Jacek Ablewicz[1], Rafał Kopaczka[2]
1 Biblioteka Politechniki Krakowskiej im. Tadeusza Kościuszki
abl@biblos.pk.edu.pl
2 Biblioteka Krakowskiej Akademii im. Andrzeja Frycza Modrzewskiego
rkopaczka@afm.edu.pl

SYSTEM KOHA – PODSTAWOWE ZAGADNIENIA INFORMATYCZNE

Słowa kluczowe: Koha – architektura systemu, Koha – narzędzie informatyczne, Koha – wymagania, Koha – skalowalność, Koha – społeczność, Koha – perspektywy rozwoju, Koha – protokoły wymiany danych, Koha – współpraca z innymi systemami

Streszczenie: Niniejszy rozdział stanowi próbę opisu wymagań systemu Koha z perspektywy informatyków wdrażających. Przedstawione zostały następujące kwestie i zagadnienia:

- architektura systemu (w zarysie) oraz interfejs użytkownika (przeglądarka WWW jako podstawowa forma dostępu),
- wymagania sprzętowe i programowe, skalowalność systemu,
- możliwe warianty instalacji i posadowienia – dostępne i w praktyce najczęściej stosowane rozwiązania,
- społeczność Koha i stosowane przez nią narzędzia programistyczne (git, git-bz, bugzilla, dashboard i inne); obowiązujące procedury zgłaszania, weryfikacji oraz zatwierdzania poprawek i rozszerzeń,
- współpraca i wymiana danych z innymi systemami i usługami (SIP2, self-check, RFID), protokoły: LDAP, CAS, Z39.50,
- perspektywy na przyszłość, między innymi kwestia dodania alternatywnych silników wyszukiwawczych (ElasticSearch, Solr).

1. Wprowadzenie

Głównym celem rozdziału jest przybliżenie technicznych i informatycznych zagadnień dotyczących instalowania i funkcjonowania systemu Koha. Przedstawione zostały podstawowe informacje o systemie i jego architekturze. Kolejne części poświęcone są wymaganiom Koha w odniesieniu do sprzętu i oprogramowania (ze szczególnym uwzględnieniem tych, które wpływają na koszty wdrożenia i utrzyma-

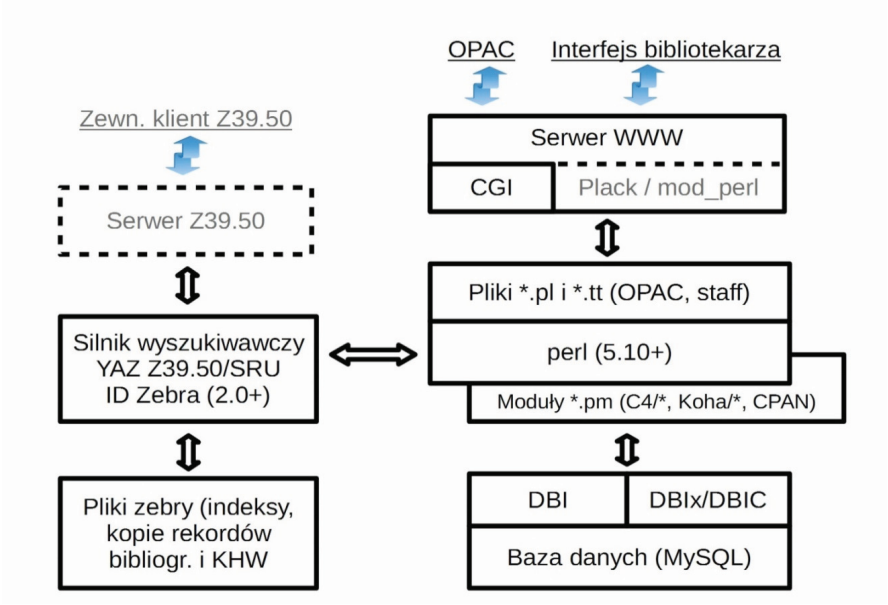
nia), jak również innym wybranym tematom dotyczącym informatycznych aspektów Koha (wydajność, skalowalność, dostępne warianty instalacji i posadowienia). W dalszych częściach tekstu omówiono takie zagadnienia jak: obsługiwane przez Koha protokoły, kwestie związane z rozwijaniem systemu (w szczególności: sposoby, techniki i obowiązujące w społeczności Koha procedury do tego celu stosowane) oraz wybrane perspektywy na przyszłość.

2. Podstawowe informacje o systemie

Koha jest systemem działającym w architekturze klient-serwer, co jest typowe dla współczesnych systemów bibliotecznych. Trudno byłoby wskazać przykład profesjonalnego systemu bibliotecznego, który nie działa w takiej architekturze. Ciekawszą (i mniej typową) kwestią jest to, że w przypadku Koha w charakterze klienta i zarazem interfejsu użytkownika stosowana jest przeglądarka WWW. Stanowi ona podstawową (i właściwie jedyną) formę dostępu do systemu zarówno dla czytelnika, jak i bibliotekarza.

Kolejnym istotnym czynnikiem wyróżniającym system Koha spośród innych dostępnych na rynku zintegrowanych systemów bibliotecznych (i również jedną z bardziej znaczących kwestii branych pod uwagę przez osoby czy instytucje rozważające wdrożenie) jest jego otwartość i darmowość. Wszystkie programowe komponenty samego systemu Koha (jak również oprogramowanie dodatkowe, niezbędne do funkcjonowania systemu w praktyce) dostępne są na darmowych licencjach *open source*. Konkretne rodzaje i warianty licencjonowania poszczególnych komponentów są dość zróżnicowane (GPL 2/3, BSD, MIT, Apache 2.0, Creative Commons itp.), ale wszystko wskazuje na to, że „darmowość i otwartość" są ich wspólną cechą, co do zasady również niezależnie od docelowych obszarów zastosowań (niekomercyjnych, komercyjnych, edukacyjnych itp.).

Rysunek 1. Architektura systemu Koha (w zarysie)



Źródło: opracowanie własne.

