Uniwersytet w Białymstoku

Wydział Matematyczno-Fizyczny

Instytut Matematyki

Robert Rychliński

Zastosowanie języków skryptowych do budowy interfejsów aplikacji bazodanowych

Praca dyplomowa napisana
pod kierunkiem
dr. hab. Krzysztofa Prażmowskiego, prof. UwB

Białystok 2003

Składam serdeczne podziękowania
panu mgr. Mariuszowi Żynelowi
za cenne rady i wskazówki,
za pomoc w przygotowaniu
i realizacji tego projektu.

Robert Rychliński

Spis treści

Wstęp

Obraz dzisiejszego Internetu diametralnie różni się od obrazu sprzed kilku lat, gdy korzystanie z Sieci koncentrowało się głównie na przeglądaniu i wyszukiwaniu potrzebnych informacji. Wtedy Internet uważany był za kolejną ciekawostkę, jeszcze jedno medium komunikacyjne. Dzisiaj jego postrzeganie jest zupełnie inne - z pewnością bardziej poważne i zorientowane komercyjnie.

Nadzieje i oczekiwania pokładane w Internecie, a zwłaszcza z tym wszystkim, co określane jest mianem *e-commerce*, wciąż przyciągają żądnych wielkich zysków inwestorów, angażujących wielkie kapitały. Codziennie w lawinowym tempie rośnie liczba transakcji handlowych zawieranych za pośrednictwem globalnej sieci. E-commerce to nie tylko handel, to również coraz popularniejsze usługi oferowane przez wirtualne oddziały banków. W ciągu niespełna roku kilka polskich banków wprowadziło do swojej oferty usługę polegającą na prowadzeniu rachunków internetowych. Dzięki tej usłudze, bez wychodzenia z domu, możliwe jest przeprowadzanie wszelkiego rodzaju operacji na koncie ROR - wydawanie dyspozycji przelewu, dokonywanie stałych opłat, poznanie historii rachunku, itd. Ale Internet to nie tylko miejsce do pomnażania pieniędzy to również wielka skarbnica wiedzy i możliwości.

Nieograniczone możliwości Internetu zostały wykorzystane do stworzenia niniejszego projektu, który powstał w ramach dwóch prac dyplomowych: mojej i mgr Elżbiety Misiewicz pt.: *Projekt i implementacja systemu do układania harmonogramu zajęć*. Moim zadaniem w projekcie było stworzenie prostego i łatwego w obsłudze interfejsu graficznego oraz zadbanie o komunikację z bazą danych. Stworzona aplikacja skraca czas oraz zmniejsza wysiłek wkładany w tworzenie harmonogramu zajęć. Efektem pracy jest *Asystent wspomagający układanie harmonogramu zajęć*, dostępny pod poniższym adresem.

`http://theta.uwb.edu.pl/plan`

W chwili obecnej dostęp do zgromadzonych zasobów nie jest chroniony, jednak ze względu na charakter umieszczonych danych zostanie ograniczony, gdy aplikacja będzie wykorzystywana w praktyce.

Rozdział 1

Języki skryptowe

Początki internetu i serwisów internetowych to proste i statyczne strony HTML, całkowicie pozbawione interaktywności z użytkownikiem je oglądającym. Korzystanie z internetu opiera się na następującym mechaniźmie: przeglądarka (klient) nawiązuje połączenie z serwerem wysyłając żądanie pobrania określonego dokumentu; serwer odbiera żądanie, poszukuje odpowiedniego dokumentu na swoim dysku, pobiera go i wysyła w postaci strumienia tekstu do przeglądarki; przeglądarka interpretuje otrzymane dane i wyświetla w odpowiedniej postaci.

Dopóki światowe zasoby internetu były dostępne dla nielicznych i ilość danych, które publikowano w sieci była niewielka, rozwiązanie takie satysfakcjonowało wszystkich. Sytuacja zaczęła się zmieniać, gdy rozpoczęto prace na pierwszymi katalogami i wyszukiwarkami internetowymi - zaszła konieczność wprowadzenia interaktywności do stron i dynamicznego ich tworzenia, zaczęły pojawiać się coraz częściej strony zawierające formularze. Od tego momentu możemy mówić o rozwoju skryptowych języków server-side (czyli działających po stronie serwera). Ich ewolucja spowodowała, iż tak naprawdę oglądane przez nas strony internetowe są niczym innym jak wizualizacją odpowiedniej bazy danych, w zależności od preferencji zmieniają nie tylko zakres danych, ale kolor czy ułożenie modułów całej strony.

W odróżnieniu od typowych języków wysokiego poziomu, których kod źródłowy zamieniany jest przez kompilator na postać zrozumiałą bezpośrednio dla procesora i systemu operacyjnego, języki skryptowe do wykonania i uruchomienia odpowiedniego programu potrzebują - interpretera, który po załadowaniu i przeparsowaniu skryptu, rozpoczyna jego wykonywanie. Języki skryptowe pod względem sposobu pisania kodu, pozwalają na mniej zobowiązujący styl, brak w nich jawnego deklarowania stałych i zmiennych, nie posiadają jawnie zdefiniowanych typów danych.

Nie istnieje język uniwersalny, który nadawałby się do wszystkich zastosowań i w dodatku odpowiadał każdemu programiście. Wybierając język programowania, powinniśmy wziąć pod uwagę zarówno specyfikę danego projektu, jak i własne przyzwyczajenia i nawyki. W większości przypadków wybór ję-

zyka programowania jest kwestią gustu. Ponieważ producenci pomimo wielu różnic oferują podobne możliwości, mniej więcej tyle samo czasu musimy poświęcić na ich naukę oraz znajdziemy dla nich podobne ilości dodatkowych modułów. Jednak warto zapoznać się z możliwościami różnych języków dostępnych na rynku i dopiero wtedy zdecydować, który będzie dla nas i dla projektu najlepszy.

1.1 Meta-HTML

Meta-HTML: A Dynamic Programming Language for WWW Applications to stworzony przez Briana J. Fox dynamiczny język do budowy aplikacji WWW [4]. Jest to język który można stosować w przetwarzaniu symbolicznym na dużą skalę, posiada przy tym wiele wbudowanych funkcji, które oszczędzają czas potrzebny na przygotowanie projektu. Oto tylko najważniejsze cechy Meta-HTML:

- trwałe sesje (stateful sessions),
- łatwy dostęp do środowiska operacyjnego,
- system pakietów oparty na listach,
- strumienie plikowe i sieciowe,
- podstawowe operacje na relacyjnych bazach danych takich jak:
  - mSQL,
  - mySQL,
  - Oracle,
  - SyBase,
  - Informix,
  - oraz innymi kompatybilnymi z sterownikiem ODBC.
- możliwość pisania własnych makr i funkcji,
- łatwa i intuicyjna składnia zgodna z HTML, co uzasadnia nazwę Meta-HTML.

1.2 Perl

Perl jest jednym z najstarszych języków skryptowych używanym nie tylko do generacji stron WWW [2]. Jego autorem jest Larry Wall wspomagany oczywiście przez innych programistów. Perl powstał w 1987. Aby móc na swoim

komputerze uruchamiać skrypty Perla, trzeba mieć go zainstalowanego. Ponieważ jest on dostępny za darmo, pobranie go z Sieci i instalacja nie stanowi żadnego problemu. Głównym źródłem dokumentacji jest CPAN (Comprehensive Perl Archive Network) [2]. Można tam znaleźć: samego Perla (interpreter, źródła, wersje skompilowane), dokumentację, przykładowe skrypty, moduły (biblioteki).

