

UNIwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki

Mateusz Kwiatkowski

nr albumu: 194 925

Walidacja w elektronicznym systemie zarządzania osiągnięciami studenta

Praca magisterska na kierunku:

INFORMATYKA

Promotor:

dr Włodzimierz Bzyl

Gdańsk 2015

Streszczenie

Pracę poświęcono zagadnieniu walidacji, kwestii ważnej i integralnie związanej z odpowiednim funkcjonowaniem sieci. W szczególności zwrócono uwagę na aspekt prawidłowego zarządzania jej jakością, co obligatoryjnie wiąże się z problemem odpowiedniego zabezpieczenia i odpowiedzialnego korzystania z niej.

Na potrzeby pracy powstała aplikacja *Elektroniczny indeks*, w której zostało zaimplementowane przeglądanie ocen przez studenta, wykładowcę oraz pracownika dziekanatu, wystawianie ocen przez wykładowcę oraz przegląd studentów, a także ich osiągnięć na zajęciach, które dany wykładowca prowadzi. Program został również przetestowany w celu weryfikacji czy funkcje działają prawidłowo. Funkcja zapisu studenta na zajęcia i dodawanie nowych użytkowników oraz przedmiotów dostępna jest jedynie dla administratora systemu.

Aplikacja jest dostępna pod adresem eindeks.meteor.com, a dostęp do jej funkcjonalności można uzyskać logując się na jedno z poniższych kont.

Dane do logowania		
Klasa	Login	Hasło
Administrator	admin	abcdef
Wykładowca	wbzył	abcdef
Biurokrata	iszreder	abcdef
Student	194925	abcdef

W ramach pracy powstał pakiet *Validation*¹, który zapewni aplikacji poprawne działanie. W aplikacji nie zostało zaimplementowane automatycznie przypisywanie loginu studentowi, podział studentów oraz zajęć na grupy, wyszukiwanie użytkowników oraz zapisywanie się na zajęcia przez studentów, ponieważ funkcjonalność ta nie jest niezbędna w tej pracy.

¹Pakiet *Validation* dostępny jest na *Atmospherejs*

Spis treści

Wprowadzenie	5
1. Walidacja oprogramowania	9
1.1. Rozróżnienie walidacji i weryfikacji	9
1.2. Specjalistyczny wstęp do walidacji	9
2. Elektroniczny indeks	11
2.1. Filtrowanie danych	11
2.2. Zarządzanie elektronicznym indeksem przez administratora	12
2.3. Funkcjonalność dla prowadzącego zajęcia	15
2.4. Funkcjonalność dla studenta	16
2.5. Funkcjonalność dla pracownika dziekanatu	17
2.6. Importowanie i eksportowanie danych	18
3. Aplikacja Elektroniczny indeks w Meteor	19
3.1. Opis tworzenia aplikacji	19
3.2. Opis testowania aplikacji	24
4. Pakiet walidujący operacje elektronicznego indeksu	27
4.1. Funkcjonalność pakietu	27
4.2. Opis tworzenia pakietu	27
4.3. Przetestowanie pakietu	31
4.4. Udostępnienie pakietu	34
Zakończenie	35
Bibliografia	37
Spis rysunków	39
Oświadczenie	41

Wprowadzenie

Wprowadzając dane do systemu, użytkownik może – świadomie lub nie – popełnić pomyłkę. Jeżeli dane odebrane przez użytkownika poddamy przetworzeniu bez walidacji, wówczas, w zależności od odporności aplikacji, możemy mieć do czynienia z różnymi rodzajami błędów, od drukowania w przeglądarce klienta komunikatów diagnostycznych, poprzez utratę spójności bazy danych, aż po ujawnienie niepowołanym użytkownikom informacji poufnych. Z tego powodu nie wolno ignorować wagi problemu.

Aplikacje pozbawione walidacji pozwalają użytkownikowi na wprowadzenie niespójnych danych do systemu. Przykładem takiej aplikacji jest elektroniczny indeks. Operacje, takie jak: wystawianie studentowi ocen z ćwiczeń czy też oceny z egzaminu kończącej edukację z danego przedmiotu, dodawanie nowych użytkowników, a także przedmiotów powinny być odpowiednio walidowane. Dzięki temu nie dojdzie do niepożądanych zjawisk typu:

- student nie uzyskał pozytywnej oceny z ćwiczeń, a otrzymuje ocenę z egzaminu kończącego przedmiot,
- student otrzymuje ocenę spoza skali oceniania systemu danej uczelni,
- student uzyskuje ocenę od osoby nieuprawnionej do jej wystawienia,
- nowy przedmiot nie ma przypisanego prowadzącego,
- tworząc nowego użytkownika nie wprowadzamy wymaganych danych.

W celu zilustrowania przydatności walidacji podczas korzystania z elektronicznego systemu zarządzania osiągnięciami studenta, pokazano w pracy działanie tego zjawiska w aplikacji stworzonej w frameworku *Meteor* oraz zaprezentowano ułożony pakiet oraz wyjaśniono, jak go udostępnić. W pracy użyto framework *Meteor* w wersji 1.1.0.3. Jest to framework javascriptowy, który zapewnia aplikacji działanie w czasie rzeczywistym¹, dzięki czemu użytkownikom korzystającym

¹ang. Real-Time Application - RTA

z aplikacji widoki aktualizują się natychmiast, gdy w bazie danych zachodzą zmiany. Dodatkowo należy podkreślić, że atutem tej technologii jest fakt, że wszystko, zarówno back-end jak i front-end, piszemy w taki sam sposób, korzystając jedynie z javascriptu. *Meteor* pozwala również zaoszczędzić czas, dostarczając deweloperom gotowe rozwiązania w postaci pakietów. Framework ten do przechowywania danych używa nierelacyjnej bazy danych, *MongoDB*[11]. Charakteryzuje się dużą skalowalnością, wydajnością oraz brakiem ściśle zdefiniowanej struktury obsługiwanych baz danych. Zamiast tego, dane składowane są jako dokumenty w stylu JSON, co umożliwia aplikacjom bardziej naturalne ich przetwarzanie, przy zachowaniu możliwości tworzenia hierarchii oraz indeksowania. *Meteor* od wersji 1.0 dostarcza narzędzie do testowania *Velocity*. Pozwala ono tworzyć testy w popularnych bibliotekach, takich jak: *Mocha*, *Jasmine*, *Cucumber* czy *Selenium*. W pracy wykorzystana została biblioteka *Jasmine*. Jednak same testy nie wystarczą, aby zapewnić aplikacji poprawne działanie. Korzystając z testów jednostkowych sprawdzamy czy funkcja wykonuje interesującą nas operację prawidłowo, ale bez walidacji wywołanie funkcji może się zakończyć powodzeniem także dla błędnych danych.

