

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»**

Институт радиоэлектроники и информационных технологий
Кафедра “Прикладная математика”

Отчет по лабораторной работе курса “Базы данных”

Лабораторная работа №4.2
«NodeJS и AJAX»

Выполнил студент группы 18-ПМ:

Винокуров М.С.

Проверил:

Моисеев А.Е.

НИЖНИЙ НОВГОРОД

2021 г.

Оглавление

Введение.....	3
AJAX	3
Задание	4
Выполнение работы.....	5
Добавление иного фильтра на основе «out»:Вывод.....	11
Список источников	13
Приложение. Листинг.....	14
GameDB-4-2.js	14
db4.2.html.....	16

Введение

AJAX

AJAX, Ajax (англ. Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.

Сравнение стандартного подхода и AJAX:

В классической модели веб-приложения:

- Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент.
- Браузер формирует и отправляет запрос серверу.
- В ответ сервер генерирует совершенно новую веб-страницу и отправляет её браузеру и т. д., после чего браузер полностью перезагружает всю страницу.

При использовании AJAX:

- Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент.
- Скрипт (на языке JavaScript) определяет, какая информация необходима для обновления страницы.
- Браузер отправляет соответствующий запрос на сервер.
- Сервер возвращает только ту часть документа, на которую пришёл запрос.
- Скрипт вносит изменения с учётом полученной информации (без полной перезагрузки страницы).

Задание

- Создать встраиваемую базу данных из 20-ти объектов
- Вывести на веб-страницу содержимое базы данных:
 - На странице кроме таблицы с содержимом базы данных находятся поля для ввода фильтров
 - Содержимое полей отправляется на сервер
 - Сервер на основе содержимого фильтров формирует запрос к базе данных
 - Общение между клиентом и сервером в формате JSON
 - Таблица формируется на стороне клиента

Выполнение работы

Был создан проект Node.JS.

В качестве встраиваемой базы данных была выбрана SQLite.

В качестве объектов хранящихся в базе данных выступают «игры».

В начале программы подключаются необходимые модули:

Для создания веб-сервера

```
var http = require("http");
```

Для работы с базой данных

```
var sqlite3 = require("sqlite3").verbose();
```

Соединение с базой данных

```
var db = new sqlite3.Database("./GameDB.db");
```

Делаем обращение к базе данных, для проверки таблицы

```
db.all("SELECT name FROM sqlite_master WHERE type='table' AND name='GameDB_4';",  
function (err, rows)
```

В случае её отсутствия выполняется создание и заполнение таблицы

```
db.run("CREATE TABLE GameDB_4 (game TEXT NOT NULL, price integer NOT NULL, releas  
e_date TEXT NOT NULL, platform TEXT NOT NULL);", function (err) {  
    //Заполнение  
    var statement = db.prepare("INSERT INTO GameDB_4 VALUES(?, ?, ?, ?);")  
    statement.run("Fallout", 35, "1994", "Bethesda Launcher");  
    (частичный код программы)
```

Выполняется создание и логирование в консоль

```
http.createServer(server_callback).listen(3000);  
console.log("Listen at http://localhost:3000/");
```

В функции обработки запроса к серверу

```
var server_callback = function (request, response) {  
    console.log("request to: " + request.url + " method: " + request.method)  
    if (request.method == "GET") {  
        handle_GET(request, response);  
    } else {  
        handle_POST(request, response);  
    }  
}
```

Разделяются POST и GET запросы к серверу

В обработчике GET запросов

```
var handle_GET = function (request, response) {
  switch (request.url) {
    case "/":
      fs.readFile("./db4.2.html", function (err, content) {
        if (!err) {
          response.writeHead(200, { "Content-Type": "text/html; charset=utf-8" });
          response.end(content, "utf-8")
        } else {
          response.writeHead(500, { "Content-Type": "text/plain; charset=utf-8" });
          response.end(err.message, "utf-8");
          console.log(err);
        }
      });
      break;
    default:
      response.writeHead(404, { "Content-Type": "text/html; charset=utf-8" });
      response.end("<!DOCTYPE html>\n" +
        "<html>\n" +
        "  <head>\n" +
        "    <meta charset='utf-8'>\n" +
        "  </head>\n" +
        "  <body>\n" +
        "404, NOT FOUND: " + request.url +
        " \n</body>\n" +
        "</html>"
      );
  }
}
```

Клиенту отправляется либо содержимое файла db4.2.html, либо сообщение об ошибке.

