Лекция 1
Клиент-серверная архитектура. Протокол HTTP.

Introduction to Web Development

План лекции

- О курсе
- Клиент-серверная архитектура
- Протоколы прикладного уровня: HTTP и HTTPS
- Структура и методы HTTP запроса и ответа
- Обзор Web-серверов
- Обзор серверов приложений
- Обзор Web-браузеров

О курсе

Продолжительность курса: **1 семестр**

- **16** лекций
- **16** практических занятий
- **2** модульных контроля

О курсе

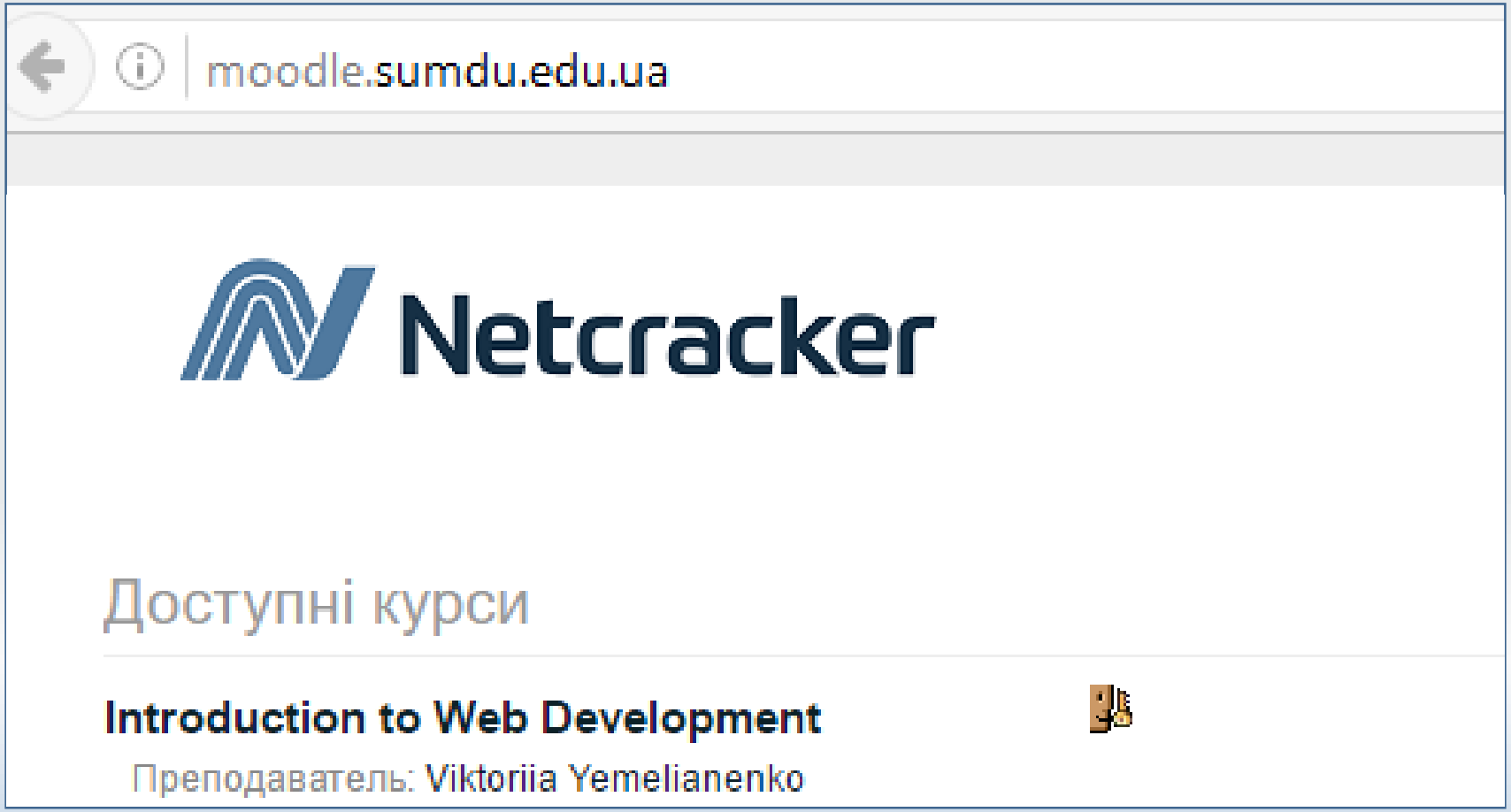
Итоговая оценка:

