

РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев

Ас. към ПУ „Паисий Хилендарски“

@rkyurkchiev

RESTFUL WEB SERVICES

Какво е REST?

- Representational State Transfer (REST) е архитектурен модел за изграждане на разпределени системи. Уеб е пример за такава система.

REST технологии

- REST не е обвързан с конкретен набор от технологии.

Най-често използваните технологии включват:

- *URI*
- *HTTP глаголи*
- *XML и JSON*

Ограниченията на архитектурния стил REST засягат следните архитектурни свойства:

- Производителност при взаимодействие на компоненти, включително ефективността на мрежата.
- Машабируемост, позволяваща поддръжката на голям брой компоненти и техните взаимодействия.
- Простота на единния интерфейс.
- Изменяемост на компонентите, за да отговарят на променящите се нужди, дори при работещо приложение.

- Видимост на комуникацията между компонентите от страна на услугите агенти.
- Преносимост на компонентите.
- Надеждност при възникване на проблеми на системно ниво в компоненти, конектори или данните.

REST ограничения

- Клиент-сървър архитектура (Client-server architecture)
 - Принципът зад ограниченията клиент-сървър е разделянето на проблемите. Разделянето на проблемите на потребителския интерфейс от проблемите за съхранение на данни подобрява преносимостта на потребителските интерфейси в множество платформи.

■ Безсъстоятелност (Statelessness)

- Комуникацията клиент-сървър е ограничена от това, че клиентският контекст не се съхранява на сървъра между заявките. Всяка заявка от всеки клиент съдържа цялата информация, необходима за обслужване на заявката, и състоянието на сесията се съхранява от клиента. Състоянието на сесията може да бъде прехвърлено от сървъра към друга услуга, чрез база данни, за да поддържа устойчиво състояние за период и да позволява верификация.

■ Възможност за кеширане (Cacheability)

- Както в *World Wide Web*, клиентите и посредниците могат да кешират отговорите. Добре управляваният кеш елиминира допълнителните заявки между клиент-сървър и подобрява производителността и мащабируемостта.

■ Код по заявка (Code on demand) – optionalno

- Сървърите могат временно да разширят или персонализират функционалността на клиента, като прехвърлят изпълним код: например, компилирани компоненти като *Java applet* или клиентски скриптове като *JavaScript*.

■ Слойна система (Layered System)

- Клиентът не може да определи дали е свързан директно към крайния сървър или към посредник. Ако прокси (*reverse proxy*) или балансър (*load balancer*) е поставен между клиента и сървъра, това няма да повлияе на комуникацията и няма да е необходимо да актуализираме клиентския или сървърния код. Посредническите сървъри могат да подобрят мащабируемостта на системата, като позволяват балансиране на натоварването и предоставяне на споделен кеш.

■ Единен интерфейс (Uniform interface)

- Това е основното ограничение за начина, по който трябва да работят REST уеб услугите. В основата е разделянето на архитектурата от компонентите и тяхното независимо развитие.

Идентификация на ресурса в заявките

Манипулиране на ресурси чрез репрезентации

Самоописателни съобщения

Какво представлява RESTful web service?

- Всяка уеб услуга, която отговаря на REST ограниченията, може да бъде наричана RESTful web service (уеб услуга).

Ако някое от ограниченията не е изпълнено, системата не може да бъде наречена RESTful.

