

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Elektronika i telekomunikacja (EIT)
SPECJALNOŚĆ: Zastosowania inżynierii komputerowej
w technice (EZI)

**PRACA DYPLOMOWA
MAGISTERSKA**

Projektowanie systemów zdalnego nauczania z
wykorzystaniem usług sieciowych w środowisku
MS .NET

Designing distance learning systems using Web
Services in MS .NET environment

AUTOR:
Michał Franc

PROWADZĄCY PRACĘ:
dr inż. Robert Wójcik, W4/I-6

OCENA PRACY:

*Pracę tę dedykuję kochającej Ma-
mie, która nauczyła mnie jak być
dobrym człowiekiem oraz wspa-
niałemu Tacie, który pokazał mi
świat techniki.*

Spis treści

Spis rysunków	3
Spis tabel	5
Wstęp	6
1 Cel i zakres pracy	7
2 Kształcenie na odległość	8
2.1 Definicja kształcenia na odległość	8
2.2 Korzyści płynące z używania systemów kształcenia na odległość	9
2.3 Ograniczenia kształcenia na odległość	9
2.4 Klasyfikacja systemów kształcenia na odległość	10
2.4.1 Systemy do zarządzania kursami(CMS)	10
2.4.2 Systemy wspomagające proces nauczania (LMS)	11
3 Wybrane technologie realizacji systemów webowych	12
3.1 Formaty danych JSON oraz XML	12
3.2 Protokoły SOAP oraz REST	15
3.3 Framework Asp.Net MVC	19
3.4 Technologia WCF - Windows communication foundation	21
3.5 Mapowanie obiektowo-relacyjne - NHibernate	22
4 Projekt systemu zdalnego nauczania	24
4.1 Wymagania projektowe	24
4.1.1 Wymagania funkcjonalne	24
4.1.2 Wymagania нефункционалне	27
4.2 Projekt bazy danych	29
4.2.1 Opis encji	29
4.2.2 Uproszczony model conceptualny (CDM) - struktura tabel i relacje	31
4.2.3 Model fizyczny bazy danych PDM	33
4.3 Architektura Systemu	35
4.3.1 Diagramy sekwencji wybranych funkcjonalności	36
4.3.2 Mechanizmy zabezpieczeń	37
5 Implementacja systemu zdalnego nauczania	39
5.1 Zewnętrzny hosting	39
5.2 Realizacja bazy danych	39
5.3 Realizacja aplikacji	40
5.3.1 Mechanizmy dostępu do bazy danych	40

5.3.2	Mechanizmy przetwarzania danych	43
5.3.3	Realizacja protokołu komunikacji	45
5.4	Wykorzystane narzędzia	48
5.4.1	Mechanizm logowania zdarzeń - NLog	48
5.4.2	Testy jednostkowe - NUnit	50
5.4.3	Mapowanie obiektowo relacyjne - NHibernate	54
5.5	Interfejs użytkownika	56
6	Testowanie i ocena efektywności	59
6.1	Wybrane testy funkcjonalne	59
6.1.1	Test procedury tworzenia nowego materiału nauczania	59
6.1.2	Test procedury tworzenia nowego testu	63
6.1.3	Test procedury rozwiązywania testu	64
6.1.4	Test procedury tworzenia nowego kursu	64
6.2	Testy zabezpieczeń przed atakiem poprzez wstrzykiwanie skryptów	65
6.3	Testy oraz optymalizacja zapytań generowanych przez framework NHibernate	67
6.4	Ocena wydajności aplikacji oraz proces optymalizacji mechanizmów przetwarzania danych	71
6.5	Ocena wpływu konfiguracji rozłożenia usług sieciowych na czas odpowiedzi	77
6.6	Ocena wydajności mechanizmu przekierowania na zapasowy serwer usług sieciowych	78
6.7	Wnioski z testów i badań	80
7	Podsumowanie	82
	Bibilografia	84

Spis rysunków

Rysunek 2.1	Podstawowe funkcje systemu klasy CMS	10
Rysunek 2.2	Podstawowe funkcje systemu klasy LMS	11
Rysunek 3.1	Uproszczony Schemat formatu JSON	12
Rysunek 3.2	Uproszczony schemat formatu XML	13
Rysunek 3.3	Porównanie standardów (JSON , XML)	14
Rysunek 3.4	Przykładowa wiadomość SOAP	15
Rysunek 3.5	Przykładowa wiadomość zwrotna	16
Rysunek 3.6	Porównanie protokołów Rest / Soap [15]	17
Rysunek 3.7	Przykładowa wiadomość zwrotna protokołu REST	18
Rysunek 3.8	Uproszczona koncepcja wzorca MVC	19
Rysunek 3.9	Framework WCF	21
Rysunek 3.10	Framework WCF	21
Rysunek 3.11	ORM obsługujący bazę danych	22
Rysunek 3.12	Rysunek przedstawiający proces mapowania.	23
Rysunek 4.1	Diagram CDM cz. 1	31
Rysunek 4.2	Diagram CDM cz. 2	32
Rysunek 4.3	Diagram PDM cz. 1	33
Rysunek 4.4	Diagram PDM cz. 2	34
Rysunek 4.5	Schemat proponowanego rozwiązania	35
Rysunek 4.6	Diagram sekwencji generowania strony WWW	36
Rysunek 4.7	Diagram sekwencji logowania do aplikacji	36
Rysunek 4.8	Diagram sekwencji procesu nauczania	37
Rysunek 4.9	Rozdzielenie mechanizmu logowania na dwie bazy danych	38
Rysunek 5.1	Diagram klasy DataAccess	40
Rysunek 5.2	Mechanizm dostępu do bazy przy użyciu klasy DataAccess	41
Rysunek 5.3	Klasa Repository oraz QueryObject	41
Rysunek 5.4	Klasa QueryCourseByName opakowująca zapytanie języka HQL	42
Rysunek 5.5	Skomplikowane zapytanie HQL	42
Rysunek 5.6	Przykład przekształcenia do formatu XML	43
Rysunek 5.7	Przykład przekształcenia do formatu JSON	43
Rysunek 5.8	Graficzne przedstawienie procesu przetwarzania danych	44
Rysunek 5.9	Przekształcanie obiektów typu ViewModel	45
Rysunek 5.10	Ustawienia usługi po stronie serwera	45
Rysunek 5.11	Ustawienia bindingu po stronie klienta	46
Rysunek 5.12	Ustawienia endpointu po stronie klienta	46
Rysunek 5.13	Ustawienia endpointu po stronie klienta korzystającego z API	47
Rysunek 5.14	Przykładowy log z programu Sentinel [8]	49

Rysunek 5.15	Przykładowy alert	49
Rysunek 5.16	Prosty test jednostkowy bazy danych	50
Rysunek 5.17	Test jednostkowy z bazą danych w pamięci	51
Rysunek 5.18	Test jednostkowy kontrolera bez zewnętrznych zasobów	52
Rysunek 5.19	Test kontrolera z zewnętrznymi zasobami	52
Rysunek 5.20	Test kontrolera przy wykorzystaniu mocka	53
Rysunek 5.21	Przykładowy kod testu z mockiem	53
Rysunek 5.22	Proces mapowania relacyjno-objektowego	54
Rysunek 5.23	Przykładowe mapowanie klasy Kurs	55
Rysunek 5.24	Widok logowania	56
Rysunek 5.25	Widok strony głównej	56
Rysunek 5.26	Widok kursu	57
Rysunek 5.27	Widok materiału nauczania	57
Rysunek 5.28	Widok testu	58
Rysunek 5.29	Widok edycji testu	58
Rysunek 6.1	Panel edycji kursu	59
Rysunek 6.2	Lista materiałów nauczania	60
Rysunek 6.3	Baza danych przed dodaniem materiału	60
Rysunek 6.4	Baza danych po dodaniu materiału	61
Rysunek 6.5	Nowy element na liście	61
Rysunek 6.6	Panel edycji materiału nauczania	61
Rysunek 6.7	Test formatowania tekstu	62
Rysunek 6.8	Alert informujący o poprawnym zapisie danych	62
Rysunek 6.9	Baza danych po dodaniu opisu	62
Rysunek 6.10	Alert informujący o poprawnym stworzeniu nowej sekcji	63
Rysunek 6.11	Tabela zawierająca rekordy sekcji przed dodaniem	63
Rysunek 6.12	Tabela zawierająca rekordy sekcji po dodaniu nowej sekcji	63
Rysunek 6.13	Przykładowe wstrzyknięcie skryptu do edytora tekstowego	65
Rysunek 6.14	Przykład testowego skryptu	66
Rysunek 6.15	Udany atak typu script injection	66
Rysunek 6.16	Główny widok aplikacji NHProf.	67
Rysunek 6.17	Przykładowy widok prezentujący wynik profilowania.	68
Rysunek 6.18	Zawartość pliku script.txt	71
Rysunek 6.19	Zawartość pliku distribution.txt	71
Rysunek 6.20	Zawartość pliku config.txt	72
Rysunek 6.21	WCAT ekran kontrolera	72
Rysunek 6.22	WCAT ekran klienta	73
Rysunek 6.23	Statystyka działania aplikacji otrzymana w programie Ants Performance Profiler	73
Rysunek 6.24	Wynik programu profilującego - po poprawce	74
Rysunek 6.25	Wynik programu profilującego - aplikacja udostępniająca usługi sieciowe	75
Rysunek 6.26	Uproszczony schemat lokalnej konfiguracji	77
Rysunek 6.27	Uproszczony schemat zdalnej konfiguracji	77
Rysunek 6.28	Uproszczony schemat mechanizmu zmiany serwera	78

Spis tabel

Tabela 6.1	Testy funkcjonalne operacji tworzenia materiałów nauczania	63
Tabela 6.2	Testy funkcjonalne operacji tworzenie testu	64
Tabela 6.3	Testy funkcjonalne operacji rozwiązywanie testu	64
Tabela 6.4	Testy funkcjonalne operacji tworzenie kursu	64
Tabela 6.5	Parametry generowanych zapytań - wstępna próba	68
Tabela 6.6	Parametry generowanych zapytań - po wprowadzeniu poprawek . .	69
Tabela 6.7	Parametry generowanych zapisów - wstępna próba	70
Tabela 6.8	Parametry generowanych zapisów - po wprowadzeniu poprawek . .	70
Tabela 6.9	Test wstępny - średni czas odpowiedzi	73
Tabela 6.10	Średnie czasy odpowiedzi - po wprowadzeniu modyfikacji	74
Tabela 6.11	Operacje na bazie danych	75
Tabela 6.12	Średnie czasy odpowiedzi	75
Tabela 6.13	Średnie czasy odpowiedzi - po wyłączeniu profilera	76
Tabela 6.14	Średnie czasy odpowiedzi - po wyłączeniu profilera i po włączeniu z mechanizmem cachowania zapytań	76
Tabela 6.15	Porównanie czasów odpowiedzi	78
Tabela 6.16	Wyłączony mechanizm - średnie czasy odpowiedzi	79
Tabela 6.17	Przekierowanie: parametr 150ms - średnie czasy odpowiedzi	79
Tabela 6.18	Przekierowanie: parametr 100ms - średnie czasy odpowiedzi	79

Wstęp

W czasach globalnie postępującego procesu informatyzacji podejście do zagadnienia uczenia ulega ciągłym zmianom. Coraz więcej instytucji publicznych oraz prywatnych, związanych z rynkiem kształcenia, modyfikuje swoje mechanizmy wspierające nauczanie dodając do wachlarza swoich usług, zdalne platformy dydaktyczne. Wraz z rozwojem technologicznym nauczanie na odległość będzie stawało się wiodącym sposobem przekazywania wiedzy. Jest to stosunkowo tani mechanizm posiadający wiele zalet, pozwalających usprawnić proces uczenia zarówno po stronie uczącego się, jak i nauczyciela prowadzącego.

Rozwój tego dość stosunkowo młodego obszaru daje wiele możliwości. W ostatnich latach powstało wiele platform nauczania, zarówno tych małych posiadających kilka kluczowych funkcjonalności, jak i tych większych zapewniających dostęp do szerokiego spektrum zaawansowanych opcji, pozwalających skonfigurować własny system nauczania dopasowany do precyzyjnie określonych potrzeb klienta. Dziedzina ta ciągle się rozwija i poszukiwane są nowe sposoby przekazu wiedzy oraz informacji [12].

Nowe metody kształcenia rozwijają się nie tylko w placówkach edukacyjnych kojarzonych od lat z tą dziedziną. Procesy nauczania stają się coraz to bardziej popularne w środowisku prywatnych firm. Przy bardzo szybkiej dynamice zmian wykorzystywanych technologii wiele branż systematycznie zwiększa nakłady finansowe przeznaczane na procesy szkoleń pracowniczych. Zacieśnia się również więź współpracy pomiędzy firmami oferującymi swoje technologie a placówkami edukacyjnymi, oferującymi doświadczenie naukowe oraz studentów stanowiących pulę przyszłych kandydatów do podjęcia pracy.

Branżą szczególnie podatną na szybkość zmian i braki kadrowe jest sektor nowoczesnych technologii informacyjnych. Ciągłe mówi się o braku specjalistów w dziedzinach takich jak: inżynieria oprogramowania, projektowanie systemów teleinformatycznych, czy chociażby wyspecjalizowane wsparcie techniczne. By zaspokoić ten popyt powstawać będą coraz to nowsze rozwiązania oferujące asynchroniczny proces nauczania. Jest to segment rynku, na którym już dochodzi do rewolucji. Powstaje coraz więcej firm oferujących usługi dydaktyczne, szkolenia oraz mechanizmy pozwalające oceniać postępy pracowników.

Motywe, który zainspirował mnie do zainteresowania się tematyką tej pracy była chęć zbudowania uniwersalnej platformy wspomagającej proces nauczania. Ogrom wiedzy i informacji dostępnych w 21 wieku powoduje ogromne przeciążenie informacyjne. Prowadząc wiele szkoleń oraz prelekcji w ramach działalności studenckiego koła naukowego zauważyłem jak wiele problemów studentom stwarza nadmiar dostępnych informacji. Docelową tematyką tworzonego serwisu jest wspomaganie nauczania dziedziny inżynierii oprogramowania.

Rozdział 1

Cel i zakres pracy

Celem niniejszej pracy jest:

- przedstawienie zagadnień związanych z dziedziną zdalnego nauczania,
- przedstawienie zagadnień związanych ze współczesnymi metodami tworzenia aplikacji webowych,
- analiza oraz praktyczna nauka technologii oraz narzędzi przydatnych przy realizacji aplikacji webowych,
- zaprojektowanie oraz zaimplementowanie systemu zdalnego nauczania,
- testy i ocena efektywności rozwiązania,
- optymalizacja wykorzystywanych mechanizmów.

Merytoryczna zawartość pracy zaczyna się od drugiego rozdziału, zawiera on przegląd literatury związanej ze zdalnym nauczaniem. Główny nacisk położony jest na doprecyzowanie klasyfikacji systemów zdalnego nauczania oraz określenie wymagań funkcjonalnych i нефunkcjonalnych jakie powinna spełniać aplikacja z danej klasy. W trzecim rozdziale opisane są technologie, wspomagające proces implementacji oprogramowania internetowego, wykorzystane w stworzonej aplikacji. Rozdział czwarty prezentuje projekt proponowanego rozwiązania. W rozdziale piątym szczegółowo opisano zrealizowaną aplikację pod kątem wykorzystanych narzędzi oraz zastosowanych wzorców projektowych. Rozdział szósty zawiera procedury testowe oraz prezentuje techniki analizy i optymalizacji aplikacji. Wnioski końcowe przedstawiono w podsumowaniu.

Rozdział 2

Kształcenie na odległość

2.1 Definicja kształcenia na odległość

Nie ma jednoznacznej definicji w pełni określającej problem zdalnego nauczania. W literaturze można znaleźć wiele zwrotów próbujących jednoznacznie określić to zagadnienie.

Terminy często używane w kontekście zdalnego nauczania to e-nauczanie, nauczanie internetowe, nauczanie rozproszone, nauczanie sieciowe, tele-nauczanie, wirtualne nauczanie, nauczanie wspomagane komputerowo, nauczanie webowe, oraz nauczanie na odległość [12].

W wielu opracowaniach naukowych termin ten sprowadza się do postaci "**Kształcenie na odległość**", w niniejszej pracy będę posługiwał się tym terminem.

Wszystkie definicje łączy jedna wspólna cecha. Zakładają one, że podstawą kształcenia na odległość jest niebezpośredni proces uczenia się. Tzn proces, w którym osoba ucząca się nie przebywa w bezpośrednim kontakcie z nauczycielem. Interesant taki korzysta z dostępnych zasobów zdalnie, najczęściej za pomocą komputera. Jest to nowoczesny sposób przekazywania wiedzy, który staje się bardzo popularny na świecie. Wiele uczelni edukacyjnych oraz firm konsultingowych wdraża takie systemy. Alan Clarke w swojej książce słusznie zauważa, że [14]:

E - learning ... Stanowi swoistą rewolucję, której skutki są często porównywalne do wpływu , który wcześniej na kształcenie wywarły wynalezienie druku i masowa produkcja książek.

Do kształcenia na odległość możemy zaliczyć zarówno proces samego nauczania jak i proces wspomagania, bądź wzbogacania mechanizmów dydaktycznych. Do działań wspomagających możemy zaliczyć przede wszystkim sposób prezentacji materiałów oraz sposób dostarczania kolejnych materiałów, bazując na wynikach uczącego się. Założenia te nie są częścią tej pracy dyplomowej. W niniejszej pracy skupimy się na stworzeniu platformy posiadającej mechanizmy wspierające proces zdalnego nauczania, zarówno od strony uczącego się, jak i nauczyciela odpowiedzialnego za tworzenie i nadzorowanie procesu nauczania.

2.2 Korzyści płynące z używania systemów kształcenia na odległość

Proces kształcenia do 19 wieku odbywał się w charakterze bezpośredniego kontaktu w specjalnie wydzielonej do tego celu sali. Rewolucja informatyczna oraz coraz większa znajomość technologii informatycznych wśród społeczeństwa spowodowały, że pod koniec 20 wieku wzrosło zainteresowanie systemami kształcenia na odległość. Kształcenie na odległość posiada wiele cech pozytywnych w porównaniu do starego systemu i podejścia do nauczania

Alan Clark [14] wymienia wiele korzyści płynących z zastosowania e-learningu. Zwraca uwagę m.in. na:

- dostępność materiałów; przeważnie otrzymujemy całodobowy dostęp do interesujących nas materiałów,
- asynchroniczny charakter procesu kształcenia tzn. uczący sam decyduje kiedy chce korzystać z materiałów,
- swoboda wyboru miejsca, czasu oraz tempa uczenia się,
- ułatwiony proces dystrybucji materiałów oraz informacji,
- ułatwienie komunikacji z osobami z całego świata.

Józef Bednarek [13] zwraca również uwagę na zwiększoną efektywność procesu nauczania. Poprzez zwiększony przedział zrozumienia tematu oraz konieczność większego zaangażowania uczących się. Ważnym aspektem przemawiającym na korzyść kształcenia na odległość jest również znaczna oszczędność czasu i zasobów poprzez ułatwiony dostęp do nauczyciela oraz materiału dydaktycznego.

2.3 Ograniczenia kształcenia na odległość

Każde rozwiązanie posiada także i negatywne cechy. Podobnie jest w przypadku zagadnienia kształcenia na odległość. Do najważniejszych problemów związanych z wdrażaniem kształcenia na odległość możemy zaliczyć:

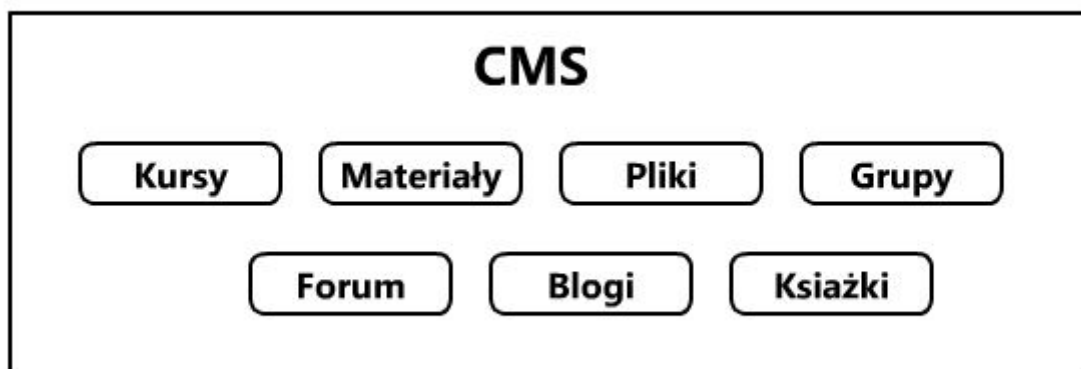
- konieczność przeszkolenia personelu dydaktycznego oraz uczniów,
- wdrożenie kosztownego systemu informatycznego obsługującego proces nauczania,
- wymagane większe zaangażowanie i samodzielna inicjatywa od uczącego się,
- konieczność dopasowania obecnego materiału dydaktycznego do nowych standardów,
- brak typowych dla bezpośredniego procesu nauczania mechanizmów interakcji społecznych mogących jedynie zajść w momencie, gdy nauczyciel oraz uczniowie są w bezpośrednim kontakcie,
- anonimowość nauczyciela.

2.4 Klasyfikacja systemów kształcenia na odległość

A. Clark [14] klasyfikuje systemy E-learningowe w obrębie dwóch podstawowych klas:

- Systemy do zarządzania kursami(CMS),
- Systemy wspomagające proces nauczania (LMS).

2.4.1 Systemy do zarządzania kursami(CMS)



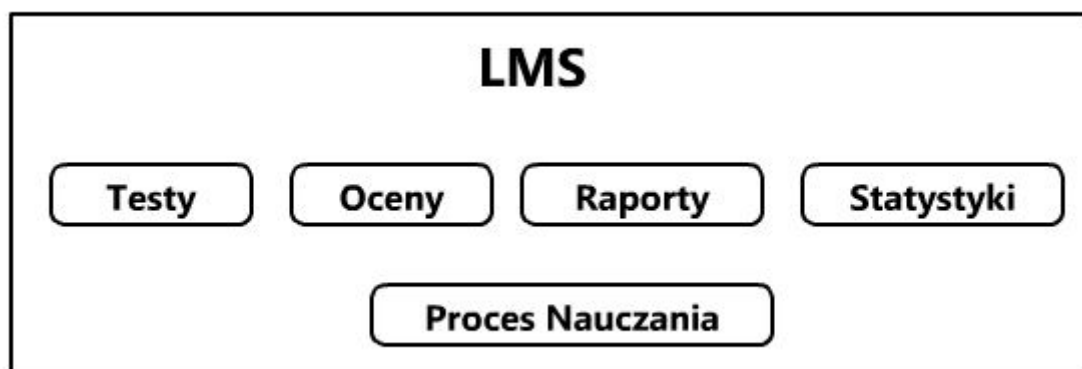
Rysunek 2.1 Podstawowe funkcje systemu klasy CMS

Systemy klasy CMS (Content Management System) są aplikacjami dostarczającymi funkcji pozwalających zarządzać materiałami wspomagającymi proces nauczania, poprzez dostarczanie funkcji ułatwiających tworzenie oraz udostępnianie nowych materiałów. W systemach takich możemy wyróżnić kilka podstawowych funkcji:

- katalogowanie dostępnych materiałów dydaktycznych,
- system administracji pozwalający zarządzać procesem udostępniania materiałów,
- mechanizmy wspomagające proces tworzenia oraz publikowania nowych materiałów.

Systemy takie zazwyczaj sprowadzają się do postaci katalogów udostępniających materiały.

2.4.2 Systemy wspomagające proces nauczania (LMS)



Rysunek 2.2 Podstawowe funkcje systemu klasy LMS

Systemy LMS są bardziej wyspecjalizowanymi systemami wspomagającymi proces nauczania poprzez dostarczanie funkcjonalności takich jak:

- moduł testów pozwalających zdalnie ewaluować umiejętności kursanta,
- moduł ocen pozwalający dokumentować postęp oraz wyniki kursanta,
- moduł wspomagający kolaborację oraz komunikację pomiędzy kursantami i nauczycielami,
- moduł statystyk oraz raportów pozwalających ewaluować skuteczność procesu kształcenia,
- moduł autonomicznego agenta, podającego np sugestie kursantowi, i w ten sposób kierując oraz dostosowując jego tok nauczania do potrzeb.

W niniejszej pracy stworzony zostanie system łączący wybrane cechy obu systemów, tj. charakteryzujące się następującymi własnościami:

- katalogowanie dostępnych materiałów dydaktycznych,
- system administracji pozwalający zarządzać procesem udostępniania materiałów,
- mechanizmy wspomagające proces tworzenia oraz publikowania nowych materiałów,
- moduł testów,
- moduł ocen.

Rozdział 3

Wybrane technologie realizacji systemów webowych

3.1 Formaty danych JSON oraz XML

JSON

JSON (JavaScript Object Notation) jest lekkim formatem danych wykorzystywanym przy transferze danych po sieci. Jest to stosunkowo młody format danych opisany w 2006 roku w publikacji RFC4627 [17]. Charakteryzuje się on bardzo prostą, ale zarazem czytelną reprezentacją danych. Zbiór danych składa się z pary ciągów znaków: nazwa, wartość.



Rysunek 3.1 Uproszczony Schemat formatu JSON

Najważniejszą cechą przemawiającą za używaniem formatu JSON jest jego prostota, czytelność oraz co najważniejsze ułatwione przetwarzanie danych w formacie JSON na obiekty języka Javascript za pomocą funkcji `eval()`. Cecha ta pozwala usprawnić proces tworzenia aplikacji webowych po stronie klienta, tj. z przeglądarki.

Cechy formatu JSON

- ułatwione przetwarzanie formatu do obiektów języka Javascript,
- prosty format,
- szybki proces przetwarzania,
- łatwy sposób modyfikowania.

XML

XML jest jednym z najbardziej rozpowszechnionych formatów danych. Wykorzystywany jest nie tylko jako nośnik danych, ale również jako popularny format m.in. plików konfiguracyjnych. Wiele skomplikowanych aplikacji wykorzystuje format XML do definiowania, m.in. : scenariuszy testowych, interakcji w systemie, przepływów danych. Na bazie tego formatu powstał również format XHTML, który łączy cechy formatu HTML oraz XML.

Format XML powstał na bazie formatu SGML, jest jego rozszerzeniem. Dokument standaryzujący opisuje nie tylko sposób formatowania danych, ale również standardową metodę przetwarzania plików [28].



Rysunek 3.2 Uproszczony schemat formatu XML

Cechy formatu XML

XML składa się ze znaczników zamykających oraz otwierających zwanych tagami, wewnątrz których znajduje się zawartość danego elementu. W formacie tym istnieje możliwość definiowania atrybutów opisujących dany tag. Standard XML charakteryzują:

- bardzo duże wsparcie narzędzi i zgodność z wieloma systemami na rynku,
- duże bezpieczeństwo przesyłanej wiadomości,
- możliwość definiowania dodatkowych atrybutów,
- pochodne XML-a, takie jak XML Schema, XSLT, wzbogacające format o dodatkowe funkcjonalności.

Porównanie JSON oraz XML

W momencie wejścia na rynek usług sieciowych, były one głównie wykorzystywane w świecie biznesowym. Rynek Enterprise zaadoptował na początku standard XML i protokół oparty na formacie SOAP [27]. XML był idealnym kandydatem, był już dobrze znany i miał spore wsparcie narzędzi oraz środowisk programistycznych.

Wzrost zainteresowania formatem JSON zaczął się w 2006 roku wraz z powstaniem pierwszego oficjalnego opisu oraz, wraz z adopcją tego formatu przez takie firmy jak **Google** czy **Yahoo** w swoich zewnętrznych API.

Format XML charakteryzuje się większym poziomem bezpieczeństwa dzięki bardziej stabilnej i mocno typowanej semantyce. W przypadku formatu JSON nie ma możliwości określenia typu danych wartości ponieważ w jego specyfikacji brakuje możliwości definiowania atrybutów. Problem ten można pominąć dzięki zastosowaniu JSON schema, określającego typy danych w oddzielnym dokumencie.

