ТЕМА 12. ТЕСТУВАННЯ ВЕБ-СЕРВІСІВ

1. Що таке веб-сервіси?

Перш за все, веб-сервіси (або веб-служби) - це технологія. І, як і будь-яка інша технологія, вони мають досить чітке окреслене середовище застосування.

Якщо подивитися на веб-сервіси в розрізі стека мережевих протоколів, ми побачимо, що це, в класичному випадку, не що інше, як ще одна надбудова над протоколом HTTP.

З іншого боку, якщо гіпотетично розділити Інтернет на кілька шарів, ми зможемо виділити, як мінімум, два концептуальних типи додатків - обчислювальні вузли, які реалізують нетривіальні функції, та прикладні веб-ресурси. При цьому другі, часто зацікавлені в послугах перших.

Але і сам Інтернет - різнорідний. Різні додатки на різних вузлах мережі функціонують на різних апаратно-програмних платформах і використовують різні технології і мови.

Щоб зв'язати все це і надати можливість одним додаткам обмінюватися даними з іншими, і були придумані веб-сервіси.

Веб-сервіси - це реалізація абсолютно чітких інтерфейсів обміну даними між різними додатками, які написані не тільки на різних мовах, але і розподілені на різних вузлах мережі.

Саме з появою веб-сервісів розвинулася ідея SOA - сервіс-орієнтованої архітектури веб-додатків (Service Oriented Architecture).

2. Протоколи веб-сервісів

На сьогоднішній день найбільшого поширення набули такі протоколи реалізації веб-сервісів:

– SOAP (Simple Object Access Protocol) - по суті це трійка стандартів SOAP / WSDL / UDDI.

– REST (Representational State Transfer)

– XML-RPC (XML Remote Procedure Call)

Насправді, SOAP пішов від XML-RPC і є наступним етапом його розвитку. У той

час як REST - це концепція, в основі якої лежить швидше архітектурний стиль, ніж нова технологія, що базується на теорії маніпуляції об'єктами CRUD (Create Read Update Delete) в контексті концепцій WWW.

Безумовно, існують і інші протоколи, але, оскільки вони не набули широкого поширення, ми зупинимося в цьому короткому огляді на двох основних - SOAP і REST. XML-RPC з огляду на те, що є дещо «застарілим», ми розглядати детально не будемо.

Нас в першу чергу цікавлять питання створення нових веб-служб, а не реалізація клієнтів до існуючих (як правило постачальники веб-сервісів поставляють пакети з функціями API і документацією, тому питання побудови клієнтів до існуючих веб-служб менш цікавий з точки зору автора).

SOAP проти REST

SOAP більш застосовний в складних архітектурах, де взаємодія з об'єктами виходить за рамки теорії CRUD, а ось в тих додатках, які не покидають межі цієї теорії, цілком прийнятним може виявитися саме REST через свою простоту і прозорість. Дійсно, якщо будь-яким об'єктам вашого сервісу не потрібні більш складні взаємовідносини, крім: «Створити», «Прочитати», «Змінити»,

«Видалити» (як правило - в 99% випадків цього достатньо), можливо, саме REST стане правильним вибором. Крім того, REST в порівнянні з SOAP, може виявитися і більш продуктивним, так як не вимагає витрат на розбір складних XML команд на сервері (виконуються звичайні HTTP запити - PUT, GET, POST, DELETE). Хоча SOAP, в свою чергу, більш надійний і безпечний.

У будь-якому випадку вам вирішувати, що більше підійде вашому додатку. Цілком ймовірно, ви навіть захочете реалізувати обидва протоколи, щоб залишити вибір за користувачами служби і – це ваше право.

Практичне застосування веб-сервісів

Оскільки мова йде про практичне застосування, нам потрібно вибрати платформу для побудови веб-служби та поставити завдання. Виберемо PHP в якості технології для побудови служби, а в якості завдання приймемо наступні вимоги.

