php solutions 4/2005

Software Wydawnictwo Sp. z o.o., ul. Lewartowskiego 6, 00-190 Warszawa, POLSKA.
redakcja@phpsolmag.org

Seagull PHP Framework

Werner M. Krauß

Identyfikacja i uwierzytelnianie użytkownika, zapytania do bazy danych czy walidacja danych wprowadzanych za pomocą formularzy to problemy, z którymi zmagasz się wielokrotnie podczas swojej codziennej pracy z PHP. A może by tak przestać odkrywać Amerykę na nowo i przerzucić całą żmudną robotę na framework obsługujący wszystkie typowe operacje? Takim rozwiązaniem jest Seagull.

FORUM

http://www.phpsolmag.org/forum/id/3/

W SIECI

1. *http://seagull.phpkitchen.com/* – strona domowa Seagull
2. *http://seagull.phpkitchen.com/docs/wakka.php?wakka=Talks&v=235* – odczyty o Seagull

NA CD

Na CD zamieściliśmy Seagull oraz wszystkie listingi z artykułu.

Zagadnieniu frameworków poświęciliśmy artykuł *Frameworki dla PHP*, czyli wydajne tworzenie aplikacji WWW w numerze 2/2005 naszego pisma. Teraz pokażemy sposoby tworzenia aplikacji przy użyciu Seagull PHP Framework.

Czym jest Seagull?

Seagull (*http://seagull.phpkitchen.com*) jest obiektowym środowiskiem tworzenia aplikacji, bazującym w dużej mierze na klasach PEAR i rozprowadzanym na licencji BSD (*http://seagull.phpkitchen.com/LICENSE.txt*). Jego zaletami są między innymi łatwa instalacja i wykorzystanie dobrych praktyk programistycznych: wzorców projektowych, abstrakcyjnego interfejsu bazy danych i rozdzielenia treści od prezentacji. Budowa Seagull jest całkowicie modularna, co pozwala łatwo rozbudowywać system o nowe możliwości. Społeczność twórców projektu zwraca baczną uwagę na zachowanie przejrzystej struktury kodu, przestrzeganie zasad bezpieczeństwa i trzymanie się standardów internetowych w rodzaju XHTML czy CSS.

Do środowiska dołączanych jest kilka gotowych modułów: *Publisher* – prosty system zarządzania treścią (CMS), *Contact-us* – moduł udostępniania informacji kontaktowych, *Guestbook* (księga gości), moduł do tworzenia list często zadawanych pytań (FAQ), a nawet koszyk na zakupy. W skład Seagull wchodzą następujące elementy:

Co należy wiedzieć...

Najlepsze wyniki pracy ze środowiskiem Seagull można uzyskać mając przynajmniej 1–2 lata doświadczenia i dobrą znajomość podejścia obiektowego, PEAR i wzorców projektowych.

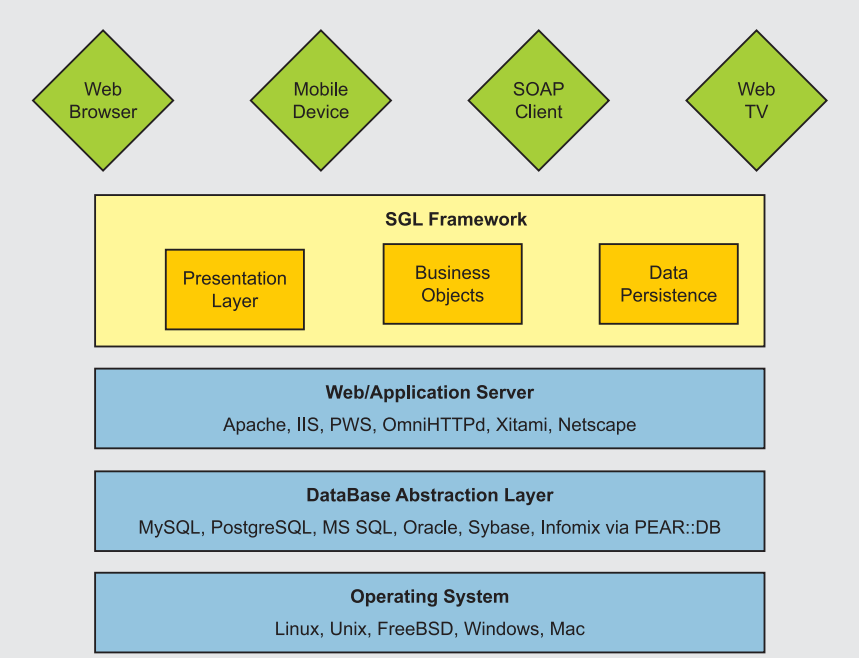
Co obiecujemy...

Pokażemy, jak łatwo jest stworzyć potężną aplikację za pomocą Seagull PHP Framework.

Wymagania systemowe

Seagull działa niemal identycznie na platformach Windows i Unix. Obecnie obsługiwane są bazy danych MySQL, PostgreSQL i Oracle, ale teoretycznie nie powinno być problemów ze współpracą z dowolną inną bazą obsługiwaną przez `PEAR::DB` (MS SQL, SQLite czy dowolną bazą dostępną przez ODBC).

- Środowisko podstawowe. Właściwe środowisko składa się z zestawu klas bazowych zorganizowanych zgodnie ze wzorcem Model-View-Controller (MVC), obsługujących kontrolę uprawnień, uwierzytelnianie, sesje, operacje wejścia-wyjścia i warstwę abstrakcji bazy danych.
- Komponenty. Każdy zakres funkcjonalny systemu jest komponentem. Konkretne wymagania logiki biznesowej aplikacji można dopasować do konkretnych komponentów, na przykład modułów, bloków czy pojedynczych elementów.
- Biblioteki. Większość funkcji obsługi konkretnych zadań pochodzi z bibliotek (często z repozytorium PEAR – *http://pear.php.net*), co pozwala łatwo aktualizować poszczególne biblioteki w przypadku wprowadzenia zmian czy ulepszeń.



Rysunek 1. *Struktura systemu*

- Encje i menedżerowie encji. Każdy obiekt w ramach aplikacji (użytkownik, grupa, właściwość, dokument, artykuł itd.) jest modelowany jako encja. Za pomocą gotowych narzędzi dostępnych w ramach Seagull można szybko tworzyć prototypy encji, na podstawie których automatycznie generowane i aktualizowane będą klasy szkieletowe.

Historia

Projekt został zainicjowany w 2001 r. przez Demiana Turnera, który potrzebował dla swoich celów prostego i niezawodnego środowiska implementującego kilka popularnych wzorców projektowych. Od października 2003 r., projekt jest prowadzony w ramach serwisu SourceForge (*http://www.sourceforge.net/projects/seagull/*).

Dlaczego właśnie Seagull?

Biorąc pod uwagę liczbę środowisk tworzenia aplikacji w PHP, jakie argumenty przemawiają akurat za Seagullem?

- Jest to projekt opensourcowy, cieszący się wsparciem licznej społeczności deweloperów.
- Większość wykorzystywanych w nim bibliotek to biblioteki PEAR, dobrze znane programistom PHP i cenione za wysoką jakość.
- Środowisko implementuje popularne wzorce projektowe, na przykład model MVC czy trzyetapowy przepływ danych.
- Dzięki licencji BSD, środowisko może być używane do tworzenia komercyjnych projektów o zamkniętym kodzie źródłowym, bez konieczności udostępniania stworzonych rozwiązań na zasadach open source.
- Kontroler interfejsu użytkownika obsługuje adresy URL o strukturze ułatwiającej indeksowanie przez wyszukiwarki, na przykład *http://www.example.com/index.php/contactus/action/list/*

Model-View-Controller

Wzorzec MVC (Model-View-Controller) pozwala jasno rozdzielić warstwy logiki biznesowej (Controller), operacji na danych (Model) i prezentacji (View). Zastosowanie tego wzorca w kontekście PHP omówiliśmy dokładniej w numerze 2/2005 PHP Solutions.

Architektura systemu

Przed rozpoczęciem pracy z Seagull, przyjrzyjmy się strukturze katalogów instalacji (kompletny schemat struktury przedstawia Rysunek 2). Oto krótki opis poszczególnych katalogów:

- katalog główny instalacji: *init.php* i *constants.php,*
- *etc/* : pliki konfiguracyjne, skrypty SQL itd.,
- *lib/* : biblioteki (Seagull, PEAR i inne), a w podkatalogu *data/* pliki danych (na przykład tablice nazw krajów lub języków),
- *modules/* : osobny podkatalog dla każdego modułu,
- *var/* : wszelkie dane tymczasowe, na przykład skompilowane szablony, encje *DB_DataObject*, pliki dziennika i zapisy sesji (serwer musi mieć uprawnienia do zapisu w tym katalogu),
- *www/* : katalog główny aplikacji, zawierający skrypt kontrolera interfejsu użytkownika, motywy wyglądu i kody JavaScript. Z Internetu powinien być dostępny wyłącznie ten katalog. Jeśli nie jest to możliwe, należy ograniczyć dostęp do pozostałych katalogów za pomocą plików *.htaccess*.

