UNIWERSYTET GDAŃSKI

Wydział Matematyki, Fizyki i Informatyki

Łukasz Ekiert

nr albumu: 187 310

Aplikacja do wspomagania nauki języków obcych oparta na frameworku *Angular 2*

Praca magisterska na kierunku:

INFORMATYKA

Promotor:

dr Włodzimierz Bzyl

Gdańsk 2017

Streszczenie

Przedmiotem projektu jest narzędzie dla szkół językowych i innych podmiotów zainteresowanych wykorzystywaniem aplikacji WWW w procesie nauczania. Przygotowano kilka szablonów ćwiczeń, które są grupowane w zestawy testowe. Użytkownicy (uczniowie) mają przypisane do swoich kont kursy składające się takich zestawów. Wyniki rozwiązanych testów są zapamiętywane, dzięki czemu istnieje możliwość śledzenia postępów w nauce.

Wykonano aplikację e-learningową w architekturze klient-serwer. Za warstwę serwerową odpowiada *Ruby on Rails 5*, który służy głównie jako interfejs komunikacyjny klienta z bazą danych działającą na silniku *PostgreSQL*. Konsumentem API jest napisany w języku TypeScript klient oparty na frameworku *Angular 2*. Kod znajduje się w publicznych repozytoriach: *https://github.com/lekiert/quickstep-api* (API) oraz *https://github.com/lekiert/quickstep-client* (klient). Informacje dotyczące dostępu do wersji demonstracyjnej znajdują się w pliku *README.md* w repozytorium aplikacji klienckiej. Wewnątrz repozytoriów klienta i serwera znaleźć można również instrukcje instalacji oraz uruchomienia testów automatycznych.

W pracy opisano założenia przyjęte podczas projektowania aplikacji, architekturę (w tym wykorzystane biblioteki) oraz istotne szczegóły implementacyjne. Projekt został wdrożony i jest wykorzystywany w szkole językowej jako jedno z narzędzi dydaktycznych w autorskim programie nauczania języka angielskiego.

Słowa kluczowe

języki obce, e-learning, TypeScript, Angular 2, Ruby on Rails 5

Spis treści

Wprowadzenie

Motywacją dla powstania pracy była chęć poznania stabilnej wersji frameworka *Angular 2*, wydanej 15 września 2016 roku, oraz wypróbowania dedykowanych dla niego narzędzi wspomagających proces tworzenia aplikacji. W ramach projektu wykorzystującego tę technologię stworzono aplikację wspomagającą naukę języków obcych.

W procesie nauki języka obcego zazwyczaj wykorzystuje się podręczniki dostosowane do potrzeb uczniów. W przypadku języka angielskiego można w ramach przykładu przytoczyć materiały udostępniane przez wydawnictwo *Pearson* (dawniej *Pearson Longman*), czy *Cambridge University Press*. Ich zawartość i stopień trudności różnią się w zależności od grupy docelowej, jednak ich generalną cechą jest organizacja ćwiczeń według przyjętego przez autorów programu nauczania[6]. Zazwyczaj jest to udogodnienie dla lektorów, gdyż nie muszą przygotowywać materiałów do zajęć (lub przygotowują ich mniej). Co więcej, jest to z korzyścią dla uczniów, którzy mają dostęp do wartościowych pomocy dydaktycznych, gdyż autorami podręczników są osoby o wysokich kompetencjach językowych i metodycznych.

Podręczniki wydawane zarówno w formie papierowej, jak i elektronicznej, składają się z dwóch głównych części: teoretycznej, wprowadzającej osobę uczącą się w nowy materiał, oraz ćwiczeń praktycznych. Te ostatnie zawierają testy służące sprawdzeniu i utrwaleniu zdobytej wiedzy. Ich forma z reguły opiera się na wypróbowanych schematach ćwiczeń, takich jak: wypełnianie luk, parowanie fraz, testy jedno- i wielokrotnego wyboru, krzyżówki. Dodatkowo wykorzystuje się pomoce multimedialne w formie obrazków, zdjęć, nagrań audio i wideo, jak również aplikacje e-learningowe.

Aplikacje takie można podzielić na dwie zasadnicze grupy: poświęcone konkretnym podręcznikom oraz samodzielne oprogramowanie, które zazwyczaj funkcjonuje w formie serwisów webowych. Istniejącymi na rynku rozwiązaniami z tej drugiej grupy są na przykład *busuu*[1] i *Duolingo*[2]. Umożliwiają one rozwiązywanie ćwiczeń oraz zapoznawanie się z przygotowanymi materiałami dydaktycznymi, ponadto zawierają szereg udogodnień,

[1] https://www.busuu.com
[2] https://www.duolingo.com

takich jak: spersonalizowane testy, kontrola postępów, konwersacje z *native speakers* (zazwyczaj w formie płatnej), zgrywalizowany proces nauki.

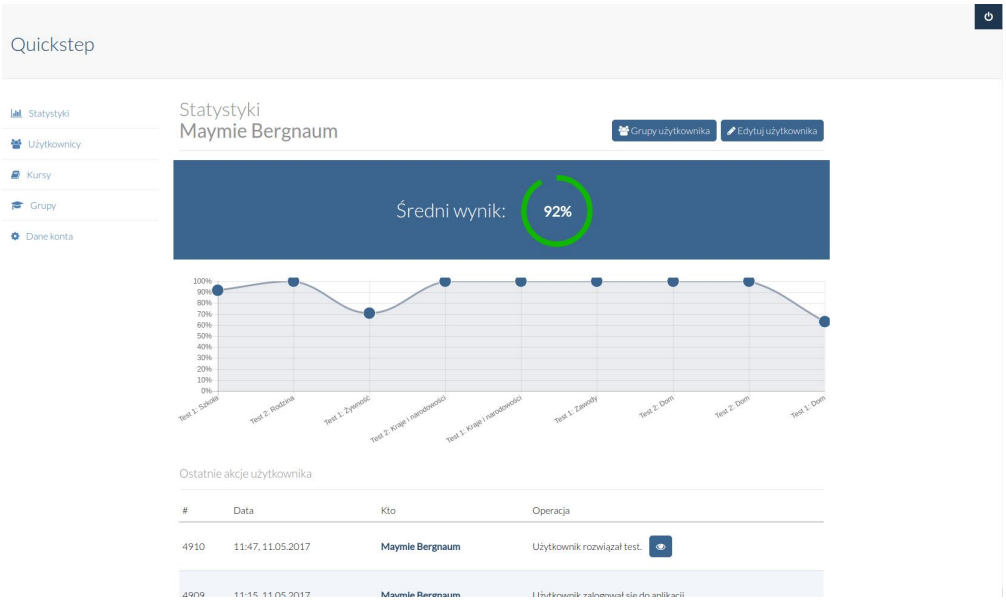
Warto zwrócić uwagę na fakt, że takie aplikacje i podręczniki zazwyczaj skierowane są do szerokiego grona odbiorców. W wielu przypadkach można taką cechę uznać za zaletę, jednak należy mieć na uwadze fakt, iż istnieją również grupy wymagające materiałów do nauki o bardzo konkretnej tematyce lub sposobie przedstawiania oraz sprawdzania wiedzy. W tym przypadku często korzysta się z autorskich podręczników i programów nauczania, stworzonych pod kątem tych wymagań. Z reguły osoby tworzące takie rozwiązania nie mają środków ani umiejętności, aby stworzyć do swoich materiałów warstwę aplikacji e-learningowej, która wspomagałaby proces dydaktyczny.