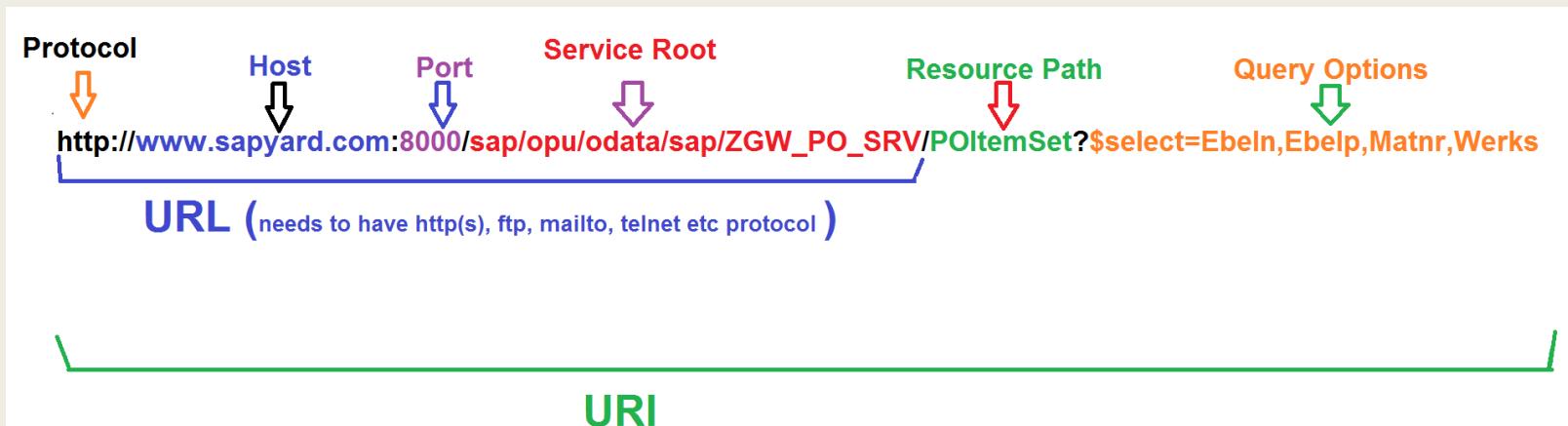
Основни елементи на RESTful имплементация:

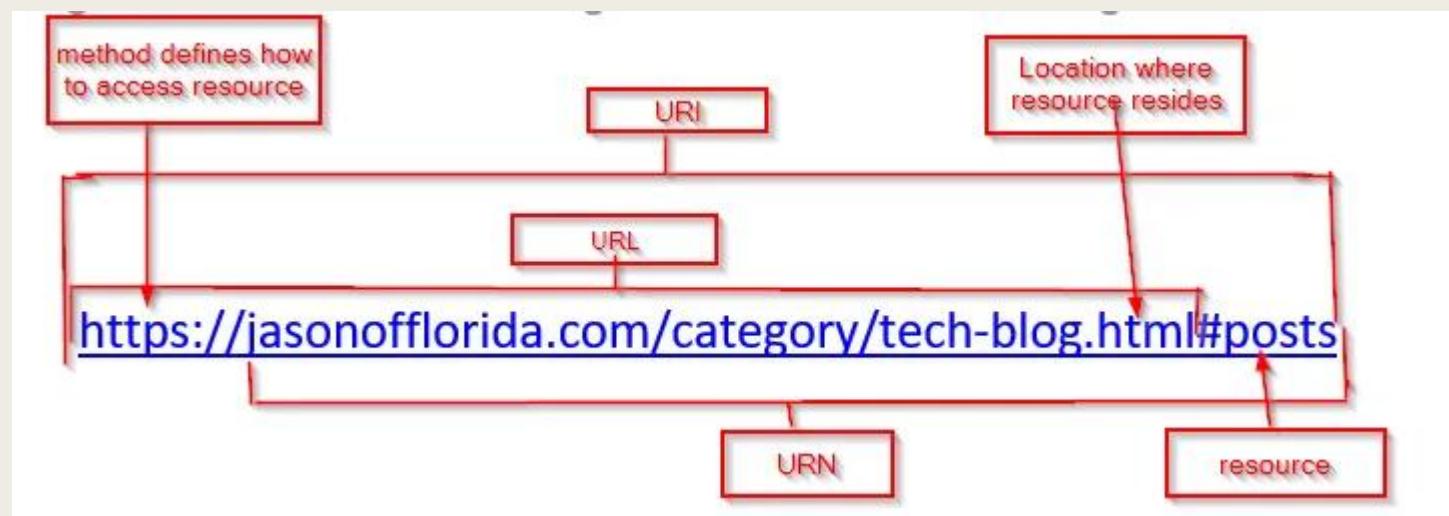
- Ресурси (Resources)
- Глаголи на заявки (Request verbs)
- Допълнителни данни на заявката (Request Headers)
- Тяло на заявката (Request Body)
- Тяло на отговора (Response Body)
- Статус на отговора (Response Status codes)

Преди да говорим за ресурси, трябва да изясним какво са URI, URL и URN.

