**«УТВЕРЖДАЮ»**

**Зав. кафедрой ИСиТ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Смелов**

**Экзаменационные вопросы дисциплины**

**«Программирование серверных кроссплатформенных приложений»**

**для студентов 3-го курса специальности ПОИТ**

1. Протокол HTTP, основные свойства HTTP, структура запроса и ответа. Протокол HTTPS. Понятие web-приложения, структура и принципы работы web-приложения. Понятие асинхронности.

***HTTP*** — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Аббревиатура HTTP расшифровывается как HyperText Transfer Protocol, «протокол передачи гипертекста». В соответствии со спецификацией OSI, HTTP является протоколом прикладного (верхнего, 7-го) уровня.

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту.

***HTTP: основные свойства***

- версии HTTP/1.1 – действующий (текстовый), HTTP/2 – черновой (не распространен, бинарный); HTTP / 3 (суть в том, что HTTP/3 — просто новый синтаксис HTTP, который работает на протоколе IETF QUIC, мультиплексированном и безопасном транспорте на основе UDP.)

- два типа абонентов: клиент и сервер;

- два типа сообщений: request и response;

- от клиента к серверу – request;

- от сервера к клиенту – response;

- на один request всегда один response, иначе ошибка;

- одному response всегда один request, иначе ошибка;

- TCP-порты: 80, 443;

- для адресации используется URI или URN;

- поддерживается W3C, описан в нескольких RFC.(RFC2616)

***Request***:

- метод;

- URI;

- версия протокола (HTTP/1.1);

- заголовки (пары: имя/заголовок);

- параметры (пары: имя/заголовок);

- расширение.

***Response:***

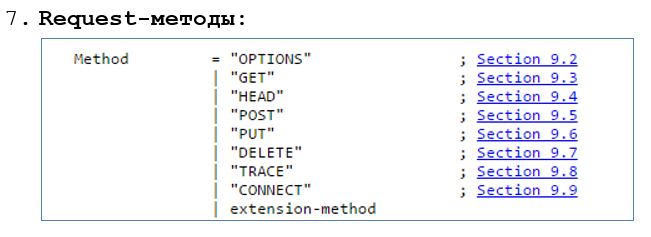
- версия протокола (HTTP/1.1);

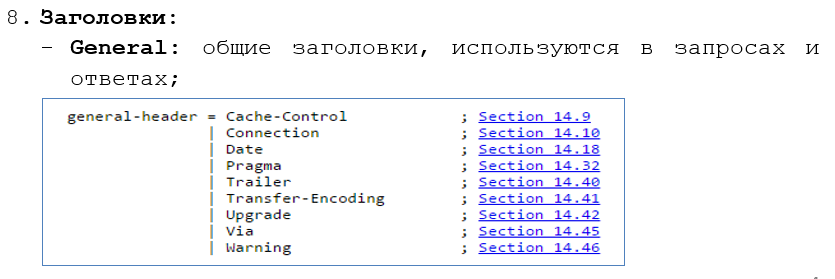
- код состояния (1xx, 2xx, 3xx, 4xx, 5xx);

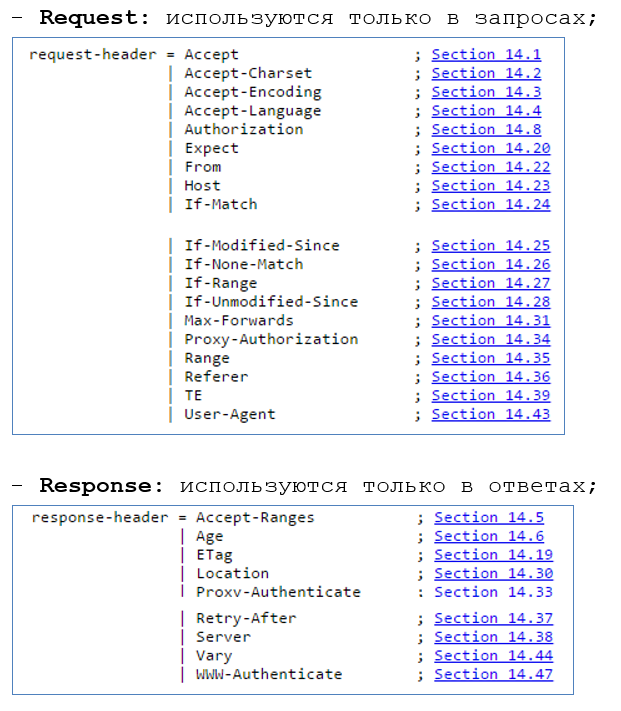
- пояснение к коду состояния;

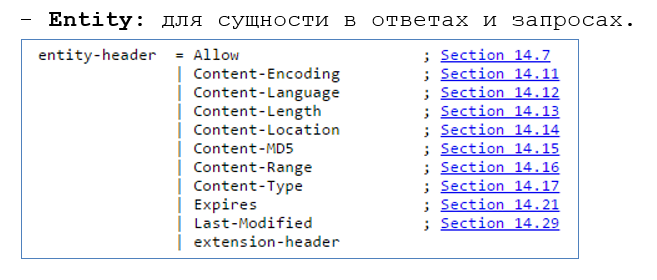
- заголовки (пары: имя/заголовок);

- расширение.









***Response: Код состояния:***

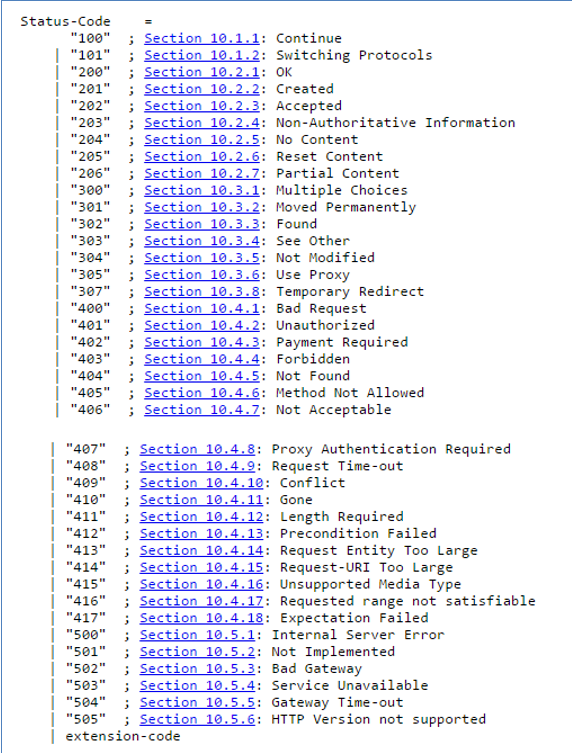
- 1xx: информационные сообщения;

- 2xx: успешный ответ;

- 3xx: переадресация;

- 4xx: ошибка клиента;

- 5xx: ошибка сервера.

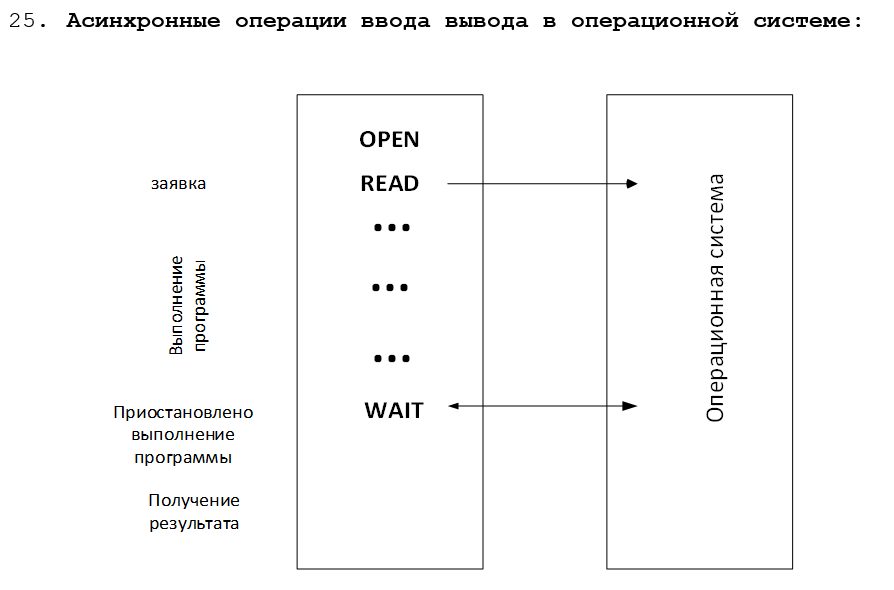


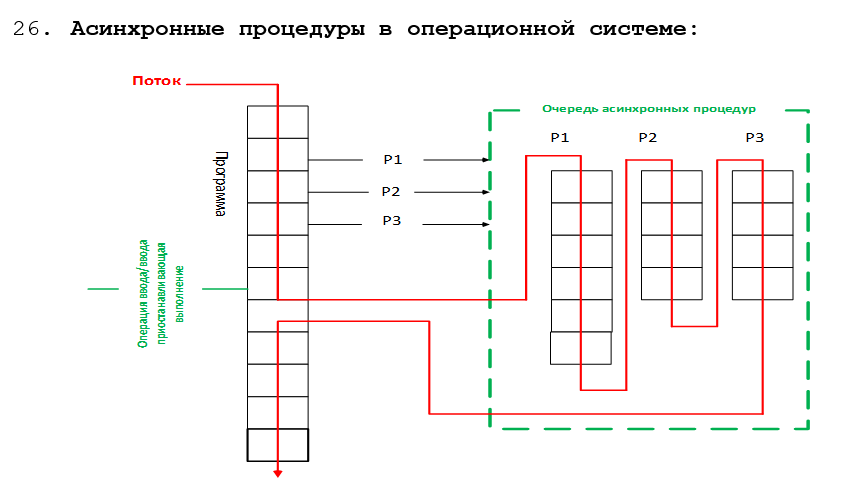
***HTTPS*** (аббр. от англ. HyperText Transfer Protocol Secure) — расширение протокола HTTP для поддержки шифрования в целях повышения безопасности. Данные в протоколе HTTPS передаются поверх криптографических протоколов TLS или устаревшего в 2015 году - SSL. В отличие от HTTP с TCP-портом 80, для HTTPS по умолчанию используется TCP-порт 443.

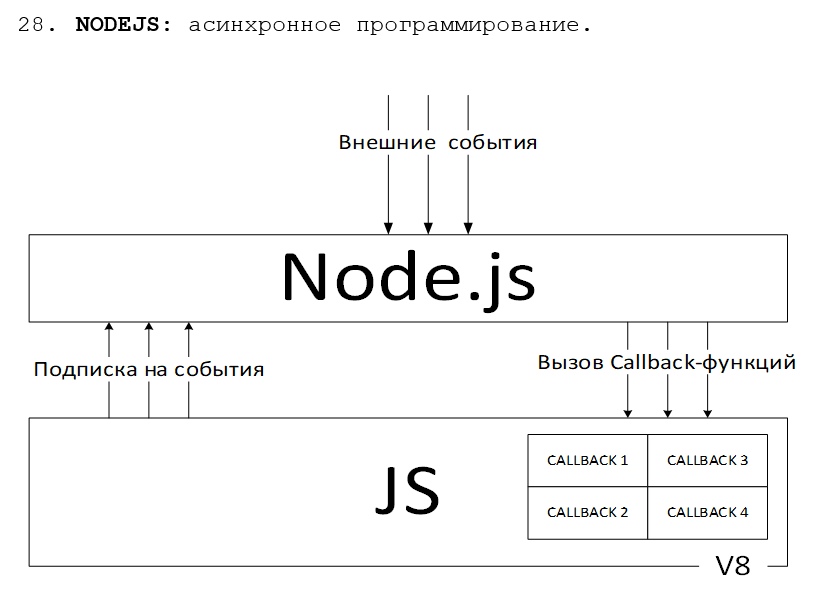
клиент-серверное приложение, у которого клиент и сервер взаимодействуют по протоколу HTTP называется ***web-приложением***. когда говорят о разработке web-приложения, говорят о разработке frontend (клиента) и backend (сервера).