Celem niniejszej pracy było napisanie aplikacji, dzięki której podmioty zajmujące się nauczaniem języków obcych przy pomocy autorskich rozwiązań mogłyby skorzystać z możliwości e-learningu. Twórca podręcznika może opracowywać własne testy (zgrupowane w kursy) zawierające ćwiczenia oparte na przygotowanych uprzednio szablonach. Uczniowie mogą rozwiązywać testy z przypisanych im kursów oraz śledzić własne postępy. Każdy podmiot uruchamia własną instancję aplikacji, więc ma możliwość ograniczenia zależności od zewnętrznych usługodawców.

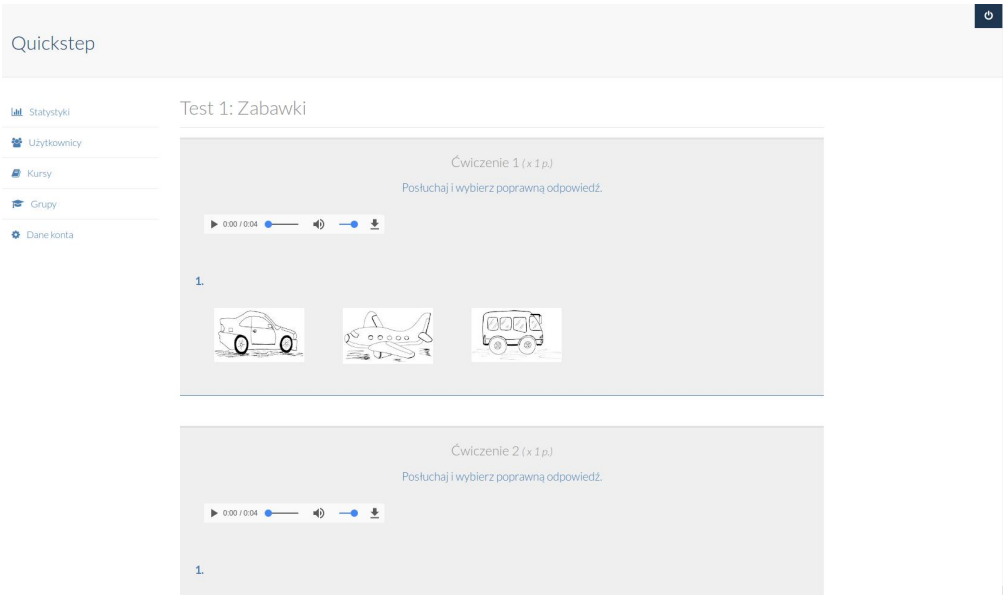
Główną częścią projektu jest aplikacja kliencka napisana we frameworku *Angular 2*. Zawiera ona interfejsy dla wszystkich przyjętych rodzajów kont (administratorów, kierowników, nauczycieli oraz uczniów) i zawiera większość logiki projektu. Jest skomunikowana z API zgodnym ze standardem JSON API[3].

Warstwą API jest aplikacja napisana we frameworku *Ruby on Rails 5*. Został on wybrany ze względu na szybkość wytwarzania gotowego produktu, ponadto posiada on rozszerzenia (*gemy*) wspierające wystawianie końcówek (*endpointów*) zgodnych ze standardem JSON API. Służy on głównie jako interfejs komunikacyjny pomiędzy aplikacją kliencką, a bazą danych, zawiera jednak w niektórych miejscach elementy logiki biznesowej, jak np. weryfikacja poprawności rozwiązań.

[3] http://jsonapi.org



Rysunek 1. Przykładowy zrzut ekranu - widok podglądu Ucznia



Rysunek 2. Przykładowy zrzut ekranu - widok testu

ROZDZIAŁ 1

Projekt aplikacji

Głównym celem projektu jest dostarczenie takiego narzędzia, aby osoby chcące wdrożyć elementy e-learningu do własnych programów nauczania mogły niewielkim kosztem uruchomić własną instancję aplikacji. W związku z tym przyjęto, iż najbardziej uniwersalnym podejściem będzie tu próba odtworzenia środowiska szkoły językowej, stąd w wymaganiach istotny jest podział na grupy, selektywny dostęp każdej z nich do zasobów edukacyjnych oraz typy kont odwzorowujące role pełnione przez poszczególne podmioty w rzeczywistej placówce.

1.1. Użyte terminy

- **Klient** – warstwa kliencka projektu konsumująca JSON API.
- **Użytkownik** – konto w aplikacji o przypisanym typie.
- **Uczeń** – typ konta użytkownika końcowego. Zorientowane jest na rozwiązywanie ćwiczeń i podgląd własnych akcji oraz postępów.
- **Nauczyciel** – typ konta lektora. Służy do nadzorowania rezultatów testów rozwiązywanych przez uczniów do przypisanych mu grup.
- **Kierownik** – typ konta przewidziany do podglądu wszystkich akcji użytkowników aplikacji. Przykładem osoby z kontem kierownika jest pracownik szkoły językowej nadzorujący wywiązywanie się lektorów z powierzonych im obowiązków.
- **Administrator** – konto o najszerszych uprawnieniach. Osoby z kontem Administratora odpowiedzialne są za tworzenie treści dostępnej dla pozostałych kont użytkowników: dodawanie kursów, testów, ćwiczeń, zarządzanie kontami użytkowników.
- **Grupa** – zbiór uczniów oraz przypisanych im nauczycieli oraz kursów.

- **Kurs** – zbiór testów. Organizacja testów zależy od administratora.
- **Test** – zbiór ćwiczeń, z założenia związanych tematyką i stopniem trudności.
- **Ćwiczenie** – jednostka testowa służąca do weryfikacji posiadanej wiedzy oparta o przygotowany szablon.
- **Szablon** – typ ćwiczenia.

1.2. Wymagania aplikacji

1.2.1. Użytkownicy aplikacji

Do korzystania z aplikacji niezbędne jest posiadanie konta. Jest ono zakładane przez Administratora, albowiem w aplikacji nie ma przewidzianej możliwości samodzielnej rejestracji. Niemniej jednak przy spełnieniu założenia prostego i otwartego API, taka funkcja może zostać łatwo zaimplementowana. Każde konto Użytkownika musi mieć przypisany konkretny typ determinujący zarówno stopień uprawnień do wyświetlania, tworzenia, edycji oraz usuwania poszczególnych zasobów, jak i wyświetlany po zalogowaniu interfejs. Krytycznymi dla poprawnego działania aplikacji typami kont są Administrator oraz Uczeń. Typy Nauczyciela oraz Kierownika pełnią funkcję pomocniczą.

Przyczyna takiej klasyfikacji wynika z założenia, iż Administrator jest kontem o największych możliwościach i to on odpowiada za treść oraz konta pozostałych Użytkowników, zaś Uczeń jest ostatecznym konsumentem aplikacji, gdyż z myślą o nim powstała. Zadaniem nauczyciela jest nadzorowanie postępów własnych podopiecznych, poza tym nie posiada on innych szczególnych uprawnień. Kierownik ma możliwość wglądu w akcje wszystkich Użytkowników aplikacji. Intencją powstania tego typu konta było zapotrzebowanie na możliwość kontrolowania zaangażowania uczniów oraz wywiązywania się ze swoich obowiązków przez lektorów. Z powyższych opisów wypływa zatem wniosek, iż to Administratorzy oraz Uczniowie z założenia mają główny wpływ na zmiany stanu modelu, zaś Nauczyciel oraz Kierownik są, co do zasady, jedynie obserwatorami, lub oddziaływują na treści pośrednio, kierując uwagi do Administratora.