Format JSON może zostać bardzo łatwo przetworzony za pomocą funkcji eval dostępnej w języku programowania Javascript. W związku z tym w przypadku aplikacji nadmier- nie wykorzystujących ten sposób przetwarzania istnieje zwiększone ryzyko wstrzyknięcia niepożądanych skryptów.

Najważniejszą różnicą pomiędzy oboma formatami jest jednak ilość bajtów wymagana do przesłania reprezentacji odpowiednich danych.

```

{
  "ID": 1,
  "Role": "Author",
  "name": "Franc",
  "first-name": "Michał",
  "age": 25,
  "hobbies": [
    "reading",
    "programming",
    {
      "sports": [
        "volley-ball",
        "football"
      ]
    }
  ],
  "address": {
    "City": "Wrocław",
    "Street": "Mosiezna",
    "NR": "19/22"
  }
}

```

```

<?xml version="1.0" encoding="UTF-8" ?>
- <json>
  <ID>1</ID>
  <Role>Author</Role>
  <name>Franc</name>
  <first-name>Michał</first-name>
  <age>25</age>
  <hobbies>reading</hobbies>
  <hobbies>programming</hobbies>
- <hobbies>
  <sports>volley-ball</sports>
  <sports>football</sports>
</hobbies>
- <address>
  <City>Wrocław</City>
  <Street>Mosiezna</Street>
  <NR>19/22</NR>
</address>
</json>

```

Rysunek 3.3 Porównanie standardów (JSON , XML)

W pracy magisterskiej zastosowałem zarówno format JSON jak i XML. Dane używane przez serwis przesyłane są w formacie XML, natomiast dane udostępniane przez API przesyłane są w formacie JSON.

3.2 Protokoły SOAP oraz REST

Współczesne aplikacje webowe, udostępniające usługi sieciowe, zbudowane są w większości na bazie protokołów REST oraz SOAP.

SOAP

Protokół SOAP jest protokołem transmisji danych wykorzystującym głównie format XML (może również wykorzystywać inne formaty jednak XML jest tym najpopularniejszym). Wiadomości przesyłane są standardowo za pomocą protokołu HTTP przy użyciu komend **GET**, **POST**, **DELETE**. Istnieje też możliwość wykorzystania protokołu RPC. Standard SOAP definiuje strukturę wiadomości przesyłanej do serwera. Jego opis został zawarty w dokumencie sporządzonym przez konsorcjum W3C [27].

Wiadomość SOAP składa się z następujących segmentów:

- Envelope,
- Header ,
- Body.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" xmlns:a="http://www.w3
  <s:Header>
    <a:Action s:mustUnderstand="1">http://tempuri.org/IProfileService/GetByName<
    <a:MessageID>urn:uuid:675b3f2e-97a9-48a8-872b-e001be035a39</a:MessageID>
    <a:ReplyTo>
      <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
    </a:ReplyTo>
    <a:To s:mustUnderstand="1">http://lam-pc:8888/ProfileService.svc</a:To>
  </s:Header>
  <s:Body>
    <GetByName xmlns="http://tempuri.org/">
      <userName>user12</userName>
    </GetByName>
  </s:Body>
</s:Envelope>
```

Rysunek 3.4 Przykładowa wiadomość SOAP

Segment Header zawiera nagłówek informacji. W przedstawionym przykładzie na rysunku (Rysunek 3.4) widać m.in. adres, do którego kierowana jest wiadomość oraz wartość **uuid** przydzielana każdej wiadomości, będąca unikatowym identyfikatorem wiadomości. Segment Body zawiera przesyłane dane, w tym przypadku m.in nazwę funkcji udostępnianej przez usługę sieciową wraz z parametrami.

```

<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" xmlns:a="http://www.w3
  <s:Header>
    <a:Action s:mustUnderstand="1">http://tempuri.org/IProfileService/GetByName<,
    <a:MessageID>urn:uuid:675b3f2e-97a9-48a8-872b-e001be035a39</a:MessageID>
    <a:ReplyTo>
      <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
    </a:ReplyTo>
    <a:To s:mustUnderstand="1">http://lam-pc:8888/ProfileService.svc</a:To>
  </s:Header>
  <s:Body>
    <GetByName xmlns="http://tempuri.org/">
      <userName>user12</userName>
    </GetByName>
  </s:Body>
</s:Envelope>

```

Rysunek 3.5 Przykładowa wiadomość zwrotna

Można zauważyć charakterystyczny **uuid** wiadomości, który jest taki sam jak w przypadku wiadomości wysłanej do serwera. Na tej podstawie serwer udostępniający usługi sieciowe dopasowuje obie wiadomości i wie, że serwer odpowiedział na żądanie. Jak widzimy (Rysunek 3.5) w tym przykładzie w segmencie body został przesłany wynik wykonania usługi.

Zalety SOAP

- stała składnia oraz mocne typowanie,
- dużo narzędzi na rynku,
- dojrzałość standardu.

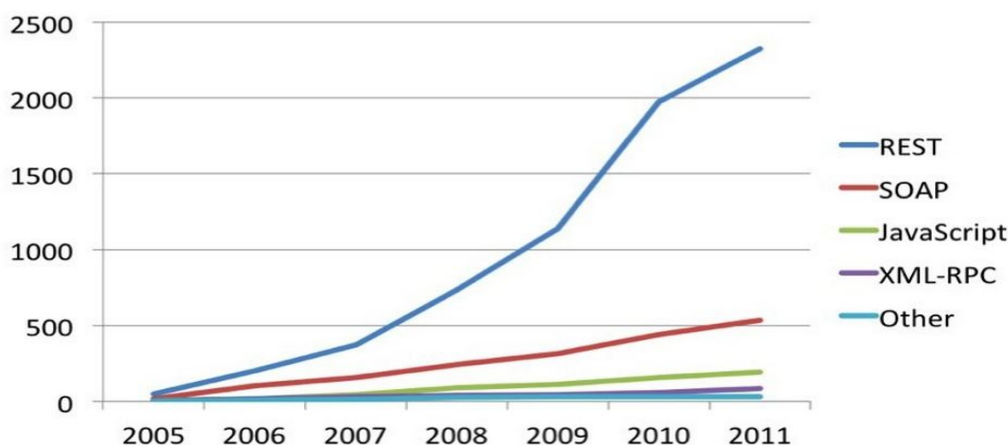
Wady SOAP

- przesyłanie nagłówków zwiększa ilość przesyłanych danych,
- duży poziom komplikacji.

REST

SOAP i jego rozszerzenia powodują, że staje się on dość skomplikowany, poza tym wysyła wraz z zapytaniem do serwera dodatkowe dane w formie segmentów. W środowisku nowych, małych firm zaistniała potrzeba na *lżejszy* protokół. Odpowiedzą na oczekiwania rynku stał się protokół REST [19]. Popularność tego protokołu zaczęła znacząco rosnąć wraz pojawieniem się usługi mikro-blogów oferowanej przez serwis **Twitter** oraz wystawienia przez tę usługę rozbudowanego API. Interfejs API **Twitter-a** używa protokołu REST oraz formatu JSON do udostępniania danych na temat użytkowników.

Badania z Maja 2011 [15] roku przeprowadzone przez serwis **ProgrammableWeb** pokazują, że prawie 73% usług sieciowych wystawionych w formie API obsługiwane jest na bazie protokołu REST.



Distribution of API protocols and styles

Based on directory of 3,200 web APIs listed at ProgrammableWeb, May 2011

Rysunek 3.6 Porównanie protokołów Rest / Soap [15]

Architektura typu REST opiera się na jednym ważnym założeniu. Pobieranie oraz modyfikowanie obiektów jest ściśle powiązane z adresem URL. Dzięki zastosowaniu takiego rozwiązania można pobierać, np zawartość bazy danych za pomocą komend protokołu HTTP. Pobieranie można realizować np za pomocą komendy GET. Modyfikacje bądź usuwanie danych za pomocą komendy POST.

GET `http://codedashservices.mfranc.com/CourseService.svc/json/Get?id=16`
HTTP/1.1

Przykładowa wiadomość protokołu REST

Komenda wysyłana do serwera **HTTP** zawiera w sobie nazwę komendy **GET** oraz Adres. Funkcja oraz parametry zawarte są w treści adresu **URL**. W tym przypadku segment **Get?id=16** zawiera nazwę funkcji oraz parametr **id=16**. Jak widać jest to zwykłe zapytanie bez dodatkowych informacji w przeciwieństwie do protokołu **SOAP**, w którym należy wysyłać całą wiadomość w formacie XML. W przypadku protokołu **REST** wszystkie istotne dane, jak nazwa funkcji oraz parametr, zawarte są w adresie **URL**.