Aplikacja korzysta również z szeregu dodatkowych pakietów takich jak:

- *Meteor-roles* jest to pakiet autoryzujący do frameworka *Meteor*, który pozwala zarządzać, jakie dane zostaną wysłane do konkretnych grup użytkowników,
- *Iron router*, którym definiujemy, jak ma wyglądać mapa strony. Działa on zarówno po stronie serwera i klienta. Routing po stronie klienta sprawia, że aplikacja jest naprawdę szybka, gdy już jest załadowana, ponieważ przy każdej zmianie podstrony, nie trzeba jej całej generować,
- *Meteor account* jest to kompletny pakiet zarządzania kontami użytkowników. Jedną linią kodu można zapewnić aplikacji możliwość logowania, tworzenia kont, walidacji email, przywracania hasła czy logowania się przez zewnętrzne serwisy jak *facebook* czy *twitter*. Dodatkowo pakiet daje możliwość dostosowania go pod własne potrzeby,
- *Underscore.string* jest to pakiet służący do manipulacji stringami,

- *Meteor-file-collection* to pakiet, który rozszerza system kolekcji, pozwalając obsługiwać także dane z plików.

Walidacja oprogramowania

W testowaniu oprogramowania ważne są pojęcia – weryfikacja i walidacja. Pojęcia te są znaczeniowo na tyle bliskie, że mogą przysporzyć trudności. Zarówno weryfikacja jak i walidacja produktu są czynnościami, które służą sprawdzeniu, czy wytworzony produkt jest taki, jaki sobie życzyliśmy my, bądź inny interesariusz. Warto wyjaśnić oba te pojęcia.

1.1. Rozróżnienie walidacji i weryfikacji

Najprościej jest zapamiętać, że weryfikujemy to, co da się wyliczyć, wykazać logicznie i nie ma jak zanegować tego, co już zweryfikowane. Przy dobrze spisanych wymaganiach weryfikacji dokonać może każdy człowiek rozumiejący tekst specyfikacji i wiedzący, jak wykonać test. Weryfikacja polega na dostarczeniu dowodów, że dany produkt spełnia zdefiniowane wymagania. Natomiast walidacja jest po stronie odbiorcy i to zaspokojenie jego potrzeb jest ostatecznym kryterium sukcesu. Polega na sprawdzeniu poprawności i dostarczeniu dowodów, że produkt procesu wytwarzania spełnia potrzeby i wymagania użytkownika.[3]

1.2. Specjalistyczny wstęp do walidacji

Każdy system informatyczny wymaga osiągnięcia odpowiedniego stopnia adekwatności, bezbłędności, stabilności oraz wyeliminowania błędów działania w modelu. Przez model, który przeszedł walidację, rozumieć należy ten, który został poddany serii operacji, mających na celu doprecyzowanie go do optymalnego poziomu, przez co, zgodnie z jego przeznaczeniem, będzie mógł sprostać postawionym przed nim zadaniom. Taki poziom wiarygodności modelu uzyskamy dzięki procesowi walidacji.

Systemy zautomatyzowane i skomputeryzowane stosowane szczególnie w przemyśle wysokich technologii, muszą być poddawane okresowym kontrolom, potwierdzającym ich jakość w celu wykrycia ewentualnych, potencjalnych zagrożeń, wynikających z bezpośredniego lub pośredniego wpływu na produkt końcowy. Walidacja zatem ma udokumentować, w jaki sposób należy zmienić i udoskonalić proces, aby zminimalizować ewentualne skutki jego nieprawidłowego działania, a także zredukować koszty wytwarzania oprogramowania.[4][5][6][7]

Elektroniczny indeks

Elektroniczny indeks jest to platforma, która służy do wystawiania ocen studentom przez prowadzących zajęcia w czasie rzeczywistym, a także do wglądu do tych ocen przez studentów. Dzięki wdrożeniu do użytku wspomnianej platformy, zarówno studenci jak i prowadzący, mogą zaoszczędzić mnóstwo czasu oraz nerwów związanych z próbą zdobycia wpisu, odtworzenia indeksu, gdy zostanie on zgubiony lub uniknięcia długiego stania w kolejce do dziekanatu. Dodatkowo znacznie ułatwiona zostanie biurokracja na linii dziekanat – wykładowca, a samo zaliczenie semestru studentowi przez osoby do tego uprawnione staje się dużo prostsze i szybsze.

2.1. Filtrowanie danych

Filtrowanie danych w elektronicznym indeksie jest jego ważną częścią. Dzięki temu można uniknąć sytuacji, w której nieautoryzowany użytkownik uzyska dostęp do danych, do których nie powinien mieć dostępu. *Meteor* nie ma domyślnie zaimplementowanego pakietu, który pozwoliłby na sprawne zarządzanie dostępnością treści w aplikacji. Dzięki rosnącej popularności frameworku *Meteor* oraz coraz szerszej grupie deweloperów z każdym dniem pojawiają się nowe pakiety, które ułatwiają i zwiększają funkcjonalność tworzonych aplikacji. Do efektywnego przekazywania danych poszczególnym użytkownikom wykorzystano pakiet *meteor-roles*, który pozwala zarządzać rozsyłanymi danymi oraz jest kompatybilny z wbudowanym pakietem do zarządzania kontami. Instalacja pakietu do zarządzania rolami wymaga dołączenia do projektu pakietu *accounts-password*, co można zrobić następującym poleceniem w konsoli:

Listing 2.1: Instalacja accounts-password

```
meteor add accounts-password
```

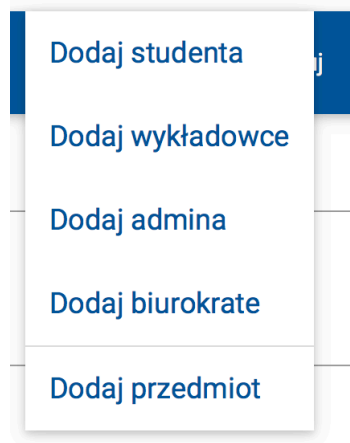
Następnie możemy dodać właściwy pakiet

Listing 2.2: Instalacja pakietu roles

```
meteor add alanning:roles
```

2.2. Zarządzanie elektronicznym indeksem przez administratora

Administrator jest osobą, która nad wszystkim czuwa, dlatego musi mieć dostęp do wszystkich danych i wszystkich funkcjonalności programu, aby móc kontrolować jego poprawność działania. Będąc głównym zarządcą platformy, administrator jako jedyny ma prawo dodawać do systemu nowe dane. Dzięki temu, że najważniejsze dane wyświetlają się administratorowi od razu po zalogowaniu do systemu, jest on w stanie szybko wykonać powierzone mu zadanie.



Rysunek 2.1: Menu dodawania

Źródło: Własne

Jako główny zarządca, administrator ma za zadanie przypisywać studentów do

przedmiotów, na które są zobowiązani uczęszczać, a także do przedmiotów, które sami wybiorą w trakcie toku nauczania. Student, który został zapisany na dany przedmiot, nie pojawi się ponownie na liście, dzięki czemu nie ma możliwości zapisania danego użytkownika dwa razy na te same zajęcia.