1.2.2. Wymagania funkcjonalne

Wymagania wspólne dla wszystkich typów kont

- Użytkownik musi dokonać autentykacji, by korzystać z aplikacji. Administrator zakłada każdej osobie, która będzie korzystać z aplikacji, konto o odpowiednim poziomie uprawnień.
- Użytkownik musi mieć dostęp do logów akcji. Każdy użytkownik musi mieć możliwość wyświetlenia widoku, w którym zawarte są logi akcji oraz ich wykres. Zakres danych wyświetlanych w takim widoku jest determinowany przez typ konta.
- Użytkownik musi mieć możliwość zmiany swojego hasła.

Wymagania dla konta Ucznia

- W widoku podsumowania muszą być wyświetlane wyłącznie logi danego Ucznia.
- Uczeń musi mieć możliwość:
  - bycia przypisanym do grupy.
  - rozwiązywania testów, które są przypisane do jego grupy.
- Po przesłaniu rozwiązania testu, Uczniowi musi zostać od razu wyświetlony wynik wyrażony w punktach procentowych oraz muszą być zaznaczone błędne odpowiedzi.
- Uczeń musi mieć dostęp do historii rozwiązanych testów.

Wymagania dla konta Nauczyciela

- Nauczyciel musi mieć:
  - możliwość bycia przypisanym do Grupy jako lektor,
  - dostęp do wszystkich rozwiązań Uczniów w grupach, w których jest zapisany jako lektor.

- W widoku podsumowania muszą być wyświetlane logi zarówno danego Nauczyciela, jak i wszystkich Uczniów z grup do niego przypisanych.

Wymagania dla konta Kierownika

- Kierownik musi mieć możliwość:
  - wyświetlenia list kont Nauczycieli i Uczniów,
  - wyświetlenia listy Grup.
- Nauczyciel musi mieć dostęp do wszystkich przesłanych rozwiązań testów.
- W widoku podsumowania muszą być wyświetlane logi wszystkich Użytkowników aplikacji.

Wymagania dla konta Administratora

- Administrator musi mieć możliwość:
  - zarządzania Użytkownikami, Grupami, Kursami oraz Testami,
  - tworzenia Testów z ćwiczeniami, które mogą być rozwiązywane przez Uczniów oraz Nauczycieli, o ile są przypisane do tych samych Grup, co konta,
  - zmiany haseł wszystkich Użytkowników na samodzielnie wybrane lub automatycznie wygenerowane,
  - edycji danych osobowych wszystkich Użytkowników.
- W widoku podsumowania muszą być wyświetlane logi wszystkich Użytkowników aplikacji.

1.2.3. Wymagania niefunkcjonalne

- **API oparte na *Ruby on Rails 5***
- **Klient oparty na frameworku *Angular 2*** – z racji tego, że zasadniczą motywacją powstania projektu była chęć poznania tej technologii, to główna część musi zostać w niej napisana. Należy mieć na uwadze, że jest wciąż

dynamicznie rozwijana. W chwili pisania pracy wydano już wersję 4.0 (przy jednoczesnej zmianie systemu wersjonowania).

- **Klient napisany w języku TypeScript** – istnieje możliwość tworzenia aplikacji w językach takich jak Dart czy JavaScript, jednak oficjalna dokumentacja przedstawiała przykłady oparte o język TypeScript, więc przyjęto to za technologię sugerowaną przez autorów frameworka.

- **Otwarte źródło** – aby zagwarantować możliwość korzystania zainteresowanym podmiotom z projektu, aplikacja musi być dostępna w formie darmowej i gotowej do własnoręcznego uruchomienia. Całość powinna znajdować się w publicznym repozytorium w serwisie *GitHub*.

- **Responsywność** – typ konta przewidziany do podglądu wszystkich akcji użytkowników aplikacji. Przykładem osoby z kontem Kierownika jest pracownik szkoły językowej nadzorujący wywiązywanie się z obowiązków przez lektorów.

- **Otwartość na rozszerzenia** – konto o najszerszych uprawnieniach. Osoby z kontem Administratora odpowiedzialne są za tworzenie treści dostępnej dla pozostałych kont użytkowników: dodawanie kursów, testów, ćwiczeń, zarządzanie kontami użytkowników.

- **Ćwiczenia z treściami audiowizualnymi** – niezbędna jest możliwość dodawania do testów ćwiczeń, w których umieszczone są pliki graficzne oraz dźwiękowe. Materiały o charakterze multimedialnym są szczególnie istotne w przypadku kształcenia dzieci i młodzieży, jak również do treningu umiejętności związanych z rozumieniem słuchanych wypowiedzi.

- **Niezależność** – aplikacja powinna działać poprawnie na popularnych przeglądarkach. Przyjęto, że takimi przeglądarkami są wymienione w oficjalnej dokumentacji frameworka *Angular 2*[3]. Ponadto Użytkownik nie powinien być zmuszony do instalacji w swoim systemie żadnych rozszerzeń ani wtyczek.

1.2.4. Przypadki użycia

Tabela 1.1. Dodawanie ćwiczenia „luki” do testu

UC5	Dodawanie ćwiczenia ”luki” do testu
Aktor	Administrator
Warunki	Administrator w widoku edycji testu.
Przebieg	1. Administrator w sekcji „Ćwiczenia w teście” wybiera rodzaj ćwiczenia „luki” i zatwierdza wybór. Pojawia się formularz dodawania ćwiczenia. 2. Administrator wypełnia dane ćwiczenia. 3. Administrator wybiera „Dodaj zdanie”. Pojawia się szerokie pole tekstowe. 4. Administrator wpisuje nowe zdanie, oznaczając brakujące frazy (luki) ustalonym ciągiem znaków specjalnych. 5. Wraz z każdym wystąpieniem luk pod polem zdania pojawiają się odpowiadające im pola. 6. Administrator w tych polach wpisuje prawidłowe rozwiązania dla każdej odpowiadającej luki. W przypadku, gdy dla danej luki poprawna jest więcej niż jedna fraza, oddziela je przecinkiem. 7. Administrator może przejść ponownie do punktu 3, aby dodać kolejne zdania, albo zakończyć proces wybierając „Dodaj ćwiczenie”.

Tabela 1.2. Dodawanie ćwiczenia „wybór” do testu

UC5	Dodawanie ćwiczenia „wybór” do testu
Aktor	Administrator
Warunki	Administrator w widoku edycji testu.
Przebieg	1. Administrator w sekcji „Ćwiczenia w teście” wybiera rodzaj ćwiczenia „wybór” i zatwierdza wybór. Pojawia się formularz dodawania ćwiczenia. 2. Administrator wypełnia dane ćwiczenia. 3. Administrator wybiera „Dodaj zdanie”. Pojawia się szerokie pole tekstowe. 4. Administrator wpisuje nowe zdanie. 5. Administrator wybiera znak plusa. Pojawia się element zawierający: *checkbox*, pole tekstowe oraz pole pozwalające na załadowanie pliku. 6. Administrator w polu tekstowym wpisuje treść odpowiedzi. Może również załadować własny plik graficzny lub dźwiękowy. 7. Administrator ustala poprawne odpowiedzi do zdania poprzez zaznaczenie pola typu *checkbox*. 8. Administrator może przejść ponownie do punktu 3, aby dodać kolejne zdania, albo zakończyć proces wybierając „Dodaj ćwiczenie”.