3. Wymagania programowe (serwer)

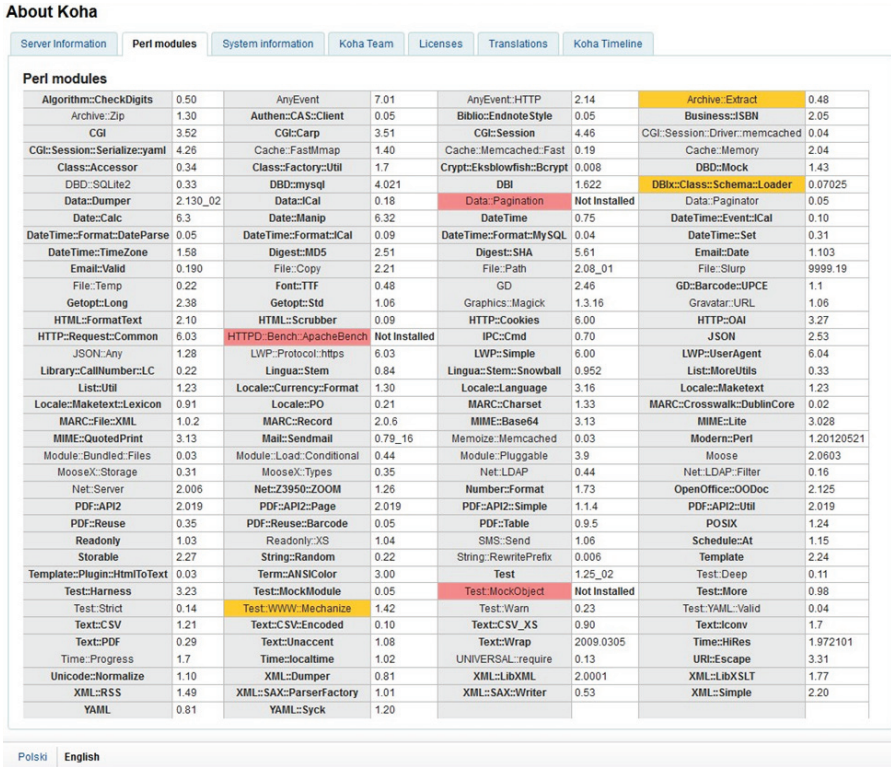
3.1. System operacyjny

Koha po stronie serwera wymaga systemu operacyjnego Unix. Teoretycznie mogą to być Unixy w różnych rodzajach i odmianach, ale w praktyce (również ze względów finansowych i licencyjnych) mocno zalecane jest, aby był to system Linux, w takiej czy innej dystrybucji. Najpopularniejszą i najbardziej optymalną dystrybucją Linuxa – pod względem łatwości instalowania Kohy oraz dalszego utrzymywania i aktualizowania systemu – wydaje się obecnie Debian (w bieżącej stabilnej wersji) oraz ewentualnie także inne wywodzące się z niego dystrybucje o „serwerowym" charakterze (np. Ubuntu LTS). O ile Koha daje się bez zasadniczych problemów uruchomić i utrzymać również na innych popularnych serwerowych dystrybucjach Linuxa (RHEL/CentOS/Scientific Linux, SLES, itp.), to trzeba liczyć się z tym, że tego typu rozwiązania będą wymagały istotnie większego nakładu pracy, jak również szeroko pojęta stabilność oraz wsparcie ze strony społeczności Koha będą najprawdopodobniej słabsze w takich przypadkach – w porównaniu z bardziej „typowymi" wariantami instalacji.

3.2. Perl

System Koha napisany jest w języku Perl (aktualnie wymagana wersja interpretera to 5.10 lub wyższa). Oprócz Perla jako takiego, Koha do działania wymaga zainstalowania stosunkowo dużej liczby najrozmaitszych dodatkowych modułów Perla (z repozytorium CPAN). Zależnie od tego, z jakiej dystrybucji Linuxa korzysta się na serwerze, niezbędne moduły Perla mogą być dostępne w formie gotowych pakietów w obrębie tejże dystrybucji (chociaż niekoniecznie zawsze w wystarczająco wysokich wersjach). W przeciwnym razie, ich „ręczna" instalacja bezpośrednio z archiwum CPAN może się okazać dość pracochłonnym i niewdzięcznym zadaniem. Na poniższym rysunku pokazano, jakich modułów Perla wymaga Koha (przykładowo w wersji 3.17). Należy również pamiętać, że wiele z tych modułów ma jeszcze własną, czasem dość długą listę niezbędnych prerekwizytów.

Rysunek 2. Wymagane moduły Perl CPAN (dla Koha 3.17.x)