```
{
  "CourseType": {
    "ID": 1,
    "TypeName": "csharp"
  },
  "CreationDate": "\/Date(1305319615000+0200)\/",
  "ID": 35,
  "Logo": "csharp",
  "Name": "C# Basic",
  "ShortDescription": "C# Basics Course"
}
```

Rysunek 3.7 Przykładowa wiadomość zwrotna protokołu REST

W tym przypadku (Rysunek 3.7) używamy formatu danych JSON do zwrócenia danych z bazy danych wystawionej za usługą sieciową.

Zalety REST

- prostota i lekkość, nie ma potrzeby wysyłania dodatkowych danych,
- czytelność.

Wady REST

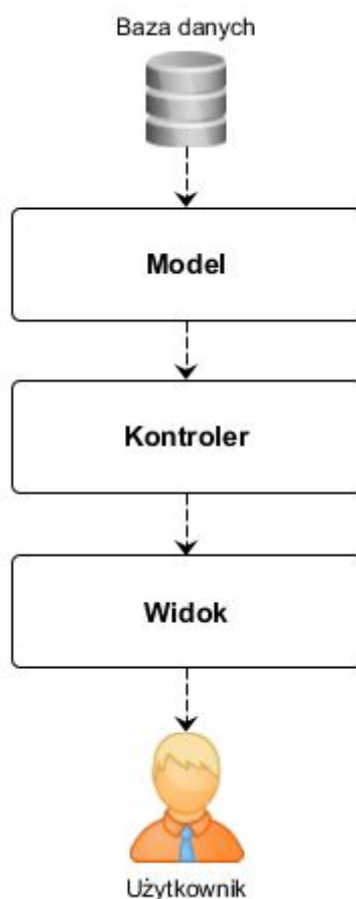
- prostota powoduje, że nie można tworzyć skompilowanych wiadomości,
- dostępnych mniej opcji,
- zmniejszone bezpieczeństwo.

3.3 Framework Asp.Net MVC

Asp.Net Mvc jest platformą stworzoną przez firmę Microsoft, służącą do tworzenia aplikacji webowych. Działa ona jako nakładka na platformę **Asp.Net**. Jest odpowiedzią na nowe trendy, zdobywające coraz to większą popularność w środowisku oprogramowania webowego. Główną zaletą tego rozwiązania jest fakt, że jest ono oparte na wzorcu projektowym **MVC**.

Wraz z ewolucją aplikacji webowych i ich poziomem skomplikowania pojawiały się nowe podejścia oraz sposoby wytwarzania aplikacji webowych pozwalające tworzyć wielowarstwowe aplikacje internetowe, np systemy zdalnego nauczania. Jednym z takich podejść jest wykorzystanie wzorca *Model View Controller*. Pierwszy opis wzorca można znaleźć w dokumencie z 1979 [26]. Prawdziwa rewolucja zaczęła się w 2004 roku wraz z pojawieniem się nowych platform developerskich takich jak : **Ruby on Rails** [7] na języku **Ruby** oraz **Django** [4] związanego z językiem **Python**.

Wzorzec ten wprowadza podział aplikacji na trzy oddzielne warstwy : Model, Widok, Kontroler, które umożliwiają niezależny rozwój aplikacji w warstwach oraz zapewnia zwiększoną skalowalność systemu.



Rysunek 3.8 Uproszczona koncepcja wzorca MVC

Model

Reprezentuje warstwę danych, która może być w postaci m.in. bazy danych, pliku. Warstwa ta jako jedyna ma dostęp do źródła danych. Dostęp do modelu jest jedynie możliwy z poziomu kontrolera, który korzysta z metod zdefiniowanych w warstwie modelu. W wielu aplikacjach pośrednio do komunikacji z warstwą źródła danych używa się dodatkowo warstwy usług.

Kontroler

Warstwa odpowiedzialna za sterowanie przepływem danych i przetwarzanie tych danych do postaci wyświetlanej w warstwie widoku. Kontroler decyduje, który widok zostanie wyświetlony.

Widok

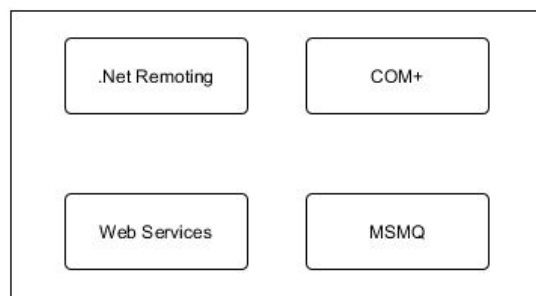
Reprezentuje warstwę bezpośrednio dostępną dla użytkownika systemu. Tworzony jest na podstawie modelu przekazanego mu przez kontrolera.

Zastosowanie wzorca MVC przy projektowaniu aplikacji webowych wymaga większego nakładu pracy w początkowej fazie projektu. Wymierne korzyści ze stosowania tego wzorca zaczynają być zauważane dopiero w późniejszych etapach życia projektu. Przede wszystkim zastosowane konwencje i separacja odpowiedzialności na trzy warstwy pozwala oddzielić od siebie logikę biznesową dostępną z poziomu klienta od logiki obsługującej dostęp do bazy danych. Jest to bardzo ważne ponieważ zmiany zachodzące w warstwie modelu, tzn. bazy danych, nie powinny powodować zmian w warstwie widoku. Dzięki takiemu rozdziałowi powstaje lepszy kod, łatwiejszy w rozbudowanie oraz utrzymaniu. Dodatkowo projekt jest bardziej czytelny. Programista wiedzący, że projekt został stworzony w oparciu o MVC automatycznie wie gdzie szukać poszczególnych implementacji systemu w celu przeprowadzenia modyfikacji.

Opis komunikacji

Komunikacja w architekturze **MVC** rozpoczyna się od klienta zgłaszającego żądanie (np. o wyświetlenie danej strony **WWW**). Żądanie to jest przechwytywane przez kontroler. Jeżeli wygenerowanie strony **www** nie wymaga pobrania danych ze źródła danych, kontroler pobiera dany widok i przekazuje go klientowi. Jeżeli natomiast widok wymaga pobrania danych z bazy danych, realizowane jest połączenie z modelem za pośrednictwem, którego pobierane są dane i przekazywany jest odpowiedni widok w formie wiadomości zwrotnej.

3.4 Technologia WCF - Windows communication foundation



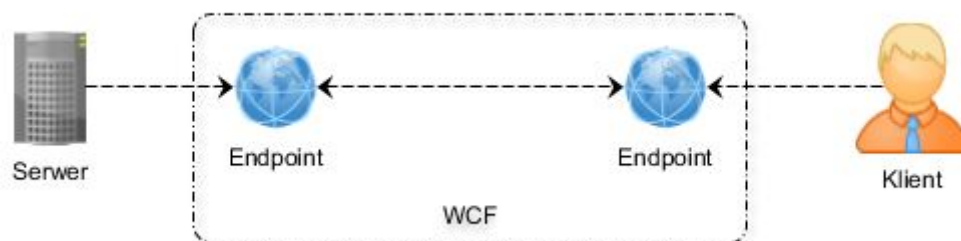
Rysunek 3.9 Framework WCF

WCF [24] jest nowym frameworkiem stworzonym przez firmę Microsoft, wykorzystywanym do tworzenia rozproszonych systemów informatycznych. Rysunek 3.10 przedstawia ogólny zarys frameworka, w którego wchodzi wszystkie poprzednie rozwiązania stosowane na platformie wspieranej przez Microsoft. Mimo tego, że framework ten wprowadza wiele usprawnień, zapewnia on kompatybilność z wcześniejszymi rozwiązaniami.

Do zalet WCF można zaliczyć m.in:

- kompatybilność ze starymi technologiami firmy Microsoft,
- łatwość konfiguracji, modyfikacji oraz tworzenia rozszerzeń
- obszerny zestaw klas i metod, wspomagających i ułatwiających proces implementacji.

Do wad WCF zaliczyć można przede wszystkim, konieczność większego nakładu pracy w przypadku tworzenia połączenia z platformą nie związaną z technologiami firmy Microsoft.



Rysunek 3.10 Framework WCF

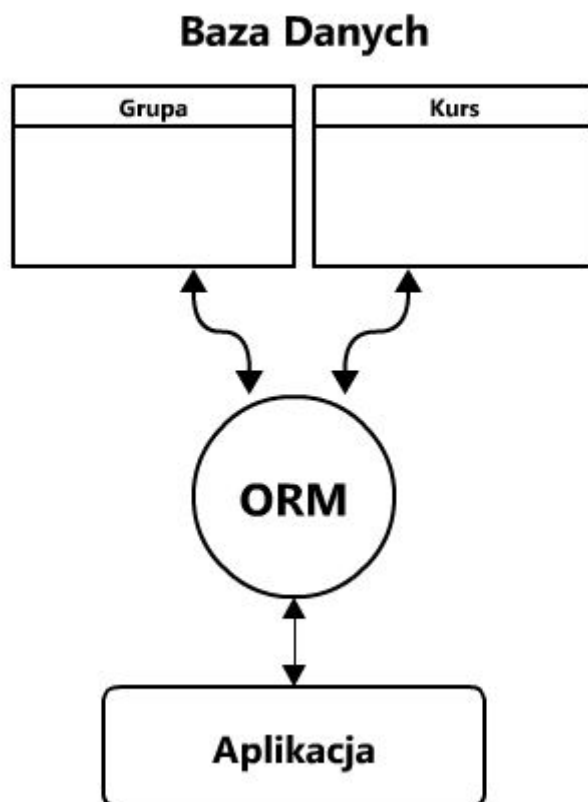
Działanie WCF opiera się na odpowiednim zdefiniowaniu ednpointów, będących swego rodzaju adresami wskazującymi na dane usługi sieciowe oferowane przez serwer. Serwer

ma możliwość udostępnienia wielu endpointów charakteryzujących się różnymi parametrami, dzięki takiemu rozwiązaniu można zapewnić komunikację z serwerem dla różnych rodzajów klientów komunikujących się za pomocą różnych protokołów, i technologii. WCF chowa całą logikę pod "fasadą" kontraktów posiadających nagłówki metod udostępnianych w ramach danej usługi.

3.5 Mapowanie obiektowo-relacyjne - NHibernate

Bazy danych są najważniejszą częścią systemu informatycznego. Implementacja dostępu do bazy jest jednym z bardziej czasochłonnych elementów realizacji projektu po stronie serwera. Dodatkowo logika ta jest szczególnie podatna na błędy. Bezpośrednie tworzenie zapytań stało się zbyt kosztowne oraz trudne w utrzymaniu. Z rozwiązaniami takimi wiąże się również inny problem, mianowicie dochodzi do niekompatybilności pomiędzy sposobem przedstawiania powiązań obiektów w relacyjnych systemach bazodanowych, a odpowiednimi mechanizmami opartymi na dziedziczeniu i kompozycji klas, znajdującymi się w współczesnych platformach programistycznych.

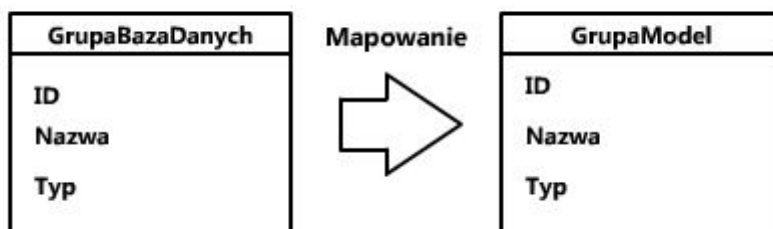
By ułatwić proces tworzenia kodu coraz więcej firm wykorzystuje specjalne biblioteki wspomagające proces tworzenia warstwy dostępu do danych. Nazywane są one Obiektowo Relacyjnymi Mapperami (dalej zwanymi ORM-ami). Na rynku dostępnych jest wiele rozwiązań tego typu. Do najpopularniejszych w środowisku .Net należą m.in: **Entity Framework** [2] oraz **NHibernate** [1].



Rysunek 3.11 ORM obsługujący bazę danych

Dzięki zastosowaniu ORM-a można wprowadzać bardzo szybko zmiany oraz w dużo łatwiejszy sposób wykonywać odpowiednie zapytania na bazie danych, nie przejmując się tak naprawdę warstwą bazodanową. Z punktu widzenia programisty posługującego się warstwą danych opartą na mapperze cała komunikacja jest realizowana z wykorzystaniem interfejsów. Dzięki temu można skupić się na implementacji konkretnej logiki związanej z funkcjonalnością aplikacji, oszczędzając czas na implementowaniu dostępu do bazy danych. Minusem takiego rozwiązania jest mniej wydajny proces pobierania danych. Problem ten można zniwelować poprzez odpowiednie sprofilowanie aplikacji i wyznaczenie części systemu wymagających optymalizacji. W tym przypadku profilowane są zapytania SQL. Zapytania wymagające optymalizacji można zamienić na zwykłe zapytania języka SQL. Dzięki takiemu rozwiązaniu oszczędza się czas oraz fundusze przeznaczane na projekt.

Proces mapowania sprowadza się do określenia, które pole z bazy danych ma być powiązane z polem obiektu wykorzystywanego w aplikacji. W ten sposób tworzone są specjalne klasy pośredniczące w komunikacji pomiędzy bazą danych a aplikacją.



Rysunek 3.12 Rysunek przedstawiający proces mapowania.

W niniejszej pracy dyplomowej zdecydowano się zastosować framework **NHibernate** [1]. Jest to rozwiązanie będące implementacją frameworka **Hibernate** dostępnego w środowisku **Javy** na potrzeby środowiska **.NET**. W **NHibernate** odpowiednie mapowania można zdefiniować za pomocą plików konfiguracyjnych XML. Jest to wygodne podejście jednakże podatne na błędy i nieczytelne. Dlatego często stosuje się rozwiązanie typu **FluentNHibernate** będące biblioteką, która pozwala generować pliki XML na podstawie kodu napisanego w języku platformy **.Net**. Mapowanie takie staje się bardziej czytelne m.in. dzięki zastosowaniu wzorca projektowego **FluentInterface** [23], pozwalającego łączyć sekwencyjnie wywoływanie metod.

Rozdział 4

Projekt systemu zdalnego nauczania

4.1 Wymagania projektowe

W niniejszym punkcie zawarto opis podstawowych elementów projektu.

4.1.1 Wymagania funkcjonalne

Tworzony system łączy w sobie pewne wybrane cechy systemów klasy CMS oraz LMS. Do najważniejszych funkcji należy możliwość edytowania oraz publikowania kursów, testów, materiałów zdalnego nauczania oraz mechanizm dzienników ocen. Aplikacja wspiera proces tworzenia materiałów oraz testów poprzez prosty, ale funkcjonalny system edycji.

Tworzenie użytkowników oraz przydzielanie uprawnień

System posiada mechanizm pozwalający zarządzać użytkownikami aplikacji wraz z możliwością przydzielania im odpowiednich poziomów uprawnień. Dostępny jest panel administracyjny zawierający listę użytkowników oraz udostępniający funkcje:

- Dodaj/Usuń użytkownika,
- Aktywuj/Dezaktywuj użytkownika,
- Modyfikacja uprawnień,
- Dodaj/Usuń użytkownika z grupy.

Usunięcie użytkownika równoznaczne jest z usunięciem wszystkich danych powiązanych z tym użytkownikiem. Szczególnym przypadkiem usuwania użytkownika jest usuwanie użytkownika posiadającego uprawnienia umożliwiające tworzenie kursów oraz materiałów e-learningowych. W przypadku usunięcia takiego użytkownika kursy zarządzane przez usuwanego użytkownika przechodzą na własność administratora systemu, który może przydzielić uprawnienia autorskie innemu użytkownikowi.

Zarządzanie kursami

Każdy użytkownik posiadający uprawnienia autorskie oraz administrator może stworzyć jeden lub więcej kursów. Kurs opisany jest parametrami:

- Nazwa,
- Krótki opis,
- Logo,
- Rodzaj kursu.

Każdy kurs posiada swoją odrębną listę materiałów nauczania, testów, mechanizm wymiany krótkich wiadomości **Shoutbox** oraz pojedynczą grupę użytkowników przynależących do danego kursu. Autor ma możliwość edycji tylko i wyłącznie kursów, którymi zarządza. Administrator może modyfikować każdy kurs znajdujący się w systemie.

Zarządzanie materiałami nauczania

Materiały nauczania są integralną częścią kursu. Każdy kurs może posiadać jeden lub więcej materiałów nauczania. W skład materiału nauczania wchodzi parametry opisowe pozwalające sklasyfikować dany materiał:

- Poziom materiału *Początkujący, Średnio-Zaawansowany, Zaawansowany*,
- Opis,
- Logo.

Uprawnienia do edycji materiałów posiada administrator oraz autor kursu, do którego należy dany materiał.

Materiały nauczania - proces nauczania

Materiał nauczania prócz parametrów opisowych podzielony jest na trzy obszary.

Obszar informacyjny

Zawiera segmenty:

- Informacyjny - parametry kursu,
- Opisowy - opisuje kontekst kursu,
- Celów - opisuje cele jakie osiągnie kursant biorący udział w kursie.

Obszar nauczania

W skład obszaru nauczania wchodzi sekcje zawierające materiały z zawartością merytoryczną, z której korzysta kursant. Materiał nauczania może posiadać jedną lub więcej sekcji. Każda sekcja opisana jest tytułem oraz polem zawartości, w którym umieszczamy treść oraz materiały.

Obszar podsumowania

Zawiera segmenty:

- Podsumowania - szybkie podsumowanie zdobytej wiedzy i najważniejszych rzeczy,
- Więcej informacji - posiada zbiór dodatkowych materiałów poszerzającym zawartość kursu,
- Testów - posiada test stworzony na potrzeby materiału nauczania pozwalający sprawdzić wiedzę kursanta,

Mechanizm nauczania stworzony jest w sposób liniowy. Kursant po kolei odkrywa kolejne segmenty oraz sekcje wchodzące w skład materiału nauczania. Po skończeniu procesu nauczania kursant może sprawdzić swoją wiedzę wykonując test.

Zarządzanie testami

Aplikacja pozwala tworzyć testy, będące istotnym elementem procesu nauczania. Testy mogą być powiązane z materiałem nauczania. Możliwość tworzenia oraz edycji testów posiada Administrator, oraz Autor kursu, posiadający odpowiednie uprawnienia.

Mechanizm rozwiązywania oraz sprawdzania testów

Kursant po skończeniu procesu nauczania może przystąpić do rozwiązywania testu. Test składa się z dwóch obszarów : treści pytania oraz przydzielonych odpowiedzi. Po rozwiązaniu testu wyświetlana jest ocena oraz wiadomość informująca o zaliczeniu bądź nie zaliczeniu testu. Następnie ocena zapisywana jest do dziennika ocen danego ucznia.

Zarządzanie ocenami

Każdy kursant może przeglądać listę swoich ocen otrzymanych w procesie nauczania po rozwiązaniu testu. Kursant ma także możliwość automatycznego wyliczenia średniej ocen z danego kursu.

Filtrowanie kursów oraz materiałów

Kursant ma możliwość filtrowania kursów na podstawie typu kursu. Może również wyświetlić listę kursów, do których jest już zapisany. Kursant ma możliwość filtrowania materiałów nauczania na podstawie poziomu trudności oraz może sortować materiały na podstawie ich nazwy.

4.1.2 Wymagania niefunkcjonalne

Bardzo dobra jakość kodu i projektu

Platforma zdalnego nauczania stworzona zostanie zgodnie z nowoczesnymi trendami oraz zasadami dobrego projektowania aplikacji tak by w przyszłości nakład pracy włożony w proces modyfikacji i utrzymania kodu był jak najmniejszy.

Bezpieczeństwo aplikacji

Dostęp do poszczególnych funkcjonalności systemu jest ograniczony w obrębie ról przydzielanych do poszczególnych użytkowników platformy. Wyróżniamy trzy role:

Administrator

Administrator reprezentuje użytkownika odpowiedzialnego za zarządzanie całym systemem zdalnego nauczania.

Autor

Autor jest użytkownikiem posiadającym możliwość tworzenia nowych kursów, edytowania kursów, materiałów nauczania oraz testów.

Kursant

Kursant jest podstawowym użytkownikiem posiadającym możliwość przeglądania zasobów aplikacji i interakcji z systemem bez możliwości wpływania na zawartość systemu. Kursanci posiadają możliwość umieszczania krótkich wiadomości tekstowych w module ShoutBox, wchodzącym w skład każdego kursu.

Mechanizm rejestracji oraz logowania do systemu zostanie zrealizowany za pomocą frameworka Asp.Net Membership [18]. W tym celu powstanie druga baza danych, zawierająca dane wymagane w procesie logowania, powiązana z główną bazą danych.

Bezpieczeństwo transmisji danych

Platforma zdalnego nauczania komunikuje się z warstwą bazodanową za pomocą usług sieciowych. Transmisja danych przebiegała będzie na dwóch poziomach zabezpieczeń. Pierwszy poziom bez szyfrowania i bez zabezpieczeń będzie służył do przesyłania danych zawierających jedynie dane na temat kursów, testów oraz materiałów nauczania. Dane przesyłane na tym poziomie będą przesyłane w niezaszyfrowanej postaci. Dane poufne, takie jak dane logowania i hasła, będą przesyłane bardziej bezpiecznym sposobem transmisji.

Komunikacja za pomocą usług sieciowych

Dostęp do bazy danych wystawiony będzie za warstwą usług sieciowych zrealizowanych w technologii **WCF(Windows Communication Foundation)** [24]. Poszczególne funkcjonalności bazy danych będą udostępnione w postaci sześciu usług sieciowych:

- Kursy,
- Testy,

- Profile,
- Materiały nauczania,
- Grupy,
- Dziennik.

Dodatkowo część funkcji zostanie udostępniona w formacie JSON na protokole REST, zapewniając dostęp do wybranych danych aplikacjom nie powiązanym z platformą. By zaprezentować tę funkcjonalność zostanie stworzony dodatek do popularnej platformy **WordPress** [11], pozwalający wyświetlić kursy danego autora na blogu.

Mechanizm dynamicznego przekierowania na serwer zapasowy

By zwiększyć niezawodność systemu stworzony zostanie system, który w momencie problemów z komunikacją z podstawowym serwerem bazy danych przełączy system na zapasową bazę danych. Co jakiś czas generowane będzie zapytanie o dostępność serwera i w momencie braku odpowiedzi bądź błędu wysyłanego protokołem SOAP nastąpi automatyczne przekierowanie na usługi sieciowe serwera zapasowego.

Tworzenie kopii zapasowych bazy danych

Każdego dnia o określonej porze będzie tworzona kopia zapasowa bazy danych. Jednorazowo trzymanych będzie siedem kopii.

Automatyczny mechanizm replikacji danych

Ponieważ rozwiązanie zakłada możliwość przekierowania na innych serwer usług, posiadający kopię zapasową bazy danych. Musi zostać zapewniony mechanizm replikacji uaktualniający bazę danych na zapasowym serwerze. Replikacja taka będzie wykonywana raz dziennie.

Dostępność aplikacji

Aplikacja będzie dostępna z poziomu przeglądarki internetowej. Będzie dostosowana funkcjonalnie do przeglądarek:

- Chrome,
- Firefox 4+,
- Internet Explorer 9+,
- Opera.

Zostanie zagwarantowany dostęp do funkcjonalności z poziomu wymienionych wyżej przeglądarek. Nie zostanie zagwarantowany "idealnie" taki sam sposób prezentacji aplikacji.

Obciążalność

Aplikacja będzie w stanie obsłużyć jednorazowo 100 żądań i będzie gwarantować czas odpowiedzi z serwera usług w granicach maksymalnie 1 sekundy

4.2 Projekt bazy danych

4.2.1 Opis encji

Kurs

Zawiera dane dotyczące kursu : Id, Typ Kursu, Nazwę, Logo, Data Utworzenia, Opis, Krótki Opis, Pole Wiadomości

TypKursu

Słownik opisujący typ kursu.

UkonczonyTest

Encja reprezentująca ukończony test. Zawiera m.in otrzymaną ocenę oraz datę ukończenia testu.

Test

Reprezentuje Test wypełniany przez kursanta, należący do materiału nauczania. Zawiera dane opisowe oraz sekcje zawierające merytoryczną treść kursu.

TypTestu

Słownik opisujący typ testu.

PytanieTestowe

Reprezentuje pytanie należące do zbioru pytań wchodzących w skład testu. Zawiera listę odpowiedzi, tytuł oraz treść pytania.

PytanieTestoweOdpowiedz

Reprezentuje odpowiedzi będące częścią pytania. Zawiera treść odpowiedzi oraz wskaźnik, określający czy jest to poprawna odpowiedź.

ShoutBox

Encja reprezentująca moduł krótkich wiadomości tekstowych. Zawiera listę wiadomości.

ShoutBoxWiadomosc

Reprezentuje wiadomość wyświetlaną w obrębie ShoutBoxa. Zawiera treść, login użytkownika oraz datę przesłania wiadomości.

Dziennik

Encja reprezentująca dziennik ocen. Każdy kursant posiada pojedynczy obiekt dziennika dla każdego kursu, do którego dołączył. Zawiera listę ocen przypisanych do danego użytkownika oraz kursu.

DziennikOcena

Encja ocena wchodząca w skład dziennika. Reprezentuje pojedynczą ocenę otrzymywaną po skończeniu testu.

MaterialNauczania

Opisuje pojedynczy materiał nauczania, będący podstawowym środkiem przekazu merytorycznej zawartości dla kursanta.

Section

Sekcja jest integralną częścią materiału nauczania. Zawiera informacje dydaktyczne. Materiał nauczania może posiadać wiele sekcji.

Profil

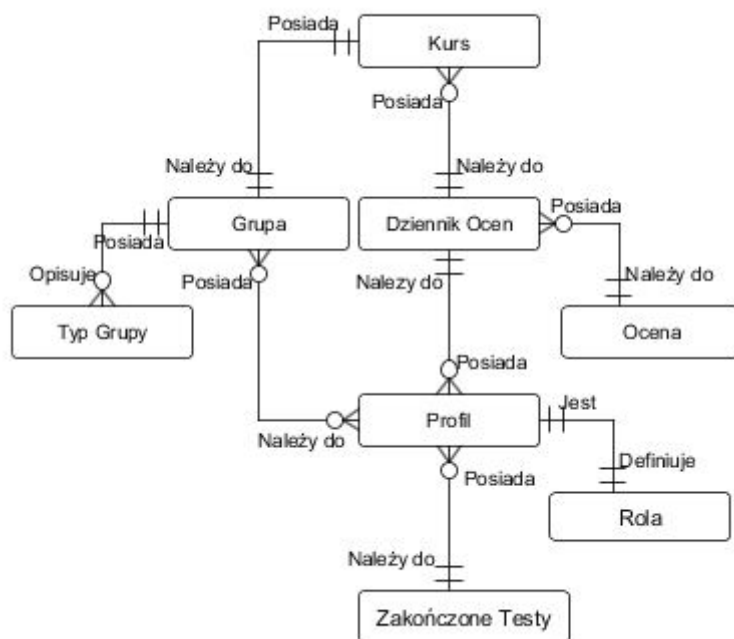
Reprezentuje użytkownika systemu.

GrupaUzytkownikow

Encja opisująca grupę użytkowników. Każdy kurs posiada pojedynczą grupę użytkowników. Zawiera listę użytkowników.

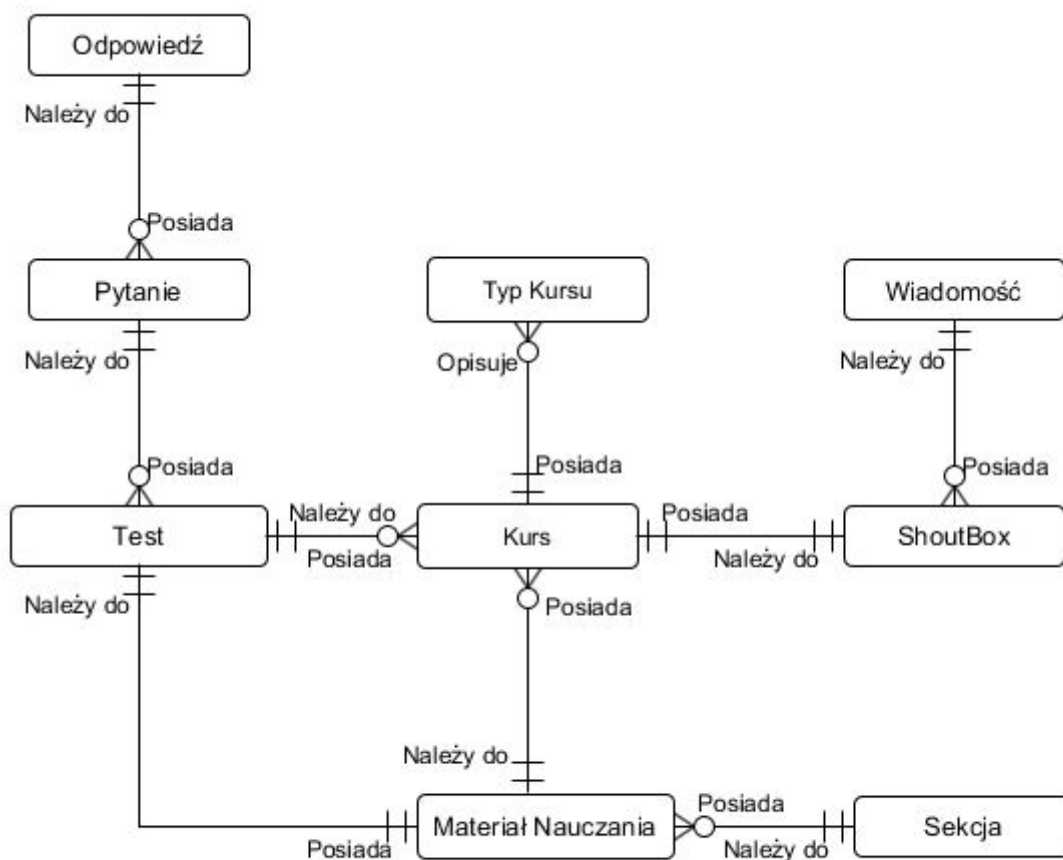
4.2.2 Uproszczony model konceptualny (CDM) - struktura tabel i relacje

Zaprezentowane diagramy przedstawiają związki zachodzące pomiędzy poszczególnymi encjami. Cały diagram został podzielony na dwie części, związane z najważniejszymi encjami tzn. encją reprezentującą Profil użytkownika, oraz encją reprezentującą pojedynczy Kurs.



Rysunek 4.1 Diagram CDM cz. 1

Rysunek 4.1 przedstawia uproszczony diagram **CDM** opisujący związki encji w obrębie Profilu. Zgodnie z wymaganiami funkcjonalnymi, każdy profil posiada dzienniki ocen, określoną rolę, zbiór zakończonych testów oraz przynależy do wielu grup. Profil użytkownika jest powiązany z Kursem przez Grupę oraz Dziennik Ocen, który posiada oceny związane z danym kursem. Przynależność do określonej Grupy powiązanej z Kursem, pozwala określić czy użytkownik posiada podstawowe uprawnienia dostępu do zasobów Kursu. Dziennik ocen posiada zbiór ocen, oraz należy do Kursu. Dzięki takiemu rozwiązaniu można szeregować oceny w zależności od Kursu. Profil jest opisany m.in przez encję Rola, która pozwala określić poziom uprawnień danego użytkownika w obrębie aplikacji.

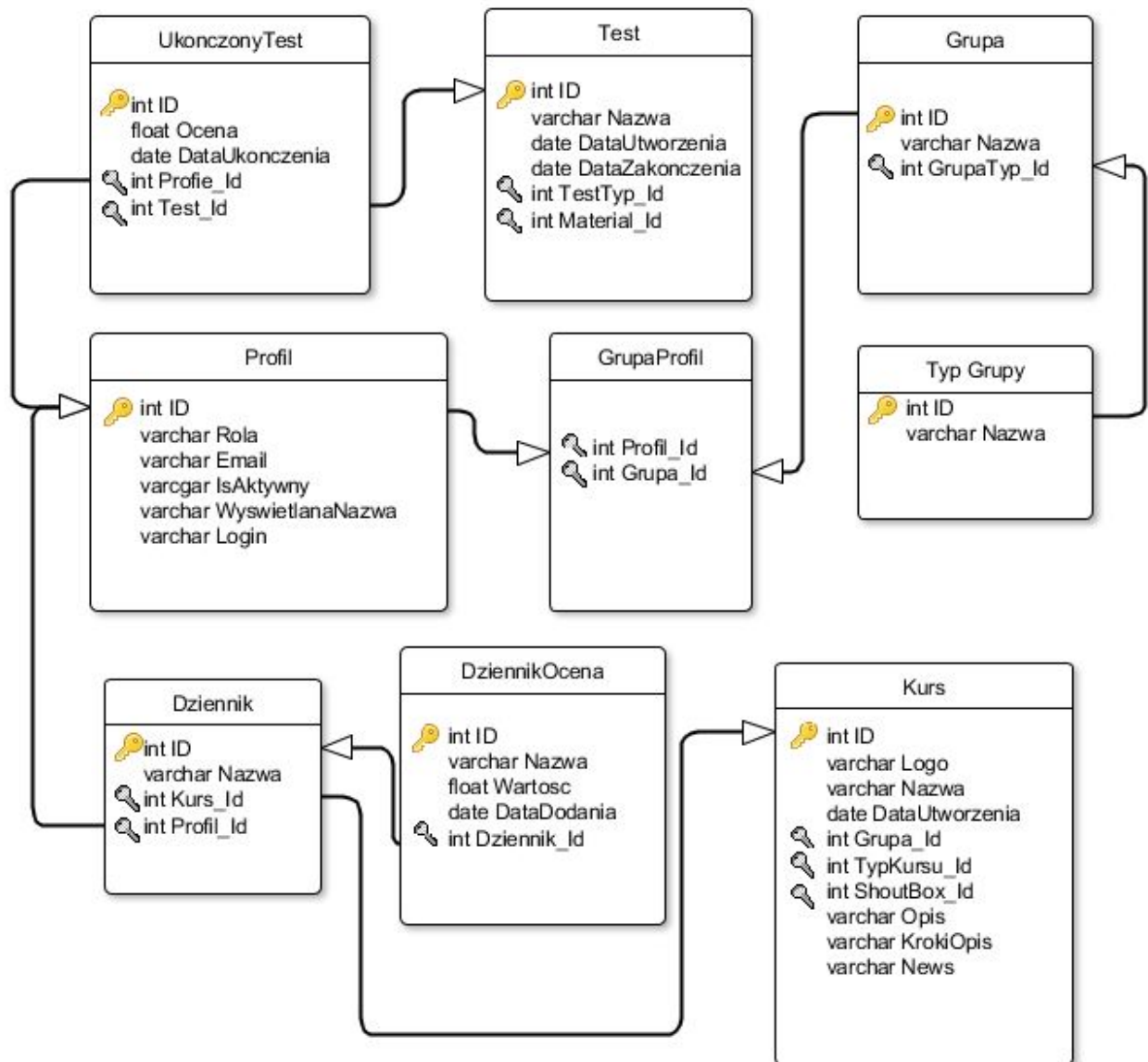


Rysunek 4.2 Diagram CDM cz. 2

Rysunek 4.2 przedstawia uproszczony diagram **CDM** opisujący związki encji w obrębie Kursu. Zgodnie z wymaganiami funkcjonalnymi każdy kurs posiada, ShoutBox, zbiór Testów, oraz zbiór Materiałów Nauczania. By zapewnić mechanizm filtrowania Kurs jest opisany m.in przez słownik określający Typ Kursu. Test składa się z Pytań, które z kolei posiadają odpowiednie odpowiedzi. Jest on ściśle powiązany z Materiałem Nauczania. W skład materiału nauczania wchodzi Sekcje zawierające informacje dydaktyczne.

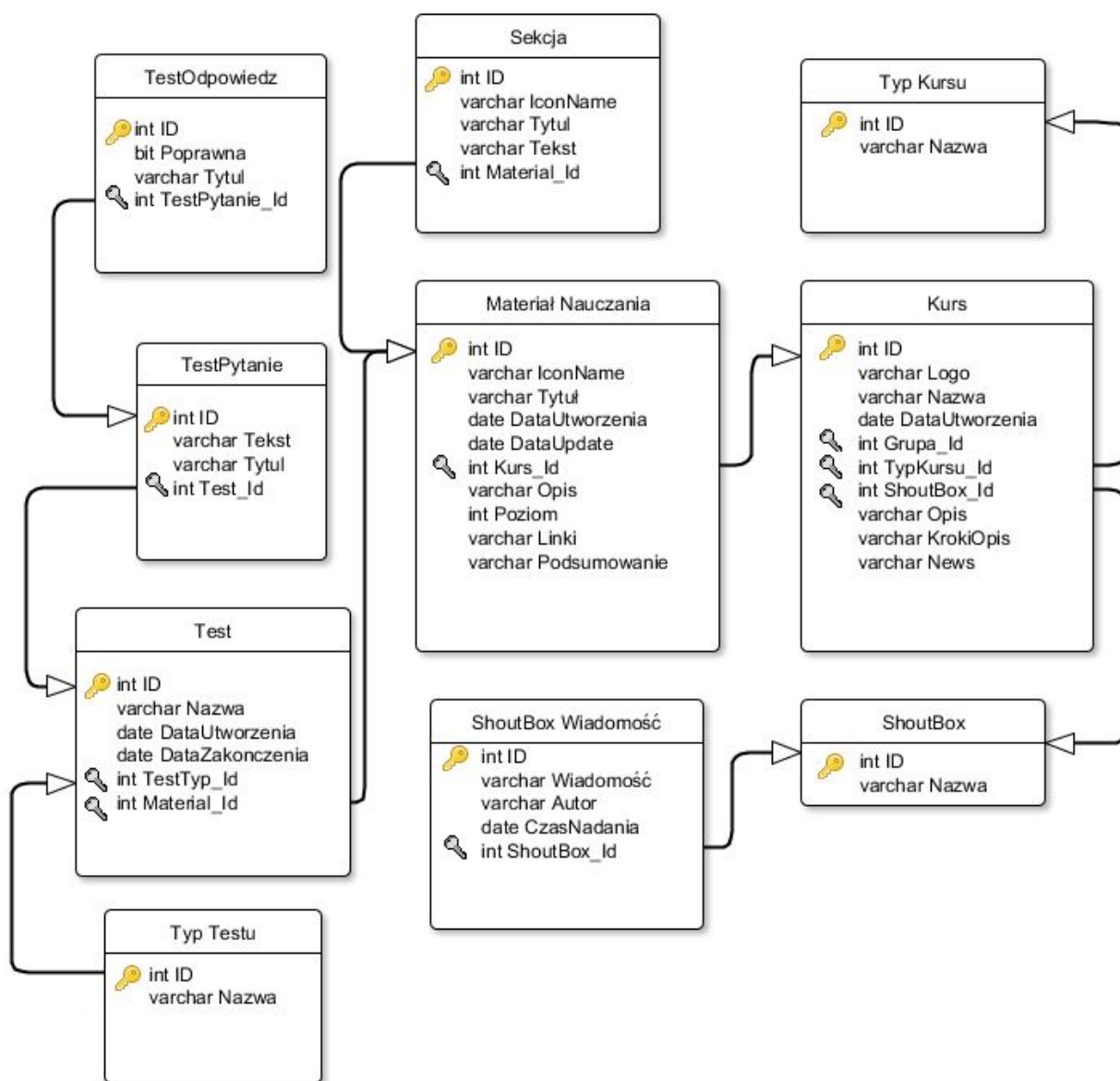
4.2.3 Model fizyczny bazy danych PDM

Diagramy zaprezentowane poniżej prezentują uproszczony model fizyczny bazy danych prezentujący m.in. tablice pomocnicze, potrzebne przy realizacji związków typu "Wiele do Wiele", oraz klucze obce i główne realizujące związki encji, przedstawione w modelu logicznym.



Rysunek 4.3 Diagram PDM cz. 1

Rysunek 4.3 przedstawia pierwszą część modelu fizycznego bazy danych powstałego z modelu logicznego. By zrealizować relację "wiele do wielu" pomiędzy encjami **Grupa** oraz **Profil**, wprowadzono pomocniczą tabelę **GrupaProfil**, zawierającą klucze obu encji. Realizację związku "wiele do jednego", można zaobserwować w przypadku związku pomiędzy encjami **Grupa** oraz **TypGrupy**. Encja **Grupa** posiada klucz obcy skierowany na encję **TypGrupy**.

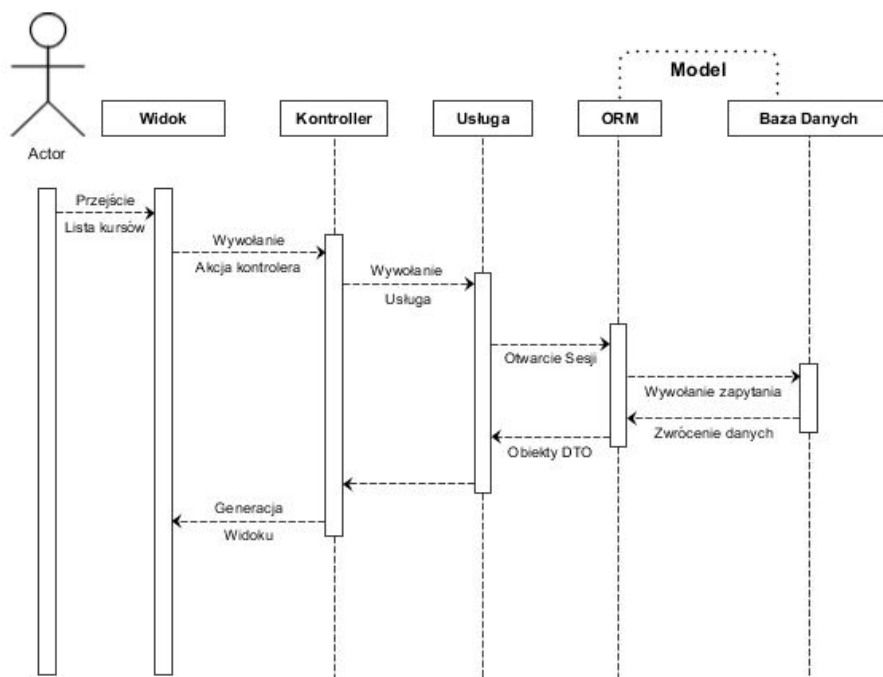


Rysunek 4.4 Diagram PDM cz. 2

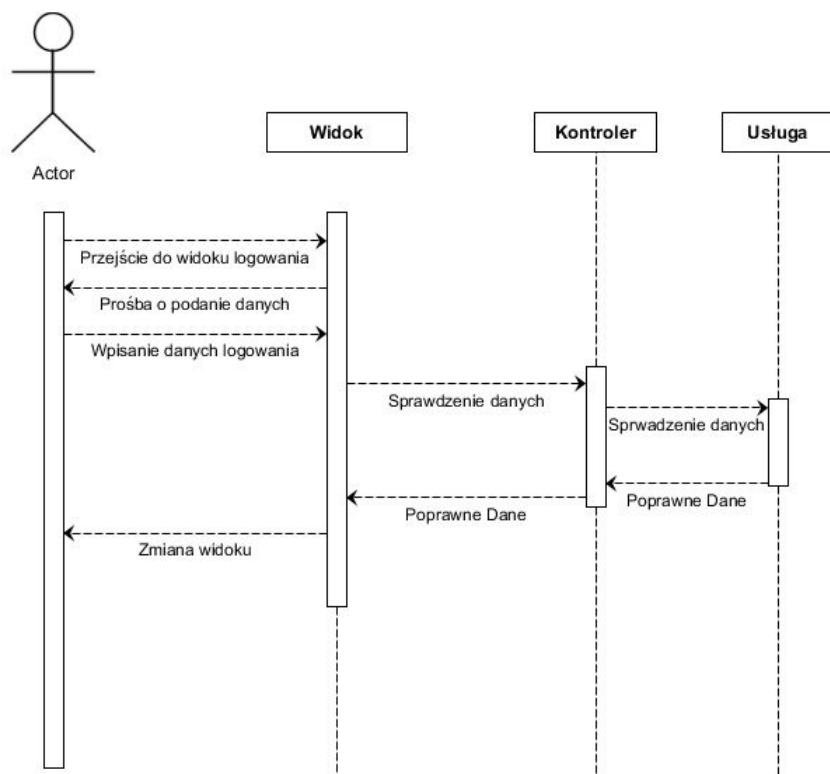
Rysunek 4.4 przedstawia drugą część modelu fizycznego bazy danych powstałego z modelu logicznego. Wprowadzono klucze obce podobnie jak w przypadku pierwszej części diagramu. Wprowadzono między innymi klucze obce realizujące związek jeden do wielu w przypadku powiązania pomiędzy encją "ShoutBoxWiadomosc" oraz "ShoutBox". "Wiadomość" zawiera klucz obcy powiązany z "ShoutBoxem".

4.3.1 Diagramy sekwencji wybranych funkcjonalności

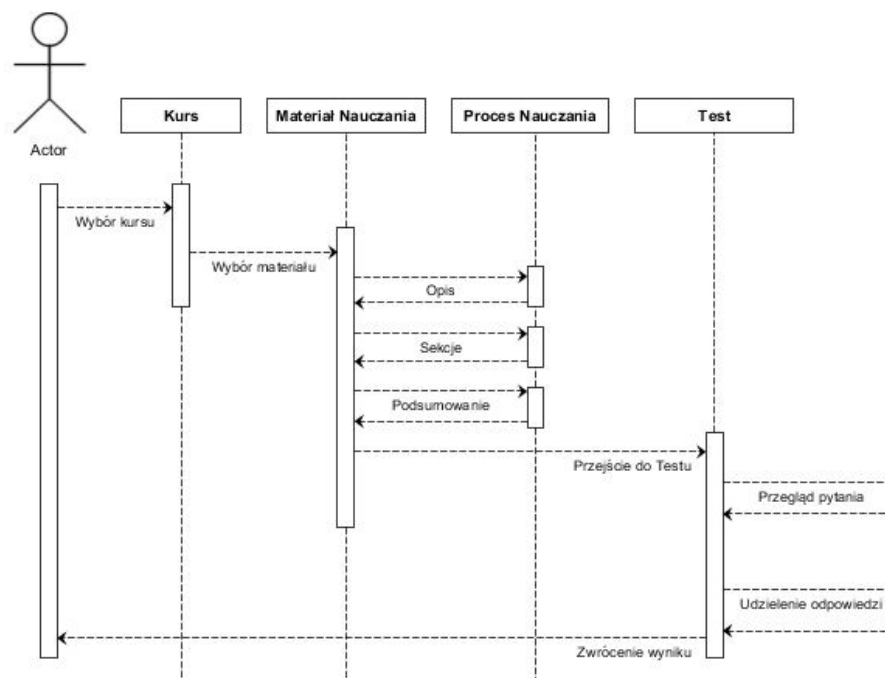
Na kolejnych rysunkach pokazano diagramy sekwencji ilustrujące sposób działania wybranych operacji systemu zdalnego nauczania.



Rysunek 4.6 Diagram sekwencji generowania strony WWW



Rysunek 4.7 Diagram sekwencji logowania do aplikacji



Rysunek 4.8 Diagram sekwencji procesu nauczania

4.3.2 Mechanizmy zabezpieczeń

Wiadomości przesyłane w formacie **JSON** wystawione w formie protokołu **REST**, do których ma dostęp użytkownik korzystający z API, nie są w ogóle zabezpieczone. Są to dane ogólnie dostępne. Jedyną formą zabezpieczenia **API** będzie wystawienie funkcji i dostępu do bazy danych tylko dla wybranych danych. Ograniczenie dostępu zostanie zrealizowane poprzez odpowiednie parametry funkcji. Tzn pobierając np listę kursów klient nie będzie mógł jedynie podać **ID** użytkownika, którego kursy ma wyświetlić.

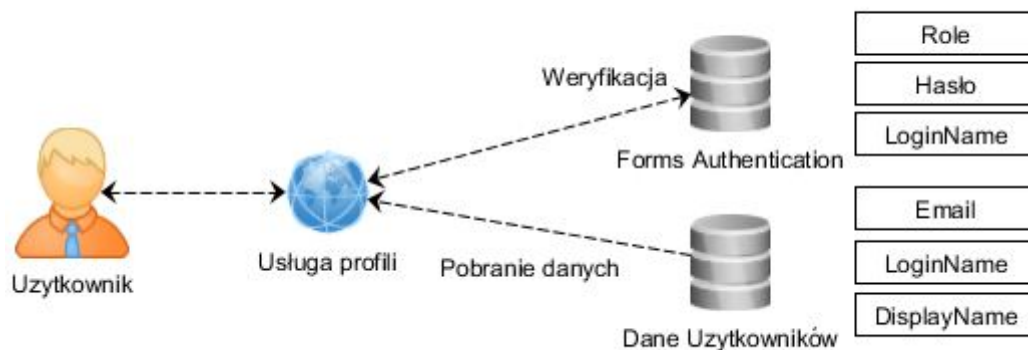
Wiadomości przesyłane protokołem **SOAP**, z wykorzystaniem formatu danych **XML**, będą już miały większy zakres zastosowanych zabezpieczeń. Wiadomości zawierające dane poufne, jak dane: logowania, hasła, dane na temat użytkownika będą szyfrowane oraz przesyłane protokołem **HTTPS** z użyciem portu **443**. W ten sposób najpierw zostanie sprawdzona poprawność certyfikatów a dopiero po tym nastąpi prawdziwa komunikacja zgodnie z protokołem **HTTP**, co zapewni dodatkowy poziom bezpieczeństwa.

Dane bez poufnej zawartości, jak dane kursów lub materiałów nauczania, nie będą zabezpieczone.

Aplikacja odporna jest na ataki skryptowe. Nie ma możliwości wstrzyknięcia skryptu do aplikacji, wywołując tym samym działania niepożądane. Każde pole tekstowe analizowane jest przez serwer pod kątem zawartości.

Dzięki zastosowaniu Mappera obiektowo relacyjnego (**ORM-NHibernate**) warstwa dostępu do danych jest oddzielona od użytkownika i mocno typowaną warstwą, która znacznie ogranicza możliwość wykonania ataku **SQL Injection** [22]. Dodatkowo wprowadzane dane są sprawdzane pod względem możliwych ataków.

Ponieważ dostęp do aplikacji realizowany jest w sposób przypominający działanie protokołu **REST**, tzn. akcje są dostępne z poziomu adresu **URL**, wykonywane operacje są opatrzone dodatkowymi zabezpieczeniami, które sprawdzają poziom uprawnień użytkownika. Dzięki temu anonimowy użytkownik po próbie wywołania takiej niedozwolonej akcji zostanie przekierowany do panelu logowania, natomiast zwykły użytkownik otrzyma informację, że nie posiada odpowiedniego poziomu uprawnień.



Rysunek 4.9 Rozdzielenie mechanizmu logowania na dwie bazy danych

Mechanizm logowania do systemu oparty jest na bazie frameworka Microsoft Forms Authentication [18] zapewniającego podstawowe funkcjonalności zarządzania użytkownikami. Forms authentication wymaga oddzielnej bazy danych. Usługa profili wiąże ze sobą profile w głównej bazie danych oraz profile z bazy związanej z Forms Authentication.

Podział związany z mechanizmem uwierzytelniania użytkowników został wprowadzony po to by nie mieszać obu baz. Baza powiązana z mechanizmem **Forms Authentication** firmy **Microsoft** jest przystosowana do trzymania danych użytkowników oraz spełnia wytyczne firmy **Microsoft**, jest to ich produkt. Dodatkowo bazę profili można umieścić w pliku, zwiększając dzięki temu bezpieczeństwo systemu oraz zmniejszając użycie zasobów. Przy hostingu na, którym wystawiłem aplikację jest określony limit ilości obsługiwanych baz danych **MSSQL 2008**, więc by zmniejszyć ilość baz mogę przenieść bez problemowo tę bazę do pliku.

Rozdział 5

Implementacja systemu zdalnego nauczania

5.1 Zewnętrzny hosting

Na serwerze zainstalowano:

- dwie bazy danych; jedna zawierająca dane logowania, wspierająca framework **AspNet Membership** provider oraz druga baza danych, zawierająca dane aplikacji, czyli m.in. dane o kursach, materiałach nauczania.
- dwie aplikacje; aplikacja główna stworzona w oparciu o framework **ASP.Net MVC 3** będąca systemem e-learningowym oraz aplikacja z usługami sieciowymi zrealizowanymi z wykorzystaniem frameworka **WCF**.

Platforma systemowa została skonfigurowana by składować logi zdarzeń oraz błędów aplikacji, w odpowiednich katalogach. Dzięki temu istnieje możliwość szybkiego zdiagnozowania i poprawienia problemów aplikacji.

Moduł testujący usługę **API**, udostępniającą dane po protokole **REST**, został uruchomiony pod adresem <http://www.mfranc.com/codedash-test/>.

Do testów skonfigurowano również lokalny serwer usług sieciowych, będzie on wykorzystany przy testach mechanizmu dynamicznej zmiany serwera w momencie zerwania połączenia.

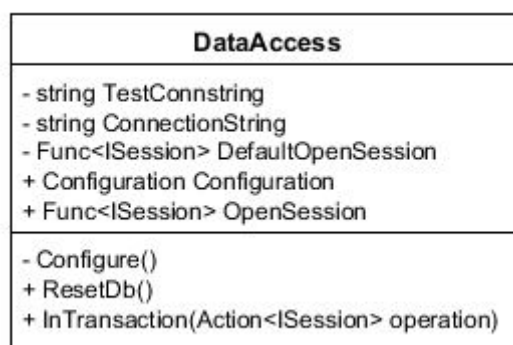
5.2 Realizacja bazy danych

Baza danych została zrealizowana na silniku bazodanowym **MSSQL 2008 R2** [25]. Na serwerze wystawione są dwie bazy danych. Jedna odpowiedzialna za przechowywanie danych o użytkownikach, druga z zawartością danych aplikacji. Dostęp do obu baz realizowany jest za pomocą frameworka **NHibernate**, który jest mapperem relacyjno obiektowym, zapewniającym dostęp do bazy danych z poziomu klas i obiektów. Wystawiono również na lokalnym serwerze zapasową bazę danych. Jest ona używana przez mechanizm dynamicznej zmiany serwera usług sieciowych. Baza ta, o ustalonej godzinie synchronizuje się z bazą główną.

5.3 Realizacja aplikacji

5.3.1 Mechanizmy dostępu do bazy danych

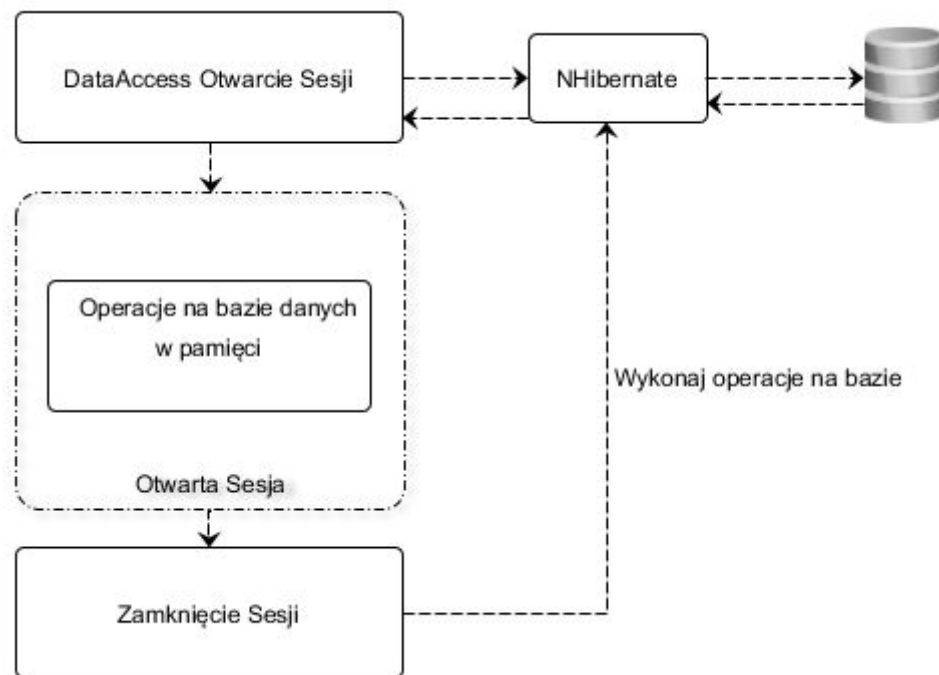
Dostęp do bazy danych opiera się na frameworku **NHibernate**. Cała komunikacja oraz inicjalizacja opakowana jest wewnątrz klasy **DataAccess**. Klasa ta wykorzystywana jest na serwerach udostępniających usługi sieciowe. Jest ona odpowiedzialna przede wszystkim za konfigurowanie połączenia z bazą danych poprzez mapper obiektowo relacyjny **NHibernate**. Ponieważ testy jednostkowe, pokrywające bazę danych, wykorzystują bazę danych generowaną w pamięci, istnieje możliwość wstrzyknięcia odpowiedniej konfiguracji zmieniającej źródło danych na pamięć serwera testowego.



Rysunek 5.1 Diagram klasy **DataAccess**

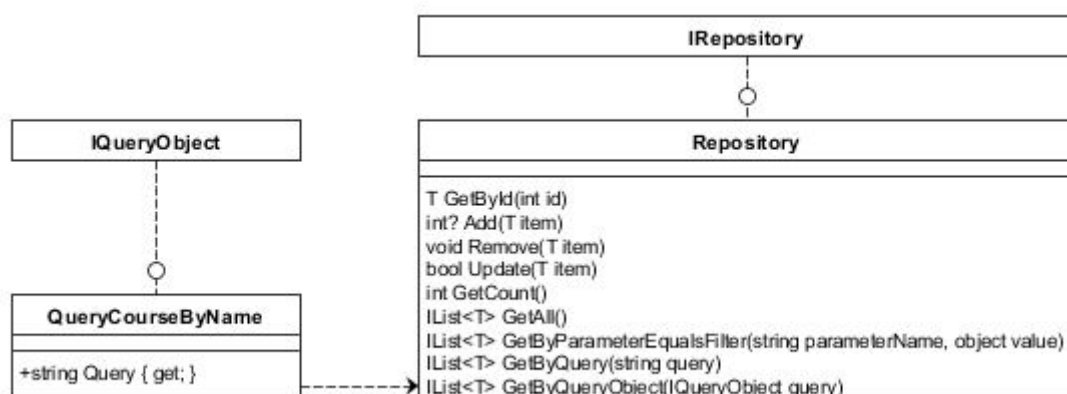
Klasa **DataAccess** (Rysunek 5.1) zbudowana jest na bazie wzorca projektowego **Singleton** [20]. Zawiera ona metody pozwalające zresetować bazę danych, tzn. usunąć wszystkie tabele oraz dane i wygenerować nową strukturę bazy wraz z danymi testowymi. Klasa ta posiada również metodę **InTransaction()**, która pozwala przeprowadzić operację na bazie w obrębie transakcji. Metoda ta jako parametr przyjmuje wskaźnik do funkcji opakowany w specjalnej klasie dostępnej na platformie **.Net**. Język **Csharp** nie posiada jawnego wsparcia dla wskaźników (istnieje możliwość uruchomienia wsparcia kosztem wyłączenia pewnego segmentu kodu z automatycznego zarządzania stertą poprzez mechanizm "garbage collector") dlatego stosuje się klasy opakowywujące. Istnieje również możliwość przesłania funkcji anonimowej podobnie jak w języku programowania **Javascript**. Podobny mechanizm został zastosowany w parametrze **OpenSession**, który również zwraca opakowany wskaźnik do funkcji. Dzięki takiemu rozwiązaniu istnieje możliwość podmienienia logiki otwarcia sesji. Szczególnie jest to przydatne w przypadku implementacji testów jednostkowych, operujących na bazie danych w pamięci.

Do otwarcia połączenia z serwerem wymagany jest ciąg znaków zwany "connection stringiem". Ciąg ten zawiera adres serwera bazodanowego, nazwę bazy danych oraz informacje wymagane w procesie logowania do serwera. Domyślnie zdefiniowany jest ciąg przekierowujący na testową bazę danych. By dostarczyć inny ciąg należy zmodyfikować odpowiednio plik konfiguracyjny aplikacji webowej ("web.config").



Rysunek 5.2 Mechanizm dostępu do bazy przy użyciu klasy DataAccess

Rozpoczęcie operacji na bazie danych wymaga otwarcia sesji. Operację tę realizuje się poprzez wykonanie metody **OpenSession()**. Funkcja ta zwraca wskaźnik do funkcji, która zwraca klasę sesji frameworka **NHibernate**. Obiekt sesji wymagany jest do przeprowadzania wszystkich operacji na bazie danych. Klasa ta jest implementacją wzorca projektowego **Unit Of Work**, który jest bardzo wygodnym sposobem zarządzania takim zasobem jak sesja (Rysunek 5.2).



Rysunek 5.3 Klasa Repository oraz QueryObject

Korzystanie bezpośrednio z obiektu sesji jest bardzo wygodnym rozwiązaniem. Jednakże w przypadku bazy danych wystawionej za warstwą usług sieciowych zarządzanie sesją dostępu do bazy danych jest bardzo skomplikowanym zagadnieniem. Po próbach przesyłania obiektu sesji, m.in. przy wykorzystaniu rozwiązania **NHibernate Remoting**, zdecydowano się opakować parametry sesji w klasach, będących implementacją wzorca

projektowego **Repozytorium**.

Operacje wykonywane na bazie danych opakowane są w formie generycznej klasy **Repository<T>**. Dostarcza ona podstawowych metod pozwalających realizować operacje z zakresu **CRUD** (Tworzenie, Czytanie, Modyfikacja, Usuwanie). Wykonywanie bardziej skomplikowanych zadań można zrealizować za pomocą metody **GetByQuery**, bądź **GetByQueryObject**. Pierwsza metoda przyjmuje jako parametr ciąg znaków, będący zapytaniem języka **HQL** [16]. Druga metoda przyjmuje klasę implementującą interfejs **IQueryObject**. Jest to specjalny interfejs pozwalający zdefiniować obiekty opakowujące zapytania języka **HQL** w ściśle typowanych klasach. Dzięki takiemu zabiegowi programista nie operuje bezpośrednio na znakach, ale na klasach. Rozwiązanie to pozwala stworzyć kod lepszej jakości. Jest to luźna implementacja wzorca **Strategia** [20]