Na stronie `http://www.perl.com/latest.html` znajdują się informacje o najnowszych wersjach Perla dla każdej platformy. Perl jest językiem prostym, jednak jego użycie przy bardziej zaawansowanych projektach jest dosyć uciążliwe, głównie przez dosyć niską czytelność kodu. Język ten charakteryzuje się bardzo mocnym zestawem funkcji służących do operacji na łańcuchach znakowych, także przy użyciu wyrażeń regularnych. Liczba bibliotek dostępnych dla Perla jest bardzo duża. Biblioteki te charakteryzują się dużą funkcjonalnością, ale nie są dostępne standardowo i nie są dobrze udokumentowane. Ostatnio jego popularność maleje, głównie na rzecz PHP. Aby mieć możliwość obsługi baz danych wystarczy zainstalować jeden z modułów Perla. Możemy wybrać jeden z nielicznie wymienionych tu modułów:

- BerkeleyDB,
- DBD,
- DBI,
- Msql,
- MSSQL,
- MySQL,
- Mysql,
- Netscape,
- Oracle,
- Postgres,
- Sybase, oraz wiele innych.

1.3 PHP

PHP powstał w 1994 roku nazwa pochodzi od, skrótu "PHP: Hypertext Preprocessor", jest szeroko używanym językiem skryptowym ogólnego zastosowania, dopasowanym do potrzeb aplikacji WWW, z możliwością zagnieżdżania w HTML. Jego składnia bazuje na językach C, Java i Perl. Celem tego języka,

jest umożliwienie twórcom serwisów WWW szybkiego pisania dynamicznych stron. PHP jest rozwijane pod kątem pisania skryptów *server-side* (działających po stronie serwera), więc można przy jego pomocy zrobić wszystko, co potrafią inne programy CGI, jak na przykład odbierać dane z formularzy, generować dynamicznie zawartość strony.

Jedną z najmocniejszych i najbardziej znaczących możliwości PHP jest obsługa wielu rodzajów baz danych. Obecnie obsługiwane są następujące bazy danych:

- Adabas D,
- dBase ,
- IBM DB2,
- Informix,
- InterBase,
- PostgreSQL,
- mSQL,
- Hyperwave,
- MS-SQL,
- MySQL,
- Oracle (OCI7 i OCI8),
- Sybase,
- Velocis,
- zgodnych z ODBC oraz wiele innych.

Istnieje także abstrakcyjne rozszerzenie DBX pozwalające na przezroczyste używanie dowolnej bazy danych obsługiwanej przez to rozszerzenie. Dodatkowo PHP obsługuje standard ODBC (Open Database Connection), przez co można połączyć się do dowolnej bazy danych obsługującej ten popularny standard.

Zarówno Meta-HTML, Perl jak i PHP powstały na zasadach Open Source (por. [6]). Wszystkie są dostępne za darmo wraz z kodem źródłowym.

Rozdział 2

Komunikacja z bazą danych

Asystent wspomagający układanie harmonogramów zajęć, który jest zasadniczą częścią tej pracy, wykorzystuje bazę danych MySQL do przechowywania danych. Tej właśnie bazie danych poświęcony jest ten rozdział. Opisujemy tutaj najczęściej wykorzystywane przy komunikacji z bazą danych funkcje języków skryptowych Meta-HTML, Perl i PHP.

Wszystkie trzy interpretery korzystają z jednego wspólnego interfejsu do bazy danych MySQL zaimplementowanego w bibliotece `libmysqlclient.so`. Powoduje to, że podobnie wyglądają interfejsy do bazy danych w odpowiednich trzech językach. O ile Perl i PHP używają wręcz identycznie brzmiących funkcji, będących przeniesieniem API bazy MySQL, to inaczej jest w Meta-HTML. Jest to, z jednej strony, spowodowane zupełnie inną syntaktyką języka i odmiennym sposobem notacji, z drugiej zaś, Meta-HTML dostarcza wyraźnie bogatszy zestaw podstawowych narzędzi do współpracy z bazami danych. Będzie to dalej widać, gdy podamy opisy przykładowych makr służących do komunikacji z bazami danych.

W dalszej części rozdziału rozważam typowe funkcje służące do nawiązania połączenia z bazą danych, wysłania zapytania i odebrania odpowiedzi, przeczytania i zapisu pojedynczego rekordu oraz zakończenia połączenia. Podaję dość dokładny opis makr Meta-HTML, gdyż jest to język używany w tworzonym projekcie, oraz odpowiedniki tych makr w Perlu i PHP. Poniższy tekst nie powinien być jednak traktowany jako podręcznik do nauki obsługi bazy danych w Perlu czy PHP.

2.1 Otwarcie połączenia

Niezależnie od języka w którym piszemy skrypt, aby połączyć się z bazą danych MySQL należy przekazać nazwę konkretnej bazy, nazwę komputera, na którym ona się znajduje, identyfikator i hasło użytkownika. W Meta-HTML zanim otworzymy połączenie z bazą musimy sprecyzować typ bazy danych, tak aby interpreter dobrał odpowiedni sterownik. Aby połączyć się z bazą danych

MySQL wołamy:

```
<sql::set-database-type TYPE>
```

gdzie jako `TYPE` podajemy `mysql`. Nawiązanie połączenia z bazą danych MySQL następuje po wywołaniu:

```
<sql::with-open-database DBVAR &key [DSN=DSN-STRING]
   [NOLOCK=TRUE]>
      body
</sql::with-open-database>
```

gdzie `DSN` (Database Service Name) jednoznacznie określa bazę danych, z którą się łączymy. Zmienna podana w `DBVAR` przyjmuje wartość *uchwytu*[1] do nawiązanego połączenia z bazą danych.

Łańcuch `DSN` ma następującą składnię:

```
HOST=hostname;DATABASE=dbname;USER=username;PASSWORD=passwd
```

gdzie `hostname` określa nazwę serwera bazy danych, `dbname` nazwę bazy danych, `username` nazwę użytkownika i `passwd` jego hasło.

Jeśli podano `NOLOCK=TRUE` wówczas dostęp do bazy danych nie jest blokowany przed innymi procesami, na czas otwieranego połączenia.

W Perlu, aby nawiązać połączenie z bazą piszemy:

```
$dbh = DBI->connect("DBI:mysql:
                     host=hostname:database=dbname",
                     "username", "passwd");
```

natomiast w PHP:

```
$dbh = mysql_connect("hostname", "dbname",
                     "username", "passwd");
```

Uchwyt do otwartego połączenia w obu przypadkach przechowywany jest w zmiennej `$dbh` (skrót od *database handle*)[2].

Jak widać wszystkie trzy opisywane funkcje są bardzo podobne, a przekazywany DSN jest identyczny, co wymuszone jest przez API bazy MySQL.

Dobrym przyzwyczajeniem jest zamknięcie połączenia do bazy po zakończeniu pracy. W przeciwieństwie do Perl'a, czy PHP, gdzie nie ma obowiązku zamykania połączenia, w Meta-HTML rozłączenie następuje w znaczniku zamykającym `</sql::with-open-database>`, bez którego nie zadziała makro `sql::with-open-database`. W przypadku Perl'a i PHP, jeśli sami nie

[1] Z ang. *handle*, w tym wypadku zmienna identyfikująca bazę danych. Z punktu widzenia systemu operacyjnego to identyfikator strumienia pomiędzy lokalnym komputerem (klientem) a serwerem bazy danych. Oczywiście jeden komputer może pełnić dwie role.