В обработчике POST запросов

```
var handle_POST = function (request, response) {  
  if (request.url != "/get_table") {  
    response.writeHead(500, { "Content-Type": "text/plain; charset=utf-8" });  
    response.end();  
  }  
  
  var data = '';  
  request.on('data', function (chunk) {  
    data += chunk;  
  });  
  request.on('end', function () {  
    var filters = JSON.parse(data);
```

формируется полученный от клиента объект в формате JSON и котором хранятся значения фильтров

```
  stmt = db.prepare("SELECT * FROM GameDB_4 WHERE " +  
    "instr(game, ?)>0 AND " +  
    "price>=? AND price<=?");  
  stmt.all([filters.game, filters.price_from, filters.price_to],  
    function (err, rows) {  
      if (err) {  
        console.log(err);  
        response.writeHead(404, { "Content-Type": "text/plain; charset=utf-8" });  
        response.end();  
      } else {  
        response.writeHead(200, { "Content-Type": "application/json; charset=utf-8" });  
        db_data.table = rows;  
        response.end(JSON.stringify(db_data));  
        stmt.finalize();  
      }  
    });
```

Выполняется запрос к базе данных и в случае успеха результат отправляется клиенту в формате JSON, в случае ошибки отправляет сообщение об ошибке.

Так же была создана веб-страница, в теле которой

```
<body>
  <p>
    GameDB_4 from ./GameDB-4-2 SQLite.
  </p>
  <p>
    Game Search<input id="game" type="text"> Game Price from: <input id="from"
      type="number" value=0> to: <input id="to" type="number" value=100>$.
  </p>
  <p>
    <span id="res" style="font-style: italic"></span>
  </p>
</body>
```

Находятся 4 поля для ввода – фильтры и поле, в котором будет находиться таблица с результатом запроса к серверу

При загрузке браузером страницы

```
window.onload = function () {
  var game_input = document.getElementById("game");
  game_input.oninput = function () {
    game_filter = game_input.value;
    update();
  };
  var fr_input = document.getElementById("from");
  fr_input.oninput = function () {
    price_filter_from = fr_input.value;
    update();
  };
  var to_input = document.getElementById("to");
  to_input.oninput = function () {
    price_filter_to = to_input.value;
    update();
  };
  update();
}
```

Устанавливаются обработчики изменения значений в полях ввода – при их изменении вызывается функция отправки серверу новых значений и изменения содержимого таблицы

```
function update() {
  var filters = {};
  filters.game = game_filter;
  filters.price_from = price_filter_from;
  filters.price_to = price_filter_to;
  readServer("/get_table", JSON.stringify(filters), function (err, response) {
    if (err) document.getElementById("res").innerHTML = err;
    else {
      var temp = "";
      temp = "<table cellpadding=\"5\" cellspacing=\"2\" border=\"1\"";
    }
  });
}
```



```

        var rows = JSON.parse(response).table;
        for (var i = 0; i < rows.length; i++) {
            temp += "<tr><td>" + rows[i].game + "</td><td align=\"center\">" + rows[i].price + "$</td><td align=\"center\">" + rows[i].release_date + "</td><td align=\"center\">" + rows[i].platform + "</td></tr>\n";
        }
        temp += "</table>";
        document.getElementById("res").innerHTML = temp;
    }
});
}

```

В ней используется функция, отправляющая серверу фоновый запрос по технологии AJAX

```

function readServer(url, data, callback) {
    var req = new XMLHttpRequest();
    req.onreadystatechange = function () {
        if (req.readyState === 4) { //"Loaded"
            if (req.status === 200) { //"OK"
                callback(undefined, req.responseText);
            } else {
                callback(new Error(req.status));
            }
        }
    };

    req.open("POST", url, true);
    req.setRequestHeader('Content-Type', 'application/json');
    req.send(data);
}

```

Эта функция принимает на вход URL сервера(можно относительный), данные для отправки в теле запроса и функцию – обработчик результата.

После загрузки страницы:

GameDB_4 from ./GameDB-4-2 SQLite.

Game Search Game Price from: to: \$.

<i>Fallout</i>	35\$	1994	<i>Bethesda Launcher</i>
<i>Fallout 2</i>	45\$	1996	<i>Bethesda Launcher</i>
<i>Fallout Nevada</i>	0\$	2003	<i>non</i>
<i>Fallout New Vegas</i>	40\$	2009	<i>Steam</i>
<i>Halo Combat Evolved</i>	60\$	2003	<i>Xbox Live</i>
<i>Halo 2</i>	60\$	2006	<i>Xbox Live</i>
<i>Dota 2</i>	0\$	2011	<i>Steam</i>
<i>League of legends</i>	0\$	2009	<i>RIOT Launcher</i>
<i>Counter Strike - Condition Zero</i>	25\$	2005	<i>Steam</i>
<i>Call of Duty 3</i>	60\$	2006	<i>Xbox Live</i>
<i>Horizon Zero Dawn</i>	60\$	2017	<i>PS Network</i>
<i>The last of Us</i>	60\$	2013	<i>PS Network</i>
<i>Fallout 76</i>	50\$	2018	<i>Bethesda Launcher</i>
<i>Halo 5</i>	60\$	2015	<i>Xbox Live</i>
<i>Stubbs the Zombie</i>	45\$	2004	<i>Xbox Live</i>
<i>Battlefield 1942</i>	55\$	2003	<i>Retail</i>
<i>Team Fortress 2</i>	5\$	2008	<i>Steam</i>
<i>Half-Life</i>	45\$	1999	<i>Retail</i>
<i>Half-Life 2</i>	60\$	2004	<i>Steam</i>
<i>Half-Life 3</i>	120\$	2077	<i>CyberPunkedSteam</i>