```
public class QueryCourseByName : IQueryObject
{
    private readonly string _value;
    public QueryCourseByName(string value)
    {
        _value = value;
    }

    public string Query
    {
        get { return String.Format
            ("from CourseModel c where c.Name = '{0}'", _value); }
    }
}
```

Rysunek 5.4 Klasa QueryCourseByName opakowująca zapytanie języka HQL

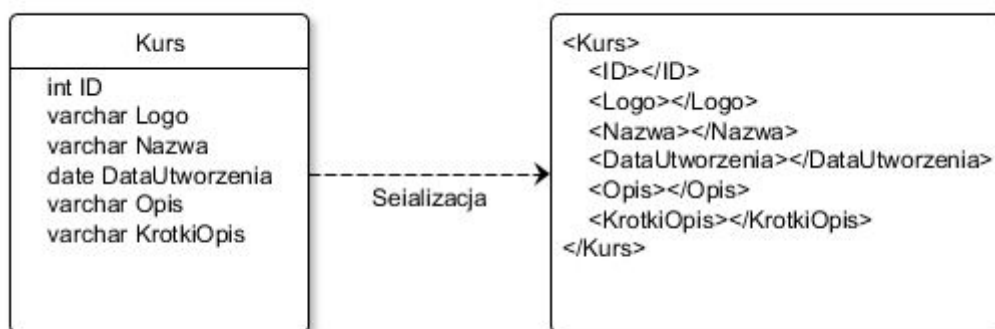
Język **HQL** [16] jest językiem przypominającym składnią standardowy język zapytań **SQL**. Wprowadza on trochę uproszczeń, ale zarazem usprawnia działanie zapytań poprzez mechanizmy pozwalające operować na obiektach i ich kolekcjach. **SQL** operuje na tabelach i modelu relacyjnym, natomiast **HQL** opiera swoje operacje na obiektach i ich kolekcjach zorientowanych obiektowo. Przykład powyżej (Rysunek 5.4) pozwala pobrać encje kursu na podstawie jego nazwy. Jest to zapytanie podobne do prostego zapytania typu **SELECT** z klauzulą **WHERE** w przypadku języka **SQL**.

```
select lm.ID from LearningMaterialModel as lm ,
TestModel as test where test.ID = {0} and test
in elements(lm.Tests)
```

Rysunek 5.5 Skomplikowane zapytanie HQL

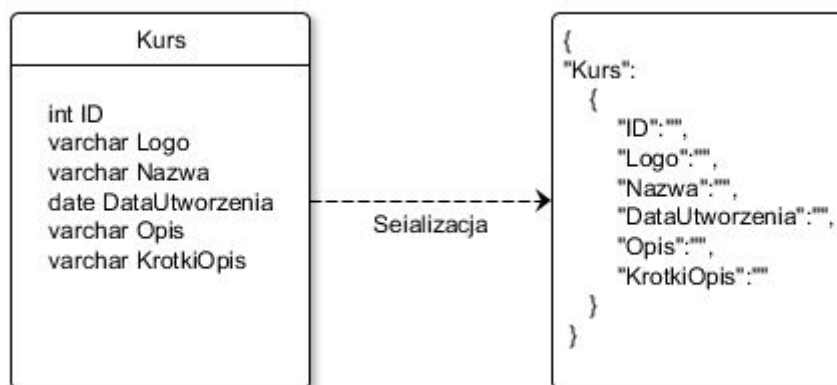
HQL pozwala również przeprowadzać bardziej skomplikowane operacje. Kod zaprezentowany na rysunku 5.5 pozwala znaleźć numer ID rodzica podanego obiektu. W tym przypadku wykorzystywana jest funkcja *in elements*, która pozwala przeszukiwać kolekcję elementów.

5.3.2 Mechanizmy przetwarzania danych



Rysunek 5.6 Przykład przekształcenia do formatu XML

Przed wysłaniem danych pobranych z bazy danych do aplikacji muszą być one przetworzone do odpowiedniego formatu. W przypadku usług sieciowych wystawionych na protokole **SOAP**, korzystających z formatu **XML**, wykorzystywany jest mechanizm *serializowania* wszystkich publicznych parametrów klasy. Element główny tworzony jest na podstawie nazwy klasy. Elementy należące do elementu głównego, *zwane jego dziećmi*, zawierają odpowiednio nazwę parametru oraz jego wartość. W projekcie użyto standardowego serializera dostępnego wraz z platformą **.NET**. Konfiguracja serializacji sprowadza się do oznaczenia specjalnymi atrybutami parametrów danej klasy.

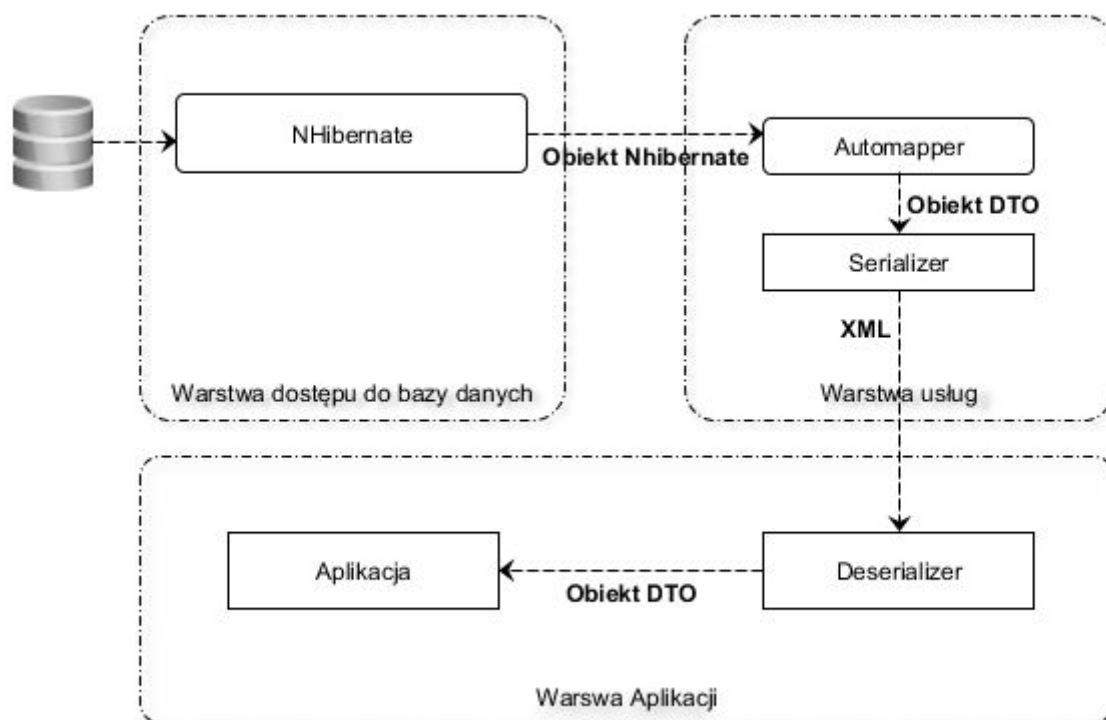


Rysunek 5.7 Przykład przekształcenia do formatu JSON

Dane wystawione na zewnątrz aplikacji przy użyciu protokołu **REST**, przetwarzane są do formatu danych **JSON**. Jest to lekki i bardzo czytelny format (Rysunek 5.7).

Przed przetworzeniem danych do odpowiedniego formatu, wykonywany jest proces *odchudzenia* klas tzn. przetwarzania klasy ogólnej na klasę szczególną, zawierającą tylko wymagane parametry oraz dane. Proces ten jest bardzo ważny w przypadku dostępu do bazy danych realizowanego za pomocą mappera obiektowo relacyjnego takego jak **NHibernate**. Framework ten pobierane dane opakuje w specjalne klasy implementujące

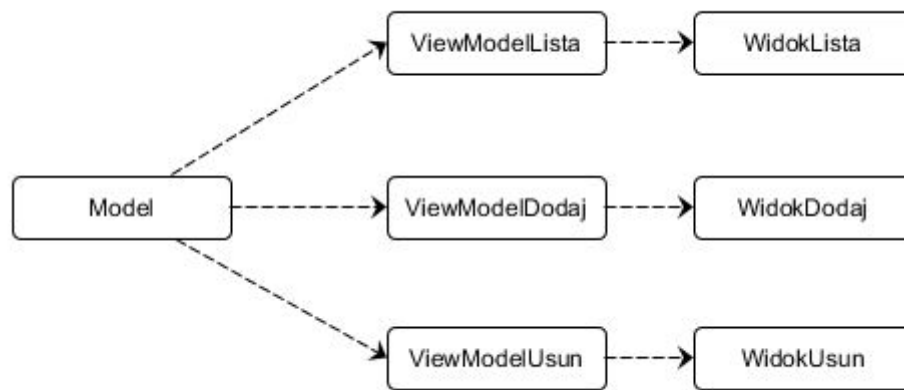
wzorzec projektowy "**Klasa proxy**". Wzorzec ten rozszerza możliwości klasy, dodając dodatkowe funkcjonalności, np. możliwość przeprowadzenia późnej inicjalizacji. Klasa generowana przez NHibernate zawiera wiele dodatkowych metod oraz pól. Ponadto wszelkie kolekcje dostępne na platformie .NET. przekształcane są w specjalne kontenery, widoczne jedynie z poziomu aplikacji mającej dostęp do biblioteki NHibernate. Zgodnie z założeniami jedynie serwer ma mieć powiązanie z tym frameworkiem. Zadaniem klienta jest jedynie wywoływanie metod wykorzystujących podstawowe obiekty frameworka .NET. Klient nie może zależeć od implementacji warstwy bazodanowej.



Rysunek 5.8 Graficzne przedstawienie procesu przetwarzania danych

Proces przetwarzania obiektów generowanych przez framework NHibernate na obiekty przesyłane po sieci, tzw. **DTO - Data Transport Object**, realizowany jest przy pomocy biblioteki **AutoMapper**. Dostarcza ona funkcjonalności usprawniających żmudny proces przetwarzania jednego typu danych w drugi. By móc korzystać z **Automappera** na początku konfiguruje się mapowania (podobnie jak w przypadku mapowań powiązanych z frameworkiem **NHibernate**).

W aplikacji każdy model reprezentujący encję bazodanową posiada odpowiednią klasę **DTO**. Klasy **DTO** dziedziczą po jednej, wspólnej generycznej klasie bazowej, dostarczającej metody pozwalające w łatwy sposób wywoływać metody dostarczane przez framework **AutoMapper**, przekształcające jedną klasę w drugą.



Rysunek 5.9 Przekształcanie obiektów typu ViewModel

Niemniej istotnym czynnikiem przemawiającym za *odchudzaniem* klas jest ilość danych przesyłana po sieci. Przesyłanie całych obiektów z danymi, z których nie będziemy korzystać byłoby bardzo niewydajnym i kosztownym rozwiązaniem. Rozważmy przypadek wyświetlania listy kursów. Kurs jest rozbudowanym obiektem posiadającym wiele parametrów nie tylko opisujących kurs, ale również zawierających kolekcje Materiałów Nauczania, Grupę itd. Do wyświetlenia listy kursów potrzebujemy jedynie jego parametrów opisowych. W takim przypadku przesyłanie całej zawartości kursu byłoby mało wydajne. By ograniczyć ilość przesyłanych danych stosuje się tzw **ViewModele - Modele Widoku**. Są to klasy specjalnie dopasowane do widoku, w którym zostaną wyświetlone. View Modele mogą dodatkowo łączyć wiele różnych encji.

5.3.3 Realizacja protokołu komunikacji

Komunikacja pomiędzy aplikacją udostępniającą usługi sieciowe, a bazą danych oparta jest na rozwiązaniu **WCF (Windows Communication Foundation)**. Jest to najnowszy framework firmy Microsoft pozwalający tworzyć aplikacje wykorzystujące usługi sieciowe. Dzięki zastosowaniu tej technologii można tworzyć bardzo łatwo konfigurowalne mechanizmy przetwarzania danych przy wykorzystaniu usług sieciowych. **WCF** nie tylko jest nowym frameworkiem, ale również pozwala korzystać z wcześniejszych rozwiązań dostępnych na platformie .NET : ASMX, COM+, .Net Remoting.

Konfiguracja frameworka **WCF**, w celu realizacji usług sieciowych, odbywa się poprzez zdefiniowanie odpowiednich bindigów oraz endpointów.