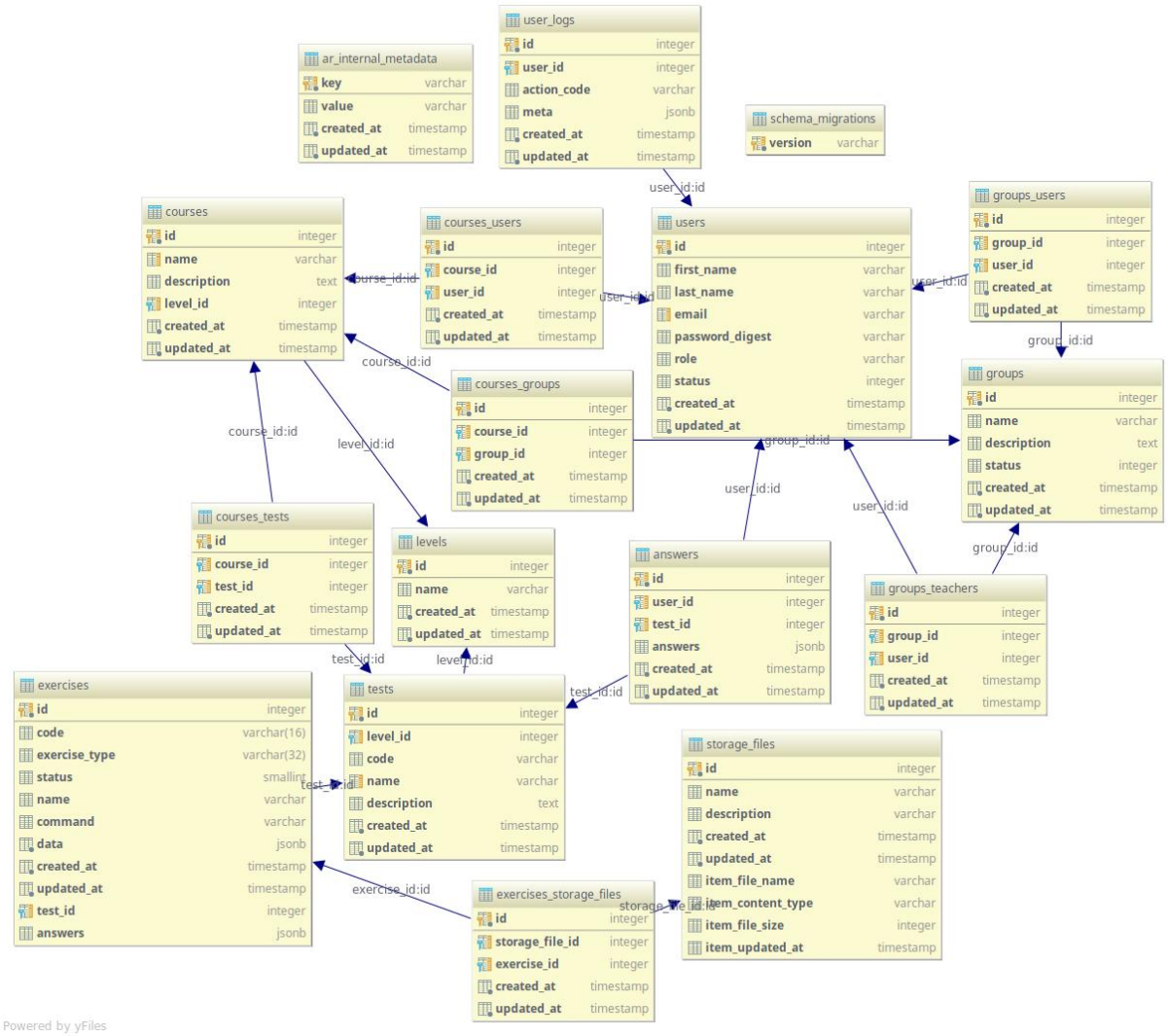
Tabela 1.3. Rozwiązywanie ćwiczenia „luki”

UC5	Rozwiązywanie ćwiczenia „luki”
Aktor	Uczeń
Warunki	Użytkownik w widoku testu zawierającego ćwiczenie o typie „luki”.
Przebieg	1. Uczeń w każdym zdaniu wpisuje odpowiedzi w polach tekstowych reprezentujących luki. 2. Po wysłaniu rozwiązania, prawidłowe odpowiedzi są podkreślone kolorem zielonym, a błędne czerwonym.

Tabela 1.4. Rozwiązywanie ćwiczenia „wybór”

UC5	Rozwiązywanie ćwiczenia „wybór”
Aktor	Uczeń
Warunki	Użytkownik w widoku testu zawierającego ćwiczenie o typie „wybór”.
Przebieg	1. Uczeń zapoznaje się ze zbiorem odpowiedzi przy każdym zdaniu. Jeżeli odpowiedź zawiera plik graficzny, jest on od razu wyświetlany. Jeżeli odpowiedź zawiera plik dźwiękowy, wyświetlana jest kontrolka umożliwiająca natychmiastowe odtworzenie nagrania. 2. Po wysłaniu rozwiązania, prawidłowe odpowiedzi są zaznaczone kolorem zielonym, a błędne czerwonym.

1.3. Schemat bazy danych



Rysunek 1.1. Schemat bazy danych

Źródło: Opracowanie własne

ROZDZIAŁ 2

Szczegóły implementacji

2.1. API

Jest aplikacją opartą na frameworku *Ruby on Rails 5*, służącą za interfejs komunikacyjny pomiędzy Klientem, a bazą danych. Oprócz udostępniania odpowiednich końcówek, do jej zadań należy również autentykacja, autoryzacja, logowanie akcji użytkowników i weryfikacja poprawności przesłanych rozwiązań. Udostępnia również zagregowane dane wyświetlane w widokach podsumowań. Do wdrożenia wybrano bazę danych *PostgreSQL*, gdyż wykorzystano typ kolumn JSONB. W przypadku potrzeby skorzystania z innego systemu bazy danych, należy zmienić w migracji typ na JSON.

2.1.1. Wykorzystane biblioteki

JSONAPI::Resources

- **JSONAPI::RESOURCES** Biblioteka rozwijana na licencji MIT, służąca jako zestaw narzędzi pomocniczych dla frameworka *Ruby on Rails* (wersji 4.2 i nowszych) do implementacji standardu *JSON API*. W ich skład wchodzą m.in. funkcje definiujące ścieżki dla kontrolerów, zasoby (*resources*) i służące do definiowania relacji pomiędzy nimi metody, jak również wsparcie dla mechanizmów filtrowania, dołączania podzasobów, paginacji oraz innych operacji wchodzących w skład standardu. Odpowiedzi na żądania HTTP są zautomatyzowane – programista nie musi martwić się o formatowanie danych wyjściowych, czy zwracane kody dla zapytań.

  Definiowanie ścieżek – funkcje definiujące ścieżki zasobów aplikacji, które wchodzą w skład biblioteki, rozszerzają zachowanie natywnych dla *Ruby on Rails* o uwzględnianie relacji pomiędzy klasami zasobów.

