Kpi-best

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут ім. І. Сікорського»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ТЕХНІЧНОЇ КІБЕРНЕТИКИ

Реферат

по курсу «Архітектура корпоративних систем»

***«Web-сервіси в корпоративних комп'ютерних мережах на основі технологій SOA»***

Виконав: студент групи ІК-61м

Клейменов Р. С.

Варіант: №9

Київ – 2017

ЗМІСТ

[1. Сервісно-орієнтована архітектура 3](#_Toc478211694)

[1.1. Визначення 3](#_Toc478211695)

[1.2. Основні поняття 4](#_Toc478211696)

[1.3. Переваги використання SOA 6](#_Toc478211697)

[2. Веб-сервіси(служби) 7](#_Toc478211698)

[2.1. Визначення 7](#_Toc478211699)

[2.2. Архітектура 8](#_Toc478211700)

[2.3. Методи розробки 9](#_Toc478211701)

[2.4. Переваги 9](#_Toc478211702)

[2.5. Недоліки 10](#_Toc478211703)

[3. Загальні поняття бізнес-процесів в SOAP 10](#_Toc478211704)

[3.1. Що таке SOAP 10](#_Toc478211705)

[3.2. Загальні бізнес-процеси 10](#_Toc478211706)

[3.3. Загальна структура повідомлення 12](#_Toc478211707)

[4. Загальні поняття бізнес-процесів в REST 13](#_Toc478211708)

[4.1. Що таке *REST*? 13](#_Toc478211709)

[4.2. Архітектурні обмеження REST бізнес процесів 14](#_Toc478211710)

[5. Структура REST бізнес-процесу 17](#_Toc478211711)

[Висновок 18](#_Toc478211712)

[ВИКОРИСТАНА ЛІТЕРАТУРА 20](#_Toc478211713)

# Сервісно-орієнтована архітектура

## Визначення

**Сервісно-орієнтована архітектура**  — архітектурний шаблон програмного забезпечення, модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабко пов'язаних замінних компонентів, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколам.

Програмні комплекси, розроблені відповідно до сервіс-орієнтованої архітектури, зазвичай реалізуються як набір веб-служб, взаємодіючих по протоколу SOAP, але існують і інші реалізації (наприклад, на базі jini, CORBA, на основі REST).

Інтерфейси компонентів в сервіс-орієнтованій архітектурі інкапсулюють деталі реалізації (операційну систему, платформу, мову програмування) від інших компонентів, таким чином забезпечуючи комбінування і багаторазове використання компонентів для побудови складних розподілених програмних комплексів, забезпечуючи незалежність від використовуваних платформ та інструментів розробки, сприяючи масштабованості і керованості створюваних систем.

За словами Клайва Фінкельштейна (Clive Finkelstein), автора інформаційної інженерії (information engineering), до появи концепції SOA при розробці систем відправним моментом для програмування бізнес-логіки слугували діаграми робочих потоків і блок-схеми систем. Розроблені вручну програми ретельно тестувалися, після чого впроваджувалися. Сьогодні ситуація змінилася докорінно: сучасні інструменти управління бізнес-процесами дозволяють обійтися без ручної розробки та тестування. Так, за допомогою методів моделювання можна перевіряти коректність виконання бізнес-логіки, представленої в діаграмах, а потім автоматично отримувати описи цих діаграм на XML-мовах управління бізнес-процесами.

На думку Клайва Фінкельштейна, така технологія управління бізнес-процесами є великим кроком вперед з точки зору підвищення ефективності розробки систем; за значимістю її можна порівняти із створенням наприкінці 50-х років компіляторів мови високого рівня. Дійсно, даний підхід дозволяє спростити виклик Web-сервісів з будь-якого місця розташування і їх виконання на основі бізнес-правил. Крім того, при зміні цих правил, коригується відповідна логіка в діаграмах: діаграми автоматично генеруються заново. Таким чином, закладаються передумови для переходу від повільного ручного кодування, використовуваного зараз при створенні систем, до автоматизованого. Завдяки цьому компанії зможуть реалізовувати зміни бізнес-правил за хвилини або години, а не за місяць або рік.

