



## 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**:

```
1 {
2   "name": "webapp",
3   "version": "1.0.0",
4   "dependencies": {
5     "express": "^4.17.0"
6   }
7 }
```

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:

```
1  [{
2      "id":1,
3      "name":"Tom",
4      "age":24
5  },
6  {
7      "id":2,
8      "name":"Bob",
9      "age":27
10 },
11 {
12     "id":3,
13     "name":"Alice",
14     "age":"23"
15 }]
```

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

```
1  const express = require("express");
2  const fs = require("fs");
3
4  const app = express();
5  const jsonParser = express.json();
6
7  app.use(express.static(__dirname + "/public"));
8
9  const filePath = "users.json";
10 app.get("/api/users", function(req, res){
11
12     const content = fs.readFileSync(filePath, "utf8");
13     const users = JSON.parse(content);
14     res.send(users);
15 });
16 // получение одного пользователя по id
17 app.get("/api/users/:id", function(req, res){
18
19     const id = req.params.id; // получаем id
20     const content = fs.readFileSync(filePath, "utf8");
21     const users = JSON.parse(content);
22     let user = null;
23     // находим в массиве пользователя по id
24     for(var i=0; i<users.length; i++){
25         if(users[i].id==id){
26             user = users[i];
27             break;
28         }
29     }
30 }
```

```
29     }
30     // отправляем пользователя
31     if(user){
32         res.send(user);
33     }
34     else{
35         res.status(404).send();
36     }
37 });
38 // получение отправленных данных
39 app.post("/api/users", jsonParser, function (req, res) {
40
41     if(!req.body) return res.sendStatus(400);
42
43     const userName = req.body.name;
44     const userAge = req.body.age;
45     let user = {name: userName, age: userAge};
46
47     let data = fs.readFileSync(filePath, "utf8");
48     let users = JSON.parse(data);
49
50     // находим максимальный id
51     const id = Math.max.apply(Math, users.map(function(o){return o.id;}))
52     // увеличиваем его на единицу
53     user.id = id+1;
54     // добавляем пользователя в массив
55     users.push(user);
56     data = JSON.stringify(users);
57     // перезаписываем файл с новыми данными
58     fs.writeFileSync("users.json", data);
59     res.send(user);
60 });
61 // удаление пользователя по id
62 app.delete("/api/users/:id", function(req, res){
63
64     const id = req.params.id;
65     let data = fs.readFileSync(filePath, "utf8");
66     let users = JSON.parse(data);
67     let index = -1;
68     // находим индекс пользователя в массиве
69     for(var i=0; i < users.length; i++){
70         if(users[i].id==id){
71             index=i;
72             break;
73         }
74     }
75     if(index > -1){
76         // удаляем пользователя из массива по индексу
77         const user = users.splice(index, 1)[0];
```

```
78     data = JSON.stringify(users);
79     fs.writeFileSync("users.json", data);
80     // отправляем удаленного пользователя
81     res.send(user);
82   }
83   else{
84     res.status(404).send();
85   }
86 });
87 // изменение пользователя
88 app.put("/api/users", bodyParser, function(req, res){
89
90   if(!req.body) return res.sendStatus(400);
91
92   const userId = req.body.id;
93   const userName = req.body.name;
94   const userAge = req.body.age;
95
96   let data = fs.readFileSync(filePath, "utf8");
97   const users = JSON.parse(data);
98   let user;
99   for(var i=0; i<users.length; i++){
100     if(users[i].id==userId){
101       user = users[i];
102       break;
103     }
104   }
105   // изменяем данные у пользователя
106   if(user){
107     user.age = userAge;
108     user.name = userName;
109     data = JSON.stringify(users);
110     fs.writeFileSync("users.json", data);
111     res.send(user);
112   }
113   else{
114     res.status(404).send(user);
115   }
116 });
117
118 app.listen(3000, function(){
119   console.log("Сервер ожидает подключения...");
120 });
```

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:

```
1 app.get("/api/users", function(req, res){
2
3     const content = fs.readFileSync(filePath, "utf8");
4     const users = JSON.parse(content);
5     res.send(users);
6 });
```

В качестве результата обработки мы должны отправить массив пользователей, которые считываем из файла. Для упрощения кода приложения в рамках 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:

```
1 app.get("/api/users/:id", function(req, res){
2
3     const id = req.params.id; // получаем id
4     const content = fs.readFileSync(filePath, "utf8");
5     const users = JSON.parse(content);
6     let user = null;
7     // находим в массиве пользователя по id
8     for(var i=0; i<users.length; i++){
9         if(users[i].id==id){
10            user = users[i];
11            break;
12        }
13    }
14    // отправляем пользователя
15    if(user){
16        res.send(user);
17    }
18    else{
19        res.status(404).send();
20    }
21 });
```

Единственное, что в этом случае нам надо найти нужного пользователя по `id` в массиве, а если он не был найден, вернуть статусный код 404: `res.status(404).send()`.