Algorithm::CheckDigits	0.50	AnyEvent	7.01	AnyEvent::HTTP	2.14	Archive::Extract	0.48
Archive::Zip	1.30	Authen::CAS::Client	0.05	Biblio::EndnoteStyle	0.05	Business::ISBN	2.05
CGI	3.52	CGI::Carp	3.51	CGI::Session	4.46	CGI::Session::Driver::memcached	0.04
CGI::Session::Serialize::yaml	4.26	Cache::FastMmap	1.40	Cache::Memcached::Fast	0.19	Cache::Memory	2.04
Class::Accessor	0.34	Class::Factory::Util	1.7	Crypt::Eksblowfish::Bcrypt	0.008	DBD::Mock	1.43
DBD::SQLite2	0.33	DBD::mysql	4.021	DBI	1.622	DBIx::Class::Schema::Loader	0.07025
Data::Dumper	2.130_02	Data::ICal	0.18	Data::Pagination	Not Installed	Data::Paginator	0.05
Date::Calc	6.3	Date::Manip	6.32	DateTime	0.75	DateTime::Event::ICal	0.10
DateTime::Format::DateParse	0.05	DateTime::Format::ICal	0.09	DateTime::Format::MySQL	0.04	DateTime::Set	0.31
DateTime::TimeZone	1.58	Digest::MD5	2.51	Digest::SHA	5.61	Email::Date	1.103
Email::Valid	0.190	File::Copy	2.21	File::Path	2.08_01	File::Slurp	9999.19
File::Temp	0.22	Font::TTF	0.48	GD	2.46	GD::Barcode::UPCE	1.1
Getopt::Long	2.38	Getopt::Std	1.06	Graphics::Magick	1.3.16	Gravatar::URL	1.06
HTML::FormatText	2.10	HTML::Scrubber	0.09	HTTP::Cookies	6.00	HTTP::OAI	3.27
HTTP::Request::Common	6.03	HTTPD::Bench::ApacheBench	Not Installed	IPC::Cmd	0.70	JSON	2.53
JSON::Any	1.28	LWP::Protocol::https	6.03	LWP::Simple	6.00	LWP::UserAgent	6.04
Library::CallNumber::LC	0.22	Lingua::Stem	0.84	Lingua::Stem::Snowball	0.952	List::MoreUtils	0.33
List::Util	1.23	Locale::Currency::Format	1.30	Locale::Language	3.16	Locale::Maketext	1.23
Locale::Maketext::Lexicon	0.91	Locale::PO	0.21	MARC::Charset	1.33	MARC::Crosswalk::DublinCore	0.02
MARC::File::XML	1.0.2	MARC::Record	2.0.6	MIME::Base64	3.13	MIME::Lite	3.028
MIME::QuotedPrint	3.13	Mail::Sendmail	0.79_16	Memoize::Memcached	0.03	Modern::Perl	1.20120521
Module::Bundled::Files	0.03	Module::Load::Conditional	0.44	Module::Pluggable	3.9	Moose	2.0603
MooseX::Storage	0.31	MooseX::Types	0.35	Net::LDAP	0.44	Net::LDAP::Filter	0.16
Net::Server	2.006	Net::Z3950::ZOOM	1.26	Number::Format	1.73	OpenOffice::OODoc	2.125
PDF::API2	2.019	PDF::API2::Page	2.019	PDF::API2::Simple	1.1.4	PDF::API2::Util	2.019
PDF::Reuse	0.35	PDF::Reuse::Barcode	0.05	PDF::Table	0.9.5	POSIX	1.24
Readonly	1.03	Readonly::XS	1.04	SMS::Send	1.06	Schedule::At	1.15
Storable	2.27	String::Random	0.22	String::RewritePrefix	0.006	Template	2.24
Template::Plugin::HtmlToText	0.03	Term::ANSIColor	3.00	Test	1.25_02	Test::Deep	0.11
Test::Harness	3.23	Test::MockModule	0.05	Test::MockObject	Not Installed	Test::More	0.98
Test::Strict	0.14	Test::WWW::Mechanize	1.42	Test::Warn	0.23	Test::YAML::Valid	0.04
Text::CSV	1.21	Text::CSV::Encoded	0.10	Text::CSV_XS	0.90	Text::Iconv	1.7
Text::PDF	0.29	Text::Unaccent	1.08	Text::Wrap	2009.0305	Time::HiRes	1.972101
Time::Progress	1.7	Time::localtime	1.02	UNIVERSAL::require	0.13	URI::Escape	3.31
Unicode::Normalize	1.10	XML::Dumper	0.81	XML::LibXML	2.0001	XML::LibXSLT	1.77
XML::RSS	1.49	XML::SAX::ParserFactory	1.01	XML::SAX::Writer	0.53	XML::Simple	2.20
YAML	0.81	YAML::Syck	1.20				

Źródło: Koha wersja 3.17.

3.3. Silnik wyszukiwawczy

Jako silnik wyszukiwawczy w Koha (w wersjach od 3.0 wzwyż) wykorzystywane jest oprogramowanie firmy IndexData (Zebra w wersji 2.0+ wraz z biblioteką YAZ Toolkit). Oprogramowanie to spełnia również w tym systemie opcjonalnie funkcję serwera Z39.50, z czym mogą wiązać się pewne dodatkowe komplikacje. W szczególności wynika z tego to, że w rekordach MARC opisów bibliograficznych wyszukiwanych za pośrednictwem Z39.50 zawarte są dane egzemplarzy (niezależnie od tego, czy chcemy i czy faktycznie potrzebujemy, aby tego typu dane były udostępniane na zewnątrz) oraz pewne sztucznie dodane pola i podpola spoza ogólnie przyjętego standardu MARC 21 (w przypadku gdy zależy nam, aby opisy dały się przeszukiwać z wykorzystaniem wariantów haseł KHW). Należy nadmienić, że możliwe są obejścia tego problemu – np. w formie serwera proxy Z39.50, który odfiltrowuje z rekordów MARC niepotrzebne dane „w locie". Nie są one jednak łatwe w instalacji i konfiguracji, jak również ich funkcjonalność pozostawia w praktyce sporo do życzenia[1]. Ogólnie rzecz biorąc, ID Zebra jako silnik wyszukiwawczy to współcześnie raczej mocno kompromisowe rozwiązanie – o ile same możliwości wyszukiwawcze tego silnika są stosunkowo duże (plusem jest również wysoki poziom jego przystosowania do specyfiki rekordów bibliograficznych), to jest to rozwiązanie mało eleganckie i mało optymalne, słabo skalowalne oraz mocno problematyczne pod wieloma innymi względami. W społeczności Koha dość powszechnie uznawane jest ono za jeden ze słabszych elementów obecnej wersji tego systemu; coraz większe obawy budzi też jego mała rozwojowość, zwłaszcza w dłuższej perspektywie.

3.4. Pozostałe komponenty programowe

Koha nie ma ściśle określonych wymagań dotyczących serwerów WWW, z jakimi funkcjonuje. W praktyce jest to zwykle Apache (2.2+) albo Nginx. Opcjonalnie, w celu poprawy wydajności (czy raczej responsywności – czasu reakcji) stosowane mogą być także narzędzia informatyczne w rodzaju Plack/fastCGI albo mod_perl.

Baza danych (RDBMS): MySQL jest obecnie jedyną relacyjną bazą danych, z którą Koha „oficjalnie" współpracuje i dla której zapewnione jest wsparcie w ramach społeczności. Zalecana wersja MySQL to co najmniej 5.5 (wersja 5.6+ może być wskazana w polskich warunkach, ze względu na problemy z sortowaniem

[1] Dotyczy instalacji Zebry skonfigurowanych w tradycyjnym trybie (modelu) GRS1; w przypadku Zebry skonfigurowanej w modelu DOM XML odfiltrowywanie (selektywnie, np. na potrzeby publicznie dostępnego serwera Z39.50) nadmiarowych danych „w locie" jest trywialne i nie wymaga instalowania żadnego dodatkowego oprogramowania.

liter ‘ł’ i ‘Ł’ występujące we wcześniejszych wersjach). Istnieją również wdrożenia systemu Koha, w których zamiast MySQL używana jest baza MariaDB (raczej bez problemów – ew. z niewielkimi i łatwymi do poprawienia), i można chyba tę bazę danych ostrożnie zacząć traktować jako potencjalnie sensowną alternatywę dla MySQL w zakresie współpracy z systemem Koha. W odróżnieniu od MySQL nie jest to jednak na razie rozwiązanie dostatecznie przetestowane w praktyce.