[2] W Perlu uchwyt do połączenia z bazą, jak również opisywany dalej uchwyt do tabeli odpowiedzi, jest obiektem. Do jego metod odwołujemy się pisząc `$zmienna->metoda()`. W tej sytuacji mówimy o "poleceniu", aby nie nadużywać określenia "funkcja".

zamkniemy otwartego połączenia, zrobi to za nas system operacyjny po zakończeniu procesu, ale nie jest to eleganckie rozwiązanie.

W Perlu zamknięcie połączenia wykonuje się przy pomocy polecenia:

```
$dbh->disconnect();
```

a w PHP:

```
mysql_close($dbh);
```

2.2 Wywołanie zapytania SELECT

Wywołanie zapytania typu SELECT, SHOW, EXPLAIN i DESCRIBE powinno odbyć się po nawiązaniu połączenia z bazą. W Meta-HTML zapytania takie realizuje:

```
<sql::database-query DBVAR EXPR QUERY &key
                     [COLNAMES=NAMELIST]
                     [PREFIXTABLENAMES=TRUE]
                     [FORMAT=FEXPR]
                     [KEYS=VARNAME]
                     [KEYNAME=FIELDNAME]
                     [WINDOW-START]
                     [WINDOW-LENGTH]>
```

Argumenty `DBVAR`, `EXPR` i `QUERY` są obowiązkowe. Pozostałe argumenty są opcjonalne i aby użyć któregoś z nich należy podać nazwę argumentu i przypisać jej wartość.

W `DBVAR` określamy uchwyt do otwartego połączenia z bazą danych.

Polecenie przekazuje zapytanie w języku SQL jako łańcuch `QUERY` do bazy danych. W odpowiedzi uzyskuje tabelę (ewentualnie pustą), której wiersze przetwarzane są jeden po drugim.

Jeśli podana jest lista nazw kolumn `COLNAMES`, zamiast nazw kolumn z tabeli używane są nazwy z tej listy tak, że pierwszej kolumnie odpowiada pierwszy element listy. Wartości pól przetwarzanego wiersza z tabeli odpowiedzi dostępne są w `EXPR` i `FORMAT` w postaci zmiennych o nazwach takich jak nazwy kolumn, lub nazwach z listy `COLNAMES`.

Użycie `PREFIXTABLENAMES=TRUE` oznacza, że nazwy kolumn będą prefiksowane nazwami tabeli.

Wyrażenie `EXPR` jest wyrażeniem w języku Meta-HTML. Może ono zawierać odwołania do wartość pól. Jeśli zwraca ono niepustą wartość dla danego rekordu, to jest on dalej przetwarzany w `FORMAT` oraz `KEYS`.

Jeśli podano `FORMAT`, wyrażenie `FEXPR` jest interpretowane w kontekście pól z wiersza odpowiedzi, podobnie jak `EXPR`, a jego wynik zwracany jest przez `sql::database-query`.

Gdy przekazano `KEYS`, w zmiennej `VARNAME` umieszczona zostanie lista wartości pól (kluczy) z kolumny określonej przez `KEYNAME`. Tak więc obecność `KEYS` wymusza określenie `KEYNAME`.

W sytuacji, gdy nie podano żadnego z argumentów `FORMAT`, `KEYS`, makro `sql::database-query` zwraca listę kluczy oddzielonych znakami nowej linii.

Parametr `WINDOW-START` określa pierwszy wiersz z tabeli odpowiedzi, który należy przetwarzać, natomiast `WINDOW-LENGTH` określa maksymalną ilość wierszy do przetworzenia. Te dwa parametry pomagają podzielić duże tabele na mniejsze porcje i wyświetlać je stronami.

Przedstawiona funkcja Meta-HTML nie ma swego odpowiednika w Perlu i PHP. Próbowaliśmy napisać własną funkcję w Perlu, która pełniłaby rolę makra `sql::database-query`, ale dość szybko okazało się, że to skomplikowane zadanie, być może temat na inną pracę dyplomową.

2.3 Inne zapytania SQL

Pracując z bazą danych poza zapytaniami SELECT często zachodzi potrzeba użycia innych zapytań, takich jak UPDATE, DELETE lub zapytań administracyjnych, dotyczących samej bazy danych.

W API MySQL wyraźnie rozdzielono dwa typy zapytań:

- zapytania, które zwracają jakieś dane z bazy,
- zapytania, które nie zwracają żadnych danych.

Do ich obsługi służą dwie różne funkcje. Pierwsza z nich wykorzystywana jest przez opisane wcześniej makro `sql::database-query`. Z drugiej korzysta

```
<sql::database-exec-sql DBVAR QUERY>
```

Makro to wykonuje przekazane w łańcuchu `QUERY` polecenie SQL na otwartym połączeniu wskazywanym przez `DBVAR`. Jeżeli polecenie zostanie wykonane poprawnie, zwraca jest wartość `TRUE`, w przeciwnym razie pusty ciąg znaków. Pusty ciąg znaków jest w Meta-HTML odpowiednikiem `FALSE` używanego w innych językach.

Odpowiednikiem tego makra w Perlu jest instrukcja:

```
$result = $dbh->do(QUERY);
```

gdzie QUERY to łańcuch tekstowy zawierający dowolne zapytanie SQL, które nie zwraca tabeli odpowiedzi. Zmienna `$result` przyjmuje wartość równą liczbie wierszy, których dotyczy zapytanie QUERY, lub -1 gdy wystąpił błąd.

W PHP uproszczono kwestię wysyłania zapytań. Dostępna tam funkcja `mysql_query()` jest uniwersalna. Jako argument pobiera treść zapytania SQL i opcjonalnie identyfikator otwartego połączenia z bazą. Odpowiednikiem wyżej przedstawionej instrukcji w Meta-HTML i Perlu jest:

```
$result = mysql_query(QUERY, $dbh);
```

W przypadku zapytań typu SELECT zwracany jest uchwyt do tabeli odpowiedzi, natomiast gdy zapytanie jest drugiego typu, zwracane jest TRUE lub FALSE w zależności od tego, czy realizacja kwerendy powiodła się, czy też nie.

2.4 Dostęp do pojedynczych rekordów

W pewnych sytuacjach wygodnie jest mieć możliwość pobrania z bazy danych lub zapisania w bazie pojedynczego rekordu, szczególnie wtedy, gdy znamy unikalny klucz określający jednoznacznie taki rekord.

Do przeczytania z tabeli wskazanego rekordu w Meta-HTML służy makro:

```
<sql::database-load-record DBVAR KEY
                           &key TABLE=TABLENAME
                                KEYNAME=FIELDNAME
                                [PACKAGE=PACKAGENAME]>
```

Podobnie jak wcześniej, `DBVAR` określa otwarte połączenie z bazą danych i jest wymaganym argumentem. Drugim wymaganym argumentem jest `KEY`, w którym podajemy wartość pola identyfikującego rekord. W przypadku bazy MySQL jest to zazwyczaj wartość z kolumny z atrybutem *primary-key*.

Makro pobiera jeden rekord z bazy danych, wskazany przez `KEY` z tabeli określonej przez `TABLENAME`. Wartość `KEY` będzie wyszukiwana w kolumnie o nazwie `FIELDNAME`.

Opcjonalnie możemy podać nazwę pakietu, do którego zostanie zapisany pobrany rekord. Nazwy zmiennych w pakiecie `PACKAGENAME` są takie same jak nazwy kolumn w tabeli w bazie. Jeśli argument ten zostanie pominięty, załadowany rekord trafi do pakietu domyślnego (z prefixem `DEFAULT`).