```

<service behaviorConfiguration="DefaultELearnServices" name="ELearnServices.CourseService">
  <endpoint name="jsonEP" address="json" binding="webHttpBinding" bindingConfiguration="NoneSecurity"
  <endpoint address="" binding="basicHttpBinding" contract="ELearnServices.ICourseService" />
  <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
  <host>
    <baseAddresses>
      <add baseAddress="http://codedashservices.mfranc.com/" />
    </baseAddresses>
  </host>
</service>

```

Rysunek 5.10 Ustawienia usługi po stronie serwera

Usługa po stronie serwera zdefiniowana jest za pomocą endpointów. Są to węzły pozwalające konfigurować sposób, adres oraz protokół pod jakim będą dostępne usługi sieciowe. W zaprezentowanym przykładzie zdefiniowane są trzy endpointy.

Pierwszy, nazwany JsonP, jest endpointem wystawiającym daną usługę po protokole REST w formacie JSON. Konfiguracja tego endpointa ustawia brak zabezpieczeń, tzn dane nie są szyfrowane. Jako binding używany jest **webHttpBinding**. Jest to binding wystawiający usługę sieciową, dostępną z poziomu zwykłych zapytań protokołu HTTP, bez opakowywania żądań w ramki SOAP.

Drugi endpoint definiuje dostęp z poziomu protokołu SOAP. Mechanizm ten realizowany jest poprzez binding **basicHttpBinding**.

Trzecim endpointem jest wystawienie katalogu usług w postaci wiadomości WS-MEX (WS-MetadataExchange). Jest to endpoint, do którego zwraca się klient w momencie gdy chce zbadać jakie usługi są dostępne na danym serwerze.

Po stronie klienta definiujemy własny binding i ustawiamy odpowiedni endpoint.

```
<binding name="WSHttpBinding_ICourseService" closeTimeout="00:01:00"
  openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
  allowCookies="false" bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard"
  maxBufferSize="65536" maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
  messageEncoding="Text" textEncoding="utf-8" transferMode="Buffered"
  useDefaultWebProxy="true">
  <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
    maxBytesPerRead="4096" maxNameTableCharCount="16384" />
  <security mode="None">
    <transport clientCredentialType="None" proxyCredentialType="None"
      realm="" />
    <message clientCredentialType="UserName" algorithmSuite="Default" />
  </security>
</binding>
```

Rysunek 5.11 Ustawienia bindingu po stronie klienta

Binding po stronie klienta pozwala zdefiniować szereg opcji, konfigurujących wygląd żądań wysyłanych do serwera z usługami sieciowymi.

```
<endpoint address="http://codedashservices.mfranc.com/CourseService.svc" binding="basicHttpBinding"
  bindingConfiguration="WSHttpBinding_ICourseService" contract="CourseService.ICourseService"
  name="WSHttpBinding_ICourseService" />
```

Rysunek 5.12 Ustawienia endpointu po stronie klienta

Najważniejszymi częściami definicji endpointu po stronie klienta jest wskazanie adresu oraz bindingu, który określa w jakiej postaci oczekujemy danych.

Przy komunikacji SOAP, która w tym przypadku jest ściśle powiązana typami, serializer po stronie serwera i deserializer po stronie klienta operują na tych samych typach i muszą być zgodne w wersji. Klasami tymi są obiekty DTO. Takie rozwiązanie wprowadza pewne ograniczenia, ale za to usprawnia proces tworzenia systemu. Obiekty DTO są

prostymi klasami kontenerowymi, służącymi jedynie do transferu danych, a więc uznano, że takie rozwiązanie będzie przynosiło więcej plusów niż minusów.

```
$.ajax({
  type:"GET",
  url:"http://codedashservices.mfranc.com/CourseService.svc/json/Get",
  data : "id=16",
  datatype:'json',
  success:function(data){
    $.each(data , function(index,item){
      //Przetwarzanie danych
    });
  },
  error : function(data){
    alert('error');
  },
  statusCode: {
    404: function() {
      alert('page not found');
    }
  }
})
```

Rysunek 5.13 Ustawienia endpointu po stronie klienta korzystającego z API

W przypadku danych wystawionych przez protokół REST, dostępnych dla innych klientów, sytuacja jest trochę inna. Klient nie posiada informacji jak deserializować dane obiektów. Nie musi posiadać biblioteki z obiektami DTO. Klient nie potrzebuje specjalnej konfiguracji i nie potrzebuje być nawet powiązany z technologią .Net. W tym przypadku dane otrzymywane przez klienta przekazywane są w czystej, nie przetworzonej postaci. Jedyne co jest potrzebne do konfiguracji to adres serwera z usługami sieciowymi. W zaprezentowanym kodzie wykorzystywana jest funkcja popularnej biblioteki **JQuery** wspomagającej funkcjonalności języka **JavaScript**.

5.4 Wykorzystane narzędzia

5.4.1 Mechanizm logowania zdarzeń - NLog

W procesie wytwarzania oprogramowania bardzo ważnym aspektem, zwiększającym znacząco powodzenie projektu, jest zapewnienie odpowiedniego mechanizmu logowania zdarzeń oraz błędów występujących wewnątrz aplikacji. Dzięki zaimplementowaniu takiego mechanizmu, przyspieszamy proces naprawy oraz identyfikacji błędów. Dodatkowo możemy analizować poprawność działania aplikacji poprzez analizę wiadomości zwracanych przez system. Przyczynia się to znacząco do zmniejszenia kosztów utrzymania oraz wprowadzania zmian i poprawek w aplikacji.

W niniejszej pracy do stworzenia mechanizmu logowania zastosowano popularną, darmową bibliotekę **NLog**, stworzoną przez Jarosława Kowalskiego. Jest to stosunkowo prosta, a zarazem rozbudowana biblioteka, pozwalająca tworzyć mechanizmy logowania.

By "podczepić" mechanizm logowania do danej klasy wystarczy stworzyć wewnątrz niej element statyczny, będący instancją klasy **NLog.Logger**. Dla każdej klasy, którą chcemy objąć mechanizmem logowania, tworzymy oddzielną instancję loggera.

```
private static NLog.Logger logger = NLog.LogManager.GetCurrentClassLogger();
```

Po zainicjalizowaniu klasy Loggera wystarczy, że w interesującym nas miejscu wywołamy określoną metodę reprezentującą rodzaj logowanej wiadomości.

```
logger.Debug(Starting Add Mark [Get] with : journalId 0",id);
```

Nlog udostępnia m.in wiadomości typu:

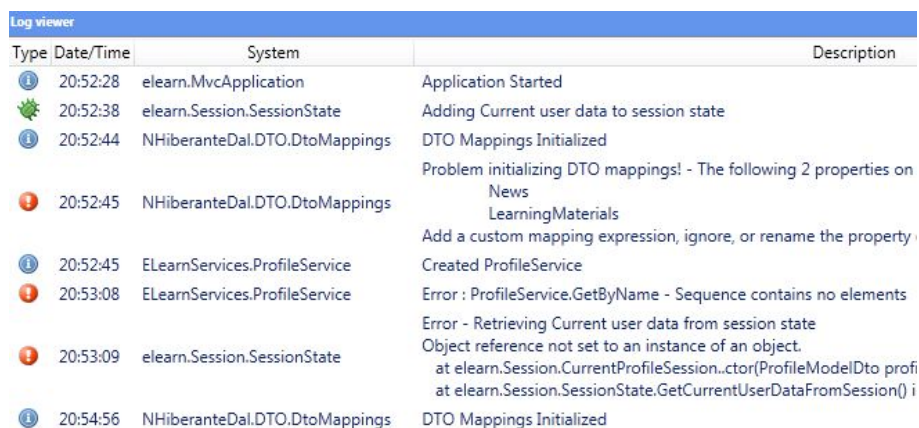
- Informacja,
- Debugowanie,
- Błąd,
- Błąd Krytyczny.

Prócz wywołania NLoga bardzo istotną rzeczą jest odpowiednie skonfigurowanie pliku konfiguracyjnego aplikacji. W pliku konfiguracyjnym możemy bowiem ustalić miejsce, do którego chcemy zapisywać logi. Do wyboru mamy m.in:

- Plik,
- Adres E-mail,
- Wysłanie danych po porcie,
- Konsola,
- Baza Danych.

Dla każdego źródła istnieje możliwość skonfigurowania rodzaju informacji jakie ma zapisywać.

W aplikacji zastosowano mechanizm rzucania logów informacyjnych związanych z procesem logowania do pliku oraz wysyłanie protokołem UDP na port 9999. Na tym porcie lokalnie nasłuchuje darmowa aplikacja **Sentinel** [8], która pozwala analizować wysyłane logi. Dzięki takiej konfiguracji przyspieszył się proces implementacji ponieważ był wgląd w proces wykonywania akcji wewnątrz aplikacji.



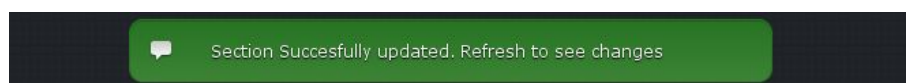
Type	Date/Time	System	Description
Information	20:52:28	elearn.MvcApplication	Application Started
Information	20:52:38	elearn.Session.SessionState	Adding Current user data to session state
Information	20:52:44	NHiberanteDal.DTO.DtoMappings	DTO Mappings Initialized
Warning	20:52:45	NHiberanteDal.DTO.DtoMappings	Problem initializing DTO mappings! - The following 2 properties on News LearningMaterials Add a custom mapping expression, ignore, or rename the property
Information	20:52:45	ELearnServices.ProfileService	Created ProfileService
Error	20:53:08	ELearnServices.ProfileService	Error : ProfileService.GetByName - Sequence contains no elements
Error	20:53:09	elearn.Session.SessionState	Error - Retrieving Current user data from session state Object reference not set to an instance of an object. at elearn.Session.CurrentProfileSession..ctor(ProfileModelDto profi at elearn.Session.SessionState.GetCurrentUserDataFromSession() i
Information	20:54:56	NHiberanteDal.DTO.DtoMappings	DTO Mappings Initialized

Rysunek 5.14 Przykładowy log z programu Sentinel [8]

Dodatkowo by zapewnić szybszy czas reakcji na poprawki, logi oznaczone jako krytyczne skonfigurowane zostały tak by były wysyłane w formie wiadomości email z logiem na określony adres. Dzięki takiemu zabiegowi administrator otrzymuje szybko informację o tym, że dzieje się coś naprawdę ważnego, powodującego błędne działanie aplikacji w stopniu, w którym aplikacja nie może działać stabilnie.

Logami objęte zostały:

- Operacje wykonywane na bazie danych,
- Akcje wywoływane w aplikacji po stronie serwera.



Rysunek 5.15 Przykładowy alert

Prócz generowania logów po stronie serwera w aplikacji zaimplementowany jest mechanizm wyświetlania alertów z informacjami po stronie użytkownika przeglądarki. Zadaniem alertów jest informowanie użytkownika o zachodzących akcjach takich jak np. zapisanie danych do systemu, bądź wystąpienie problemu z połączeniem.

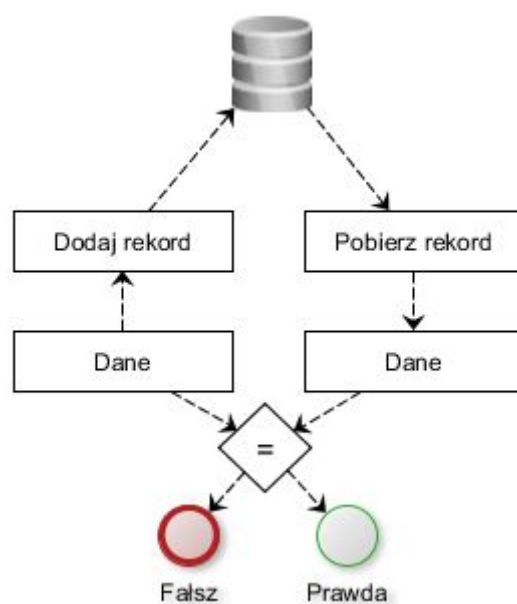
5.4.2 Testy jednostkowe - NUnit

Testy jednostkowe są nowoczesnym narzędziem pozwalającym testować pojedyncze, *jednostkowe* funkcjonalności aplikacji. Do przeprowadzenia testów w aplikacji użyłem frameworka **NUnit**. Testami jednostkowymi została pokryta logika bazodanowa oraz część akcji wywoływanych przez kontrolery.

Testy dostępu do bazy danych

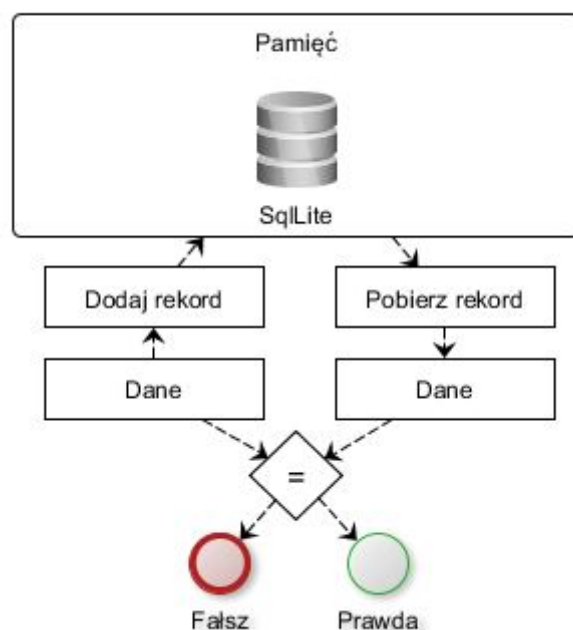
Testowanie jednostkowe bazy danych jest dość skomplikowanym zagadnieniem. Testowanie jednostkowe zakłada wykonywanie testów tylko pojedynczej funkcjonalności. W przypadku bazy danych byłaby to np. operacja dodawania rekordu do bazy. Procedura przeprowadzenia takiego testu polega na:

- Wywołaniu metody dodawania rekordu do bazy danych,
- Wywołaniu metody pobierania rekordu z bazy danych,
- Operacji porównania rekordu pobranego z rekordem dodanym.



Rysunek 5.16 Prosty test jednostkowy bazy danych

By przeprowadzić taki test potrzebujemy dostępu do bazy danych. Dostęp do bazy można zrealizować na kilka sposobów. Mianowicie można przeprowadzać testy na fizycznej, istniejącej bazie danych. W takim przypadku jednak należy pamiętać by rekord testowy po teście usunąć. Innym wyjściem jest stworzenie oddzielnej bazy danych i przeprowadzanie testów na niej. W tym przypadku wystarczy, że za każdym razem odtworzymy pustą, bądź domyślną bazę danych. Testowanie na prawdziwej bazie danych jest czasochłonne. Lepszym rozwiązaniem jest przeprowadzanie testów na bazie danych generowanej w pamięci. W tym przypadku można skorzystać z bazy danych opartej na silniku **SQLite** [9], która jest bardzo popularnym darmowym rozwiązaniem.



Rysunek 5.17 Test jednostkowy z bazą danych w pamięci

Przed każdym rozpoczęciem testu generowana jest baza danych oraz wypełniana danymi potrzebnymi przy testach. W przypadku np. testowania elementu "Kurs", posiadającego listę elementów *Test*, generowany jest kurs z przykładową listą testów. Ponieważ wykorzystywany jest mapper relacyjno obiektowy **NHibernate**, przed rozpoczęciem testów bazy danych przeprowadzany jest test konfiguracji frameworka.

Testy bazy danych obejmują :

- Testy generycznej klasy repozytorium,
- Testy zapytań wykorzystujących język HQL.

Baza danych pokryta jest łącznie ponad 90 testami jednostkowymi.

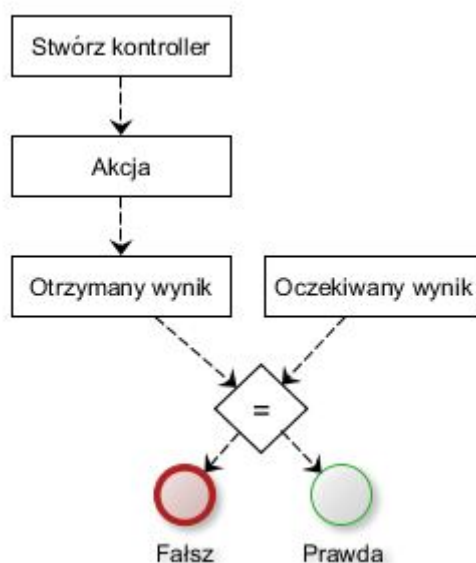
Testy kontrolerów - RhinoMocks

Testy jednostkowe kontrolerów są bardziej złożone niż testy wykonywane na warstwie dostępu do bazy danych. Kontroler bardzo często komunikuje się z warstwą bazodanową wyciągając dane potrzebne do wygenerowania widoku. Baza danych jest ich zewnętrzną zależnością. Testowanie dostępu do bazy danych nie jest sensem testów kontrolera. Test kontrolera musi testować jedynie proces przetwarzania danych pozyskiwanych z bazy danych. W tym przypadku najlepszym rozwiązaniem byłoby w ogóle pominięcie bazy danych i wstrzyknięcie danych testowych, na których przetestujemy zachowanie kontrolera. Na szczęście istnieje taka możliwość i do tego celu stosuje się specjalne obiekty zwane **Mockami/Stubami**. W niniejszej pracy wybrano pierwszy rodzaj obiektów i posłużono się frameworkiem RhinoMocks do ich generacji.

Test kontrolera realizowany jest podobnie jak test bazodanowy.

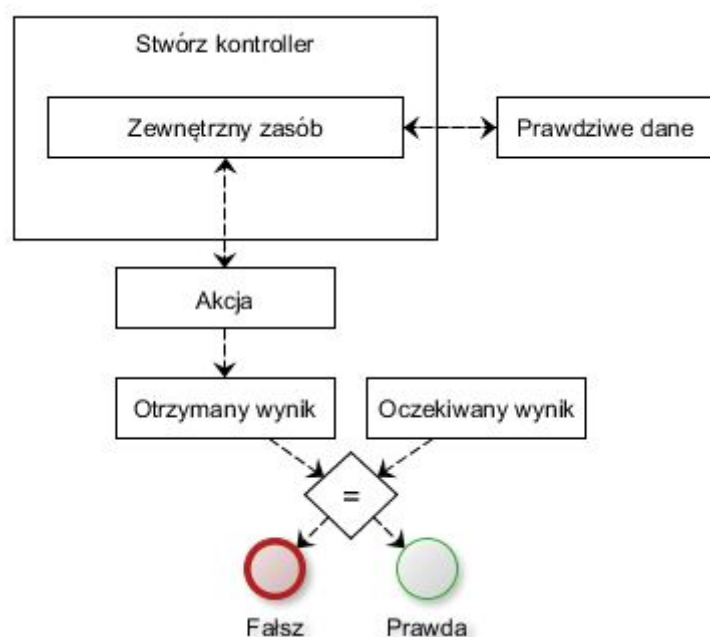
- Tworzymy dany kontroler wywołując jego konstruktor.

- Wywołujemy metodę ("Akcje") na kontrolerze podając odpowiednie parametry.
- Weryfikujemy otrzymany wynik.

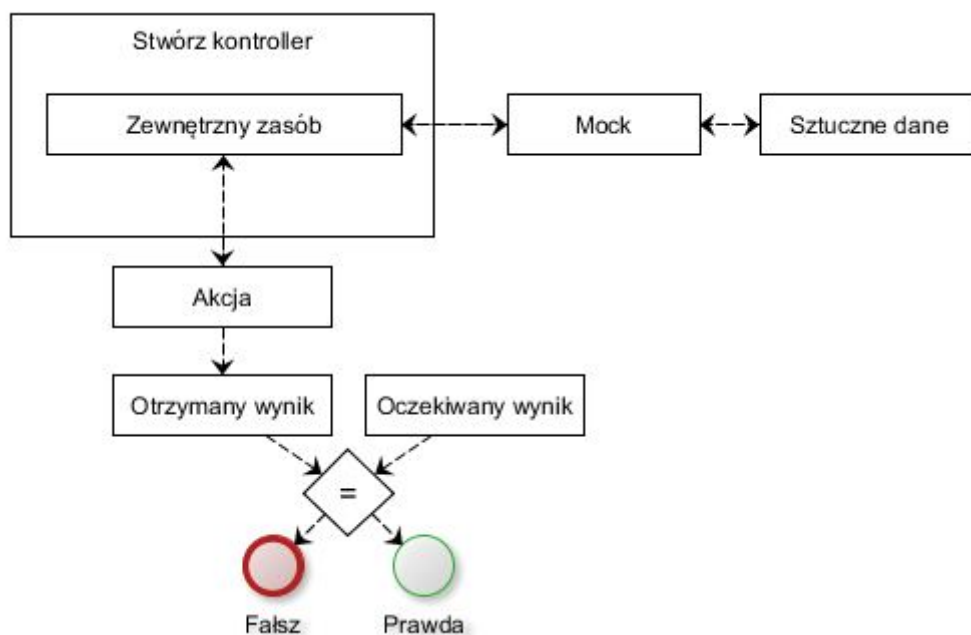


Rysunek 5.18 Test jednostkowy kontrolera bez zewnętrznych zasobów

W przypadku kontrolera korzystającego z zewnętrznych zasobów takich jak baza danych, bądź globalna sesja, przed stworzeniem kontrolera należy zainicjować obiekt typu **MOCK**, imitujący zasób, i wstrzyknąć go do środka kontrolera.



Rysunek 5.19 Test kontrolera z zewnętrznymi zasobami



Rysunek 5.20 Test kontrolera przy wykorzystaniu mocka

```

[Test]
public void Post_if_create_in_db_course_failes_then_return_error_view()
{
    #region Arrange
    using (Mock.Record())
    {
        Expect.Call(CourseService.AddCourse(Course)).IgnoreArguments().Return(null);
    }
    #endregion

    #region Act
    ViewResult view;
    using (Mock.Playback())
    {
        view = (ViewResult)CourseController.Create(Course);
    }
    #endregion

    #region Assert
    Assert.That(view.ViewName, Is.EqualTo("Error"));
    Assert.That(view.ViewBag.Error, Is.EqualTo(elearn.Common.ErrorMessages.Course.AddToDbError));
    #endregion
}

```

Rysunek 5.21 Przykładowy kod testu z mockiem

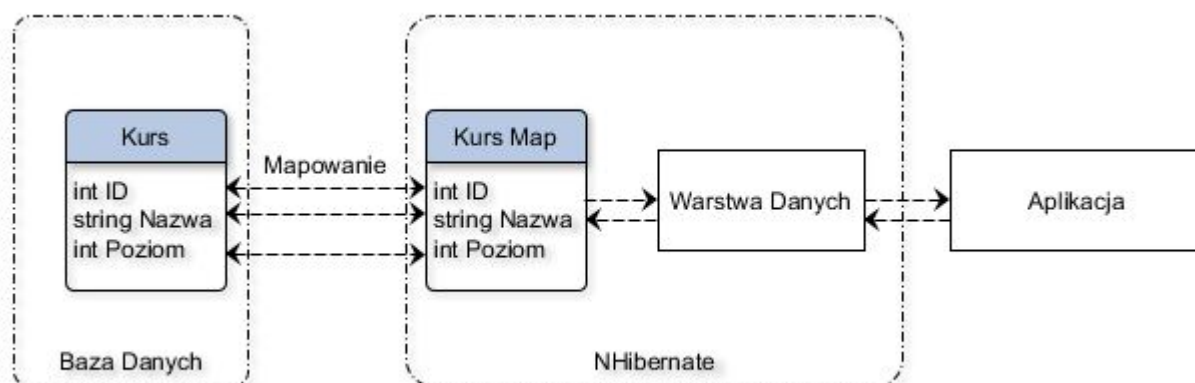
W powyższym kodzie testowany jest kontroler Kursów i jego metoda **Create**. Test weryfikuje przypadek, gdy kontroler kursów pobierając dane z bazy danych otrzyma wartość **null**. W tym przypadku ma zwrócić widok błędu. Linijka tworzenia **Mocka** po pierwsze definiuje, że oczekuje wywołania danej funkcji, następnie określa jakich oczekujemy parametrów, i na końcu określa jakie dane mają być zwrócone.

Po zdefiniowaniu **MOCK-a** uruchamiane jest odtwarzanie jego zachowania. W momencie przeprowadzania testu podczas wywoływania metody **Create**, po natrafieniu na podmienioną metodę (w przypadku przedstawionym w zamieszczonym kodzie źródłowym **AddCourse**), test nie skorzysta z bazy danych, ale z udawanego obiektu, otrzymując wartość **null**.

Logika kontrolerów pokryta jest łącznie ponad 50 testami jednostkowymi.

5.4.3 Mapowanie obiektowo relacyjne - NHibernate

Dostęp do bazy danych jest realizowany za pomocą mappera obiektowo relacyjnego "NHibernate". Jest to open source'owa implementacja frameworka Hibernate popularnego na platformie Java. Na rysunku 5.22 pokazano proces mapowania relacyjno obiektowego.



Rysunek 5.22 Proces mapowania relacyjno-obiektowego

Tworzenie mapowań - FluentNHibernate

Standardowo mapowania w **NHibernacie** definiuje się w plikach konfiguracyjnych **XML**. Jest to dość problematyczna metoda podatna na błędy, a tworzony kod nie jest do końca czytelny. Innym sposobem generowania mapowań jest zastosowanie frameworka **FluentNHibernate**, pozwalającego definiować mapowania w oparciu o mocno typowany i kompilowany kod jednego z języków zgodnych ze specyfikacją CLR - Common Language Runtime.

