Лабораторная работа №7. Сервер.

Целью данной работы является развитие приложения разработанного в рамках предыдущей лабораторной работе. В теоретической части показывается то, каким образом организовать работу веб-сервера на базе Node.js и Express. В практической части следует продолжить выполнение индивидуального задания из предыдущей лабораторной работы с учётом требований заданных к практической части в данной лабораторной работе.

Теоретическая часть

Используя Express и Node.js, мы можем реализовать полноценный API в стиле REST для взаимодействия с пользователем. Архитектура REST предполагает применение следующих методов или типов запросов HTTP для взаимодействия с сервером:

- GET
- POST
- PUT
- DELETE

Зачастую REST-стиль особенно удобен при создании всякого рода Single Page Application.

Пример №1 RESTful API

Рассмотрим, как создать свой API. Для нового проекта создадим новую папку. Сразу добавим в проект пакет express с помощью команды:

```
npm install express
```

В данном случае мы создадим экспериментальный проект, который призван продемонстрировать применение REST в приложении на Node.js+Express. Для обработки запросов определим в проекте следующий файл app.js:

```
const express = require("express");

const app = express();
app.use(express.json());

app.use(express.static("public"));

// условная база данных
const users = [];
let id = 1;     // для установки идентификаторов

// вспомогательная функция для поиска индекса пользователя по id
function findUserIndexById(id){
    for(let i=0; i < users.length; i++){
        if(users[i].id==id) return i;
    }
    return -1;
}
app.get("/api/users", function(_, res){

    res.send(users);
});
// получение одного пользователя по id
app.get("/api/users/:id", function(req, res){

    const id = req.params.id; // получаем id
    // находим в массиве пользователя по id
```

```
    const index = findUserIndexById(id);
    // отправляем пользователя
    if(index > -1){
        res.send(users[index]);
    }
    else{
        res.status(404).send("User not found");
    }
});
// получение отправленных данных
app.post("/api/users", function (req, res) {

    if(!req.body) return res.sendStatus(400);

    const userName = req.body.name;
    const userAge = req.body.age;
    const user = {name: userName, age: userAge};
    // присваиваем идентификатор из переменной id и увеличиваем ее на единицу
    user.id = id++;
    // добавляем пользователя в массив
    users.push(user);
    res.send(user);
});
 // удаление пользователя по id
app.delete("/api/users/:id", function(req, res){

    const id = req.params.id;
    const index = findUserIndexById(id);
    if(index > -1){
        // удаляем пользователя из массива по индексу
        const user = users.splice(index, 1)[0];
        res.send(user);
    }
    else{
        res.status(404).send("User not found");
    }
});
// изменение пользователя
app.put("/api/users", function(req, res){

    if(!req.body) return res.sendStatus(400);

    const id = req.body.id;
    const userName = req.body.name;
    const userAge = req.body.age;

    const index = findUserIndexById(id);
    if(index > -1){
        // изменяем данные у пользователя
        const user = users[index];
        user.age = userAge;
        user.name = userName;
        res.send(user);
    }
    else{
        res.status(404).send("User not found");
    }
});

app.listen(3000, function(){
    console.log("Сервер ожидает подключения...");
});
```

В данном приложении для простоты в качестве базы данных мы будем использовать обычный массив - массив users. По умолчанию он пуст

```
const users = [];
```

Каждый объект, который будет попадать в этот массив, будет иметь определённый числовой идентификатор. И для установки идентификаторов объектов при их создании определим переменную id:

```
let id = 1;
```

Также нам потребуется вспомогательная функция для поиска индекса пользователя по id:

```
function findUserIndexById(id){
    for(let i=0; i < users.length; i++){
        if(users[i].id==id) return i;
    }
    return -1;
}
```

Если пользователь найден, то возвращается его индекс в массиве. Если не найден, то возвращается число -1.
Для обработки запросов определено пять методов для каждого типа запросов: app.get()/app.post()/app.delete()/app.put().
Когда приложение получает запрос типа GET по адресу "api/users", то срабатывает следующий метод:

```
app.get("/api/users", function(req, res){
    res.send(users);
});
```