10% - посещение занятий

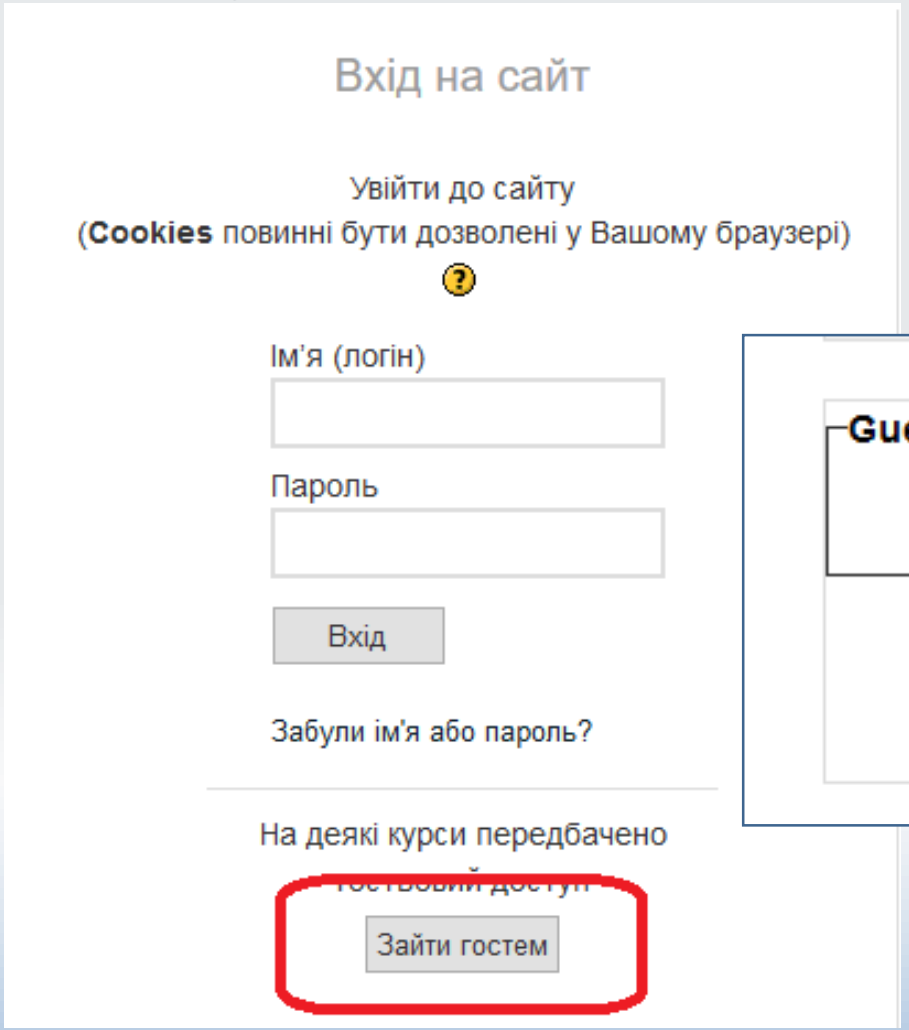
50% - практические работы

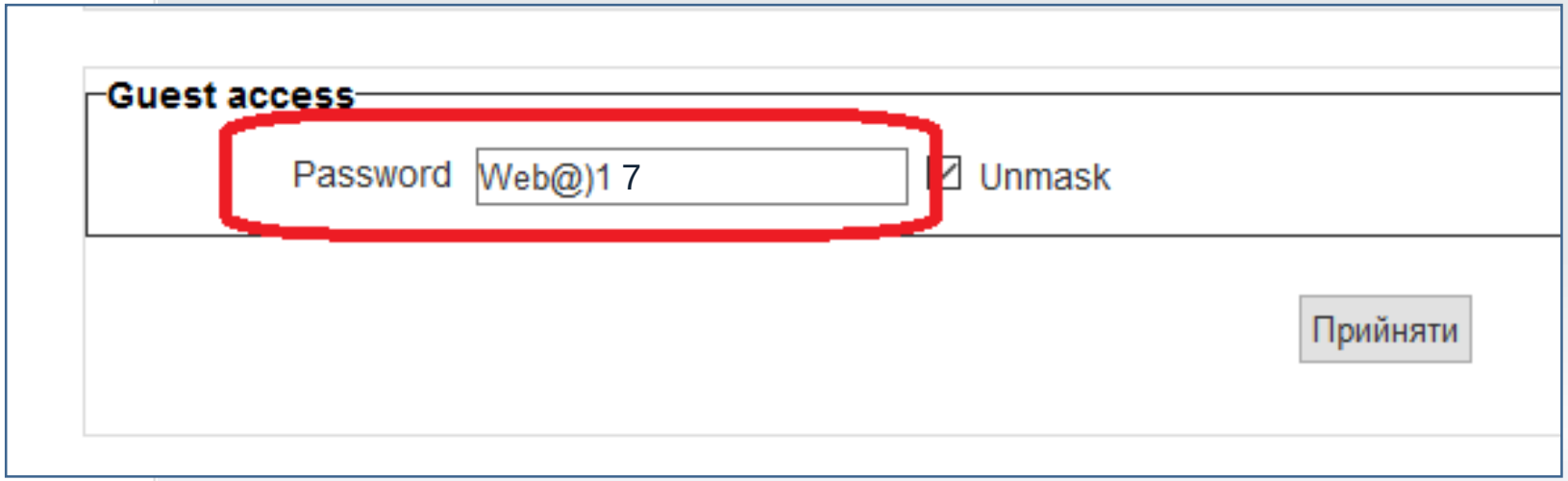
40% - модульные контроли

О курсе


moodle.sumdu.edu.ua
Netcracker
Доступні курси
Introduction to Web Development
Преподаватель: Viktoriia Yemelianenko

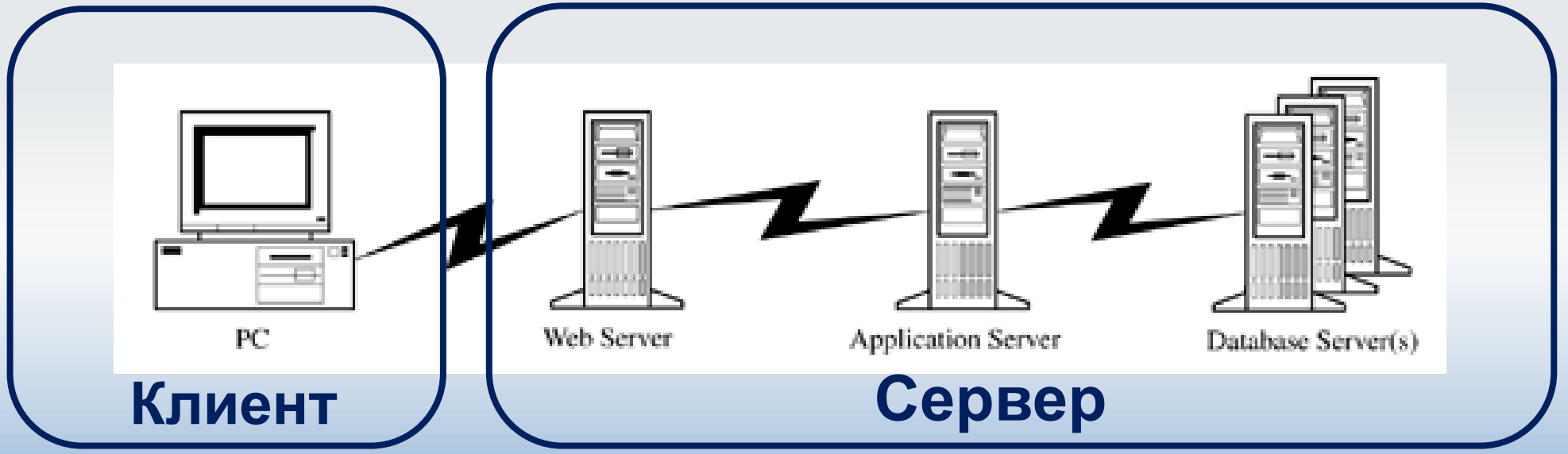
О курсе


Вхід на сайт
Увійти до сайту
(**Cookies** повинні бути дозволені у Вашому браузері)
Ім'я (логін)
Пароль
Вхід
Забули ім'я або пароль?
На деякі курси передбачено
Зайти гостем


Guest access
Password
Web@)1 7
Unmask
Прийняти

Клиент-серверная архитектура

- **Архитектура клиент-сервер (Client-server)** определяет общие принципы организации взаимодействия в сети, где имеются ***серверы***, узлы-поставщики некоторых специфичных функций (сервисов) и ***клиенты***, потребители этих функций.
- **Клиентом (front end)** является запрашивающая машина (обычно ПК), **сервером (back end)** — машина, которая отвечает на запрос.