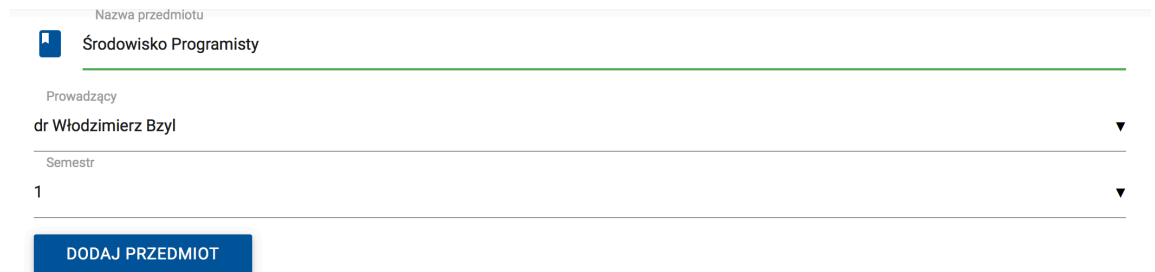
Zapisz na przedmiot

Indeks	Imię	Nazwisko	
194955	Jakub	Bełcik	
186506	Dominik	Białkowski	
194979	Damian	Brzeziński	
194950	Jacek	Dermont	

Rysunek 2.2: Fragment listy studentów do zapisu na zajęcia

Źródło: Własne

Kolejnym zadaniem jest dodawanie przedmiotów, które odbywają się na uczelni. Administrator podaje jego nazwę, wybiera semestr, na którym dane zajęcia się odbywają i z rozwijanej listy wybiera prowadzącego dane wykłady czy ćwiczenia.



Nazwa przedmiotu

Środowisko Programisty

Prowadzący

dr Włodzimierz Bzyl

Semestr

1

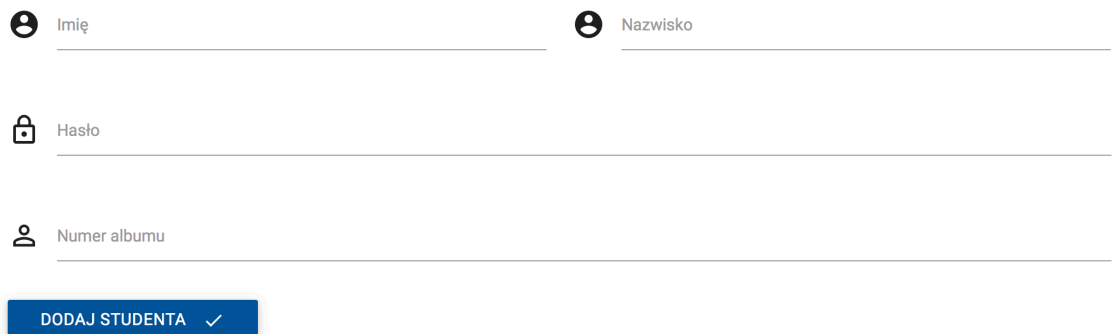
DODAJ PRZEDMIOT

Rysunek 2.3: Formularz dodawania przedmiotu

Źródło: Własne

Następną równie ważną funkcją jest dodawanie studentów, prowadzących zajęcia, adminów oraz pracowników dziekanatu. Z menu dodawania admin wybiera, do jakiej grupy chce dodać użytkownika.

1. Dodawanie studenta



Imię

Nazwisko

Hasło

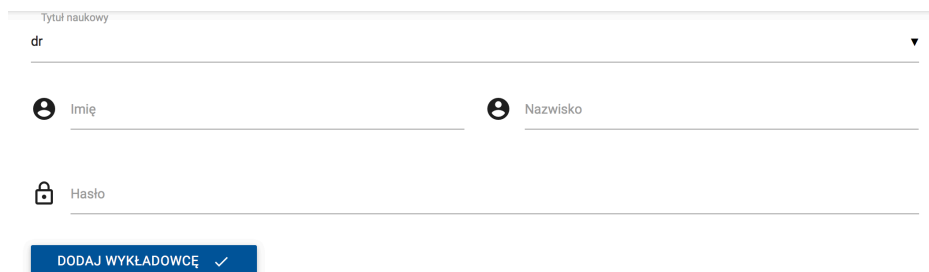
Numer albumu

DODAJ STUDENTA ✓

Rysunek 2.4: Formularz dodawania studenta

Źródło: Własne

2. Dodawanie wykładowcy

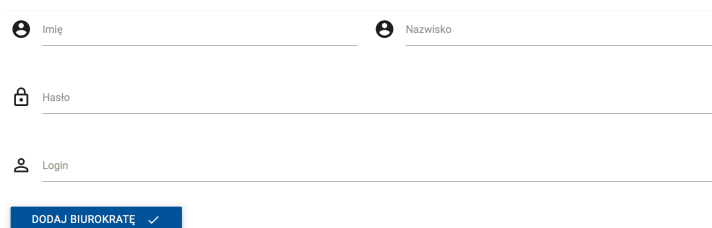


Formularz dodawania wykładowcy. Zawiera pole wyboru tytułu naukowego (aktualnie 'dr'), pola tekstowe na imię i nazwisko, pole hasła oraz przycisk 'DODAJ WYKŁADOWCĘ' z ikoną potwierdzenia.

Rysunek 2.5: Formularz dodawania wykładowcy

Źródło: Własne

3. dodawanie biurokraty



Formularz dodawania pracownika dziekanatu. Zawiera pola tekstowe na imię i nazwisko, pole hasła oraz pole loginu, oraz przycisk 'DODAJ BIUROKRATĘ' z ikoną potwierdzenia.

Rysunek 2.6: Formularz dodawania pracownika dziekanatu

Źródło: Własne

Jako login dla studenta posłuży jego numer albumu, natomiast wykładowca oraz pracownik dziekanatu logować się będzie do systemu dzięki loginowi składającemu się z pierwszej litery imienia oraz nazwiska bez używania polskich znaków.

2.3. Funkcjonalność dla prowadzącego zajęcia

Użytkownicy należący do grupy wykładowców muszą mieć możliwość do zarządzania listą studentów, na której widnieją ich nazwiska przypisane do przedmiotów, które prowadzą. Wykładowca ma dostęp wyłącznie do przedmiotów przez

siebie prowadzonych. Dzięki temu, podobnie jak w przypadku administratorów, do interesujące wykładowcę dane, można uzyskać w prosty sposób, od razu po zalogowaniu. A prowadzący jest w stanie efektywnie zarządzać swoimi podopiecznymi. Na stronie głównej wykładowca znajdzie listę przedmiotów, które wykłada.

Przedmioty
Bogaty interfejs użytkownika w aplikacjach webowych
JVM Internals
Języki programowania
Technologie Java Enterprise
Technologie utrwalania danych dla języka Java

Rysunek 2.7: Lista przedmiotów wykładowcy

Źródło: Własne

Po wybraniu przez użytkownika interesującego go przedmiotu na ekranie pojawi się lista studentów, zapisanych na dane zajęcia. Przy każdym uczestniku znajdują się jego imię i nazwisko, numer indeksu oraz rozwijane listy z wyborem oceny za zaliczenie ćwiczeń, a także ocena za egzamin końcowy.

