Створення API

Останнє оновлення: 07.09.2021

Використовуючи Express та Node.js, ми можемо реалізувати повноцінний API у стилі REST для взаємодії з користувачем. Архітектура REST передбачає застосування наступних методів або типів запитів HTTP для взаємодії із сервером:

- ОТРИМАТИ
- ПОСТ
- ПОСТАВИТИ
- ВИДАЛИТИ

Найчастіше REST-стиль особливо зручний при створенні різноманітних Single Page Application, які нерідко використовують спеціальні javascript-фреймворки типу Angular, React або Knockout.

Розглянемо, як створити API. Для нового проекту створимо нову папку, яка нехай буде називатися **webapp** . Відразу визначимо у проекті файл **package.json** :

```
{
  "name": "webapp",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.17.0"
  }
}
```

У проекті нам знадобляться express для обробки запиту. Далі перейдемо до цього каталогу в командному рядку/терміналі та для додавання пакету виконаємо команду:

```
встановити npm
```

В даному випадку ми створимо експериментальний проект, який зберігатиме дані у файлі json і який покликаний просто показати створення API в Node.js у стилі REST. А поки

що додамо до папки проекту новий файл **users.json** з наступним змістом:

```
[{
    "id":1,
    "name":"Tom",
    "age":24
},
{
    "id":2,
    "name":"Bob",
    "age":27
},
{
    "id":3,
    "name":"Alice",
    "age":"23"
}]
```

Для читання та запису до цього файлу ми будемо використовувати вбудований модуль fs. Для обробки запитів визначимо у проекті наступний файл *app.js* :

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

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

app.use(express.static(__dirname + "/public"));

const filePath = "users.json";
app.get("/api/users", function(req, res){

    const content = fs.readFileSync(filePath,"utf8");
    const users = JSON.parse(content);
    res.send(users);
});
// получение одного пользователя по id
app.get("/api/users/:id", function(req, res){

    const id = req.params.id; // получаем id
    const content = fs.readFileSync(filePath, "utf8");
    const users = JSON.parse(content);
    let user = null;
    // находим в массиве пользователя по id
    for(var i=0; i<users.length; i++){
        if(users[i].id==id){
            user = users[i];
            break;
        }
```

```
    }
    // отправляем пользователя
    if(user){
        res.send(user);
    }
    else{
        res.status(404).send();
    }
});
// получение отправленных данных
app.post("/api/users", jsonParser, function (req, res) {

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

    const userName = req.body.name;
    const userAge = req.body.age;
    let user = {name: userName, age: userAge};

    let data = fs.readFileSync(filePath, "utf8");
    let users = JSON.parse(data);

    // находим максимальный id
    const id = Math.max.apply(Math,users.map(function(o){return o.id;}))
    // увеличиваем его на единицу
    user.id = id+1;
    // добавляем пользователя в массив
    users.push(user);
    data = JSON.stringify(users);
    // перезаписываем файл с новыми данными
    fs.writeFileSync("users.json", data);
    res.send(user);
});
 // удаление пользователя по id
app.delete("/api/users/:id", function(req, res){

    const id = req.params.id;
    let data = fs.readFileSync(filePath, "utf8");
    let users = JSON.parse(data);
    let index = -1;
    // находим индекс пользователя в массиве
    for(var i=0; i < users.length; i++){
        if(users[i].id==id){
            index=i;
            break;
        }
    }
    if(index > -1){
        // удаляем пользователя из массива по индексу
        const user = users.splice(index, 1)[0];
```

```
        data = JSON.stringify(users);
        fs.writeFileSync("users.json", data);
        // отправляем удаленного пользователя
        res.send(user);
    }
    else{
        res.status(404).send();
    }
});
// изменение пользователя
app.put("/api/users", jsonParser, function(req, res){

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

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

    let data = fs.readFileSync(filePath, "utf8");
    const users = JSON.parse(data);
    let user;
    for(var i=0; i<users.length; i++){
        if(users[i].id==userId){
            user = users[i];
            break;
        }
    }
    // изменяем данные у пользователя
    if(user){
        user.age = userAge;
        user.name = userName;
        data = JSON.stringify(users);
        fs.writeFileSync("users.json", data);
        res.send(user);
    }
    else{
        res.status(404).send(user);
    }
});

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

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){

    const content = fs.readFileSync(filePath,"utf8");
    const users = JSON.parse(content);
    res.send(users);
});
```

В качестве результата обработки мы должны отправить массив пользователей, которые считываем из файла. Для упрощения кода приложения в рамкаха данного экспериментального проекта для чтения/записи файла применяются синхронные методы **fs.readFileSync()/fs.writeFileSync()**. Но в реальности, как правило, работа с данными будет идти через базу данных, а далее мы все это рассмотрим на примере MongoDB.

И чтобы получить данные из файла с помощью метода `fs.readFileSync()` считываем данные в строку, которую парсим в массив объектов с помощью функции **JSON.parse()**. И в конце полученные данные отправляем клиенту 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
    const content = fs.readFileSync(filePath, "utf8");
    const users = JSON.parse(content);
    let user = null;
    // находим в массиве пользователя по id
    for(var i=0; i<users.length; i++){
        if(users[i].id==id){
            user = users[i];
            break;
        }
    }
    // отправляем пользователя
    if(user){
        res.send(user);
    }
    else{
        res.status(404).send();
    }
});
```