4. Wymagania sprzętowe

4.1. Serwer

W teorii system Koha powinien działać na każdej platformie serwerowej, na której działa Unix/Linux i dla której dostępne są (tzn. dają się skompilować oraz funkcjonują prawidłowo) wcześniej wymienione podstawowe komponenty programowe systemu. W praktyce, analogicznie do wcześniej omówionych zagadnień dotyczących systemu operacyjnego, niestandardowe wersje sprzętu w charakterze serwera Koha nie są zalecane. W szczególności na nietypowych platformach sprzętowych (o architekturze CPU odbiegającej fundamentalnie od IA-32/x86-64/AMD64) można się spodziewać określonych trudności wynikających m.in. z innego *endianness*[2]. O ile można z dużym prawdopodobieństwem liczyć na to, że sam interpreter Perla (plus dystrybuowane wraz z nim tzw. *core modules*) będzie działać bezproblemowo, to inne wymagane przez Koha moduły Perla z CPAN w wariantach XS (tj. kompilowalne częściowo z języka C) – już niekoniecznie.

Wyjściowe wymagania systemu Koha w zakresie wydajności serwera, ilości pamięci RAM, niezbędnego miejsca na dysku itp. są raczej minimalne (przynajmniej teoretycznie). Konkretne, realne wymagania uzależnione są w zasadniczy sposób od wielkości i charakteru biblioteki wdrażającej. W szczególności od takich kwestii jak: wielkość zbiorów (liczba opisów bibliograficznych i rekordów KHW w bazie), liczba obsługiwanych użytkowników, oraz od przewidywanego obciążenia poszczególnych modułów i silnika wyszukiwawczego (średniego obciążenia, a także szacowanego „szczytowego” obciążenia w dających się przewidzieć okresach wzmożonego ruchu, charakterystycznego m.in. dla bibliotek akademickich, w których obciążenie systemu przez 2–3 określone miesiące w roku może być nawet o rząd wielkości większe niż średnia z całego roku).

Poniżej przedstawione zostały w zarysie dwie przykładowe konfiguracje serwera Koha („minimalna rozsądna” i „komfortowa, optymalna”), które rozważane były z myślą o wdrożeniu tego systemu w Bibliotece PK:

[2] Kolejność porządkowania bajtów w pamięci urządzenia.

- *case study* (Biblioteka PK): około 130 tys. opisów, 90 tys. rekordów KHW, 300 tys. rekordów egzemplarzy, 30 tys. rekordów użytkowników (aktywnych około 40–50%),
- konfiguracja systemu: sam system (serwer WWW + skrypty Perla), baza MYSQL oraz silnik wyszukiwawczy działające na pojedynczym (tym samym) fizycznym serwerze,
- rozmiar danych po migracji ze starego systemu – baza MySQL: około 2.5 GB (+ 1 GB pliki ib_logfile* InnoDB); spodziewany przyrost rzędu 1–1.5 GB rocznie (zależnie od tego, jakie dane archiwalne i jak daleko wstecz sięgające zamierzamy przechowywać, oraz na jakim poziomie szczegółowości skonfigurowane jest logowanie zdarzeń); indeksy Zebry: 4.5 GB,
- „minimalna rozsądna" konfiguracja (w rozumieniu: taka, na jakiej dałoby się pracować, ale niekoniecznie płynnie/komfortowo w okresie wzmożonego ruchu): 24–32GB RAM; procesor(y): 6–8 rdzeni, 2.0+ GHz,
- „komfortowa, optymalna" konfiguracja (z uwzględnieniem pewnego zapasu na przyszłość, rzędu 30%): 48–64GB RAM, CPU 8–10 rdzeni / 2 x 4–6 rdzeni, 2.4+ GHz; kontroler dysków z własną, podtrzymywaną bateryjnie pamięcią cache 1 GB+ (i/lub dyski SSD).

Można zauważyć, że konfiguracja określona powyżej jako „komfortowa, optymalna" zawiera stosunkowo dużą ilość pamięci RAM (nawet do 50% większą niż w niektórych przykładowych oszacowaniach dla bibliotek o zbliżonej wielkości i podobnym charakterze, jakie można znaleźć w Internecie, na stronach społeczności Koha itp.). Jest to pochodna testów obciążeniowych i wydajnościowych przeprowadzanych przez Bibliotekę PK przed wdrożeniem, z których wnioski (w pewnym uproszczeniu) wskazywały na to, że ilość RAM jest czynnikiem mającym w praktyce największy wpływ na szybkość, płynność działania i czas reakcji systemu, zwłaszcza pod średnim i dużym obciążeniem. Testy te wypadały zdecydowanie bardziej zadowalająco w sytuacjach, kiedy zapas dostępnej wolnej pamięci RAM był wystarczająco duży na to, aby wszystkie pliki samego systemu (1 GB), pliki Zebry (4.5 GB), pamięć przewidziana dla szeroko pojętej obsługi bazy danych (2 x 2.5 GB + 1.5 GB + 1 GB + 1 GB), we wszystkich przewidywalnych okolicznościach mogły w miarę możliwości rezydować w pamięci cache (buforach zapisu/odczytu) systemu operacyjnego. Innymi słowy, im rzadziej zachodziła konieczność odwoływania się do danych przechowywanych na fizycznych dyskach, tym wyraźnie lepsze były efekty testów obciążeniowych.