Seminarium magisterskie 3

Uczestnicy

Indeks	Imię	Nazwisko	Ocena z ćwiczeń	Ocena z egzaminu
186410	Szymon	Domurat	2	2
194925	Mateusz	Kwiatkowski	2	2

Rysunek 2.8: Fragment listy studentów zapisanych na przedmiot

Źródło: Własne

2.4. Funkcjonalność dla studenta

Studenci są grupą, która ma najmniej do powiedzenia w działaniu aplikacji, ale to studenci najbardziej oczekują szybkiego umieszczenia danych w indeksie.

Jedyną funkcjonalnością, z której mogą korzystać studenci, jest przegląd własnych osiągnięć w nauce. Po zalogowaniu do systemu student otrzymuje listę przedmiotów, w których uczestniczy. Po wejściu w interesujący użytkownika przedmiot, pojawiają się informacje o prowadzącym zajęcia, oraz oceny z ćwiczeń i egzaminu.

Przedmiot	Prowadzący	Ocena z ćwiczeń	Ocena z egzaminu
Seminarium magisterskie 3	dr Włodzimierz Bzyl	4.5	5

Rysunek 2.9: Oceny studenta z wybranego przedmiotu

Źródło: Własne

2.5. Funkcjonalność dla pracownika dziekanatu

Kiedy już wykładowca wystawi zaliczenia w elektronicznym indeksie, nadchodzi czas, gdy zarówno wykładowca jak i student, muszą rozliczyć się z tych danych w dziekanacie. Zadaniem jego pracownika jest utrzymanie porządku oraz spójności danych pomiędzy tymi, które znajdują się w elektronicznej platformie oraz jej papierowym odpowiedniku. Po zalogowaniu do systemu, użytkownikowi pojawi się lista studentów oraz lista wykładowców. Gdy pracownik dziekanatu wybierze z listy interesującego go studenta, na ekranie pojawi się lista przedmiotów danego ucznia wraz z jego osiągnięciami w nauce, co pozwoli na szybką weryfikację czy dane, które dostarczył student zgadzają się z tymi w aplikacji.

Przedmiot	Prowadzący	Ocena z ćwiczeń	Ocena z egzaminu
Seminarium magisterskie 1	dr Włodzimierz Bzyl	5	5
Seminarium magisterskie 2	dr Włodzimierz Bzyl	3.5	4
Seminarium magisterskie 3	dr Włodzimierz Bzyl	4.5	5

Rysunek 2.10: Oceny studenta

Źródło: Własne

Jeśli wybrany zostanie z listy wykładowca, pojawi się na ekranie lista przedmiotów, które dany pracownik prowadzi. Z nowo wyświetlonej listy pracownik dziekanatu

może wybrać interesujący go przedmiot, po czym na ekranie pojawią się wszyscy studenci, zapisani na wybrany przedmiot razem ze swoimi osiągnięciami, co pozwoli na szybką weryfikację czy dane dostarczone przez wykładowcę zgadzają się z tymi w elektronicznej platformie.

Przedmiot		Prowadzący	
Seminarium magisterskie 3		dr Włodzimierz Bzyl	
Seminarium magisterskie 3			
Uczestnicy			
Indeks	Imię	Nazwisko	
165334	5	4	
179796	4.5	4.5	

Rysunek 2.11: Fragment listy uczestników zajęć

Źródło: Własne

2.6. Importowanie i eksportowanie danych

Importowanie oraz eksportowanie danych są ważną funkcją w *elektronicznym indeksie*, ponieważ pozwala to użytkownikom na przenoszenie danych pomiędzy systemami, a także umożliwia prace na danych na maszynach lokalnych użytkowników systemu. Funkcje te również wymagają walidacji aby nie dopuścić do sytuacji gdzie użytkownik zaimportuje błędne dane bądź niepasujące do schematu trzymanych w kolekcji informacji. Korzystając z *MongoDB* w wielu przypadkach niewykonalne jest zaimplementowanie systemu, który w momencie wystąpienia błędu podczas importowania danych przywróci bazę do stanu z przed importu, ponieważ *MongoDB* nie wspiera aktualizacji wielu dokumentów jednocześnie. W systemach gdzie baza nie jest rozbudowana i znajduje się w jednym dokumencie można skorzystać z dwu fazowego systemu aktualizacji co pozwoli osiągnąć efekt jak przy transakcjach.[11]

ROZDZIAŁ 3

Aplikacja Elektroniczny indeks w Meteor


3.1. Opis tworzenia aplikacji

Tworzenie aplikacji w frameworku *Meteor* zaczyna się od utworzenia nowego projektu poleceniem w konsoli:

Listing 3.1: Tworzenie projektu

```
meteor create nazwa_projektu
```

Po wykonaniu tej komendy zostanie utworzony folder z projektem, w którym znajdują się trzy pliki.

A diagram showing three files: `index.css`, `index.html`, and `index.js`. Each file name is enclosed in a black rectangular box with white text, and the boxes are arranged horizontally.

Rysunek 3.1: Nowy projekt

Źródło: Własne

Aby skorzystać aplikacji musimy uruchomić lokalny serwer, który pozwoli zobaczyć przetworzony javascript w przeglądarce oraz uruchomić bazę na lokalnym komputerze. W konsoli zmieniamy lokalizację na folder z projektem, a następnie wprowadzamy następującą komendę:

Listing 3.2: Uruchomienie aplikacji

```
meteor run
```

Jeżeli na ekranie pojawi się

```
MacBook-Pro-Mateusz:indeks flover$ meteor run
[[[[[[ ~/szkola/github/praca_magisterska/indeks ]]]]]

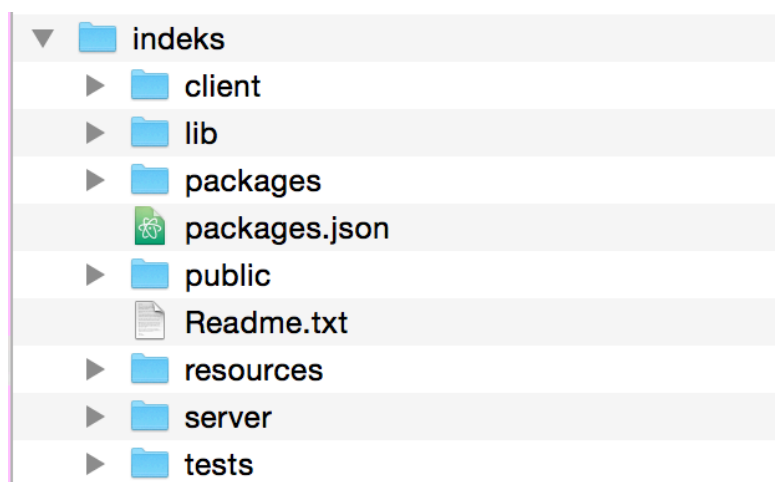
=> Started proxy.
=> Started your app.

=> App running at: http://localhost:3000/
```

Rysunek 3.2: Start aplikacji bez błędów

Źródło: Własne

oznacza to, że aplikacja uruchomiła się bez błędów i od tego momentu można z niej korzystać pod podanym adresem. Na początku warto podzielić projekt na część serwerową, kliencką oraz dostępną zarówno dla części serwerowej oraz klienckiej.