Клиент-серверная архитектура

- **Веб-сервер (Web Server)** — это сервер, **принимающий HTTP-запросы от клиентов**, обычно веб-браузеров, и **выдающий им HTTP-ответы**, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными. Обеспечивает взаимодействие клиента с сервером.

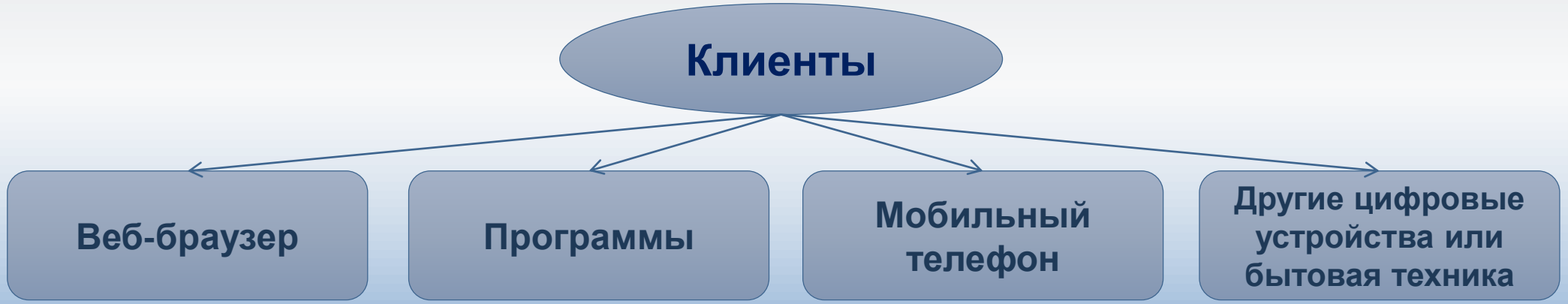
- **Сервер приложений (Application Server)** — это сервисная программа, которая обеспечивает доступ клиентов к прикладным программам, выполняющимся на сервере. Реализует бизнес-логику и позволяет вызывать на исполнение процедуры и функции ПО.

- **Сервер баз данных (Database Server)** выполняет обслуживание и управление базой **данных** и отвечает за целостность и сохранность **данных**, а также обеспечивает операции ввода-вывода при доступе к информации.