## Основні поняття

Дуже часто становлення того чи іншого підходу супроводжується появою невірних або хибних трактувань, як це було, наприклад, з концепцією федеративного сховища даних. Не оминуло стороною це і сервіс-орієнтовану архітектуру. Так вважає представник компанії BEA Джерімі Уестерман (Jeremy Westerman). Саме тому в одній із своїх статей, присвячених SOA, він спеціально зупиняється на «проблемних місцях» і наводить докладні пояснення:

* SOA не є чимось новим: IT-відділи компаній успішно створювали і розгортали застосунки, що підтримують сервіс-орієнтовану архітектуру, вже багато років — задовго до появи XML і Web-сервісів.
* SOA — це не технологія, а спосіб проектування і організації інформаційної архітектури та бізнес-функціональності.
* Купівля найновіших продуктів, що реалізують XML і Web-сервіси, не означає побудову застосунків згідно з принципами SOA.

Джерімі Уестерман дає наступне визначення SOA: це парадигма, призначена для проектування, розробки та управління дискретних одиниць логіки (сервісів) в обчислювальному середовищі. Застосування цього підходу вимагає від розробників проектування застосунків як набору сервісів, навіть якщо переваги такого рішення відразу неочевидні. Розробники повинні вийти за межі своїх застосунків і подумати, як скористатися вже наявними сервісами, або вивчити, як їх сервіси можуть бути використані їх колегами.

SOA підштовхує до використання альтернативних технологій і підходів (таких як обмін повідомленнями) для побудови застосунків за допомогою зв'язування сервісів, а не за допомогою написання нового програмного коду. У цьому випадку, при належному проектуванні, застосування повідомлень дозволяє компаніям своєчасно реагувати на зміну ринкових умов — настроювати процес обміну повідомленнями, а не розробляти нові програми.

Ще до недавніх пір термін «сервіс-орієнтована архітектура» був синонімом «Web-сервіс». SOA — виклик Web-сервісів за допомогою засобів і мов управління бізнес-процесами. SOA — це термін, який з'явився для опису виконуваних компонентів — таких як Web-сервіси — які можуть викликатися іншими програмами, які виступають клієнтами або споживачами цих сервісів. Ці сервіси можуть бути повністю сучасними — або навіть застарілими — прикладними програмами, які можна активізувати як чорний ящик. Від розробника не потрібно знати, як працює програма, необхідно лише розуміти, які вхідні та вихідні дані потрібні, і як викликати ці програми для виконання. У найзагальнішому вигляді SOA припускає наявність трьох основних учасників: постачальника сервісу, споживача сервісу та реєстру сервісів. Взаємодія учасників виглядає досить просто: постачальник сервісу реєструє свої сервіси в реєстрі, а споживач звертається до реєстру із запитом.

Для використання сервісу необхідно дотримуватися угоди про інтерфейс для звернення до сервісу — інтерфейс повинен не залежати від платформи. SOA реалізує масштабованість сервісів — можливість додавання сервісів, а також їх модернізацію. Постачальник сервісу і його споживач виявляються непов'язаними — вони спілкуються за допомогою повідомлень. Оскільки інтерфейс повинен не залежати від платформи, то і технологія, використовувана для визначення повідомлень, також повинна не залежати від платформи. Тому, як правило, повідомлення є XML-документами, які відповідають XML-схемі.

Дійсно, відкриті стандарти, що описують XML і Web-сервіси, дозволяють застосовувати SOA до всіх технологій і застосунків, встановлених в компанії. Як відомо, Web-сервіси базуються на широко поширених і відкритих протоколах: HTTP, XML, UDDI, WSDL і SOAP. Саме ці стандарти реалізують основні вимоги SOA — по-перше, сервіс повинен піддаватися динамічному виявленню і викликом (UDDI, WSDL і SOAP), по-друге, повинен використовуватися незалежний від платформи інтерфейс (XML). Нарешті, HTTP забезпечує функціональну сумісність.

## Переваги використання SOA

**Стратегічна цінність SOA**

* Скорочення часу реалізації проектів, або «часу виходу на ринок».
* Підвищення продуктивності.
* Швидша і дешевша інтеграція застосунків і інтеграція B2B.

**Тактичні переваги SOA**

* Простіша розробка і впровадження застосунків.
* Використання поточних інвестицій.
* Зменшення ризику, пов'язаного з впровадженням проектів в області автоматизації послуг і процесів.
* Можливість безперервного поліпшення наданої послуги.
* Скорочення числа звернень за технічною підтримкою.
* Підвищення показника повернення інвестицій (ROI).
* Перспективи

# Веб-сервіси(служби)

## Визначення

Веб-служба, веб-сервіс ­– програмна система що ідентифікується веб-адресою зі стандартизованими інтерфейсами.

