

Задание 00

1. Разработайте приложение (сервер) **08-00** предназначенное для обработки следующих запросов.

HTTP-метод	URI	Задание
GET	/connection?set= set	01
GET	/headers	02
GET	/parameter?x= x &y= y	03
GET	/parameter/ x/y	04
GET	/close	05
GET	/socket	06
GET	/req-data	07
GET	/resp-status?code= c ?mess= m	08
POST	/formparameter	09
POST	/json	10
POST	/xml	11
GET	/files	12
GET	/files/ filename	13
GET/POST	/upload	14

Задание 01 /connection?set=**set**

- При GET-запросе **/connection** в окно браузера вывести текущее значение системного параметра **KeepAliveTimeout**.
- При GET-запросе **/connection/set=set** установить новое значение системного параметра **KeepAliveTimeout = set** и вывести в окно браузера сообщение установлено новое значение параметра **KeepAliveTimeout=set**.
- Продемонстрируйте влияние системного параметра **KeepAliveTimeout** на работу приложения.

Задание 02 /headers

- Отобразите в окне браузера все заголовки запроса и ответа.
- Сформируйте собственный пользовательский заголовок ответа.

7. Убедитесь, что созданный пользовательский заголовок ответа, доставлен клиенту (с помощью браузера или POSTMAN).
8. Объясните назначение каждого заголовка.

Задание 03 /parameter?x=**x**&y=**y**

9. Проанализируйте значения параметров **x** и **y**.
10. Если **x** и **y** имеют числовые значения, то выведите в окно браузера сумму, разность, произведение и частное этих чисел
11. Иначе выведите сообщение об ошибке.

Задание 04 /parameter/**x/y**

12. Проанализируйте значения параметров **x** и **y**.
13. Если **x** и **y** имеют числовые значения, то выведите в окно браузера сумму, разность, произведение и частное этих чисел.
14. Иначе выведите **URI**.

Задание 05 /close

15. При получении этого запроса, в окно браузера выведите сообщение о закрытии сервера и остановите сервер через 10 секунд.

Задание 06 /socket

16. При получении этого запроса, в окно браузера выведите ip- адрес, порт клиента и ip-адрес и порт сервера.

Задание 07 /req-data

17. Продемонстрируйте в этом запросе, подобрав достаточно длинное сообщение в POSTMAN, порционную обработку запроса.

Задание 08 `resp-status?code=c?mess=m`

18. При получении этого запроса, сформируйте ответ, имеющий статус, заданный значением **c** и пояснение к статусу, заданное значением **m**.

Задание 09 `/formparameter`

19. Используя HTML-форму включающую теги **input** с **type: text, number, date, checkbox, radiobutton**, тег **textarea**, а также два тега **input type=submit**, имеющих одно и тоже имя, но разные значения.
20. В окно браузера выведите значения параметров, полученных в запросе.

Задание 10 `/json`

21. Принимаются POST-запросы, содержащие данные в json-формате и отправляет ответы в json-формате.
22. Сообщение в запросе имеет следующую структуру:

```
{
  "__comment": " Запрос.Лабораторная работа 8/10",
  "x": 1,
  "y": 2,
  "s": "Сообщение",
  "m":["a","b","c","d"],
  "o":{"surname":"Иванов", "name":"Иван"}
}
```

23. Сообщение в ответе имеет следующую структуру:

```
{
  "__comment": " Ответ.Лабораторная работа 8/10",
  "x_plus_y": 3,
  "Concatination_s_o": "Соообщение: Иванов, Иван",
  "Length_m": 4
}
```

Поле **x+y** ответа содержит сумму полей **x** и **y** запроса.

Поле **Concatination_s_o** ответа конкатенация полей **s** и свойств объекта **o** запроса.

Поле **Length_m** ответа содержит количество элементов в массиве **m** запроса.

24. Проверьте работоспособность приложения с помощью POSTMAN.

Задание 11. /xml

25. Принимаются POST-запросы, содержащие данные в xml-формате и отправляет ответы в xml-формате.
26. Сообщение в запросе имеет следующую структуру.

```
<request id = "28">
  <x value = "1"/>
  <x value = "2"/>
  <m value = "a"/>
  <m value = "b"/>
  <m value = "c"/>
</request>
```

27. Количество элементов **x** и **m** в запросе может быть произвольным.
28. Сообщение в ответе имеет следующую структуру.

```
<response id="33" request="28">                                <!-- ответ на запрос 28 -->
  <sum element = "x" result="3" />                                <!-- сумма значений элементов x -->
  <concat element = "m" result = "abc" />                        <!-- конкатенация всех элементов m -->
</response>
```

29. Элемент **sum** в ответе один и содержит сумму всех значений элементов **x** (в атрибуте **value**).
30. Элемент **concat** в ответе один и содержит сумму всех значений элементов **m** (в атрибуте **value**).
31. Проверьте работоспособность приложения с помощью POSTMAN.

Задание 12. /files

32. В ответ на запрос высылается ответ с заголовком **X-static-files-count: n**, где **n** – количество файлов в директории **static**. Используйте функции модуля **fs**.
33. Проверьте работоспособность приложения с помощью POSTMAN.

Задание 13. /files/**filename**

34. В ответ на запрос высылается ответ, пересылающий файл с именем **filename** из директории **static**.
35. Если файл **filename** не найден возвращается ответ со статусом 404.
36. Проверьте работоспособность приложения с помощью браузера.

Задание 14. /upload

37. В ответ на GET-запрос к **/upload** высылается web-форма позволяющая отправить POST-запрос к **/upload**, пересылающий серверу файл.
38. Сервер сохраняет файл в директории **static**.
39. Проверьте работоспособность приложения с помощью браузера.

Задание 15. Ответьте на следующие вопросы.

40. Поясните назначение заголовка **Content-Type**.
41. Поясните назначение заголовка **Accept**.
42. Для чего используется значение **Multipart/form-data** заголовка **Content-Type**.
43. Как с помощью тега **form**, обеспечить значение **Multipart/form-data** заголовка **Content-Type**.
44. Какое значение заголовка **Content-Type** отправляется тегом **form** в запросе по умолчанию.
45. Где и в каком формате передаются параметры в GET-запросе?
46. Где и в каком формате передаются параметры в POST-запросе?
47. Поясните понятие **JSON**?

48. Поясните понятие **XML**?