ОНЛАЙН-ОБРАЗОВАНИЕ

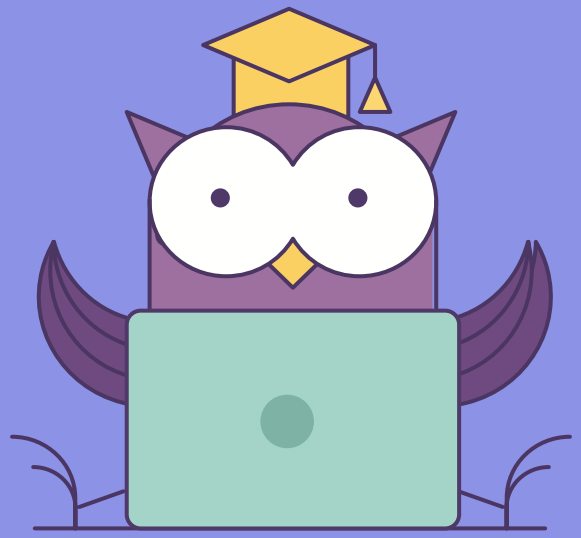
OTUS

Что такое веб-приложение и из чего состоит.

Основные подходы к построению.

Колесников Александр Александрович

Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

Содержание

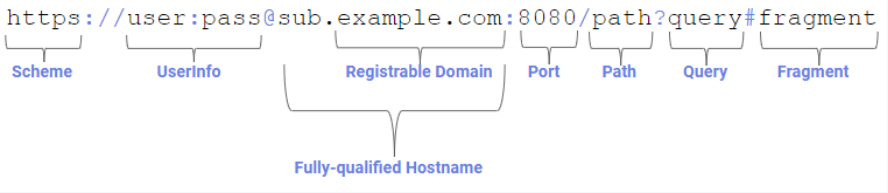
- Что такое веб приложение
  - URL
  - Cookie
  - Основные протоколы
  - Архитектура
  - Web серверы
- Механизмы защиты:
  - HSTS
  - CORS
  - SOP
  - HPKP
  - CSP

01

URL

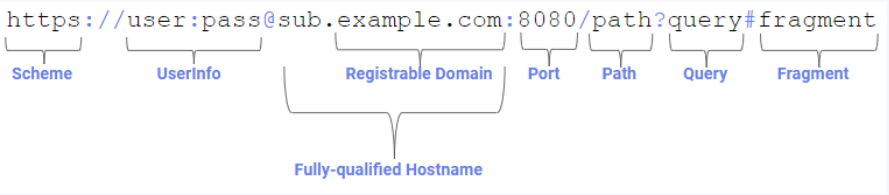
URL

- **Uniform Resource Identifier** - один из видов идентификатора ресурсов
- **Шаблон:**
  - URI = scheme:[//authority]path[?query][#fragment]
- **Особенности шаблона:**
  - Каждый фрагмент может содержать собственное допустимое значение
- Некоторые значения scheme: https://en.wikipedia.org/wiki/List_of_URI_schemes

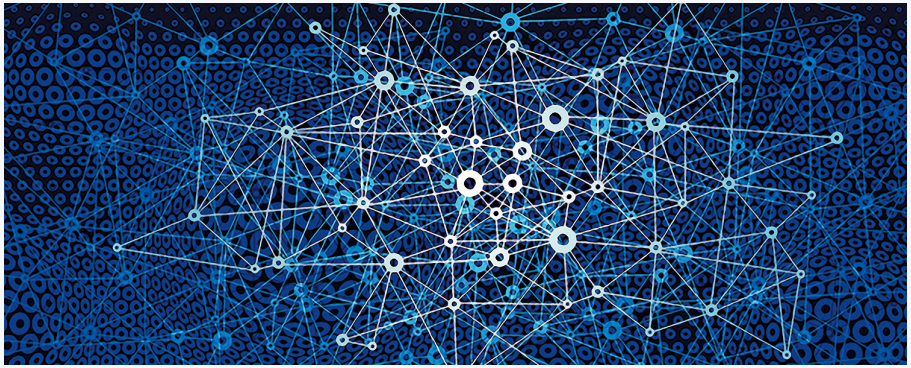


Scheme

- Важная часть URL, имеет следующие характеристики:
  - Регистро-независимая строка заканчивается двоеточием
  - Используется для управления логикой работы клиента
  - Согласно с RFC 1738, после двоеточия могут быть только цифры или буквы, а так же «+,-,.»