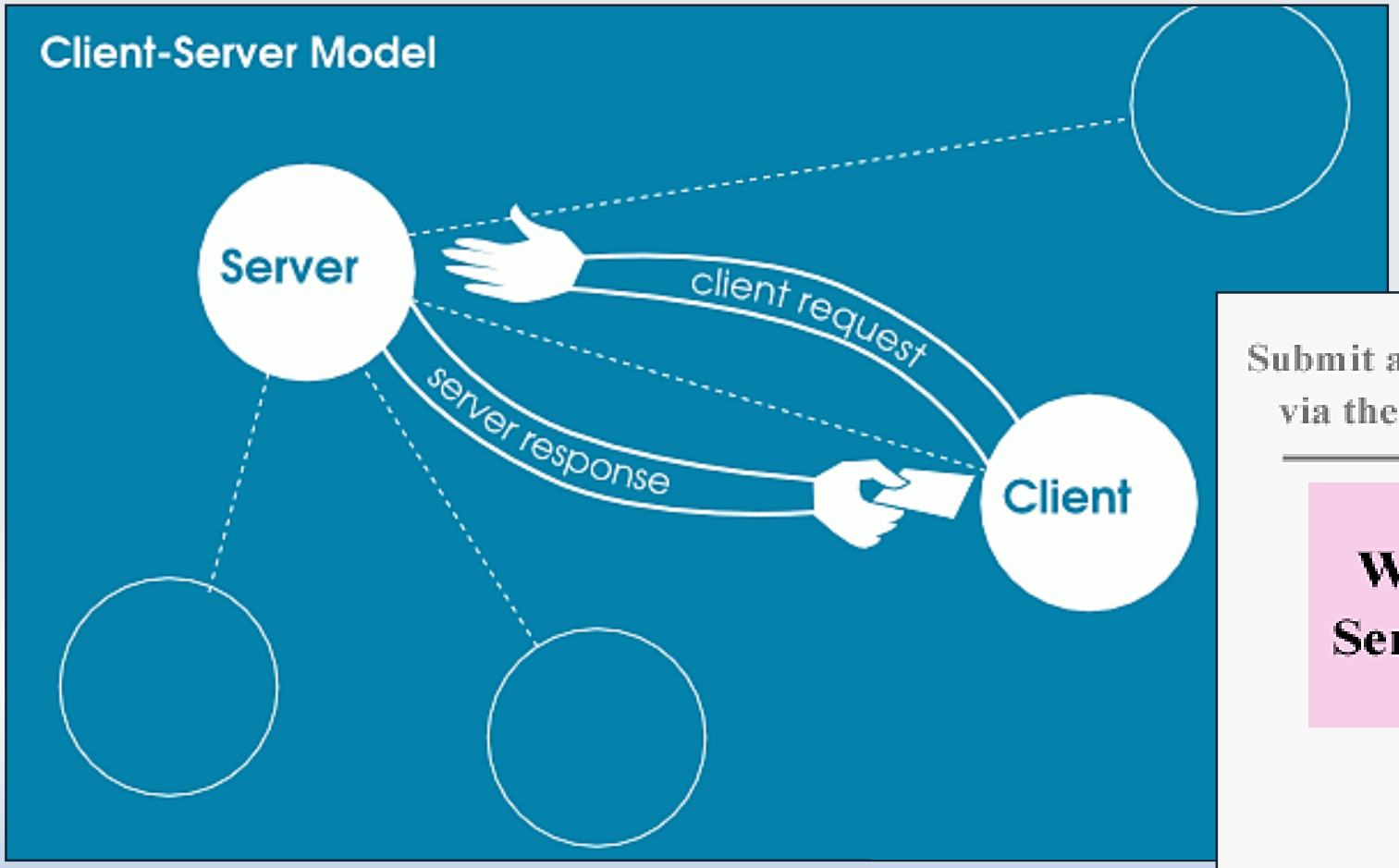
Клиент-серверная архитектура

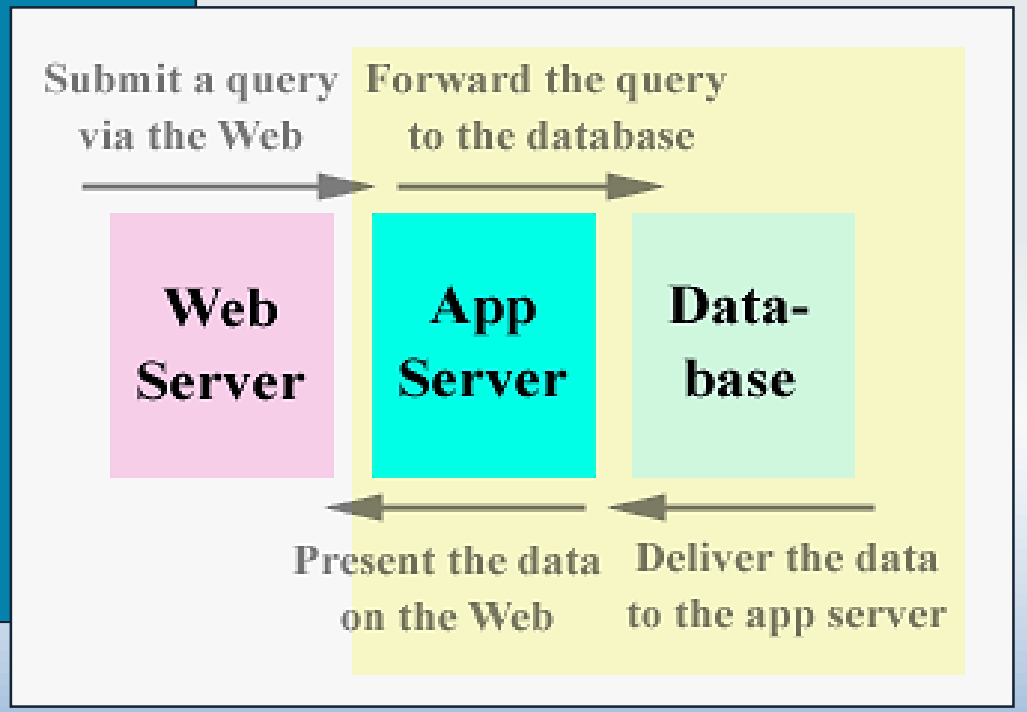
Веб-сервер — это сервер, **принимающий HTTP-запросы от клиентов**, обычно веб-браузеров, и **выдающий им HTTP-ответы**, как правило, вместе с **HTML-страницей, изображением, файлом, медиа-потоком или другими данными.**

Клиент — это аппаратный или программный компонент вычислительной системы, **посылающий запросы серверу**.



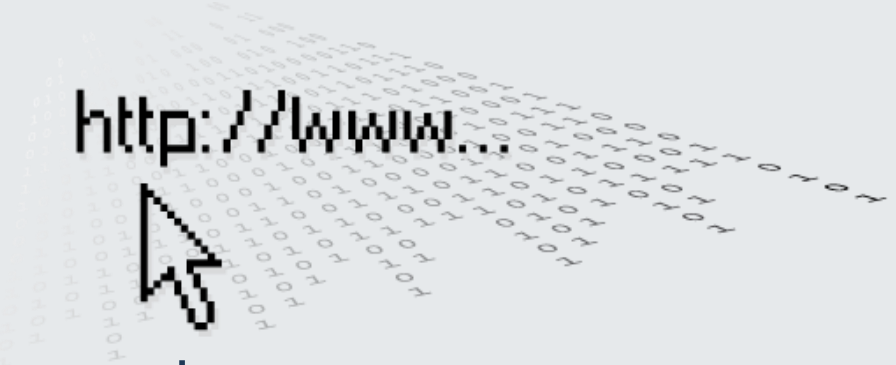
Клиент-серверная архитектура





Протоколы прикладного уровня: HTTP и HTTPS

HTTP (Hypertext Transfer Protocol):



- протокол **прикладного уровня**;
- **обмен контентом** между web-сервером и web-клиентом.
- обмен сообщениями по схеме «**запрос-ответ**».
- для идентификации ресурсов HTTP использует **глобальные URI**.
- работает **поверх TCP/IP**, обеспечивая обмен запросами/отметами между клиентом и сервером.

Порт: 80, 8080

Протоколы прикладного уровня: HTTP и HTTPS

Порт: 443

HTTPS (Secure HTTP):

- **расширение протокола HTTP**, поддерживающее **шифрование**;
- **безопасный обмен контентом**;
- **защита от атак;**
- используется для приложений, в которых **важна безопасность соединения**;
- **проприетарный протокол.**

Протоколы прикладного уровня: HTTP и HTTPS

HTTP (Hypertext Transfer Protocol)

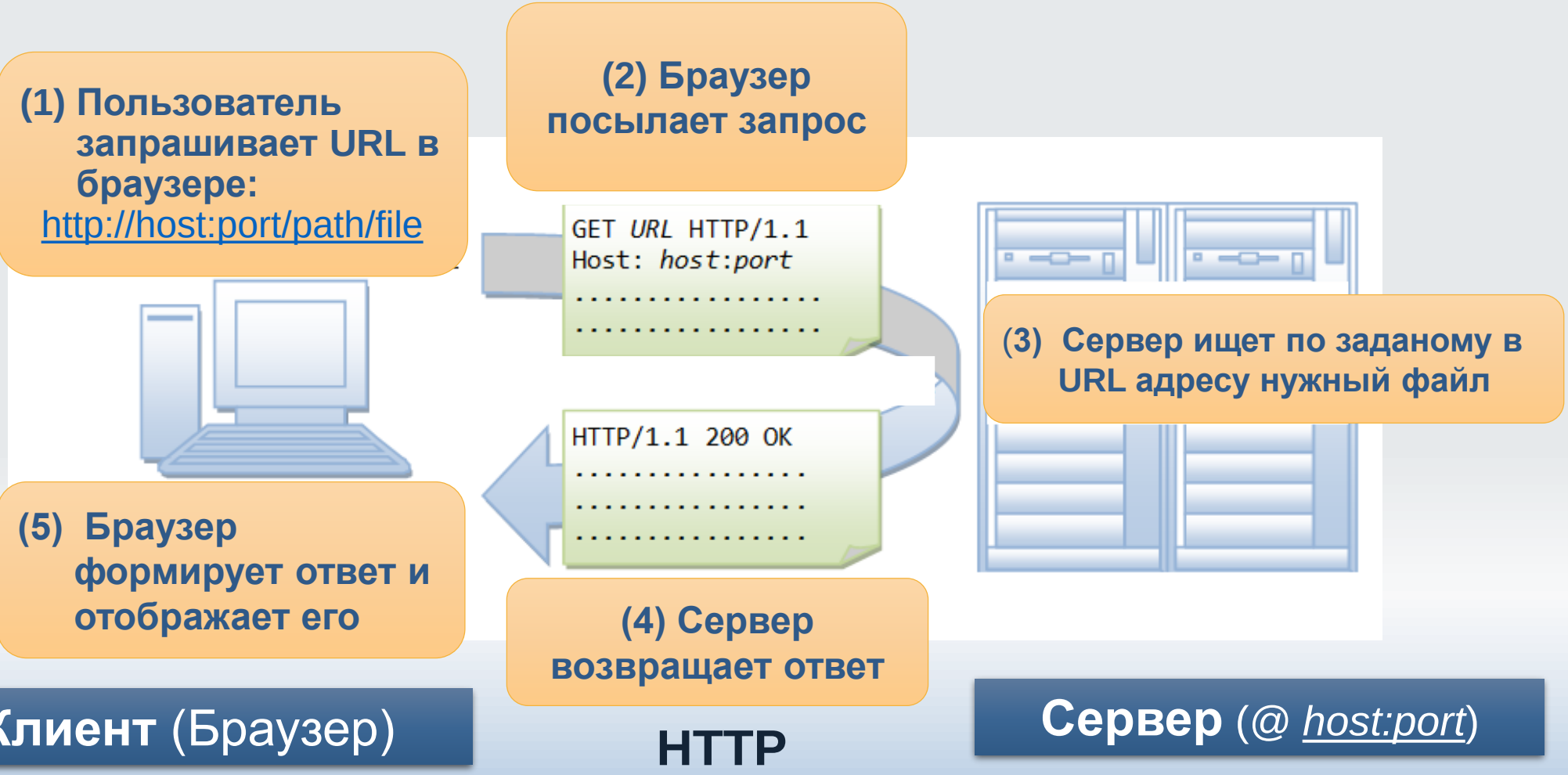
Почему он так важен?