Какво е URI и URL?

- URI е Uniform Resource Identifier - низ от знаци, който недвусмислено идентифицира определен ресурс. Най-често срещаната форма на URI е Uniform Resource Locator (URL), често наричан неофициално уеб адрес.





RESTful URLs или Clean URLs

- RESTful URLs или Clean URLs представляват едно и също нещо. Идеята е да се подобри преизползваемостта и достъпа до услугите и уеб сайтовете от неексперти. Това се постига чрез преработката на query string.

Примери за Clean URLs

Uncleaned URL	Clean URL
http://example.com/index.php?page=name	http://example.com/name
http://example.com/about.html	http://example.com/about
http://example.com/index.php?page=consulting/marketing	http://example.com/consulting/marketing
http://example.com/products?category=12&pid=25	http://example.com/products/25/categories/12p
http://example.com/cgi-bin/feed.cgi?feed=news&frm=rss	http://example.com/news.rss
http://example.com/services/index.jsp?category=legal&id=patents	http://example.com/services/legal/patents
http://example.com/kb/index.php?cat=8&id=41	http://example.com/kb/8/41
http://example.com/index.php?mod=profiles&id=193	http://example.com/profiles/193
http://en.wikipedia.org/w/index.php?title=Clean_URL	http://en.wikipedia.org/wiki/Clean_URL

Ресурси (Resources)

- Ресурсите се дефинират от URI
 - Ресурсите не могат да бъдат достъпни или манипулирани директно.
 - *RESTful* работи с ресурсни репрезентации.

Нека предположим, че уеб приложение на сървър има записи на няколко служители. Да приемем, че URL адресът на уеб приложението е <http://demo.com>. Сега, за да получите достъп до ресурс за запис на служители чрез REST, може да се издаде команда <http://demo.com/employee/1>. Тази команда казва на уеб сървъра да предостави подробности за служителя, чийто номер е 1.

Какво наричаме съществителни в RESTful?

- Съществителните са имената на ресурсите
 - При повечето дизайни, тези имена са *URI*-тата.
 - *URI* дизайнът е много важна част от REST-базирания системен дизайн.
- Всичко значимо би трявало да бъде именувано
 - Поддържайки добре създадени имена (*RESTful URLs*).

Глаголи на заявки (Request verbs)

- Операциите, които могат да бъдат извършвани върху ресурси.
- Основната идея на REST е да използва само универсални глаголи
 - Универсалните глаголи могат да бъдат приложени върху всички съществителни.

■ За повечето приложения, основните HTTP глаголи са достатъчни

- *GET*: Връща репрезентация на ресурс (трябва да няма странични ефекти).
- *PUT/PATCH*: Прехвърля репрезентация на конкретен ресурс (презаписва вече съществуваща такава).
- *POST*: Добавя репрезентация към конкретен ресурс.
- *DELETE*: Премахва репрезентация.

■ Глаголите съответстват на най-популярните операции:

- *CRUD*: *Create, Read, Update, Delete* (Създаване, Четене, Промяна, Изтриване).

Допълнителни данни на заявката (Request Headers)

- Това са допълнителни инструкции, изпращани със заявката. Те могат да определят типа на необходимия отговор или подробните за верификациите.

Тяло на заявката (Request Body)

- Представлява информацията, изпращана със заявка.
Най-често се използва в комбинация с глаголите
POST, PUT и PATCH.

Тяло на отговора (Response Body)

- Представлява информацията, получавана при отговор на заявката. Тя може да бъде предоставена в различни формати: обикновен текст, XML, HTML и JSON.

Заключение

- SOAP представлява Simple Object Access Protocol, докато REST представлява Representational State Transfer.
- SOAP е протокол, докато REST е архитектурен модел.
- SOAP използва сервисни интерфейси, за да изложи функционалността си на клиентските приложения, докато REST използва локални услуги за достъп до компонентите на хардуерното устройство.
- SOAP се нуждае от голям bandwidth за използването му, докато REST не се нуждае от голям bandwidth.
- SOAP работи само с XML формати, докато REST работи с обикновен текст, XML, HTML и JSON.
- SOAP не може да използва REST, докато REST може да използва SOAP.

ВЪПРОСИ ?