Rysunek 3.3: Podział projektu

Źródło: Własne

Warto również usunąć pakiety *autopublish* oraz *insecure*, dzięki czemu zwiększone zostanie bezpieczeństwo aplikacji. Usunięcie pakietu *insecure* blokuje możliwość użytkownikom do zarządzania bazą danych po stronie klienta, natomiast

wyrzucenie *autopublish* spowoduje, że trzeba samemu zadbać o wysyłanie danych z serwera do klienta.[8]

Listing 3.3: Usuwanie pakietów

```
meteor remove autopublish  
meteor remove insecure
```

W części klienckiej trzymane są widoki aplikacji, funkcje pomocnicze oraz router, a po stronie serwera plik *publish.js*, który definiuje, jakie dane zostaną wysłane do użytkowników. Po podzieleniu projektu trzeba utworzyć kolekcje, w których będziemy trzymać informacje o przedmiotach oraz ocenach studentów zdobytych na zajęciach. Do tego celu utworzymy kolekcję *subjects*, w której będzie trzymana nazwa przedmiotu, tytuł naukowy i nazwisko prowadzącego zajęcia, semestr, na którym przedmiot się odbywa oraz lista zapisanych studentów. Kolekcja *grades* będzie łączyła informacje z kolekcji z użytkownikami oraz przedmiotami. Znajdą się tu login oraz id użytkownika oraz id przedmiotu, nazwa przedmiotu, jego prowadzący i oceny z ćwiczeń, i egzaminu końcowego.

```
Subjects = new Mongo.Collection('subjects');  
Grades = new Mongo.Collection('grades');
```

Listing 1: Utworzenie kolekcji

Kiedy dysponujemy już kolekcjami, trzeba utworzyć widoki. Zaczynamy od widoku strony głównej, na której w zależności od grupy danego użytkownika, zawierać będzie imiona i nazwiska studentów, poszczególne przedmioty oraz ich wykładowcy.

Analogicznie działa wyświetlanie przedmiotów dla wykładowcy czy administratora. W tym momencie, nawet jeśli w bazie będą wprowadzone przedmioty, strona nadal będzie pusta. Do przesyłu danych wykorzystamy *iron-router*.

```

<template name="subjectList">
  <div class="container">
    <div class="subject-list">
      <div class="list-group">
        {{#if isInRole 'admin, wykładowca, student'}}
        <a class="list-group-item active">
          Przedmioty
        </a>
        {{#if isInRole 'student'}}
        {{#each mySubjects}}
        <a href="/subjects/{{id}}" class="list-group-item">{{subject}}</a>
        {{/each}}
        {{/if}}
      </div>
    </div>
  </div>

```

Listing 2: Template wyświetlający wszystkie przedmioty, na które uczęszcza student

```

waitOn: function () {
  return [ Meteor.subscribe('theSubjects') ];
},
onBeforeAction: function () {
  if(!Meteor.user()){
    this.layout('appLayout');
    this.render('login');
  }
  else {
    this.next();
  }
},
action: function () {
  this.layout('appLayout');
  this.render('subjectList', {
    'data': {
      'User': Meteor.user(),
      'mySubjects': Subjects.find(),
      'teacherSubjects': Subjects.find({'leading': Meteor.user().profile.title + " " +
        Meteor.user().profile.firstName + " " + Meteor.user().profile.lastName},
        {sort: {'subject': 1}}),
      'myStudents': Meteor.users.find({'roles': 'student'}, {sort: {'profile.lastName':
        1, 'profile.firstName': 1, 'username': 1}}),
      'myLeaders': Meteor.users.find({'roles': 'wykładowca'}, {sort:
        {'profile.lastName': 1, 'profile.firstName': 1, 'username': 1}}),
      'myGrades': Grades.find()
    }
  });
}

```

Listing 3: Powyższy fragment renderuje widok z przedmiotami oraz użytkownikami

W drugiej linii kodu widzimy `return`, który zwraca zasubskrybowane dane z `theSubjects`. Żeby te dane można było odebrać, serwer musi je opublikować.

```
Meteor.publish('theSubjects', function () {
  if(this.userId){
    if (Roles.userIsInRole(this.userId, 'admin')) {
      return [Meteor.users.find({}), Subjects.find({})];
    } else if(Roles.userIsInRole(this.userId, 'dziekanat')){
      return [Meteor.users.find({}), Subjects.find({})];
    } else if(Roles.userIsInRole(this.userId, 'student')){
      var user = Meteor.users.findOne({'_id': this.userId})._id;
      return Subjects.find({"students": user});
    } else if(Roles.userIsInRole(this.userId, 'wykładowca')){
      return [Subjects.find({}), Meteor.users.find({})];
    }
  }
  else {
    this.ready();
  }
});
```

Listing 4: udostępnianie danych poszczególnym grupom użytkowników

Dzięki pakietowi *meteor-roles* jesteśmy w stanie dokładniej określić, jakie dane chcemy wysyłać poszczególnym grupom użytkowników. Dzięki temu w szybki sposób możemy również zarządzać treściami, które pojawiają się na widokach dzięki blokom pomocniczym.

W aplikacji znajduje się szereg managerów, które przekazują operacje wykonywane przez użytkownika do metod, które zostaną wykonane na serwerze. Funkcja *assignSubject*, która zostaje wywołana w momencie, gdy użytkownik chce przypisać studenta do przedmiotu, pobiera id przedmiotu, jego nazwę, nazwę użytkownika, id użytkownika oraz prowadzącego zajęcia i przekazuje je do metody *addStudentToSubject*, która umieszcza te informacje w kolekcji *grades*, jednocześnie przypisując id studenta do listy osób zapisanych na przedmiot w kolekcji *subjects*. Przypisuje także id przedmiotu do listy przedmiotów, na które student uczęszcza, w kolekcji użytkowników. Funkcje *updateExerciseGrade* i *updateExamGrade* działają w taki sam sposób. Gdy wykładowca zatwierdza zmianę oceny, pobierają one z pola ocenę, jaką zatwierdził prowadzący, a następnie przekazują id studenta, id

przedmiotu oraz ocenę do metod o tych samych nazwach. Dzięki temu, że wysłane zostały także id przedmiotu i studenta, możliwa jest aktualizacja oceny konkretnego studenta dla konkretnego przedmiotu. Gdy administrator wysyła formularz *form*, aby dodać nowy przedmiot, z pól formularza zostają pobrane informacje o nazwie przedmiotu, prowadzącym przedmiot oraz informacja, na którym semestrze zajęcia się odbywają, a następnie są one przesłane do metody *addSubject*, która umieszcza przedmiot w kolekcji *subjects*. W trakcie tworzenia nowego użytkownika, zostaje mu przypisana jedna z czterech dostępnych ról. Jeśli tworzenie użytkownika zakończy się powodzeniem, zostaje wywołana metoda *assignRole*, która otrzymuje stworzonego użytkownika oraz rolę, do której będzie przypisany i ustawia danemu użytkownikowi wybraną rolę.