***Понятие асинхронности***: операция называется асинхронной, если ее выполнение осуществляется в 2 фазы: 1) заявка на исполнение; 2) получение результата; при этом участвуют два механизма: A-механизм, формирующий заявку и потом получающий результат; B-механизм, получающий заявку от A, исполняющий операцию и отправляющий результат A; продолжительность исполнения операции B-механизмом, как правило, непредсказуемо; в то время пока B-механизм исполняет операцию, А-механизм выполняет собственную работу. Применение асинхронности не противоречит применению многопоточности.









1. HTTP-аутентификация.

**Идентификация** – заявление пользователя о себе.

**Аутентификация** – процедура проверки подлинности идентификации пользователя.

**Авторизация** -процедура проверки прав аутентифицированного пользователя.

BASIC:



Digest



Forms:



1. Протокол HTTPS. Протокол TSL. Сертификаты. Взаимодействие центра сертификации и владельца защищенного ресурса.

**Клиент** выдает запрос серверу (Client Hello).

**Сервер** подписывает свой сертификат и высылает клиенту (Server Hello).

**Клиент** проверяет сертификат в центре сертификации, которому доверяет.

**Клиент** сравнивает данные сертификата с информацией центра сертификации.

**Клиент** сообщает серверу, какие ключи шифрования он поддерживает.

**Сервер** выбирает подходящую длину ключа.

**Клиент** генерирует симметричный ключ, шифрует его открытым ключом.

**Сервер** получает симметричный ключ и расшифровывает его.





1. Генерация приватного ключа СА
2. Генерация сертификата СА
3. Генерация приватного ключа ресурса
4. Генерация запроса на сертификат для ресурса
5. Генерация сертификата для ресурса
6. Импорт СА сертификата в хранилище сертификатов
7. Протокол WebSockets, основные свойства, процедура установки соединения. WebSockets API.

WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени. (стандарт RFC 6455) Он позволяет пересылать любые данные, на любой домен, безопасно и почти без лишнего сетевого трафика.

WebSocket представляет собой альтернативу HTTP, его можно применять для организации обмена данными в веб-приложениях. Этот протокол позволяет создавать долгоживущие двунаправленные каналы связи между клиентом и сервером. После установления соединения канал связи остаётся открытым, что даёт в распоряжение приложения очень быстрое соединение, характеризующееся низкими задержками и небольшой дополнительной нагрузкой на систему. Протокол WebSocket поддерживают все современные браузеры.

HTTP и WebSocket — это очень разные протоколы, в которых используются различные подходы к обмену данными. HTTP основан на модели «запрос — ответ»: сервер отправляет клиенту некие данные после того, как они будут запрошены. В случае с WebSocket всё устроено иначе. А именно:

Сервер может отправлять сообщения клиенту по своей инициативе, не дожидаясь поступления запроса от клиента.

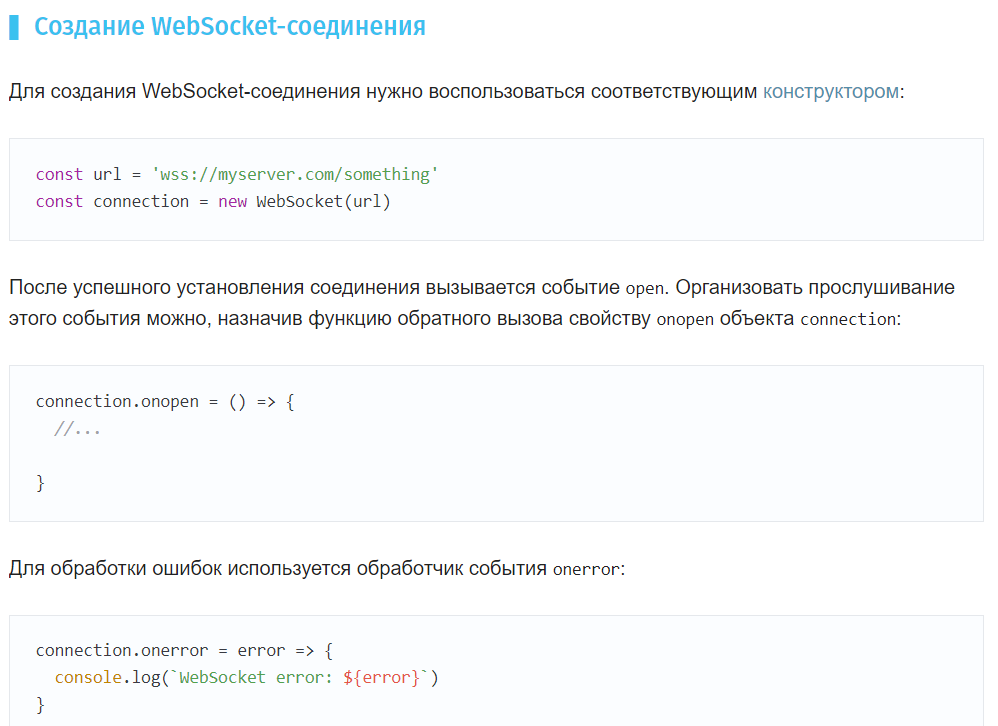
Клиент и сервер могут обмениваться данными одновременно.

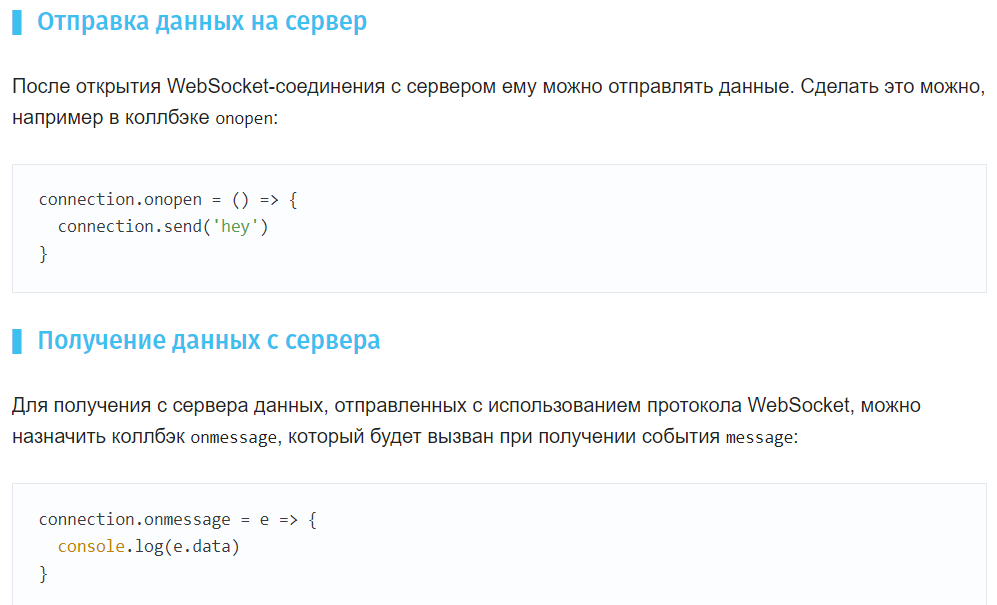
При передаче сообщения используется крайне малый объём служебных данных. Это, в частности, ведёт к низким задержкам при передаче данных.

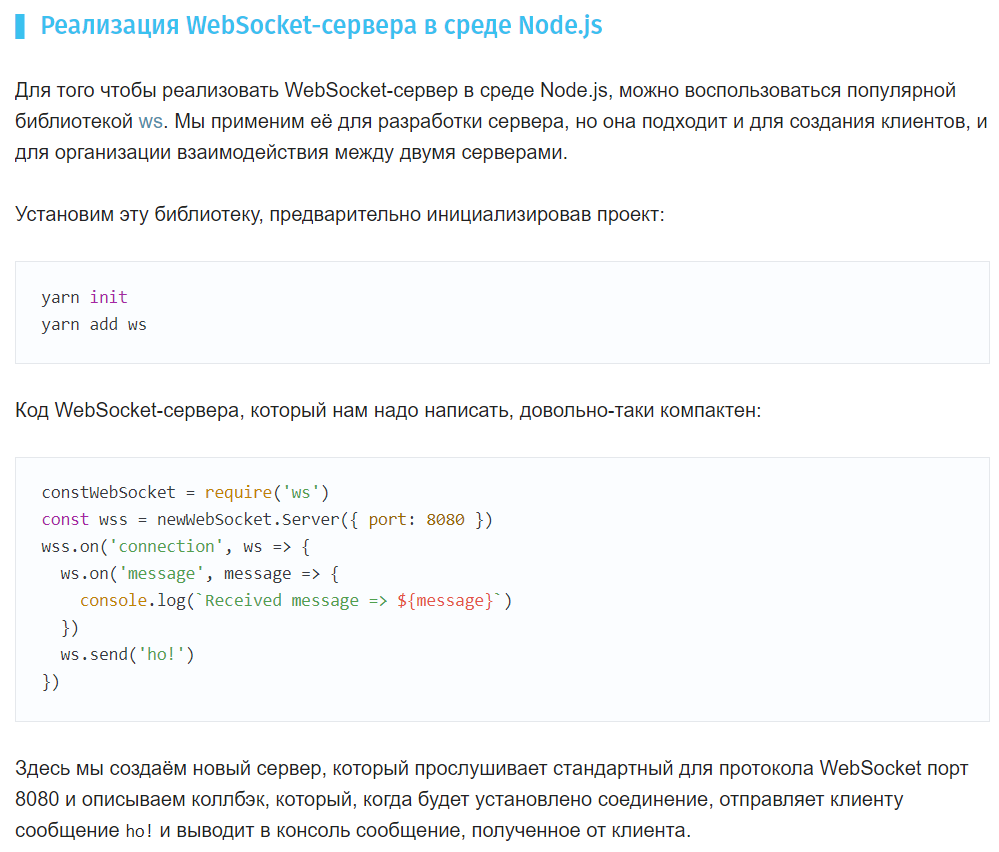
Протокол WebSocket очень хорошо подходит для организации связи в режиме реального времени по каналам, которые долго остаются открытыми. HTTP, в свою очередь, отлично подходит для организации эпизодических сеансов связи, инициируемых клиентом. В то же время надо отметить, что, с точки зрения программирования, реализовать обмен данными по протоколу HTTP гораздо проще, чем по протоколу WebSocket.

Существует небезопасная версия протокола WebSocket (URI-схема ws://), которая напоминает, в плане защищённости, протокол http://. Использования ws:// следует избегать, отдавая предпочтение защищённой версии протокола — wss://.