При получении запроса методом POST нам надо применить парсер `jsonParser` для извлечения данных из запроса:

```
1 // получение отправленных данных
2 app.post("/api/users", jsonParser, function (req, res) {
3
4     if(!req.body) return res.sendStatus(400);
5
6     const userName = req.body.name;
7     const userAge = req.body.age;
8     let user = {name: userName, age: userAge};
9
10    let data = fs.readFileSync(filePath, "utf8");
11    let users = JSON.parse(data);
12
13    // находим максимальный id
14    const id = Math.max.apply(Math, users.map(function(o){return o.id;}))
15    // увеличиваем его на единицу
16    user.id = id+1;
17    // добавляем пользователя в массив
18    users.push(user);
19    data = JSON.stringify(users);
20    // перезаписываем файл с новыми данными
21    fs.writeFileSync("users.json", data);
22    res.send(user);
23 });
```

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:

```
1 // удаление пользователя по id
2 app.delete("/api/users/:id", function(req, res){
3
4     const id = req.params.id;
5     let data = fs.readFileSync(filePath, "utf8");
6     let users = JSON.parse(data);
7     let index = -1;
8     // находим индекс пользователя в массиве
9     for(var i=0; i < users.length; i++){
10         if(users[i].id==id){
11             index=i;
12             break;
13         }
14     }
15     if(index > -1){
16         // удаляем пользователя из массива по индексу
17         const user = users.splice(index, 1)[0];
```

```
18     data = JSON.stringify(users);
19     fs.writeFileSync("users.json", data);
20     // отправляем удаленного пользователя
21     res.send(user);
22 }
23 else{
24     res.status(404).send();
25 }
26 });
```

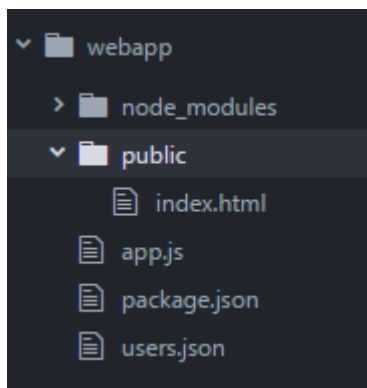
Если объект не найден, возвращаем статусный код 404.

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

```
1 app.put("/api/users", jsonParser, function(req, res){
2
3     if(!req.body) return res.sendStatus(400);
4
5     const userId = req.body.id;
6     const userName = req.body.name;
7     const userAge = req.body.age;
8
9     let data = fs.readFileSync(filePath, "utf8");
10    const users = JSON.parse(data);
11    let user;
12    for(var i=0; i<users.length; i++){
13        if(users[i].id==userId){
14            user = users[i];
15            break;
16        }
17    }
18    // изменяем данные у пользователя
19    if(user){
20        user.age = userAge;
21        user.name = userName;
22        data = JSON.stringify(users);
23        fs.writeFileSync("users.json", data);
24        res.send(user);
25    }
26    else{
27        res.status(404).send(user);
28    }
29 });
```

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:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <meta name="viewport" content="width=device-width" />
6   <title>Список пользователей</title>
7   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstr
8 </head>
9 <body>
10  <h2>Список пользователей</h2>
11  <form name="userForm">
12    <input type="hidden" name="id" value="0" />
13    <div class="form-group">
14      <label for="name">Имя:</label>
15      <input class="form-control" name="name" />
16    </div>
17    <div class="form-group">
18      <label for="age">Возраст:</label>
19      <input class="form-control" name="age" />
20    </div>
21    <div class="panel-body">
22      <button type="submit" class="btn btn-sm btn-primary">Сохранить</button>
23      <a id="reset" class="btn btn-sm btn-primary">Сбросить</a>
24    </div>
25  </form>
26  <table class="table table-condensed table-striped table-bordered">
27    <thead><tr><th>Id</th><th>Имя</th><th>возраст</th><th></th></tr></thead>
28    <tbody>
29  </tbody>
30 </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));
```