Zapis pojedynczego rekordu odbywa się przy użyciu makra:

```
<sql::database-save-package DBVAR KEY PACKAGE
                            &key TABLE=TABLENAME
                                 KEYNAME=FIELDNAME>
```

Makro zapisuje zmienne z pakietu `PACKAGE` w tabeli `TABLENAME` w wierszu określonym przez wartość `KEY` z kolumny `FIELDNAME`.

Zadanie zapisania rekordu realizowane jest w ten sposób, że najpierw wywoływane jest zapytanie typu INSERT, a gdy ono się nie powiedzie wołane jest zapytanie UPDATE, gdzie `FIELDNAME` ustawione jest na `KEY`. INSERT nie powiedzie się, gdy rekord o podanym kluczu istnieje, tak więc zapis rekordu ma wówczas charakter aktualizacji UPDATE.

W przypadku bazy MySQL, gdzie kolumnie *primaty-key* typu całkowitego można dodatkowo nadać atrybut *auto-increment*, gdy wstawiamy nowy

rekord, jako wartość `KEY` można przekazać NULL. MySQL zadba o nadanie unikalnej wartości klucza [3] do tego rekordu.

Jak zwykle `DBVAR` określa otwarte połączenie z bazą danych.

Zapis rekordu jest wykonywany poprzez INSERT/UPDATE dlatego, że w API MySQL nie ma funkcji zapisu pojedynczego rekordu. Prawdopodobnie, z tego samego powodu Perl i PHP nie dostarczają gotowych funkcji, przy pomocy których łatwo można by odczytywać i zapisywać pojedyncze rekordy na podstawie podanego klucza.

2.5 Czytanie tabeli wierszami

Do wysłania zapytania typu SELECT i odebrania odpowiedzi wystarczające jest w zupełności makro `sql::database-query`. Meta-HTML dostarcza dodatkowo zestaw kilku makr do obsługi zapytań, które zwracają dane. Makra te są bliższe idei API MySQL i przypominają funkcje Perl'a i PHP.

```
<sql::database-exec-query DBVAR QUERY
                          &key [CURSOR=VARNAME]>
```

Funkcja wykonuje polecenie zawarte w `QUERY` na otwartym połączeniu z bazą wskazywanym przez uchwyt `DBVAR`. Zmienna `VARNAME` przyjmuje wartość uchwytu do tabeli odpowiedzi. Będzie on wykorzystany w dalej opisywanych makrach.

Odpowiednikiem tego makra w Perlu jest wywołanie poleceń:

```
$sth = $dbh->prepare(QUERY);
$sth->execute();
```

gdzie QUERY to stała lub zmienna tekstowa zawierająca zapytanie SQL, a `$dbh` to zmienna zawierająca uchwyt do połączenia z bazą. Zmienna `$sth` pełni tutaj rolę uchwytu do odpowiedzi (skrót od *statement handle*).

W PHP rolę `sql::database-exec-query` pełni, opisana wcześniej, uniwersalna funkcja `mysql_query()`.

Makro Meta-HTML:

```
<sql::affected-rows CURSOR>
```

zwraca liczbę rekordów przetworzonych podczas ostatniej operacji na bazie danych, związanej z uchwytem `CURSOR`.

W PHP przy pomocy funkcji `mysql_num_rows()` można sprawdzić ilość wierszy zwróconych w tabeli odpowiedzi na zapytanie typu SELECT, natomiast funkcja `mysql_affected_rows()` zwraca ilość wierszy przetworzonych w wyniku zapytania różnego od SELECT.

W Perlu, w pierwszym wypadku należy pobrać wynik jako listę i sprawdzić jej długość, w drugi zaś można użyć polecenia `$sth->affected_rows`.

[3] Wartość tę można odczytać z MySQL poprzez zapytanie `select last_insert_id()` bezpośrednio po zapisie danych.

```
<sql::database-next-record CURSOR
                           &key [COLNAMES=NAMELIST]
                                [PREFIXTABLENAMES=TRUE]
                                [PACKAGE=PACKAGENAME]>
```

Makro pobiera pojedynczy wiersz (rekord) z tabeli odpowiedzi poprzez odwołanie do zmiennej `CURSOR`. Wynik zapisuje w postaci listy asocjatywnej oraz zwraca `TRUE` jeżeli w tabeli odpowiedzi pozostały jeszcze rekordy. Opcjonalnie możemy podać nazwy kolumn jako listę w zmiennej `NAMELIST`, co pozwoli na odwoływanie się do wartość pola z pobranego rekordu poprzez te nazwy, nie zaś prawdziwe nazwy kolumny w bazie. Jeżeli podamy nazwę pakietu `PACKAGENAME`, pobrane wartości zostaną w nim zapisane, podobnie jak w przypadku `sql::database-load-record`.

Odpowiednikiem `sql::database-next-record` w Perlu są następujące trzy polecenia:

```
$sth->fetchrow_array()
$sth->fetchrow_arrayref()
$sth->fetchrow_hashref()
```

Pierwsze zwraca listę wartość poszczególnych pól wiersza, drugie zwraca wskaźnik do takiej listy, trzecie zwraca wskaźnik do listy asocjatywnej. Ostatnia funkcja jest dość wygodna, ponieważ do pól odwołujemy się poprzez nazwy kolumn z tabeli. Poza wymienionymi trzema funkcjami jest jeszcze jedna funkcja `fetchall_arrayref()`, która zwraca wskaźnik do listy list komórek z tabeli odpowiedzi, innymi słowy zwraca całą tabelę dwuwymiarową.

W PHP jest bardzo podobnie. Mamy to do dyspozycji następujące funkcje:

```
mysql_fetch_object()
mysql_fetch_row()
mysql_fetch_assoc()
mysql_fetch_array()
```

które jako argument pobierają identyfikator (uchwyt) zapytania a zwracają odpowiednio:

1. obiekt, którego właściwości zawierają pobrany wiersz,
2. tablicę wyliczeniową, do której elementów można odwoływać się przez indeksy,
3. tablicę asocjatywną, do której elementów odwołujemy się poprzez nazwy kolumn,
4. tablicę, do której elementów można odwoływać się przez indeksy lub nazwy kolumn.

W Meta-HTML makro

```
<sql::set-row-position CURSOR POS>
```

ustawia pozycję wiersza w tabeli odpowiedzi, wskazywanej przez zmienną `CURSOR`, na wartość `POS`. W Perlu zadanie to wykonuje polecenie

```
$sth->dataseek(POS)
```

a w PHP:

```
mysql_data_seek(POS)
```

2.6 Debugging

Podczas pisania skryptów, nie tylko w Meta-HTML, ważne są funkcje pozwalające diagnozować błędy, czyli ułatwiające debugging i uruchamianie.

```
<sql::recent-query>
```

To makro wykorzystywane jest do sprawdzania poprawności poleceń wysyłanych do bazy danych. W odpowiedzi zwraca treść ostatnio wysłanego do bazy zapytania, w postaci tekstu zgodnego ze składnią SQL, co umożliwia przetestowanie tego zapytania przy pomocy innych narzędzi, np. konsoli MySQL.

Komunikaty o ewentualnych błędach w zapytaniu lub problemach z jego realizacją zwracane są poprzez ogólne makro

```
<debugging-output>
```

W Perlu numer błędu związanego z realizacją zapytania możemy odczytać przy pomocy polecenia `$dbh->err`, a pełny komunikat przy pomocy polecenia `$dbh->errstr`. W PHP do tego celu służą odpowiednio funkcje `mysql_errno()` oraz `mysql_error()`.