4.2. Komputery pracowników i stanowiska OPAC

Wymagania sprzętowe i programowe po stronie użytkownika (tj. w rozważanym przypadku dotyczące *de facto* przeglądarek WWW, wykorzystywanych na komputerach pracowników oraz na stanowiskach OPAC) nie są dla systemu Koha zbyt wygórowane. Warto jednak, zwłaszcza przy szacowaniu niezbędnych nakładów na sprzęt przed wdrożeniem systemu, wziąć pod uwagę m.in. następujące kwestie:

- interfejs bibliotekarza: jako stacje robocze zalecane są komputery z zainstalowaną „typową" przeglądarką WWW (Firefox, Chrome, IE 9.0+) w możliwie aktualnej wersji,
- system operacyjny: obojętnie jaki, liczy się wersja przeglądarki i jej zgodność ze standardami W3C,
- szybkość działania, ilość pamięci RAM: Koha po stronie klienta dość intensywnie korzysta m.in. z rozmaitych funkcji i bibliotek Javascript (JQuery + DataTables), stąd strony WWW, które system generuje, mogą być miejscami dość „ciężkie" – na słabych komputerach takie strony się wprawdzie prędzej bądź później wczytają i wyświetlą, ale już np. filtrowanie, stronicowanie i sortowanie wyników w tabelach generowanych przy pomocy DataTables może działać niezbyt zadowalająco,
- istotna jest również rozdzielczość ekranów (ew. konieczność przewijania stron w przeglądarce, zwłaszcza w poziomie, znacząco wpływa na komfort pracy) – dobrze, aby były to monitory o rozdzielczości co najmniej 1680x1050 (WSXGA+), w szczególności w oddziale udostępniania,
- stanowiska OPAC: wymagania co do wydajności urządzeń oraz zgodności przeglądarek WWW ze standardami są zdecydowanie łagodniejsze niż dla stacji roboczych; w obecnej wersji Koha, OPAC funkcjonuje w oparciu o framework Bootstrap – framework ten kładzie duży nacisk na zgodność z możliwie jak największą liczbą rozmaitych wersji przeglądarek i wariantów urządzeń klienckich, tradycyjnych czy mobilnych, i nieźle się sprawdza w praktyce w zastosowaniach, w których rozrzut parametrów sprzętowych czy wersji przeglądarek WWW bywa stosunkowo duży,
- czytelnicy używający przeglądarek WWW z wyłączonym JavaScript i/lub zablokowaną obsługą „ciasteczek" mogą (w najlepszym razie) korzystać z systemu jedynie w bardzo ograniczonym zakresie.

5. Skalowalność i ograniczenia systemu Koha w kwestii wydajności

Pod względem skalowalności Koha jest systemem sytuującym się na względnie dobrym poziomie. Istnieją instalacje Koha obsługujące w wersjach produkcyjnych setki tysięcy, a nawet miliony opisów bibliograficznych, dziesiątki, setki tysięcy użytkowników oraz setki tysięcy transakcji udostępniania rocznie (w tym m.in. wdrożenia w dużych i bardzo dużych bibliotekach uniwersyteckich)[3]. Można się tu jednak pokusić o zidentyfikowanie potencjalnie słabych punktów. Główny mankament wydaje się stanowić ograniczona skalowalność silnika wyszukiwawczego (Zebry), w dalszej kolejności także ewentualnie skalowalność czy niedostateczna wydajność silnika bazy danych.

Zebra nieźle skaluje się w zakresie od około miliona do paru milionów zaindeksowanych rekordów. Powyżej tej liczby pojawiają się mniejsze lub większe komplikacje i coraz bardziej dostrzegalne opóźnienia w działaniu, m.in. przy operacjach modyfikacji indeksów „w locie". Takich operacji może być w bardziej obciążonych instalacjach stosunkowo dużo, ponieważ oprócz samych opisów i rekordów KHW, dane z rekordów egzemplarzy, takie jak terminy zwrotu czy sumaryczne liczniki wypożyczeń i prolongat, również podlegają indeksowaniu. Zatem duże obciążenie modułu udostępniania przekłada się nie wprost także na zwiększone obciążenie silnika wyszukiwawczego. Większy problem w tym, że możliwości zrównoleglania operacji wyszukiwania są w Zebrze bardzo ograniczone. Obciążenie silnika wyszukiwawczego da się co prawda rozkładać na 2+ fizyczne serwery, ale samych atomicznych operacji wyszukiwania nie da się niestety w Zebrze znacząco przyspieszać ani zrównoleglić. Jako że tego typu operacje wyszukiwawcze mają (w dużym uproszczeniu) złożoność obliczeniową rzędu O(log N), wraz ze wzrostem rozmiarów indeksów spada wydajność wyszukiwania oraz wydłuża się czas oczekiwania na wyniki. Początkowo ten spadek wydajności jest w praktyce nieznaczny i trudno dostrzegalny, ale w bardzo dużych instalacjach zaczyna stopniowo stanowić coraz większy i trudniejszy do rozwiązania problem.

Ponieważ MySQL to obecnie jedyna baza danych, z którą Koha oficjalnie funkcjonuje, brak możliwości wyboru alternatywnego (szybszego, lepiej zoptymalizowanego itp.) silnika bazy danych również można wskazać jako potencjalnie znaczące ograniczenie pod względem wydajności i skalowalności. Podejmowane od dość dawna w ramach społeczności Koha próby dostosowywania systemu do współpracy z bazą danych PostgreSQL są wciąż odległe od osiągnięcia etapu, na którym można by na poważnie rozważać zastosowanie tego rozwiązania w wersjach pro-

[3] Na portalu Library Technology Guides [Breeding, 2015] zgłoszonych jest dziewięć bibliotek używających systemu Koha z ponad milionem egzemplarzy, w tym m.in. biblioteki narodowe Wenezueli i Filipin.

dukcyjnych. Można zresztą odnieść wrażenie, że główną motywacją dla tych prób jest w większym stopniu chęć uniezależnienia się od MySQL, ze względu na niepewność dalszych losów tej bazy danych po przejęciu MySQL przez firmę Oracle, a w mniejszym stopniu ewentualne kwestie wydajnościowe. Trwają również (i nabierają ostatnio tempa) prace nad wykorzystaniem w kodzie Koha technik programistycznych typu ORM (DBIx: Class, DBIC), co z czasem, teoretycznie, powinno poskutkować pełnym uniezależnieniem od konkretnych systemów RDBMS. Trudno jednak na tym etapie przewidzieć, jak to się docelowo odbije na wydajności systemu – w małych i prostych instalacjach przypuszczalnie raczej negatywnie, ze względu na dodatkowe narzuty ze strony DBIx.