[9] [10] [11] [12] [13]

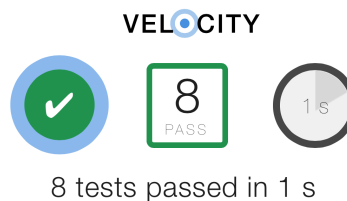
3.2. Opis testowania aplikacji

Od wersji *Meteor* 1.0, zostało oficjalnie udostępnione narzędzie do testowania aplikacji o nazwie *Velocity*. Instaluje się ono razem z jednym z czterech dostępnych frameworków do testowania. W tej pracy użyty został framework *Jasmine*. Aby dodać go do projektu wystarczy w konsoli wpisać

Listing 3.4: Instalacja Velocity, Jasmine i html reporter

```
meteor add sanjo:jasmine
meteor add velocity:html-reporter
```

Html-reporter jest to reaktywny plugin, który prezentuje wyniki testów.



Rysunek 3.4: Html-reporter

Źródło: Własne


```
describe('Set user a role', function () {
  it('Should assign user to role', function () {
    if (Meteor.users.find().count() === 0) {
      Accounts.createUser({
        username: 'username1',
        email: 'username1@test.pl',
        password: 'abcdef',
        profile: {
          name: 'student',
          firstName: 'firstName1',
          lastName: 'lastName1',
          subjects: []
        }
      });
      Accounts.createUser({
        username: 'username2',
        email: 'username2@test.pl',
        password: 'abcdef',
        profile: {
          name: 'student',
          firstName: 'firstName2',
          lastName: 'lastName2',
          subjects: []
        }
      });
    }

    user = Meteor.users.find({'username': 'username1'}).fetch();

    Meteor.call('assignRole', user[0], 'student');
    expect(Meteor.users.findOne({'username': 'username1'}).roles[0]).toBe('student');

    Meteor.users.remove({'username': 'username1'});
    Meteor.users.remove({'username': 'username2'});
  });
});
```

Listing 5: Test przypisywania roli użytkownikowi

describe mówi nam, jaka funkcjonalność będzie testowana, *it* opisuje, co dana funkcjonalność powinna robić. Następnie do replikowanej bazy na potrzeby testu dodajemy użytkowników. Gdy użytkowników w bazie, wskazujemy konkretnego użytkownika i pobieramy wszystkie informacje o nim. Wywołujemy metodę, która przypisuje użytkownikowi rolę, a następnie sprawdzamy czy naszemu użyt-

kownikowi została przypisana taka, której oczekiwaliśmy. Po wszystkim czyścimy zreplikowaną bazę. [14]

ROZDZIAŁ 4

Pakiet walidujący operacje elektronicznego indeksu

Pakiet ma na celu nie dopuścić do sytuacji, w której użytkownik wprowadzi do bazy danych błędne lub niekompletne treści.

4.1. Funkcjonalność pakietu

Użytkownicy, którzy mają dostęp do wprowadzania danych do aplikacji, są szczególnie narażeni na umieszczenie nieprawidłowych informacji w systemie. Dzięki pakietowi dane w aplikacji będą spójne, a użytkownik je modyfikujący będzie miał pewność, że nawet przez pomyłkę nie wprowadzi do systemu błędnych informacji.

Aplikacja stworzona na potrzeby pracy posiada tylko standardową, niezbyt rozbudowaną funkcjonalność, przez co procesowi rozszerzonej walidacji musiały zostać poddane takie operacje, jak: wystawianie zgodnie ze skalą ocen, wystawianie ocen z egzaminu, biorąc pod uwagę czy student spełnił wymagania, aby przystąpić do egzaminu, a także poprawne dodawanie przedmiotów. Do walidacji pól skorzystano z gotowego rozwiązania, pakietu *Mesosphere*

4.2. Opis tworzenia pakietu

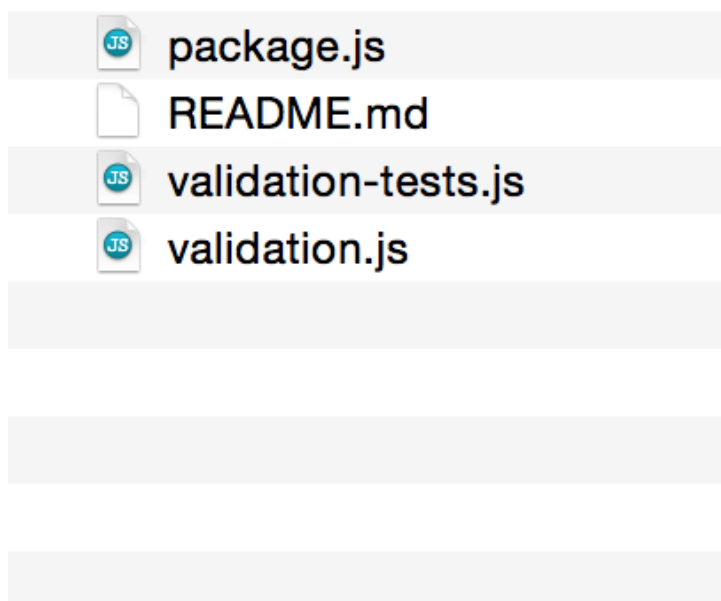
Aby utworzyć pakiet, który można będzie wykorzystać w *Meteor*, trzeba w aplikacji utworzyć folder *packages*, a następnie w jej głównym folderze należy użyć polecenia w terminalu bardzo podobnego do tego, którym tworzy się nową aplikację.

```
meteor create --package emflower:validation
```

Listing 4.1: Tworzenie nowego pakietu

Interpretacja: Przełącznik *package* sprawia, że utworzony zostanie pakiet. Jako nazwę przed znakiem dwukropka podajemy swój login dewelopera Meteor, a po dwukropku nazwę pakietu.

Jak widać różnicą jest dodanie przełącznika. Jako nazwę przed znakiem dwukropka trzeba podać swój login dewelopera Meteor, a po dwukropku nazwę pakietu. Po wykonaniu tego polecenia w folderze *packages* zostaną utworzone pliki.



Rysunek 4.1: Nowy pakiet

Interpretacja: *README.md* zawiera opis pakietu; w *validation.js* mieści się właściwy kod tworzonego pakietu; *validation-tests.js* zawiera testy pakietu; do testowania pakietów służy framework *Tinytest*; *package.js* zawiera opis pakietu oraz jego zależności i gdzie wykonywać mają się poszczególne funkcje.

Źródło: Własne

Samo utworzenie nie wystarczy, żeby z niego korzystać. Trzeba go podpiąć do projektu. Podobnie, jak robi się to z każdym innym pakietem, nawet jeśli nie jest on udostępniony na *atmospherejs.com*.^{[15][10][8]}

Tworzenie pakietu wygląda identycznie, jak tworzenie każdej innej funkcji w *javascript*.

```
checkBeforeAction = function(studentId, exercise, exam){
  if(exercise > 2 && exercise <=5){
    Meteor.call('updateExamGrade', studentId, Router.current().params.subjectId, exam);
  } else {
    bootbox.alert("Twoja ocena z ćwiczeń to: " + exercise + ". Nie możesz uczestniczyć w  
↪ egzaminie.");
  }
}
```

Listing 6: Funkcja sprawdzająca czy student może otrzymać pozytywną ocenę z egzaminu

Interpretacja: funkcja przyjmuje trzy argumenty: id studenta, jego ocenę z ćwiczeń oraz ocenę z egzaminu. Jeżeli ocena z ćwiczeń jest negatywna system nie pozwoli na wystawienie pozytywnej oceny studentowi, o czym poinformuje wyskakujące okno z informacją o błędzie.