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 нам надо применить парсер jsonParser для извлечения данных из запроса:

```
// получение отправленных данных
app.post("/api/users", jsonParser, function (req, res) {

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

    const userName = req.body.name;
    const userAge = req.body.age;
    let user = {name: userName, age: userAge};

    let data = fs.readFileSync(filePath, "utf8");
    let users = JSON.parse(data);

    // находим максимальный id
    const id = Math.max.apply(Math,users.map(function(o){return o.id;}))
    // увеличиваем его на единицу
    user.id = id+1;
    // добавляем пользователя в массив
    users.push(user);
    data = JSON.stringify(users);
    // перезаписываем файл с новыми данными
    fs.writeFileSync("users.json", data);
    res.send(user);
});
```

После получения данных нам надо создать новый объект и добавить его в массив объектов. Для этого считываем данные из файла, добавляем в массив новый объект и перезаписываем файл с обновленными данными.

При удалении производим похожие действия, только теперь извлекаем из массива удаляемый объект и опять же перезаписываем файл:

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

    const id = req.params.id;
    let data = fs.readFileSync(filePath, "utf8");
    let users = JSON.parse(data);
    let index = -1;
    // находим индекс пользователя в массиве
    for(var i=0; i < users.length; i++){
        if(users[i].id==id){
            index=i;
            break;
        }
    }
    if(index > -1){
        // удаляем пользователя из массива по индексу
        const user = users.splice(index, 1)[0];
```

```
        data = JSON.stringify(users);
        fs.writeFileSync("users.json", data);
        // отправляем удаленного пользователя
        res.send(user);
    }
    else{
        res.status(404).send();
    }
});
```

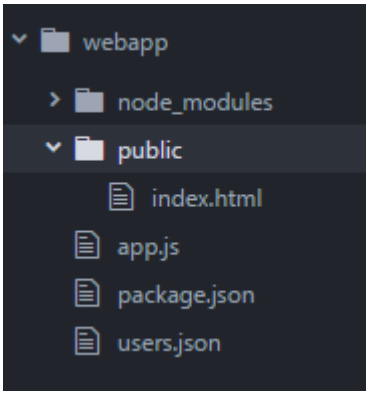
Если объект не найден, возвращаем статусный код 404.

Если приложению приходит PUT-запрос, то он обрабатывается методом **app.put()**, в котором с помощью jsonParser получаем измененные данные:

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

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

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

    let data = fs.readFileSync(filePath, "utf8");
    const users = JSON.parse(data);
    let user;
    for(var i=0; i<users.length; i++){
        if(users[i].id==userId){
            user = users[i];
            break;
        }
    }
    // изменяем данные у пользователя
    if(user){
        user.age = userAge;
        user.name = userName;
        data = JSON.stringify(users);
        fs.writeFileSync("users.json", data);
        res.send(user);
    }
    else{
        res.status(404).send(user);
    }
});
```

Здесь также для поиска изменяемого объекта считываем данные из файла, находим изменяемого пользователя по id, изменяем у него свойства и сохраняем обновленные данные в файл.

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:



Далее определим в файле index.html следующий код:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>Список пользователей</title>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstr
</head>
<body>
    <h2>Список пользователей</h2>
    <form name="userForm">
        <input type="hidden" name="id" value="0" />
        <div class="form-group">
            <label for="name">Имя:</label>
            <input class="form-control" name="name" />
        </div>
        <div class="form-group">
            <label for="age">Возраст:</label>
            <input class="form-control" name="age" />
        </div>
        <div class="panel-body">
            <button type="submit" class="btn btn-sm btn-primary">Сохранить<
            <a id="reset" class="btn btn-sm btn-primary">Сбросить</a>
        </div>
    </form>
    <table class="table table-condensed table-striped table-bordered">
        <thead><tr><th>Id</th><th>Имя</th><th>возраст</th><th></th></tr></t
        <tbody>
        </tbody>
    </table>
```