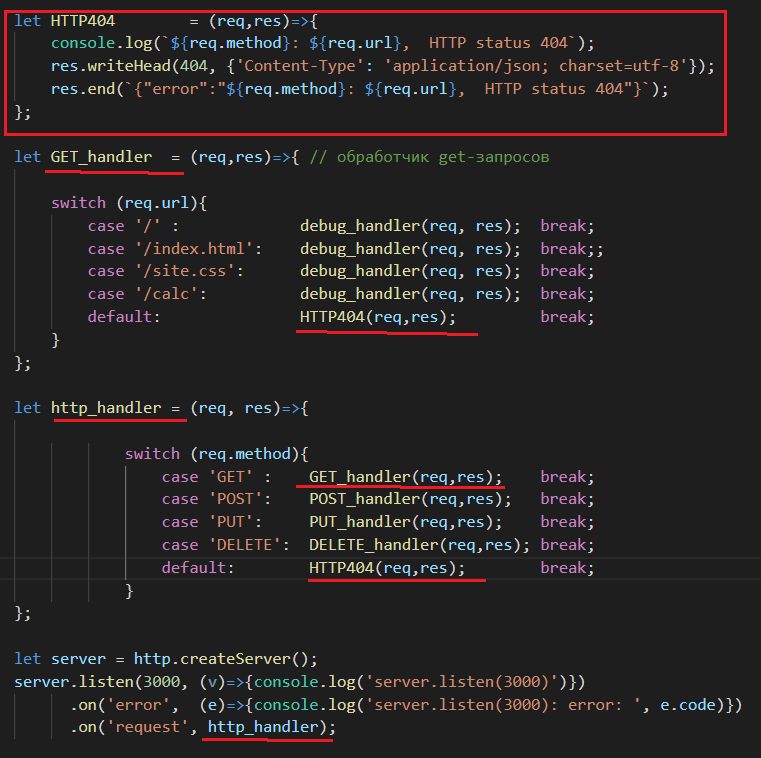


1. Разработка простейшего HTTP-сервера в Node.js. Извлечение данных из HTTP-запроса, формирование данных HTTP-ответа. Пример. Тестирование с помощью браузера AJAX (XMLHTTPRequest/Fetch).

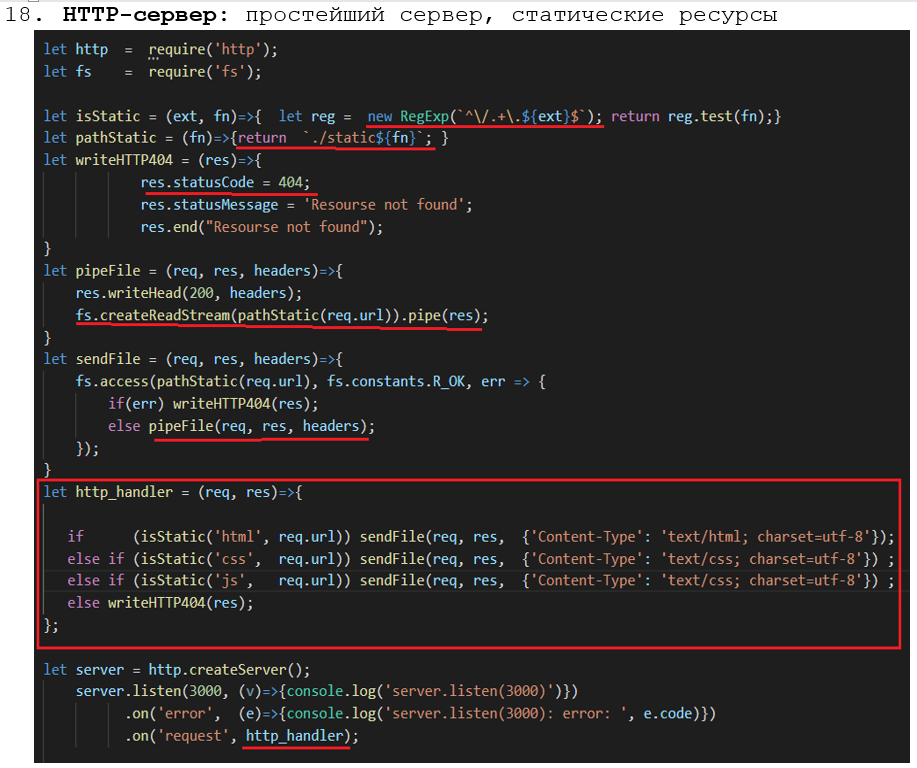


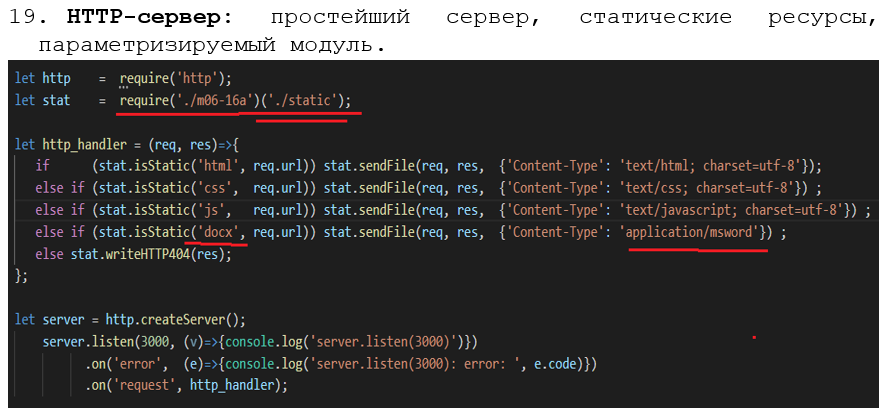


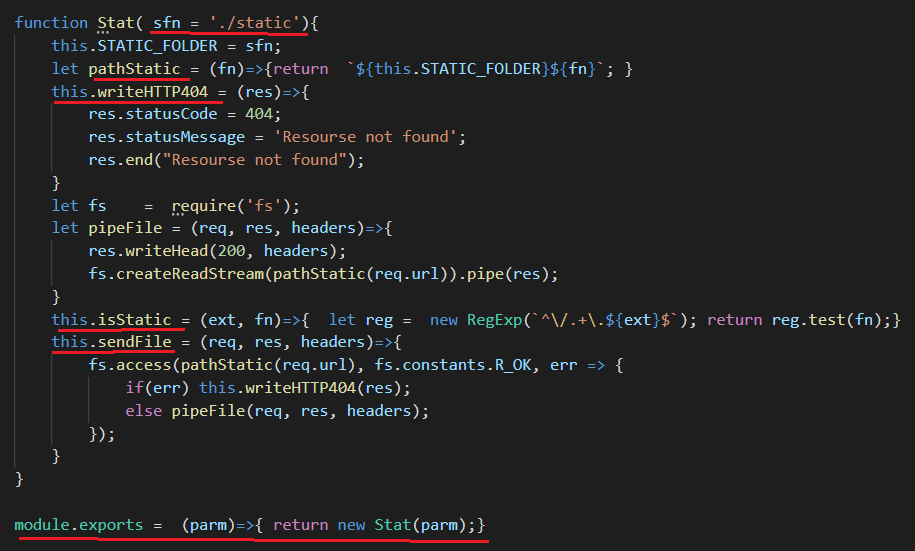
1. Разработка HTTP-сервера в Node.js. Обработка GET, POST, PUT и DELETE-запросов. Генерация ответа с кодом 404. Пример. Тестирование с помощью POSTMAN.

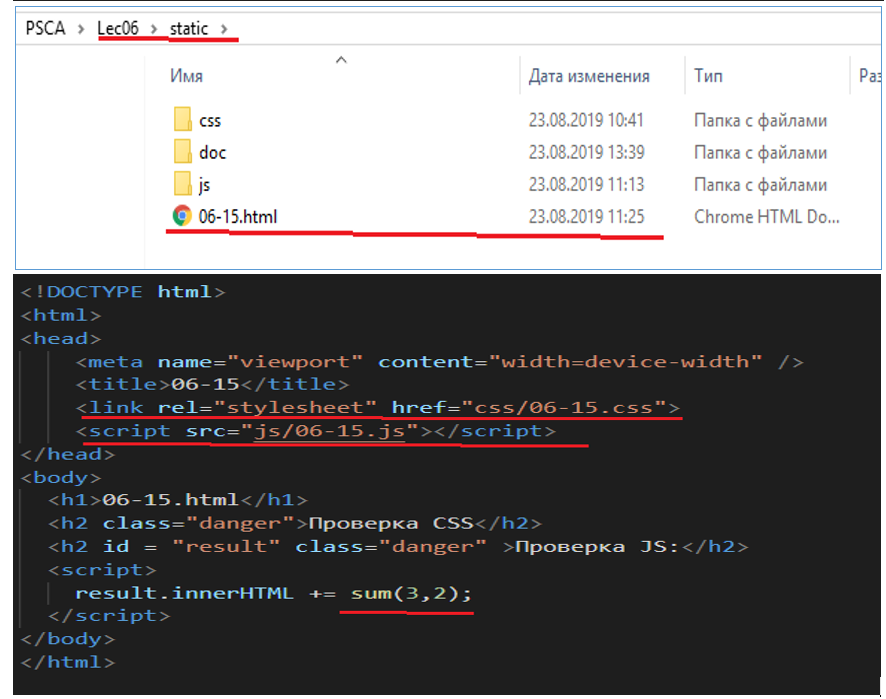
****

1. Разработка HTTP-сервера в Node.js. Обработка запросов к статическим ресурсам: html, css, js, png, msword. Пример. Тестирование с помощью браузера.





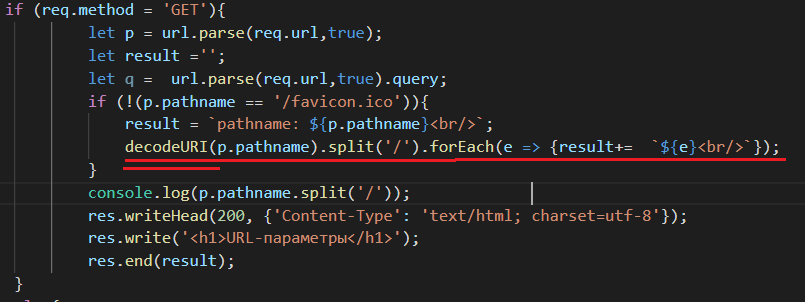
****



1. Разработка HTTP-сервера в Node.js. Обработка query-параметров GET-запроса. Пример. Тестирование с помощью браузера.



1. Разработка HTTP-сервера в Node.js. Обработка uri-параметров GET-запроса. Пример. Тестирование с помощью браузера.



1. Разработка HTTP-сервера в Node.js. Обработка параметров POST-запроса. Пример. Тестирование с помощью браузера (<form>) и POSTMAN.