Веб-служби можуть взаємодіяти один з одним і зі сторонніми додатками за допомогою повідомлень, заснованих на певних протоколах (SOAP, XML-RPC, REST і т.д.). Веб-служба є одиницею модульності при використанні сервіс-орієнтованої архітектури додатку.

Термін "веб-служба" організацією W3C застосовується до багатьох різних систем, але в основному термін стосується клієнтів та серверів, що взаємодіють за допомогою повідомлень протоколу SOAP. В обох випадках припускається що існує також опис доступних операцій у форматі WSDL. Хоча наявність цього опису не є вимогою SOAP, а радше передумовою для автоматичного генерування коду на платформах Java та .NET на стороні клієнта.

## Архітектура

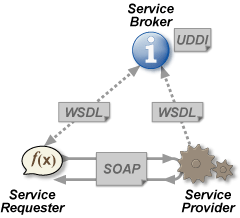


Рис. 1. Інстанції веб-служб

Як показано на малюнку, можна виділити три інстанції, взаємодіючі в рамках веб-служби. Перекладемо їх назви як:

* замовник (service requester);
* виконавець (service provider);
* каталог (service broker).

Коли служба розроблена, виконавець реєструє її в каталозі, де її можуть знайти потенційні замовники. Замовник, знайшовши в каталозі відповідну службу, імпортує звідти її WSDL-специфікацію і розробляє відповідно до неї своє програмне забезпечення. WSDL описує формат запитів і відповідей, якими обмінюються замовник і виконавець в процесі роботи. Для забезпечення взаємодії використовуються такі нормативні документи:

* XML: Розширювана мова розмітки, призначена для зберігання і передачі структурованих даних;
* SOAP: Протокол обміну повідомленнями на базі XML;
* WSDL: Мова опису зовнішніх інтерфейсів веб-служби на базі XML;
* UDDI: Універсальний інтерфейс розпізнавання опису та інтеграції (Universal Discovery, Description, and Integration). Каталог веб-служб і даних про компанії, що надаються веб-служби для загального користування або конкретним компаніям.

## Методи розробки

Існують засоби автоматизації розробки веб-служб, які поділяються на дві основні групи. При розробці знизу-вгору спочатку пишуться класи, а з їх вихідного тексту генеруються WSDL-файли, що документують службу. Недоліком цього методу є схильність Java-класів частих змін. При підході зверху-вниз спочатку готується WSDL, а з нього генерується скелет Java-класу, що імплементує службу. Поки формат повідомлень, якими обмінюються замовник і виконавець, не змінюється, зміни в кожному з них не порушують взаємодії. Ця техніка називається іноді «contract first», так як вихідною точкою є WSDL ( «договір» між замовником і виконавцем).

## Переваги

* Веб-служби забезпечують взаємодію програмних систем незалежно від платформи
* Веб-служби базуються на відкритих стандартах та протоколах. Завдяки використанню XML досягається простота розробки та відлагодження веб-служб
* Використання інтернет-протокола HTTP забезпечує взаємодію програмних систем через міжмережевий екран.

## Недоліки

* Більш низька продуктивність у порівнянні з технологіями CORBA, DCOM за рахунок використання текстових XML повідомлень.

# Загальні поняття бізнес-процесів в SOAP

## Що таке SOAP

SOAP (англ. Simple Object Access Protocol) — протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі XML.

Спочатку SOAP призначався, в основному, для реалізації віддаленого виклику процедур (RPC), а назва була абревіатурою: Simple Object Access Protocol — простий протокол доступу до об'єктів. Зараз протокол використовується для обміну повідомленнями в форматі XML, а не тільки для виклику процедур. SOAP є розширенням мови XML-RPC.

SOAP може використовуватись з будь-яким протоколом прикладного рівня: SMTP, FTP, HTTP та інші. Проте, його взаємодія з кожним із цих протоколів має свої особливості, які потрібно відзначити окремо. Найчастіше SOAP використовується разом з HTTP.

SOAP є одним із стандартів, на яких ґрунтується технологія веб-сервісів.

## Загальні бізнес-процеси

Протокол SOAP не розрізняє виклик процедури і відповідь на нього, а просто визначає формат послання (message) у вигляді документа XML. Послання може містити виклик процедури, відповідь на нього, запит на виконання якихось інших дій або просто текст.