Dość słabym punktem systemu Koha jest jego niezbyt zadowalająca responsywność[4]. Problem polega na tym, że w standardowej konfiguracji systemu (tj. w „zwykłym", bezstanowym środowisku CGI) średnio nawet ponad 80% czasu wykonania przeciętnego skryptu *.pl traci się na każdorazowe wstępne ładowanie i kompilację rozlicznych modułów Perla (typowo zajmuje to ponad 500 ms, nawet na szybkim procesorze). Dostępne rozwiązania tego problemu (środowiska uruchomieniowe fastCGI/Plack/mod_perl itd.), pozwalające na skrócenie o rząd wielkości wstępnego czasu wykonania skryptów *.pl i na zasadniczą poprawę średniego czasu reakcji systemu, są w przypadku interfejsu OPAC z powodzeniem już wykorzystywane w wielu produkcyjnych instalacjach Koha. Natomiast w przypadku interfejsu bibliotekarza można mieć pewne wątpliwości, czy na tym etapie rozwiązania te są już na tyle stabilne i wystarczająco dobrze przetestowane w praktyce, aby je bez żadnych obaw stosować w wersjach produkcyjnych.

Podsumowując temat wydajności i skalowalności, Koha ze swej natury (chociażby ze względu na charakterystykę języka Perl, w którym jest napisany) nie jest nadprzeciętnie „lekkim" ani też wyjątkowo dobrze zoptymalizowanym systemem bibliotecznym – i w związku z tym przesadnie daleko idące oszczędności (np. na sprzęcie serwerowym) są w jego przypadku raczej niewskazane. Wnosząc jednak z doświadczeń autorów, system ten nieźle sprawdza się pod tymi względami w bibliotekach akademickich średniej wielkości. W przypadku dużych i bardzo dużych bibliotek trzeba się jednak liczyć z rozmaitymi problemami dotyczącymi wydajności i skalowalności (i tylko częściowo, do pewnej granicy, dającymi się kompensować dzięki nieproporcjonalnie dużym nakładom finansowym na sprzęt).

4 W znaczeniu „czas reakcji" – łączny czas oczekiwania na efekt (wygenerowanie, załadowanie i wyświetlenie wynikowej strony) np. po kliknięciu przez użytkownika określonego przycisku lub linku w przeglądarce WWW stanowiącej interfejs do systemu.

6. Możliwe warianty instalacji i posadowienia systemu

W kwestii dostępnych sposobów instalacji i posadowienia, Koha jest systemem wyjątkowo mało wymagającym, tj. nie narzuca żadnych istotnych ograniczeń co do możliwych wariantów posadowienia. Najczęściej spotykane w praktyce warianty to między innymi:

- własny serwer, fizyczny albo wirtualny (w miarę możliwości, optymalnie – dedykowany wyłącznie albo przede wszystkim pod system biblioteczny); może być utrzymywany lokalnie, zdalnie, dzierżawiony (np. VPS) itp.,
- wspólny serwer (lub serwery: w szczególności MySQL i Zebra mogą być posadowione na innych maszynach niż sam system), współdzielony przez wiele osobnych jednostek organizacyjnych, instytucji itp. (w różnych wariantach: osobne/wspólne/częściowo wspólne poszczególne komponenty systemu); w przypadku różnych instytucji – bazy danych raczej osobne,
- wykupiona na zewnątrz usługa, całościowo obejmująca wszystkie kwestie związane z zapewnieniem dostępu, bieżącego utrzymania, aktualizacji i obsługi informatycznej systemu bibliotecznego,
- inne rozmaite odmiany i kombinacje trzech wyżej wymienionych rozwiązań.

Przy wyborze konkretnego rozwiązania jedną z bardziej kluczowych kwestii do rozważenia jest także zapewnienie odpowiednio wysokiej jakości obsługi informatycznej: wsparcia i utrzymania, zarówno serwera, jak i samego systemu bibliotecznego. Koha, jako system o raczej ponadprzeciętnie wysokim poziomie złożoności, ma pod tym względem stosunkowo duże wymagania.

7. Protokoły obsługiwane przez Koha

Ze względu na użytkowanie Koha przez wiele rozmaitych instytucji z całego świata, zaimplementowany został szereg protokołów – zarówno typowo bibliotecznych, jak i ogólnie stosowanych.

Do autoryzacji użytkowników, poza standardowym użyciem loginu i hasła, można wykorzystać protokoły:

- LDAP
- CAS
- Shibboleth
- Mozilla Persona.

Konfiguracja i sposób wykorzystania powyższych protokołów zależy oczywiście od konkretnej instytucji.

Kolejny rodzaj protokołów jest ściśle związany z obsługą biblioteki. Zdefiniowane są w nich polecenia dotyczące obsługi czytelników, wypożyczeń, zwrotów książek itp. Obecnie dostępne są dwa tego typu protokoły. ILS-DI (Integrated Library System – Discovery Interface), protokół stworzony przez DLF (Digital Library Fundation) w celu ustandaryzowania komunikacji pomiędzy systemami bibliotecznymi oraz usługami z nimi współpracującymi. Zaimplementowano niemal wszystkie polecenia definiowane przez protokół. Obecnie nie są prowadzone żadne prace nad rozwojem tej funkcjonalności.

Jednym z ciekawszych protokołów udostępnianych przez Koha jest SIP2 (Standard Interchange Protocol). Jest to protokół opracowany przez firmę 3M, początkowo na potrzeby współpracy ich urządzeń z systemami bibliotecznymi. Specyfikacja została jednak udostępniona przez producenta, dzięki czemu w niedługim czasie stał się on standardowym protokołem. Obecnie jego obsługą może pochwalić się większość zintegrowanych systemów bibliotecznych. Używany jest przede wszystkim do obsługi czytelników przez stacje samoobsługi oraz do współpracy z systemami RFID. Koha obsługuje oba rodzaje usług. Do prawidłowej pracy wymagana jest odpowiednia, w zależności od urządzenia, konfiguracja serwera SIP2 dostarczanego wraz z Koha. Współpraca ze stacjami samoobsługi może być realizowana przy użyciu samego systemu bądź z wykorzystaniem oprogramowania dostarczanego przez producenta urządzenia. W takim przypadku wykorzystywany jest do komunikacji protokół SIP2.