****

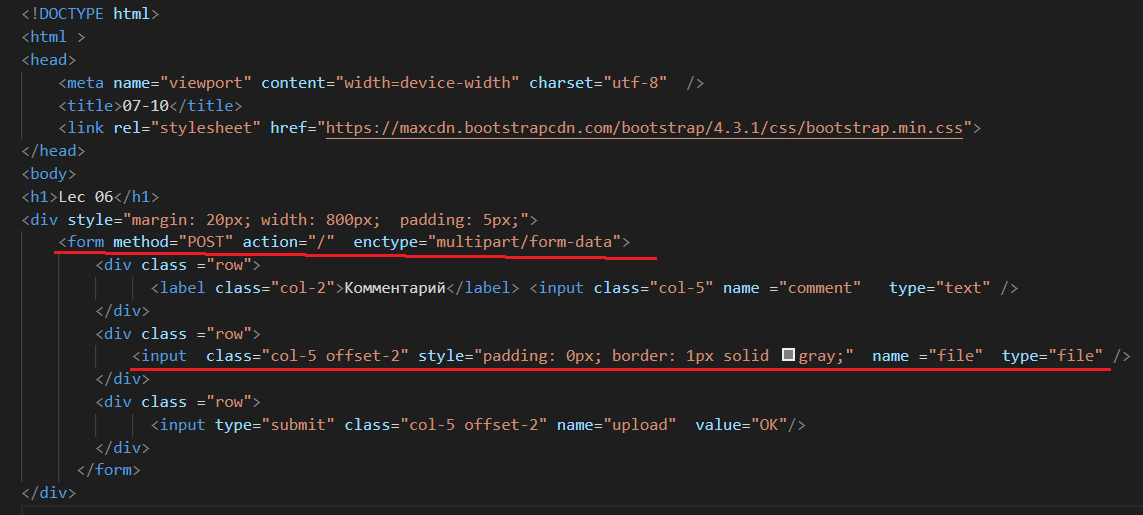
****

1. Разработка HTTP-сервера в Node.js. Обработка json-сообщения в POST-запросе. Пример. Тестирование с помощью POSTMAN.

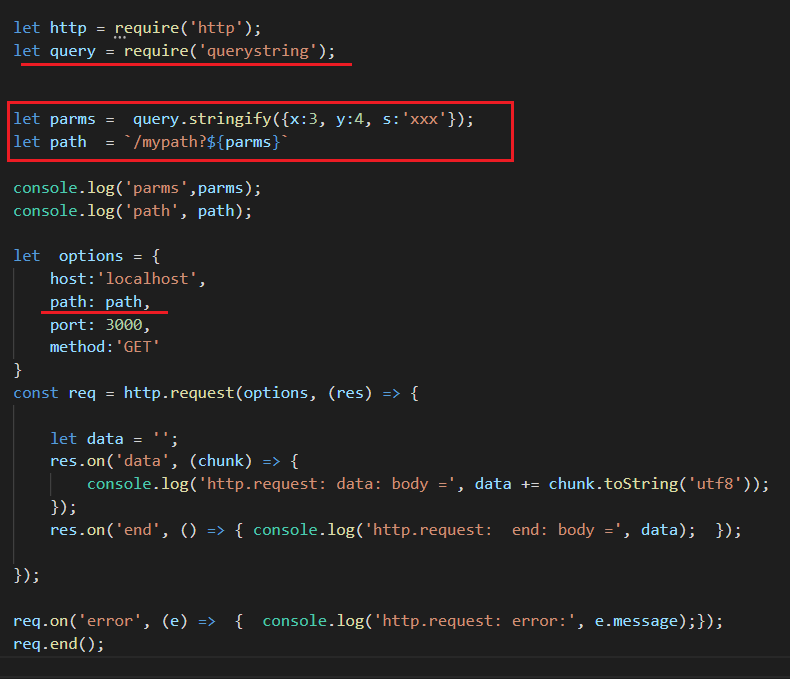
****

1. Разработка HTTP-сервера в Node.js. Пересылка файла в POST-запросе (upload). Пример. Тестирование с помощью браузера.





1. Разработка HTTP-сервера в Node.js. Пересылка файла в ответе (download). Пример. Тестирование с помощью браузера.
2. Разработка HTTP-клиента в Node.js. Оправка GET запроса с query-параметрами. Пример. Тестирование с помощью с Node.js-сервера.

****

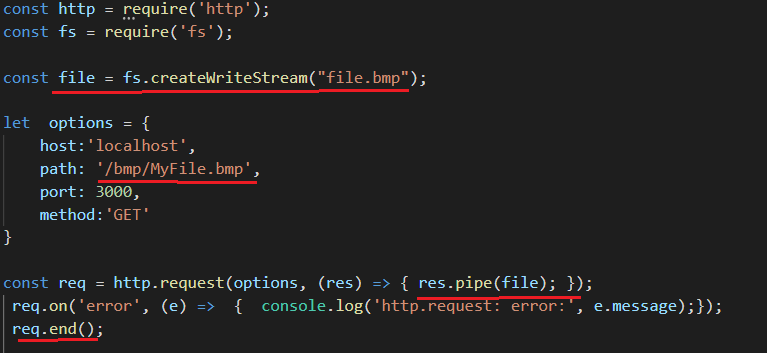
1. ****Разработка HTTP-клиента в Node.js. Оправка POST-запроса с параметрами в теле. Пример. Тестирование с помощью с Node.js-сервера.
2. Разработка HTTP-клиента в Node.js. Оправка POST-запроса с json-сообщением. Пример. Тестирование с помощью с Node.js-сервера.



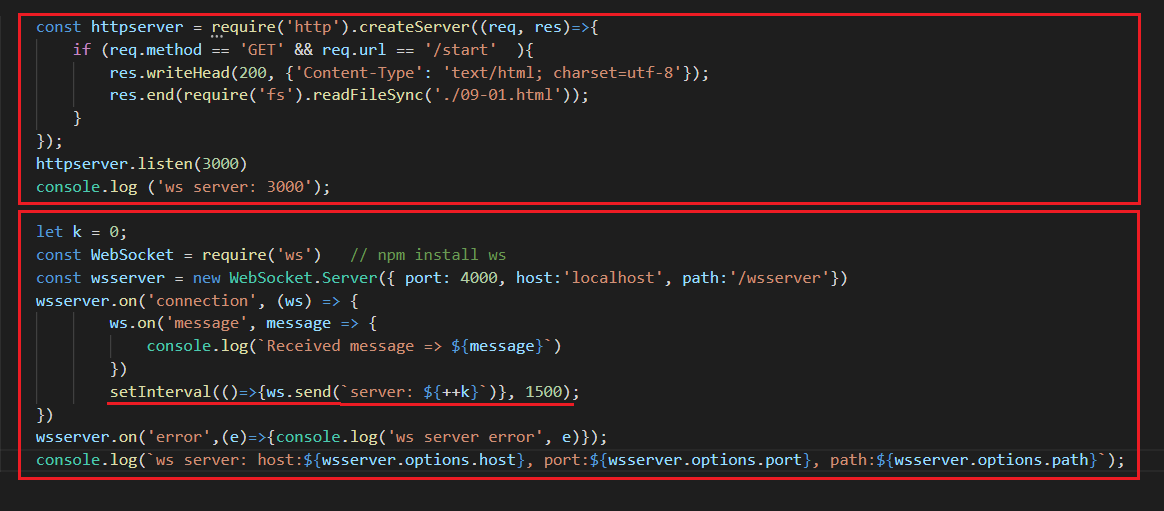
1. Разработка HTTP-клиента в Node.js. Обработка json-ответа. Пример. Тестирование с помощью с Node.js-сервера.
2. Разработка HTTP-клиента в Node.js. Пересылка файла на сервер в POST-запросе (upload). Пример. Тестирование с помощью с Node.js-сервера.

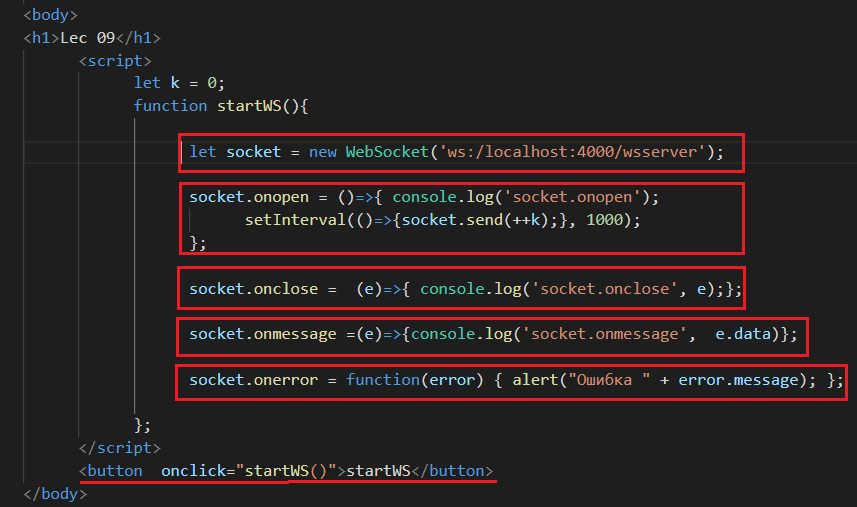
****

1. Разработка HTTP-клиента в Node.js. Обработка ответа с файлом (download). Пример. Тестирование с помощью с Node.js-сервера.

****

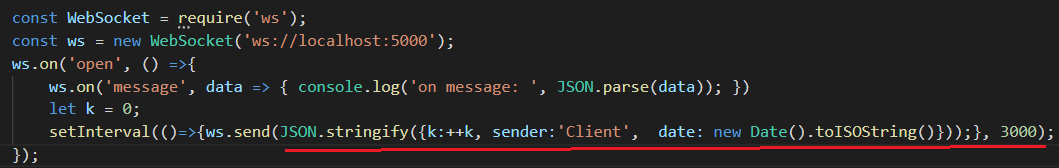
1. Разработка Websockets-приложения: Node.js-сервер, браузер-клиент. Пример.





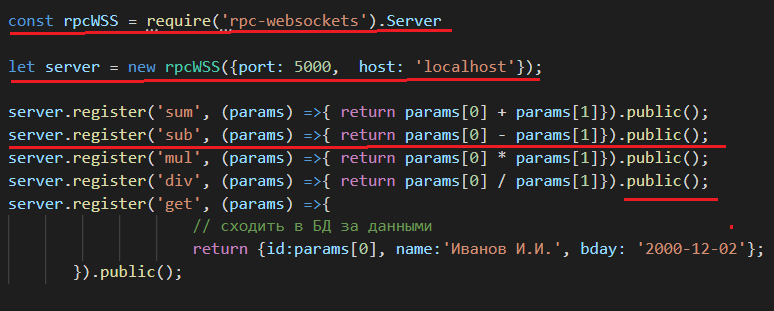
1. Разработка Websockets-приложения: обработка json-сообщений, Node.js-сервер, Node.js-клиент. Пример.

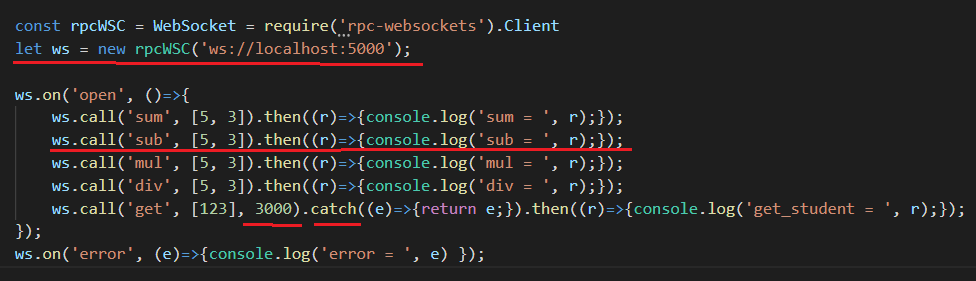
****

****

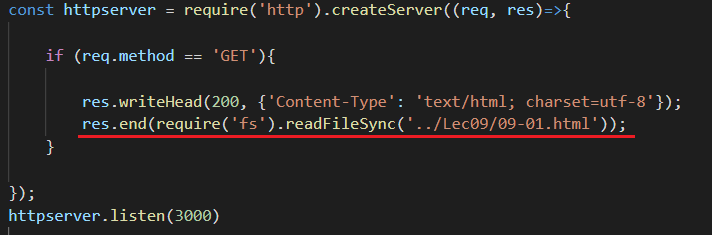
1. Разработка RPC-Websockets-сервера. Пример. Тестирование: Node.js-клиент.

