# Лабораторная работа № 1
# Изучение протокола HTTP
# Задание

Данная работа является вводной в курсе и призвана ознакомить студента с базовым протоколом web – протоколом HTTP.
*Примечание: под заданием вы найдете подсказки и необходимый теоретический материал.

**Результатом работы является отчет в формате MD и код сервера, выложенные в репозиторий.**

1. Базовая часть работы
   1.1. Цель данной работы – ознакомится с применением протокола HTTP на практике, в реальных системах. Каждый из рассмотренных типов запросов предлагается отправить на несколько известных интернет-сервисов. Впрочем, сервисы указаны лишь как примеры и при желании вы можете выбрать другие (социальные сети, почта, облака, новостные сайты и т.д.).
   1.2. С помощью специального ПО (Postman, либо многочисленные аналоги, например, Restlet Clent - расширение для Chrome) вручную отправить следующие запросы и ответить на предлагаемые вопросы.

   1.2.1. Запрос OPTIONS. Отправьте запрос на http://mail.ru, http://ya.ru, www.rambler.ru, https://www.google.ru, https://github.com/, www.apple.com/.

   Для чего используется запрос OPTIONS? Какие коды ответов приходят при этом запросе? Какие сайты правильно обработали запрос и вернули ожидаемые данные?

   1.2.2. Запрос HEAD. vk.com, www.apple.com, www.msn.com.

   Для чего нужен запрос HEAD? Какой сайт прислал ожидаемый ответ?

1.2.3. Запросы GET и POST. Отправьте по запросу на yandex.ru, google.com и apple.com. Что они вернули? Что содержится в теле ответа?

1.3. Работа с api сайта. Многие крупные сервисы предоставляют открытое api. Как правило, оно реализовано на подходе REST, но это необязательно. Такое api используется сторонними сервисами и приложениями, которые хотят воспользоваться услугами предоставляющего такое api сервиса. Рассмотрим такое api на примере сайта vk.com (при желании можно выбрать другой подходящий сервис).

1.3.1. Зайдите на https://vk.com/dev/api_requests и посмотрите структуру запросов к данному api.

1.3.2. Используя документацию (https://vk.com/dev/methods) выполните следующие задания (обратите внимание, запросы нужно отправлять не из предложенной на сайте формы, а как в предыдущем задании):

1.3.2.1. Получите список всех факультетов МГТУ им. Н.Э.Баумана.

1.3.2.2. Получите свою аватарку.

1.3.2.3. Ответьте на вопросы: какой код ответа присылается от api? Что содержит тело ответа? В каком формате и какой кодировке содержаться данные? Какой веб-сервер отвечает на запросы? Какая версия протокола HTTP используется?

1.3.3. POST запросы проще отправлять с формы, встроенной в документацию api. Чтобы посмотреть, как выглядит запрос, можно воспользоваться панелью разработчика браузера (F12 в Chrome -> вкладка Network).

1.3.3.1. Отправьте запись на стену любому пользователю/группе и убедитесь, что она пришла.

1.3.3.2. Ответьте на вопрос: каким образом передаются данные от пользователя к серверу в POST-запросах?

2. Реализуйте небольшое серверное приложение, с использованием любого фреймворка. Лучшего всего для этой цели подойдет NodeJS: решение получится очень компактным и простым.

   Сервер должен содержать предоставлять некоторое API с поддержкой (GET, POST, DELETE, PUT, OPTION). Данные отправлять в формате json. Конкретное содержание запросов - на ваше усмотрение. Подключите фантазию. (Можно сделать простейший CRUD-сервис с хранением данных в RAM).

3. Доп. задание. Статика и маршрутизация.

3.1. Добавьте папку static (классическое название для статически раздаваемой папки).

3.2. В папке static создайте папки html и img.

3.3. В папке static/html создайте файл index.html со следующим содержанием (или любым другим):

```
<head></head>
<body>
    <h1>Hello, world!</h1>
    <img src=”/img/image.jpg”>
</body>
```

3.3. Настройте сервер так, чтобы при запросе из браузера отображалась эта страница.

3.4. Настройте routing (маршрутизацию) на вашем сервере. Например, чтобы путь /hack тоже отдавал файл index.html, а путь /, по умолчанию отдающий index, выдавал дополнительную страницу hack.html.

3.5. Переименуйте hack.html (содержащую теги html) в hack.txt. Что изменилось? Почему? Как сделать так, чтобы страница отображалась корректно?

# Теоретический материал

**Справка по HTTP:**

1. **Wikipedia (в целом, статья нормальная) - https://ru.wikipedia.org/wiki/HTTP**
2. **RFC по HTTP/1.1 https://tools.ietf.org/html/rfc2616**
3. **RFC по HTTP/2 https://tools.ietf.org/html/rfc7540**

Простейший http-сервер на NodeJS

## Простой сервер

```
/* Подключение модуля http. NodeJS имеет модульную структуру. */
var http = require('http');

/*Обращение к модулю - вызов метода createServer. */
http.createServer(function (req, res) {

/* В метод передается анонимная callback-функция для обработки запросов к серверу. В ней происходит формирование ответа: проставляется код 200 и строка Hello, world */
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('<h1>Hello World</h1>');

/* Сервер создается на 8080 порту (относится к диапазону пользовательских портов, поэтому может быть без опасений использован). И на локальной машине (localhost) */
}).listen(8080);

/*Вывод в консоль - запустились*/
```

```
console.log('Server running...');
```

## Простая отдача статических файлов по имени

```
var fs = require('fs');
var http = require('http');
var fileName = "page.html";
http.createServer(function (req, res) {
     fs.readFile(fileName, 'utf8', function(err, data)
{
          if (err){
               console.log('Could  not  find  or  open
file for reading\n');
          } else {
               res.writeHead(200,      {'Content-Type':
'text/html'});
               res.end(data);
          }
     })
}).listen(8080);

console.log('Server running on 8080');
```

## Express

```
$ npm install express –save
```

Express - популярный фреймворк для NodeJS для разработки веб-приложений.

## Простой сервер на express

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
    res.send('Hello World!');
});

var server = app.listen(8080, function () {
    var host = server.address().address;
    var port = server.address().port;
    console.log('Example app listening at http://%s:%s', host, port)
});
```

## Простая маршрутизация (по методам запроса и путям)

```
var express = require('express');
var app = express();

// получение GET запроса на главную страницу
app.get('/', function (req, res) {
```

```
  res.send('Got a Get request');
})

// получение POST запроса на главную страницу
app.post('/', function (req, res) {
  res.send('Got a POST request');
})

// получение PUT запроса по адресу /user
app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user');
})

// получение DELETE запроса по адресу /user
app.delete('/user', function (req, res) {
  res.send('Got a DELETE request at /user');
})

var server = app.listen(8080, function () {
     var host = server.address().address
     var port = server.address().port
     console.log('Example app listening at http://%s:%s', host, port)
});
```

## Пример обработчика запроса GET, возвращающего любой статический файл из корня

```
app.get('/:name', function (req, res, next) {
  var options = {
     root: './',
```

```
    dotfiles: 'deny',
    headers: {
        'x-timestamp': Date.now(),
        'x-sent': true
    }
  };

  var fileName = req.params.name;
  res.sendFile(fileName, options, function (err) {
    if (err) {
     console.log(err);
    res.status(err.status).end();
    }
    else {
     console.log('Sent:', fileName);
    }
  });
})
```

Для того, чтобы просто раздать статические файлы, можно включить раздачу интересующих каталогов:

```
app.use(express.static('static'));
```