Rozdział 3

Technologie wykorzystywane w budowie aplikacji internetowych

3.1 JavaScript

JavaScript (JS) to język skryptowy przystosowany do tworzenia interaktywnych stron WWW. Skrypty (czyli proste, niekompilowane programy) są najczęściej zagnieżdżane w kodzie HTML - połączenie JS i HTML jest często nazywane "dynamiczny HTML" - DHTML. Składnia JS jest podobna do Javy, ale w przeciwieństwie do tego języka skrypty JS nie są kompilowane - kod źródłowy JavaScript działa po stronie klienta (jest interpretowany przez przglądarkę), w odróżnieniu od dotąd omawianych Meta-HTML, PHP i Perl. Dzięki kompletowi wbudowanych obiektów udostępnianych przez przeglądarkę JS może komunikować się z użytkownikiem, manipulować zawartością stron, pomagać w nawigacji itd.

W dalszej części tego rozdziału przedstawiam elementy interfejsu *Asystenta wspomagającego układanie harmonogramów zajęć*. Są to elementy formularzy HTML specyficzne dla tego projektu. Mogą one jednak znaleźć szersze zastosowanie. Opisuję tutaj również technologie weryfikacji poprawności danych oraz kontroli dostępu do danych zastosowane w projekcie.

3.1.1 Trójwartościowy Checkbox

HTML pozwala nam przekazywać informacje z przeglądarki do serwera za pomocą formularzy. W standardzie przewidziane są następujące elementy takich formularzy:

- pole tekstowe (text),
- pole wprowadzania hasła (password),
- duże pole tekstowe (textarea),

- przycisk radiowy (radio),
- pole wyboru (checkbox),
- przycisk wysyłający dane (submit),
- przycisk zerujący (reset),
- przycisk (button),
- lista wyboru (select),
- wybór pliku (file),
- element ukryty (hidden).

Wstawiając element do formularza określa się jego nazwę, tak aby była unikalna, możemy również zadeklarować jego wartość. Podana wartość elementu przekazywana jest do serwera jako para "nazwa=wartość".

W projekcie wspomagania układania harmonogramu zajęć potrzebowaliśmy taki element formularza, który przyjmowałby jedną z możliwych trzech wartości. Każdy z nauczycieli ma możliwość złożenia swoich życzeń co do planu zajęć, to znaczy, kiedy może prowadzić zajęcia, kiedy nie może lub kiedy nie sprawia mu to różnicy. Wiedza tego typu jest potrzebna do układania planu ponieważ niektórzy z pracowników naszego Instytutu dojeżdżają lub też prowadzą zajęcia na innych uczelniach.

Problem można rozwiązać stosując grupę trzech przycisków radiowych lub listę wyboru. Pierwsze rozwiązanie jest nieodpowiednie ze względu na ilość zajmowanego miejsca. Otóż potrzebujemy umieścić 60 takich trójek przycisków radiowych. Drugie rozwiązanie z kolei ma tę wadę, że wymaga od użytkownika rozwijania listy przy podejmowaniu wyboru, co znacznie wydłuża czas poświęcony na wypełnianie formularza.

Przyjęliśmy rozwiązanie, które wykorzystuje możliwość wymiany elementu graficznego. Element ten wstawiany jest do formularza w następujący sposób:

```
<a href="javascript:klik('<get-var aID>')"><img
  src=<downcase <get-var posted::<get-var aID>>>.gif
    name=<get-var aID> border=0></a>
<input type=hidden name=<get-var aID>
  value="<get-var posted::<get-var aID>>">
```

Jak widać element graficzny `IMG` sprzężony jest z elementem ukrytym (hidden). Oba te elementy mają przypisaną tę samą nazwę. Element graficzny reprezentuje wartość, która jest przechowywana w formularzu. Synchronizację między tymi dwoma elementami zapewnia funkcja `klik()`:

```
function klik(p){
  if (document[p].src.match('o.gif')) {
```

```
        document[p].src = 'n.gif';
        document['tabelka'][p].value = 'N';
        return;
      }
      if (document[p].src.match('n.gif')) {
        document[p].src = 't.gif';
        document['tabelka'][p].value = 'T';
        return;
      }
      if (document[p].src.match('t.gif')) {
        document[p].src = 'o.gif';
        document['tabelka'][p].value = 'O';
        return;
      }
    }
```

Funkcja ta jako argument `p` pobiera identyfikator obrazka. Jest ona wywoływana w momencie kliknięcia na obrazek, co powoduje zamianę obrazka na kolejny i w polu formularza ustawia jego wartość (odpowiednio, kolor czerwony – wartość "N" – "nie mogę prowadzić zajęć", kolor niebieski – wartość "T" – "chętnie w tym terminie poprowadzę zajęcia", kolor czarny – wartość "N" – "jest mi obojętne").

Rysunek 3.1 przedstawia formularz, w którym zastosowano opisywany element.

Godziny	Poniedziałek	Wtorek	Środa	Czwartek	Piątek	Godziny
8.15-9.00	●	●	●	●	●	8.15-9.00
9.15-10.00	●	●	●	●	●	9.15-10.00
10.15-11.00	●	●	●	●	●	10.15-11.00
11.15-12.00	●	●	●	●	●	11.15-12.00
12.15-13.00	●	●	●	●	●	12.15-13.00
13.15-14.00	●	●	●	●	●	13.15-14.00
14.15-15.00	●	●	●	●	●	14.15-15.00
15.15-16.00	●	●	●	●	●	15.15-16.00
16.15-17.00	●	●	●	●	●	16.15-17.00
17.15-18.00	●	●	●	●	●	17.15-18.00
18.15-19.00	●	●	●	●	●	18.15-19.00
19.15-20.00	●	●	●	●	●	19.15-20.00

Rysunek 3.1: Źródło http://theta.uwb.edu.pl/plan/zyczenia/tabelka.mhtml

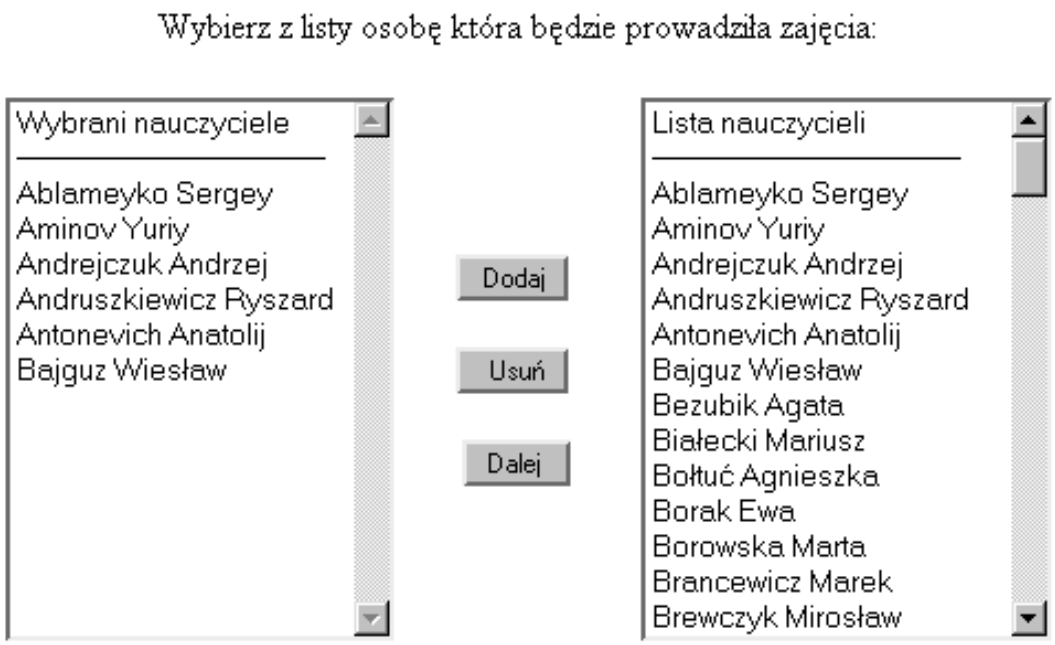
Aby uprościć wypełnianie formularza przedstawionego na rysunku 3.1 napisano dodatkowo dwie funkcje:

```
function klikDzien(i) {
  var r = 'k1-'+i;
  for (var g = 2; g <= 12; g++) {
    p = 'k'+g+'-'+i;
    document[p].src = document[r].src;
    document['tabelka'][p].value =
      document['tabelka'][r].value;
  }
}
function wyczysc() {
  for (var i = 1; i <= 5; i++) {
    for (var g = 1; g <= 12; g++) {
      p = 'k'+g+'-'+i;
      document[p].src = 'o.gif';
      document['tabelka'][p].value = 'O';
    }
  }
}
```