- Подробное описание приведено в RFC 1738. В URL обязательно должны быть «//» перед секцией authority.
- Примеры для исследования:
  - mailto:user@example.com?subject=This+message
  - javascript://example.com/0Aalert(1)
  - mailto://user@example.com
  - data:text/plain,Why,%20hello%20there!
  - http://[0:0:0:0:0:0:0:1]
  - http://127.0.0.1/
  - http://0x7f.1
  - http://017700000001/
  - http://google.com;.ya.ru



Кодировка для Web

- Важная составляющая большинства взаимодействий, которые происходят в web-приложениях.

- **Основная задача** - представление одного алфавита в другой.

- **Зачем применяется:**
  - Сокращение размера передаваемых данных
  - Контроль целостности

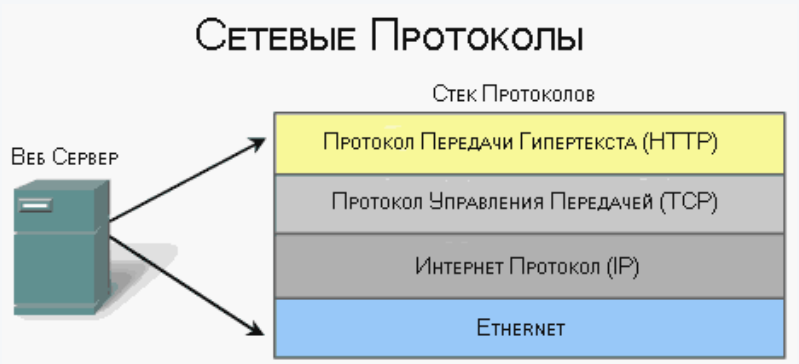
- **Что означает для URL:**
  - URL поддерживает ASCII печатаемые символы. Остальные символы кодируются %hh
  - Существует поддержка Unicode символов
  - Фрагмент DNS имя может быть закодировано в PunyCode

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Обозначение протоколов в URL

Примеры:

- view-source:
- jar:
- view-cache:
- config:
- feed:
- hcp:
- its:
- mhtml:



02

Cookie

Механизм или протокол?

- Механизм описан в RFC 6265
- Позволяет хранить данные небольшими кусками name=value
- Управляется со стороны сервера: Set-Coookie:
- Параметры: Expires, Max-age, Domain, Path.
- Параметры безопасности:
- Secure attribute - запрещает передачу Cookie через http
- HttpOnly attribute - Не позволяет читать Cookie document.cookie

03

Основные протоколы

HTTP

- Изначально - протокол для передачи страниц со специальной разметкой
- Протокол предназначался для передачи текстовой информации
- На данный момент существует 2 версии

- HTTP 1.0,1.1:
  - Пакет разделен на заголовок и тело
  - Заголовки включают в себя специальные параметры
  - Протокол не поддерживает состояние соединения. Поэтому приложение должно самостоятельно контролировать, что сейчас работает сессия с конкретным клиентом.

Основные понятия протокола

- Используется для клиент-серверного взаимодействия
- Сервер никогда не начинает взаимодействие первым
- Какое действие выполнить - определяется специальным запросом - методом

Метод	Описание
GET	Используется для того чтобы получать информацию по указанному URL
POST	Используется для отправки данных серверу без перезагрузки страницы. Обычно отправка проводится через HTML
DELETE	Удаление ресурса по указанному URL
PUT	Загрузка файла по указанному URL
TRACE	Трассировка на основании возвращаемого контента
HEAD	Получение только заголовка, без тела запроса
OPTIONS	Список методов, которые разрешены на сервере

Секретики методов HTTP протокола

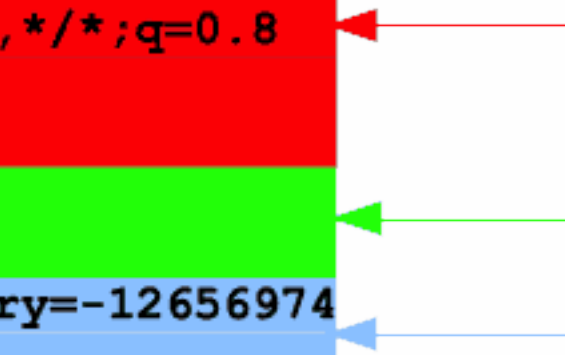
GET	POST
Запрос может быть закеширован	Запрос никогда не кэшируется
Остаётся в памяти браузера	Не хранится в памяти браузера
Может быть добавлен в закладки	Нет такой возможности
Не может использоваться для секретных данных	Может быть использован для секретных данных
Имеет ограничения на размер запроса	Нет ограничений по размеру
Используется для получения информации	Не основной функционал
Не считается защищённым	Считается безопасным