В качестве результата обработки мы должны отправить массив пользователей методом res.send().
Аналогично работает другой метод app.get(), который срабатывает, когда в адресе указан id пользователя:

```
app.get("/api/users/:id", function(req, res){

    const id = req.params.id; // получаем id
    // находим в массиве пользователя по id
    const index = findUserIndexById(id);
    // отправляем пользователя
    if(index > -1){
        res.send(users[index]);
    }
    else{
        res.status(404).send("User not found");
    }
});
```

Единственное, что в этом случае нам надо найти нужного пользователя по id в массиве, а если он не был найден, возвратить статусный код 404: res.status(404).send().
При получении запроса методом POST извлекаем отправленные клиентом данные из запроса:

```
app.post("/api/users", function (req, res) {
```

```
    if(!req.body) return res.sendStatus(400);

    const userName = req.body.name;
    const userAge = req.body.age;
    const user = {name: userName, age: userAge};
    // присваиваем идентификатор из переменной id и увеличиваем ее на единицу
    user.id = id++;
    // добавляем пользователя в массив
    users.push(user);
    res.send(user);
});
```

Поскольку вначале файла мы встроили в конвейер обработки запрос автоматического парсинга в json, то тело запроса будет представлять объект json, из которого мы можем взять свойства name и age и создать по ним объект пользователя. Для установки идентификатора применяем глобальную переменную id, значение которой инкрементируется.
При удалении производим похожие действия, только теперь извлекаем из массива удаляемый объект:

```
// удаление пользователя по id
app.delete("/api/users/:id", function(req, res){

    const id = req.params.id;
    const index = findUserIndexById(id);
    if(index > -1){
        // удаляем пользователя из массива по индексу
        const user = users.splice(index, 1)[0];
        res.send(user);
    }
    else{
        res.status(404).send("User not found");
    }
});
```

Если объект не найден, возвращаем статусный код 404.
Если приложению приходит PUT-запрос, то он обрабатывается методом app.put(), в котором получаем изменённые данные:

```
app.put("/api/users", function(req, res){

    if(!req.body) return res.sendStatus(400);

    const id = req.body.id;
    const userName = req.body.name;
    const userAge = req.body.age;

    const index = findUserIndexById(id);
    if(index > -1){
        // изменяем данные у пользователя
        const user = users[index];
        user.age = userAge;
        user.name = userName;
        res.send(user);
    }
    else{
        res.status(404).send("User not found");
    }
});
```

Здесь также для поиска изменяемого объекта находим по id его индекс, по индексу получаем объект и изменяем у него свойства.
Таким образом, мы определили простейший API. Теперь добавим код клиента. Итак, как установлено в коде, Express для хранения статических файлов использует папку public, поэтому создадим в проекте подобную папку. В этой папке определим новый файл index.html, который будет выполнять роль клиента. В итоге весь проект будет выглядеть следующим образом:

- app.js
- public
    - index.html

Далее определим в файле index.html следующий код:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <style>
    td, th {padding:5px;min-width:90px;max-width:200px; text-align:start;}
    .btn {padding:4px; border:1px solid #333; background-color: #eee;
          border-radius: 2px; margin:5px; cursor:pointer;}
    </style>
</head>
<body>
<h2>Список пользователей</h2>
<form name="userForm">
    <input type="hidden" name="id" value="0" />
    <p>
        <label>Имя:</label><br>
        <input name="name" />
    </p>
    <p>
        <label>Возраст:</label><br>
        <input name="age" type="number" />
    </p>
    <p>
        <button id="submitBtn" type="submit">Сохранить</button>
        <button id="resetBtn">Сбросить</button>
    </p>
</form>
<table>
    <thead><tr><th>Id</th><th>Имя</th><th>Возраст</th><th></th></tr></thead>
    <tbody></tbody>
</table>
<script>
const tbody = document.querySelector("tbody");
// Получение всех пользователей
async function GetUsers() {
// отправляет запрос и получаем ответ
    const response = await fetch("/api/users", {
        method: "GET",
        headers: { "Accept": "application/json" }
    });
    // если запрос прошел нормально
    if (response.ok === true) {
    // получаем данные
    const users = await response.json();
        users.forEach(user => {
            // добавляем полученные элементы в таблицу
            tbody.append(row(user));
```