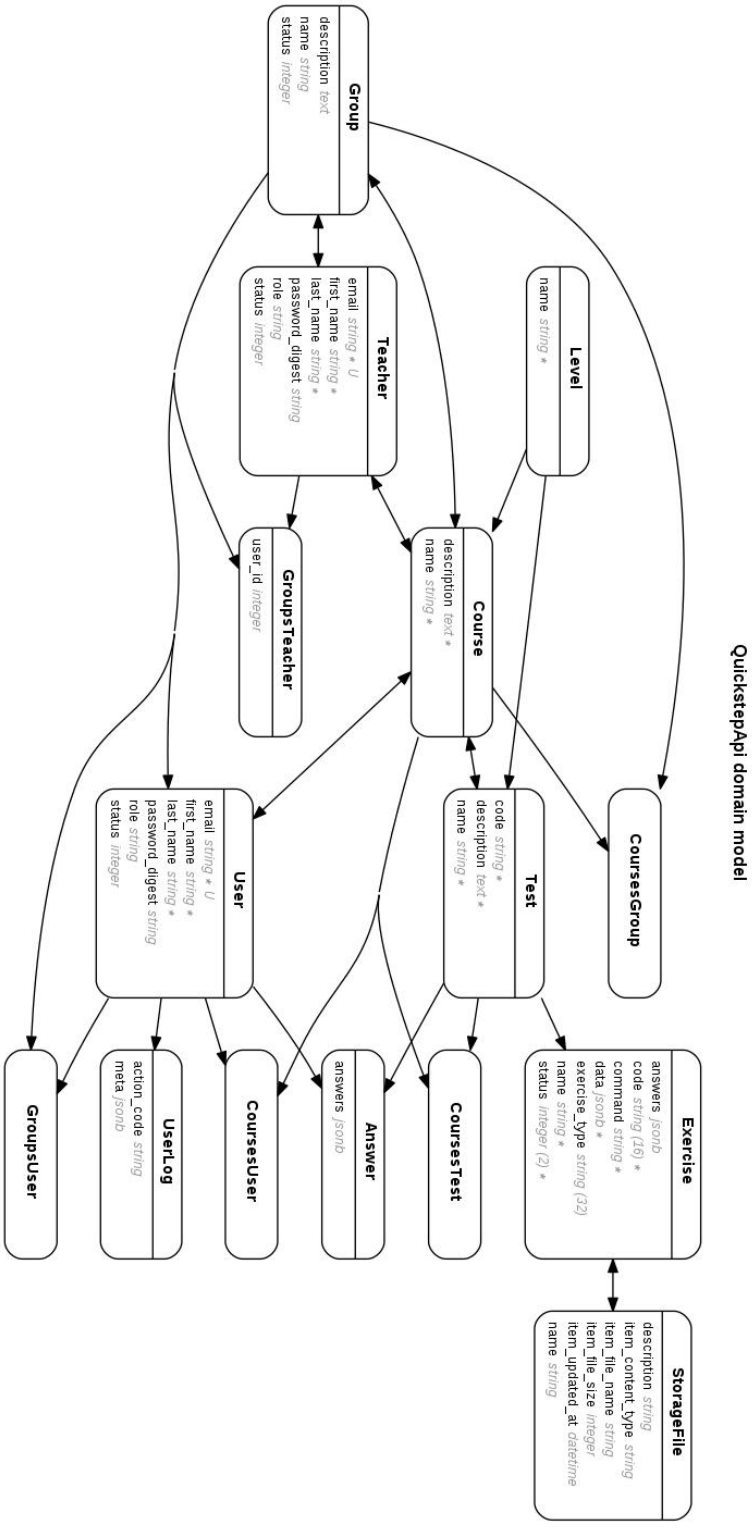
  Definiowanie kontrolerów – dziedziczenie po klasie *ResourceController*

niemal całkowicie automatyzuje logikę przetwarzania żądań do API oraz odpowiedzi. Walidują nagłówki i format przesłanych danych, przekazują przetworzone żądanie do klas modelowych, formatują zwracane dane.

Definiowanie zasobów – każda końcówka API musi być reprezentowana przez klasę zasobu. Klasy zasobów zawierają główną logikę odpowiedzialną za spełnianie wymagań *JSON API*, do których należą m.in.: definiowanie relacji pomiędzy zasobami, strategie filtrowania, paginacji oraz sortowania, określanie atrybutów i reguł dostępu (np. *read-only*), rodzaju zasobu (np. *singleton*).

- **Knock** – służy do obsługi autentykacji użytkowników aplikacji opartych o Rails API przy pomocy standardu *JSON Web Token* (*JWT*). Zastosowanie takiego podejścia gwarantuje bezstanowość, wymaganą przez *REST*, a więc również *JSON API*.
- **jwt** – implementacja standardu RFC 7519 OAuth JSON Web Token.
- **pg** – sterownik bazy danych *PostgreSQL*.
- **Paperclip** – biblioteka do obsługi załączników dla *ActiveRecord*. Zawiera pomocnicze metody dla modeli i migracji. Odwoływanie się do plików w kodzie odbywa się poprzez atrybut modelu.

2.1.2. Diagram związków encji



Rysunek 2.1. Diagram związków encji

Źródło: Wygenerowane przy pomocy gemu *Rails ERD*

2.1.3. Architektura

API udostępnia następujące zasoby:

- *exercises* – ćwiczenia,
- *answers* – rozwiązania Uczniów,
- *courses* – kursy,
- *tests* – testy,
- *users* – użytkownicy. Dostępny jest również zasób *teachers*, filtrujący użytkowników w taki sposób, że są zwracani wyłącznie Nauczyciele. Wirtualny podzasób *password_updates* służy do aktualizacji haseł użytkowników.
- *groups* – grupy,
- *user_logs* – logi akcji użytkowników,
- *storage_files* – pliki.

Każdy zasób posiada zdefiniowane relacje oraz logikę autoryzacyjną i filtrującą dostępne dla użytkownika dane. Klasy zasobów umieszczone są w katalogu *app/resources*.

2.2. Aplikacja kliencka

Jest główną częścią projektu i konsumentem API. Do jej odpowiedzialności należy obsługa warstwy prezentacji oraz dostarczanie wszystkich niezbędnych mechanizmów i funkcji dla każdego z rodzajów użytkowników. Napisana jest we frameworku *Angular 2* w architekturze komponentowej.

2.2.1. Wykorzystane biblioteki i narzędzia

Poniżej przedstawiono narzędzia wykorzystane do obsługi logiki aplikacyjnej oraz ułatwiających proces pisania kodu:

- **Angular CLI** – ważne narzędzie pomagające w procesie wytwarzania aplikacji napisanych we frameworku *Angular 2*. Nie jest ono integralną częścią repozytorium frameworka, więc musi być instalowane w systemie odrębnie. Zawiera skrypty inicjalizujące nowe projekty, generatory dla wszystkich rodzajów obiektów (klas, komponentów, dyrektyw, usług itp.), zintegrowany serwer deweloperski (oparty na *webpack-dev-server*), narzędzia budujące projekt oraz testujące kod (*Karma*, *Jasmine* oraz *Protractor*). Narzędzie *webpack-dev-server* jest szczególnie pomocne podczas pisania kodu, gdyż dzięki funkcji *live reload* programista może w czasie rzeczywistym obserwować efekty zmian wprowadzanych w kodzie – przeglądarka jest na bieżąco odświeżana podczas zapisu plików spełniających reguły określone w konfiguracji.
- **angular2-jwt** – biblioteka służąca do obsługi autentykacji przy użyciu standardu *JWT*. Dzięki niej można korzystać z klasy *AuthHttp*, która jest wrapperem dla natywnej klasy frameworka *Angular 2 – Http*, wykonującej zapytania do API wraz z automatycznie ustawionymi nagłówkami zawierającymi dane autentykacyjne. Ponadto zajmuje się dekrypcją tokenów, sprawdzaniem daty ich ważności oraz dostarcza funkcje pomocnicze, które można wykorzystać przy definiowaniu reguł dostępu do ścieżek. Biblioteka implementuje wzorzec projektowy *Strategia*, gdyż wykorzystuje ten sam interfejs, co natywny moduł *Http*.

