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. Każda taka jednostka ma możliwość uruchomienia własnej instancji oraz wypełnienia jej treścią. 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 odpowiąda *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*.

W części poświęconej projektowi aplikacji zostały opisane główne założenia, wraz z analizą potrzeb użytkowników. Opisano typy kont: administratora, kierownika, nauczyciela, ucznia. Przedstawiono możliwości przypisane każdemu rodzajowi konta. W dalszej części zaprezentowano diagramy obrazujące strukturę bazy danych. W szczegółach implementacyjnych przedstawiono architekturę i wykorzystane technologie każdej z warstw aplikacji. Opisano biblioteki i narzędzia, umieszczono diagramy klas. Następnie poruszono problematykę testowania aplikacji. Przedstawiono scenariusze testów funkcjonalnych oraz opisano wybrane testy jednostkowe.

Słowa kluczowe

języki obce, e-learning, 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 konkretnej metody nauczania. 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 materiału według przyjętego przez autorów programu nauczania [1]. Zazwyczaj jest to udogodnienie dla lektorów, gdyż nie muszą przygotowywać materiałów do zajęć (lub przygotowują ich mniej). Uczniowie również korzystają z tego, że przy tworzeniu takich podręczników biorą udział profesjonalni metodycy nauczania, zatem mają dostęp do pomocy dydaktycznych wysokiej jakości.

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 sprawdzonych 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: dedykowane dla konkretnych podręczników oraz samodzielne oprogramowanie, które zazwyczaj funkcjonuje w formie serwisów webowych. Istniejącymi na rynku rozwiązaniami z tej drugiej grupy są na przykład *busuu* i *Duolingo*. Umożliwiają one rozwiązywanie ćwiczeń oraz zapoznawanie się z przygotowanymi materiałami dydaktycznymi,

ponadto zawierają szereg udogodnień, 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, 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 stworzenie aplikacji, dzięki której podmioty zajmujące się nauczaniem języków obcych przy pomocy autorskich rozwiązań mogłyby skorzystać z możliwości e-learningu. Twórca podręcznika może tworzyć 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.

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 z obowiązków przez lektorów.
- **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. Konta są zakładane przez Administratora, w aplikacji nie ma przewidzianej możliwości samodzielnej rejestracji, 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.

Przyczyną takiej klasyfikacji jest to, że Administrator jest z założenia kontem o największych możliwościach i to on odpowiada za treść oraz konta pozostałych Użytkowników, zaś Uczeń jest ostatecznym konsumentem aplikacji, gdyż to z myślą o nich 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.

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, Uczeniowi 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, dlatego

- **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*. Ponadto użytkownik nie powinien być zmuszony do instalacji w swoim systemie żadnych rozszerzeń ani wtyczek.

1.2.4. Przypadki użycia

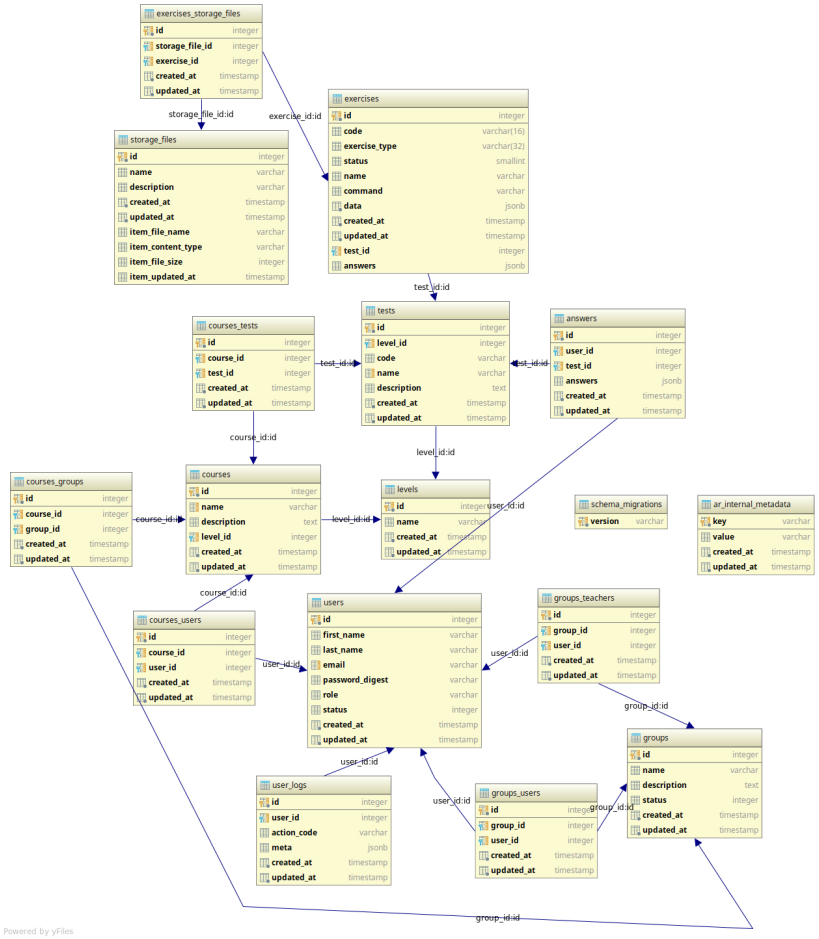
UC5	Dodawanie ćwiczenia ”luki” do testu
Aktor	Administrator
Warunki	Administrator w widoku edycji testu.
Przebieg	1. Administrator w sekcji „Ćwiczenia w teście” wybiera rodzaj ćwiczenia „luki” i zatwierdza wybór. Pojawia się formularz dodawania ćwiczenia. 2. Administrator wypełnia dane ćwiczenia. 3. 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”.