Podstawowe klasy

Typowe zadania, jak na przykład połączenia z bazą danych, wysyłanie poczty elektronicznej czy formatowanie danych wyjściowych, obsługują klasy bazowe Seagull znajdujące się w podkatalogu */lib/SGL/*. Na stronie domowej projektu dostępna jest kompletna dokumentacja API Seagull (*http://seagull.phpkitchen.com/apidocs/*).

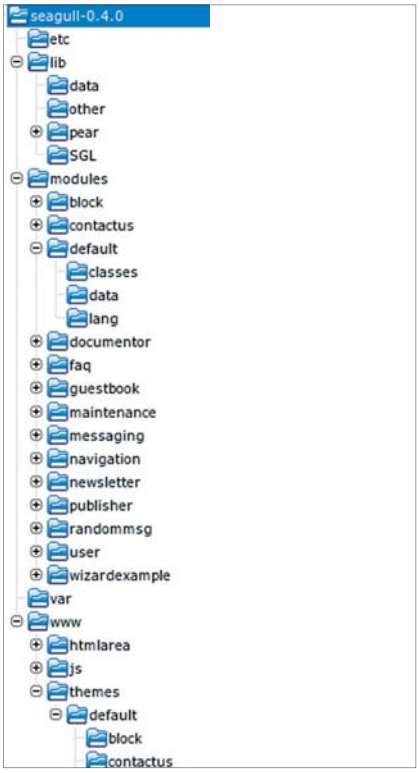
Szablony i motywy

W celu oddzielenia danych od ich prezentacji, Seagull wykorzystuje szablony i motywy. Domyślnie używany jest pakiet PEAR o nazwie *HTML_Template_Flexy* (*http://pear.php.net/package/HTML_Template_Flexy*). Flexy kompiluje wszystkie szablony HTML do postaci skryptów PHP, których twórca aplikacji nie musi nawet dotykać, dzięki czemu nie jest konieczne każdorazowe przetwarzanie szablonów w reakcji na żądanie użytkownika.

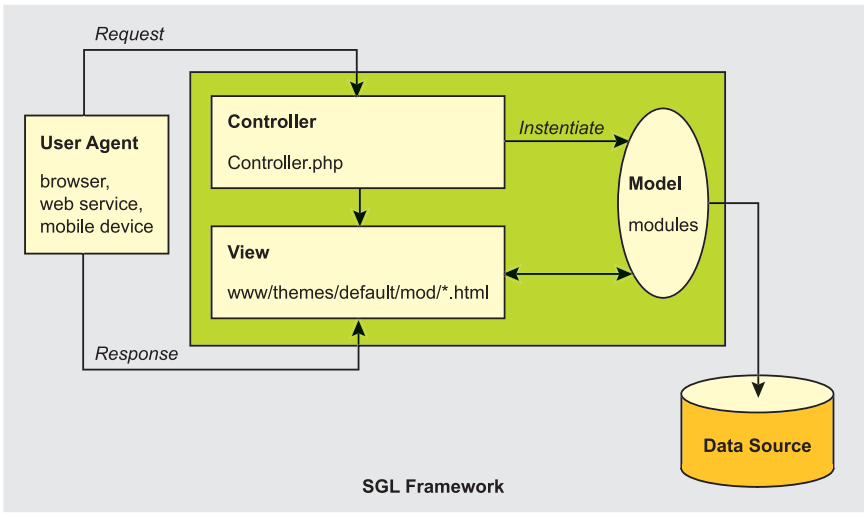
Z kolei motyw reprezentowany jest przez zbiór podkatalogów w katalogu *www/themes/*. Każdy taki podkatalog zawiera szablony HTML dla danego modułu.

Zaczynamy wyprawę

Załóżmy, że chcemy stworzyć portal, w ramach którego użytkownicy będą dodawać strony zawierające treść i udostępniać dokumenty. Wykorzystanie Seagull PHP



Rysunek 2. *Struktura katalogów*



Rysunek 3. *Implementacja Model-View-Controller z SGL*

Framework znacznie ułatwia to zadanie. Stworzenie naszej aplikacji sprowadza się do zainstalowania środowiska Seagull, zaimportowania użytkowników, odpowiedniego przypisania uprawnień, dodania modułów realizujących potrzebne zadania (na przykład strony FAQ czy modułu CMS do udostępniania artykułów) oraz dostosowania wyglądu i zachowania witryny do potrzeb konkretnego zastosowania. Oczywiście można potem rozszerzać możliwości bazowej aplikacji dodając nowe moduły – Seagull udostępnia na przykład prosty kalendarz zdarzeń.

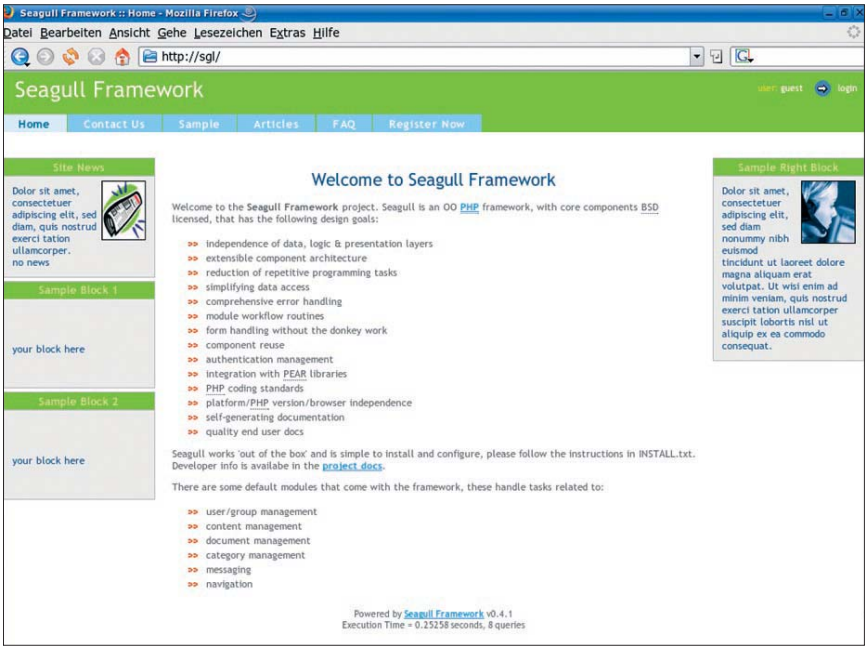
Po zainstalowaniu środowiska (patrz ramka *Instalacja*), możemy przystąpić do pracy.

Pierwsze kroki

Zaczynamy od zalogowania się do systemu jako użytkownik *admin* z hasłem *admin*. Na początek od razu zmieniamy domyślne hasło – wystarczy przejść do zakładki *My Account* i kliknąć przycisk *Change password*.

Konfiguracja i preferencje

Poza normalnym menu aplikacji, użytkownik *admin* ma dostęp do dodatkowych pozycji *Configuration* i *Modules*. Pierwsza z nich umożliwia zmianę globalnych ustawień konfiguracji, na przykład nazwy serwisu, parametrów połączenia z bazą danych itd. Opcja *Modules* udostępnia podgląd wszystkich zainstalowanych modułów, przy czym ustawienia pozycji wyświetlanych na



Rysunek 4. *Ekran poinstalacyjny*

Instalacja

Instalacja środowiska Seagull jest bardzo prosta. Potrzebny jest jedynie serwer WWW (na przykład Apache lub IIS), PHP (wersja 4.3 lub nowsza, z PHP5 włącznie) i baza danych (na przykład MySQL, PostgreSQL lub Oracle).

- Pobieramy najnowszą wersję Seagull ze strony projektu (*http://seagull.phpkitchen.com*) lub z płyty dołączonej do numeru i rozpakowujemy archiwum do katalogu publikacji serwera WWW.
- Serwer musi mieć uprawnienia do zapisu w katalogu *var/* . W systemie uniksowym możemy je nadać wydając polecenie `chown -R nobody [katalog_publikacji]/seagull/var`.
- Uruchamiamy Seagull w przeglądarce, podając adres podobny do *http://localhost/seagull/www* (zależnie od konkretnej instalacji). W formularzu instalacyjnym wprowadzamy informacje dotyczące utworzonej wcześniej, pustej bazy danych (host, nazwa użytkownika, hasło itd.).
- Po wprowadzeniu wszystkich wartości, wciskamy przycisk *Create schema and load default data*, co spowoduje utworzenie przez instalatora odpowiedniej struktury tabel w bazie danych i wprowadzenie danych domyślnych. Może to chwilę potrwać, zależnie od szybkości komputera.
- Gdy pojawią się komunikaty o udanym utworzeniu tabel i załadowaniu domyślnych danych, wystarczy kliknąć odsyłacz *launch Seagull*, by uruchomiło się świeżo zainstalowane środowisko Seagull (Rysunek 4).