Współpraca z RFID obecnie jest nieco mniej rozwinięta. Większych problemów nie przewiduje się przy korzystaniu z usług RFID dedykowanych dla bibliotek. Taka konfiguracja jest sprawdzona i używana przez co najmniej kilkanaście bibliotek na świecie. Korzystając z gotowych rozwiązań dedykowanych, dostajemy w pakiecie, poza samymi urządzeniami do czytania kodów RFID, oprogramowanie zapewniające pełną funkcjonalność takiego systemu. Dodatkowe oprogramowanie jest obecnie niezbędne do prawidłowej i pełnej współpracy Koha z systemem RFID. Jednakże trwają prace nad implementacją RFID bezpośrednio w systemie Koha. Dzięki takiemu rozwiązaniu do wdrożenia systemu RFID w bibliotece wystarczający będzie czytnik oraz tagi RFID. Rozwiązanie takie jest oczywiście dużo tańsze od systemu dedykowanego. Przykładowe wdrożenie tego typu rozwiązania zostało przedstawione przez Dobricę Pavlinušicia [Pavlinušić, 2015]. Używane jest obecnie przez co najmniej jedną bibliotekę. Kod źródłowy jest dostępny w formie repozytorium Git, pod adresem https://github.com/dpavlin/Biblio-RFID. Drugie, podobne rozwiązanie zaimplementowane zostało przez pracowników Oslo Public Library. Dokument opisujący tę implementację dostępny jest pod adresem http://wiki.koha-community.org/wiki/RFID_RFC, sposób działania jest podobny do tego zaproponowanego przez Pavlisnušicia. Również i w tym przypadku kod jest ogól-

nodostępny. Nie ma natomiast informacji o wdrożeniu tego rozwiązania w środowisku produkcyjnym, system jest ciągle w fazie testów i rozwoju.

Pozostałe protokoły dostępne w Koha dotyczą różnego rodzaju udostępniania bądź pobierania danych bibliograficznych. Dodawanie danych bibliograficznych do systemu możliwe jest poprzez protokoły Z39.50 oraz SRU (Search/Retrieve via URL). Udostępnianie danych bibliograficznych możliwe jest przy użyciu protokołów Z39.50 oraz OAI-PMH. Obsługa Z39.50 możliwa jest, jak już zostało wcześniej wspomniane, dzięki oprogramowaniu Zebra. Protokół OAI-PMH obsługiwany jest bezpośrednio przez sam system Koha. Poza włączeniem w opcjach obsługi OAI-PMH, nic więcej nie wymaga konfigurowania, nie ma również potrzeby przeindeksowywania danych bibliograficznych.

8. Rozwijanie systemu Koha

Do utrzymywania kontroli nad kodem źródłowym używany jest system kontroli wersji Git. Repozytorium kodu Koha jest dostępne dla każdego w trybie tylko do odczytu. Uprawnienia do zarządzania repozytorium posiadają wybrani członkowie społeczności. Dla każdej kolejnej wersji głównej (tzw. *major release*) wybierany jest spośród społeczności zespół osób sprawujących pieczę nad wydaniem oraz osoba odpowiedzialna za wydanie (*release manager*). Wybory odbywają się zazwyczaj po każdym wydaniu nowej wersji (co ok. 6 miesięcy: maj i październik) na kanale IRC #koha. Dodanie jakiejkolwiek poprawki do repozytorium wymaga przejścia przez procedurę „formalną" stosowaną przez społeczność. Wygląda ona, w skrócie, jak poniżej.

Pierwszym, najważniejszym krokiem jest zgłoszenie odpowiedniego błędu (nawet jeśli zmiana ma dotyczyć tylko rozszerzenia funkcjonalności systemu) w systemie zarządzania błędami Bugzilla. Zgłoszenia mogą być dodawane przez każdego zarejestrowanego użytkownika. Zgłoszenie błędu powinno zawierać:

- kroki niezbędne do odtworzenia, czyli jaka była dokładnie ścieżka działań w systemie przed wystąpieniem błędu,
- informację o błędzie podaną przez system, o ile taka się pojawiła; ewentualnie – jak system zareagował na działania informatyków w stosunku do zmian oczekiwanych,
- (ewentualnie) załącznik obrazu lub filmu z nieprawidłowym zachowaniem systemu.

Przy zgłaszaniu rozszerzenia należy opisać nowe funkcje, zmiany interfejsu, dodane lub modyfikowane preferencje systemowe. Dodawanie funkcji zmieniających działanie systemu musi być zawsze opcjonalne. Nowe wersje systemu nie mogą zmieniać aktualnie działających funkcji bez wyraźnej zgody

administratora lub bibliotekarza systemowego. W związku z tym nowe funkcje w systemie są dodawane, ale domyślnie wyłączone.

Poprawka (*patch*) błędu/zgłoszenia dodana może być przez każdego chętnego, niekoniecznie musi to być osoba zgłaszająca. Powinna zawierać między innymi elementy takie jak: numer błędu nadany przez Bugzilla, tytuł w formie krótkiego podsumowania oraz opisu błędu, najczęściej takiego jak w zgłoszeniu. Ponadto *patch* powinien zawierać „plan testów", to jest dokładny opis, jakie kroki należy wykonać, aby przetestować działanie poprawki i móc stwierdzić, że istotnie naprawia/rozszerza funkcjonowanie systemu oraz nie wprowadza regresji (utrata wcześniejszych funkcjonalności, wprowadzenie nowych błędów).