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”.

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.

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

2.1.1. Wykorzystane biblioteki

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 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*. Z szeregu udogodnień zapewnianych przez bibliotekę można wymienić:

- **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.

2.2. Aplikacja kliencka

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.

Wykorzystane biblioteki i narzędzia

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

- **Angular CLI** – ważne narzędzie pomagającym przy procesie wytwarzania aplikacji napisanych we frameworku *Angular 2*. Nie jest 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*), budujące projekt oraz testujące kod (wykorzystujące narzędzia *Karma*, *Jasmine* oraz *Protractor*). Narzędzie *webpack-dev-server* jest szczególnie pomocne podczas pisania kodu, gdyż dzięki funkcji *live reload* programista może na bieżąco 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 implementuje 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.
- **Chart.js oraz ng2-charts** – narzędzie do wizualizacji danych w formie responsywnych wykresów.

2.3. Architektura

Głównym komponentem aplikacji jest *AppComponent*, który jest kontenerem dla całej reszty komponentów. Konkretne elementy i funkcje zostały zamknięte w modułach (dekorowanych przez *@NgModule*). Poniżej znajduje się lista modułów dedykowanych dla aplikacji[1] wraz z krótkim opisem zakresu funkcji, które obejmują:

- **AnswerModule** – komponenty prezentujące rozwiązane testy oraz listę rozwiązanych testów,

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

- **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.

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.

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 również 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*[2]. 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 również mechanizm utrwalania danych klasy *Exercise* w warstwie backendowej.

TODO: jakis prosty diagram

Te same komponenty przystosowane są również 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.

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

[2] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Wzorce projektowe: Elementy oprogramowania obiektowego wielokrotnego użytku*, Helion 2010, s. 321

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 alternatywnie do szablonu i poleganiu na implicytnych mechanizmach frameworka, jednak porzucono tę metodę ze względu na osobiste preferencje dotyczące czytelności kodu.

Diagram klasy

W przypadku komponentów ćwiczeń wyświetlanych Uczniowi (znajdujących się w katalogu *student-exercises*), zastosowano wzorzec *Metoda szablonowa*[3]. Komponenty formularza Ucznia dziedziczą po klasie *BaseExerciseComponent*), która oprócz implementowania wspólnych metod publicznych definiuje również metodę *setDefaultReturnValues*, która jest wywoływana we wbudowanej metodzie *ngOnInit*[4].

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 obiekty DTO (ang. *Data Transfer Objects*), 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

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Wzorce projektowe: Elementy oprogramowania obiektowego wielokrotnego użytku*, Helion 2010, s. 264

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

ze standardem JSON API. Należy dodać, że pod koniec tworzenia projektu pojawiła się biblioteka pozwalająca na tworzenie modeli, które obsługiwałyby 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.

UserService

2.4. Testy

2.4.1. Testy jednostkowe

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

Testy API

Testy Klienta

2.4.2. Testy funkcjonalne

Testy API

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.

Zakończenie

Wnioski

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