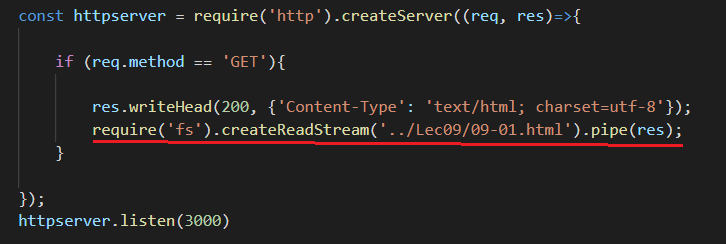
npm install rpc-websockets

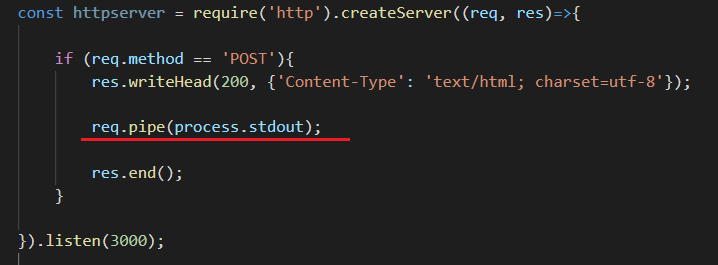




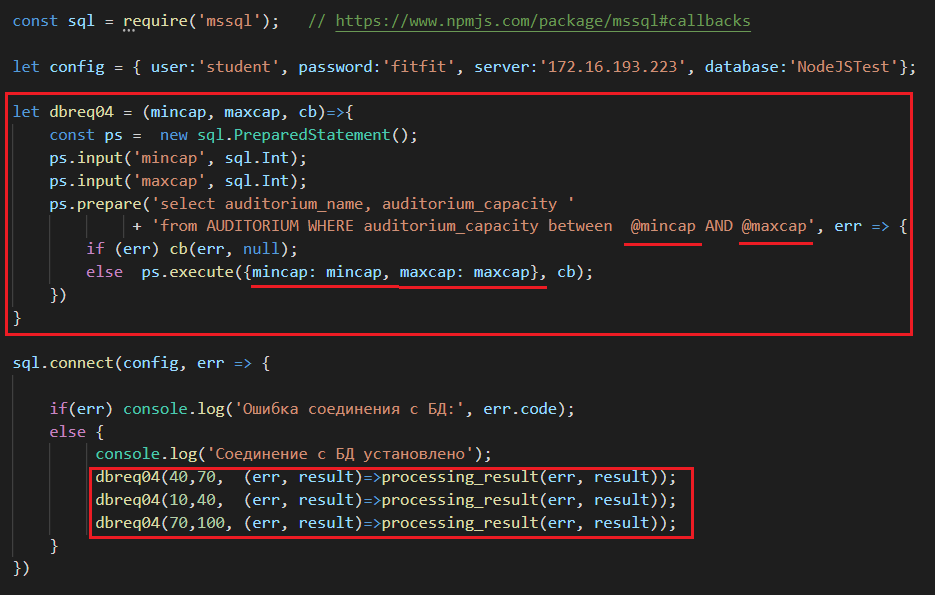
1. Применение функции pipe для обработки данных (файла) файловой системы и записи в http-ответ. Пример.

****

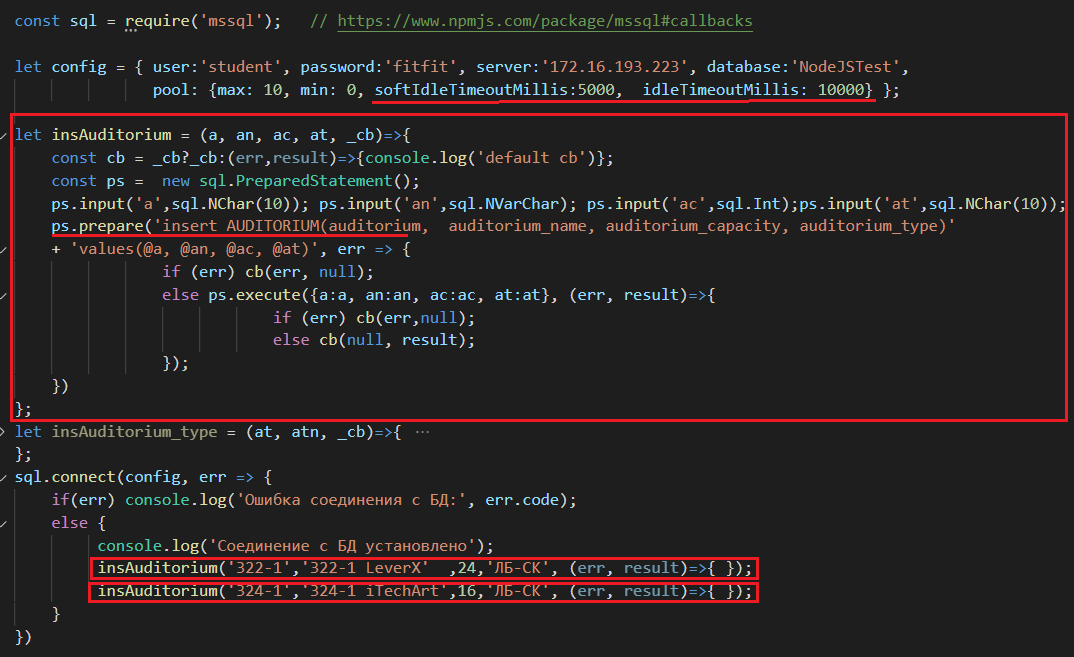
****

****

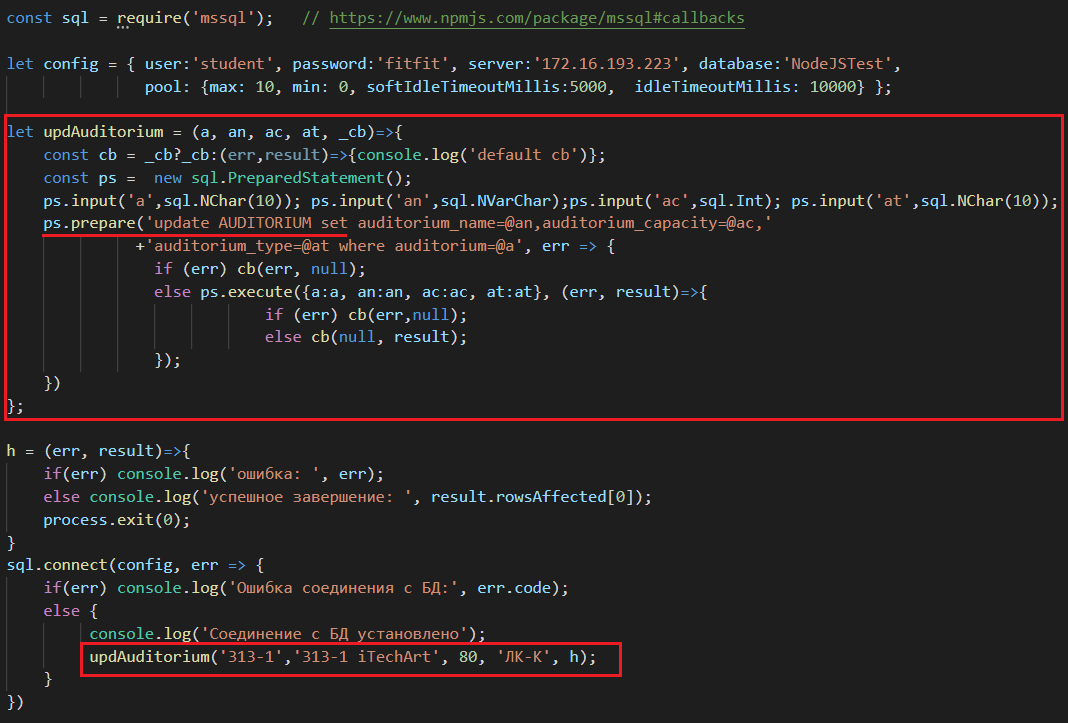
1. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического SELECT-запроса.



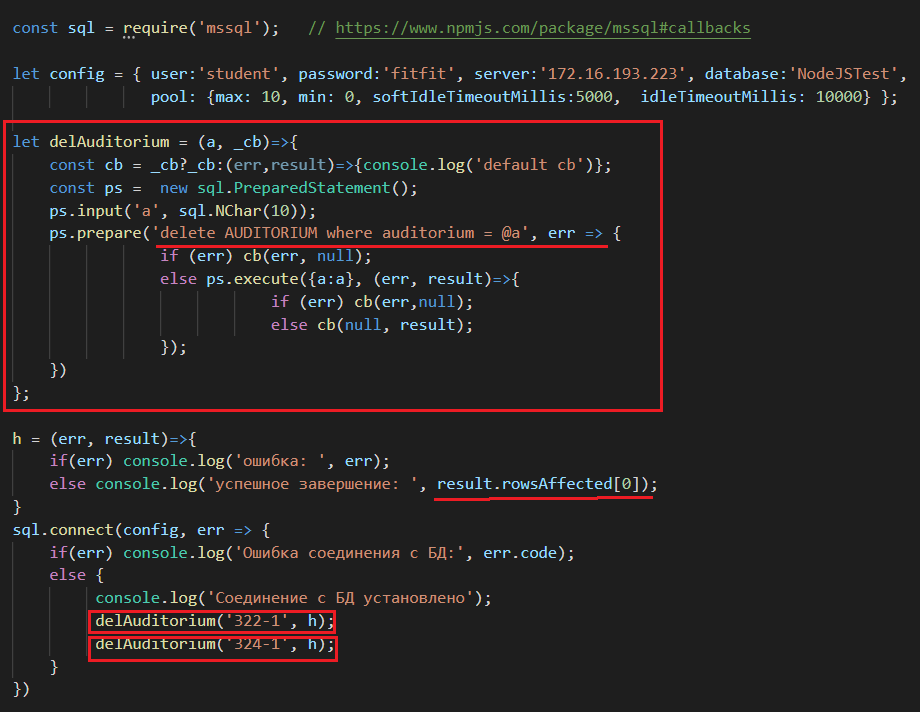
1. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического INSERT-запроса. Пример.



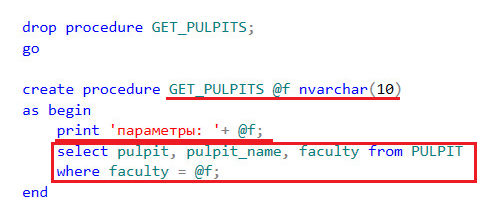
1. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического UPDATE-запроса. Пример.