```
        });
    }
}
// Получение одного пользователя
async function GetUser(id) {
    const response = await fetch("/api/users/" + id, {
        method: "GET",
        headers: { "Accept": "application/json" }
    });
    if (response.ok === true) {
        const user = await response.json();
        const form = document.forms["userForm"];
        form.elements["id"].value = user.id;
        form.elements["name"].value = user.name;
        form.elements["age"].value = user.age;
    }
}
// Добавление пользователя
async function CreateUser(userName, userAge) {
    const response = await fetch("api/users", {
        method: "POST",
        headers: { "Accept": "application/json", "Content-Type":
"application/json" },
        body: JSON.stringify({
            name: userName,
            age: parseInt(userAge, 10)
        })
    });
    if (response.ok === true) {
        const user = await response.json();
        reset();
        tbody.append(row(user));
    }
}
// Изменение пользователя
async function EditUser(userId, userName, userAge) {
    const response = await fetch("api/users", {
        method: "PUT",
        headers: { "Accept": "application/json", "Content-Type":
"application/json" },
        body: JSON.stringify({
            id: userId,
            name: userName,
            age: parseInt(userAge, 10)
        })
    });
    if (response.ok === true) {
        const user = await response.json();
        reset();
        document.querySelector(`tr[data-rowid="$
{user.id}"]`).replaceWith(row(user));
    }
}
// Удаление пользователя
async function DeleteUser(id) {
    const response = await fetch("/api/users/" + id, {
        method: "DELETE",
        headers: { "Accept": "application/json" }
    });
    if (response.ok === true) {
        const user = await response.json();
        document.querySelector(`tr[data-rowid="${user.id}"]`).remove();
```

```
    }
}

// сброс формы
function reset() {
    const form = document.forms["userForm"];
    console.log(form);
    form.reset();
    form.elements["id"].value = 0;
}
// создание строки для таблицы
function row(user) {

    const tr = document.createElement("tr");
    tr.setAttribute("data-rowid", user.id);

    const idTd = document.createElement("td");
    idTd.append(user.id);
    tr.append(idTd);

    const nameTd = document.createElement("td");
    nameTd.append(user.name);
    tr.append(nameTd);

    const ageTd = document.createElement("td");
    ageTd.append(user.age);
    tr.append(ageTd);

    const linksTd = document.createElement("td");

    const editLink = document.createElement("a");
    editLink.setAttribute("data-id", user.id);
    editLink.setAttribute("class", "btn");
    editLink.append("Изменить");
    editLink.addEventListener("click", e => {
        e.preventDefault();
        GetUser(user.id);
    });
    linksTd.append(editLink);

    const removeLink = document.createElement("a");
    removeLink.setAttribute("data-id", user.id);
    removeLink.setAttribute("class", "btn");
    removeLink.append("Удалить");
    removeLink.addEventListener("click", e => {
        e.preventDefault();
        DeleteUser(user.id);
    });

    linksTd.append(removeLink);
    tr.appendChild(linksTd);

    return tr;
}
// сброс значений формы
document.getElementById("resetBtn").addEventListener("click", e => {
    e.preventDefault();
    reset();
});

// отправка формы
document.forms["userForm"].addEventListener("submit", e => {
```

```
    e.preventDefault();
    const form = document.forms["userForm"];
    const id = form.elements["id"].value;
    const name = form.elements["name"].value;
    const age = form.elements["age"].value;
    if (id == 0)
        CreateUser(name, age);
    else
        EditUser(id, name, age);
});

// загрузка пользователей
GetUsers();
</script>
</body>
</html>
```

Основная логика здесь заключена в коде javascript. При загрузке страницы в браузере получаем все объекты из БД с помощью функции GetUsers:

```
async function GetUsers() {
    // отправляет запрос и получаем ответ
    const response = await fetch("/api/users", {
        method: "GET",
        headers: { "Accept": "application/json" }
    });
    // если запрос прошел нормально
    if (response.ok === true) {
        // получаем данные
        const users = await response.json();
        users.forEach(user => {
            // добавляем полученные элементы в таблицу
            tbody.append(row(user));
        });
    }
}
```

Для добавления строк в таблицу используется функция row(), которая возвращает строку. В этой строке будут определены ссылки для изменения и удаления пользователя.
Ссылка для изменения пользователя с помощью функции GetUser() получает с сервера выделенного пользователя:

```
async function GetUser(id) {
    const response = await fetch("/api/users/" + id, {
        method: "GET",
        headers: { "Accept": "application/json" }
    });
    if (response.ok === true) {
        const user = await response.json();
        const form = document.forms["userForm"];
        form.elements["id"].value = user.id;
        form.elements["name"].value = user.name;
        form.elements["age"].value = user.age;
    }
}
```

И выделенный пользователь добавляется в форму над таблицей. Эта же форма применяется и для добавления объекта. С помощью скрытого поля, которое хранит id пользователя, мы

можем узнать, какое действие выполняется - добавление или редактирование. Если id равен 0, то выполняется функция CreateUser, которая отправляет данные в POST-запросе:

```
async function CreateUser(userName, userAge) {
    const response = await fetch("api/users", {
        method: "POST",
        headers: { "Accept": "application/json",
                   "Content-Type": "application/json" },
        body: JSON.stringify({
            name: userName,
            age: parseInt(userAge, 10)
        })
    });
    if (response.ok === true) {
        const user = await response.json();
        reset();
        tbody.append(row(user));
    }
}
```

Если же ранее пользователь был загружен на форму, и в скрытом поле сохранился его id, то выполняется функция EditUser, которая отправляет PUT-запрос:

```
async function EditUser(userId, userName, userAge) {
    const response = await fetch("api/users", {
        method: "PUT",
        headers: { "Accept": "application/json",
                   "Content-Type": "application/json" },
        body: JSON.stringify({
            id: userId,
            name: userName,
            age: parseInt(userAge, 10)
        })
    });
    if (response.ok === true) {
        const user = await response.json();
        reset();
        document.querySelector(`tr[data-rowid="${user.id}"]`)
                .replaceWith(row(user));
    }
}
```

Запустим приложение, обратимся в браузере по адресу "http://localhost:3000" и мы сможем добавлять новых пользователей, рисунок 7.1 и 7.2:

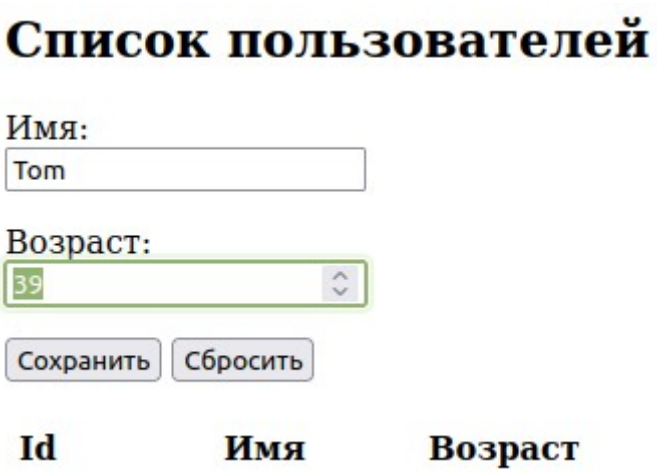

Рис. 7.1 REST и API в Express и Node.js

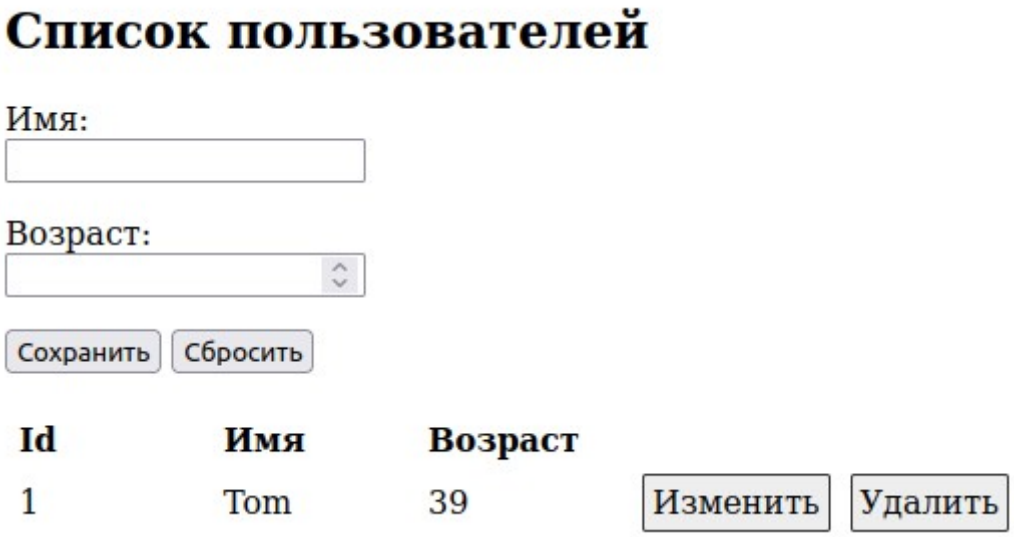

Рис. 7.2 REST и API в Express и Node.js

Аналогично пользователей можно изменять и удалять.

Пример №2 Веб приложение для хранения тэгизированных задач

Структура проекта