```

public class CourseModelMap : ClassMap<CourseModel>
{
    public CourseModelMap()
    {
        Id(x => x.ID);
        Map(x => x.Logo);
        Map(x => x.Name).NotNullable();
        Map(x => x.CreationDate).NotNullable();
        Map(x => x.Description);
        Map(x => x.ShortDescription);
        Map(x => x.News);
        Map(x => x.Password).NotNullable();

        //One
        References(x => x.Group).NotNullable().Cascade.All().Not.LazyLoad();
        References(x => x.CourseType).NotNullable().Not.LazyLoad();
        References(x => x.Forum).NotNullable().Cascade.All().Not.LazyLoad();
        References(x => x.ShoutBox).NotNullable().Cascade.All().Not.LazyLoad();

        //Many
        HasMany(x => x.Contents).KeyColumns.Add("CourseId").Cascade.SaveUpdate().LazyLoad();
        HasMany(x => x.Surveys).KeyColumns.Add("CourseId").Cascade.SaveUpdate().LazyLoad();
        HasMany(x => x.Tests).KeyColumns.Add("CourseId").Cascade.SaveUpdate().LazyLoad();
        HasMany(x => x.LearningMaterials).KeyColumns.Add("CourseId").Cascade.SaveUpdate().Not.LazyLoad();
    }
}

```

Mapowanie propercji klasy

Definiowanie relacji 1 do 1

Definiowanie relacji 1 do wielu

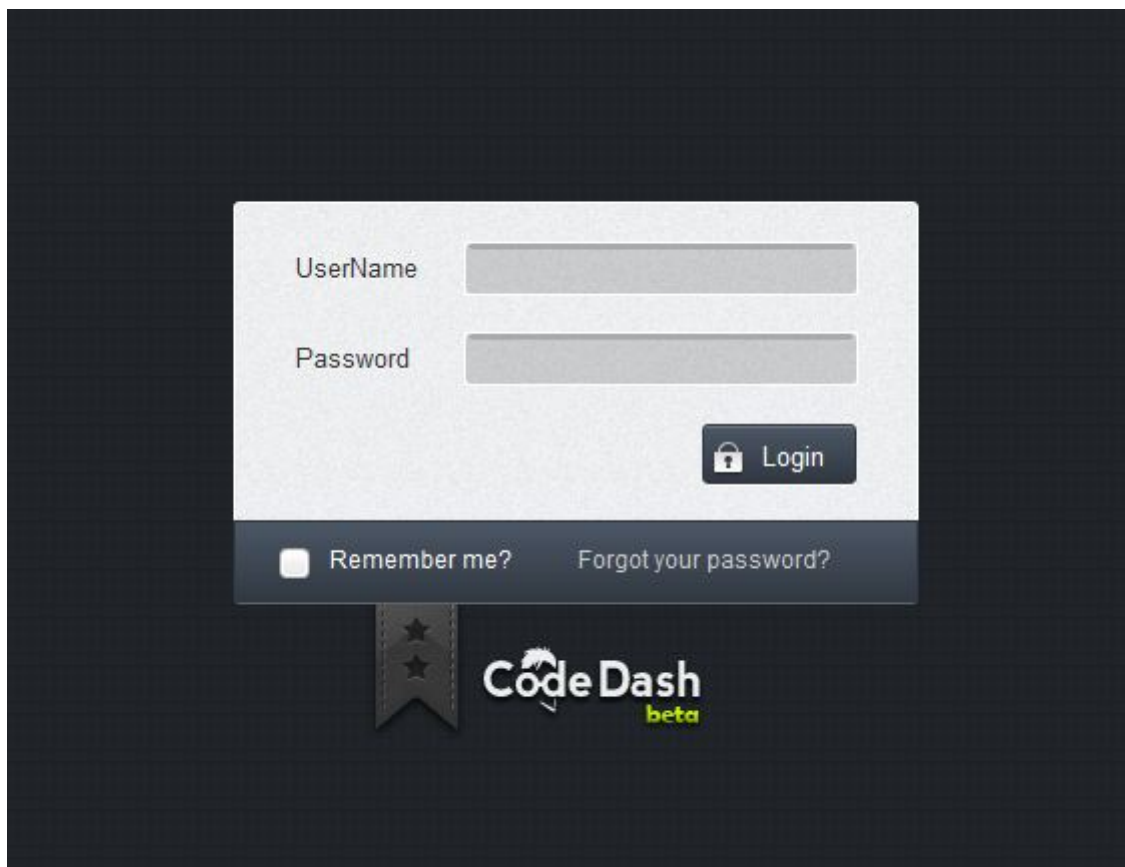
Rysunek 5.23 Przykładowe mapowanie klasy Kurs

Mapowanie realizowane jest poprzez dziedziczenie generycznej klasy **ClassMap** dostępnej w bibliotece **FluentNHibernate**. Wszystkie parametry mapowanej klasy muszą być publiczne oraz oznaczone słowem kluczowym **virtual**. Na początku należy zdefiniować parametr, który będzie zmapowany na pole, będące kluczem bazy danych. Jest to wymagany parametr bez, którego nie można przeprowadzić procesu mapowania. Następnie poprzez użycie Funkcji **Map()** definiuje się mapowania parametrów do określonych wierszy tabel. Kolejną istotną rzeczą do zdefiniowania są relacje. Metoda **Reference()** mapuje relacje jeden do jednego, natomiast **HasMany()** pozwala zdefiniować relację jeden do wielu. Przy każdej funkcji mapującej można ustawić dodatkowe opcje, jak np wiersz przechowujący klucz do elementu referencyjnego, bądź można zdefiniować czy dane powinny być późno wiązane, czy nie.

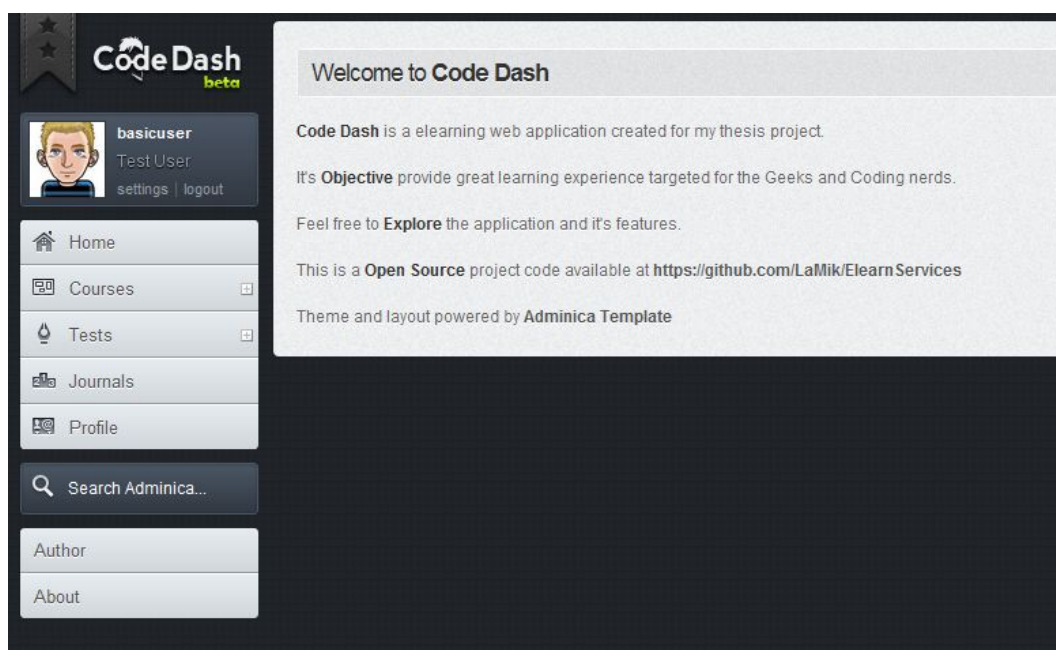
Identyfikacja tabeli, do której należy dana klasa, realizowana jest poprzez nazwę klasy, która jest mapowana. Istnieje oczywiście możliwość przeciążenia tej nazwy poprzez użycie metody **Table()**, podając jako parametr ciąg znaków określający nazwę tabeli.

5.5 Interfejs użytkownika

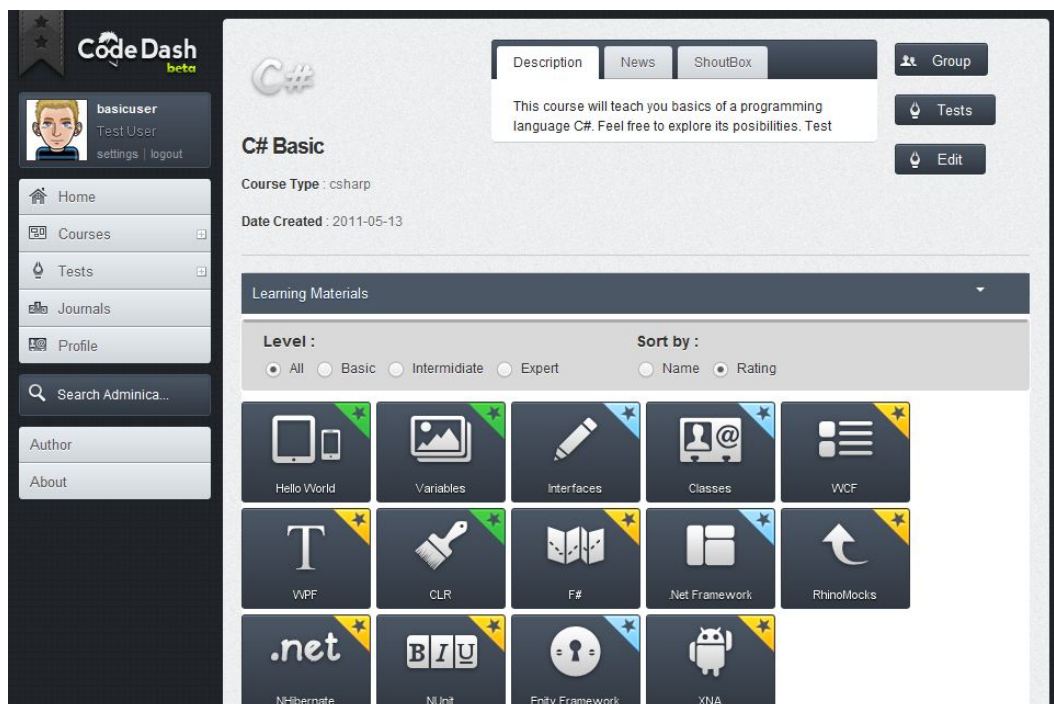
Do komunikacji z użytkownikiem systemu zdalnego nauczania wykorzystywany jest odpowiedni interfejs, którego wybrane elementy pokazano na kolejnych rysunkach.



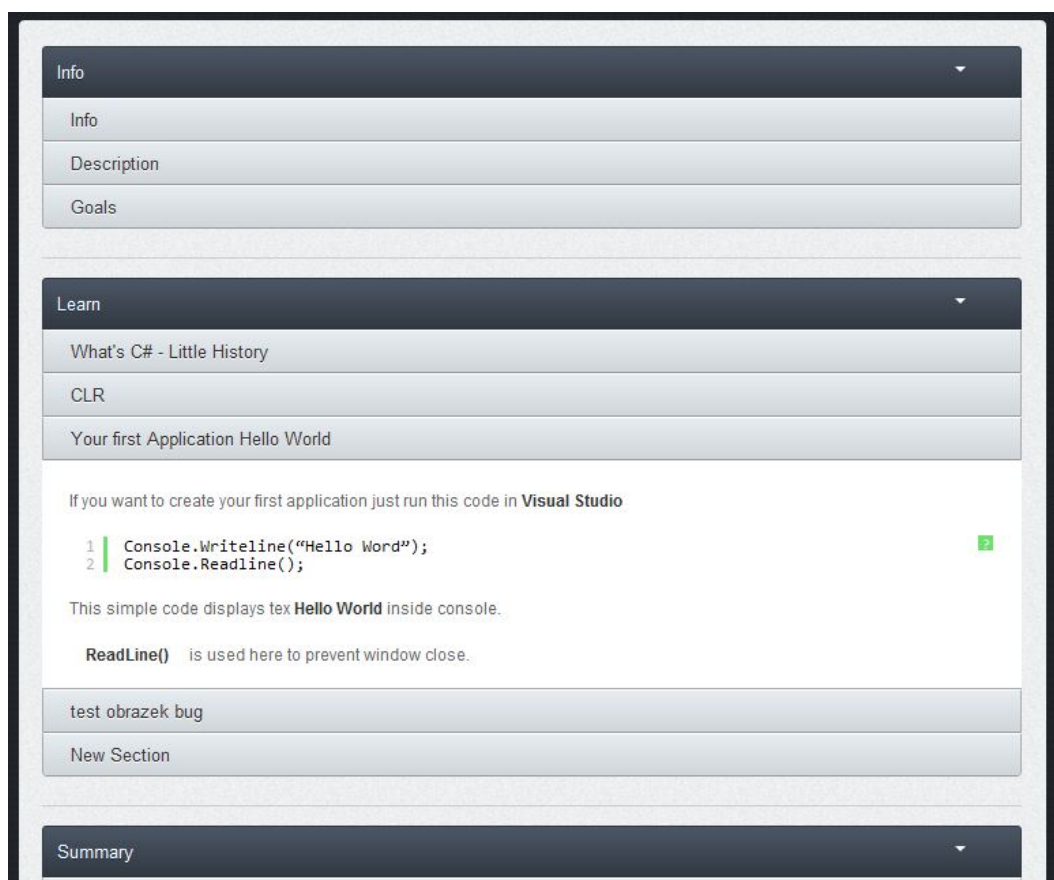
Rysunek 5.24 Widok logowania



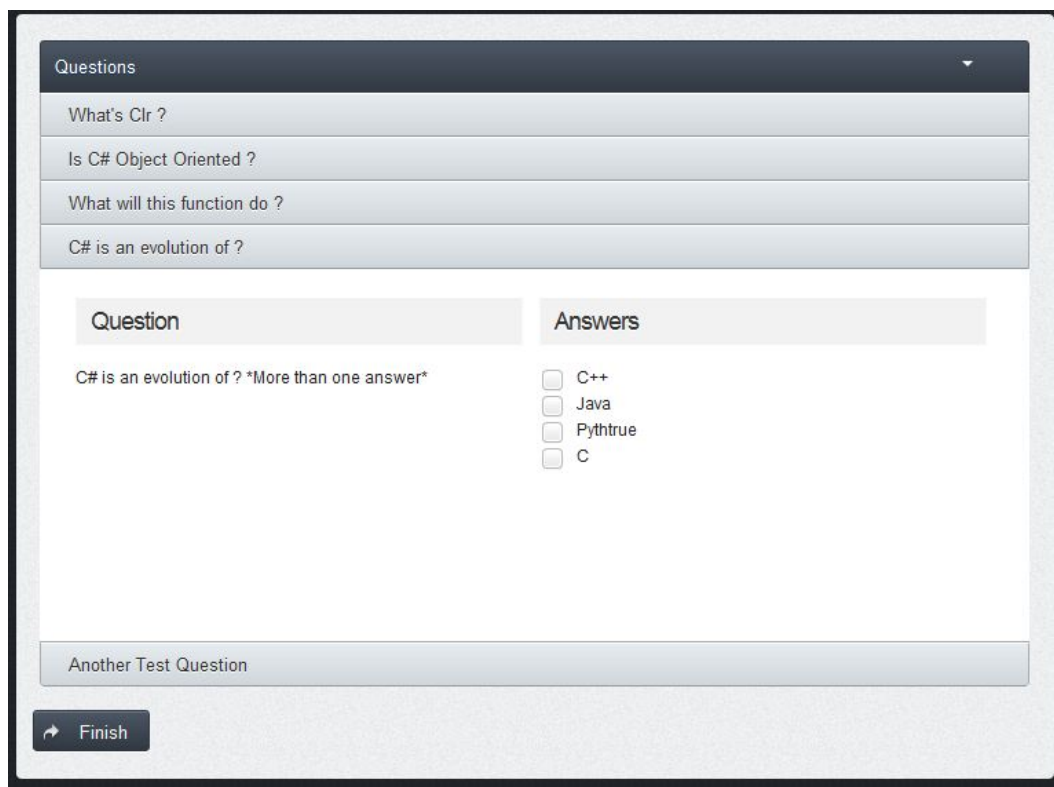
Rysunek 5.25 Widok strony głównej



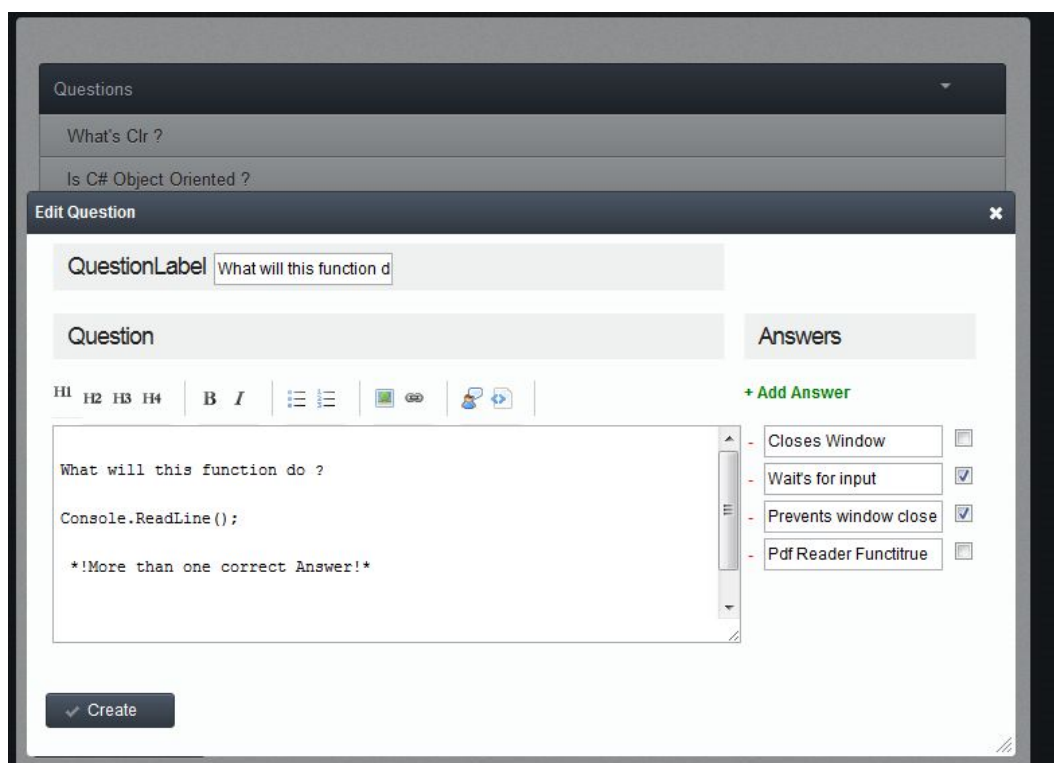
Rysunek 5.26 Widok kursu



Rysunek 5.27 Widok materiału nauczania



Rysunek 5.28 Widok testu



Rysunek 5.29 Widok edycji testu

Rozdział 6

Testowanie i ocena efektywności

6.1 Wybrane testy funkcjonalne

W niniejszym punkcie przedstawiono testy weryfikujące poprawność działania wybranych funkcjonalności systemu.

6.1.1 Test procedury tworzenia nowego materiału nauczania

Scenariusz testowy zakłada wykonanie czynności:

- stworzenie nowego materiału nauczania,
- uaktualnienie parametrów materiału nauczania,
- stworzenie sekcji z odpowiednią zawartością.

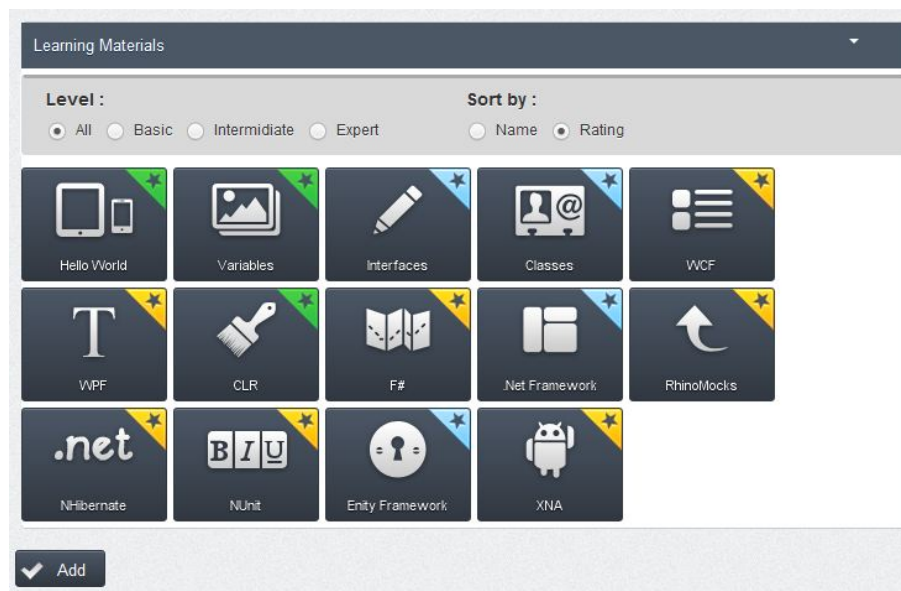
Przejdźcie do panelu edycji kursu

The screenshot displays a web interface for editing a course. It is divided into two main panels: 'Basic Info' and 'Description Info'. The 'Basic Info' panel contains fields for 'Name' (with the value 'C# Basics test'), 'Logo' (with the value 'csharp'), 'CourseType' (a dropdown menu currently showing 'csharp'), and 'IsPasswordProtected' (an unchecked checkbox). The 'Description Info' panel contains two text areas: 'Description' and 'ShortDescription'. Below these panels is a 'Save' button. At the bottom of the interface is a 'Learning Materials' section with a 'Level' filter (radio buttons for 'All', 'Basic', 'Intermediate', 'Expert', with 'All' selected) and a 'Sort by' filter (radio buttons for 'Name', 'Rating', with 'Rating' selected). A bottom navigation bar contains five icons: a mobile device, a picture, a pencil, a person with an '@' symbol, and a list icon.

Rysunek 6.1 Panel edycji kursu

Dane zostały poprawnie pobrane z bazy. Widać wypełnione pola tekstowe oraz stworzoną listę dostępnych materiałów nauczania, przynależących do danego kursu.

Dodanie nowego materiału nauczania



Rysunek 6.2 Lista materiałów nauczania

Poniżej wygenerowanej listy znajduje się przycisk *Dodaj*. Przyciśnięcie tego przycisku spowodowało wygenerowanie nowego materiału nauczania w bazie danych oraz dodanie stworzonego materiału do listy.

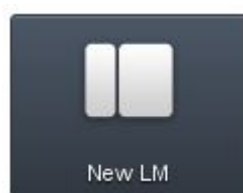
13	13	2011-07-26 00:44:47.000	Entity Framew...	login	2	35
14	14	2011-07-26 00:44:48.000	XNA	android	3	35
15	15	2011-07-26 01:29:01.000	NULL	NULL	0	37
16	16	2011-07-28 22:42:01.000	NULL	NULL	0	36
17	17	2011-07-28 22:42:35.000	NULL	NULL	0	36
18	18	2011-07-28 22:45:10.000	NULL	NULL	0	36
19	19	2011-07-28 22:45:18.000	NULL	NULL	0	36
20	20	2011-07-28 22:45:39.000	NULL	NULL	0	36
21	21	2011-07-28 22:46:46.000	NULL	NULL	0	36
22	22	2011-07-28 22:49:32.000	New LM	nologo	0	36
23	23	2011-07-28 22:52:00.000	New LM	sidebar	0	36

Rysunek 6.3 Baza danych przed dodaniem materiału

Przed procedurą dodania materiału nauczania w bazie znajdują się 23 rekordy.

14	14	2011-07-26 00:44:48.000	XNA	android	3		35
15	15	2011-07-26 01:29:01.000	NULL	NULL	0		37
16	16	2011-07-28 22:42:01.000	NULL	NULL	0		36
17	17	2011-07-28 22:42:35.000	NULL	NULL	0		36
18	18	2011-07-28 22:45:10.000	NULL	NULL	0		36
19	19	2011-07-28 22:45:18.000	NULL	NULL	0		36
20	20	2011-07-28 22:45:39.000	NULL	NULL	0		36
21	21	2011-07-28 22:46:46.000	NULL	NULL	0		36
22	22	2011-07-28 22:49:32.000	New LM	nologo	0		36
23	23	2011-07-28 22:52:00.000	New LM	sidebar	0		36
24	24	2011-09-01 06:50:12.000	New LM	sidebar	0		35

Rysunek 6.4 Baza danych po dodaniu materiału



Rysunek 6.5 Nowy element na liście

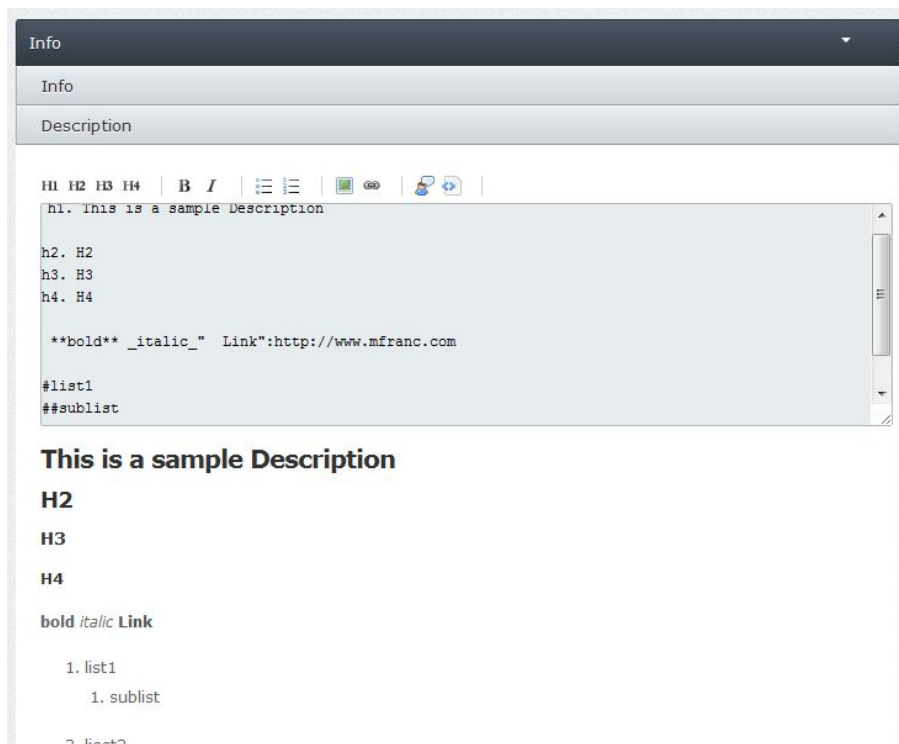
Po wykonaniu operacji dodawania w bazie pojawił się kolejny rekord o numerze ID: 24. Pojawił się również nowy element na liście materiałów.

Przejdźcie do panelu edycji materiału nauczania

Rysunek 6.6 Panel edycji materiału nauczania

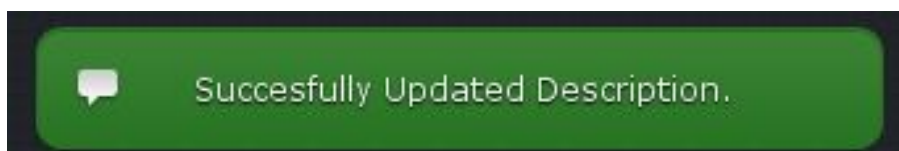
Został wygenerowany odpowiedni formularz z pustymi polami tekstowymi.

Modyfikacja opisu, sprawdzenie formatowania



Rysunek 6.7 Test formatowania tekstu

Poniżej pola tekstowego generowany jest podgląd tekstu przepuszczonego przez parser. Widać, że wybrane komendy działają i odpowiednio formatują tekst.



Rysunek 6.8 Alert informujący o poprawnym zapisie danych

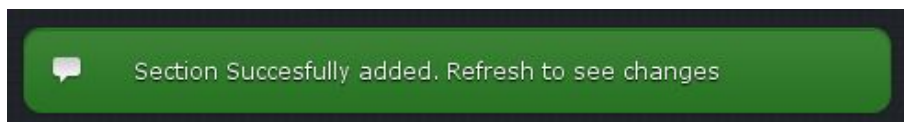
Po zapisaniu danych wyświetla się alert informacyjny.

14	14	2011-07-26 00:44:48.000	NULL	XNA	android	3	2011-07-26 00:44:48.000	35
15	15	2011-07-26 01:29:01.000	NULL	NULL	NULL	0	2011-07-26 01:29:01.000	37
16	16	2011-07-28 22:42:01.000	NULL	NULL	NULL	0	2011-07-28 22:42:01.000	36
17	17	2011-07-28 22:42:35.000	NULL	NULL	NULL	0	2011-07-28 22:42:35.000	36
18	18	2011-07-28 22:45:10.000	NULL	NULL	NULL	0	2011-07-28 22:45:10.000	36
19	19	2011-07-28 22:45:18.000	NULL	NULL	NULL	0	2011-07-28 22:45:18.000	36
20	20	2011-07-28 22:45:39.000	NULL	NULL	NULL	0	2011-07-28 22:45:39.000	36
21	21	2011-07-28 22:46:46.000	NULL	NULL	NULL	0	2011-07-28 22:46:46.000	36
22	22	2011-07-28 22:49:32.000	NULL	New LM	nologo	0	2011-07-28 22:49:32.000	36
23	23	2011-07-28 22:52:00.000	NULL	New LM	sidebar	0	2011-07-28 22:52:00.000	36
24	24	2011-09-01 06:50:12.000	h1. This is a sample Description h2. H2 h3....	New LM	sidebar	0	2011-09-01 06:50:12.000	35

Rysunek 6.9 Baza danych po dodaniu opisu

W bazie danych poprawnie zostało zaktualizowane pole *Description* dla elementu o numerze ID: 24

Dodanie sekcji nauczania



Rysunek 6.10 Alert informujący o poprawnym stworzeniu nowej sekcji

14	14	New Section	23
15	15	New Section	23
16	16	New Section	23

Rysunek 6.11 Tabela zawierająca rekordy sekcji przed dodaniem

14	14	New Section	23
15	15	New Section	23
16	16	New Section	23
17	17	New Section	24

Rysunek 6.12 Tabela zawierająca rekordy sekcji po dodaniu nowej sekcji

Przyciśnięcie przycisku dodaj sekcję spowodowało dodanie nowego rekordu do bazy oraz wyświetlenie alertu informującego. W bazie danych pojawił się nowy wpis o numerze ID: 17.

Tabela. 6.1 Testy funkcjonalne operacji tworzenia materiałów nauczania

Krok	Oczekiwany rezultat	Wynik
1. Przejście do panelu edycji Kursu	Dane pobrane z bazy	OK
	Wygenerowanie widoku	OK
2. Dodanie materiału nauczania	Nowy rekord dodany do tabeli	OK
	Przejście do widoku edycji	OK
3. Edycja parametrów Zatwierdzenie zmian	Poprawne formatowanie tekstu	OK
	Uaktualnienie rekordu w bazie danych	OK
	Wyświetlenie alertu informacyjnego	OK
4. Dodawanie Sekcji nauczania	Dodanie rekordu w bazie	OK
	Wyświetlenie alertu informacyjnego	OK

Kolejne testy są przeprowadzane przy użyciu analogicznej procedury testowej.

6.1.2 Test procedury tworzenia nowego testu

Scenariusz testowy zakłada wykonanie czynności:

- stworzenie nowego testu,
- stworzenie pytań z odpowiedziami.

Tabela. 6.2 Testy funkcjonalne operacji tworzenie testu

Krok	Oczekiwany rezultat	Wynik
1. Przejście do panelu tworzenia testu	Wygenerowanie odpowiedniego widoku	OK
2. Dodanie Testu	Dodanie nowego rekordu do bazy	OK
	Przejście do widoku edycji testu	OK
3. Dodanie nowego pytania	Wyświetlenie modalnego okna z formularzem edycji pytania	OK
4. Stworzenie pytania	Poprawne formatowanie tekstu	OK
5. Zatwierdzenie pytania	Dodanie pytania do bazy danych	OK
	Zamknięcie modalnego okna	OK

6.1.3 Test procedury rozwiązywania testu

Scenariusz testowy zakłada wykonanie czynności:

- uruchomienie testu,
- rozwiązanie testu,
- sprawdzenie wyniku.

Tabela. 6.3 Testy funkcjonalne operacji rozwiązywanie testu

Krok	Oczekiwany rezultat	Wynik
1. Przejście do panelu rozwiązywania testu	Pobranie danych z bazy danych	OK
	Wygenerowanie widoku	OK
2. Zatwierdzenie testu	Wyliczenie wyniku	OK
	Dodanie rekordu do bazy danych	OK
	Wygenerowanie widoku z wynikiem	OK

6.1.4 Test procedury tworzenia nowego kursu

Scenariusz testowy obejmuje:

- stworzenie nowego kursu

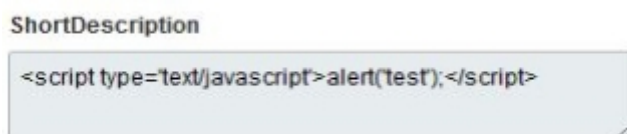
Tabela. 6.4 Testy funkcjonalne operacji tworzenie kursu

Krok	Oczekiwany rezultat	Wynik
1. Przejście do widoku tworzenia kursu	Wygenerowanie widoku z formularzem	OK
2. Zatwierdzenie danych	Stworzenie kursu w bazie danych	OK
	Stworzenie rekordu w tabeli "Grupa"	OK
	Stworzenie rekordu w tabeli "Shoutbox"	OK

6.2 Testy zabezpieczeń przed atakiem poprzez wstrzykiwanie skryptów

Test ten ma na celu sprawdzenie odporności aplikacji na próby wstrzyknięcia skryptów do pól tekstowych. Dobrze zabezpieczona aplikacja w przypadku próby wysłania tekstu zawierającego różnego rodzaju skrypty (np. fragmenty kodu języka **Javascript**) powinna odrzucić, bądź zmodyfikować dane tak, by były one bezpieczne w momencie wyświetlania strony **WWW** z poziomu przeglądarki internetowej.

Błędnie zabezpieczona aplikacja pozwoli "wrzucić" taki skrypt w niezmienionej formie do bazy danych. W momencie, gdy skrypt ten zostanie pobrany z bazy i wrzucony na wygenerowaną stronę **WWW**, przeglądarka podejmie próbę uruchomienia podrzuconego skryptu.



Rysunek 6.13 Przykładowe wstrzyknięcie skryptu do edytora tekstowego

W przypadku zaprezentowanym na powyższym rysunku, dane wpisane do edytora tekstu, po odtworzeniu i wyświetleniu na stronie **WWW**, zostaną zinterpretowane jako skrypt języka **Javascript**. Wyświetlony zostanie komunikat z wiadomością test. Jest to nieszkodliwy skrypt, jednakże taka luka w aplikacji pozwala wrzucić skrypty mogące wyrządzić spore szkody w systemie, bądź wygenerować straty dla użytkownika aplikacji. Przykładem niebezpiecznego skryptu może być wstrzyknięcie mechanizmu przechwytywania znaki wprowadzane przez użytkownika do pól tekstowych. Dzięki temu osoba przeprowadzająca atak mogłaby pozyskać np. informacje dotyczące konta użytkownika, a nawet hasła dostępu.

Scenariusz testowy - włączony domyślny mechanizm zabezpieczający

Scenariusz testowy:

- Otwarcie widoku edycji kursu,
- Wpisanie skryptu pokazanego na rysunku 6.13 do jednego z edytorów tekstowych,
- Wysłanie formularza do serwera.

Przeprowadzenie testu spowodowało wystąpienie wyjątku **HttpRequestValidationException**. Jest to wiadomość zwrotna domyślnego zabezpieczenia dostępnego na platformie Asp.Net [18]. Mechanizm ten analizuje dane wysyłane w postaci żądania do serwera **HTTP**, wyszukując niebezpiecznych ciągów znaków.

Scenariusz testowy - wyłączony domyślny mechanizm zabezpieczający

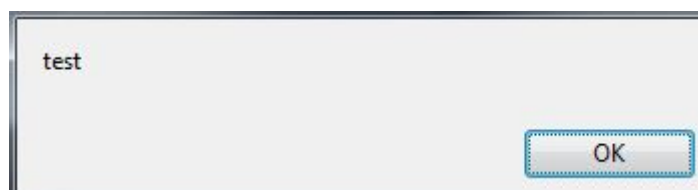
Istnieje możliwość wyłączenia domyślnego zabezpieczenia dla poszczególnych stron, bądź dla całej aplikacji. Zaimplementowana aplikacja posiada rozbudowany edytor tekstu dostępny m.in z poziomu formularza edycyjnego materiału nauczania. Dla tego konkretnego widoku edytora, zabezpieczenie to zostało wyłączone. Dzięki temu będzie istniała możliwość przesłania fragmentów kodu, które w tym przypadku są materiałem dydaktycznym. W przypadku pozostawienia włączonego zabezpieczenia sprawdzającego zawartość wysyłanych danych, wystąpiłby odpowiedni wyjątek.

Scenariusz testowy:

- Otwarcie widoku edycji materiału nauczania,
- Wpisanie skryptu do jednego z edytorów tekstowych,
- Wysłanie formularza do serwera,
- Otwarcie widoku wyświetlającego materiał nauczania.



Rysunek 6.14 Przykład testowego skryptu

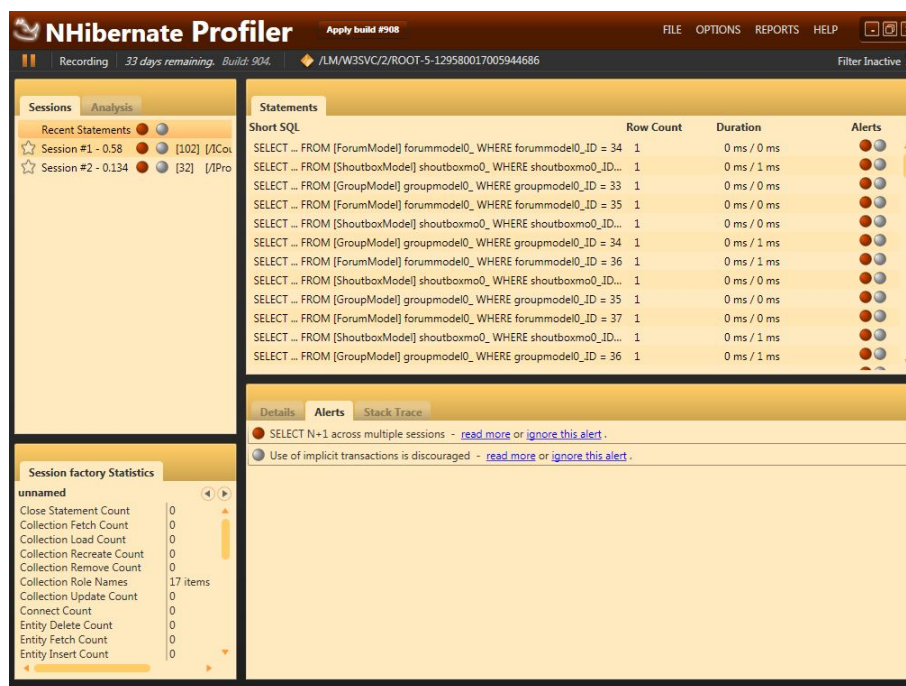


Rysunek 6.15 Udany atak typu script injection

Po przeprowadzeniu podpunktu czwartego wyświetlił się alert. Oznacza to, że w tym przypadku po wyłączeniu domyślnego zabezpieczenia wysłane dane nie zostały zabezpieczone. Po przeanalizowaniu kodu okazało się, że w testowanej wersji aplikacji nie istniał żaden dodatkowy mechanizm zabezpieczający wprowadzany tekst. Ponieważ edytor ten musi zezwolić na wrzucenie zawartości, która jest interpretowana jako niebezpieczna nie możemy zastosować domyślnego mechanizmu. Jedną z opcji pozwalających zabezpieczyć aplikację, a jednocześnie zezwalającą na dodanie bardziej rozbudowanych tekstów, jest oczyszczenie wprowadzanych danych.

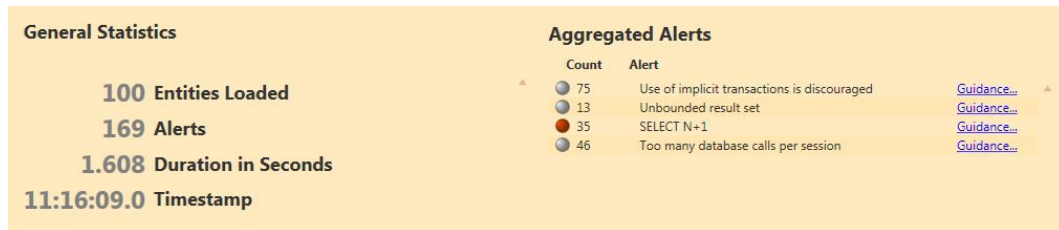
6.3 Testy oraz optymalizacja zapytań generowanych przez framework NHibernate

W projekcie warstwa dostępu do danych wykorzystuje mechanizm mapowania relacyjno-obiektowego przy użyciu framework-a **NHibernate**. Jest to bardzo rozbudowane narzędzie posiadające wiele mechanizmów pozwalających usprawnić proces komunikacji z bazą danych. Niestety jest on również bardzo złożony. Konfiguracja mechanizmów przetwarzania danych wymaga szczegółowego testowania i usprawniania. Narzędziem ułatwiającym ten proces jest aplikacja **NHProf**. Jest to narzędzie pozwalające analizować proces przetwarzania danych z wykorzystaniem **NHibernate-a**.



Rysunek 6.16 Główny widok aplikacji NHProf.

Jedną z zalet wykorzystania mechanizmu mapowania relacyjno obiektowego jest wyeliminowanie konieczności ręcznego tworzenia zapytań języka **SQL**. Dobry framework mapujący zapewni mechanizm generowania zapytań do bazy danych pod fasadą prostych interfejsów. Generowane zapytania można zaobserwować używając darmowego narzędzia **SqlProfile** dostępnego wraz z serwerem **MSSQL**. Lepszym jednak rozwiązaniem jest wykorzystanie narzędzia **NHProf**, które pozwala w prosty i przejrzysty sposób przeanalizować generowane zapytania. Prócz mechanizmu przechwytywania zapytań posiada świetne narzędzia analizujące działanie **ORM-a**. Potrafi wykryć najczęstsze problemy wynikające z błędnej konfiguracji, błędnych mapowań bądź błędnie zaprojektowanego mechanizmów przetwarzania danych. Podłączenie profilera pod testowaną aplikację jest stosunkowo proste. Cała procedura sprowadza się do dodania jednej linijki kodu odpowiedzialnej za podłączenie naszej aplikacji do mechanizmu logowania używanego przez **NHProf**. Inicjalizacja mechanizmu profilowania jest uruchamiana w momencie startu usługi sieciowej.



Rysunek 6.17 Przykładowy widok prezentujący wynik profilowania.

NHProf pozwala wyznaczyć trzy istotne parametry, na których oprzemy analizę i proces optymalizacji mechanizmu przetwarzania danych.

- Czas zapytania - określa całkowity czas potrzebny na wykonanie danej operacji,
- Ilość generowanych zapytań - określa liczbę zapytań potrzebnych do wykonania danej operacji,
- Ilość pobieranych encji - określa ilość zwróconych obiektów,
- Ilość operacji zapisu - określa ilość operacji zapisujących do bazy danych.

Testowanie operacji pobierania danych

Pobieranie danych realizowane przy pomocy frameworka **NHibernate** zostanie przetestowane poprzez wykonanie procedury pobrania danych na temat jednego kursu.

Tabela. 6.5 Parametry generowanych zapytań - wstępna próba

Czas zapytania	1.608s
Ilość zapytań	75
Ilość encji	100

Wstępna próba wykazała kilka problemów. Po pierwsze pobierane jest łącznie 100 encji. Jest stosunkowo duża liczba oznaczająca, że podczas pobierania jednego rodzaju danych z powodu nieoptymalnej konfiguracji mapowań oraz mechanizmów późnej inicjalizacji pobieramy dużo niepotrzebnych danych, które nie zostaną wykorzystane w procesie tworzenia strony **WWW**. Kolejnym parametrem rzucającym się w oczy jest ilość generowanych zapytań. Idealny przypadek zakłada generację tylko jednego zapytania dla wszystkich operacji. W moim przypadku jednak mamy 75 zapytań. Ilość generowanych zapytań jest szczególnie ważna w przypadku skalowalności aplikacji. Zbyt duża ilość zapytań może

bardzo szybko wyczerpać pulę dostępnych połączeń zmuszając innych użytkowników do oczekiwania na zwolnienie się połączenia. Tak duża liczba zapytań najprawdopodobniej spowodowana jest błędną implementacją mechanizmu zarządzania sesją. Czas potrzebny na przeprowadzenie zapytań nie spełnia założeń projektowych, które oczekują czasu odpowiedzi nie większego niż 1s.

Optymalizacja operacji pobierania danych

Proces wprowadzania poprawek zaczniemy od analizy danych wymaganych do generacji testowanego widoku. Widok przedstawiający szczegóły kursu zawiera informacje na temat kursu: tytuł, opis, logo, typ itd. oraz zawiera listę dostępnych materiałów nauczania. Generacja tego widoku dodatkowo wymaga pobrania danych na temat grupy przynależącej do danego kursu. Encja ta jest wykorzystywana do określenia czy dany użytkownik należy do grupy.

Po przeanalizowaniu w programie **NHProf** listy generowanych zapytań okazało się, że generowanych jest bardzo dużo niepotrzebnych encji tzn. takich które nie zostaną wykorzystane w procesie generowania strony **WWW**. By poprawić ten błąd przede wszystkim należy sprawdzić konfigurację mapowań. Bardzo ważne są ustawienia późnej inicjalizacji. W tym przypadku wyłączenie późnej inicjalizacji dla kolekcji powiązanych z pobieranym kursem spowodowało automatyczne pobranie tych kolekcji.

Kolejna poprawka dotyczyła odchudzenia pobieranych encji. Kurs posiada listę materiałów nauczania. Na liście wyświetlana jest tylko nazwa oraz typ materiału nauczania. Przed poprawką w momencie pobierania kursu pobierane były całe obiekty materiałów nauczania wraz z ich kolekcjami. By poprawić ten stan stworzyłem specjalną klasę zawierającą tylko dane niezbędne do wyświetlenia materiału nauczania na liście.

Tabela. 6.6 Parametry generowanych zapytań - po wprowadzeniu poprawek

	Przed poprawką	Po poprawce
Czas zapytania	1.608s	0.308s
Ilość zapytań	75	8
Ilość encji	100	25

Po wprowadzeniu widać znaczną poprawę zarówno w czasie trwania zapytania, ilości generowanych zapytań jak i ilości pobieranych encji.

Testowanie operacji zapisywania danych

Analizę operacji zapisu do bazy danych przeprowadzimy na przykładzie edytowania obiektu materiał nauczania. Encja ta składa się przede wszystkim z parametrów opisowych oraz wielu sekcji nauczania. Scenariusz testowy przewiduje modyfikację jednej sekcji nauczania i zapis zmian do bazy danych. W przypadku tego testu nie otrzymamy parametrów liczby pobranych encji oraz liczby generowanych zapytań ponieważ jest to operacja generująca jedynie operacje uaktualniania bazy danych.

Tabela. 6.7 Parametry generowanych zapisów - wstępna próba

Czas operacji	3.668s
Ilość zapisów	60

Wstępna próba wykazała, że zarówno czas operacji jak i ilość zapytań są zbyt duże i należy je poprawić. Scenariusz testowy zakładał modyfikację jednego pola sekcji. Tak wysoka liczba operacji zapisu. Jak widać coś tu jest nie tak skoro wykonujemy aż 60 operacji zapisu. Po analizie logów generowanych przez aplikację **NHProf** okazało się, że podczas uaktualniania danych materiału nauczania automatycznie wykonywane były operacje uaktualniania elementów z nim powiązanych.

Optymalizacja operacji zapisywania danych

Analiza generowanych operacji zapisu wykazała, że główny problem z tą procedurą wynikał były spowodowane błędnym mapowaniem wewnątrz materiału nauczania. Błąd ten powodował przy każdej operacji zapisu zmian do tabeli materiału nauczania jednocześnie uaktualnianie tabeli testów oraz pytań i odpowiedzi.

Tabela. 6.8 Parametry generowanych zapisów - po wprowadzeniu poprawek

	Przed poprawką	Po poprawce
Czas operacji	3.668s	0.571s
Ilość zapisów	60	13

Wprowadzone poprawki wyłączyły kaskadowe uaktualnianie danych powodujące niepotrzebne uaktualnianie tabel testów, pytań i odpowiedzi. Widać wyraźną poprawę zarówno w czasie potrzebnym na wykonanie operacji jak i w liczbie generowanych zapytań.

6.4 Ocena wydajności aplikacji oraz proces optymalizacji mechanizmów przetwarzania danych

Założenia projektowe zakładają czas odpowiedzi dla 100 użytkowników korzystających z aplikacji nie większy od 1 sekundy. Przeprowadzenie odpowiednich testów wydajnościowych pozwoli sprawdzić, czy aplikacja spełnia te wymagania. Do przeprowadzenia testów obciążeniowych zostanie użyte darmowe narzędzie firmy Microsoft **WCAT - Web Capacity Analysis Tool** [10] dostępne z pakietem narzędzi serwera **IIS**. Jest to aplikacja konsolowa, pozwalająca generować zapytania protokołu HTTP, a także pozwala zasymulować proces korzystania z aplikacji webowej przez wielu użytkowników jednocześnie.

Silnik narzędzia używa do przeprowadzania symulacji prostego języka skryptowego. Do poprawnego działania wymaga trzech plików, zawierających odpowiednie instrukcje.

- script.txt - zawiera instrukcje potrzebne do wygenerowania zapytania,
- distribution.txt - pozwala skonfigurować symulację, rozkładając proporcjonalnie testy na różne strony www,
- config.txt - zawiera parametry konfiguracyjne symulacji.

Ustawienia skryptu generującego zapytania