Как выглядит заголовок

OTUS

```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```



Заголовки для описания данных

Connection: keep-alive - определяет состояние соединения.

Transfer-Encoding: deflate - определяет в каком виде доставить данные

На стороне клиента возможно частичное сохранение информации полученной от сервера.

Запросы **GET** могут быть закэшированны (**200 и 301**)

Не могут кэшироваться запросы, которые участвую в аутентификации

Контроль кэширования:

- **Date/If-Modified-Since, ETag/If-None-Match**
- **Pragma:no-cache**
- **Cache-Control: no-cache, no-store**

HTTP Response коды

- **Логика работы приложения основывается на кодах возврата:**

Коды 200 - 299: Успех

200 OK Обычный ответ на запросы GET и POST. Агент получает запрошенную информацию.

204 No Content Этот код используется для завершения запроса. Агент остается на предыдущей странице

206 Partial Content Код используется для частичного возврата результата

Коды 300 - 399 Перенаправление и другие статусы

301 Moved Pernamently, 302 Found используется для редиректа агента

304 Not Modified Код, который показывает был ли модифицирован ресурс

307 Temprorary Redirect Похож на 302, но не требует специального алгоритма для редиректа

Коды возврата

Коды 400-499 Ошибки на стороне пользователя

400 Bad Request Сервер не может или не хочет обрабатывать запрос.

401 Unauthorized Код показывает, что пользователь должен подтвердить свою личность

403 Forbidden Ресурс существует, но доступ не может быть показан пользователю из-за ограничений

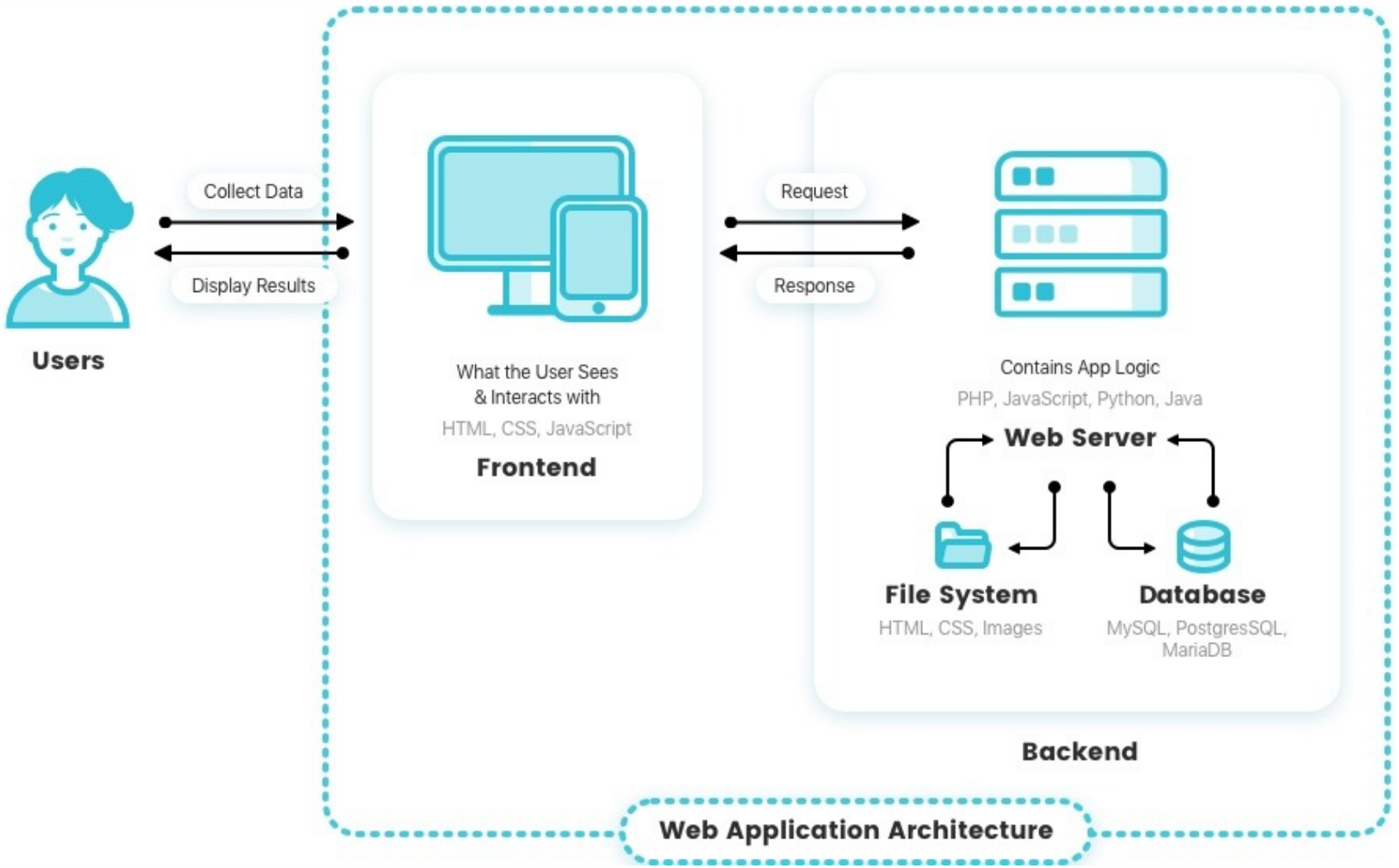
Коды 500-599 Ошибки на стороне сервера:

500 Internal Server Error, 503 Service Unavaliable Сервер сообщает о проблеме, которая возникла и не дает ему ответить на запрос.

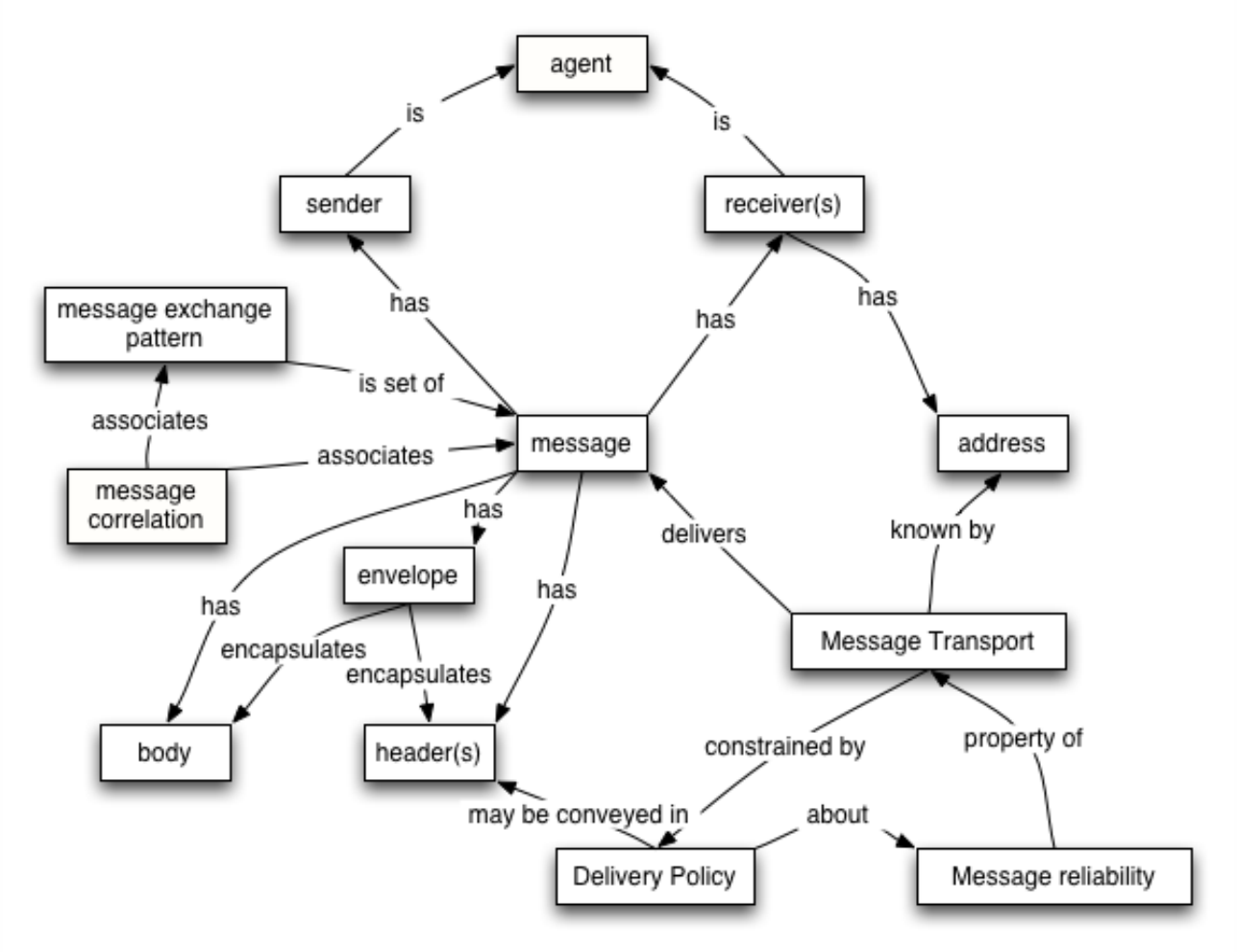
04

Архитектура

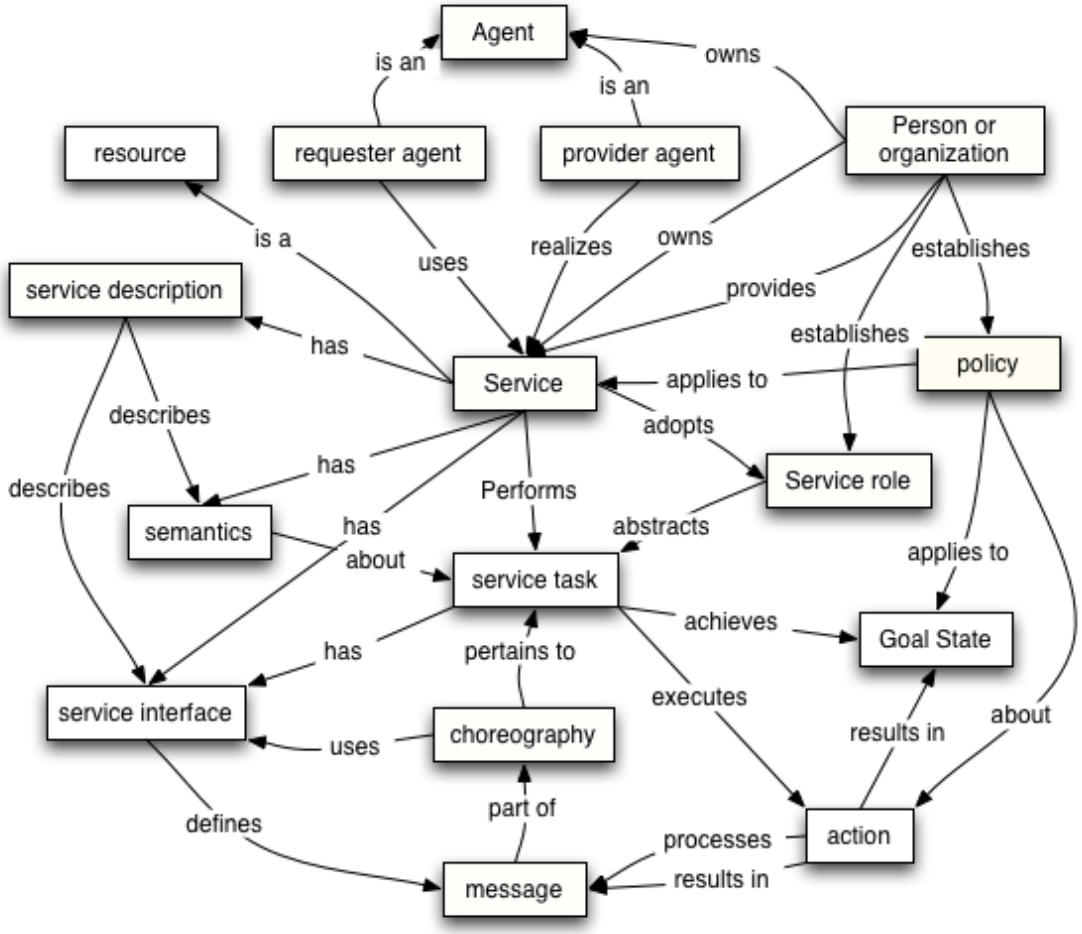