```
client\
      javascripts\
            app.js
      stylesheets\
            reset.css
            style.css
      index.html
node_modules\
            ...
package.json
server.js
```

Экранные формы приложения

На рисунках 7.3 и 7.4 приводятся экранные формы веб приложение для хранения тэгизированных задач.

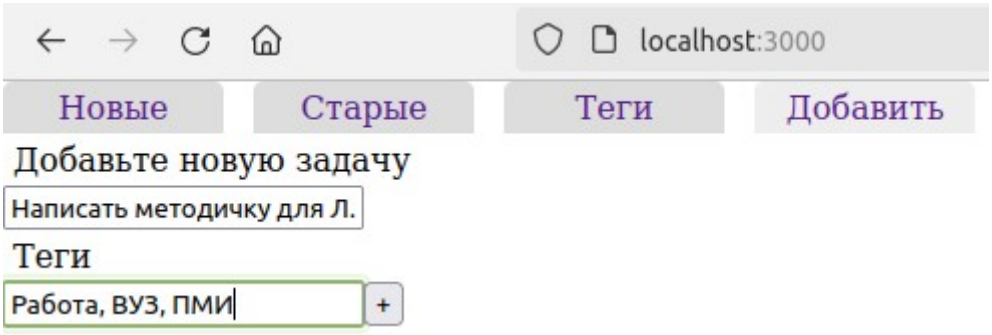

Рис. 7.3 Экранные формы: веб приложение для хранения тэгизированных задач

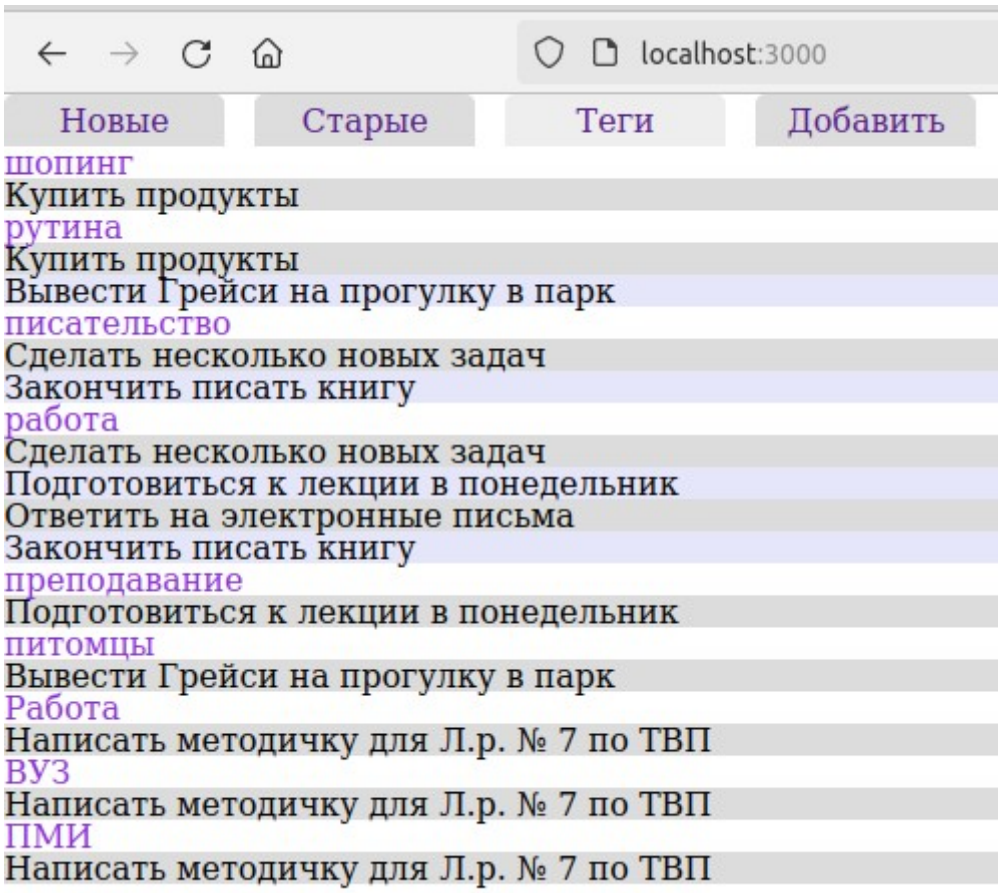

Рис. 7.4 Экранные формы: веб приложение для хранения тэгизированных задач

Исходный код приложения

app.js