```
80     }
81   }
82   // Изменение пользователя
83   async function EditUser(userId, userName, userAge) {
84     const response = await fetch("api/users", {
85       method: "PUT",
86       headers: { "Accept": "application/json", "Content-Type": "a
87       body: JSON.stringify({
88         id: userId,
89         name: userName,
90         age: parseInt(userAge, 10)
91       })
92     });
93     if (response.ok === true) {
94       const user = await response.json();
95       reset();
96       document.querySelector("tr[data-rowid='" + user.id + "']").
97     }
98   }
99   // Удаление пользователя
100  async function DeleteUser(id) {
101    const response = await fetch("/api/users/" + id, {
102      method: "DELETE",
103      headers: { "Accept": "application/json" }
104    });
105    if (response.ok === true) {
106      const user = await response.json();
107      document.querySelector("tr[data-rowid='" + user.id + "']").
108    }
109  }
110
111  // сброс формы
112  function reset() {
113    const form = document.forms["userForm"];
114    form.reset();
115    form.elements["id"].value = 0;
116  }
117  // создание строки для таблицы
118  function row(user) {
119
120    const tr = document.createElement("tr");
121    tr.setAttribute("data-rowid", user.id);
122
123    const idTd = document.createElement("td");
124    idTd.append(user.id);
125    tr.append(idTd);
126
127    const nameTd = document.createElement("td");
128    nameTd.append(user.name);
```

```
129     tr.append(nameTd);
130
131     const ageTd = document.createElement("td");
132     ageTd.append(user.age);
133     tr.append(ageTd);
134
135     const linksTd = document.createElement("td");
136
137     const editLink = document.createElement("a");
138     editLink.setAttribute("data-id", user.id);
139     editLink.setAttribute("style", "cursor:pointer;padding:15px;");
140     editLink.append("Изменить");
141     editLink.addEventListener("click", e => {
142
143         e.preventDefault();
144         GetUser(user.id);
145     });
146     linksTd.append(editLink);
147
148     const removeLink = document.createElement("a");
149     removeLink.setAttribute("data-id", user.id);
150     removeLink.setAttribute("style", "cursor:pointer;padding:15px;");
151     removeLink.append("Удалить");
152     removeLink.addEventListener("click", e => {
153
154         e.preventDefault();
155         DeleteUser(user.id);
156     });
157
158     linksTd.append(removeLink);
159     tr.appendChild(linksTd);
160
161     return tr;
162 }
163 // сброс значений формы
164 document.getElementById("reset").click(function (e) {
165
166     e.preventDefault();
167     reset();
168 })
169
170 // отправка формы
171 document.forms["userForm"].addEventListener("submit", e => {
172     e.preventDefault();
173     const form = document.forms["userForm"];
174     const id = form.elements["id"].value;
175     const name = form.elements["name"].value;
176     const age = form.elements["age"].value;
177     if (id == 0)
```

```
178         CreateUser(name, age);
179     else
180         EditUser(id, name, age);
181     });
182
183     // загрузка пользователей
184     GetUsers();
185 </script>
186 </body>
187 </html>
```

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

```
1  async function GetUsers() {
2      // отправляет запрос и получаем ответ
3      const response = await fetch("/api/users", {
4          method: "GET",
5          headers: { "Accept": "application/json" }
6      });
7      // если запрос прошел нормально
8      if (response.ok === true) {
9          // получаем данные
10         const users = await response.json();
11         let rows = document.querySelector("tbody");
12         users.forEach(user => {
13             // добавляем полученные элементы в таблицу
14             rows.append(row(user));
15         });
16     }
17 }
```

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:

```
1  async function GetUser(id) {
2      const response = await fetch("/api/users/" + id, {
3          method: "GET",
4          headers: { "Accept": "application/json" }
5      });
6      if (response.ok === true) {
7          const user = await response.json();
8          const form = document.forms["userForm"];
9          form.elements["id"].value = user.id;
10         form.elements["name"].value = user.name;
11         form.elements["age"].value = user.age;
```