После ввода значений в поля фильтров:

GameDB_4 from ./GameDB-4-2 SQLite.

Game Search Game Price from: to: \$.

<i>Fallout Nevada</i>	0\$	2003	<i>non</i>
<i>Dota 2</i>	0\$	2011	<i>Steam</i>
<i>League of legends</i>	0\$	2009	<i>RIOT Launcher</i>
<i>Team Fortress 2</i>	5\$	2008	<i>Steam</i>

Добавление иного фильтра на основе «out»:

GameDB_4 from ./GameDB-4-2 SQLite.

Game Search Game Price from: to: \$.

<i>Fallout</i>	35\$	1994	<i>Bethesda Launcher</i>
<i>Fallout 2</i>	45\$	1996	<i>Bethesda Launcher</i>
<i>Fallout Nevada</i>	0\$	2003	<i>non</i>
<i>Fallout New Vegas</i>	40\$	2009	<i>Steam</i>
<i>Fallout 76</i>	50\$	2018	<i>Bethesda Launcher</i>

Вывод

В ходе выполнения работы был создан веб-сервер и веб-страница, которые взаимодействуют с помощью AJAX. Веб-сервер взаимодействует с базой данных и отправляет данные клиенту, на стороне клиента формируется таблица с содержимым базы данных и отправляются содержащиеся в полях ввода – фильтрах значения на сервер.

СПИСОК ИСТОЧНИКОВ

1. Документация NodeJS - <https://nodejs.org/en/docs/>
2. Статья о AJAX в Википедии - <https://ru.wikipedia.org/wiki/AJAX>

Приложение. Листинг.

GameDB-4-2.js

```
var fs = require("fs")
var http = require("http");
var sqlite3 = require("sqlite3").verbose();

var db = new sqlite3.Database("./GameDB-4-2.db");

var handle_GET = function (request, response) {
  switch (request.url) {
    case "/":
      fs.readFile("./db4.2.html", function (err, content) {
        if (!err) {
          response.writeHead(200, { "Content-Type": "text/html; charset=utf-8" });
          response.end(content, "utf-8")
        } else {
          response.writeHead(500, { "Content-Type": "text/plain; charset=utf-8" });
          response.end(err.message, "utf-8");
          console.log(err);
        }
      });
      break;
    default:
      response.writeHead(404, { "Content-Type": "text/html; charset=utf-8" });
      response.end("<!DOCTYPE html>\n" +
        "<html>\n" +
        "  <head>\n" +
        "    <meta charset='utf-8'>\n" +
        "  </head>\n" +
        "  <body>\n" +
        "404, NOT FOUND: " + request.url +
        " \n</body>\n" +
        "</html>"
      );
  }
}

var handle_POST = function (request, response) {
  if (request.url != "/get_table") {
    response.writeHead(500, { "Content-Type": "text/plain; charset=utf-8" });
    response.end();
  }

  var data = '';
  request.on('data', function (chunk) {
```

```

        data += chunk;
    });
    request.on('end', function () {
        var filters = JSON.parse(data);
        var db_data = {};
        stmt = db.prepare("SELECT * FROM GameDB_4 WHERE " +
            "instr(game, ?)>0 AND " +
            "price>=? AND price<=?");
        stmt.all([filters.game, filters.price_from, filters.price_to],
            function (err, rows) {
                if (err) {
                    console.log(err);
                    response.writeHead(404, { "Content-
Type": "text/plain; charset=utf-8" });
                    response.end();
                } else {
                    response.writeHead(200, { "Content-
Type": "application/json; charset=utf-8" });
                    db_data.table = rows;
                    response.end(JSON.stringify(db_data));
                    stmt.finalize();
                }
            });
    });
}

var server_callback = function (request, response) {
    console.log("request to: " + request.url + " method: " + request.method)
    if (request.method == "GET") {
        handle_GET(request, response);
    } else {
        handle_POST(request, response);
    }
}

db.all("SELECT name FROM sqlite_master WHERE type='table' AND name='GameDB_4';",
function (err, rows) {
    if (err || rows.length == 0) {
        db.run("CREATE TABLE GameDB_4 (game TEXT NOT NULL, price integer NOT NULL
, release_date TEXT NOT NULL, platform TEXT NOT NULL);", function (err) {
            //fill here
            var statement = db.prepare("INSERT INTO GameDB_4 VALUES(?, ?, ?, ?);")

            statement.run("Fallout", 35, "1994", "Bethesda Launcher");
            statement.run("Fallout 2", 45, "1996", "Bethesda Launcher");
            statement.run("Fallout Nevada", 0, "2003", "non");
            statement.run("Fallout New Vegas", 40, "2009", "Steam");
            statement.run("Halo Combat Evolved", 60, "2003", "Xbox Live");
            statement.run("Halo 2", 60, "2006", "Xbox Live");
            statement.run("Dota 2", 0, "2011", "Steam");
            statement.run("League of legends", 0, "2009", "RIOT Launcher");
        }
    }
}