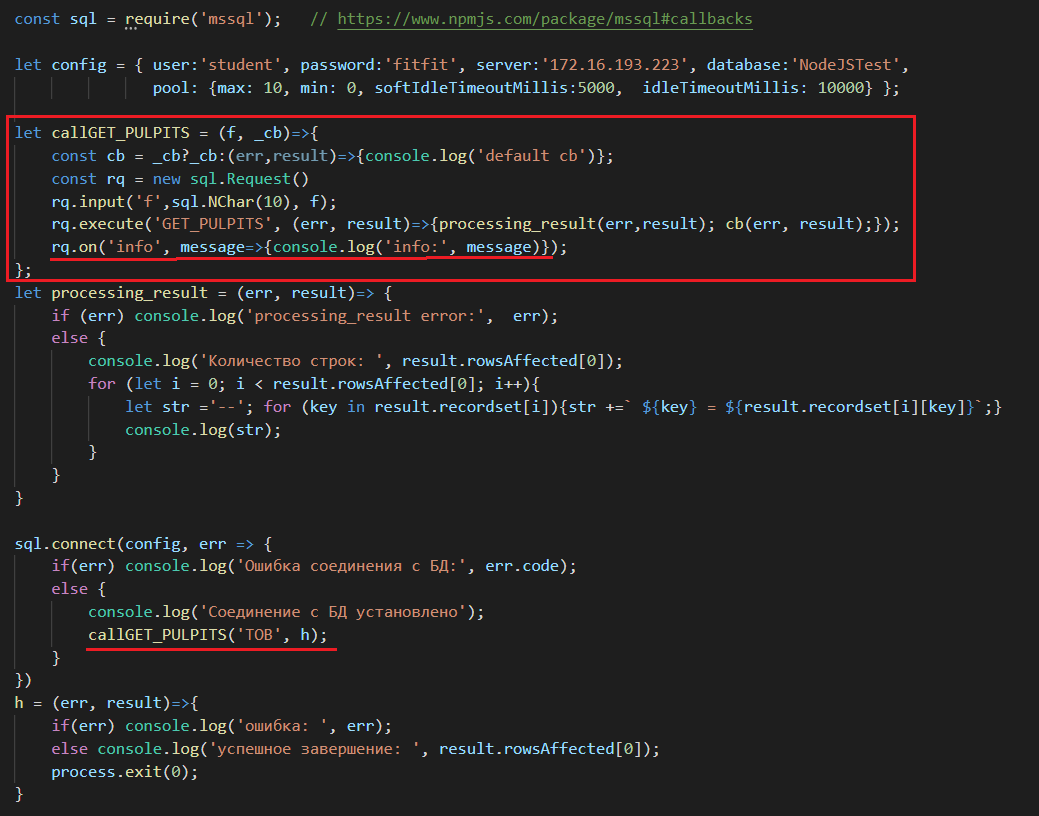


1. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического DELETE-запроса. Пример.



1. Разработка приложения, выполняющего запрос к SQL-базе данных: вызов удаленной процедуры. Пример.





1. Применение СУБД Redis. Основные принципы работы. Пример.

**Redis:** noSQL СУБД с открытым кодом (BSD-лицензия), Redis Labs, Сальваторе Санфилиппо:

хранилище данных в оперативной памяти;

для кэша;

для посредника сообщений;

структуры данных: строки, хэш-таблицы, списки, наборы, отсортированные наборы, растровые изображения, геопространственные индексы, HyperLogLog;

СУБД ориентирована на быстрое выполнение атомарных операций (до 100тыс. set/get-операций);

механизм снимков для асинхронного сохранения (с потерями);

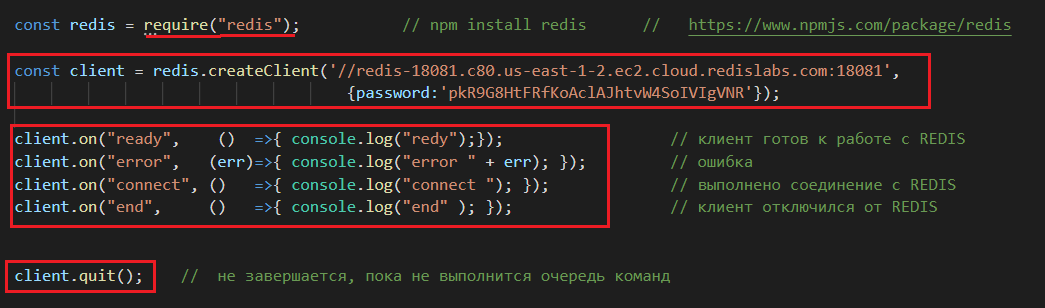
механизм упреждающей записи;

написана на ANSI С;

последняя стабильная версия: 5**;**

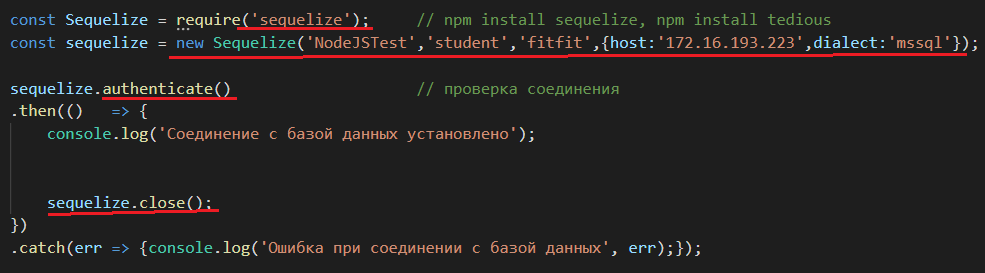
API: C, C++, C#, Java, JavaScript, Python, … ;

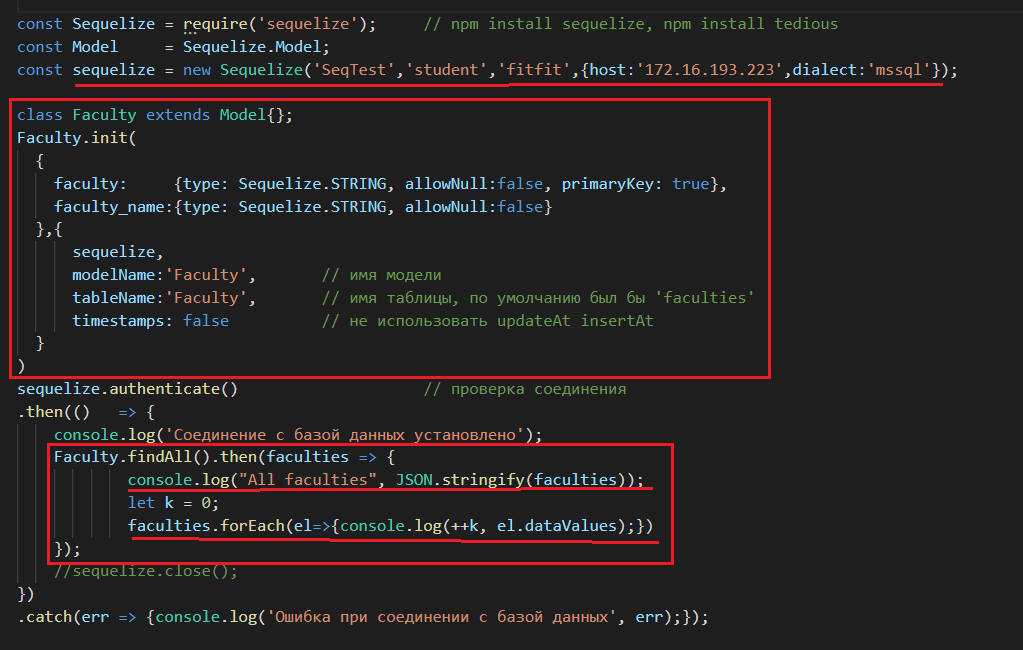
работает только под Linux.

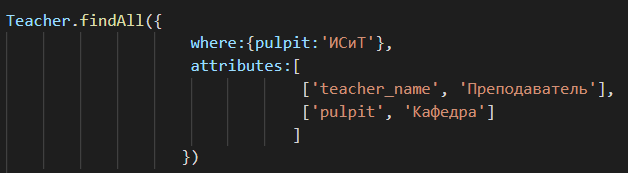
****Set, Get, incr, incrby, decr, decrby, getset, hset, hget, hlen, hdel, hexists, hgetall, hvals,hmset, hmget, hincrby, hincrbyfloat, publish, subscribe

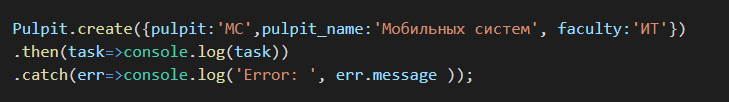
1. Применение пакета Sequelize. Основные принципы работы. Пример.

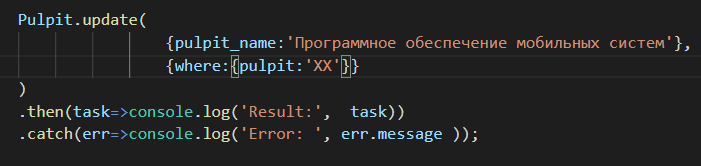
**ORM: Object-Relational Mapping**  - технология программирования, которая позволяет работать с SQL-базой данных, как с набором программных объектов. **Mapping**: база данных – объект contextDB, таблица – коллекция объектов, строка в таблице – объект, структура таблицы – класс.

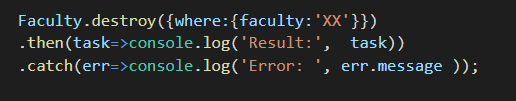
****

****

****

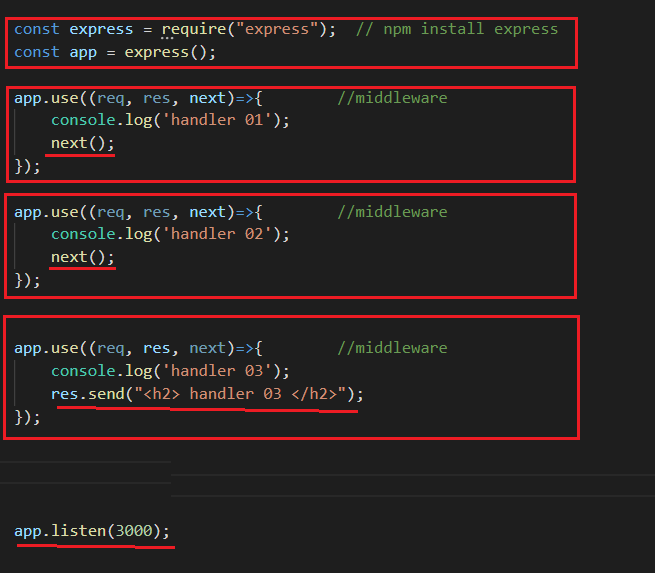
****

****

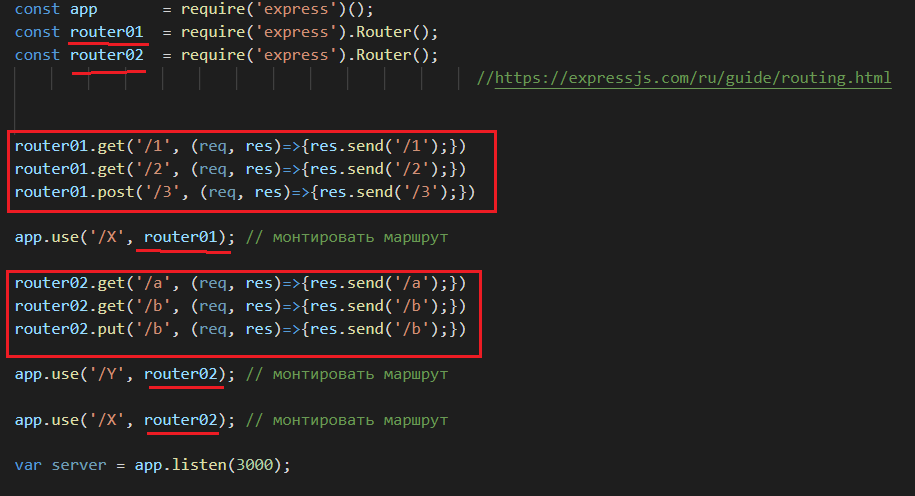
****

****

1. Пакет Express. Основные принципы работы. Middleware-код. Пример.