Podajemy nazwę funkcji, przyjmowane argumenty, a następnie podajemy jej ciało. Aby móc skorzystać z nowo stworzonej funkcji, musimy wyeksportować ją z naszego pakietu.

```
Package.onUse(function(api) {  
  api.versionsFrom('1.1.0.2');  
  api.add_files("validation-client.js", "client");  
  api.export('checkBeforeAction', 'client');  
  api.export('checkIfChooseTeacher', 'client');  
});
```

Listing 7: Eksport funkcji i użycie plików, w których znajdują się eksportowane funkcje

Interpretacja: *versionsFrom* mówi, od jakiej wersji *Meteor* można używać danego pakietu; *add_files* dodaje pliki, w których znajduje się kod pakietu oraz mówi czy ma być przesłany do serwera, czy klienta; *export* - jak sama nazwa wskazuje, eksportuje funkcje z dodanego wcześniej pliku i określa, czy mają zostać wykonane po stronie serwera czy klienta.

Aby użyć pakietu w aplikacji musimy, wywołać ją po stronie klienta.

```
'click .updateExamGrade': function (event, template) {  
  var examGrade = template.find('#subjectExamGrade_'+this._id).value;  
  var exerciseGrade = Grades.findOne({'subjectId': Router.current().params.subjectId,  
    ↪ 'studentId': this._id}).exerciseGrade;  
  checkBeforeAction(this._id, exerciseGrade, examGrade);  
}
```

Listing 8: Funkcja wystawiająca studentowi oceny

Interpretacja: Funkcja pobiera wartość pola z oceną z egzaminu oraz pobiera z bazy ocenę studenta z ćwiczeń i przekazuje je do funkcji walidującej.

Podczas wystawiania oceny do funkcji przekazane zostają id użytkownika, któremu wystawiana jest ocena, ocena jaką wykła

4.3. Przetestowanie pakietu

Do testowania pakietów *Meteor* służy framework *TinyTest*. Jest to oficjalne i jedyne narzędzie służące do przeprowadzania testów pakietów. Wadą tego frameworka jest fakt, że nie został on w żaden sposób opisany, przez co korzystanie z niego dla osoby niedoświadczonej może być nie do końca zrozumiałe.

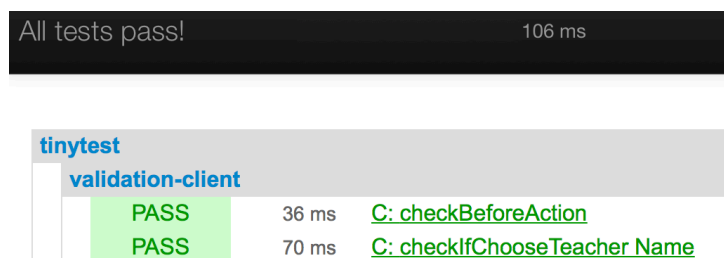
Aby przetestować pakiety, należy w głównym folderze aplikacji wpisać w konsoli polecenie:

```
meteor test-packages ./packages/validation/
```

Listing 4.2: Testowanie pakietu

Interpretacja: *test-packages* mówi o tym, że uruchamiamy testy pakietu; *./packages/validation/* jest to ścieżka do testowanego pakietu

Po wykonaniu polecenia, jeżeli stworzone testy nie zawierają błędów, uruchomi się serwer, na którym zostaną one przeprowadzone, a wyniki zostaną przedstawione na czytelnym interfejsie użytkownika. Jeśli test zostanie wykonany prawidłowo, na ekranie wyświetli się:

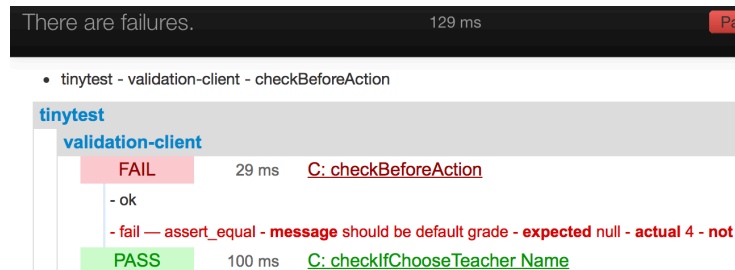


Rysunek 4.2: Wynik testów bez błędów

Interpretacja: Testy po stronie klienta zakończyły się sukcesem.

Źródło: Własne

Jeżeli w teście będą błędy i zostanie otrzymany wynik odbiegający od oczekiwań, na ekranie pojawi się:



Rysunek 4.3: Wynik testów zakończone niepowodzeniem

Interpretacja: Test *checkBeforeAction* zakończył się niepowodzeniem. Oczekiwano wartości null, natomiast funkcja zwróciła 4.

Źródło: Własne

Aby wykonać test, musimy w pliku *package.js* dodać:

```
Package.onTest(function(api) {
  api.use('tinytest');
  api.use(['mizzao:bootboxjs']);
  api.use('emflover:validation');
  api.addFiles('validation-tests.js');
});
```

Listing 9: Zależności potrzebne do testowania aplikacji

Interpretacja: *api.use* pobiera pakiety, które będą potrzebne do poprawnego przeprowadzenia testu; *api.addFiles* dodaje pliki potrzebne do wykonania testu.

Pomimo tego, że w projekcie mamy dodane potrzebne pakiety, podczas przeprowadzania testów *tinytest* nie jest w stanie z nich skorzystać, jeśli sami nie wskażemy, że odpowiednie narzędzia są dostępne.

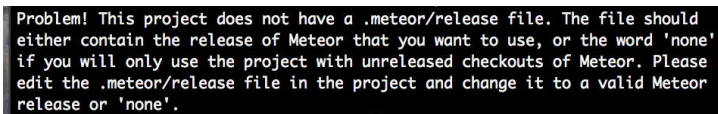

```
Tinytest.add('validation-client - checkBeforeAction', function (test) {  
  var exerciseGrade = 2;  
  var examGrade = 3;  
  var Person = function(id, username){  
    this.id = id;  
    this.username = username;  
  }  
  var Subject = function(id, subjectName, leading){  
    this.id = id;  
    this.subjectName = subjectName;  
    this.leading = leading;  
  }  
  var Grade = function(id, subjectId, studentId, exercise, exam){  
    this.id = id;  
    this.subjectId = subjectId;  
    this.studentId = studentId;  
    this.exercise = exercise;  
    this.exam = exam;  
  }  
  var student = new Person('1', 'username1');  
  var sub = new Subject('1', 'Subject 1', 'Leading 1');  
  var gradez = new Grade('1', '1', '1', exerciseGrade, null);  
  checkBeforeAction('1', exerciseGrade, examGrade);  
  test.equal(gradez.exercise, 2, 'Should be grade 2');  
  test.equal(gradez.exam, null, 'should be default grade');  
}
```

