
Express і MongoDB

Тепер об'єднаємо в одному додатку обробку запитів за допомогою Express та роботу з даними MongoDB. Для цього визначимо наступний файл програми app.js:

```
1  const express = require("express");
2  const MongoClient = require("mongodb").MongoClient;
3  const ObjectId = require("mongodb").ObjectId;
4
5  const app = express();
6  const jsonParser = express.json();
7
8  const mongoClient = new MongoClient("mongodb://127.0.0.1:27017/");
9
10 app.use(express.static(`${__dirname}/public`));
11
12 (async () => {
13     try {
14         await mongoClient.connect();
15         app.locals.collection = mongoClient.db("usersdb").collection("users");
16         app.listen(3000);
17         console.log("Сервер ожидает подключения...");
18     } catch (err) {
19         return console.log(err);
20     }
21 }) ();
22
23 app.get("/api/users", async (req, res) => {
24
25     const collection = req.app.locals.collection;
26     try{
27         const users = await collection.find({}).toArray();
28         res.send(users);
29     }
30     catch(err) {
31         console.log(err);
32         res.sendStatus(500);
33     }
```

```
34 });
35 app.get("/api/users/:id", async(req, res) => {
36
37     const collection = req.app.locals.collection;
38     try{
39         const id = new ObjectId(req.params.id);
40         const user = await collection.findOne({_id: id});
41         if(user) res.send(user);
42         else res.sendStatus(404);
43     }
44     catch(err){
45         console.log(err);
46         res.sendStatus(500);
47     }
48 });
49
50 app.post("/api/users", jsonParser, async(req, res)=> {
51
52     if(!req.body) return res.sendStatus(400);
53
54     const userName = req.body.name;
55     const userAge = req.body.age;
56     const user = {name: userName, age: userAge};
57
58     const collection = req.app.locals.collection;
59
60     try{
61         await collection.insertOne(user);
62         res.send(user);
63     }
64     catch(err){
65         console.log(err);
66         res.sendStatus(500);
67     }
68 });
69
70 app.delete("/api/users/:id", async(req, res)=>{
71
72     const collection = req.app.locals.collection;
73     try{
74         const id = new ObjectId(req.params.id);
75         const result = await collection.findOneAndDelete({_id: id});
76         const user = result.value;
77         if(user) res.send(user);
78         else res.sendStatus(404);
79     }
80     catch(err){
81         console.log(err);
82         res.sendStatus(500);
```

```

83     }
84   });
85
86   app.put("/api/users", jsonParser, async(req, res)=>{
87
88     if(!req.body) return res.sendStatus(400);
89     const userName = req.body.name;
90     const userAge = req.body.age;
91
92     const collection = req.app.locals.collection;
93     try{
94       const id = new ObjectId(req.body.id);
95       const result = await collection.findOneAndUpdate({_id: id}, { $set:
96         {returnDocument: "after" }});
97
98       const user = result.value;
99       if(user) res.send(user);
100      else res.sendStatus(404);
101    }
102    catch(err){
103      console.log(err);
104      res.sendStatus(500);
105    }
106  });
107
108  // прослушиваем прерывание работы программы (ctrl-c)
109  process.on("SIGINT", async() => {
110
111    await mongoClient.close();
112    console.log("Приложение завершило работу");
113    process.exit();
114  });

```

Для каждого типа запросов здесь определен свой обработчик Express. И в каждом из обработчиков мы каждый раз обращаемся к базе данных. Чтобы не открывать и закрывать подключение каждый раз при каждом запросе, мы открываем подключение в самом начале в IIFE-функции и только после открытия подключения запускаем прослушивание входящих запросов:

```

1  (async () => {
2    try {
3      await mongoClient.connect();
4      app.locals.collection = mongoClient.db("usersdb").collection("users");
5      app.listen(3000);
6      console.log("Сервер ожидает подключения...");
7    } catch(err) {
8      return console.log(err);
9    }
10  }) ();

```

Поскольку все взаимодействие будет идти с коллекцией `users`, то получаем ссылку на эту коллекцию в локальную переменную приложения `app.locals.collection`. Затем через эту переменную мы сможем получить доступ к коллекции в любом месте приложения.