HTTP — протокол, пронизывающий веб

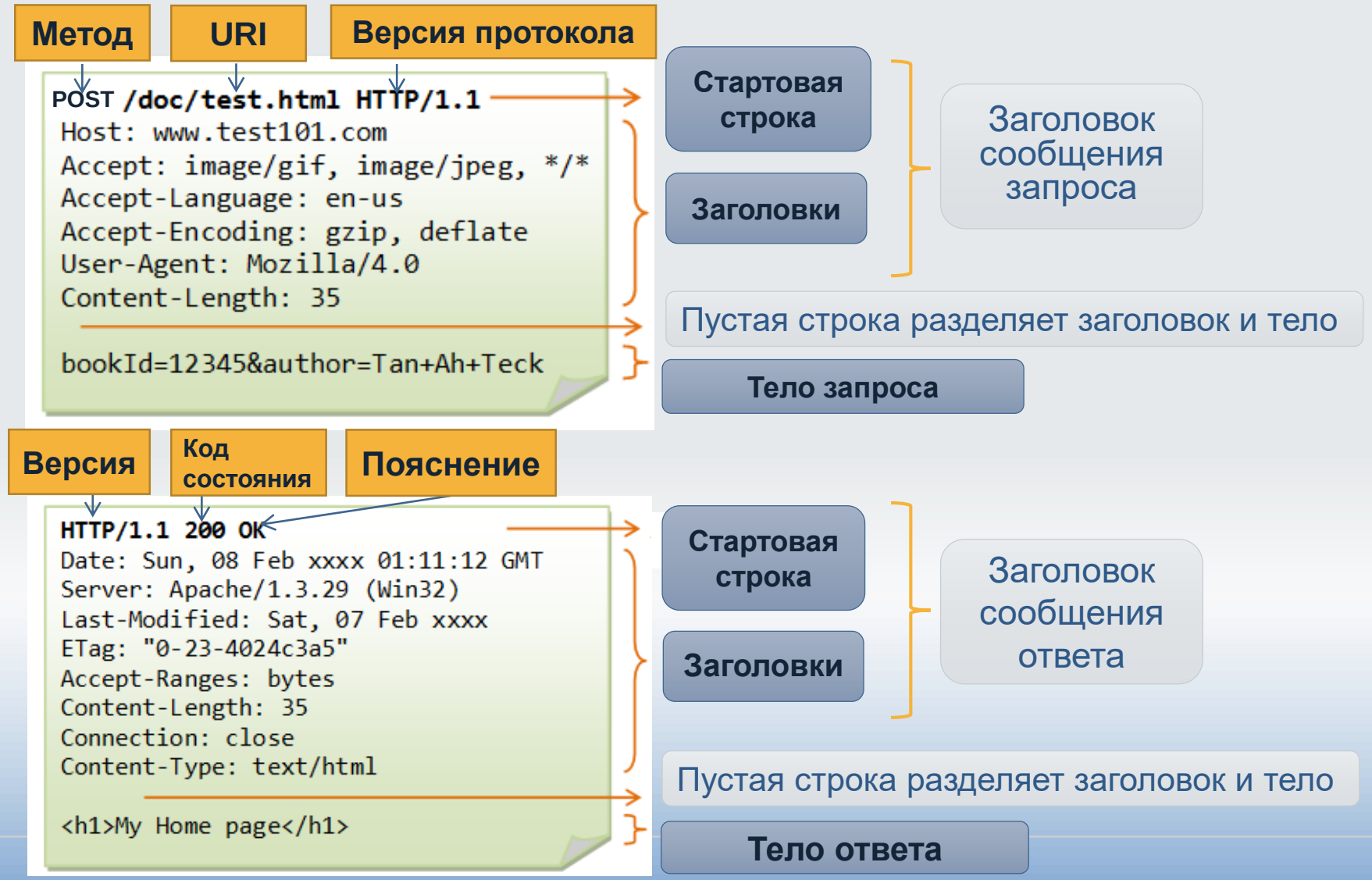
Знать структуру и методы HTTP обязан каждый веб-разработчик

Понимание принципов работы HTTP поможет вам делать более качественные веб-приложения

Протоколы прикладного уровня: HTTP и HTTPS



Структура и методы HTTP запроса и ответа



Структура и методы HTTP запроса и ответа

1xx: Informational (информационные):

102 Processing («идёт обработка»).

2xx: Success (успешно):

200 OK («хорошо»).
206 Partial Content («частичное содержимое»).

3xx: Redirection (перенаправление):

302 Moved Temporarily («перемещено временно»).
304 Not Modified (не изменялось).

4xx: Client Error (ошибка клиента):

400 Bad Request («плохой, неверный запрос»).
401 Unauthorized («неавторизован»).
404 Not Found («не найдено»).

5xx: Server Error (ошибка сервера):