Zaleca się przeprowadzenie instalacji na hoście wirtualnym (patrz ramka *Host wirtualny*).

Instalacja z wykorzystaniem PEAR-a

Jest to najłatwiejsza i najszybsza metoda uruchomienia Seagull. Istnieje jednak kilka dodatkowych wymagań:

- W systemie musi być zainstalowana wersja PHP 4.3.4 lub późniejsza oraz pakiety bazowe PEAR.
- Jako wartość opcji PEAR `data_dir` należy podać główny katalog publikacji lub dowolny inny katalog, ale z późniejszym utworzeniem hosta wirtualnego w celu udostępnienia wyłącznie katalogu *www*. Do określenia katalogu służy opcja `-d data_dir=/ścieżka/do/katalogu`. Do wyświetlenia bieżących ustawień używamy polecenia `pear config-show`.
- Preferowany stan pakietu musi być ustawiony na *beta*, gdyż projekt Seagull jest obecnie w fazie alpha. Do ustawienia tego parametru służy opcja `-d preferred_state=alpha`.

Tak więc instalacja Seagull w systemie uniksowym sprowadza się do wykonania następującego polecenia:

```
pear -d data_dir=/ścieżka/do/katalogu/publikacji -d ↩
   preferred_state=alpha install --onlyreqdeps ↩
   http://kent.dl.sourceforge.net/sourceforge/seagull/seagull-0.4.3.tgz
```

Po wykonaniu instalacji za pomocą menedżera pakietów PEAR, należy pamiętać o przywróceniu pierwotnych ustawień konfiguracji PEAR-a. Pozostała część instalacji przebiega zgodnie z wcześniejszym opisem.

Host wirtualny

Lepiej i bezpieczniej jest instalować Seagull na osobnym serwerze lub hoście wirtualnym. Katalogiem głównym publikacji powinien być *ścieżka/do/seagull]/www*, dzięki czemu z Internetu dostępne będą wyłącznie dane z tego katalogu, a pozostałe pliki instalacji (pliki konfiguracyjne, biblioteki czy pliki wysłane na serwer przez użytkowników) będą chronione.

niebieskim tle można modyfikować bezpośrednio lub za pomocą funkcji konfiguracyjnych. Każdy moduł posiada też w swoim katalogu własny plik *conf.ini*, niedostępny z poziomu menu konfiguracji.

Po zainstalowaniu środowiska Seagull pojawia się interfejs użytkownika w domyślnym języku angielskim, który można zmienić w menu preferencji. W przeciwieństwie do ustawień konfiguracyjnych, które dotyczą wszystkich użytkowników, każdy użytkownik ma możliwość samodzielnego ustawienia własnych preferencji w zakładce *My Account -> Preferences*. Preferencje obejmują takie ustawienia, jak na przykład język, strefa czasowa czy ilość wyników wyświetlanych na każdej stronie.

Aby zmienić ustawienia języka, należy kliknąć opcję *Modules*, a następnie wybrać moduł *Users and Security*. Spowoduje to przejście do zarządzania danymi użytkownika. Na górze pojawi się drugie menu, z którego należy wybrać skrajną prawą pozycję, czyli *prefs*, co spowoduje przejście do menedżera preferencji.

Teraz zmieniamy wartość w polu *language*. Aby zmienić język interfejsu użytkownika na polski, wprowadzamy *pl-iso-8859-2* (połączenie dwuliterowego skrótu lokalizacji i używanego zestawu znaków ISO). Obecnie Seagull obsługuje 12 języków, w tym niemiecki, francuski, hiszpański, polski, czeski, a nawet chiński. Pełna lista obsługiwanych języków wraz z odpowiadającymi im ustawieniami znajduje się w dokumentacji (*http://seagull.phpkitchen.com/docs/wakka.php?wakka=Localisation*).

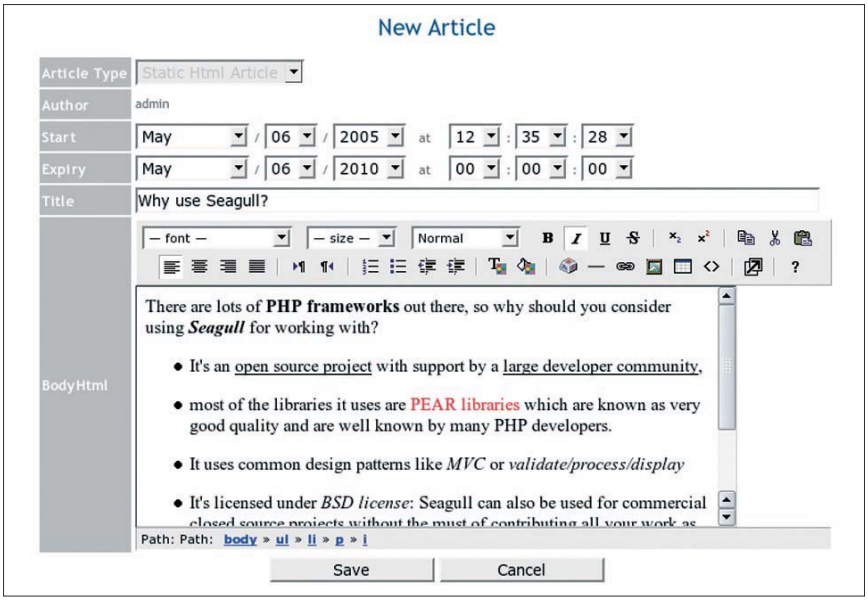
Rejestracja nowych użytkowników

Nowi użytkownicy mogą się rejestrować samodzielnie za pomocą formularza logowania dostępnego po kliknięciu *Register Now* lub mogą być ręcznie dodawani przez administratora korzystającego z funkcji *Users and Security -> add user*. Użytkownik *admin* może też zmieniać rolę użytkownika, ustawiać domyślną organizację dla użytkownika (patrz ramka *Grupowanie użytkowników w organizacje*) oraz status użytkownika (aktywny lub nieaktywny). Administratorzy mogą też zmienić hasło dowolnego użytkownika, ustawić automatyczne powiadomienie użytkownika o nowym haśle (opcjonalnie) oraz szczegółowo modyfikować i dostrajać uprawnienia użytkownika, stosownie do potrzeb.

Istnieje możliwość domyślnego nadawania nowemu użytkownikowi uprawień administratora, jednak trzeba ją dodatkowo ustawić w pliku *conf.ini* modułu użytkownika.

Zarządzanie uprawnieniami

Seagull posiada bardzo szczegółowy system kontroli uprawnień użytkowników. Każda funkcja lub metoda menedżera każdego modułu ma własne uprawnienie dostępu. Nazwy uprawnień tworzone są według konwencji `menedżermodułu_funkcja`. Na przykład dla modułu *FAQ* dostępne są takie uprawnienia, jak `faqmgr_list` do wyświetlania listy pytań, `faqmgr_add` i `faqmgr_insert` do dodawania wpisów do listy itd. Możliwe jest również zdefiniowanie uprawnienia dla wszystkich funkcji całego modułu, co przydaje się szczególnie w przypadku administratorów. W takiej sytuacji, nazwą uprawnienia jest po prostu nazwa mene-



Rysunek 5. *Dodawanie statycznego artykułu HTML*

dżera – w przypadku menedżera FAQ byłoby to `faqmgr`.

Często pojawia się konieczność ustawienia tych samych uprawnień dla wielu użytkowników, na przykład gdy pracują oni w tym samym obszarze. Każdorazowe ręczne dodawanie uprawnień dla każdego użytkownika z osobna jest zajęciem żmudnym i czasochłonnym. Znacznie wygodniejszym rozwiązaniem będzie utworzenie odpowiedniej roli, czyli grupy uprawnień. Wystarczy stworzyć grupę o nazwie na przykład *FAQAdmin* i dodać do niej pożądane uprawnienia (na przykład uprawnienie `faqmgr` dla całego modułu). Odtąd każdy użytkownik pracujący w ramach modułu FAQ będzie mógł otrzymać tę rolę, zyskując tym samym odpowiednie uprawnienia.