В конце работы скрипта мы можем закрыть подключение, сохраненное в переменную `dbClient`:

```
1 process.on("SIGINT", async() => {
2
3     await mongoClient.close();
4     console.log("Приложение завершило работу");
5     process.exit();
6 });
```

В данном случае мы прослушиваем событие "SIGINT", которое генерируется при нажатии комбинации CTRL+C в консоли, что завершит выполнение скрипта.

Когда приходит GET-запрос к приложению, то возвращаем в ответ клиенту все документы из базы данных:

```
1 app.get("/api/users", async(req, res) => {
2
3     const collection = req.app.locals.collection;
4     try{
5         const users = await collection.find({}).toArray();
6         res.send(users);
7     }
8     catch(err){
9         console.log(err);
10        res.sendStatus(500);
11    }
12 });
```

Если в GET-запросе передается параметр `id`, то возвращаем только одного пользователя из базы данных по этому `id`:

```
1 app.get("/api/users/:id", async(req, res) => {
2
3     const collection = req.app.locals.collection;
4     try{
5         const id = new ObjectId(req.params.id);
6         const user = await collection.findOne({_id: id});
7         if(user) res.send(user);
8         else res.sendStatus(404);
9     }
10    catch(err){
11        console.log(err);
12        res.sendStatus(500);
13    }
```

```
13     }  
14   });
```

Когда приходит POST-запрос, с помощью парсера `jsonParser` получаем отправленные данные и по ним создаем объект, который добавляем в базу данных посредством метода `insertOne()`:

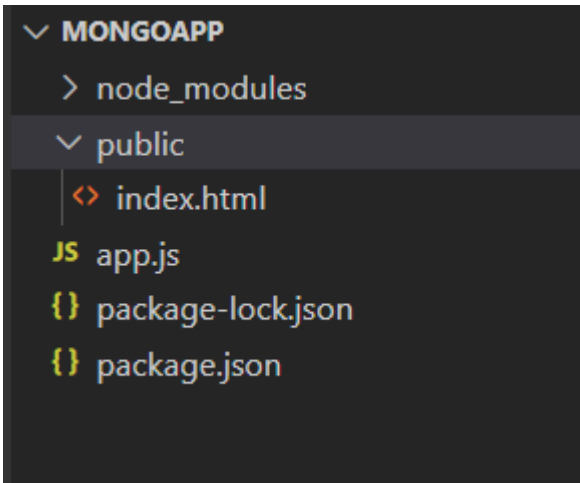
```
1  app.post("/api/users", jsonParser, async(req, res)=> {  
2  
3    if(!req.body) return res.sendStatus(400);  
4  
5    const userName = req.body.name;  
6    const userAge = req.body.age;  
7    const user = {name: userName, age: userAge};  
8  
9    const collection = req.app.locals.collection;  
10  
11   try{  
12     await collection.insertOne(user);  
13     res.send(user);  
14   }  
15   catch(err) {  
16     console.log(err);  
17     res.sendStatus(500);  
18   }  
19 });
```

При получении PUT-запроса также получаем отправленные данные и с помощью метода `findOneAndUpdate()` обновляем данные в БД.

И в методе `app.delete()`, который срабатывает при получении запроса DELETE, вызываем метод `findOneAndDelete()` для удаления данных.

Таким образом, в каждом обработчике Express задействуем определенный метод по работе с MongoDB.