500 Internal Server Error («внутренняя ошибка сервера»).

Больше: https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP

Структура и методы HTTP запроса и ответа

Методы HTTP

POST
передача пользовательских данных заданному ресурсу

GET
запрос содержимого указанного ресурса

LINK
устанавливает связь указанного ресурса с другими

DELETE
удаляет указанный ресурс

PUT
загрузка содержимого запроса на указанный в запросе URI

PATCH
аналогичен PUT, но применяется только к фрагменту ресурса

UNLINK
убирает связь указанного ресурса с другими

CONNECT
преобразует соединение запроса в прозрачный TCP/IP туннель

HEAD
извлечение метаданных, проверки наличия ресурса, обновлений

OPTIONS
определение возможностей веб-сервера или параметров соединения

TRACE
возвращает полученный запрос так, что клиент может увидеть какую информацию промежуточные серверы добавляют или изменяют в запросе

Структура и методы HTTP запроса и ответа

GET

- **тело запроса пустое;**
- запрос обрабатываются **быстрее и с меньшим потреблением ресурсов**;
- передача переменных **в адресной строке** (данные не защищены);
- способен передать **небольшое количество данных**: есть ограничения на длину URL (1024 симв.);
- может передать **только ASCII символы**;
- запрос можно **скопировать, сохранить**;
- запрос может **кэшироваться**;
- доступны **условные и частичные запросы**;
- **не разрывает HTTP соединение** (при включенном на сервере Keep Alive).

?

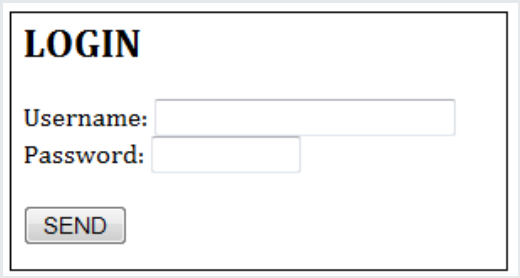
POST

- передача данных **в теле запроса**;
- обработка **медленнее и «тяжелее»**;
- способен передать **большие объемы данных (**лимит устанавливается веб-сервером);
- способен **передать файлы**;
- **нельзя сохранить** в закладки;
- **разрывает HTTP соединение**;
- для передачи информации браузер отправляет **минимум два TCP пакета**: заголовок, а потом тело запроса.

Структура и методы HTTP запроса и ответа

Примеры HTTP запросов

Web-страница:



Исходный код Web-страницы:

POST или GET?

```
<body>
  <h2>LOGIN</h2>
  <form method="    " action="/bin/login">
    Username: <input type="text" name="user" size="25" /><br />
    Password: <input type="password" name="pw" size="10" /><br /><br />
    <input type="hidden" name="action" value="login" />
    <input type="submit" value="SEND" />
  </form>
</body>
```

HTTP **GET** запрос

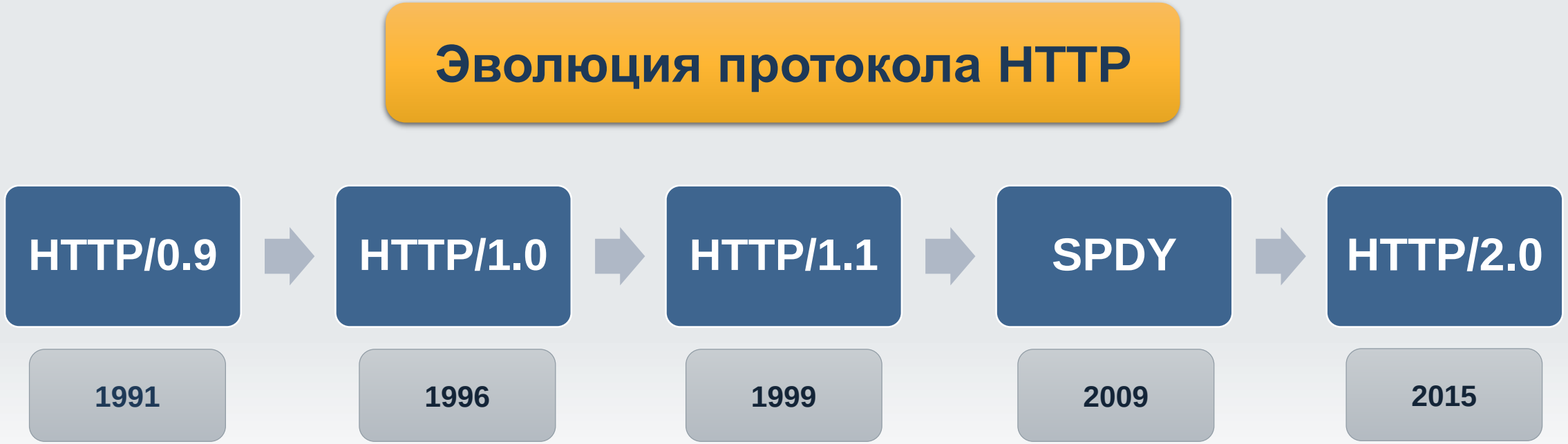
```
GET /bin/login?user=Peter+Lee&pw=123456&action=login HTTP/1.1
Accept: image/gif, image/jpeg, */*
Referer: http://127.0.0.1:8000/login.html
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
Connection: Keep-Alive
```

HTTP **POST** запрос

```
POST /bin/login HTTP/1.1
Host: 127.0.0.1:8000
Accept: image/gif, image/jpeg, */*
Referer: http://127.0.0.1:8000/login.html
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Length: 37
Connection: Keep-Alive
Cache-Control: no-cache

User=Peter+Lee&pw=123456&action=login
```

История развития протокола HTTP



История развития протокола HTTP HTTP

HTTP/0.9

- Единственный метод — **GET**.
- **Нет заголовков**.
- Спроектирован только для **HTML**-ответов

Если клиенту нужно было получить какую-либо страницу на сервере, он делал запрос:

GET /index.html

Ответ выглядел примерно так:

(response body)
(connection closed)

Сервер получает запрос, посылает HTML в ответ, и как только весь контент будет передан, закрывает соединение.

История развития протокола HTTP

HTTP/1.0

Теперь сервер мог послать любой тип контента клиенту, поэтому словосочетание «Hyper Text» в аббревиатуре HTTP стало искажением. HMTP, или Hypermedia Transfer Protocol, стало бы более уместным названием, но все к тому времени уже привыкли к HTTP.