Trzeba jednak pamiętać, że w rzeczywistości Seagull nie przypisuje użytkownikowi roli, a jedynie zestaw uprawnień zdefiniowanych w ramach roli. Ewentualna zmiana uprawnień składających się na rolę nie spowoduje więc automatycznej aktualizacji uprawnień użytkowników posiadających tę rolę – w takim przypadku konieczna jest ręczna synchronizacja roli każdego użytkownika. W zamian za tę niedogodność, zyskujemy możliwość dawania i zabierania poszczególnym użytkownikom uprawnień nieujętych w ramach pierwotnej roli. Zarządzanie całym procesem sprowadza się do kilku kliknięć uruchamiających odpowiednie funkcje pomocnicze Seagull.

Korzystanie z ról

Dodanie nowej roli jest proste. Można zacząć od pustej roli, wybierając funkcję *add role* i wprowadzając odpowiednią nazwę i opis albo skopiować istniejącą i zmienić jej nazwę. Nowo utworzona rola jest teraz gotowa do wypełnienia uprawnieniami – po kliknięciu przycisku *change* pojawią się dwie listy uprawnień: po lewej lista dostępnych, a po prawej już posiadanych. Po utworzeniu kilku ról, możemy przejść do przypisania wybranej roli istniejącym użytkownikom poprzez edycję danych użytkownika.

Teraz dodamy do systemu kilku nowych użytkowników za pomocą przycisku *User Import Manager* (patrz ramka *Importowanie użytkowników*). Jest to bardzo wygodny sposób dodawania większej liczby użytkowników naraz, jednak najnowsza wersja Seagull dostępna w chwili pisania tego artykułu pozwala importować jedynie nazwę, imię, nazwisko i adres e-mail użytkownika. Pozostałe dane (na przykład adres czy pytanie do przypomnienia hasła) trzeba póki co wprowadzać ręcznie – funkcja ta zostanie w przyszłości rozbudowana.

Wstęp do pracy z modułem Publisher

Pora przyjrzeć się kolejnemu modułowi, którym jest *Publisher.* Jak sugeruje nazwa, jest to moduł systemu zarządzania treścią (CMS). Pozwala on zarządzać artykułami, plikami umieszczanymi na serwerze oraz kategoriami treści.

Grupowanie użytkowników w organizacje

Użytkowników Seagull można dzielić na organizacje, czyli dowolnie zdefiniowane grupy, co jest szczególnie przydatne w przypadku osób pracujących w tej samej firmie lub dziale. Podział na organizacje jest funkcją opcjonalną i można go wyłączyć w pliku *conf.ini* modułu użytkowników – w przypadku niewielkich, prostych instalacji Seagull, grupowanie może po prostu nie być potrzebne.

Organizacje mają strukturę hierarchii zagnieżdżonych zbiorów, co pozwala łatwo obsłużyć na przykład taką sytuację:

- firma ABC ma wielu dostawców,
- firma ta ma też pewną ilość dystrybutorów,
- niektórzy dostawcy są również klientami lub mają własnych klientów,
- niektórzy klienci mają pod sobą sprzedawców.

Odpowiadająca tej sytuacji struktura organizacji wyglądałaby tak:

- firma ABC -> dostawca DEF -> klient GHI -> sprzedawca JKL.

Przy takim układzie, prześledzenie całego łańcucha dostaw od firmy do sprzedawcy wymaga jedynie wywołania na zbiorze metody `getBranch()`, co znacznie zmniejsza ilość kodu niezbędnego do czynności administracyjnych. W menedżerze organizacji można też ustawić typ organizacji, co dodatkowo ułatwia grupowanie – dla powyższego przykładu, typami byłyby: firma, dostawca, klient i sprzedawca. Każdej organizacji można też przypisać domyślną rolę i domyślne preferencje, co pozwala automatycznie przydzielać zestawy uprawnień użytkownikom dodawanym do organizacji. Możliwość ta przydaje się szczególnie w przypadku zarządzania systemami o większej ilości użytkowników.

Importowanie użytkowników

Tworząc portal z wieloma użytkownikami, można skorzystać z funkcji importu użytkowników, jako źródło danych wykorzystując na przykład książkę adresową programu Thunderbird. W tym celu należy wysłać na serwer plik CSV z danymi korzystając z modułu *Publisher* (opcja *Documents*), po czym przejść do modułu użytkownika, kliknąć opcję *import users*, wskazać wysłany właśnie plik, określić organizację i rolę, po czym kliknąć *process file.* Prawda, że łatwe?

Pozostaje tylko aktywować użytkowników i wprowadzić dla nich dodatkowe informacje. Specyfikację budowy pliku importu zawiera dokumentacja Seagull (*http://seagull.phpkitchen.com/docs/*).

Otwieramy więc interfejs administracyjny tego modułu. Na górze strony widoczne są trzy sekcje: *Categories*, *Documents* i *Articles*. Sekcja *Categories* pozwala definiować poddrzewa kategorii w ramach modułu. Każdej kategorii można przypisać oddzielne prawa dostępu, zależne od roli użytkownika. Sekcja *Documents* umożliwia wysyłanie dokumentów na serwer. Dostępne opcje dokumentu to kategoria, nazwa i opis.

Dodawanie właściwej treści do strony odbywa się za pośrednictwem menedżera artykułów (*Article Manager*). Podstawowa wersja Seagull pozwala tworzyć trzy rodzaje artykułów. Zakres dostępnych artykułów można łatwo rozszerzyć, ale w podstawowej instalacji dostępne są:

- *HTML Article* (artykuł HTML): zwykły artykuł, widoczny w hierarchii kategorii. Ten typ artykułu przydaje się w przypadku aplikacji intranetowych lub kategoryzacji już istniejących materiałów.
- *Static HTML Article* (statyczny artykuł HTML): samodzielna strona serwisu, dostępna za pośrednictwem własnej zakładki i niewyświetlana w hierarchii kategorii.

Rysunek 6. *Gotowy statyczny artykuł HTML*

- *News Item* (krótka wiadomość): notatka pojawiająca się w lewej kolumnie strony, w ramach bloku *Site News*.

Tworzymy stronę statyczną

Prześledźmy proces tworzenia strony statycznej, dostępnej za pośrednictwem własnej zakładki nawigacyjnej. Po zalogowaniu się jako *admin*, przechodzimy do modułu *Publisher* i uruchamiamy menedżera artykułów klikając przycisk *Articles*. Z listy rozwijanej wybieramy pozycję *Static HTML Article* i naciskamy *new article*. Pojawi się formularz umożliwiający dodawanie treści do artykułu. Na początek wprowadźmy daty publikacji i wygaśnięcia oraz tytuł i treść. Jeśli pracujemy pod Internet Explorerem lub przeglądarką z rodziny Mozilli, możemy zmieniać układ artykułu korzystając z edytora wizualnego, co ilustruje Rysunek 5.

Po zapisaniu, nowy artykuł pojawi się na liście oznaczony jako *For Approval*, czyli oczekujący na zatwierdzenie. Aby artykuł stał się widoczny dla użytkowników strony, klikamy *approve*, a następnie *publish*. Te same czynności należy wykonać po każdej zmianie zawartości artykułu.

Nawigacja

Teraz stworzymy zakładkę nawigacyjną dla napisanego wcześniej statycznego artykułu HTML. W zakładce *Page Manager* naciskamy przycisk *new section* i podajemy tytuł strony, po czym z list rozwijanych wybieramy odpowiednio *static articles* i nasz artykuł. Następnie przełączamy pole *Status* na *active* i określamy role mające uprawnienia do czytania tego artykułu. Po zapisaniu zmian powinna być widoczna nowa zakładka nawigacyjna dla artykułu (Rysunek 6).

Równie łatwo można dodawać zakładki nawigacyjne dla modułów. Wystarczy tylko wybrać opcję *dynamic pages* zamiast *static pages*, a pojawi się formularz pozwalający wybrać moduł (*module*), menedżera modułu (*manager*) oraz podejmowane działanie (*action*). W przypadku modułu FAQ są to odpowiednio wartości *faq*, *faqMgr* i *none* – domyślnym działaniem dla każdego modułu jest brak działania.

