIs*. (2018). Pobrano z lokalizacji Vue.js: https://vuejs.org/v2/cookbook/using-axios-to-consume-apis.html

[30] *Vue.js Guide*. (2018). Pobrano z lokalizacji Vue.js: https://vuejs.org/v2/guide/

[31] Walls, C. (2011). *Spring in Action, Third Edition.* Manning Publications Co.

[32] *WebStorm*. (2018). Pobrano z lokalizacji JetBrains: https://www.jetbrains.com/webstorm/

**Załączniki**

Rys 1.1. Model kaskadowy cyklu życia oprogramowania; Staszak, U. Projektowanie oprogramowania. Wykład 1. Politechnika Wrocławska, 20174

Rys 2.1. Widok tworzenia testu w aplikacji Formularze Google; Zrzut ekranu ze strony: ttps://docs.google.com/forms/d/e/1FAIpQLSdFIFB5c2ZphgFoJnsf69URQ0lmZuccaYaTHiBCOOODnbThcg/viewscore?viewscore=AE0zAgAlA8Du2XTy8qyrQ-0qoIAwWhd\_AIWjGhrQcfoJTSYCgAE7VQ6uk739JQ 6

Tabela 3.1. Użytkownicy aplikacji 9

Tabela 3.2. Historyjki użytkowników 10

Tabela 3.3. Wymagania funkcjonalne10

Tabela 3.4. Wymagania niefunkcjonalne11

Tabela 3.5. PU001 12

Tabela 3.6. PU00212

Tabela 3.7. PU00312

Tabela 3.8. PU00413

Tabela 3.9. PU00513

Tabela 3.10. PU00613

Tabela 3.11. PU00714

Tabela 3.12. PU00814

Tabela 3.13. PU009 15

Tabela 3.14. PU01015

Tabela 3.15. PU01116

Tabela 3.16. PU01216

Tabela 3.17. PU01316

Tabela 3.18. PU01417

Tabela 3.19. PU01517

Tabela 3.20. PU01617

Tabela 3.21. PU017 18

Tabela 3.22. PU01818

Rys 3.1. Diagram przypadków użycia 19

Rys 3.2. Spis przedmiotów 20

Rys 3.3. Dodawanie przedmiotu 21

Rys 3.4. Spis działów przedmiotu 21

Rys 3.5. Dodawanie działu przedmiotu 22

Rys 3.6. Spis pytań 22

Rys 3.7. Dodawanie nowego pytania 23

Rys 3.8. Spis testów 23

Rys 3.9. Dodawanie testu 24

Rys 3.10. Rozwiązywanie testu. 24

Rys 3.11. Wynik testu 25

Rys 3.12. Edytowanie profilu 25

Rys 4.1. Diagram pakietów 31

Rys 4.2. Diagram klas pakietu model 32

Rys 4.3. Diagram klas pakietu service 33

Rys 4.4. Diagram klas pakietu controller 34

Rys 4.5. Diagram klas pakietu dto 35

Rys 4.5. Diagram ERD 36

Rys 5.1. Struktura strony 37

Rys 5.2. Implementacja komponentu 38

Rys 5.3. Wynik implementacji komponentu 39

Rys 5.4. Implementacja kontolera 40

Rys 5.5. Implementacja serwisu studenta 35

Rys 5.6. Implementacja serwisu testu 36

Rys 5.7. Implementacja DTO 34

Rys 5.8. Implementacja repozytorium 35

Rys 5.9. Spis przedmiotów 42

Rys 5.10. Dodawanie przedmiotu 43

Rys 5.11. Spis działów przedmiotu 43

Rys 5.12. Dodawanie działu przedmiotu 44

Rys 5.13. Spis pytań 44

Rys 5.14. Dodawanie nowego pytania 45

Rys 5.15. Spis testów 45

Rys 5.16. Dodawanie testu 46

Rys 5.17. Rozwiązywanie testu. 46

Rys 5.18. Wynik testu 47

Rys 5.19. Edytowanie profilu 47