Припустимо, нам необхідно створити службу, яка надає доступ до інформації про курси валют, яка збирається нашим додатком, і накопичується в базі даних. Далі за

допомогою веб-сервісу, дана інформація передається стороннім додаткам для відображення в зручному для них вигляді.

Як бачимо завдання досить просте і, з точки зору самої служби, обмежується лише читанням інформації, але в практичних цілях нам цього буде достатньо.

Реалізація додатку збору інформації про курси валют.

Інформацію про курси валют будемо збирати із сторінок сайту НБУ щоденно та складати їх в базу даних під керівництвом СУБД MySQL.

Створимо структуру даних.

Таблиця валют (currency):

```
+-------------+------------------+
| Field       | Type             |
+-------------+------------------+
| code        | int(10) unsigned |
| charcode    | char(3)          |
| description | varchar(100)     |
| value       | int(10) unsigned |
| base        | tinyint(1)       |
+-------------+------------------+
```

Таблиця номіналів обміну (exchange):

```
+------------+------------------+
| Field      | Type             |
+------------+------------------+
| id         | bigint(20) ai    |
| rate_date  | timestamp        |
| rate_value | float            |
| code       | int(10) unsigned |
+------------+------------------+
```

Для роботи із базою даних скористаємось ORM рівнем в базі пакету PHP Doctrine. Реалізуємо граббер:

```
<?php
/*
 * @package Currency_Service
 */
class Grabber {
    /**
     * Extracts data from outer web resource and returns it
     *
     * @param  void
     * @return array
     */
    public static function getData() {
        /**
         * Extracting data drom outer web-resource
         */
        $content = file_get_contents(
'http://www.bank.gov.ua/Fin_ryn/OF_KURS/Currency/FindByDate.aspx');
        if(preg_match_all( '/(\d+)<\/td>([A-
Z]+)<\/td>(\d+)<\/td>(.+?)<\/td>(\d+\.\d+)<\/td>/i', $content, $m) ==
false) {
            throw new Exception( 'Can not parse data!');
        }
        /**
         * Preformatting data to return;
         */
        $data = array();
        foreach ($m[1] as $k => $code) {
            $data[] = array(
```

```
                'code'       =>     $code,
                'charcode'   => $m[2][$k],
                'value'    =>   $m[3][$k],
                'description'           =>
                $m[4][$k],    'rate_value'
                => $m[5][$k]
            );
        }
        return $data;
    }

    public static function run() {
        $data = self::getData();
        /**
         * Sets default currency if not exists
         */
        if (!Doctrine::getTable( 'Currency')->find( 980)) {
            $currency = new Currency();
            $currency->setCode( 980)
                     ->setCharcode( 'UAH')
                     ->setDescription( 'українська гривня')
                     ->setValue( 1)
                     ->setBase( 1)
                     ->save();
        }

        foreach ($data as $currencyData) {
            /**
             * Updating table of currencies with found values
             */
            if (!Doctrine::getTable( 'Currency')->find(
$currencyData['code'])) {
                $currency = new Currency();
                $currency->setCode( $currencyData['code'])
                         ->setCharcode( $currencyData['charcode'])
                         ->setDescription( $currencyData['description'])
                         ->setValue( $currencyData['value'])
                         ->setBase( 0)
                         ->save();
            }
            /**
             * Updating exchange rates
             */
            $date = date( 'Y-m-d 00:00:00');
            $exchange = new Exchange();
            $exchange->setRateDate( $date)
                     ->setRateValue( $currencyData['rate_value'])
                     ->setCode( $currencyData['code'])
                     ->save();
        }
    }
}
?>
```

І сам граббер (grabber.php):

```
<?php
require_once('config.php');
Doctrine::loadModels('models')
; Grabber::run();
?>
```

Тепер змусимо наш граббер працювати раз на добу в 10:00 ранку, шляхом додавання команди запуску граббера в таблиці cron:

```
0 10 * * * /usr/bin/php /path/to/grabber.php
```

Все – в нас є достатньо корисний сервіс.