```
var main = function (toDoObjects) {
    "use strict";
    var toDos = toDoObjects.map(function (toDo) {
        return toDo.description;
    });
    var organizeByTags = function (toDoObjects) {
        //console.log("organizeByTags вызвана - 0");
        var toArr = [];
        var toTags = [];
        toDoObjects.forEach(function (e1) {
            //console.log(e);
            e1.tags.forEach(function (tag) {
                if (toArr.indexOf(tag) === -1) {
```

```
                //console.log(tag);
                toArr.push(tag);
                toTags.push({
                    "name": tag,
                    "toDos": [e1.description]
                });
            } else {
                toTags.forEach(function (e2) {
                    if (e2.name === tag) {
                        e2.toDos.push(e1.description);
                    }
                });
            }
        });
    });
    //console.log(toTags);
    return toTags;
};
$(".tabs a span").toArray().forEach(function (element) {
    $(element).on("click", function () {
        var $element = $(element),
                $content;
        $(".tabs a span").removeClass("active");
        $element.addClass("active");
        $("main .content").empty();

        if ($element.parent().is(":nth-child(1)")) {
            var i;
            $content = $("<ul>");
            for (i = toDos.length; i > -1; i--) {
                $content.append($("<li>").text(toDos[i]));
            }
        } else if ($element.parent().is(":nth-child(2)")) {
            $content = $("<ul>");
            toDos.forEach(function (todo) {
                $content.append($("<li>").text(todo));
            });
        } else if ($element.parent().is(":nth-child(3)")) {
            //console.log("Щелчок на вкладке Теги");
            //console.log(organizeByTags(toDoObjects));
            var organizedByTag = organizeByTags(toDoObjects);
            organizedByTag.forEach(function (tag) {
                var $tagName = $("<h3>").text(tag.name),
                        $content = $("<ul>");
                tag.toDos.forEach(function (description) {
                    var $li = $("<li>").text(description);
                    $content.append($li);
                });
                $("main .content").append($tagName);
                $("main .content").append($content);
            });
        } else if ($element.parent().is(":nth-child(4)")) {
            $content = $("<div>");
            $content.append($("<p>").text("Добавьте новую задачу"));
            var $input = $("<input>");
            var $button = $("<button>");
            $content.append($input);
            $content.append("<p>Теги</p>");
            var $input_tags = $("<input>");
            $content.append($input_tags);
            $content.append($button.text("+"));
            var description = $input.val();
```

```
                var tags = $input_tags.val().split(",");
                var addTaskFromInputBox = function () {
                    if ($input.val() !== "") {

                        var newToDo = {
                                    "description": $input.val(),
                                    "tags": $input_tags.val().split(",")
                        };
                        $.post("todos", newToDo, function (response) {
                            console.log("Мы отправили данные"
                                    " и получили ответ сервера!");
                            console.log(response);
                            toDos.push($input.val());
                            toDoObjects.push({
                              "description": $input.val(),
                              "tags": $input_tags.val().split(",")});
                            $input.val("");
                            $input_tags.val("");
                        });
                    }
                };
                $button.on("click", function (event) {
                    addTaskFromInputBox();
                });
                $input.on("keypress", function (event) {
                    if (event.keyCode === 13) {
                        addTaskFromInputBox();
                    }
                });
                //console.log(toDoObjects);
            }
            $("main .content").append($content);
            return false;
        });
    });
    $(".tabs a:first-child span").trigger("click");
};
$(document).ready(function () {
    $.getJSON("todos.json", function (toDoObjects) {
// вызываем функцию main с задачами в качестве аргумента
        main(toDoObjects);
    });
});
```

reset.css