- Может получать в качестве ответа, помимо HTML, другие форматы: **изображения, видео, текст и другие типы контента**.
- Добавлены новые методы (**POST** и **HEAD**)
- Изменился **формат запросов/ответов**:
  - к запросам и ответам добавились **HTTP-заголовки**;
  - добавлены **коды состояний**, чтобы различать разные ответы сервера.
- Введена **поддержка кодировок**.
- Добавлены составные типы данных (multi-part types), авторизация, кэширование, различные кодировки контента и ещё многое другое.

Один из недостатков HTTP/1.0 — невозможно отправить несколько запросов во время одного соединения.

История развития протокола HTTP

HTTP/1.1

Потоковая передача данных, при которой клиент может в рамках соединения посылать множественные запросы к серверу, не ожидая ответов, а сервер посылает ответы в той же последовательности, в которой получены запросы.

Новые HTTP-методы — PUT, PATCH, HEAD, OPTIONS, DELETE.

Идентификация хостов (обязательность заголовка HOST) . В HTTP/1.0 заголовок Host не был обязательным.

Постоянные соединения, т.е. соединения, которые по умолчанию не закрывались, оставаясь открытыми для нескольких последовательных запросов. **Connection: keep-alive.**
Чтобы закрыть соединение, нужно было при запросе добавить заголовок **Connection: close**.
Клиенты обычно посылали этот заголовок в последнем запросе к серверу, чтобы безопасно закрыть соединение.

История развития протокола HTTP

HTTP/1.1

HTTP/1.1 ввёл **chunked encoding** — механизм разбиения информации на небольшие части (chunks) и их передачу.

Но как же клиент узнает, когда закончится один ответ и начнётся другой? Для разрешения этой задачи устанавливается **заголовок Content-Length**, с помощью которого клиент определяет, где заканчивается один ответ и можно ожидать следующий.

Chunked Transfers. Если контент строится динамически и сервер в начале передачи не может определить *Content-Length*, он начинает отсылать контент частями, друг за другом, и добавлять *Content-Length* к каждой передаваемой части. Когда все части отправлены, посылается пустой пакет с заголовком *Content-Length*, установленным в 0, сигнализируя клиенту, что передача завершена. Чтобы сказать клиенту, что передача будет вестись по частям, сервер добавляет заголовок *Transfer-Encoding: chunked*.

История развития протокола HTTP

HTTP/1.1

HTTP/1.1 появился в 1999 и пробыл стандартом долгие годы.

В отличие от базовой аутентификации в HTTP/1.0, в HTTP/1.1 добавились:

- **Дайджест-аутентификация и прокси-аутентификация.**
- **Кэширование**.
- **Диапазоны байт** (byte ranges).
- **Кодировки.**
- **Согласование содержимого (content negotiation).**
- **Клиентские куки.**
- **Улучшенная поддержка сжатия.**
- И другие...

История развития протокола HTTP

SPDY

В 2015 в Google решили, что не должно быть двух конкурирующих стандартов, и объединили SPDY с HTTP, дав начало HTTP/2

Основная идея SPDY:
сделать веб быстрее и улучшить уровень безопасности за счёт уменьшения времени задержек веб-страниц.

SPDY включал в себя мультиплексирование, сжатие, приоритизацию, безопасность и т.д…

SPDY не старался заменить собой HTTP. Он был переходным уровнем над HTTP

История развития протокола HTTP

HTTP/2.0

HTTP/2 разрабатывался **для транспортировки контента с низким временем задержки**.

HTTP/2 уже здесь, и уже обошёл SPDY в поддержке

Главные отличия от HTTP/1.1:

- **бинарный вместо текстового.** Бинарные сообщения быстрее разбираются автоматически, но, в отличие от HTTP/1.x, не удобны для чтения человеком. Основные составляющие HTTP/2 — фреймы (Frames) и потоки (Streams).
- **мультиплексирование — передача нескольких асинхронных HTTP-запросов по одному TCP-соединению:** клиенту не придётся простаивать, ожидая обработки длинного запроса, ведь во время ожидания могут обрабатываться остальные запросы.
- **сжатие заголовков методом HPACK**
- **Server Push — несколько ответов на один запрос:** сервер, зная, что клиент собирается запросить определённый ресурс, может отправить его, не дожидаясь запроса.
- **приоритизация запросов**
- **безопасность**

Обзор Web-серверов

- свободное ПО
- кроссплатформенный
- надёжность и гибкость конфигурации
- поддержка модулей для работы с различными языками программирования и системами разработки

- коммерческое ПО
- бесплатно с ОС семейства Windows NT
- поддержка технологий ASP, ASP.NET, CGI, FastCGI, ISAPI, SSI
- поддержка работы почтовых сервисов SMTP/POP3 сервисов

NGINX™

- свободный веб-сервер и почтовый прокси-сервер
- бесплатно с ОС семейства Windows NT
- целесообразно использовать для статических веб-сайтов и как прокси-сервера перед динамическими сайтами.

Обзор Web-серверов

Google Web Server (GWS) — веб-сервер, используемый Google для организации своей веб инфраструктуры.

Обзор Web-серверов

Most popular web servers

© W3Techs.com	usage	change since 1 August 2016
1. Apache	51.8%	-0.2%
2. Nginx	30.8%	+0.2%
3. Microsoft-IIS	12.0%	
4. LiteSpeed	2.3%	-0.1%
5. Google Servers	1.3%	

percentages of sites

© W3Techs.com	usage	change since 1 August 2015
1. Apache	56.6%	-0.3%
2. Nginx	25.2%	+0.3%
3. Microsoft-IIS	13.0%	-0.1%
4. LiteSpeed	2.3%	+0.1%
5. Google Servers	1.3%	