```

```

        statement.run("Counter Strike - Condition Zero", 25, "2005", "Steam");
    };

    statement.run("Call of Duty 3", 60, "2006", "Xbox Live");
    statement.run("Horizon Zero Dawn", 60, "2017", "PS Network");
    statement.run("The last of Us", 60, "2013", "PS Network");
    statement.run("Fallout 76", 50, "2018", "Bethesda Launcher");
    statement.run("Halo 5", 60, "2015", "Xbox Live");
    statement.run("Stubbs the Zombie", 45, "2004", "Xbox Live");
    statement.run("Battlefield 1942", 55, "2003", "Retail");
    statement.run("Team Fortress 2", 5, "2008", "Steam");
    statement.run("Half-Life", 45, "1999", "Retail");
    statement.run("Half-Life 2", 60, "2004", "Steam");
    statement.run("Half-Life 3", 120, "2077", "CyberPunkedSteam");
    statement.finalize();
    });
    console.log("db GameDB_4 created");
}
http.createServer(server_callback).listen(3000);
console.log("Listen at http://localhost:3000/");
});

```

db4.2.html

```

<!DOCTYPE html>
<html>

<head>
    <title>NodeJS+AJAX</title>
    <meta charset="utf-8" />
    <script type="text/javascript">
        var game_filter = "";
        var price_filter_from = 0;
        var price_filter_to = 100;

        function readServer(url, data, callback) {
            var req = new XMLHttpRequest();
            req.onreadystatechange = function () {
                if (req.readyState === 4) { //"Loaded"
                    if (req.status === 200) { //"OK"
                        callback(undefined, req.responseText);
                    } else {
                        callback(new Error(req.status));
                    }
                }
            };

            req.open("POST", url, true);
            req.setRequestHeader('Content-Type', 'application/json');
            req.send(data);
        }
    </script>

```



```

function update() {
    var filters = {};
    filters.game = game_filter;
    filters.price_from = price_filter_from;
    filters.price_to = price_filter_to;
    readServer("/get_table", JSON.stringify(filters), function (err, response) {
        if (err) document.getElementById("res").innerHTML = err;
        else {
            var temp = "";
            temp = "<table cellpadding=\\\"2\\\" border=\\\"1\\\" cellspacing=\\\"5\\\">\\n";

            var rows = JSON.parse(response).table;
            for (var i = 0; i < rows.length; i++) {
                temp += "<tr><td>" + rows[i].game + "</td><td align=\\\"center\\\">" + rows[i].price + "$</td><td align=\\\"center\\\">" + rows[i].release_date + "</td><td align=\\\"center\\\">" + rows[i].platform + "</td></tr>\\n";
            }
            temp += "</table>";
            document.getElementById("res").innerHTML = temp;
        }
    });
}

window.onload = function () {
    var game_input = document.getElementById("game");
    game_input.oninput = function () {
        game_filter = game_input.value;
        update();
    };
    var fr_input = document.getElementById("from");
    fr_input.oninput = function () {
        price_filter_from = fr_input.value;
        update();
    };
    var to_input = document.getElementById("to");
    to_input.oninput = function () {
        price_filter_to = to_input.value;
        update();
    };
    update();
}
</script>
</head>

<body>
    <p>
        GameDB_4 from ./GameDB-4-2 SQLite.
    </p>
    <p>

```

```
Game Search<input id="game" type="text"> Game Price from: <input id="from"
"
    type="number" value=0> to: <input id="to" type="number" value=100>$.
</p>
<p>
    <span id="res" style="font-style: italic"></span>
</p>
</body>
</html>
```