Тепер реалізуємо веб-сервіс, котрий дозволить іншим додаткам видобувати дані із нашої бази.

Реалізація SOAP сервісу

Для реалізації веб-сервісу на базі SOAP протоколу, ми скористаємося вбудованим пакетом в PHP для роботи з SOAP.

Оскільки наш веб-сервіс буде публічним, хорошим варіантом буде створення WSDL файлу, який описує структуру нашого веб-сервісу.

WSDL (Web Service Definition Language) - вдає із себе XML файл певного формату.

На практиці буде зручно скористатися функцією автоматичної генерації файлу, яку надає IDE Zend Studio for Eclipse. Ця функція дозволяє генерувати WSDL файл з класів PHP. Тому, перш за все, треба написати клас, який реалізує функціональність сервісу.

клас CurrencyExchange (models / CurrencyExchange.php):

```
<?php
/**
 * Class providing web-service with all necessary methods
 * to provide information about currency exchange values
 *
 * @package Currency_Service
 */
class CurrencyExchange {
    /**
     * Retrievs exchange value for a given currency
     *
     * @param  integer $code - currency code
     * @param  string $data - currency exchange rate date
     * @return float - rate value
     */
    public function getExchange( $code, $date) {
        $currency = Doctrine::getTable( 'Currency')->find( $code);
        $exchange = $currency->getExchange( $date);
        return $exchange ? (float)$exchange->getRateValue() : null;
    }
    /**
     * Retrievs all available currencies
     *
     * @return array - list of all available currencies
     */
    public function getCurrencyList() {
        return Doctrine::getTable( 'Currency')->findAll()->toArray();
    }
}
?>
```

Відзначимо, що для автоматичної генерації WSDL, необхідно написати коментарі в стилі javadoc, тому що саме в них прописуємо інформацію про типи прийнятих аргументів і значень. Непогано також описувати в декількох словах роботу методів - адже WSDL послужить описом API для сторонніх розробників, які будуть використовувати ваш веб-сервіс.

Тепер в Zend Studio входимо в меню File-> Export ..., вибираємо PHP-> WSDL, додаємо наш клас, прописуємо URL-адресу сервісу і створюємо WSDL- файл.

Якщо ви будете додавати нову функціональність в ваш веб-сервіс, вам потрібно буде перетворити WSDL-файл. Але тут не так все легко. Слід враховувати, що SOAP-клієнт, який вже запитував ваш WSDL файл, кешує його на своїй стороні. Тому, якщо ви замініте старий вміст новим в WSDL файлі, деякі клієнти його не прочитають. А значить, при додаванні нової функціональності, дописуйте версію в ім'я вашого файлу. І не забудьте забезпечити сумісність для старих клієнтів, особливо якщо ви не є їх постачальником.

З іншого боку, WSDL досить жорстко задає структуру веб-сервісу, а це значить, що, якщо існує необхідність обмежити функціональність клієнта в порівнянні з сервером, ви можете не включати певні методи ваших класів в WSDL. Таким чином вони не зможуть бути викликані, незважаючи на те, що існують.

Реалізація ж самого серверу не представляє тепер ніякої складності:

файл index.php:

```
<?php require_once('config.php');

Doctrine::loadModels('models');

$server = new SoapServer( 'http://mikhailstadnik.com/ctws/currency.wsdl');
$server->setClass( 'CurrencyExchange');
$server->handle();
?>
```

Код найпростішого клієнту може бути наступним:

```
<?php
$client = new SoapClient( 'http://mikhailstadnik.com/ctws/currency.wsdl');
echo 'USD exchange: ' . $client->getExchange( 840, date( 'Y-m-d'));
?>
```

Реалізація REST сервісу

REST - це не стандарт і не специфікація, а архітектурний стиль, побудований на існуючих, добре відомих і контрольованих консорціумом W3C стандартах, таких, як HTTP, URI (Uniform Resource Identifier), XML і RDF (Resource Description Format). У REST-сервісах акцент зроблений на доступ до ресурсів, а не на виконання віддалених сервісів; в цьому їх кардинальна відмінність від SOAP-сервісів.