percentages of sites

Fastest growing web servers since 1 August 2016

© W3Techs.com	sites
1. Nginx	568
2. Tomcat	35
3. Microsoft-IIS	21

daily number of additional sites in the top 10 million

© W3Techs.com	sites
1. Nginx	919
2. LiteSpeed	137
3. Google Servers	52

daily number of additional sites in the top 10 million

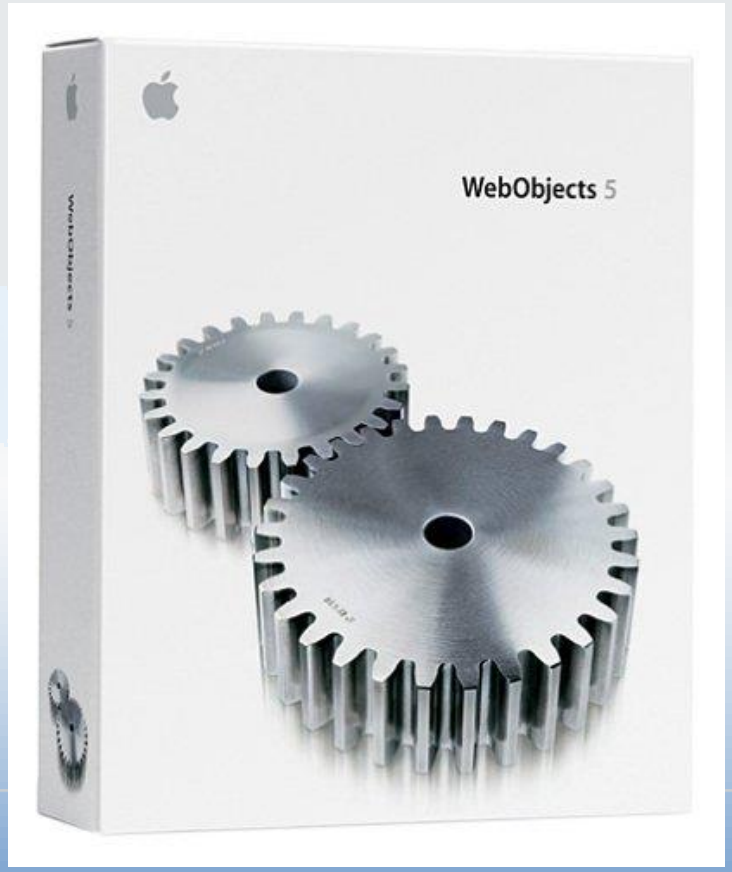
Обзор серверов приложений

ZOPE

3.1

ORACLE

Oracle GlassFish Server



IBM WebSphere

Обзор Web-браузеров

Opera
бесплатно с версии 8.5

Internet Explorer
с Microsoft Windows

Safari
с Mac OS X и бесплатно для Microsoft Windows

Mozilla Firefox
бесплатно, свободное ПО,
с дистрибутивами Linux (Ubuntu)

Microsoft Edge

Только для Windows 10

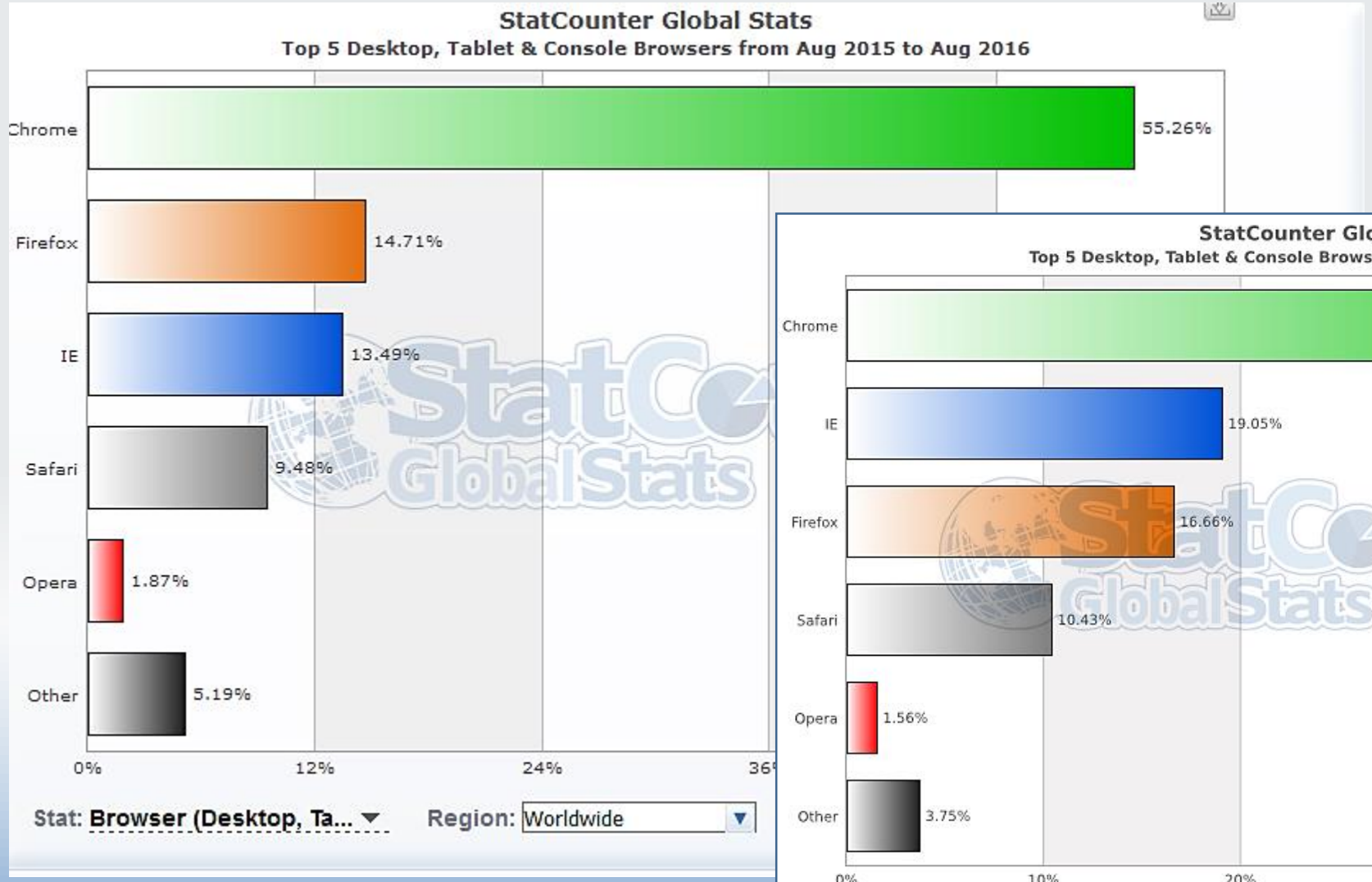
Google Chrome
бесплатно

Обзор Web-браузеров

Браузер	Движок
Internet Explorer	**Trident** (Win) **Tasman** (Mac)
Mozilla Firefox	**Gecko**
Safari	**WebKit**
Google Chrome	**Blink (с версии 28),** ранее - WebKit
Opera	**WebKit (с версии 15.0),** ранее - Presto
Microsoft Edge	**EdgeHTML,** Новый интерпретатор JavaScript — **Chakra**

Больше информации: https://ru.wikipedia.org/wiki/Сравнение_браузеров

Обзор Web-браузеров



Рекомендованная литература

- http://www.google.com/
- http://www.w3schools.com/
- https://www.w3.org/
- http://www.codenet.ru/webmast/php/HTTP-POST.php
- http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html