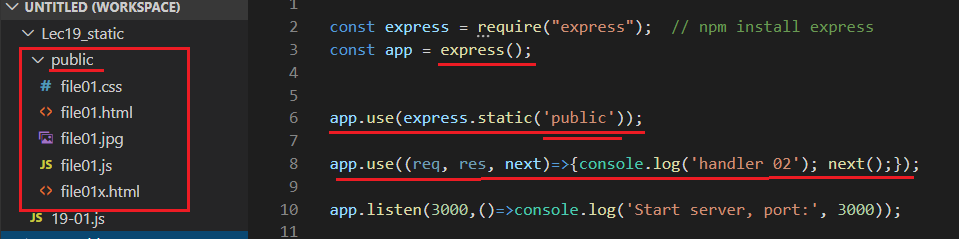


1. Пакет Express. Основные принципы работы. Маршрутизация. Пример.

****

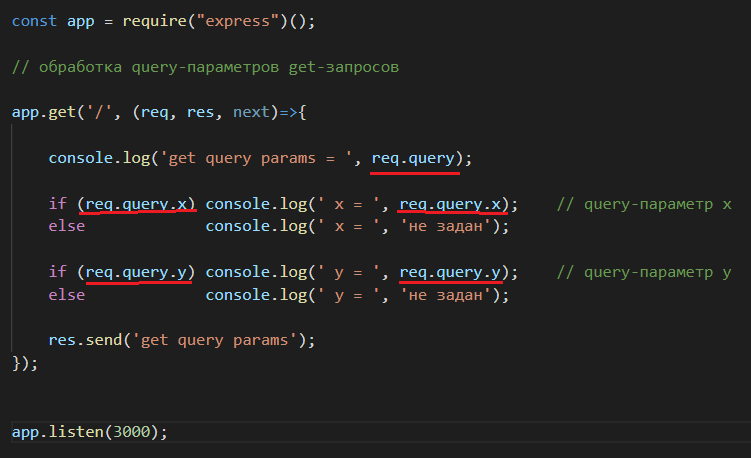
1. Пакет Express. Основные принципы работы. Статические файлы. Пример.

**Статические файлы:** файлы расположенные на стороне сервера и предназначенные для считывания их без изменения с помощью HTTP GET-запроса по имени ресурса, включающего имя файла.

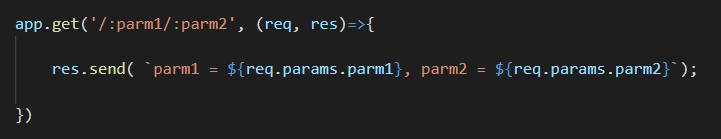


**Attachment –** указание http-клиенту не отображать содержимое body, а сохранить в файловой структуре.

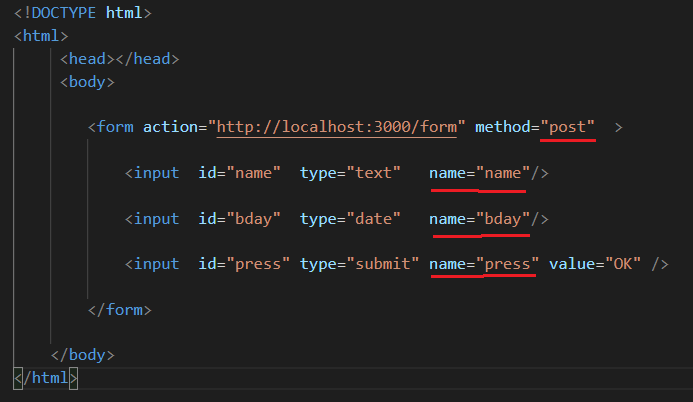
1. Пакет Express. Основные принципы работы. Обработка query-параметров GET-запроса. Пример (POSTMAN).

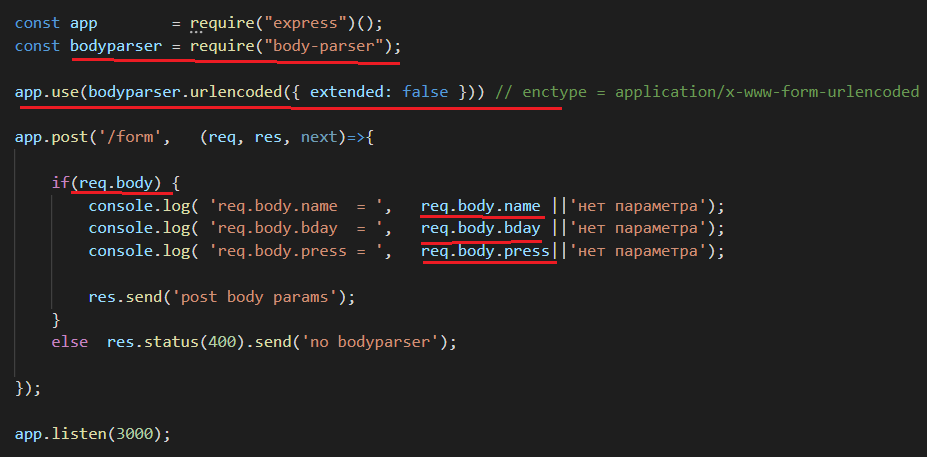


1. Пакет Express. Основные принципы работы. Обработка uri-параметров запроса. Пример (POSTMAN).

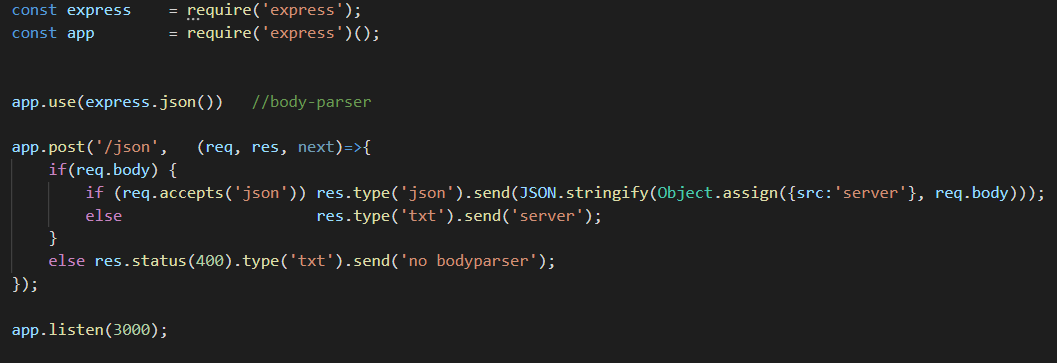
****

1. Пакет Express. Основные принципы работы. Обработка body-параметров POST-запроса. Пример (POSTMAN).

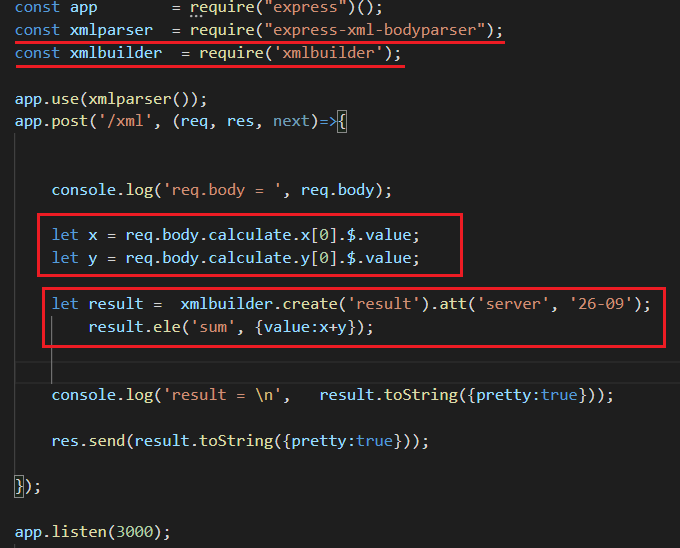
****

****

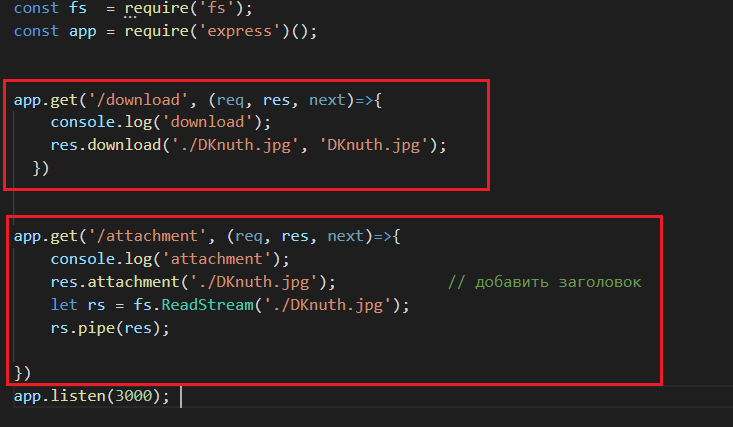
1. Пакет Express. Основные принципы работы. Обработка json-данных POST-запроса. Пример (POSTMAN).

****

1. Пакет Express. Основные принципы работы. Обработка xml-данных POST-запроса. Пример (POSTMAN).

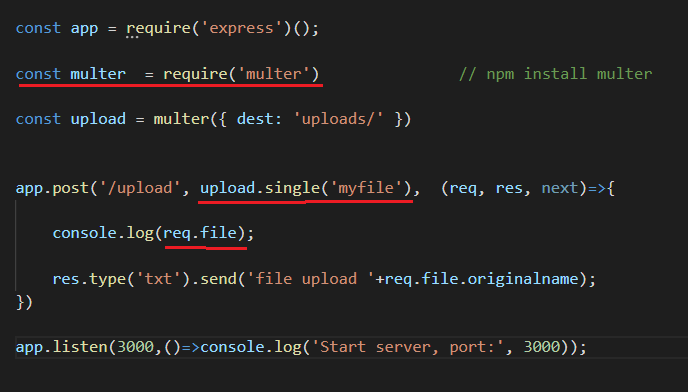
****

1. Пакет Express. Основные принципы работы. download/attachment файлы GET-запроса. Пример (браузер).

****

1. Пакет Express. Основные принципы работы. upload файла в POST-запросе. Пример (браузер).





1. Пакет Express. Основные принципы работы. Обработка Cookie. Signed cookie. Пример(POSTMAN).

**Cookie:** фрагмент данных, хранится на http-клиент, создается по инициативе сервера (заголовок Set-Cookie), пересылается http-клиентом (заголовок Cookie), http-клиент может отказаться от создания cookie, http-клиент может удалить cookie, содержимое cookie доступно, с помощью JS можно изменить cookie, один из методов XSS-атаки (cross-site scripting) основана на подмене cookie, применение cookie надо избегать.