```
NEW TRANSACTION
  classId = 1
  NEW REQUEST HTTP
    Verb = "GET"
    URL = "http://localhost/elearn/Course/List"
```

Rysunek 6.18 Zawartość pliku script.txt

Skrypt konfiguruje aplikację **WCAT** [10] tak by wykonywała zapytanie typu **GET**, wykorzystując protokół HTTP. Każdy symulowany użytkownik wyśle żądanie do lokalnego serwera **IIS** z prośbą o wygenerowanie widoku, zawierającego listę dostępnych kursów.

Ustawienia konfiguracji rozłożenia testów

1 100

Rysunek 6.19 Zawartość pliku distribution.txt

Plik ten zawiera jedynie informację, że wszystkie zapytania powinny zostać wygenerowane na podstawie skryptu numer jeden znajdującego się w pliku script.txt.

Ustawienia symulacji

```
Warmuptime 5s
Duration 60s
CooldownTime 5s
NumClientThreads 100
NumClientMachines 1
```

Rysunek 6.20 Zawartość pliku config.txt

Test został skonfigurowany tak by symulować działanie 100 użytkowników jednocześnie. Czas działania symulacji został ustawiony na 60s. W ciągu tego okresu każdy symulowany użytkownik będzie wykonywał zapytania do serwera **WWW**.

Scenariusz testowy

Symulowany scenariusz zakłada wysłanie zapytania w formie komendy **GET** do serwera **IIS**. Zapytanie będzie odwoływało się do widoku wyświetlającego listę dostępnych kursów. By umożliwić wykonanie testu zostały wyłączone wszelkie zabezpieczenia aplikacji. Pozwoli to symulowanym użytkownikom, którzy są anonimowi, korzystać z zasobów serwera.

Przeprowadzenie testu

Narzędzie **WCAT** składa się z dwóch aplikacji, klienta oraz kontrolera. By przeprowadzić test należy przede wszystkim uruchomić kontroler, który jest odpowiedzialny za przeprowadzanie symulacji. Analizuje on dostarczone pliki konfiguracyjne i odpowiednio ustawia środowisko, w którym przeprowadzimy test.

```
Total Requests           :          4145    (    50/Sec)
Total Responses           :          4145    (    50/Sec)
Total Bytes               :       33127622    (   393 KB/Sec)
Total Success Connections :          4144    (    50/Sec)
Total Connect Errors      :              0    (    0/Sec)
Total Socket Errors       :              0    (    0/Sec)
Total I/O Errors          :              0    (    0/Sec)
Total 200 OK               :          4142    (    50/Sec)
Total 30X Redirect        :              0    (    0/Sec)
Total 304 Not Modified     :              0    (    0/Sec)
Total 404 Not Found       :              0    (    0/Sec)
Total 500 Server Error    :              0    (    0/Sec)
Total Bad Status          :              0    (    0/Sec)
```

Rysunek 6.21 WCAT ekran kontrolera

Po uruchomieniu kontrolera uruchamiamy aplikację kliencką, która będzie odpowiedzialna za wykonywanie zapytań.