Funkcja `klikDzien()` pobiera jako argument liczbę całkowitą, która odpowiada kolejnemu dniu tygodnia. Działa ona w ten sposób, że wartości w godzinach 9:15 - 20:00 są ustawiane na takie jak wartość o godzinie 8:15 - 9:00. Natomiast funkcja `wyczysc()` odpowiada elementowi `reset` formularza. Ustawia ona wszystkie wartości w całej tabeli na "O".

3.1.2 Lista wybierana z listy

W czasie tworzenia obsady, czyli gdy ustala się kto będzie prowadził określone zajęcia, połączona zostaje nazwa przedmiotu z osobą prowadzącego. Z powodu dużej ilości studentów, podziału na specjalności oraz grupy, w praktyce oznacza to, że dany przedmiot prowadzony jest przez kilka osób. Dobrze to widać na przykładzie ćwiczeń z `Analizy matematycznej I`, które prowadzone są na pierwszym roku każdej ze specjalności i do prowadzenia zajęć wyznaczonych jest kilka osób. Tak więc przy obsadzie wybieramy z listy wszystkich nauczycieli, listę nauczycieli przypisanych do ustalonego rodzaju zajęć z danego przedmiotu.



Rysunek 3.2: Źródło http://theta.uwb.edu.pl/plan/obsada/obsada.mhtml

HTML nie posiada dobrych narzędzi, które pozwoliłyby na wybranie kilku elementów z danej listy. Zgodnie ze specyfikacją HTML 4.01, jeżeli chcemy mieć możliwość zaznaczenia więcej niż jednego elementu na liście wyboru typu `select` należy w jej deklaracji dodać argument `multiple`. Niestety to nie specyfikacja HTML decyduje o tym jak napiszemy aplikację, lecz jak dany kod HTML zostanie zinterpretowany przez przeglądarki. W praktyce większość przeglądarek internetowych źle interpretuje opcję `multiple`.

Problem wyboru podlisty został rozwiązany za pomocą grupy dwóch list wyboru (patrz rys. 3.2) (elementów formularzy typu `select`) powiązanych ze sobą programem napisanym w języku JS. Przypomnijmy (por. 3.1.1), że element formularza typu `select`, to nic innego jak rozwijalne menu z możliwością wyboru jednej z umieszczonych opcji.

Zawartość listy `Lista nauczycieli` pobierana jest z bazy danych. Pobierane są imię, nazwisko oraz unikalny numer każdego z nauczycieli. Na ekranie wyświetlane jest imię i nazwisko, natomiast przekazywany z formularza do obróbki jest identyfikator. Za dodawanie i usuwanie elementów na liście `Wybrani nauczyciele` odpowiedzialne są funkcje `dodaj()` i `usun()`:

```
function dodaj(f) {
    var i = f.lista.selectedIndex;
    var t = f.lista.options[i].text;
    var v = f.lista.options[i].value;
    var j = 2;
```

```
    // Sprawdza czy nie jest żju wybrany
    while (j < f.wybor.options.length &&
                f.wybor.options[j].value != v) {
        j++;
    }
    if (j != f.wybor.options.length) {
            return;
    }
    // Dopisz do listy, o ile nie jest 0 lub 1
    if (1 < i) {
        f.wybor.options[f.wybor.options.length] =
            new Option(t,v);
    }
}

function usun(f) {
    var i = f.wybor.selectedIndex;

    // Usuwaj tylko wtedy gdy nie jest 0 i 1
    if (1 < i) {
        f.wybor.options[i] = null;
    }
}
```

Funkcja `dodaj()` wołana jest po naciśnięciu *Dodaj*. W efekcie zaznaczone na liście `Lista nauczycieli` imię i nazwisko, pojawią się na liście `Wybrani nauczyciele`. Funkcja `usun()` wołana jest po naciśnięciu *Usuń* i usuwa wybranego nauczyciela z listy `Wybrani nauczyciele`. Obie funkcje nie robią nic, gdy zostały wybrane dwie pierwsze pozycje na listach, które zostały umieszczone w celu uproszczenia obsługi formularza. W `dodaj()` zadbano również o to, aby nie dodać jednej osoby wielokrotnie.

Ponieważ z listy wyboru możemy przesłać tylko jeden element – ten zanzaczony – potrzebne jest jeszcze jedno pole pomocnicze w formularzu oraz funkcja do jego obsługi. Pole jest typu `hidden` i nazywa się `wybrani` natomiast funkcja nazywa się `fixSelect()`:

```
function fixSelect(f) {
    f.wybrani.value = '';

    for (var i = 2; i < f.wybor.options.length; i++)
        f.wybrani.value += f.wybor.options[i].value + ',';
    }
    return true;
}
```

Funkcja `fixSelect()` przepisuje wartości pól, czyli identyfikatory nauczycieli (zaczynając od pola trzeciego o indeksie 2) z listy `Wybrani nauczyciele` do pola `wybrani` oddzielając kolejne identyfikatory przecinkami. Funkcja wołana jest w momencie naciśnięcia *Dalej*, czyli tuż przed wysłaniem formularza dalej do obróbki.

Efekt działania opisanych funkcji możemy zobaczyć na stronie:

`http://theta.uwb.edu.pl/plan/obsada/`

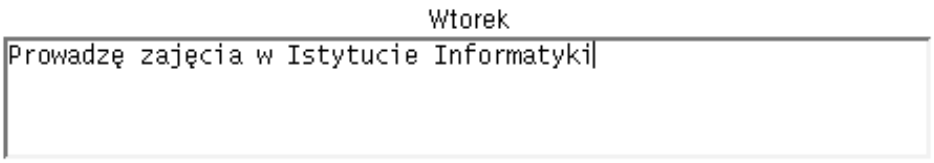
3.1.3 Walidatory w JavaScript

Ważnym zastosowaniem JavaScript w budowie aplikacji WWW jest programowanie tzw. *walidatorów*. Przy wysyłaniu danych z formularza do serwera pojawia się pytanie o poprawność tych danych. Aby na przykład zapewnić jednoznaczne wprowadzenie daty do bazy danych ustala się format wprowadzania daty. Przed zapisem jednak należy się upewnić, czy wprowadzony tekst jest w ogóle datą, to znaczy, czy odpowiada ustalonemu formatowi. Co więcej można sprawdzić, czy liczba miesiąca jest z przedziału $[1, 12]$, czy nie jest przekroczona liczba dni w danym miesiącu. Aby odciążyć serwer od wykonywania dodatkowej pracy i skrócić czas potrzebny na prawidłowe wypełnienie formularza, program sprawdzający poprawność danych, czyli walidator, wykonuje się po stronie klienta (w przeglądarce).