Kolejnym etapem, jest „podpisywanie" poprawki. Polega ono na przetestowaniu poprawki przez inną osobę. Najczęściej testowanie polega na wykonaniu „planu testów" załączonego do *patcha*. Jeżeli testy wypadną pomyślnie, poprawka może zostać podpisana, czyli zatwierdzona jako sprawdzona i działająca zgodnie z założeniami jej autora. Według obecnych zasad zatwierdzania poprawek preferuje się, chociaż nie jest to warunek konieczny, aby podpis składała osoba niezwiązana z firmą/organizacją autora poprawki. Pozwala to do pewnego stopnia uniknąć sytuacji, gdy zmiany forsowane są przez jednego producenta. Niestety obecnie, ze względu na zbyt małą liczbę wolontariuszy, dopuszcza się odstępstwa od tej reguły. Okazuje się bowiem, że programistów zgłaszających swoje poprawki jest dużo więcej niż wolontariuszy testujących te zgłoszenia. Z jednej strony sprawia to, że kod Koha rozwijany jest dość szybko, ale z drugiej – ciekawe i potrzebne funkcjonalności oczekują stosunkowo długo na dołączenie do oficjalnego wydania. Po podpisaniu przez co najmniej jedną osobę, status błędu zmieniany jest na *signed-off* (podpisany) i oczekuje on na ostateczną akceptację przez zespół ds. jakości (QA – Quality Assurance). Członkowie zespołu QA wybierani są na wcześniej opisanych zasadach. Najczęściej wykonywana przez nich praca ma charakter wolontariatu. Są oni odpowiedzialni za jakość kodu, który zostanie dołączony do Koha. Sprawdzana jest przede wszystkim poprawność „formalna" kodu przy użyciu narzędzia koha-qa.pl. Sprawdza ono, czy *patch* jest sformatowany poprawnie, czy nie zawiera błędów kompilacji. Wykrywane są również niektóre przestarzałe konstrukcje używane w kodzie, niedozwolone znaki itp. Następnie sprawdzane są podstawowe funkcjonalności systemu przy użyciu testów jednostkowych. Mają one na celu wykrycie ewentualnych regresji. Utworzenie procedur testów jednostkowych jest obecnie wymagane dla niemal każdej nowej funkcji/procedury/metody/klasy dodawanej do bibliotek w Koha. Odstępstwa od tej zasady są rzadkie i niechętnie przyjmowane. Jeżeli w poprawce dodawane są nowe funkcje i brakuje testów jednostkowych, najprawdopodobniej zostanie ona zatrzymana do czasu dodania potrzebnych testów. *Patch* sprawdzany jest również, według załączonego

planu testów, w ten sam sposób jak wcześniej przez osobę podpisującą poprawkę. Ostatnimi elementami sprawdzanymi przez zespół QA jest ogólna jakość kodu, zgodność z zasadami przyjętymi przez społeczność, m.in. konwencja nazw zmiennych, nowe procedury w odpowiednich modułach, sprawdzenie, czy podobne procedury już istnieją w bibliotekach i itp. Jeżeli pojawią się zastrzeżenia do poprawki, informacja odnośnie do tego, co należy poprawić, zostanie umieszczona w komentarzu do tego błędu w systemie Bugzilla. Po wymaganych modyfikacjach dodaje się nową „poprawkę do poprawek". Przechodzi ona tą samą procedurę co inne poprawki, chyba że zmiany w niej zawarte są kosmetyczne (literówka, zmiana nazewnictwa, formatowania). Jeżeli pojawią się zastrzeżenia do ogólnego sposobu rozwiązania problemu lub wprowadzanej nowej funkcji przez poprawkę, błędowi może zostać nadany status *In discussion* (do omówienia), gdzie każdy ma prawo się wypowiedzieć i przedstawić swoją wizję działania omawianego fragmentu systemu. Rozwiązywanie tego typu problemów jest jednak bardzo czasochłonne – może trwać nawet kilka lat.

Jeżeli wszystkie poprzednie etapy zakończyły się pomyślnie, poprawka uzyskuje status *Passed QA*. Oznacza to, iż może zostać włączona do kodu Koha przez menadżera wersji. Najczęściej tak właśnie się dzieje. W sporadycznych przypadkach poprawka może zostać odrzucona, z różnych względów. Poprawki i rozszerzenia funkcjonalności dodawane są przede wszystkim do wersji bieżącej, tzw. *master*. Jest to najbardziej aktualna wersja, z której wydzielane są następnie wersje główne. W przypadku poprawek mogą być one *backportowane* do wcześniejszych wersji, jeżeli błąd również w nich występował. Z zasady nowych funkcji nie dodaje się do istniejących już wersji głównych.

9. Uwagi końcowe. Perspektywy na przyszłość

Obecnie w systemie Bugzilla zgłoszonych jest ponad 1500 bugów, skategoryzowanych jako rozszerzenie bądź nowa funkcjonalność. Od bardzo prostych, polegających na dodaniu pól w formularzach, po skomplikowane, zmieniające zasadniczo (w sensie informatycznym) działanie całych modułów. Z ciekawszych poprawek oczekujących na kompletną implementację lub przetestowanie i wdrożenie można wymienić:

- obsługę nowego silnika wyszukiwawczego Elastic Search. Wśród zalet tego silnika wymienić należy dużo łatwiejszą konfigurację indeksów wyszukiwawczych, która w Zebrze dla osoby początkującej jest często barierą nie do przejścia. Ponadto silnik ten jest dużo lepiej skalowalny od Koha, chociaż z drugiej strony jego wymagania początkowe są dużo wyższe od Zebry, w szczególności w zakresie zapotrzebowania na pamięć RAM. Dlate-

go dla mniejszych baz, ze skromniejszymi zasobami sprzętowymi może to być rozwiązanie gorsze od Zebry,
- dodanie interfejsu profesjonalnego katalogera. Nowy wygląd formularza MARC istotny jest szczególnie dla bibliotek migrujących z innych (komercyjnych) systemów. Zbliżony jest do wyglądu formularza z oprogramowania Virtua, pozwoli na ręczne wpisywanie poszczególnych pól i podpól formatu MARC/UNIMARC, bez konieczności definiowania szablonów,
- dodanie obsługi protokołu OAI-PMH w formie klienta. Umożliwi to ściąganie opisów z dowolnego serwera OAI, np. z repozytorium cyfrowego instytucji,
- na ukończeniu są prace związane ze znacznym przyspieszeniem działania systemu, poprzez uruchamianie Koha w środowisku PLACK. Prawdopodobnie z nową wersją systemu (3.20), Koha bez większych problemów będzie mógł być uruchamiany w środowisku PLACK.

Ponadto, według długoterminowych założeń społeczności, planowana jest zmiana paradygmatu programowania, z proceduralnego na obiektowy. Obecnie większość kodu jest napisana proceduralnie, a ostateczny konsensus dotyczący szczegółów implementacji obiektów wciąż nie został osiągnięty.

Bibliografia

Pavlinušić Dobrica (2015). Dobrica Pavlinušić's Weblog/Blog. http://blog.rot13.org/projects/rfid/ [odczyt: 14.06.2015].

Breeding Marshall (2015). Library Automation Survey 2014: ILS System Report. http://librarytechnology.org/survey-2014-product-report.pl?CurrentILS=Koha%20–%20Independent [odczyt: 14.06.2015].