**Cookie:** опция **domain** – привязка cookie к поддомену.

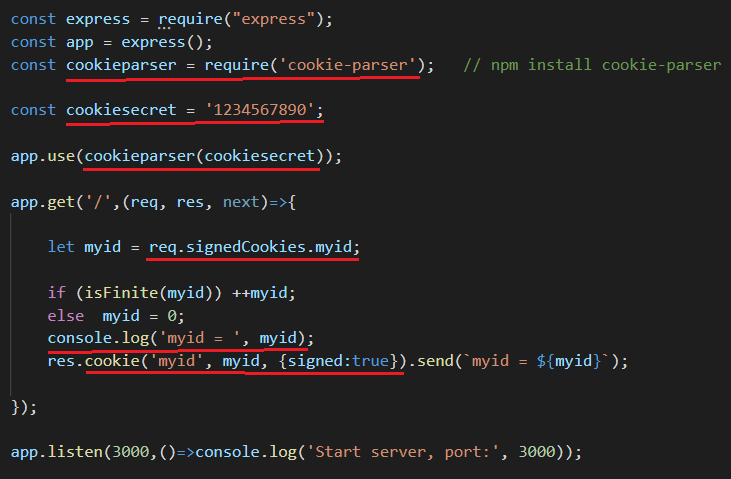
**Cookie:** опция **path** – путь, на который распространяется действие cookie.

**Cookie:** опция **maxAge** –время жизни cookie в миллисекундах.

**Cookie:** опция **Expires** –дата истечения жизни cookie.

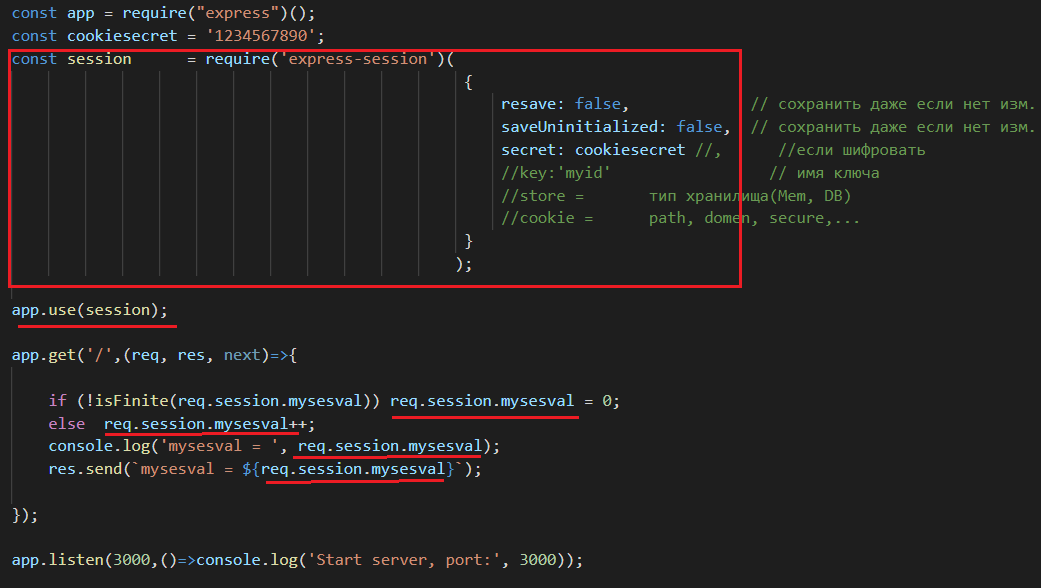
**Cookie:** опция **secure** –может применяться только с HTTPS.

**Cookie:** опция **httpOnly = true** –может изменяться сервером.

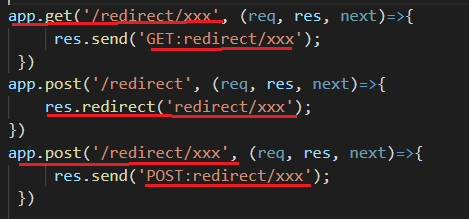


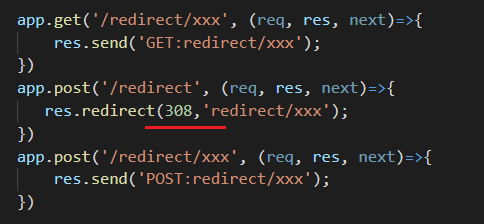
1. Пакет Express. Основные принципы работы. Применение объекта Session для сохранение состояния. Пример (POSTMAN).

**Session:** серверный объект, хранящий информацию о соединении с клиентом, создается при первом обращении время жизни: **timeout** (системный параметр, обычно равен 10 – 30 минутам) – максимальное время между запросами клиента. Если **timeout** превышен, то Session разрушается и при следующем запросе создается новый экземпляр. Каждая сессия имеет собственный идентификатор (**Session ID**, 16 или более байт). Каждый Requestпринадлежит, какой-то сессии (имеет ссылку на объект Session или содержит Session ID). Обычно объект Session предоставляет приложению возможность хранить данные в формате ключ/значение.

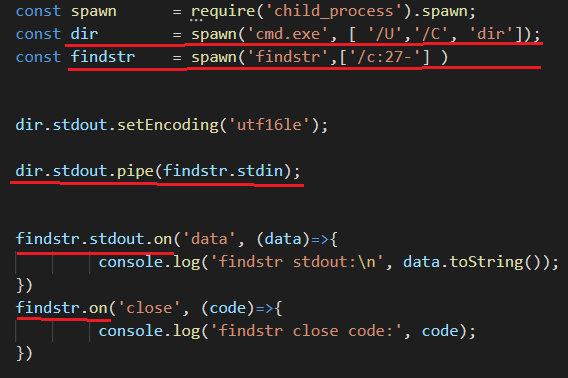
****

1. Пакет Express. Основные принципы работы. Переадресация. Пример(POSTMAN).





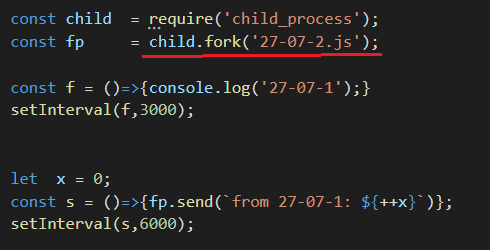
1. Пакет Express. Основные принципы работы. Выполнение shell-команд (spawn, pipe). Пример.

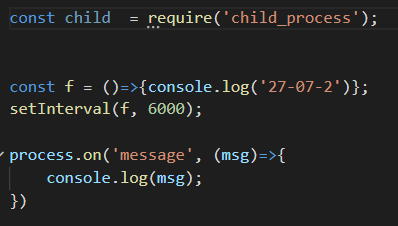


1. Пакет Express. Основные принципы работы. Запуск процесса операционной системы (exec), работа со стандартными потоками ввода/вывода. Пример.



1. Пакет Express. Основные принципы работы. Выполнение js-скриптов в отдельном процессе (fork, send, worker). Пример.

****

****

1. Протокол WebDav. Разработка приложения с применением WebDav. Пример(лабораторная работа).

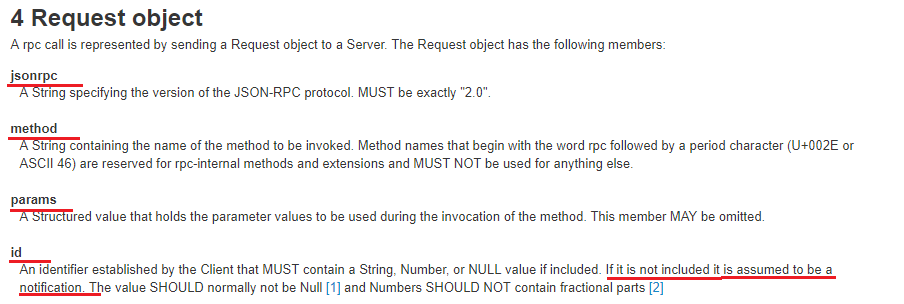
Web Distributed Authoring and Versioning – расширение протокола HTTP/HTTPS, поддерживающее совместную работу по управление файлами на удаленных web-северах; применяется для создания сетевой файловой системы; в системах документооборота (document management system).

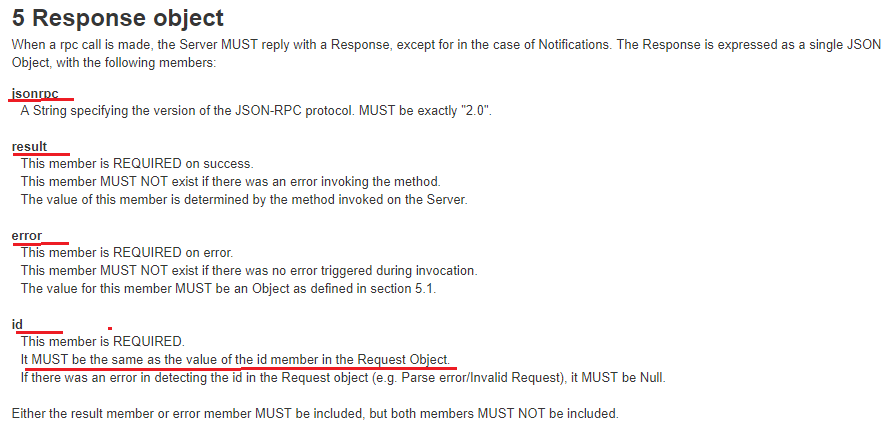
**WebDAV** расширяет HTTP следующими методами запроса:

* PROPFIND — получение свойств объекта на сервере в формате [XML](https://ru.wikipedia.org/wiki/XML). Также можно получать структуру [репозитория](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%B9" \o "Репозиторий) (дерево каталогов);
* PROPPATCH — изменение свойств за одну транзакцию;
* MKCOL — создать коллекцию объектов (каталог в случае доступа к файлам);
* COPY — копирование из одного [URI](https://ru.wikipedia.org/wiki/URI) в другой;
* MOVE — перемещение из одного [URI](https://ru.wikipedia.org/wiki/URI) в другой;
* LOCK — поставить блокировку на объекте. WebDAV поддерживает эксклюзивные и общие (shared) блокировки;
* UNLOCK — снять блокировку с ресурса.

1. Протокол JSON-RPC. Разработка клиент-серверное приложение использующее протокол JSON-RPC.

Протокол удаленного вызова процедур, использующий формат JSON для передачи сообщений

****

****

1. Разработка клиент-серверного приложения с применением технологии WebAssembly на стороне браузера. Пример(WasmFiddle-компиляция).

**WebAssembly** –бинарный формат исполняемого файла, который может исполняться в JavaScript Engine (виртуальная стековая машина). Код быстрее, чем JS; поддерживается большинством браузеров; выполняется в sandbox; есть отладчики; открытый стандарт.

1. Разработка клиент-серверного приложения с применением технологии WebAssembly на стороне сервера Node.js. Пример(WasmFiddle-компиляция).
2. Long pool–сервер, принцип работы. Пример (Telegram bot, лабораторная работа).

**3ый вопрос в билете (лабораторные работы)**

1. Лабораторная 17 (REDIS).
2. Лабораторная 18 (SEQUELIZE).
3. Лабораторная 20 (HBS).
4. Лабораторная 21 (Basic, Digest, Forms).
5. Лабораторная 22 (HTTPS).
6. Лабораторная 23 (CRYPTO).
7. Лабораторная 24 (WEBDAV).
8. Лабораторная 25 (JSONRPC).
9. Лабораторная 26 (WASM).
10. Лабораторная 27 (TLGBOT).