Seagull wykorzystuje system nawigacji bazujący na motywach i sterownikach. W obecnej wersji jedynym dostępnym ste-

Standardy kodowania

Kod Seagull jest tworzony w maksymalnej zgodności ze standardami kodowania PEAR-a (*http://pear.php.net/manual/en/standards.php*). Oto najbardziej charakterystyczne cechy tego stylu kodowania:

PHP

- nazwy wszystkich zmiennych są zapisywane literami o zmiennej wielkości, czyli `$toJestMojaZmienna`, a nie `$to_jest_moja_zmienna`,
- nazwy wszystkich klas zaczynają się wielką literą, po czym stosowane są litery o zmiennej wielkości,
- nazwy stałych są pisane wielkimi literami, a poszczególne słowa są rozdzielane znakami podkreślenia,
- słowa kluczowe w rodzaju `true` czy `null` są zawsze pisane małymi literami,
- większość klas jest skomentowana zgodnie z formatem PHPdoc,
- `require_once` i `include_once` nie wymagają nawiasów, gdyż nie są to funkcje,
- kod jest możliwie najobficiej komentowany, by pomóc innym jego użytkownikom,
- krótkie znaczniki php, czyli `<? ?>`, nie są używane, gdyż są niezgodne z XML-em – w ich miejsce używane są pełne `<?php ?>`,
- w celu zwiększenia czytelności kodu stosowane są opisowe nazwy zmiennych, choć nie tak wyczerpujące, jak na przykład w Javie,
- cudzysłowy (na przykład "string") są używane jak najrzadziej, gdyż konieczność ich przetwarzania przez PHP spowalnia wykonanie kodu,
- wiersze kodu nie mogą być dłuższe niż 80 znaków.

SQL

Istnieje też kilka zasad, których należy przestrzegać tworząc nową tabelę:

- nazwy tabel i kolumn pisane są małymi literami,
- kolumna klucza głównego ma zawsze nazwę *nazwatabeli_id*,
- nie należy korzystać z automatycznej inkrementacji MySQL-a – lepiej korzystać z sekwencji udostępnianych przez PEAR::DB,
- nazwy encji powinny być w liczbie pojedynczej, a nie mnogiej – *książka*, a nie *książki*,
- nazwy relacji między tabelami powinny być rozdzielane podkreśleniami, na przykład *user_preference*.

Przepływ danych (workflow)

Seagull dostarcza gotowego modelu przepływu danych walidacja-przetwarzanie-wyświetlanie, zgodnie z którym wszystkie dane wprowadzane do systemu są poddawane działaniu następujących metod:

- `validate()`: przyjmuje surową treść żądania `$_REQUEST` i wykonuje na niej walidację, po czym dopuszczalne dane są odwzorowywane na obiekt `$input`,
- `process()`: jeśli wszystkie wprowadzone dane są poprawne, obiekt `$input` jest przekazywany metodzie `process()`, która przekierowuje go do metody obsługi konkretnego działania. Po zakończeniu przetwarzania, dane są odwzorowywane na obiekt `$output`,
- jeśli choć jeden etap walidacji nie powiódł się, wszystkie dane są uznawane za błędne i są przekazywane bezpośrednio do metody `display()` wraz z odpowiednim komunikatem, co powoduje ponowne przedstawienie danych użytkownikowi w celu ich poprawienia,
- `display()`: pobiera przekazane dane (poprawne lub nie), dodaje kilka właściwości systemowych (na przykład czas wykonania) i przesyła je do mechanizmu szablonów, który generuje wynikowy kod HTML.

rownikiem jest *SimpleNav*, ale nowe sterowniki można łatwo tworzyć bazując na przykład na klasie *PEAR::HTML_Menu*.

W pakiecie instalacyjnym Seagull dostępnych jest kilka różnych arkuszy stylów. Wszystkie wykorzystują czysty CSS, bez udziału JavaScriptu. W celu zmiany motywu należy kliknąć przycisk *modify nav bar appearance* w górnej części ekranu *Page Manager*, co spowoduje wyświetlenie strony podglądu, pozwalającej edytować motywy nawigacji dla poszczególnych ról. Tworzenie nowego motywu dla sterownika *SimpleNav* jest proste, gdyż zwraca on zwykłą listę wypunktowaną (`<ul>`) – wystarczy zatem utworzyć dla niej nowy plik CSS.

Używanie bloków

Za pomocą modułu *Block* można łatwo dodawać do serwisu bloki treści. Do pakietu Seagull załączono gotowe bloki do obsługi aktualności, logowania, losowych wiadomości i nawigacji po kategoriach. Wybieramy *Manage Modules ->Blocks*, by wyświetlić menedżera bloków. Tu możemy dodawać i edytować wpisy, jak również zmieniać kolejność wyświetlanych bloków. Nowe rodzaje bloków dodamy umieszczając nową klasę w katalogu *modules/block/classes/blocks.*

Podczas edycji danych bloku, parametr *Name* odpowiada plikowi PHP używanemu do generowania faktycznej treści. Jeśli na przykład chcemy dodać do serwisu blok logowania dostarczany w pakiecie instalacyjnym Seagull, musimy go aktywować na ekranie administracyjnym modułu bloku. Do edycji dostępny jest tytuł bloku, klasa CSS używana do jego wyświetlania, pozycja wyświetlania (z lewej lub z prawej strony ekranu) oraz sekcje nawigacji (czyli zdefiniowane wcześniej zakładki), w których blok ma być widoczny. Oczywiście można też zmieniać kolejność bloków. W dalszej części artykułu zobaczymy też, jak napisać kod tworzący nowy blok nowego modułu.

Własny moduł: terminarz

Udało nam się dotąd stworzyć ciekawy i w pełni funkcjonalny portal. Co jednak zrobić, jeśli potrzebujemy rozbudować nasz serwis o funkcje nieobsługiwane w ramach podstawowej dystrybucji Seagull? Jedną z zalet środowiska Seagull jest możliwość łatwego tworzenia nowych modułów. Trzeba wprawdzie nauczyć się nazw struktur i klas, ale to samo dotyczy każdego innego środowiska programowania. Wystarczy zrozumieć zasady działania całego systemu, a tworzenie modułów będzie łatwe i szybkie. Napiszemy własny moduł terminarza umożliwiający dodawanie, edycję i usuwanie zaplanowanych terminów oraz przeglądanie ich listy.

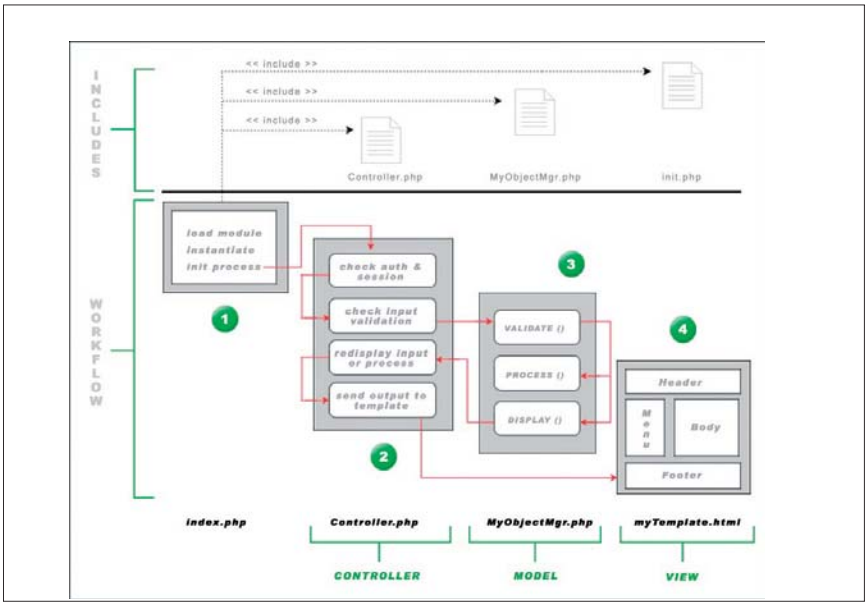
Generator szkieletów modułów

Istnieją dwa sposoby utworzenia nowego modułu Seagull. Po pierwsze, możemy wziąć istniejący moduł o możliwościach zbliżonych do naszych potrzeb, a następnie stosownie go zmodyfikować.

W tym celu wystarczy skopiować wszystkie pliki istniejącego modułu i odpowiednio pozmieniać ich nazwy. Drugą opcją jest utworzenie nowego modułu. Możemy je zacząć od zera lub skorzystać z generatora szkieletów modułów (narzędzie *Module Skeleton Generator* w module *Maintenance*).

Narzędzie to nie pozwala jeszcze na utworzenie kompletnego modułu: korzystając z danych pobranych z formularza tworzy ono tylko podstawową strukturę modułu, w tym katalogi klas, tłumaczeń i szablonów oraz podstawowe pliki. Korzystanie z generatora wymaga nadania serwerowi WWW uprawnień zapisu do katalogów *modules/* i *www/themes/default/*, w których tworzone będą pliki. Następnie przechodzimy do modułu *Maintenance* i wypełniamy formularz *Create a module* podając nazwy: modułu, menedżera (klasy PHP) oraz funkcji, które mają być dostępne w ramach modułu.