І все ж віддалений виклик процедур застосуємо і в REST. Він використовує методи PUT, GET, POST, DELETE HTTP протоколу для маніпуляції об'єктами. Кардинальна відмінність його від SOAP в тому, що REST залишається HTTP- запитом.

Оскільки в PHP поки ще немає реалізації REST, ми скористаємося Zend Framwork, в який включена реалізація як REST клієнта, так і REST сервера.

Скористаємося вже готовим класом CurrencyExchange. Напишемо сам сервер:

rest.php:

```
<?php
require_once 'config.php'; require_once 'Zend/Rest/Server.php';

Doctrine::loadModels('models');

$server = new Zend_Rest_Server();
$server->setClass( 'CurrencyExchange');
$server->handle();
?>
```

Як бачите все дуже схоже і просто.

Однак, слід домовитися про те, що наш REST-сервіс менш захищений, ніж SOAP-сервіс, так як будь-який доданий метод в клас CurrencyExchange при його виклику відпрацює (сам клас визначає структуру сервісу).

Перевіримо роботу нашого сервісу. Для цього достатньо передати параметри виклику методу в терміні GET-запиту:

```
?method=getExchange&code=840&date=2008-11-29
```

чи

```
?method=getExchange&arg1=840&arg2=2008-11-29
```

При бажанні або необхідності ви можете самостійно задавати структуру ваших XML відповідей для сервісу REST. В цьому випадку, також буде необхідно подбати і про

створення визначення типу вашого XML документа (DTD - Document Type Definition). Це буде мінімальним описом API вашого сервісу.

Найпростіший тестовий клієнт до REST сервісу може бути в нашому випадку таким:

```
<?php
$client = new Zend_Rest_Client( 'http://mikhailstadnik.com/ctws/rest.php');
$result = $client->getExchange( 840, date( 'Y-m-d'))->get(); if ($result-
>isSuccess()) {
echo 'USD exchange: ' . $result->response;
}
?>
```

В принципі, Zend_Rest на сьогоднішній день не можна назвати найбільш точною реалізацією принципів REST. Перебільшуючи, можна говорити про те, що ця реалізація звелася до віддаленого виклику процедур (RPC), хоча філософія REST набагато ширше.

3. API та його тестування

API надає можливість використовувати чужі напрацювання в своїх цілях. Сучасні API часто приймають форму веб-сервісів, які надають користувачам (як людям, так і іншим веб-сервісам) якусь інформацію. Зазвичай процедура обміну інформацією і формат передачі даних структуровані, щоб обидві сторони знали, як взаємодіяти між собою.

API(Application Programming Interface) - набір готових класів, процедур, функцій, структур та констант, наданих додатком (бібліотекою, сервісом) для використання в зовнішніх програмних продуктах.

За посиланням (https://github.com/hhru/api) ви зможете прочитати детальний опис взаємодії клієнтів та сервісів HeadHunter API.

Формати передачі даних. Існує багато форматів даних за допомогою яких користувачі взаємодіють з API, зокрема XML та JSON.

Щоб вказати потрібну дію, яку необхідно зробити з ресурсом, в HTTP означено набір *методів запиту* ***(request methods).*** Ці методи іноді називають HTTP-*дієсловами,* незважаючи на те, що вони можуть бути іменниками. Кожен з них реалізує іншу семантику, але вони мають деякі спільні риси, за якими їх поділяють на групи: наприклад

методи запиту можуть бути safe, idempotent, або cacheable.