Специфікацію SOAP не цікавить вміст послання, вона задає тільки його оформлення.

SOAP - це базова однонаправлена модель з'єднання, що забезпечує узгоджену передачу повідомлення від відправника до одержувача, потенційно допускає наявність посередників, які можуть обробляти частину повідомлення або додавати до нього додаткові елементи. Специфікація SOAP містить угоди по перетворенню однонаправленного обміну повідомленнями відповідно до принципу «Запит / відповідь», а також визначає як здійснювати передачу всього XML-документа.

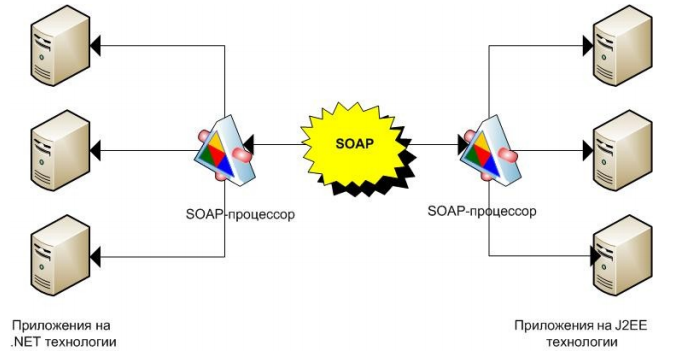


Рис. 2. Архітектура додатків з використанням SOAP

Як видно з малюнку SOAP призначений для підтримки незалежного абстрактного протоколу зв'язку, що забезпечує комунікацію двох і більше додатків, сайтів, підприємств і т.п., реалізованих на різних технологіях і апаратних засобів.

## Загальна структура повідомлення

SOAP-повідомленням є XML-документ; повідомлення складається з трьох основних елементів: конверт (SOAP Envelope), заголовок (SOAP Header) і тіло (SOAP Body).

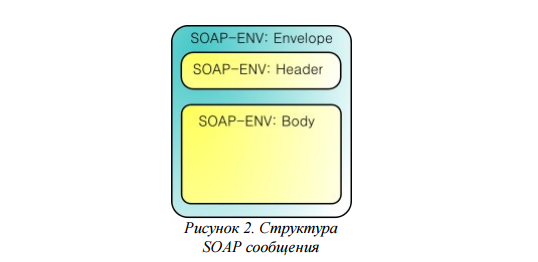


Рис. 3. SOAP message

**Конверт (SOAP Envelope)**

Є самим «верхнім» елементом SOAP повідомлення. Містить кореневої елемент XML-документа. Описується за допомогою елемента Envelope з обов'язковим простором імен http://www.w3.org/2003/05/soap-envelope для версії 1.2 і http://schemas.xmlsoap.org/soap/ для версії 1.1.

**Заголовок SOAP (SOAP Header)**

Перший прямий дочірній елемент конверта. Чи не обов'язковий.

**Тіло SOAP (SOAP Body)**

Елемент Body обов'язково записується відразу за елементом Header, якщо він є в повідомленні, або першим в SOAP-повідомленні, якщо заголовок відсутній. В елемент Body можна вкласти довільні елементи, специфікація ніяк не визначає їх структуру. Визначено тільки один стандартний елемент, який може бути в тексті листа - Fault, що містить повідомлення про помилку.

# Загальні поняття бізнес-процесів в REST

## Що таке *REST*?

**REST** (скор. англ. Representational State Transfer, «передача репрезентативного стану») — підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). REST протокол (як і HTTP) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформацію про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабовність системи і дозволяє їй еволюціонувати з новими вимогами.

Антиподом REST є підхід, заснований на виклику віддалених процедур (Remote Procedure Call, RPC). Підхід RPC дозволяє використовувати невелику кількість мережевих ресурсів з великою кількістю методів і складним протоколом. При підході REST кількість методів і складність протоколу суворо обмежені, що призводить до того, що кількість окремих ресурсів має бути великою.

REST — це архітектурний стиль для розподілених гіпертекстових систем.

## Архітектурні обмеження REST бізнес процесів

REST, як і кожен архітектурний стиль відповідає ряду архітектурних обмежень. Це гібридний стиль який успадковує обмеження з інших архітектурних стилів.