Nasz moduł nazwiemy *Event*, a nową klasę – `EventMgr`. Kod tej wygenerowanej klasy przedstawia Listing 1. Dziedziczy ona z klasy `SGL_Manager`, która implementuje bazowy model obiegu dokumentów (patrz ramka *Przepływ*



Rysunek 7. *Schemat obiegu dokumentów*

danych). Do naszego nowego modułu musimy jedynie dodać nowe funkcje. W konstruktorze `EventMgr()` zdefiniowanych jest kilka zmiennych: nazwa modułu, tytuł strony, domyślny szablon Flexy do generowania kodu HTML oraz tablica `$this->_aActionsMapping`, odwzorowująca żądania działań zapisane jako tekst i pochodzące z `$_GET` lub `$_POST` na odpowiednie metody ich obsługi. Niektóre działania mają przypisanych kilka metod, na przykład `'insert' => array('insert', 'redirectToDefault')`. W tym przypadku, dane są najpierw wstawiane do bazy (bez informacji zwrotnych), po czym skrypt przekierowuje użytkownika do działania domyślnego – tutaj będzie to metoda `list()`, wypisująca dane z bazy. Dla każdego działania możemy definiować tyle operacji, ile potrzeba.

Funkcja `validate($req, &$input)` waliduje zmienne wejściowe pobierane z obiektu żądania `$req`, a następnie zapisuje je w obiekcie `$input`. Metody dodawania, wyświetlania i inne są na razie puste, ale do każdej metody obsługi działania przekazywane są referencje do obiektów `$input` i `$output`.

Do uruchomienia modułu w ramach Seagull wymaganych jest też kilka innych plików: */modules/event/conf.ini*, definiujący konfigurację modułu (patrz Listing 2) oraz pliki języków w katalogu */modules/event/lang/* – są to zwykłe pliki PHP, zawierające tablice przetłumaczonych komunikatów (patrz Listing 3).

Listing 1. *Wynik pracy generatora szkieletu modułu*

```
<?php
/**
 * Miejsce na opis klasy
 * @package event
 * ...
 */
class EventMgr extends SGL_Manager
{
    function EventMgr()
    {
        SGL::logMessage(null, PEAR_LOG_DEBUG);
        $this->module       = 'event';
        $this->pageTitle    = 'Event Manager';
        $this->template     = 'eventList.html';

        $this->_aActionsMapping =  array(
            'add'       => array('add'),
            'insert'    => array('insert', 'redirectToDefault'),
            'edit'      => array('edit'),
            'update'    => array('update', 'redirectToDefault'),
            'delete'    => array('delete', 'redirectToDefault'),
            'list'      => array('list'),
        );
    }

    function validate($req, &$input)
    {
        SGL::logMessage(null, PEAR_LOG_DEBUG);
        $this->validated    = true;
        $input->error       = array();
        $input->pageTitle   = $this->pageTitle;
        $input->masterTemplate = $this->masterTemplate;
        $input->template    = $this->template;
        $input->action      = ($req->get('action'))
           ? $req->get('action') : 'list';
        $input->aDelete     = $req->get('frmDelete');
        $input->submit      = $req->get('submitted');

        //  w razie wystąpienia błędów
        if (isset($aErrors) && count($aErrors)) {
            SGL::raiseMsg('Proszę wypełnić wskazane pola');
            $input->error = $aErrors;
            $this->validated = false;
        }
    }

    function _add(&$input, &$output)
    {
        SGL::logMessage(null, PEAR_LOG_DEBUG);
    }
[...]
}
?>
```

SQL

Pora stworzyć dla modułu odpowiednią tabelę w bazie danych. Powinna ona zawierać identyfikator terminu i daty: rozpoczęcia, zakończenia, utworzenia i aktualizacji wpisu, jak również tytuł oraz krótki i długi opis. Listing 4 przedstawia kompletny kod SQL skryptu tworzącego tabelę.

Następnym etapem będzie utworzenie encji *DB_DataObject* dla nowej tabeli. Seagull pozwala programistom korzystać w swych modułach z pakietu *PEAR::DB_DataObject* (*http://pear.php.net/manual/en/package.database.db-dataobject.php*) – mechanizmu odwzorowania obiektowo-relacyjnego, znacznie ułatwiającego współpracę z bazą danych. Skorzystamy z niego również w naszym przykładzie.

Przechodzimy do modułu *Maintenance* i naciskamy przycisk *Rebuild DataObjects Now*, co spowoduje wygenerowanie nowego pliku *var/cache/entities/Event.php*. Zawiera on klasę udostępniającą elementarne funkcje interakcji z nowo utworzoną tabelą w bazie danych, noszącą w tym przypadku nazwę *DataObjects_Event*. Wykorzystanie nowej encji wymaga jeszcze dodania jej do skryptu modułu, zatem przed definicją klasy dopisujemy:

```
require_once SGL_ENT_DIR . '/Event.php';
```

Metody wyświetlania, dodawania i edycji terminów

Mamy już podstawowe struktury, pora więc wypełnić ten szkielet obsługą konkretnych działań. Zaczniemy od metody

Listing 2. *Zawartość pliku /modules/event/conf.ini*

```
[EventMgr]
requiresAuth    = false
```

`list()` – jej kod przedstawia Listing 5. Stworzenie tej funkcji zaczynamy od wysłania komunikatu do dziennika i zdefiniowania tytułu strony, po czym tworzony jest obiekt `$eventList` klasy `DB_DataObject`. Chcemy sortować zbiór wyników według daty wygaśnięcia, więc do naszego obiektu listy dodajemy polecenie `orderBy`: `$eventList->orderBy('date_expire');`.

Teraz ustawimy wyświetlanie wyłącznie obiektów aktualnych. W tym celu musimy stworzyć odpowiednio sformatowany ciąg daty i godziny SQL, co ułatwia wbudowana funkcja `SGL_Date::arrayToString()`.

Listing 3. *Przykładowy plik języka*

```
<?php
$words = array('Events' => 'Events',
/*  Events */
    'Event Manager' => 'Event Manager',
    'Event Manager :: Browse' => 'Event Manager :: Browse',
    'Event ID'    => 'ID',
    'Date created' => 'Date created',
    'Start' => 'from',
    'Expiry' => 'to',
    'Short Description' => 'Short Description',
    'Long Description' => 'Long Description',
/*  list */
    'No Events found' => 'No Events found');
?>
```

Listing 4. *Polecenie SQL tworzące tabelę dla modułu terminarza*

```
CREATE TABLE event (
   event_id int(11) NOT NULL default '0',
   date_start datetime default NULL,
   date_expire datetime default NULL,
   date_created datetime default NULL,
   date_last_updated datetime default NULL,
   title varchar(255) default NULL,
   text text,
   PRIMARY KEY  (event_id)
) ;
```

Listing 5. *Metoda list()*

```
function _list(&$input, &$output)
{
   SGL::logMessage(null, PEAR_LOG_DEBUG);
   $output->pageTitle = 'Events';
   $eventList = & new DataObjects_Event();
   $eventList->orderBy('date_expire');

   // klauzula where w celu pobrania tylko aktualnych wydarzeń
   list($day, $month, $year) = explode('/', date('d/m/Y'));

   // inicjalizacja tablicy wyjściowej bieżącą datą i godziną
   $aDate = array( 'day' => $day, 'month' => $month, 'year' => $year);
   $eventList->whereAdd('date_expire >= \''. SGL_Date::arrayToString($aDate). '\'');
   $result = $eventList->find();
   $aEvents = array();

   if ($result > 0) {
      while ($eventList->fetch()) {
         // sprawdza, czy data początkowa i data wygaśnięcia są równe.
         // jeśli tak, wyświetla w szablonie tylko jedną datę.
         if ($eventList->date_start == $eventList->date_expire)
            $eventList->date_start = '';
         $aEvents[] = clone($eventList);
        }
    }
    $output->results = $aEvents;
}
```

Metoda `$eventList->find()` sprawdza, czy baza danych zwraca jakiekolwiek informacje – jeśli tak jest, metoda `$eventList->fetch()` zwróci wszystkie terminy, jakie chcemy przeglądać.

Pozostaje tylko skopiować informacje z obiektu do tablicy wyjściowej za pomocą metody `clone()`. W tej samej pętli sprawdzamy też, czy daty: początkowa i wygaśnięcia nie są przypadkiem jednakowe. Jeśli są, to zostanie wyświetlona tylko jedna data. Na zakończenie przekazujemy tablicę z wynikami do obiektu wynikowego (polecenie `$output->results = $aEvents;`), gdzie nosi ona nazwę `{results}`.

Jeśli spróbujemy teraz uruchomić skrypt, ujrzymy informację o błędzie spowodowanym brakiem szablonu. I bardzo słusznie – w końcu w katalogu szablonów tego modułu, czyli */www/themes/default/event/* nie ma ani jednego pliku. Stwórzmy więc pierwszy szablon dla operacji wyświetlenia terminów – zawartość pliku *eventList.html* przedstawia Listing 6.