GET

Метод GET запитує представлення вказаного ресурсу. Запити, які використовують GET, повинні лише отримувати дані.

HEAD

Метод HEAD запитує відповідь, ідентичну запиту GET, але без тіла.

POST

Метод POST використовується для відправки об'єкта на вказаний ресурс, часто викликаючи зміну стану або побічних ефектів на сервері.

PUT

Метод PUT замінює всі поточні представлення цільового ресурсу на корисне навантаження, що вказане в запиті.

DELETE

Метод DELETE видаляє вказаний ресурс.

CONNECT

Метод CONNECT встановлює тунель до сервера, ідентифікованого цільовим ресурсом.

OPTIONS

Метод OPTIONS використовується для опису варіантів зв'язку до цільового ресурсу.

TRACE

Метод TRACE виконує тест зворотного зв'язку по шляху до цільового ресурсу.

PATCH

Метод PATCH використовується для застосування часткових модифікацій в ресурсі.

HTTP коди відповідей

Сервер може надсилати різні коди у відповідь на запис користувачів. Це можуть бути коди помилок чи просто коди, які інформують користувачів про стан сервера. (Див. тему «Особливості тестування веб-додатків»).

REST API – це ідеологія побудови API, розшифровується як Representation State Transfer API. Вона основується на наступних принципах, сформульованих її творцем

Роєм Філдінгом:

1. Клієнт-серверна архітектура.
2. Stateless сервер.
3. Кешованість.
4. Багатошарова структура.
5. Єдиний інтерфейс.
6. Код за вимогою.

Детальніше з вимогами до API можна ознайомитись за посиланням (https://secl.com.ua/article_rest_api_web.html).

Аутентифікація

Зазвичай для використання API потрібен спеціальний ключ, за допомогою якого сервер пізнає користувача. У відкритих API ключ може бути відсутнім або надаватися за запитом (наприклад, після реєстрації на сайті).

Для інтеграції сервісів один з одним широко використовується протокол аутентифікації OAuth (докладніше про нього можна почитати в статті http://m.geektimes.ru/post/77648/).

Інструменти для роботи з API

Звичайні GET запити можна надсилати за допомогою браузера. Але існує безліч спеціальних інструментів, які призначені для розробки і тестування API. Вони надають можливість не тільки відправляти різні типи запитів, але і зберігати запити, показувати результати в різних форматах, виступати в ролі proxy сервера. І багато багато іншого.

Серед таких інструментів:

Postman. Розширення для Google Chrome, яке в безкоштовній версії дозволяє посилати запити, записувати їх, показувати історію. Зручно і зрозуміло.

jMeter. Інструмент, який здобув популярність перш за все завдяки навантажувального тестування, яке можна проводити з його допомогою. Але це лише одна з безлічі його застосувань.

Fiddler. Дозволяє переглядати надіслані HTTP запити. І ще багато чого.

SoapUI. Потужний продукт для розробки і тестування веб-додатків.

Advanced REST Client. Ще одне розширення для Chrome для роботи з API (конструкція запитів, їх показ в зручному вигляді і інше).

Тестування API

А тепер - дуже коротко про те, як тестувати API.

Звичайно, тут є своя специфіка, але ми можемо використовувати такі загальноприйняті техніки, як:

Аналіз граничних значень. В API запитах в явному вигляді можуть передаватися значення параметрів. Це відмінний привід виділити кордони вхідних і вихідних значень і перевірити їх.

Розбиття на класи еквівалентності. Навіть у невеликого API є безліч варіантів використання і безліч комбінацій вхідних і вихідних змінних. Тому ми можемо зайвий раз використовувати наші навички виділення еквівалентних класів.

Як мені здається, при тестуванні потрібно враховувати, що API створюються багато в чому для інтеграції сервісів. І працюють з ними часто не люди, а інші програмні системи. Тому потрібно оцінювати API з позиції зручності його використання іншими продуктами, з позиції легкої інтеграції з ним. Поважаючий себе API повинен також мати зрозумілу і детальну документацію.