```
/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
*/

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
```

```
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
      margin: 0;
      padding: 0;
      border: 0;
      font-size: 100%;
      font: inherit;
      vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
      display: block;
}
body {
      line-height: 1;
}
ol, ul {
      list-style: none;
}
blockquote, q {
      quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
      content: '';
      content: none;
}
table {
      border-collapse: collapse;
      border-spacing: 0;
}
```

style.css

```
.tabs a span {
    display: inline-block;
    border-radius: 5px 5px 0 0;
    width: 100px;
    margin-right: 10px;
    text-align: center;
    background: #ddd;
    padding: 5px;
}
.tabs a span.active {
    background: #eee;
}

.content p {
    margin-right: 10px;
    padding: 5px;
    border-radius: 30px;
}

.content li:nth-child(even) {
    background: lavender;
}
.content li:nth-child(odd) {
    background: gainsboro;
}
```

```
.content h3 {
    color: blueviolet;
}
```

index.html

```
<!doctype html>
<html>
    <head>
        <title>Простое приложение</title>
        <meta charset="UTF-8">
        <link href="stylesheets/reset.css"
                  rel="stylesheet" type="text/css">
        <link href="stylesheets/style.css"
                  rel="stylesheet" type="text/css">
        <link href='http://fonts.googleapis.com/css?family=Ubuntu'
                  rel='stylesheet' type='text/css'>
        <link href='http://fonts.googleapis.com/css?family=Droid+Sans'
                  rel='stylesheet' type='text/css'>

    </head>
    <body>
        <main>
            <div class="container">
                <div class="tabs">
                    <a href=""><span class="active">Новые</span></a>
                    <a href=""><span>Старые</span></a>
                    <a href=""><span>Теги</span></a>
                    <a href=""><span>Добавить</span></a>
                </div>
                <div class="content">
                </div>
            </div>
        </main>
        <footer>
        </footer>
        <script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
        <script src="javascripts/app.js"></script>
    </body>
</html>
```

package.json

```
{
    "name": "example",
    "version": "1.0.0",
    "description": "",
    "dependencies": {
        "express": "^4.17.1"
    }
}
```

server.js

```
var express = require("express"),
        http = require("http"),
        app = express(),
```

```
        toDos = [
            {
                "description": "Купить продукты",
                "tags": ["шопинг", "рутина"]
            },
            {
                "description": "Сделать несколько новых задач",
                "tags": ["писательство", "работа"]
            },
            {
                "description": "Подготовиться к лекции в понедельник",
                "tags": ["работа", "преподавание"]
            },
            {
                "description": "Ответить на электронные письма",
                "tags": ["работа"]
            },
            {
                "description": "Вывести Грейси на прогулку в парк",
                "tags": ["рутина", "питомцы"]
            },
            {
                "description": "Закончить писать книгу",
                "tags": ["писательство", "работа"]
            }
        ];
app.use(express.static(__dirname + "/client"));
http.createServer(app).listen(3000);
app.get("/todos.json", function (req, res) {
    res.json(toDos);
});
app.use(express.urlencoded());
app.post("/todos", function (req, res) {
    console.log("Данные были отправлены на сервер!");
    var newToDo = req.body;
    console.log(newToDo);
    toDos.push(newToDo);
    res.json({"message": "Вы размещаетесь на сервере!"});
});
```

Практическая часть

1. Ознакомиться с теоретической частью.
2. Продемонстрировать работу приложения из примера №1.
3. Пояснить преподавателю работу приложения из примера №1.
4. Разработать веб сервер на базе Node.js + Express, который хранит данные в массиве в оперативной памяти. За основу взять пример №2.
5. Реализовать запросы на добавление (POST) данных в указанный массив и просмотр (GET) данных, хранящихся на сервере в указанном массиве.
6. Разработку вести на основе проекта выполненного по индивидуальному заданию из предыдущей лабораторной работы.
7. Для конструирования содержимого html-документов использовать jQuery.