```
31
32      <script>
33      // Получение всех пользователей
34          async function GetUsers() {
35              // отправляет запрос и получаем ответ
36              const response = await fetch("/api/users", {
37                  method: "GET",
38                  headers: { "Accept": "application/json" }
39              });
40              // если запрос прошел нормально
41              if (response.ok === true) {
42                  // получаем данные
43                  const users = await response.json();
44                  let rows = document.querySelector("tbody");
45                  users.forEach(user => {
46                      // добавляем полученные элементы в таблицу
47                      rows.append(row(user));
48                  });
49              }
50          }
51          // Получение одного пользователя
52          async function GetUser(id) {
53              const response = await fetch("/api/users/" + id, {
54                  method: "GET",
55                  headers: { "Accept": "application/json" }
56              });
57              if (response.ok === true) {
58                  const user = await response.json();
59                  const form = document.forms["userForm"];
60                  form.elements["id"].value = user.id;
61                  form.elements["name"].value = user.name;
62                  form.elements["age"].value = user.age;
63              }
64          }
65          // Добавление пользователя
66          async function CreateUser(userName, userAge) {
67
68              const response = await fetch("api/users", {
69                  method: "POST",
70                  headers: { "Accept": "application/json", "Content-Type": "a
71                  body: JSON.stringify({
72                      name: userName,
73                      age: parseInt(userAge, 10)
74                  })
75              });
76              if (response.ok === true) {
77                  const user = await response.json();
78                  reset();
79                  document.querySelector("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": "a
                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 + "']").
            }
        }
        // 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 + "']").
            }
        }

        // сброс формы
        function reset() {
            const form = document.forms["userForm"];
            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("style", "cursor:pointer;padding:15px;");
            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("style", "cursor:pointer;padding:15px;"
            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("reset").click(function (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();
        let rows = document.querySelector("tbody");
        users.forEach(user => {
            // добавляем полученные элементы в таблицу
            rows.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;
```

```
    }
}
```

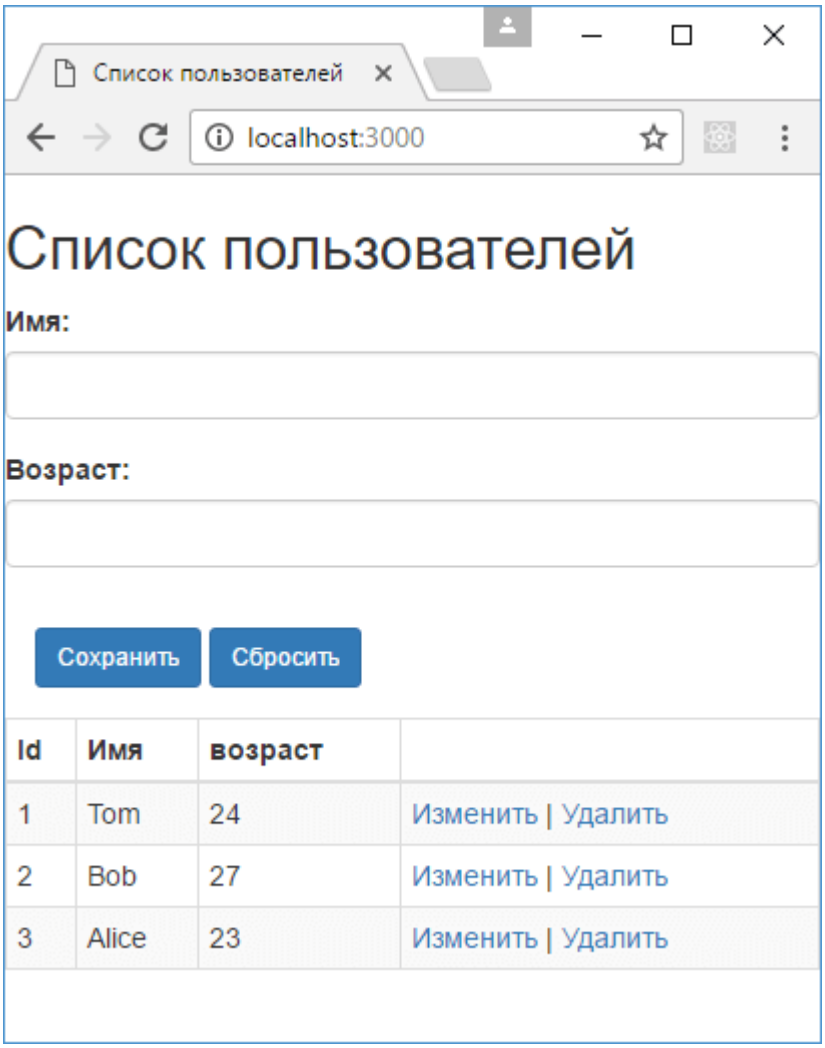
И выделенный пользователь добавляется в форму над таблицей. Эта же форма применяется и для добавления объекта. С помощью скрытого поля, которое хранит id пользователя, мы можем узнать, какое действие выполняется - добавление или редактирование. Если id равен 0, то выполняется функция CreateUser, которая отправляет данные в POST-запросе:

```
async function CreateUser(userName, userAge) {

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

Якщо ж користувач був завантажений на форму, і в прихованому полі зберігся його id, то виконується функція EditUser, яка відправляє PUT-запит:

```
async function EditUser(userId, userName, userAge) {
    const response = await fetch("api/users", {
        method: "PUT",
        headers: { "Accept": "application/json", "Content-Type": "applicatic
        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 + "']").replaceWi
    }
}
```

Запустимо програму, звернемося у браузері за адресою "http://localhost:3000" і ми зможемо керувати користувачами, які зберігаються у файлі json:



Назад Зміст Вперед