Можна зробити висновок, що всі види тестування, до яких ми звикли - функціональне тестування, навантажувальне, тестування безпеки, юзабіліті, тестування документації - не чужі при тестуванні API. В принципі, це не дивно, тому що API є повноцінним самостійним продуктом.

4. Що таке JSON?

JSON або JavaScript Object Notation - це формат, який реалізує неструктуроване текстове представлення структурованих даних, засноване на принципі пар ключ-значення і упорядкованих списках. Хоча JSON почав своє поширення з JavaScript, він підтримується в більшості мов, або спочатку, або за допомогою спеціальних бібліотек. Зазвичай Json використовується для обміну інформацією між веб-клієнтами і веб-сервером.

За останні 15 років JSON став формальним стандартом обміну даними і використовується практично скрізь в інтернеті. Сьогодні він використовується практично всіма веб-серверами. До такої популярності привело ще й те, що багато баз даних підтримували JSON. Сучасні реляційні бази даних, такі як PostgreSQL і MySQL

тепер підтримують зберігання і експорт даних в JSON. Бази даних, такі як MongoDB і Neo4j також підтримують JSON, хоча MongoDB використовує злегка модифіковану версію JSON. У цій статті ми розглянемо що таке JSON, його переваги над XML, його недоліки, а також коли його краще використовувати.

Щоб зрозуміти навіщо потрібен формат JSON і як він працює не обійтися без практики. Спочатку давайте розглянемо такий приклад:

```
{
“firstName”:
“Jonathan”,
“lastName”:
“Freeman”,
“loginCount”: 4,
“isWriter”: true,
“worksWith”: [“Spantree Technology Group”,
“InfoWorld”], “pets”: [
{
“name”: “Lilly”,
“type”: “Raccoon”
}  ]
}
```

У цій структурі ми чітко визначили деякі атрибути людини. Спочатку ми визначили ім'я, прізвище, кількість авторизацій в системі, чи є ця людина письменником, список компаній, з якими він працює і список домашніх тварин. Така або схожа структура може бути передана з сервера в веб-браузер або мобільний додаток, яке вже може робити все що потрібно з цими даними, наприклад, відобразити їх або зберегти.

JSON - це загальний формат даних з мінімальною кількістю типів значень - рядки, числа, булеві значення (одиниця або нуль), списки, об'єкти і нуль. Незважаючи на те, що JSON є підмножиною JavaScript, такі типи даних є в більшості популярних мов програмування, що робить JSON хорошим кандидатом для передачі даних між програмами, написаними на різних мовах.

Чому варто використовувати JSON?

Щоб зрозуміти корисність і важливість JSON нам потрібно трохи розібратися в історії інтерактивності в інтернет. На початку 2000 років інтерактивність роботи веб-сайтів почала змінюватися. У той час браузер служив тільки для відображення

інформації, а всю роботу по підготовці контенту до відображення виконував веб-сервер. Коли користувач натискав кнопку в браузері, запит вирушав на сервер, де збиралася і відправлялася сторінка HTML, готова для відображення. Такий механізм був повільним і неефективним. Це вимагало, щоб браузер повторно перемальовував все на сторінці, навіть, якщо змінилася невелика частина даних.

У той час передача тарифікація виконувалася за кількість переданих даних, тому розробники розуміли, що перезавантаження цілої сторінки обходиться дуже дорого і розглядали нові технології для поліпшення користувальницького інтерфейсу. Тоді можливість створення веб-запитів в фоновому режимі, яка була додана в Internet Explorer 5 виявилася досить життєздатним підходом до поетапного завантаження даних для відображення. Замість перезавантаження сторінки, натискання на кнопку просто виконає веб-запит, який буде працювати у фоновому режимі. Вміст буде оновлено, як тільки завантажиться. Ним можна управляти за допомогою JavaScript, універсальної мови програмування для браузерів.