```

Total Socket Errors      : 0 < 0/Sec>
Total I/O Errors         : 0 < 0/Sec>
Total Internal Errors    : 0 < 0/Sec>
Total Time Outs          : 0 < 0/Sec>
Total 200 OK              : 4583 < 76/Sec>
Total 30X Redirect       : 0 < 0/Sec>
Total 304 Not Modified    : 0 < 0/Sec>
Total 404 Not Found       : 0 < 0/Sec>
Total 500 Server Error    : 0 < 0/Sec>
Total Bad Status         : 0 < 0/Sec>
Min. Connect Time        : 0 MS
Avg. Connect Time        : 0 MS
Max. Connect Time        : 62 MS
Min. Resp Time (1st Byte) : 265 MS
Avg. Resp Time (1st Byte) : 647 MS
Max. Resp Time (1st Byte) : 4539 MS
Min. Response Time (Last) : 265 MS
Avg. Response Time (Last) : 647 MS
Max. Response Time (Last) : 4539 MS
Current Outstanding Connects : 0 < 0 Max>
Current Waitable Connects   : 0 < 0 Max>
Total Asynchronous Connects : 0 < 0/Sec>
Total Discarded Connects    : 0 < 0/Sec>

```

Rysunek 6.22 WCAT ekran klienta

Wyniki pierwszego testu

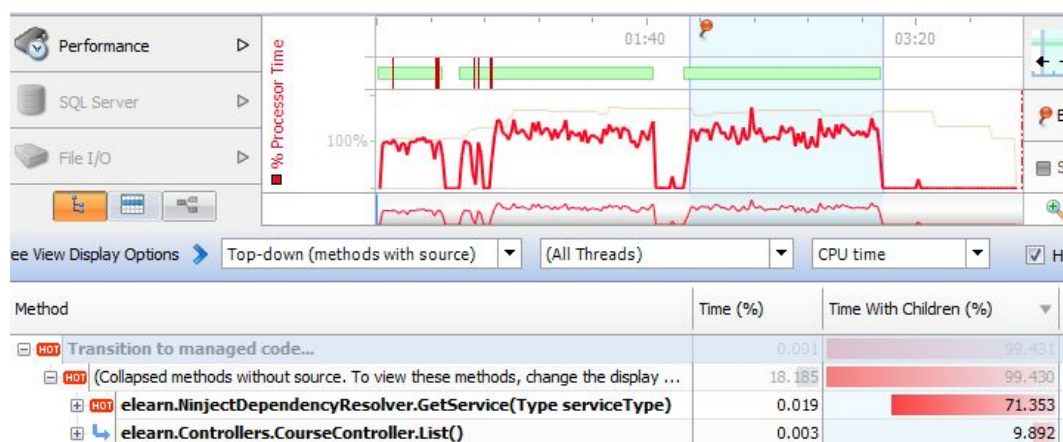
Minimalny	3.120s
Średni	3.773s
Maksymalny	5.600s

Tabela. 6.9 Test wstępny - średni czas odpowiedzi

Otrzymane wyniki nie spełniają założeń projektowych, zakładających czas odpowiedzi nie większy niż 1 sekunda.

Analiza działania aplikacji

Do przeprowadzenia analizy działania aplikacji użyto narzędzia **Ants Performance Profiler** [3]. Jest to narzędzie pozwalające analizować działanie aplikacji na poziomie kodu źródłowego. Dzięki niemu można wyznaczyć fragmenty aplikacji, które były używane najczęściej, i które wymagały najwięcej mocy obliczeniowej, bądź czasu działania aplikacji.



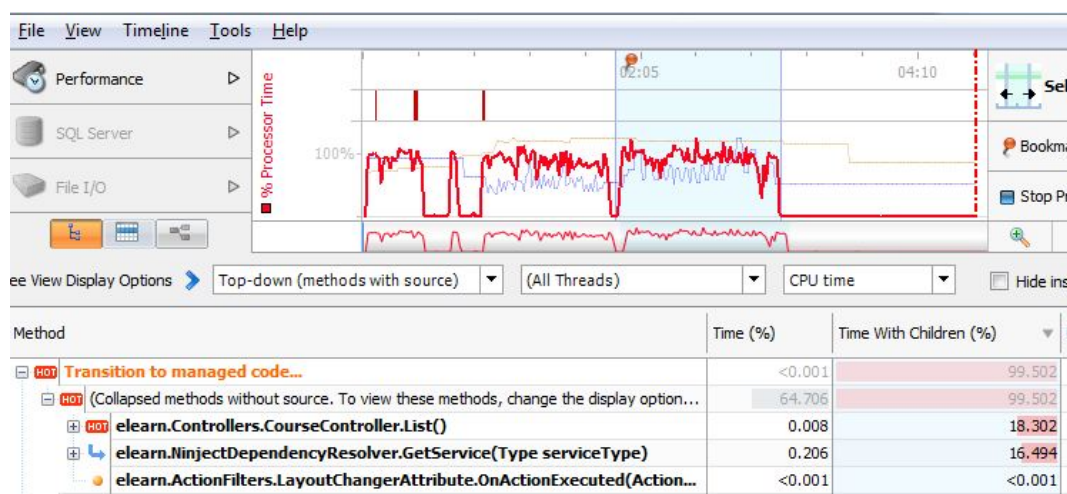
Rysunek 6.23 Statystyka działania aplikacji otrzymana w programie Ants Performance Profiler

Po analizie logów programu profilującego możemy zauważyć, że większość czasu działania aplikacji jest poświęcane na oczekiwanie na wiadomość zwrotną z bazy danych ukrytej za warstwą usług sieciowych. Bardzo dużo czasu jest również marnowane na wykorzystywanie mechanizmów biblioteki **Ninject** [6], zapewniającej implementację wstrzykiwania zależności. Biblioteka ta pozwala odseparować inicjalizację odpowiednich klas, w tym przypadku klas obsługujących usługi. Dzięki temu można w bardzo prosty sposób podmienić implementację danego interfejsu bez konieczności zmieniania kodu w wielu miejscach.

Ninject [6] pozwala definiować zasięg dostępności tworzonych obiektów. W powyższym przypadku bardzo dużo czasu poświęcanego jest na tworzenie instancji obiektów. Ustawiona była opcja tworzenia każdorazowo nowej instancji klasy dla każdego żądania realizowanego przez użytkownika. **Ninject** [6] opiera inicjalizację obiektów na kosztownym mechanizmie refleksji dlatego należy ograniczyć ilość tworzonych obiektów. W przypadku naszej aplikacji jedynymi tworzonymi obiektami przez bibliotekę **Ninject** są konkretne implementacje interfejsów udostępniających usługi sieciowe. Nie potrzebujemy tworzyć tych klas w momencie wykonywania żądania. Lepszym wyjściem będzie tworzenie jednej globalnej, statycznej klasy, wykorzystywanej przez wszystkich symulowanych użytkowników. **Ninject** pozwala zdefiniować takie zachowanie.

Analiza działania aplikacji po wprowadzeniu poprawki

Pierwsza poprawka modyfikuje sposób inicjalizowania obiektów przez bibliotekę **Ninject**.



Rysunek 6.24 Wynik programu profilującego - po poprawce

Tabela. 6.10 Średnie czasy odpowiedzi - po wprowadzeniu modyfikacji

Minimalny	3.020s
Średni	3.673s
Maksymalny	5.300s

Analiza otrzymanych czasów odpowiedzi wykazała, że wprowadzona poprawka nie spowodowała znacznego poprawienia wydajności aplikacji. Poprawka dotyczyła części

klienckiej tzn. serwera WWW. Proces analizy i optymalizacji należy przeprowadzić po stronie usług sieciowych, zapewniających dostęp do bazy danych.

Analiza generowanych zapytań przez framework NHibernate

Analiza narzędziem **NHProf** [5] wykazała, że nieoptymalizowane operacje na bazie danych zajmują 200 milisekund czasu do pobrania danych. NHProf wykazał, że wykonujemy za dużo zapytań do bazy, pobierając niepotrzebne dane. By poprawić tę sytuację należy zmodyfikować mapowania i "odchudzić" pobierane encje, podobnie jak w przypadku optymalizacji generowanych zapytań. Dodatkowo uruchomiony zostanie mechanizm cachowania zapytań. Ponieważ scenariusz testowy zakłada równoległy dostęp wielu użytkowników należy uruchomić cache drugiego poziomu, który udostępnia zawartość globalnie dla wszystkich użytkowników.

Tabela. 6.11 Operacje na bazie danych

	Pierwszy Test	Poprawki mapowań	Cache 2 poziomu
Ilość encji	36	20	0
Ilość zapytań	21	5	0
Czas zapytania	200ms	18ms	13ms

Poprawienie mapowań oraz uruchomienie cache-u drugiego poziomu pozwoliło znacznie ograniczyć czas potrzebny na wykonanie zapytania.

Tabela. 6.12 Średnie czasy odpowiedzi

Minimalny	2.300s
Średni	3.007s
Maksymalny	4.600s

Jak widać zmiany te poprawiły lekko średnie czasy odpowiedzi. Nadal jednak nie jest to wynik spełniający założenia projektowe.

Analiza działania usług sieciowych

Namespace	Method Name	Time (%)
System.Web.Hosting	UnsafeIISMethods.MgdSyncReadRequest(Int...	34.185
log4net.Appender	AppenderSkeleton.DoAppend(LoggingEvent lo...	21.459
HibernatingRhinos.Profiler.App...	StackTraceGenerator +<>c__DisplayClass5.<...	4.939
	(JIT overhead)	4.689
System	RuntimeMethodHandle.InvokeMethodFast(IRu...	2.943
HibernatingRhinos.Profiler.App...	NHibernateProfiler.Initialize(NHibernateAppen...	1.891

Rysunek 6.25 Wynik programu profilującego - aplikacja udostępniająca usługi sieciowe

Po przeprowadzeniu analizy generowanych zapytań przyszedł czas na zbadanie działania serwera udostępniającego usługi sieciowe. Analizy działania aplikacji wykazała, że prawie 30 procent czasu wykorzystywanego przez serwer marnowane jest na generowanie raportów dla aplikacji **NHProf**. Rysunek powyżej wyraźnie pokazuje, że bardzo dużo czasu zajmuje

generowanie logów przez bibliotekę log4net. Logi te są wykorzystywane przez profile **NHiberante-a** do generowania raportów.

Tabela. 6.13 Średnie czasy odpowiedzi - po wyłączeniu profilera

Minimalny	1.014s
Średni	1.198s
Maksymalny	1.701s

Wyłączenie profilera znacznie poprawiło średnie czasy odpowiedzi zbliżając je do oczekiwanego wyniku zgodnego z przyjętymi założeniami

Analiza działania aplikacji po stronie klienta wykazała, że większość czasu poświęca on na generowanie oraz oczekiwanie na odpowiedź zapytania wysyłane do serwera usług sieciowych. W związku z tym duży zysk w czasach odpowiedzi uzyskamy po uruchomieniu mechanizmu cachowania zapytań. Wiadomości wysyłane do serwera usług sieciowych są zwykłymi wiadomościami używanymi w protokole **HTTP**, więc możemy uruchomić standardowy mechanizm dostępny na platformie Asp.Net. Cachowanie zostało ustawione tak, by trzymać dane zapytania przez 60 sekund. Jest to wystarczający czas by jednocześnie zapewnić zwiększony czas odpowiedzi serwera oraz dostarczać świeże dane do klienta.

Tabela. 6.14 Średnie czasy odpowiedzi - po wyłączeniu profilera i po włączeniu z mechanizmem cachowania zapytań

Minimalny	405ms
Średni	455ms
Maksymalny	656ms

Włączenie mechanizmu cachowania zapytań pozwoliło zredukować średni czas odpowiedzi serwera do akceptowalnego poziomu, spełniającego założenia projektowe.

6.5 Ocena wpływu konfiguracji rozłożenia usług sieciowych na czas odpowiedzi

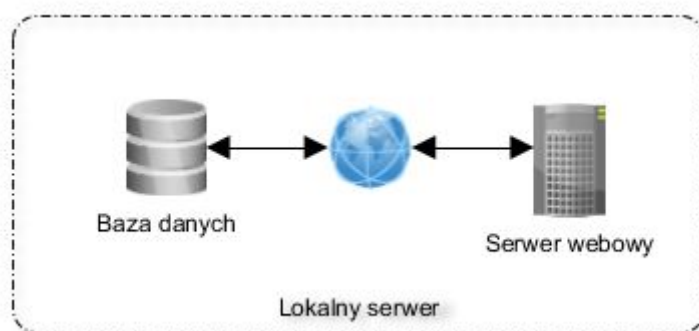
W tym teście przeprowadzona zostanie analiza różnicy czasu odpowiedzi usług sieciowych w zależności od konfiguracji serwerów udostępniających usługi.

Scenariusz testowy

Scenariusz testowy zakłada uruchomienie symulacji tworzącej 10 użytkowników, wysyłających jednocześnie zapytania do serwera. Zostaną zbadane czasy odpowiedzi. Wykorzystana zostanie aplikacja **WCAT**.

Rozpatrzone zostaną dwa przypadki konfiguracji.

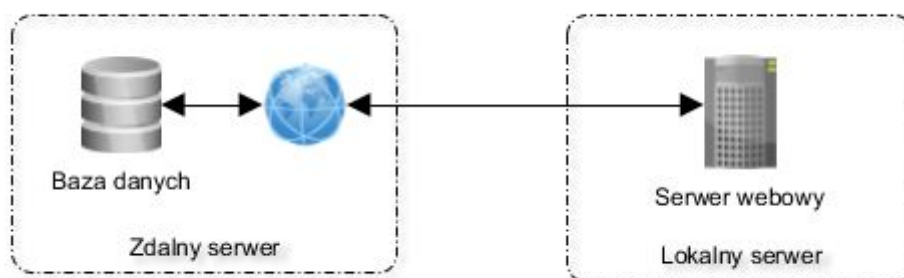
Lokalny serwer usług sieciowych



Rysunek 6.26 Uproszczony schemat lokalnej konfiguracji

W tej konfiguracji aplikacja webowa znajduje się na tym samym serwerze, co aplikacja udostępniająca bazę danych za pomocą mechanizmu usług sieciowych.

Zdalny serwer usług sieciowych



Rysunek 6.27 Uproszczony schemat zdalnej konfiguracji

Tabela. 6.15 Porównanie czasów odpowiedzi

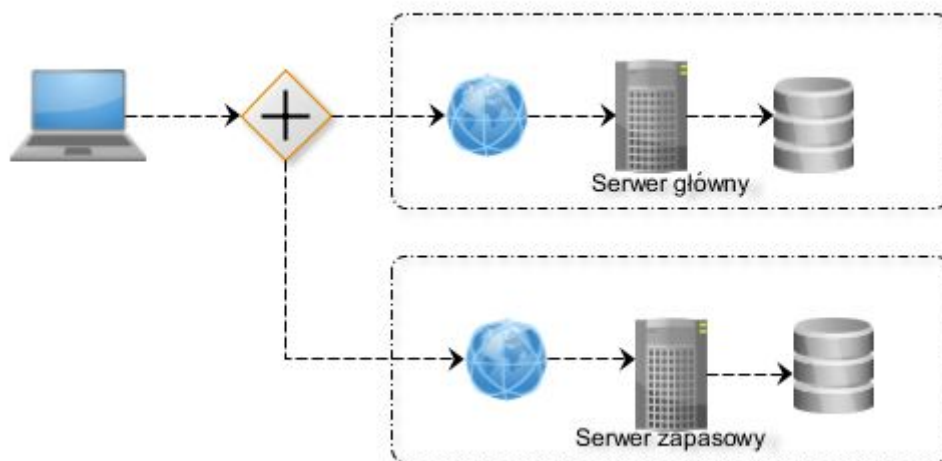
	Lokalny	Zdalny
Minimalny	15ms	62ms
Średni	53ms	217ms
Maksymalny	265ms	1794ms

W tej konfiguracji baza danych wystawiona jest na zewnętrznym serwerze hostowanym przez firmę **www.webio.pl**.

Otrzymane średnie czasy odpowiedzi wskazują wyraźnie, że serwer bazy danych wystawiony zdalnie wprowadza dodatkowe opóźnienie w komunikacji klient serwer. Średni czas odpowiedzi jest prawie cztero krotnie większy w porównaniu do przypadku używającego konfiguracji lokalnej. Spowodowane jest to przede wszystkim koniecznością oczekiwania na transfer danych z i do zdalnego serwera.

6.6 Ocena wydajności mechanizmu przekierowania na zapasowy serwer usług sieciowych

By zaprezentować korzyści wynikające z architektury opartej na usługach sieciowych. Stworzony został mechanizm pozwalający podmienić serwer usług sieciowych w zależności od poziomu obciążenia głównego serwera.



Rysunek 6.28 Uproszczony schemat mechanizmu zmiany serwera

Scenariusz testowy

Procedura testowa zakłada zasymulowanie jednoczesnej aktywności 50 użytkowników wysyłających zapytania do serwera **WWW**. Główny serwer został wystawiony na zdalnym serwerze, natomiast serwer zapasowy znajduje się w przestrzeni lokalnej.

W teście tym przeanalizujemy trzy konfiguracje:

- Wyłączony mechanizm przekierowania,
- Przekierowanie: parametr 150ms, oznaczający przekierowanie na zapasowy serwer po przekroczeniu czasu oczekiwania 150ms,
- Przekierowanie: parametr 100ms, oznaczający przekierowanie na zapasowy serwer po przekroczeniu czasu oczekiwania 100ms.

Analiza działania aplikacji z wyłączonym mechanizmem przekierowania

Tabela. 6.16 Wyłączony mechanizm - średnie czasy odpowiedzi

Minimalny	515ms
Średni	1141ms
Maksymalny	4134ms

Podstawowa implementacja wygenerowała dość znaczne czasy odpowiedzi. Test ten będzie punktem odniesienia w analizie zysków, bądź ewentualnych strat, po zastosowaniu mechanizmu przekierowania.

Analiza działania aplikacji z włączonym mechanizmem przekierowania - 150ms

Tabela. 6.17 Przekierowanie: parametr 150ms - średnie czasy odpowiedzi

Minimalny	421ms
Średni	759ms
Maksymalny	4041ms
Procent przekierowanych połączeń	24

Widać lekką poprawę po uruchomieniu mechanizmu przekierowania. Ustawienie parametru przekierowania na 150ms spowodowało, że prawie 25 procent zapytań zostało skierowanych na serwer zapasowy, zmniejszając obciążenie serwera głównego.

Analiza działania aplikacji z włączonym mechanizmem przekierowania - 100ms

Tabela. 6.18 Przekierowanie: parametr 100ms - średnie czasy odpowiedzi

Minimalny	265ms
Średni	580ms
Maksymalny	2026ms
Procent przekierowanych połączeń	54

Po ustawieniu parametru przekierowania na 100ms można zauważyć znaczną poprawę w czasie odpowiedzi serwera. Ponad 50 procent połączeń zostało skierowanych na serwer zapasowy. Można powiedzieć, że wykorzystanie serwerów zostało rozłożone równomiernie.

6.7 Wnioski z testów i badań

Współczesna inżynieria oprogramowania jest bardzo podzielona w zakresie podejść do sposobu prowadzenia projektów. Wiodącą metodyką, najczęściej stosowaną przy dużych i średnich projektach jest bardzo dobrze znana i opisana w literaturze metodyka **"Waterfall"** [21], coraz częściej jednak można usłyszeć o rodzinie metodyk zrzeszonych w ramach ruchu **"Extreme Programming"**, które stają się bardzo popularne w środowisku małych firm. Mimo wielu różnic metody te nadal posiadają jeden stały i niezmienny element, konieczność przeprowadzania testów aplikacji. Testy przeprowadzone w tym rozdziale również pokazały tę zależność. Aplikacja może się dobrze prezentować, spełniać wymagania funkcjonalne jednakże dopiero przeprowadzenie testów symulujących scenariusze realnego użytkowania pokazują czy produkt nadaje się do użytku, czy wymaga wielu poprawek nim zawita na serwer produkcyjny.

Pierwszym podstawowym testem jest analiza funkcjonalna aplikacji. Test ten ma na celu potwierdzenie czy stworzony produkt spełnia oczekiwania klienta ustalone w zatwierdzonym projekcie. Jest to procedura najczęściej przeprowadzana przez dział wyznaczony specjalnie do tego celu. Jednakże z doświadczenia wiem, że często praca ta wykonywana jest przez osobę piszącą kod w celu zweryfikowania wprowadzonych modyfikacji. Nie inaczej było tym razem. Testy funkcjonalne wybranych modułów aplikacji wykazały, że spełniają one wyznaczone wymagania. Podczas przeprowadzania testów nie wydarzyło się nic nieoczekiwanego. Owszem w trakcie trwania procesu implementacji aplikacji testy te były przeprowadzane w sposób ciągły. Między innymi dlatego, ostateczna weryfikacja działania aplikacji nie wykazała żadnych poważniejszych błędów. Podczas tworzenia projektu do testowania fragmentów kodu najbardziej narażonych na pomyłki i błędy został użyty nowoczesny mechanizm testów jednostkowych, wspomaganych mockowaniem zależności. Testy te przeprowadzane były po każdej większej zmianie kodu, gdy pojawił się jakiś problem bądź błąd, który był szybko eliminowany. Szybka weryfikacja kodu jest ostatnio bardzo popularna w środowisku programistów popierających ruch **Extreme programming**. W szybkim wykrywaniu błędów nie bez znaczenia był także fakt zaimplementowania rozbudowanego mechanizmu generującego logi aplikacji, pozwalającego obserwować zachowanie produktu. Typowa sesja programistyczna wyglądała tak, że na jednym ekranie monitora modyfikowano kod, natomiast na drugim ekranie monitora obserwowane były logi z działania aplikacji, oraz z przeprowadzanych testów jednostkowych. Wdrożenie tych mechanizmów okazało się bardzo dobrą inwestycją.

W dalszej części, procedurze testowania został poddany mechanizm zabezpieczeń neutralizujący próby wstrzyknięcia niepożądanych skryptów. Był to ważny test ponieważ część edytorów tekstowych wykorzystuje rozbudowany mechanizm pozwalający składać tekst przy użyciu z góry ustalonego języka znaczników. Dodatkowo, ponieważ stworzona platforma tworzona była pod kątem użytkowników chcących poszerzać wiedzę z zakresu inżynierii oprogramowania, bardzo istotną funkcjonalnością była możliwość umieszczenia fragmentów kodu na generowanej stronie **WWW**. Standardowe mechanizmy zabezpieczające, dostępne na platformie **.Net**, okazały się nie wystarczające. Przeprowadzony test wykazał lukę w systemie, która została naprawiona. Weryfikacja na tym etapie wdrażania aplikacji oszczędziła wielu problemów i kosztów, które pojawiłyby się po wystawieniu aplikacji na serwerze produkcyjnym.

Kolejnym przetestowanym mechanizmem była zaimplementowana obsługa bazy danych, wykorzystująca mechanizm mapowania obiektowo-relacyjnego przy użyciu framework-a **NHibernate**. W analizie tej istotną rolę odegrało narzędzie **NHProf**, które pozwoliło szybko wyznaczyć krytyczne fragmenty kodu, powodujące najwięcej problemów. Analiza generowanych zapytań wykazała problemy wydajnościowe, wynikające z błędnie skonfigurowanych mapowań, oraz mechanizmów wspomagających proces pobierania danych takich jak późna inicjalizacja. Po wykryciu problemów podjęto próbę naprawienia błędów. Dzięki wprowadzeniu poprawek udało się znacząco zmniejszyć między innymi: liczbę generowanych zapytań, ilość pobieranych danych oraz czas wykonywania zapytania. W analizowanych przypadkach (generowanie listy dostępnych kursów), udało się na przykład zredukować czas potrzebny na wykonanie jednego zapytania z **1.608s** na **300ms** oraz czterokrotnie ograniczyć ilość pobieranych encji. Jest to znaczący zysk odczuwalny po stronie użytkownika, korzystającego z aplikacji. Przeprowadzone badania i testy pokazały jak bardzo ważny jest stopień zaznajomienia i doświadczenia projektanta/programisty z daną technologią. Osoba dopiero co poznająca tak złożoną technologię jak **NHibernate** jest skazana na popełnienie błędów, dlatego też proces profilowania aplikacji pod kątem dostępu do bazy danych jest krytycznym elementem wytwarzania aplikacji.

Po przeprowadzeniu testów funkcjonalnych oraz konfiguracji mapowania obiektowo-relacyjnego, wykonane zostały testy wydajnościowe sprawdzające działanie aplikacji w symulowanym środowisku, udającym rzeczywiste wykorzystanie aplikacji. Zgodnie z założeniami projektowymi założono dostęp do aplikacji dla 100 użytkowników równocześnie otrzymujących odpowiedź od serwera w czasie nie większym od 1 sekundy. Wstępne testy symulowane za pomocą narzędzia **WCAT - Web Capacity Analysis Tool** pokazały, że aplikacja w warunkach zbliżonych do rzeczywistego użytkowania działa bardzo wolno i nie spełnia założeń projektowych. Wykonanie analizy zachowania aplikacji po stronie klienta i serwera, przy pomocy narzędzia **Ants Performance Profile**, pozwoliło wyznaczyć najmniej wydajne fragmenty kodu, powodujące zastój w działaniu aplikacji. Przeprowadzenie takiej analizy pozwoliło wyznaczyć części systemu, których poprawienie przyniesie podczas wdrażania systemu największy zysk czasowy. Optymalizacja we wczesnych fazach projektu często jest nietrafiona i przynosi niewielkie zyski ponieważ nie znamy jeszcze wszystkich interakcji jakie będą zachodziły w systemie. Projekty informatyczne są ciągle modyfikowane, zmieniane, dlatego też nie bez powodu **Donald Knuth** mawiał, że "**Przedwczesna optymalizacja jest źródłem wszelkiego zła**". Wprowadzenie poprawek w wyznaczonych miejscach poprawiło działanie aplikacji do zadowalającego poziomu, spełniającego przyjęte założenia.

Następnie w rozdziale przeprowadzono badanie potwierdzające, że korzystanie ze zdalnego serwera bazy danych jest kosztowniejszym rozwiązaniem niż serwer lokalny.

Ostatnim testem, który zostanie omówiony będzie analiza działania mechanizmu Load balancingu prezentującego jedno z proponowanych zastosowań usług sieciowych. Testy te wykazały, że dzięki odseparowaniu warstwy bazy danych za fasadą usług sieciowych można stworzyć rozwiązania poprawiające skalowalność aplikacji. W zaimplementowanym mechanizmie kluczową rolę odgrywa algorytm decyzyjny, który w przyjętym rozwiązaniu jest dość prosty, gdyż polega na przekierowaniu żądania na zapasowy serwer usług, gdy czas odpowiedzi serwera głównego na dane zapytanie przekroczy ustalony limit. Mimo tej prostoty otrzymane wyniki są zadowalające i wyraźnie pokazują, że takie rozwiązanie poprawia czasy odpowiedzi serwera.

Rozdział 7

Podsumowanie

Celem pracy było zaprojektowanie oraz zaimplementowanie serwisu webowego, wspierającego zdalne procesy dydaktyczne, a także zapoznanie się ze współczesnymi narzędziami i rozwiązaniami stosowanymi w projektach internetowych. Cel pracy został zrealizowany. Przeanalizowano cechy dobrego systemu zdalnego nauczania na podstawie, których stworzono projekt oraz jego implementację. W trakcie tworzenia rozwiązania zastosowano nowoczesne techniki oraz rozwiązania dostępne na rynku. Przeprowadzone testy wykazały konieczność wprowadzeniu wielu poprawek, które ostatecznie spowodowały, że aplikacja spełnia założone wymaganie zapewniające 100 użytkownikom czas odpowiedzi serwera usług nie większy od 1 sekundy. Taki wynik osiągnięto głównie dzięki przeprowadzeniu testów profilujących, za pomocą których udało się wykryć najkosztowniejsze fragmenty kodu. Realizacja badań wydajnościowych stała się możliwa dzięki zapoznaniu się z aplikacją **Ants Performance Profile**, która pozwala uprościć proces profilowania.

W pracy przeanalizowano również mechanizm mapowania obiektowo relacyjnego oraz framework **NHibernate**, realizując w ten sposób dostęp do bazy danych. Podejście to przyniosło znaczne korzyści, szczególnie w trakcie trwania procesu implementacyjnego, znacznie go upraszczając. Podczas profilowania aplikacji okazało się jednak, że kluczem do optymalnego działania aplikacji wykorzystującej ORM jest odpowiednia konfiguracja mapowań i mechanizmów takich jak cachowanie pobieranych danych, czy późna inicjalizacja. Zapoznano się z narzędziem **NHProf**, które zostało użyte do wyznaczenia i wprowadzenia poprawek w konfiguracji **NHibernate-a**.

W ramach testów opracowanej aplikacji wykorzystano testy jednostkowe oraz mechanizm mockowania zależności. Zaznajomiono się z biblioteką **NUnit**. Zastosowanie mechanizmu testów jednostkowych przy wsparciu biblioteki **RhinoMocks** przyniosło znaczne korzyści zmniejszając ilość błędów i zwiększając stopień ich wykrywalności. Aplikacja pokryta jest łącznie ponad 130 testami jednostkowymi, pozwalającymi szybko zweryfikować działanie poszczególnych mechanizmów. Niestety jak się okazało dążenie do całkowitego pokrycia aplikacji testami jednostkowymi było błędnym założeniem, które zostało szybko obalone przez konieczność częstych i żmudnych modyfikacji aplikacji. Idealnie testami jednostkowymi powinno pokrywać się fragmenty kodu najbardziej podatne na wystąpienie nieoczekiwanych błędów. Pisanie testów jednostkowych okazało się przydatne, ale i żmudne.

Stworzone rozwiązanie jest fundamentem na bazie, którego można tworzyć systemy posiadające bardziej rozbudowane mechanizmy wspierające procesy dydaktyczne. Zastosowanie architektury **MVC**, daje możliwość stosunkowo "bezbolesnego" procesu modyfikowania platformy. Rozwój serwisu powinien iść w kierunku implementowania większej ilości funkcjonalności cechujących systemy klasy **LMS**. Posiadając już prosty system klasy **CMS**, to właśnie funkcjonalności takie jak między innymi wspomaganie nauczania poprzez wspieranie działalności dydaktycznej użytkownika przez system agentowy, pozwolą zwiększyć atrakcyjność platformy. Również warto by było zaimplementować bardziej rozbudowane mechanizmy zachęcające do zwiększonej interakcji pomiędzy użytkownikami aplikacji. Procesy społeczne oraz interakcje wewnątrz grupy tematycznej, występujące w standardowym, stacjonarnym przekazie wiedzy, są bardzo ważne i wpływają korzystnie na efektywność procesów nauczania. Dlatego istotną funkcjonalność stanowiłby chociażby interaktywny video-chat, bądź wirtualna tablica, pozwalająca komunikować się wielu użytkownikom jednocześnie.

Bibliografia

- [1] <http://nhforge.org/>.
- [2] The ado.net entity framework overview. [http://msdn.microsoft.com/en-us/library/aa697427\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/aa697427(v=vs.80).aspx).
- [3] Ants memory profiler. <http://www.red-gate.com/products/dotnet-development/ants-memory-profiler/>.
- [4] Django. <https://www.djangoproject.com/>.
- [5] Nhprof. <http://nhprof.com/>.
- [6] Ninject. <http://ninject.org/>.
- [7] Ruby on rails. <http://rubyonrails.org/>.
- [8] Sentinel - log viewer. <http://sentinel.codeplex.com/>.
- [9] Sqlite. <http://www.sqlite.org/>.
- [10] Wcat 6.3 (x86). <http://www.iis.net/community/default.aspx?tabid=34&i=1466&g=6>.
- [11] Wordpress. <http://wordpress.org/>.
- [12] Anderson T., Elloumi F.. *Theory and Practice of Online Learning*. Athabasca University, Athabasca, 2004.
- [13] Bednarek J., Lubina E.. *Kształcenie na odległość podstawy dydaktyki*. PWN, Łódź, 2008.
- [14] Clarke A.. *E-learning : nauka na odległość*. WKiŁ, Warszawa, 2007.
- [15] Clay L.. Lessons from the failures of soap. <http://public.iwork.com/document/pl/?d=LessonsLearnedFromFailureOfSOAP.key&a=p1355173147f>, Maj 2011.
- [16] Hibernate Community Documentation. Hql: The hibernate query language. <http://docs.jboss.org/hibernate/core/3.3/reference/en/html/queryhql.html>.
- [17] Douglas C., The application/json media type for javascript object notation (json). <http://tools.ietf.org/html/rfc4627>, 2006.
- [18] Evjen B., Hanselman S., Rader D., *Asp.Net 3.5 z wykorzystaniem Csharp. Zaawansowane Programowanie*. Helion, Gliwice, 2010.

- [19] T. Fielding R., Representational state transfer (rest). http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000.
- [20] Gamma E., Helm R., Johnson R., M. Vissides J., *Wzorce Projektowe : elementy oprogramowania wielokrotnego użytku*. WNT, Warszawa, 2005.
- [21] Górski J., *Inżynieria Oprogramowania w projekcie informatycznym*. MIKOM, Warszawa, 2000.
- [22] J.D. M., Alex M., Blaine W., Bansode P., Wigley A., Microsoft patterns and practices. <http://msdn.microsoft.com/en-us/library/ff648339.aspx>, Maj 2005.
- [23] Martin F., Fluentinterface. <http://www.martinfowler.com/bliki/FluentInterface.html>, 2005.
- [24] McMurty C., Mercuri M., Waiting N., Winkler M., *Windows Communication Foundation Unleashed*. SAMS, 2007.
- [25] Microsoft. Sql server books online. <http://msdn.microsoft.com/en-us/library/ms130214.aspx>.
- [26] Trygve R., Models - views – controller. http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf, 1979.
- [27] W3C. Simple object access protocol. <http://www.w3.org/TR/soap/>, Maj 2000.
- [28] W3C. Extensible markup language xml 1.0 fifth edition. <http://www.w3.org/TR/xml/>, 2008.