I to w zasadzie wszystko. Tak utworzony szablon zostanie dołączony do szablonu nadrzędnego, dzięki czemu trzeba w nim uwzględnić jedynie działanie naszego konkretnego modułu. Polecenie `{if: results}` sprawdza, czy w bazie znajdują się wyniki do wyświetlenia. Jeśli wyników nie ma, wyświetlany jest komunikat *No Events found*, przetłumaczony na wybrany język (o plikach języków wspominaliśmy już wcześniej).

Przyjrzyjmy się bliżej następującemu wierszowi:

```
<tr class="{switchRowClass()}"
   flexy:foreach="results,key,valueObj">
```

Mamy tu prostą pętlę `foreach`, wyświetlającą kolejne zestawy wyników. Jej działanie jest podobne do `foreach($results as $key => $valueObj)`. Tło każdego wiersza tabeli zmieniamy za pomocą funkcji `switchRowClass()`.

Nadal jednak nie mamy w bazie żadnych terminów, następnym krokiem będzie więc rozbudowanie naszego modułu o możliwość ich dodawania. Listing 7 przedstawia niezbędny kod. Dodawanie danych do bazy odbywa się w dwóch etapach: najpierw wyświetlamy formularz, a potem odbieramy dane wpisane przez użytkownika, walidujemy je i wstawiamy do bazy. Z tego też względu skorzystamy z dwóch osobnych metod: `add()` i `insert()`.

W ramach metody `add()` definiujemy jedynie kilka zmiennych i przekazujemy je do szablonu. Warto przyjrzeć się bliżej kontrolkom wyboru daty, gdyż wykorzystane są tutaj bardzo przydatne funkcje wbudowane. Najpierw definiujemy tablicę zawierającą wartości dnia, miesiąca i roku, a następnie przekazujemy ją metodzie `SGL_Output::showDateSelector()`. Działanie odwrotne ma metoda `SGL_Date::arrayToString()`, której używamy w ramach operacji `insert()` po zgłoszeniu danych przez użytkownika.

Wprowadzenie walidacji wymaga dodania nieco kodu do treści metody `validate()`. Na początek definiujemy wartości, jakie chcemy pobrać z danych wprowadzanych przez użytkownika, a następnie sprawdzamy, czy data wygaśnięcia jest późniejsza od daty początkowej i czy podany został tytuł terminu. W razie błędu wyświetlany jest stosowny komunikat, po czym użytkownik ponownie otrzymuje formularz do wypełnienia, tym razem z wypełnionymi polami, które wprowadził wcześniej. Pełen kod metody walidacji przedstawia Listing 8.

Metody obsługi edycji, aktualizacji i usuwania terminów implementujemy w podobny sposób. Pozostaje tylko oprogramować dostęp do nich – najwygodniej będzie stworzyć dodatkowy interfejs administracyjny. W tym celu wystarczy dodać jeden blok warunkowy w ramach metody `list()` (patrz Listing 9). Wykorzystany szablon administracyjny zawiera tabelę wyświetlającą wszystkie istniejące terminy oraz odsyłacze do funkcji dodawania, edycji i usuwania terminów. Adresy URL są generowane za pomocą funkcji `{makeUrl(#add#,#event#)}`, generującej poprawny adres URL wykorzystywany przez konkretny kontroler interfejsu.

Rejestracja nowego modułu

Teraz musimy dodać nowy moduł do menedżera modułów. W tym celu przechodzimy do zakładki *Modules -> Manage*, naciskamy przycisk *Add a module* i w wyświetlonym formularzu podajemy niezbędne informacje: nazwę modułu (*event*), tytuł (na przykład *Prosty harmonogram wydarzeń*), możliwość konfiguracji (w polu *configurable* podajemy *yes*), opis wyświetlany dla tego modułu w menedżerze, URL *admin* oraz plik ikony. Jeśli nie mamy własnej ikony dla modułu, możemy użyć jednej z istniejących, na przykład *faqs.png*. Po kliknięciu przycisku *Add* ujrzymy ponownie ekran menedżera modułów, ale tym razem nasz nowy moduł pojawi się na końcu listy. Kliknięcie nowej pozycji przeniesie nas do interfejsu administracyjnego modułu.

Dodawanie praw i ról oraz synchronizacja użytkowników

Jeśli użytkownik niebędący administratorem spróbuje wyświetlić listę terminów, zostanie poinformowany, że nie ma do tego dostatecznych uprawnień. W czym problem? Po prostu nie usta-

Listing 6. *Szablon do wyświetlania wszystkich terminów – plik eventList.html*

```
<h1 class="pageTitle">{translate(pageTitle)}</h1>
<span flexy:if="msgGet()">{msgGet()}</span>
<table class="wide">
{if:results}
    <tr class="{switchRowClass()}" flexy:foreach="results,key,valueObj">
        <td><em>{if:valueObj.date_start}{formatDate(valueObj.date_start)} - {end:}
          {formatDate(valueObj.date_expire)}: </em>
        <strong>{valueObj.title}</strong><br />
        {valueObj.text}
        </td>
    </tr>
    {else:}
    <tr>
        <td>{translate(#No Events found#)}.</td>
    </tr>
    {end:}
</table>
```

Listing 7. *Metody add() i insert()*

```
function _add(&$input, &$output)
{
   SGL::logMessage(null, PEAR_LOG_DEBUG);
   $output->template = 'eventEdit.html';
   $output->action   = 'insert';

   // generowanie kontrolek wyboru daty początkowej i daty wygaśnięcia
   $output->todaysDate = SGL::getTime();
   list($day, $month, $year, $hour, $minute, $second) =
      explode('/', date('d/m/Y/H/i/s'));
   // inicjalizacja tablicy wejściowej bieżącą datą i godziną
   $aDate = array( 'day' => $day, 'month' => $month, 'year' => $year);
   $output->dateSelectorStart =
      SGL_Output::showDateSelector($aDate, 'frmStartDate', false);
   $output->dateSelectorExpiry =
      SGL_Output::showDateSelector($aDate, 'frmExpiryDate', false);
}

function _insert(&$input, &$output)
{
   SGL::logMessage(null, PEAR_LOG_DEBUG);

   // insert record
   $event = & new DataObjects_Event();
   $event->setFrom($input->event);
   $dbh = $event->getDatabaseConnection();
   $event->event_id = $dbh->nextId('event');
   $event->last_updated = $event->date_created = SGL::getTime(true);
   $event->date_start = SGL_Date::arrayToString($input->aStartDate);
   $event->date_expire = SGL_Date::arrayToString($input->aExpiryDate);
   $success = $event->insert();
   if ($success) {
     SGL::raiseMsg('Termin zapisany pomyślnie');
   } else {
   SGL::raiseError('Błąd podczas wstawiania rekordu', SGL_ERROR_NOAFFECTEDROWS);
   }
}
```

wiliśmy jeszcze w systemie uprawnień dostępu do naszego modułu. Na szczęście jest to bardzo proste. W module *User & Security* wybieramy *manage permissions*, po czym naciskamy przycisk *detect and add*, zaznaczamy wszystkie uprawnienia dla modułu *event* i dodajemy je klikając raz jeszcze listę uprawnień.

Teraz możemy przydzielić nowe uprawnienia rolom. Na przykład gość (rola *Guest*) otrzyma uprawnienie *eventmgr_list*, dzięki czemu będzie mógł przeglądać wszystkie terminy. Uprawnienia danego gościa będą działać również wtedy, gdy w międzyczasie wyloguje się on lub zamknie przeglądarkę, gdyż są one zapisywane w trwałym pliku cookie na komputerze lokalnym.

Zarejestrowani użytkownicy (rola *Member*) powinni dodatkowo mieć możliwość dodawania i aktualizacji terminów, ale samo dodanie tych uprawnień do roli jeszcze nie nadaje ich użytkownikom. Dzieje się tak dlatego, że zmiana roli nie powoduje automatycznej aktualizacji uprawnień wszystkich użytkowników: musimy to zrobić przeprowadzając ręczną synchronizację uprawnień, co sprowadza się do kliknięcia funkcji *sync perms with role*. Możliwe jest wybranie użytkowników do synchronizacji, określenie, czy mają być synchronizowani z rolą bieżącą czy też inną oraz wskazanie, czy chcemy dodać brakujące uprawnienia, usunąć nadmiarowe, czy też jedno i drugie. Jak widać, wykorzystanie tej funkcji bardzo pomaga w zarządzaniu uprawnieniami i ich bieżącej aktualizacji.