Walidatory formularzy można podzielić na dwie kategorie:

- sprawdzające poprawność danych na bieżąco, w trakcie wypełniania formularza,
- sprawdzające poprawność danych tuż przed ich wysłaniem, po kliknięciu `submit`.

Przykładem walidatora, który został zastosowany w aplikacji wspomagającej układanie harmonogramu zajęć, jest program sprawdzający czy podane uzasadnienie (por. rys. 3.3) dotyczące preferencji nauczyciela nie jest zbyt krótkie.



Rysunek 3.3: Formularz do wprowadzania uzasadnienia.

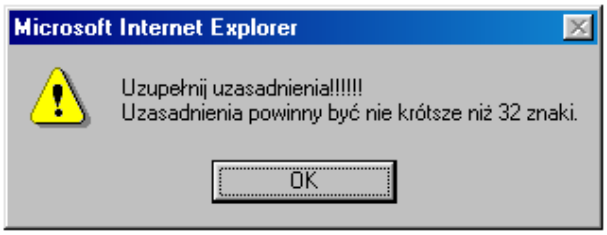
```
function validate(f) {
 for (var i=1; i <= 5; i++){
```

```
        if (typeof(f['u'+i]) == 'object'
            && f['u'+i].value.length < 32) {
alert('łUzupenij uzasadnienia !\n
 Uzasadnienia powinny ćby nie ókrtsze żni 32 znaki.')
                              return false
                    }
          }   return true
    }
```

Funkcja `validate()` jako argument pobiera wskaźnik do formularza. Następnie sprawdza dwa warunki:

- czy w formularzu znajduje się pole o nazwie `ui`, gdzie `i` jest liczbą całkowitą z przedziału $[1, 5]$ (traktowaną jako numer dnia),
- czy ciąg znaków zapisany jako wartość w polu o nazwie `ui` jest krótszy niż 32 znaki.

Wymienione warunki połączone są ze sobą logicznym spójnikiem koniunkcji `&&`. Jeżeli oba są spełnione, czyli gdy podano uzasadnienie krótsze niż 32 znaki, po naciśnięciu *Zapisz*, pojawi się komunikat przedstawiony na rysunku 3.4.



Rysunek 3.4: Ostrzeżenie o zbyt krótkim uzasadnieniu

Jeżeli oba warunki zostaną spełnione `validate()` zwróci wartość logiczną `TRUE`, co pozwoli zapisać informacje do bazy danych.

Należy jednak pamiętać, że użytkownik może wyłączyć obsługę JavaScript w swojej przeglądarce. Poza tym nie wszystkie przeglądarki mają obsługę JavaScript. Dlatego dobrze zbudowany walidator powinien zawierać zarówno elementy sprawdzające zawartość formularz po stronie klienta jak również po stronie serwera. Na przykład, za pomocą `JavaScript`, można sprawdzić, czy podany w polu formularza adres e-mail, jest poprawnie sformułowany. Serwer może dodatkowo wysłać wiadomość pod ten adres z prośba o uwierzytelnienie przesłanych danych. Metodę tą często stosują sklepy internetowe.

W przypadku uzasadniania życzeń funkcja `JavaScript` działa po stronie przeglądarki, natomiast następujący kod sprawdza poprawność postronne serwera:

```
<set-var tmp::OK=true>
<foreach D list::Dni iter=I>
  <when <get-var posted::u<add I 1>>>
    <set-var tmp::OK=<and <get-var tmp::OK>
       <lt 32 <string-length
              <get-var posted::u<add I 1>>>>>>>
  </when>
</foreach>
```

Jeżeli po wykonaniu pętli `foreach` zmienna `tmp::OK` będzie miała wartość `TRUE`, to informacje zostaną zapisane w bazie danych. W przypadku niepowodzenia, użytkownik dostanie komunikat, że musi uzupełnić uzasadnienia.

3.2 Zabezpieczanie zasobów

Bezpieczeństwo i ochrona danych są ważnym aspektem przy budowie aplikacji bazodanowej. Nikt chyba niechciał by, aby informacje prywatne dostały się w niepowołane ręce. Dlatego też należy zadbać o pakiet zabezpieczeń realizujący następujące funkcje:

Uwierzytelnianie Aby kontrolować dostęp do zasobów sieciowych, interfejs musi mieć przede wszystkim możliwość identyfikacji użytkowników. Innymi słowy, konieczne jest upewnienie się, czy użytkownik jest rzeczywiście tym, za kogo się podaje.

Autoryzacja Po zidentyfikowaniu lub uwierzytelnieniu, a przed udzieleniem dostępu do zasobów sieci, użytkownik powinien zostać poddany autoryzacji. Proces ten musi być tak rozbudowany, aby na podstawie ustalonych reguł oraz profilu użytkownika można było dokładne określić, do jakich zasobów użytkownik ma prawa dostępu.

Audyt Nawet gdy istnieje infrastruktura zabezpieczeń, niezbędny jest pakiet zabezpieczeń monitorujący środowisko sieciowe pod kątem ataków i nadużyć oraz informujący administratora systemu o wszelkich próbach naruszenia bezpieczeństwa. Pakiet zabezpieczeń powinien także obejmować usługi zapewniające tzw. niezaprzeczalność (non-repudiation), które umożliwiają śledzenie transakcji i potwierdzanie, czy określone działanie jest wykonywane przez osobę upoważnioną, a nie przez kogoś, kto się pod nią podszywa.

Administrowanie Pakiet zabezpieczeń musi być łatwy i wygodny w zarządzaniu z dowolnego miejsca przez osoby do tego powołane.

Cookies i sesje

Jednym ze sposobów identyfikacji klienta (przeglądarki) są *ciasteczka*, bynajmniej nie są to słodycze.

Cookies (z ang. ciasteczka) wysyłane są przez serwer WWW do przeglądarki w postaci nagłówka HTTP. Serwer wysyłając:

```
Set-cookie: SID=34201137716009; path=/;
  expires=Fri, 31 Dec 2010 19:59:59 GMT
```

prosi klienta o zapamiętanie zmiennej SID o podanej wartości. Przeglądarka zapisuje sobie ciasteczka w formie plików tekstowych (Internet Expolorer) lub bazy danych (Mozilla/Netscape). Następnie w czasie każdego połączenia ze stroną, z której pochodzi dane Cookie, przeglądarka do żądania dodaje nagłówek:

```
Cookie: SID=34201137716009
```

Cookie mogą zawierać rozmaite rodzaje informacji o użytkowniku danej strony WWW i "historii" jego łączności z daną stroną (a właściwie serwerem). Zazwyczaj Cookie jest wykorzystywane do automatycznego rozpoznawania danego użytkownika przez serwer, dzięki czemu serwer może wygenerować stronę ściśle dedykowaną danemu użytkownikowi. Umożliwia to tworzenie spersonifikowanych serwisów WWW. Inne zastosowania Cookie to np. "koszyki zakupów" w internetowych sklepach.