Перша архітектура від якої він успадковує обмеження — це **клієнт-серверна архітектура**. Її обмеження вимагає розділення відповідальності між компонентами які займаються зберіганням та оновленням даних (сервером) і тими компонентами які займаються відображенням даних на інтерфейсі користувача та реагування на дії з цим інтерфейсом (клієнтом). Таке розділення дозволяє компонентам еволюціонувати незалежно.

Наступним обмеженням є те що взаємодії між сервером та клієнтом **не мають стану**, тобто кожен запит містить всю необхідну інформацію для його обробки, і не покладається на те що сервер знає щось з попереднього запиту. Таким чином, наприклад дані про стан сесії (користувача який аутентифікувався) зберігаються на клієнті, і передаються з кожним запитом. Це покращує масштабовність, бо сервер після закінчення обробки запиту може звільнити всі ресурси задіяні для цієї операції, без жодного ризику втратити цінну операцію. Також спрощується моніторинг і зневадження, бо для того аби розібратись що відбувається в певному запиті, досить подивитись лише на той один запит. Збільшується надійність, бо помилка в одному запиті не зачіпає інші.

Мінусом цього обмеження є те, що знижуться продуктивність, через те що в кожен запит тепер доводиться додавати дані сесії з клієнта. Також збереження стану на різних клієнтах важче підтримувати, бо реалізації клієнтів можуть відрізнятись, в той час як середовище сервера повністю під контролем розробника.

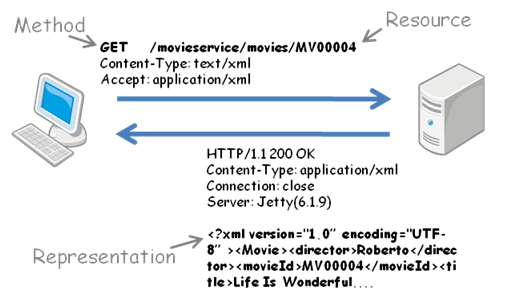


Рис. 4. Приклад REST запиту

Додатковим обмеженням стилю REST є те що системи написані в цьому стилі **повинні підтримувати кешування**, тобто дані які передаються між сервером повинні містити інформацію про те чи можна їх кешувати, і якщо можна, то як довго. Це дозволяє збільшувати продуктивність уникаючи зайвих запитів, але також зменшує надійність системи, через те що дані в кеші можуть бути застарілі.

Рання архітектура веб, створена Тімом Бернерсом-Лі відповідала цим трьом обмеженням — клієнт-сервер без стану з підтримкою кешування. Проте стиль REST додає ще додаткові обмеження.

Всі компоненти в архітектурі REST підтримують **однорідний інтерфейс**. Це зменшує зв'язність між компонентами і сервісами які вони надають і дозволяє нескладно змінювати компоненти при потребі. Це досягається кількома точнішими обмеженнями:

* ідентифікація ресурсів
* маніпуляція ресурсами через представлення
* самоописові повідомлення
* гіпермедіа як рушій стану застосунку

Наступним обмеженням для REST є **поділ на шари абстрації**. Кожен компонент потрапляє в якийсь шар, і спілкується лише з компонентами в шарі під ним або в шарі над ним. Обмежнення знання системи одним шаром зменшує складність компонентів.

Останнім архітектурним обмеженням в REST є те що клієнти повинні дозволяти розширювати свою функціональність дозволяючи **завантаження додаткового коду** в формі аплетів чи скриптів. Це спрощує клієнти, дозволяючи не реалізовувати всі необхідні функції попередньо. Щоправда це необов'язкове обмеження, і якщо воно не дає переваг для конкретного застосування, то його не обов'язково реалізовувати. Наприклад, дозвіл завантаження стороннього коду може бути не бажаним з точки зору безпеки.

# Структура REST бізнес-процесу

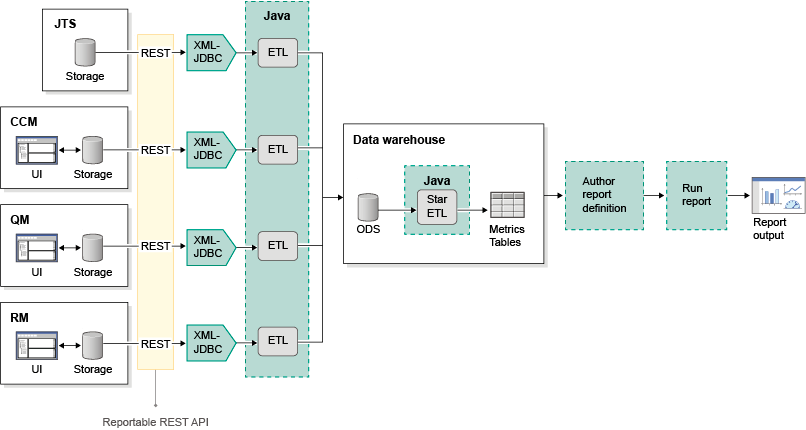


Рис. 5. Бізнес-процес взаємодії різноманітних додатків з warehouse системою шляхом REST запитів.