Listing 8. *Walidacja danych wprowadzonych przez użytkownika*

```
function validate($req, &$input)
{
   SGL::logMessage(null, PEAR_LOG_DEBUG);
   $this->validated    = true;
   $input->error       = array();
   $input->pageTitle   = $this->pageTitle;
   $input->masterTemplate = $this->masterTemplate;
   $input->template    = $this->template;
   // definicje zmiennych dla wartości pobieranych od użytkownika
   $input->action      = ($req->get('action')) ? $req->get('action') : 'list';
   $input->aDelete     = $req->get('frmDelete');
   $input->submit      = $req->get('submitted');
   $input->event       = (object)$req->get('event');
   $input->aStartDate  = $req->get('frmStartDate');
   $input->aExpiryDate = $req->get('frmExpiryDate');
   // sprawdzenie poprawności
   if ($input->submit) {
      // tytuł jest obowiązkowy
      if (empty($input->event->title)) {
         $aErrors['title'] = 'Proszę podać tytuł';
      }
      // generowanie znaczników czasu
      $startDate = mktime(0,0,0,$input->aStartDate['month'],
         $input->aStartDate['day'],$input->aStartDate['year']);
      $expiryDate = mktime(0,0,0,$input->aExpiryDate['month'],
         $input->aExpiryDate['day'],$input->aExpiryDate['year']);
      if ($startDate > $expiryDate) {
         $aErrors['date_expiry'] = 'Data wygaśnięcia musi być późniejsza
            od daty początkowej';
      }
   }

   // jeśli wystąpiły błędy
   if (isset($aErrors) && count($aErrors)) {
      SGL::raiseMsg('Proszę wypełnić wskazane pola');
      $input->error = $aErrors;
      $input->template = 'eventEdit.html';
      // generowanie kontrolek wyboru daty
      $input->dateSelectorStart = SGL_Output::showDateSelector($input->aStartDate,
        'frmStartDate', false);
      $input->dateSelectorExpiry =
         SGL_Output::showDateSelector($input->aExpiryDate,
        'frmExpiryDate', false);

      $this->validated = false;
   }
}
```

Listing 9. *Warunek w metodzie list(), decydujący o wyświetlaniu szablonu administratora*

```
if (SGL_HTTP_Session::getUserType() == SGL_ADMIN) {
   $output->template = 'eventListAdmin.html';
   $output->pageTitle = $this->pageTitle . ' :: Browse';
} else {
   $output->pageTitle = 'Events'; // używamy szablonu standardowego
}
```

Blok dla nowego modułu

Na zakończenie możemy jeszcze dodać nowy blok dla naszego modułu. W tym celu musimy stworzyć w katalogu *modules/block/classes/blocks/* nową klasę bloku – jej kod przedstawia Listing 10. Następnie dodajemy blok korzystając z formularza *New Block* w menedżerze bloków. Podajemy nazwę *EventBlock* i tytuł *Terminarz* oraz określamy, w których sekcjach i z której części strony blok powinien się pojawić. Po wysłaniu formularza blok zostanie utworzony i zapisany, ale domyślnie będzie on nieaktywny. Pozostaje zatem go aktywować – otwieramy blok do edycji i zaznaczamy pole *activated*. Teraz powinien on już być widoczny w naszym portalu.

Korzystanie z dzienników podczas tworzenia aplikacji

Przydatne informacje można często uzyskać śledząc dziennik generowany przez Seagull podczas tworzenia i testowania nowego modułu. W systemach uniksowych służy do tego polecenie:

```
tail -f [ścieżka/do/seagull]
  /var/log/php_log.txt
```

Opcja `-f` nakazuje poleceniu *tail* nasłuchiwać zmian w pliku dziennika, dzięki czemu wpisy dodawane do dziennika są wyświetlane w oknie konsoli.

Zmiana wyglądu i zachowania za pomocą motywów

Wszystko pięknie, ale nasz portal nadal wygląda jak świeża instalacja Seagull. Na szczęście bardzo łatwo możemy to zmienić – wystarczy zmodyfikować dwa, może trzy pliki, a wkrótce całość będzie wyglądać zupełnie inaczej. Zdefiniowanie własnego wyglądu i zachowania aplikacji wymaga stworzenia własnego motywu poprzez utworzenie w katalogu *seagull/www/themes* nowego podkatalogu, na przykład o nazwie *myTheme*. Do podkatalogu *default* katalogu nowego motywu kopiujemy teraz szablony modułu domyślnego z katalogu *www/themes/default/default/* . To samo robimy w przypadku podkatalogów *css* i *images*. Kopiowanie pozostałych szablonów do nowego motywu nie jest konieczne, jeśli nie chcemy ich zmieniać – gdy Seagull nie znajdzie potrzebnego szablonu w naszym motywie, to automatycznie pobierze go z motywu domyślnego. Dużą część wyglądu można zmienić po prostu modyfikując definicje stylów CSS, bez konieczności ruszania kodu XHTML w szablonach. Prawda, że łatwe? Dzięki temu nowe motywy są zawsze aktualne, nawet jeśli szablony modułu ulegną zmianie. Tworzenie nowego motywu tylko po to, by zmienić kilka plików może się wprawdzie wydawać przesadne, ale wygodniej jest dzięki niemu oddzielić własne dodatki od domyślnego kodu Seagull – dzięki temu późniejsze aktualizacje środowiska nie będą miały wpływu na zdefiniowane przez nas motywy.

Spróbujmy wprowadzić zmiany w dwóch plikach z katalogu *default* naszego motywu: *banner.html* i *footer.html*. Jest tam jeszcze plik *header.html*, jednak zawiera on tylko kod pomiędzy otwierającym znacznikiem `<html>` a końcem otwierającego znacznika `<body>`, więc nie ma potrzeby go modyfikować. Najważniejsze zmiany wyglądu wprowadzamy w głównym pliku CSS – *seagull/www/themes/myTheme/css/style.php*.

Nowy motyw aktywujemy dodając go do preferencji globalnych. W tym celu zmieniamy wartość preferencji *theme* na nazwę katalogu nowego motywu (w tym przypadku *myTheme*), *et voilà* – oto modyfikując kilka plików stworzyliśmy całkowicie nowy układ aplikacji.

Listing 10. *Blok dla terminu*

```
<?php
require_once SGL_ENT_DIR . '/Event.php';
/**
 * Blok dla terminu
 * @package event
 * @author  Werner M. Krauß <werner.krauss@hallstatt.net>
 * @version 1.0
 * @since   PHP 4.1
 */
class EventBlock
{
   var $webRoot = SGL_BASE_URL;
   function init()
   {
      SGL::logMessage(null, PEAR_LOG_DEBUG);
      return $this->getBlockContent();
   }
   function getBlockContent()
   {
      SGL::logMessage(null, PEAR_LOG_DEBUG);
      $theme = $_SESSION['aPrefs']['theme'];
      $aResult = $this->retrieveAll();
      $eventList = $this->toHtml($aResult);
      return $eventList;
   }
    function retrieveAll()
    {
       $eventList = & new DataObjects_Event();
       $eventList->orderBy('date_expire');
       $eventList->limit(5);
       // klauzula where w celu pobrania tylko aktualnych wydarzeń
       list($day, $month, $year) = explode('/', date('d/m/Y'));
       // inicjalizacja tablicy wejściowej bieżącą datą i godziną
       $aDate = array( 'day' => $day,'month' => $month,'year' => $year);
       $eventList->whereAdd('date_expire >= \''. SGL_Date::
                     arrayToString($aDate).'\'');
       $result = $eventList->find();
       $aEvents = array();
       if ($result > 0) {
          while ($eventList->fetch()) {
              // sprawdza, czy data początkowa i data wygaśnięcia są równe.
              // jeśli tak, wyświetla w szablonie tylko jedną datę.
              if ($eventList->date_start == $eventList->date_expire)
                $eventList->date_start = '';
              $aEvents[] = clone($eventList);
          }
       }
      return $aEvents;
   }
{...}
```

Podsumowanie

Przedstawione w artykule możliwości i budowa środowiska Seagull PHP Framework pokazują jego potęgę. Łatwość rozbudowy wynikająca z modularności, wielojęzyczność, zgodność ze standardami, popularność projektu, duży zakres gotowych komponentów oraz cechy łączące światy CMS-ów i frameworków czynią z niego idealne rozwiązanie dla każdego, kto pracuje nad średnimi lub dużymi projektami w PHP, wymagającymi zaawansowanego zarządzania użytkownikami, uprawnieniami i treścią. ■

O autorze

Werner M. Krauß *dołączył do społeczności twórców Seagull w kwietniu 2004, gdy szukał dobrego środowiska PHP dla swoich projektów. Obecnie zajmuje się utrzymywaniem wiki z dokumentacją projektu. W wolnych chwilach grywa na gitarze w zespole muzyki ludowej Schwartenhals.*
Kontakt z autorem:
werner.krauss@hallstatt.net