Listing 10: Test dodawania oceny z egzaminu studentowi gdy ma on negatywną ocenę z ćwiczeń

Interpretacja: Pierwszym argumentem *Tinytest.add* jest nazwa naszego testu, a drugim funkcja, która go wykona. Na potrzeby testu definiujemy i tworzymy nowego użytkownika, przedmiot oraz ocenę, a następnie wywołujemy testowaną funkcję, a po jej wykonaniu sprawdzamy czy oceny z ćwiczeń i egzaminu się zgadzają.^[16]

4.4. Udostępnienie pakietu

Kiedy już pakiet, który tworzymy został ukończony, a także przetestowany, można go opublikować na <https://atmospherejs.com>.^[17] Czasami może dojść do sytuacji gdzie postępując zgodnie z instrukcją nie będziemy w stanie opublikować naszego pakietu, a na ekranie pojawi się błąd

A screenshot of a terminal window showing an error message. The text is white on a black background. The message reads: "Problem! This project does not have a .meteor/release file. The file should either contain the release of Meteor that you want to use, or the word 'none' if you will only use the project with unreleased checkouts of Meteor. Please edit the .meteor/release file in the project and change it to a valid Meteor release or 'none'."

```
Problem! This project does not have a .meteor/release file. The file should
either contain the release of Meteor that you want to use, or the word 'none'
if you will only use the project with unreleased checkouts of Meteor. Please
edit the .meteor/release file in the project and change it to a valid Meteor
release or 'none'.
```

Rysunek 4.4: Problem z publikowaniem pakietu

Źródło: Własne

W takiej sytuacji należy w aplikacji, w której utworzyliśmy pakiet, wejść do ukrytego folderu `.meteor/local`, a następnie usunąć folder o nazwie `isopacks`. Jeżeli to nie rozwiąże problemu należy przenieść pakiet poza aplikację i w konsoli wpisać odpowiednią komendę.

Zakończenie

Celem pracy magisterskiej było stworzenie aplikacji - elektroniczny indeks oraz pakietu, który walidował operacje wykonywane podczas użytkowania oprogramowania. Cele pracy zostały osiągnięte, dając zadowalające rezultaty.

Technologie wykorzystane w pracy okazały się dobrym wyborem, ponieważ oferowały szeroki wachlarz narzędzi, które usprawniały pracę nad projektem, pozwalały w niedługim czasie stworzyć aplikację, która posiadała wymaganą podstawową funkcjonalność, dzięki czemu można było ją przetestować i walidować.

Podczas tworzenia pracy pojawiły się problemy ze strony dostarczonych technologii. Dla sprawnego przeglądu informacji przez użytkowników, autor chciał sortować dane wyświetlane użytkownikowi leksykograficznie, na co pozwala *MongoDB*, ale technologia ta ze względu na wykorzystywaną funkcję *memcmp* nie obsługuje poprawnie formatu UTF-8 w różnych ustawieniach lokalnych. Twórcy bazy danych planują wprowadzić zmiany w systemie, ale przybliżona data nie jest znana, ponieważ wymaga to wielu poważnych modyfikacji programu.

Niewątpliwą zaletą systemu jest możliwość jego rozdudowy bez konieczności ingerencji w kod, który został napisany wcześniej, czego dowodem jest dodanie funkcji pobierania danych z aplikacji oraz ich import z powrotem do systemu z poziomu aplikacji, już po zakończeniu jej tworzenia. Ta dodatkowa funkcjonalność pozwala użytkownikom zarządzać danymi w swoich własnych aplikacjach typu — elektroniczny indeks, utworzonym przez nich na własne potrzeby.

Bibliografia

- [1] Włodzimierz Gajda. *Walidacja danych*. gajdaw, 2008.
- [2] Wikipedia. *Walidacja (technika)*. Wikipedia, 2014.
- [3] Piotr Furtak. *Jak zrozumieć różnicę między weryfikacją a walidacją systemu, by nie mieć problemu podczas odpowiadania na egzaminie ISTQB?* ALT-KOM Akademia, 2014.
- [4] Wikipedia. *Weryfikacja i walidacja oprogramowania*. Wikipedia, 2014.
- [5] Wikipedia. *Aspekty ekonomiczne procesu walidacji*. DPK Consulting, 2008.
- [6] Wikipedia. *Verification and validation*. Wikipedia, 2014.
- [7] W. Szkolnikowski A. Łobzowski. *Co firma LAB-EL ma do powiedzenia w tematyce GAMP 4 (cz. 2)*. LAB-EL, 2009.
- [8] Tom Coleman and Sacha Greif. *Discover Meteor: Building Real-Time JavaScript Web Apps*. First edition edition, 2013.
- [9] Stephen Walther. *An Introduction to Meteor*. stephenwalther, 2013.
- [10] MeteorJS. *Meteor Documentation*, 2014.
- [11] MongoDB. *The MongoDB Manual*, 2014.
- [12] Kristina Chodorow. *Scaling MongoDB*. First edition edition, 2011.
- [13] Arunoda Susiripala. *Let's Scale Meteor*. 2013.
- [14] Sam Hatoum, Jonas Aschenbrenner, Mike Risse. *Official tool for testing Meteor apps*. Meteor, 2015.
- [15] Matthew Platts. *Meteor JS packages tutorial*. Web Tempest, 2015.
- [16] Eventedmind. *Testing Packages with Tinytest*, 2013.
- [17] Percolate Studio. *Learn how to add your packages to the package index*, 2015.

Spis rysunków

2.1.	Menu dodawania	12
2.2.	Fragment listy studentów do zapisu na zajęcia	13
2.3.	Formularz dodawania przedmiotu	14
2.4.	Formularz dodawania studenta	14
2.5.	Formularz dodawania wykładowcy	15
2.6.	Formularz dodawania pracownika dziekanatu	15
2.7.	Lista przedmiotów wykładowcy	16
2.8.	Fragment listy studentów zapisanych na przedmiot	16
2.9.	Oceny studenta z wybranego przedmiotu	17
2.10.	Oceny studenta	17
2.11.	Fragment listy uczestników zajęć	18
3.1.	Nowy projekt	19
3.2.	Start aplikacji bez błędów	20
3.3.	Podział projektu	20
3.4.	Html-reporter	24
4.1.	Nowy pakiet	

Interpretacja: *README.md* zawiera opis pakietu; w *validation.js* mieści się właściwy kod tworzonego pakietu; *validation-tests.js* zawiera testy pakietu; do testowania pakietów służy framework *Tinytest*; *package.js* zawiera opis pakietu oraz jego zależności i gdzie wykonywać mają się poszczególne funkcje. 28

4.2.	Wynik testów bez błędów	
------	-------------------------	--

Interpretacja: Testy po stronie klienta zakończyły się sukcesem. 31

4.3. Wynik testów zakończone niepowodzeniem

Interpretacja: Test *checkBeforeAction* zakończył się niepowodzeniem. Oczekiwano wartości null, natomiast funkcja zwróciła 4. 32

4.4. Problem z publikowaniem pakietu 34

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