Sesja to zapamiętane po stronie serwera dane skojarzone z konkretną przeglądarki. Skojarzenie to jest jednoznaczne dzięki mechanizmowi Cookie. W podanym wyżej przykładzie koszyka z zakupami, serwer pamięta listę towarów zakupionych przez użytkownika.

W szczególności sesje pozwalają zapamiętać, kto jest zalogowany podczas generowania kolejnych stron. Zostało to wykorzystane w *Asystencie układania planu*. Aby uzupełnić np. Życzenia, pracownik musi się zalogować. Do logowania służy prosty formularz przedstawiony na rysunku 3.5.

Rysunek 3.5: Źródło http://theta.uwb.edu.pl/plan/zyczenia/login.mhtml

Formularz składa się z czterech elementów odpowiednio są to: pole typu `text`, pole typu `password` oraz dwa przyciski *Wyczyść* (przycisk typu `reset`) i *ZALOGUJ* (przycisk typu `submit`). W Meta-HTML w prosty sposób inicjalizuje się sesje. Przygotowane są do tego gotowe funkcje. Przed zainicjalizowanie sesji należy przejść przez kolejne etapy:

1. usuwanie istniejącej sesji — służy do tego funkcja `unset-sesion-var`, jako argument pobierana jest nazwa zmiennej, którą chcemy z sesji usunąć, tutaj będzie to nazwa użytkownika.

2. sprawdzenie, czy w bazie danych istnieje użytkownik o podanym identyfikatorze i czy podane hasło jest poprawne. Informacje o użytkowniku i haśle przekazywane są za pomocą formularza na rys. 3.5.

Jeżeli drugi etap zakończył się pomyślnie możemy zainicjalizować sesje. Wygląda to w następujący sposób:

```
<session::initialize 60>
```

Polecenie to inicjalizuje sesje, to znaczy: nadaje sesji unikalny identyfikator SID (Session Identifier), dodaje do listy nagłówków przetwarzanego dokumentu nagłówek `SetCookie` zawierający wartość SID oraz ustawia czas trwania sesji na 60 minut. W przypadku, gdy nie podamy czasu trwania sesji ustawiany on jest domyślnie na 200 minut.

Następne polecenie zapisuje w sesji zmienną `auth::UserName`, w której zapamiętana jest nazwa użytkownika, przekazana z formularza logowania:

```
<set-session-var auth::UserName=
            <get-var posted::iUserName>>
```

Aby ochronić wszystkie podstrony, tak aby niepowołane osoby nie mogły się dostać do naszych zasobów, na każdej podstronie należy umieścić prosty warunek:

```
<if <not  <get-session-var auth::UserName>>
      <redirect login.mhtml>>
```

Warunek ten sprawdza, czy zmienna `auth::UserName` jest ustawiona. Jeśli nie, to zostaniemy przekierowani do strony z formularzem logowania (rys. 3.5).

Autoryzacja na poziomie serwera HTTP

Innym sposobem ochrony dostępu do danych na WWW jest wykorzystanie mechanizmów autoryzacji na poziomie serwera HTTP. Takie rozwiązanie gwarantuje lepsze zabezpieczenie zasobów, gdyż działa ono na niższym poziomie niż aplikacja internetowa, a mianowicie na poziomie systemu plików serwera. Daje to również pewną wygodę dla autora aplikacji, bo mechanizm jest przezroczysty z punktu widzenia aplikacji – aplikacja nie musi nic o nim wiedzieć,

a jej autor ma mniej pracy. Niestety nie ma rozwiązań idealnych. Aby zastosować autoryzację na poziomie serwera HTTP konieczna jest rekonfiguracja serwera.

W przypadku serwera Apache system autoryzacji działa w ten sposób, że w pliku konfiguracyjnym wskazuje się kartotekę w systemie plików serwera i określa się kto ma dostęp do niej i podkatalogów poniżej. Nazwy użytkowników wraz z hasłami mogą być trzymane w zwykłym pliku tekstowym lub w bazie danych np. MySQL, ale wówczas niezbędny jest dodatkowy moduł Apache. Hasła mogą być przechowywane jako otwarty tekst lub w postaci zaszyfrowanej.

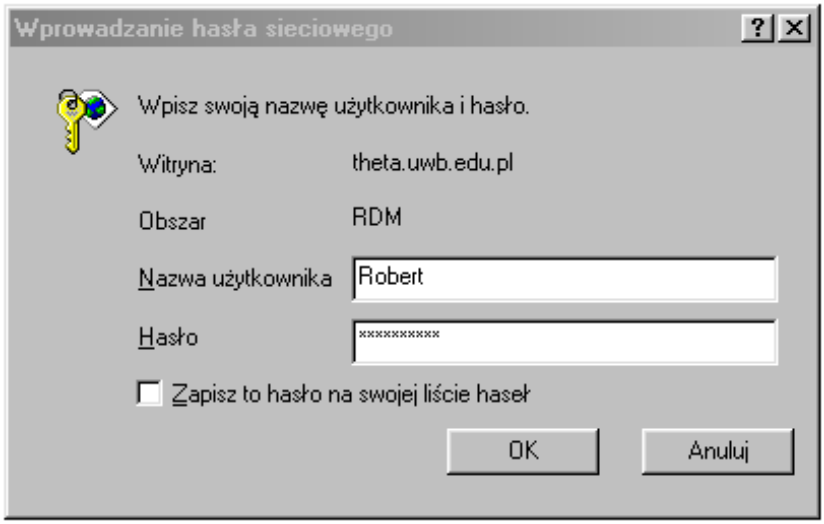
Jeśli chodzi o realizację tego typu autoryzacji użytkowników, z punktu widzenia protokołu HTTP, mechanizm jest bardzo podobny do Cookies. Serwer w odpowiedzi na żądanie chronionego dokumentu zwraca w nagłówku HTTP prośbę o podanie nazwy użytkownika i hasła, formalnie serwer zwraca kod statusu 401 (AUTH_REQUIRED). Przeglądarka winna wówczas wyświetlić okno dialogowe (por. rys. 3.6), w którym użytkownik wpisuje swój identyfikator i hasło. Jeśli zalogowanie powiedzie się, przy każdym żądaniu przeglądarka wysyła do serwera nagłówek HTTP o nazwie `Authorization`, np:

```
Authorization: Basic username:passwd
```

gdzie `username` i `passwd` są zakodowane.

Różnica pomiędzy Cookies a opisywanym mechanizmem polega na tym, że w przypadku Cookies programista sam musi zadbać o zainicjowanie sesji i wysłanie jej identyfikatora do przeglądarki. Musi też dopilnować, aby dostęp do podstron zabezpieczanej witryny był kontrolowany. Przy autoryzacji poprzez serwer HTTP natomiast, wystarczy wykonać niewielką rekonfigurację serwera.

Autoryzację na poziomie serwera HTTP zastosowano w aplikacji intranetowej Instytutu Matematyki UwB, skąd pochodzi rys. 3.6.



Rysunek 3.6: Źródło http://theta.uwb.edu.pl/plan/rdm

Spis literatury

[1] Spainhour V.S., Quercia V., *Webmaster – podręcznik administratora*, ReadMe, Warszawa 1997.

[2] Comprehensive Perl Archive Network, http://www.cpan.org

[3] Meta-HTML Language Reference, http://theta.uwb.edu.pl/mhtml

[4] Meta-HTML Project, http://metahtml.sf.net

[5] MySQL Reference Manual, http://www.mysql.com/documentation/

[6] Open Source Initiative, http://www.opensource.org

[7] World Wide Web Consortium, http://www.w3c.org