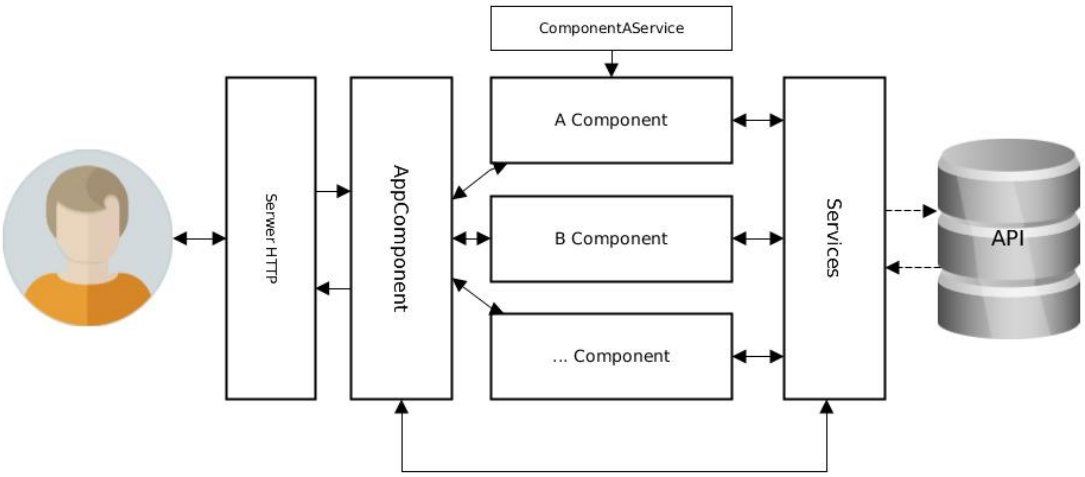
Kolejną grupą, którą można wyodrębnić, są biblioteki wykorzystywane w warstwie prezentacji:

- **Gridle** – biblioteka napisana w *SCSS*, podtypie języka *Sass*. Dzięki niej można zdefiniować siatkę (*grid*), która pozwala na sterowanie zachowywaniem się elementów *DOM* w zależności od szerokości okna przeglądarki. Zawiera również makra mające na celu ułatwić określanie reguł wyświetlania na urządzeniach mobilnych.
- **Font Awesome** – zestaw ikon na otwartej licencji.

- **Chart.js oraz ng2-charts** – narzędzie do wizualizacji danych w formie responsywnych wykresów.

2.3. Architektura

Głównym komponentem aplikacji jest *AppComponent*, który jest korzeniem drzewa reszty komponentów. W jej szablonie znajduje się element *router-outlet*, odpowiadający za wyświetlanie komponentów-dzieci (subkomponentów) w zależności od URI. Subkomponenty pogrupowane są w moduły, które wymienione są w dalszej części rozdziału. Zarówno *AppComponent* jak i jego dzieci komunikują się z modelem danych poprzez klasy serwisowe. Niektóre z tych klas tworzą warstwę komunikacyjną z modelem danych, a ich architektura spełnia założenia wzorca *Repozytorium*[5].



Rysunek 2.2. Architektura aplikacji

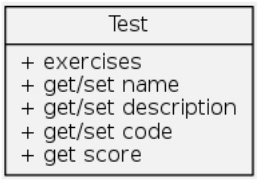
Źródło: Opracowanie własne

Poniżej znajduje się lista modułów dedykowanych dla aplikacji[1] wraz z krótkim opisem zakresu funkcji, które obejmują:

[1] Komponent *AppComponent* importuje również inne moduły pochodzące z bibliotek i samego frameworka

- **AnswerModule** – komponenty prezentujące rozwiązane testy oraz listę rozwiązanych testów,
- **CourseModule** – komponenty wyświetlające listę kursów oraz formularz ich dodawania,
- **ExerciseModule** – komponenty listy ćwiczeń, formularze dodawania/edycji, formularze rozwiązywania ćwiczeń,
- **GroupModule** – komponenty tworzenia, wyświetlania/edycji oraz listy grup,
- **SharedServicesModule** – serwisy komunikujące się z API,
- **TestModule** – komponenty dodawania, edycji oraz wyświetlania testów Uczniowi,
- **UserModule** – komponenty dodawania, edycji, listy użytkowników, wyświetlania statystyk oraz grup przypisanych Uczniowi,
- **UtilModule** – pomocnicze komponenty: *FileUploadComponent* obsługujący wgrywanie plików na serwer oraz *SearchFieldComponent* ułatwiający wyszukiwanie użytkowników.

Oprócz wspomnianych klas serwisowych oraz komponentów wyróżnić można również klasy encji, które składają się głównie z getterów i setterów. Niektóre z nich implementują także metody pomocnicze. Przykładem takiej encji jest klasa *Test*.



Rysunek 2.3. Diagram klasy *Test*

2.3.1. Komponenty

Poniżej opisano wybrane komponenty, które zawierają charakterystyczną dla aplikacji logikę. Pominięto pozostałe, takie jak formularz edycji danych konta, czy lista użytkowników, gdyż nie zawierają one cech wyróżniających ich spośród kanonicznych rozwiązań.

Style komponentów napisane są w języku *SCSS* zgodnie z metodologią BEM.

Komponent: *Summary*

Ten komponent służy do wyświetlania ostatnich akcji użytkownika oraz ostatnio rozwiązanych testów. Gdy zalogowany jest Administrator lub Kierownik, prezentowane są wszystkie akcje oraz rozwiązane testy w kolejności odwrotnej chronologicznie. Komponent ten jest również wyświetlany, gdy Administrator jest w widoku podglądu statystyk konkretnego użytkownika. W takiej sytuacji pojawia się także w prawym górnym rogu element *button*, kierujący do listy grup, do których przypisany jest dany użytkownik.

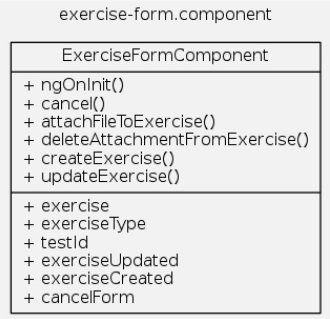
Komponenty: *Ćwiczenia*

Każdy rodzaj ćwiczenia jest podzielony na dwa komponenty: formularz dla Administratora (dodawanie i edycja) oraz ten wyświetlany Uczniowi. W przypadku ćwiczenia „wybór” będą to odpowiednio: *BracketsForm* oraz *StudentBrackets*. Ćwiczenia pogrupowano kierując się logiką prezentacji, a nie typem, zatem wszystkie komponenty zawierające formularz są zamknięte w katalogu *exercise-forms*, a te wyświetlane Uczniom w *student-exercises*.

Taki podział spowodowany jest tym, że w obu przypadkach zastosowano wzorzec *Strategia*[4, s. 321]. Kontekstem *Strategii* jest tutaj klasa *ExerciseFormComponent*, która jako parametr przyjmuje *exerciseType* determinujący rodzaj wyświetlanego i obsługiwanego formularza. Komponent kontekstowy tworzy klasę *Exercise*, której pola są połączone z elementami formularza. Komponent formularza, reprezentujący konkretną implementację *Strategii*, zawiera logikę kreacyjną struktury danych ćwiczenia. W komponencie kontekstowym zawarty jest także mechanizm utrwalania danych klasy *Exercise* w warstwie backendowej.

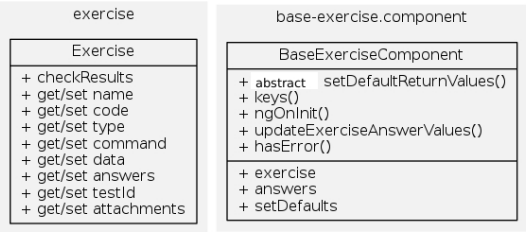
Te same komponenty przystosowane są do edycji konkretnego ćwiczenia – wtedy

komponent *ExerciseFormComponent* otrzymuje jako parametr klasę *Exercise*. W tym przypadku założenie wzorca *Strategia* o wymienności algorytmu zostaje złamane – ćwiczenie o strukturze danych szablonu „luki” (*BracketsFormComponent*) nie zostanie poprawnie wyświetlone w formularzu szablonu „wybór” (*ChoiceFormComponent*) ze względu nie niekompatybilność struktury danych z szablonem komponentu. Przypadek ten jest świadomym działaniem, gdyż niewielkim kosztem wykorzystano istniejące mechanizmy do obsługi niezbędnej funkcji edycji ćwiczenia, przy jednoczesnym respektowaniu zasady *DRY* oraz uniknięciu definiowania nadmiaru komponentów.