```
12     }  
13 }
```

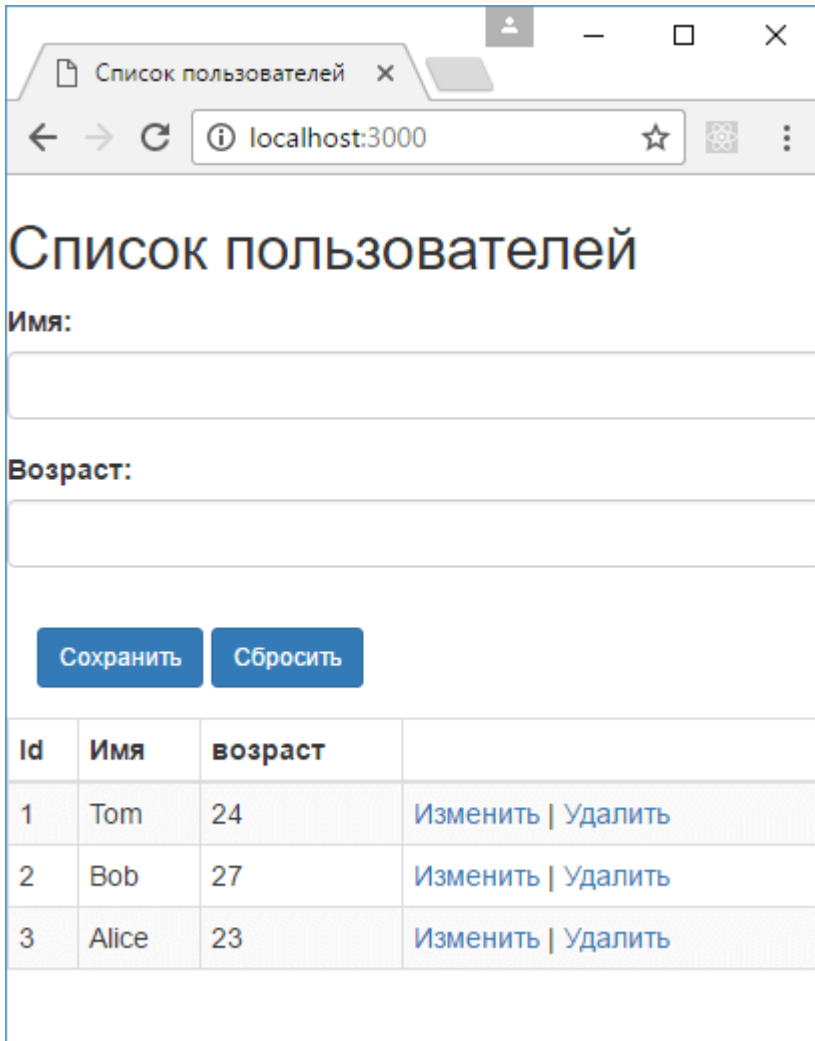
И выделенный пользователь добавляется в форму над таблицей. Эта же форма применяется и для добавления объекта. С помощью скрытого поля, которое хранит id пользователя, мы можем узнать, какое действие выполняется - добавление или редактирование. Если id равен 0, то выполняется функция CreateUser, которая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

```
1  async function CreateUser(userName, userAge) {  
2  
3      const response = await fetch("api/users", {  
4          method: "POST",  
5          headers: { "Accept": "application/json", "Content-Type": "applicatio  
6          body: JSON.stringify({  
7              name: userName,  
8              age: parseInt(userAge, 10)  
9          })  
10     });  
11     if (response.ok === true) {  
12         const user = await response.json();  
13         reset();  
14         document.querySelector("tbody").append(row(user));  
15     }  
16 }
```

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

```
1  async function EditUser(userId, userName, userAge) {  
2      const response = await fetch("api/users", {  
3          method: "PUT",  
4          headers: { "Accept": "application/json", "Content-Type": "applicatio  
5          body: JSON.stringify({  
6              id: userId,  
7              name: userName,  
8              age: parseInt(userAge, 10)  
9          })  
10     });  
11     if (response.ok === true) {  
12         const user = await response.json();  
13         reset();  
14         document.querySelector("tr[data-rowid='" + user.id + "']").replaceWi  
15     }  
16 }
```

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



Список пользователей

Имя:

Возраст:

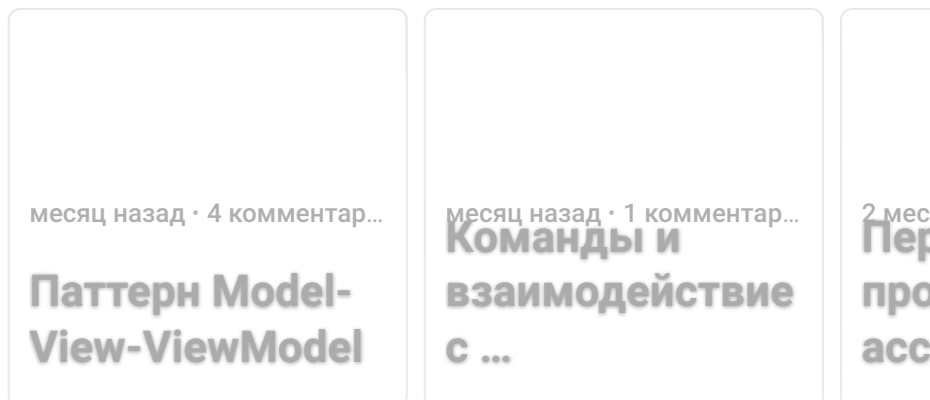
Сохранить Сбросить

Id	Имя	возраст	
1	Tom	24	Изменить   Удалить
2	Bob	27	Изменить   Удалить
3	Alice	23	Изменить   Удалить

[Назад](#) [Зміст](#) [Вперед](#)



ALSO ON METANIT.COM



месяц назад · 4 комментар...

**Паттерн Model-View-ViewModel**

месяц назад · 1 комментар...

**Команды и взаимодействие с ...**

2 мес

**Пер про асс...**