Теперь создадим в папке проекта новый каталог "public" и определим в этом каталоге файл **index.html**:



В файле **index.html** определим 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='${user._id}']`).repl
97     }
98 }
99 // Удаление пользователя
100 async function deleteUser(id) {
101     const response = await fetch("/api/users/" + id, {
102         method: "DELETE",
103         headers: { "Accept": "application/json" }
104     });
105     if (response.ok === true) {
106         const user = await response.json();
107         document.querySelector(`tr[data-rowid='${user._id}']`).remc
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);
```



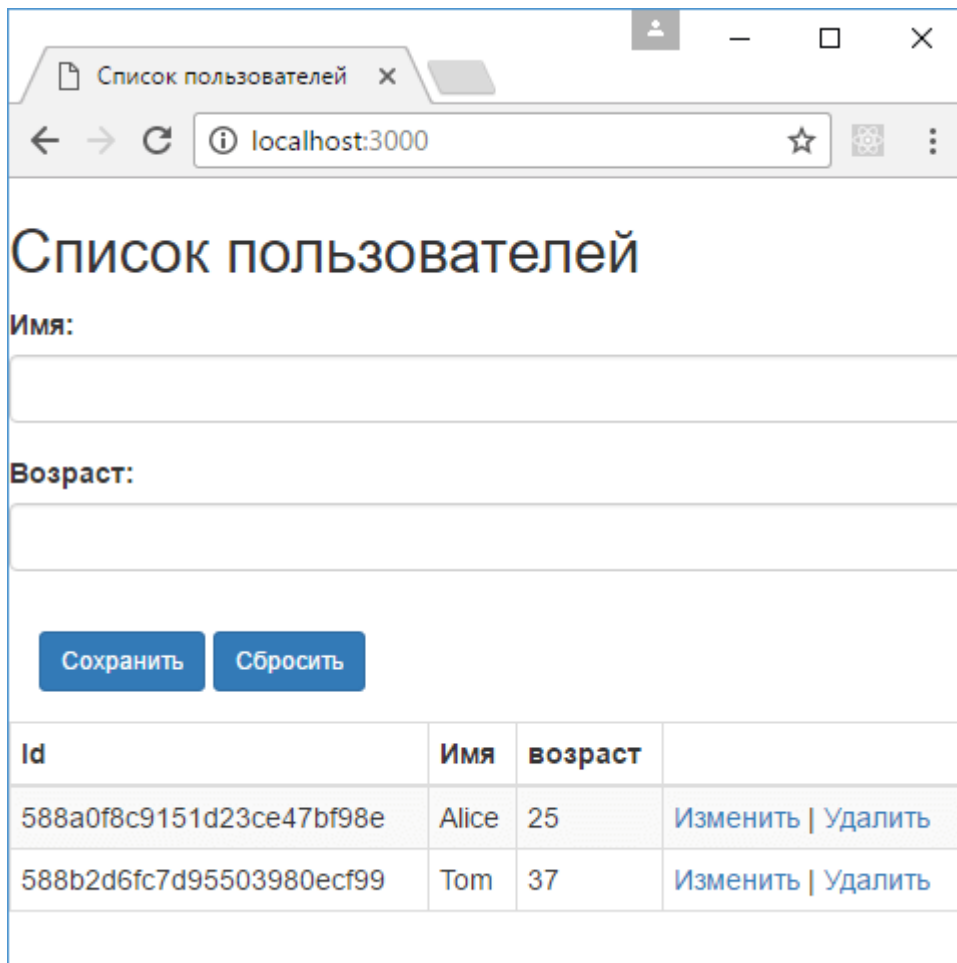
```
133     tr.append(ageTd);
134
135     const linksTd = document.createElement("td");
136
137     const editLink = document.createElement("a");
138     editLink.setAttribute("data-id", user._id);
139     editLink.setAttribute("style", "cursor:pointer;padding:15px;");
140     editLink.append("Изменить");
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").addEventListener("click", 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>
```

В принципе код index.html вкратце обсуждался в статье про создание [API в Node.js](#), здесь же весь код практически повторяется.

І оскільки Express як сховища статичних файлів використовує папку public, то при зверненні до додатку кореневим маршрутом `http://localhost:3000` клієнт отримує даний файл.

Запустимо програму, звернемося до програми за адресою `http://localhost:3000` і ми зможемо керувати користувачами, які зберігаються в базі даних MongoDB:



Id	Имя	возраст	
588a0f8c9151d23ce47bf98e	Alice	25	Изменить Удалить
588b2d6fc7d95503980ecf99	Tom	37	Изменить Удалить

[Назад](#) [Зміст](#) [Вперед](#)