Rysunek 2.4. Diagram klasy *ExerciseFormComponent*

Należy również wspomnieć o tym, że implementacja *Strategii* w tym przypadku różni się od klasycznej, ze względu na charakterystyczną architekturę frameworka *Angular 2*. Komponenty nie są wstrzykiwane w konstruktor, a tworzone bezpośrednio w szablonie, gdzie instrukcja warunkowa *ngSwitch* decyduje o tym, który komponent ma zostać wyrenderowany. Istnieje możliwość tworzenia instancji komponentów wewnątrz samej klasy komponentu-kontenera, w alternatywie do szablonu i poleganiu na implicytnych mechanizmach frameworka, jednak porzucono tę metodę ze względu na osobiste preferencje dotyczące czytelności kodu.



Rysunek 2.5. Diagramy klas *Exercise* i *StudentExerciseComponent*

W przypadku komponentów ćwiczeń wyświetlanych Uczniowi (znajdujących się w katalogu *student-exercises*), zastosowano wzorzec *Metoda szablonowa*[4, s. 264]. Komponenty formularza Ucznia dziedziczą po klasie *BaseExerciseComponent*), która oprócz implementowania wspólnych metod publicznych definiuje również metodę *setDefaultReturnValues*, która jest wywoływana we wbudowanej metodzie *ngOnInit*[2].

2.3.2. Serwisy

Serwisy (klasy usługowe) w projekcie dzielą się na klasy pomocnicze komponentów oraz niezależne, współdzielone pomiędzy komponentami i ich klasami usługowymi.

Pierwszy typ służy do wykonywania niezbędnych do działania, typowych dla komponentu operacji, a umieszczone są w tych samych katalogach, co komponenty – są ich integralną częścią. Przykładem takiego obiektu jest klasa *ExerciseFormService* i jej metody *attachFile*, wysyłająca załącznik na serwer, oraz *deleteAttachment*, służąca do usuwania powiązania ćwiczenia z plikiem.

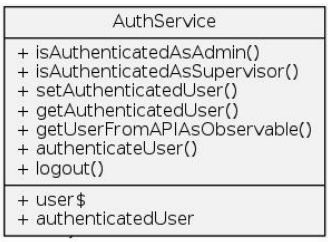
Drugi rodzaj został wyodrębniony, gdyż logika przez nie reprezentowana jest wspólna dla całego projektu. Zasadniczo są to obiekty odpowiedzialne za obsługę żądań HTTP oraz przetwarzanie danych zwracanych z API na klasy encji, po czym zwracane w formie *Promise* lub *Observable*. Niektóre z nich, jak np. *UserService*, wykorzystują dedykowane dla nich inne klasy formatujące strukturę żądań POST i PATCH do zgodnej ze standardem JSON API. Należy dodać, że pod koniec tworzenia projektu pojawiła się biblioteka pozwalająca na tworzenie modeli, które obsługiwałyby

[2]Metoda wywoływana automatycznie przez aplikację po utworzeniu komponentu i ustawieniu pól dekorowanych przez *@Input*.

taką logikę, jednak jej poziom pokrycia testami był niewielki i zarzucono pomysł jej implementacji w projekcie.

AuthService

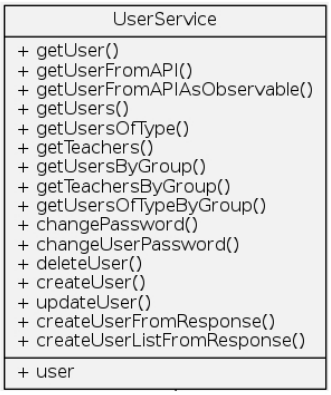
Klasa odpowiedzialna za obsługę procesu autentykacji oraz przechowująca obiekt zalogowanego użytkownika. Jest wykorzystywana w wielu miejscach, gdzie elementy interfejsu są wyświetlane warunkowo w zależności od typu konta. Zazwyczaj w takim komponencie wywoływana jest metoda serwisu *getAuthenticatedUser()*. Jest zależna od serwisu *UserService*, który udostępnia jej interfejs komunikacyjny z API.



Rysunek 2.6. Diagram klasy *AuthService*

UserService

Przykład klasy komunikacyjnej implementującej wzorzec *Repozytorium*. Zawiera metody CRUD dla endpointów zasobu API *users* oraz wykonuje żądania POST do wirtualnego zasobu *password-updates*, który jest odpowiedzialny za zmianę hasła użytkownika.



Rysunek 2.7. Diagram klasy *UserService*

2.4. Testy

Zarówno w testach jednostkowych jak i funkcjonalnych wykorzystywane są atrapy (ang. *mocks*), których część została wydzielona do dedykowanych klas w celu zapobieżenia zbędnej duplikacji kodu[3].

2.4.1. Testy jednostkowe

Testy API

Jednostkowo są testowane klasy modelowe oraz klasy obliczające wynik rozwiązywanych ćwiczeń (ewaluatory). Testowanie klas modelowych skupia się na weryfikacji reguł walidacyjnych, a ewaluatory testowane są pod kątem strategii sprawdzania poprawności odpowiedzi.

Testy Klienta

W izolacji testowane są metody serwisowe, które nie wysyłają żądań do API. Jako przykład takich metod można zaliczyć funkcje formatujące strukturę obiektów, przygotowujące je do wysłania żądania POST (metody te zostały wydzielone do

[3]Znajdują się one w katalogu *src/app/services/testing*.

osobnych klas serwisowych). Reprezentantem takich testów są testy klasy *UserPostDataService*.

2.4.2. Testy funkcjonalne

Z powodu charakterystyki architektury aplikacji oraz wykorzystanych technologii, testy funkcjonalne są głównym narzędziem testowania. Jest to spowodowane faktem, iż większość kodu API stanowi definiowanie relacji oraz zachowań zasobów, a te nie są testowalne jednostkowo. W przypadku Klienta autorzy frameworka *Angular 2* sugerują testowanie komponentów[9] funkcjonalnie, gdyż wtedy możliwe jest również testowanie interakcji pomiędzy komponentami. Należy jednak nadmienić, że podczas testowania komponentów wykorzystywane są atrapy niektórych usług w celu uniknięcia wywołań żądań do API.

Testy API

Bardzo ważne są testy kontrolera *UsersController*, gdyż nieuprawniony dostęp do informacji o innych kontach przez użytkowników może skutkować sankcjami prawnymi z powodu złamania przepisów Ustawy o ochronie danych osobowych[4].

Testy kontrolerów *CoursesController* oraz *TestsController* weryfikują, czy dany Uczeń ma dostęp wyłącznie do przypisanych mu zasobów – jest to sprawdzenie zabezpieczeń dla autora treści.