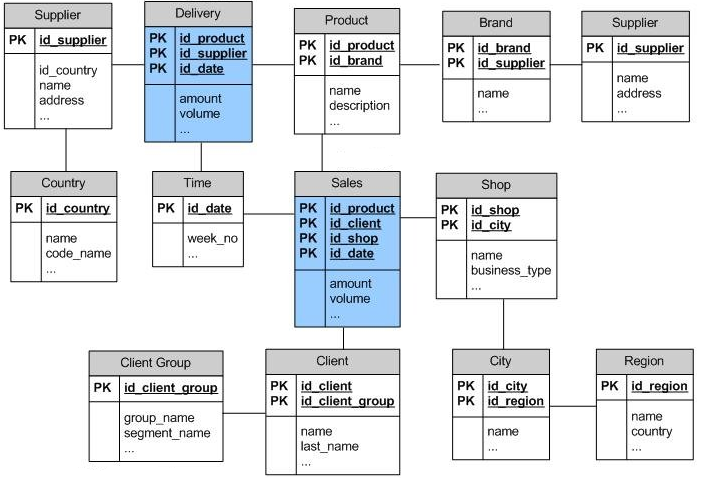


Рис. 6. Схема бази-даних warehouse системи

# Висновок

***Порівняння REST та SOAP***

* 1. SOAP - це ціле сімейство протоколів і стандартів, звідки безпосередньо випливає, що це більш великоваговий і складний варіант з точки зору машинної обробки. Тому REST працює швидше;
  2. SOAP використовують HTTP як транспортний протокол, в той час як REST базується на ньому. Це означає, що всі існуючі напрацювання на базі протоколу HTTP, такі як кешування на рівні сервера, масштабування, продовжують так само працювати в REST архітектурі, а для SOAP необхідно шукати інші засоби. Замість цього SOAP сервіси отримують такі міфічні властивості, як можливість працювати з будь-яким протоколом транспортного рівня замість HTTP, однак практичної користі від нього часто не багато;
  3. У першому результаті за запитом «REST vs SOAP» акцентується увага на тому, що відповідь REST може бути представлена в різних форматах, а SOAP прив'язаний до XML. Це дійсно важливий фактор, досить уявити собі виклик сервісу з javascript, відповідь на який ми безумовно хочемо отримувати в JSON;
  4. «REST vs SOAP» можна перефразувати в «Простота vs Стандарти», що проявляється в тому, що для SOAP ми маємо протокол WSDL для вичерпного опису веб-сервісу, який з використанням все тих же чудових засобів розробки просто-таки чарівним чином робить майже всю роботу за нас. З боку REST ми маємо загадковий і невикористаний протокол WADL, який, в принципі, і не потрібен - він заважає простоті.
  5. Другий аспект попереднього пункту - обробка помилок. У SOAP вона повністю стандартизована, а REST може використовувати давно відомі коди помилок HTTР
  6. SOAP працює з операціями, а REST - з ресурсами. Ций факт в сукупності з відсутністю клієнтського стану у RESTful сервісів приводить нас до того, що такі речі як транзакції або інша складна логіка повинна реалізовуватися «SOAP-но».

# ВИКОРИСТАНА ЛІТЕРАТУРА

1. https://habrahabr.ru/post/131343/
2. https://en.wikipedia.org/wiki/Web\_service
3. https://en.wikipedia.org/wiki/Service-oriented\_architecture
4. https://en.wikipedia.org/wiki/Web\_Services\_Description\_Language
5. https://en.wikipedia.org/wiki/SOAP
6. https://en.wikipedia.org/wiki/JSON
7. https://en.wikipedia.org/wiki/REST
8. https://www.w3.org/TR/soap/
9. https://bocoup.com/blog/documenting-your-api