Спочатку дані передавалися в форматі XML, але він був складним для використання в JavaScript. В JavaScript вже були об'єкти, які використовувалися для представлення даних в мові, тому Дуглас Крокфорд взяв синтаксис об'єктів JS і використав його в якості специфікації нового формату обміну даними, який називався JSON. Цей формат було набагато простіше читати і розбирати в браузері на JavaScript. Незабаром розробники почали використовувати JSON замість XML.

Зараз швидкий обмін даними JSON є стандартом де-факто для передачі даних між сервером і клієнтом, мобільними додатками і навіть внутрішніми системними службами.

JSON проти XML

Як говорилось вище, основною альтернативою JSON був і є XML. Однак XML стає все менш поширеним в нових системах. І дуже легко зрозуміти чому. Нижче наведено приклад запису даних, які ви бачили вище в Json через XML:

```
<?xml version="1.0"?>
<person>
<first_name>Jonathan</first_name>
<last_name>Freeman</last_name>
<login_count>4</login_count>
<is_writer>true</is_writer>
<works_with_entities>
```

```
<works_with>Spantree Technology Group</works_with>
<works_with>InfoWorld</works_with>
</works_with_entities>
<pets>
<pet>
<name>Lilly</name>
<type>Raccoon</type>
</pet>
</pets>
</person>
```

На додаток до надмірності коду, по суті запис даних зайняв в два рази більше місця, XML ще вводить деяку двозначність при аналізі структури даних. Перетворення XML в об'єкт JavaScript може зайняти від десятків до сотень рядків коду і вимагає тонкої настройки для кожного аналізованого об'єкта.

Перетворення JSON в об'єкт JavaScript виконується в один рядок і не потребує будь-яких попередніх знань про аналізований об'єкт.

Обмеження JSON

Хоча JSON відносно стислий і гнучкий формат даних, з яким легко працювати на багатьох мовах програмування, у нього є деякі недоліки. Ось деякі обмеження:

1. **Відсутність структури**. З однієї сторони, це означає, що є повна гнучкість для представлення даних будь-яким чином. З іншої, ви можете легко зберігати неструктуровані дані.
2. **Тільки один тип чисел**. Підтримується формат з плаваючою комою і подвійною точністю IEEE-754. Це досить багато, але ви не можете використовувати ту різноманітність числових типів, що є в інших мовах.
3. **Відсутність типу дати**. Розробники повинні використовувати рядкові представлення дат, що може викликати невідповідність форматування. Або ж використовувати в якості дати кількість мілісекунд, що пройшли з початку епохи Unix (1 січня 1970).
4. **Відсутність коментарів**. Ви не зможете робити нотації для полів, які вимагають цього прямо в коді.
5. **Детальність**. Хоча JSON менш детальний, ніж XML, це не самий стислий формат обміну даними. Для високопродуктивних або спеціалізованих служб ви захочете використовувати більш ефективні формати.

Коли слід використовувати JSON?

Якщо ви розробляєте програмне забезпечення, яке взаємодіє з браузером або мобільним додатком, вам краще використовувати JSON. Використання XML вважається застарілим. Для зв'язку між серверами JSON може бути не дуже ефективним і краще використовувати інфраструктуру серіалізації, схожу на Apache Avro або Apache Thrift. Навіть тут JSON - це не поганий вибір і він може давати все, що вам потрібно. Але немає точної відповіді, що вибрати.

Якщо ви використовуєте бази даних MySQL, ваша програма буде в великій мірі залежати від того, що робиться в базі даних. У реляційних базах даних, які підтримують JSON вважається хорошим тоном використовувати його якомога менше. Реляційні бази даних були розроблені для даних певної схеми. Хоча більшість з них зараз підтримує формат даних JSON, продуктивність роботи з ним буде значно нижче.