[4] Dz.U. 1997 nr 133 poz. 883 z późn. zm., art. 36 i art. 51

Tabela 2.1. Testy kontrolera *UserController*

Przypadek	**Oczekiwany rezultat**
Użytkownik niezalogowany wysyła żądanie na endpoint */users*	Zwracana jest odpowiedź z kodem HTTP 401
Uczeń 1 wysyła żądanie GET na endpoint */users*	Zwracana jest odpowiedź z kodem HTTP 200 i wynik zawierający jedynie rekord Ucznia 1
Uczeń 1 wysyła żądanie GET na endpoint */users/2*	Zwracana jest odpowiedź z kodem HTTP 404
Uczeń 1 wysyła żądanie POST na endpoint */users/2*	Zwracana jest odpowiedź z kodem HTTP 404
Administrator wysyła żądanie na endpoint */users/2*	Zwracana jest odpowiedź z kodem HTTP 200 i wynikiem zawierającym dane Ucznia 2

Testy Klienta

Testami funkcjonalnymi dla Klienta są testy komponentów. Nie są one testowane w izolacji, gdyż celem testowania komponentów jest badanie ich interakcji z innymi komponentami oraz renderowania szablonów. Najważniejsze są tu testy sprawdzające, czy dla odpowiednich typów kont wyświetlane są dedykowane im elementy interfejsu oraz czy szablony ćwiczeń renderują się poprawnie. W ramach przykładu przytoczono tu testy komponentów *AppComponent* (weryfikacja elementów interfejsu) oraz *StudentBracketsComponent* i *StudentChoiceComponent* (komponenty formularzy ćwiczeń wyświetlanych Uczniowi).

Tabela 2.2. Testy kontrolera *CoursesController*

Przypadek	**Oczekiwany rezultat**
Użytkownik niezalogowany wysyła żądanie na endpoint */courses*	Zwracana jest odpowiedź z kodem HTTP 401
Uczeń 1 wysyła żądanie GET na endpoint */users/1/courses*	Zwracana jest odpowiedź z kodem HTTP 200 i wynik zawierający rekordy kursów przypisanych do Ucznia 1 (poprzez grupy)
Uczeń 1 wysyła żądanie GET na endpoint */users/2/courses*	Zwracana jest odpowiedź z kodem HTTP 404
Uczeń 1 wysyła żądanie POST na endpoint */courses/1*	Zwracana jest odpowiedź z kodem HTTP 400
Administrator wysyła żądanie POST na endpoint */courses/1*	Zwracana jest odpowiedź z kodem HTTP 400

Tabela 2.3. Testy komponentów aplikacji klienckiej

Przypadek	**Oczekiwany rezultat**
AppComponent	
Użytkownik jest zalogowany jako Uczeń	Wyświetlane są elementy nawigacji: „Podsumowanie”, „Kursy”, „Dane konta”
Użytkownik jest zalogowany jako Nauczyciel	Wyświetlane są elementy nawigacji: „Najnowsze wyniki”, „Kursy”, „Grupy”, „Dane konta”
Użytkownik jest zalogowany jako Kierownik	Wyświetlane są elementy nawigacji: „Podsumowanie”, „Nauczyciele”, „Uczniowie”, „Grupy”, „Dane konta”
Użytkownik jest zalogowany jako Administrator	Wyświetlane są elementy nawigacji: „Statystyki”, „Użytkownicy”, „Kursy”, „Grupy”, „Dane konta”
StudentBracketsComponent	
Ćwiczenie „luki” z jedną luką pierwszym zdaniu i trzema lukami w drugim zdaniu	Wyświetlane są cztery elementy typu *input*
Ćwiczenie „luki” z jedną luką pierwszym zdaniu i trzema lukami w drugim zdaniu	Komponent zawiera pole *answers* o analogicznej strukturze
StudentChoiceComponent	
Ćwiczenie „wybór” z pięcioma pytaniami, każde zawierające 3 możliwe odpowiedzi	Wyświetlanych jest 15 elementów *input* wraz z ich kontenerami
Ćwiczenie „wybór” z pięcioma pytaniami, każde zawierające 3 możliwe odpowiedzi	Wyświetlane są odpowiedzi w kolejności analogicznej do tej w strukturze obiektu

Zakończenie

Stworzona aplikacja została z powodzeniem wdrożona w szkole językowej – ponad 470 uczniów skorzystało z sześciu przygotowanych kursów. Projekt będzie wykorzystywany również w przyszłości i w związku z tym planowany jest jego aktywny rozwój. Należy również dodać, iż niektóre elementy ulegną przepisaniu, na przykład metody w klasach serwisowych zwracające obiekty *Promise*, które docelowo będą wykorzystywały mechanizm *Observable* z biblioteki *RxJS*.

Podczas wdrożenia pojawiły się również problemy, które czekają na rozwiązanie. Przykładem takiego problemu jest aktualizacja testu i jej kompatybilność z już przesłanymi ćwiczeniami. W przypadku modyfikacji struktury obiektów reprezentujących testy i odpowiedzi, może wystąpić ich desynchronizacja. Propozycją rozwiązania takiej sytuacji jest taka modyfikacja struktury bazy danych, aby przechowywać historię edycji w formie snapshotów. Zostanie to zaimplementowane w przyszłości.

Praca z frameworkiem *Angular 2* oraz *Ruby on Rails 5* była satysfakcjonująca. Jednym z priorytetów było stworzenie działającej wersji w możliwie szybkim czasie i w efekcie powstała ona w 3 miesiące. Następnie wersja została poddana refaktoryzacji optymalizującej pewne mechanizmy (np. logikę autentykacyjną) oraz porządkującą hierarchię klas.

Istnieje kilka perspektyw kierunków dalszego rozwoju projektu, o których warto wspomnieć. Jednym z priorytetów jest rozszerzenie zbioru szablonów ćwiczeń. W planach są implementacje gry *Memory*, parowania fraz, układania wyrazów w kolejności. Również uatrakcyjnienie warstwy frontendowej przyczyni się do lepszego odbioru prezentowanych ćwiczeń przez uczniów, zwłaszcza wśród młodszych roczników. Aby zrealizować ten cel, rozważane jest przepisanie szablonów pod kątem wykorzystania frameworka CSS *Bootstrap 4*, jednak nastąpi to po wydaniu jego stabilnej wersji.

W trakcie tworzenia projektu i pisania pracy autorzy frameworka *Angular 2* zdecydowali się zmienić system wersjonowania i wydali wersję 4. Oprócz aktualizacji frameworka i dostosowania kodu do aktualnej wersji, rozważane jest również skorzystanie z innego frameworka opartego na *Angular 4 – Ionic Framework*, który pozwala tworzyć natywne aplikacje na urządzenia mobilne.

Bibliografia

[1] A. Builes, *JSON API By Example*, Leanpub, 2016.

[2] A. Lerner, F. Coury, N. Murray, C. Taborda, *ng-book 2*, Fullstack.io, 2017.

[3] Browser support. (20 maja 2017). Dostępne pod adresem https://v2.angular.io/docs/ts/latest/guide/browser-support.html

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku*, Helion, 2010.

[5] M. Fowler. (20 maja 2017). Repository. Dostępne pod adresem https://martinfowler.com/eaaCatalog/repository.html

[6] M. Kusiak-Pisowacka, *Ewaluacja podręcznika w nauczaniu języków obcych*, Lingwistyka Stosowana 14, 3/2015.

[7] K. Beck, *TDD. Sztuka tworzenia dobrego kodu*, Helion, 2014.

[8] R. C. Martin, *Czysty kod. Podręcznik dobrego programisty*, Helion, 2014.

[9] Testing. (20 maja 2017). Dostępne pod adresem https://v2.angular.io/docs/ts/latest/guide/testing.html#!#isolated-v-testing-utilities

Spis tabel

Spis rysunków

Oświadczenie

Ja, niżej podpisany(a) oświadczam, iż przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób.

.....	.....
data	podpis