Архитектура Web приложений



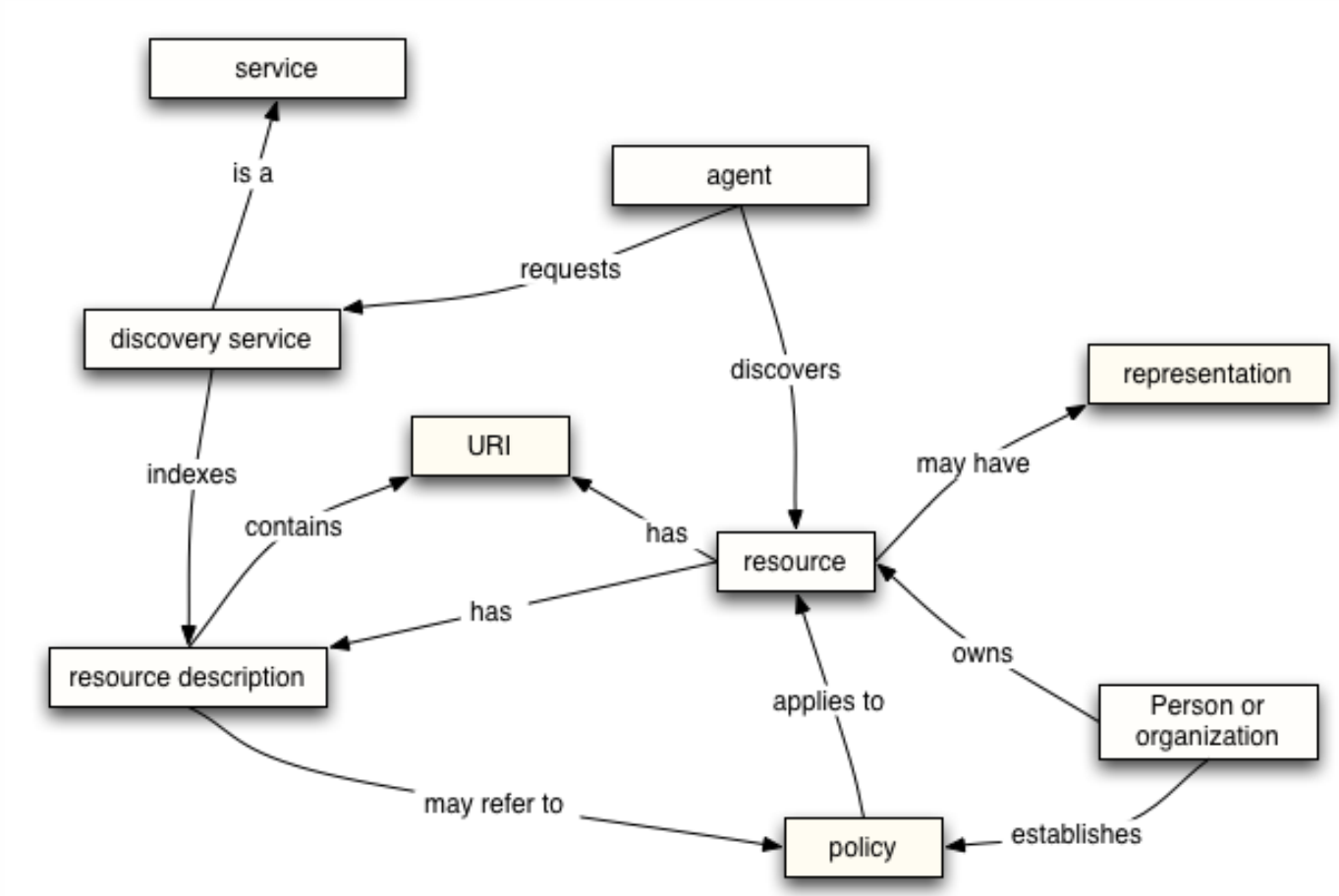
Message Oriented Architecture

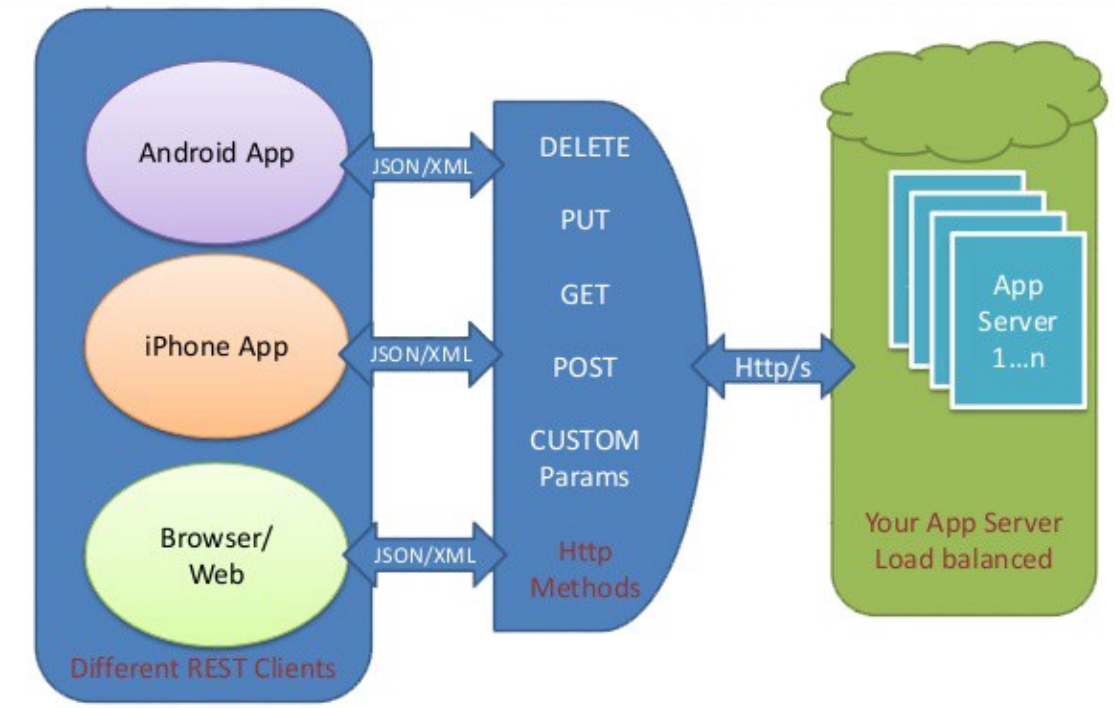


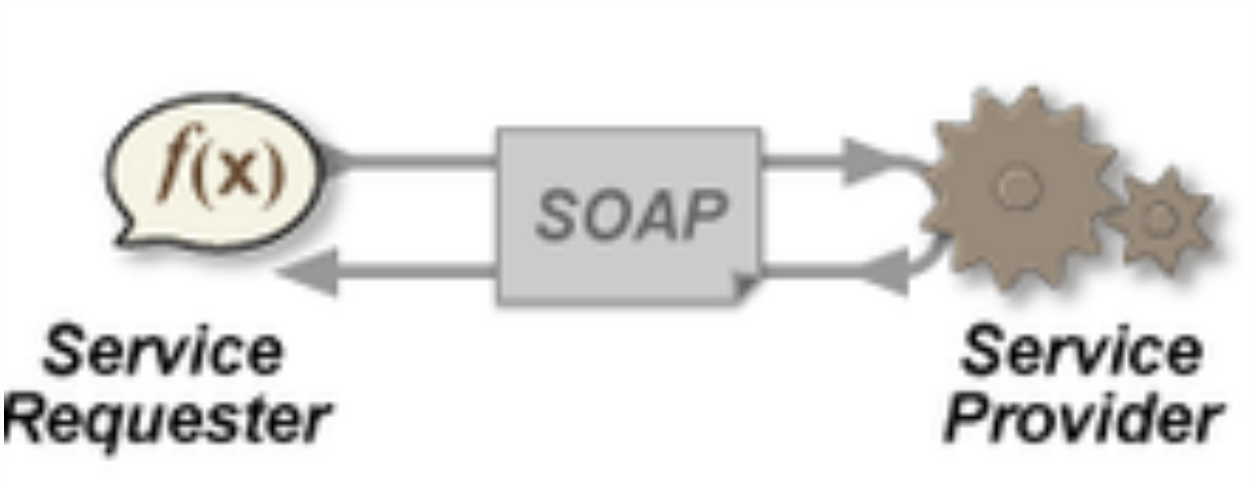
Service Oriented Architecture



Resource Oriented Architecture




Android App
iPhone App
Browser/
Web
Different REST Clients
JSON/XML
JSON/XML
JSON/XML
DELETE
PUT
GET
POST
CUSTOM
Params
Http
Methods
Http/s
App
Server
1...n
Your App Server
Load balanced


f(x)
SOAP
Service Requester
Service Provider

05

Web серверы

Что такое web сервер

- Может быть приложением или устройством
- Хранит данные: html, css, javascript файлы
- Предоставляет доступ к хранимым ресурсам
- Понимает URL схему

- **Самые популярные приложения:**
  - Nginx
  - Apache

06

Механизмы защиты

HSTS

- **HTTP Strict Transport Security** - заголовок, который заставляет браузер использовать только шифрованное соединение.
- Составные параметры:
- **Max-age** - как долго агент пользователя будет ссылаться на соединение **HTTPS**. Обычно ставится на срок 6 месяцев
- **includeSubDomains** - должен ли браузер инициировать шифрованное соединение для поддоменов.
- **preload** - должен ли сайт быть добавлен в специальный список. Может быть установлен только если установлен предыдущий заголовок.
- Пример:
  - Strict-Transport-Security: max-age=63072000; includeSubDomains; preload

HPKP

- **HTTP Public Key Pinning** - заставляет агента использовать для домена конкретный сертификат или ключ
- Используется в крайнем случае, агент инструктирует сервером, что нужно использовать заранее полученный сертификат или ключ при доступе к конкретному серверу
- **Пример:**
  - Public-Key-Pins: max-age=1296000; includeSubDomains; pin-sha256=«WoiWRyIOVNa9ihaBciRSC7XHjliYS9VwUGOIud4PB18="; pin-sha256="YLh1dUR9y6Kja30RrAn7JKnbQG/uEtLMkBgFF2Fuihg="; pin-sha256="P0NdsLTMT6LSwXLuSEHNlvg4WxtWb5rIJhfZMyeXUE0="

CSP

- **Content Security Policy** - заголовок, который позволяет организовать контроль над тем откуда будут загружаться ресурсы на сайт
- Заголовок может управлять:
  - JavaScript ресурсами
  - CSS
  - HTML Frames
  - web worker
  - fonts
  - Images
  - Java Applets
  - ActiveX
  - audio, video и HTML5 медиа

CSP

- Применяется на современных ресурсах и в современных web фреймворках.
- Имеет несколько версий.

- **Пример:**
  - Content-Security-Policy: default-src https:
  - <meta http-equiv="Content-Security-Policy" content="default-src https:»>
  - Content-Security-Policy: default-src 'none'; font-src 'https://fonts.googleapis.com'; img-src 'self' https://i.imgur.com; object-src 'none'; script-src 'self'; style-src 'self'

Cookie

Set-Cookie: MOZSESSIONID=980e5da39d4b472b9f504cac9; Path=/; Secure; HttpOnly

CORS

- **Access-Control-Allow-Origin** - HTTP заголовок, который позволяет производить доступ к ресурсам в домене A из домена B.
- Для доступа используется JavaScript
- Пример: Access-Control-Allow-Origin: *

SOP

- **Same Origin Policy** - модель безопасности, которой следуют современные веб приложения
- Модель следит за:
  - Портом ресурса
  - Используемый протокол
  - URI Scheme
- Алгоритм вычисления origin можно найти:
  - RFC 6454, RFC 3986

SOP зачем?

- Защита сайтов, где используется сессия для работы пользователя
- Без нее возможны атаки CSRF даже с включенной против них защитой

Можно ли обойти ограничения SOP?

- Две страницы могут получать доступ к данным друг друга, если установлено одинаковое значение: document.domain
- CORS
- Cross-document messaging - механизм, который позволяет выполнять асинхронную отправку сообщений из одной вкладки браузера в другую (onmessage)
- JSONP - при определенных условиях
- Web-Sockets - только если domain есть в белом списке
- Псевдо протоколы

Колесников Александр

Спасибо
за внимание!