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:

```
const express = require("express");
const MongoClient = require("mongodb").MongoClient;
const objectId = require("mongodb").ObjectId;

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

const mongoClient = new MongoClient("mongodb://127.0.0.1:27017/");

app.use(express.static(`${__dirname}/public`));

(async () => {
     try {
        await mongoClient.connect();
        app.locals.collection = mongoClient.db("usersdb").collection("users
        app.listen(3000);
        console.log("Сервер ожидает подключения...");
    }catch(err) {
        return console.log(err);
    }
})();

app.get("/api/users", async(req, res) => {

    const collection = req.app.locals.collection;
    try{
        const users = await collection.find({}).toArray();
        res.send(users);
    }
    catch(err){
        console.log(err);
        res.sendStatus(500);
    }
```

```
});
app.get("/api/users/:id", async(req, res) => {

    const collection = req.app.locals.collection;
    try{
        const id = new objectId(req.params.id);
        const user = await collection.findOne({_id: id});
        if(user) res.send(user);
        else res.sendStatus(404);
    }
    catch(err){
        console.log(err);
        res.sendStatus(500);
    }
});

app.post("/api/users", jsonParser, async(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};

    const collection = req.app.locals.collection;

    try{
        await collection.insertOne(user);
        res.send(user);
    }
    catch(err){
        console.log(err);
        res.sendStatus(500);
    }
});

app.delete("/api/users/:id", async(req, res)=>{

    const collection = req.app.locals.collection;
    try{
        const id = new objectId(req.params.id);
        const result = await collection.findOneAndDelete({_id: id});
        const user = result.value;
        if(user) res.send(user);
        else res.sendStatus(404);
    }
    catch(err){
        console.log(err);
        res.sendStatus(500);
```

```
    }
});

app.put("/api/users", jsonParser, async(req, res)=>{

    if(!req.body) return res.sendStatus(400);
    const userName = req.body.name;
    const userAge = req.body.age;

    const collection = req.app.locals.collection;
    try{
        const id = new objectId(req.body.id);
        const result = await collection.findOneAndUpdate({_id: id}, { $set:
         {returnDocument: "after" });

        const user = result.value;
        if(user) res.send(user);
        else res.sendStatus(404);
    }
    catch(err){
        console.log(err);
        res.sendStatus(500);
    }
});

// прослушиваем прерывание работы программы (ctrl-c)
process.on("SIGINT", async() => {

    await mongoClient.close();
    console.log("Приложение завершило работу");
    process.exit();
});
```

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:

```
(async () => {
    try {
        await mongoClient.connect();
        app.locals.collection = mongoClient.db("usersdb").collection("users"
        app.listen(3000);
        console.log("Сервер ожидает подключения...");
    }catch(err) {
        return console.log(err);
    }
})();
```

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:

```
process.on("SIGINT", async() => {

    await mongoClient.close();
    console.log("Приложение завершило работу");
    process.exit();
});
```

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:

```
app.get("/api/users", async(req, res) => {

    const collection = req.app.locals.collection;
    try{
        const users = await collection.find({}).toArray();
        res.send(users);
    }
    catch(err){
        console.log(err);
        res.sendStatus(500);
    }
});
```

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

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

    const collection = req.app.locals.collection;
    try{
        const id = new objectId(req.params.id);
        const user = await collection.findOne({_id: id});
        if(user) res.send(user);
        else res.sendStatus(404);
    }
    catch(err){
        console.log(err);
        res.sendStatus(500);
```

```
    }
});
```

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

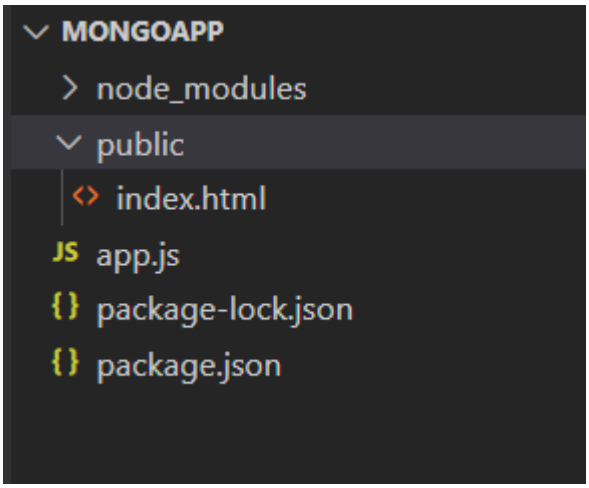
```
app.post("/api/users", jsonParser, async(req, res)=> {

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

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

    const collection = req.app.locals.collection;

    try{
        await collection.insertOne(user);
        res.send(user);
    }
    catch(err){
        console.log(err);
        res.sendStatus(500);
    }
});
```

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:

```
<!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>

    <script>
    // 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();
            let rows = document.querySelector("tbody");
            users.forEach(user => {
                // добавляем полученные элементы в таблицу
                rows.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": "a
            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));
        }
    }
    // 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='${user._id}']`).repl
            }
        }
        // Удаление пользователя
        async function deleteUser(id) {
            const response = await fetch("/api/users/" + id, {
                method: "DELETE",
                headers: { "Accept": "application/json" }
            });
            if (response.ok === true) {
                const user = await response.json();
                document.querySelector(`tr[data-rowid='${user._id}']`).remo
            }
        }

        // 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);
```

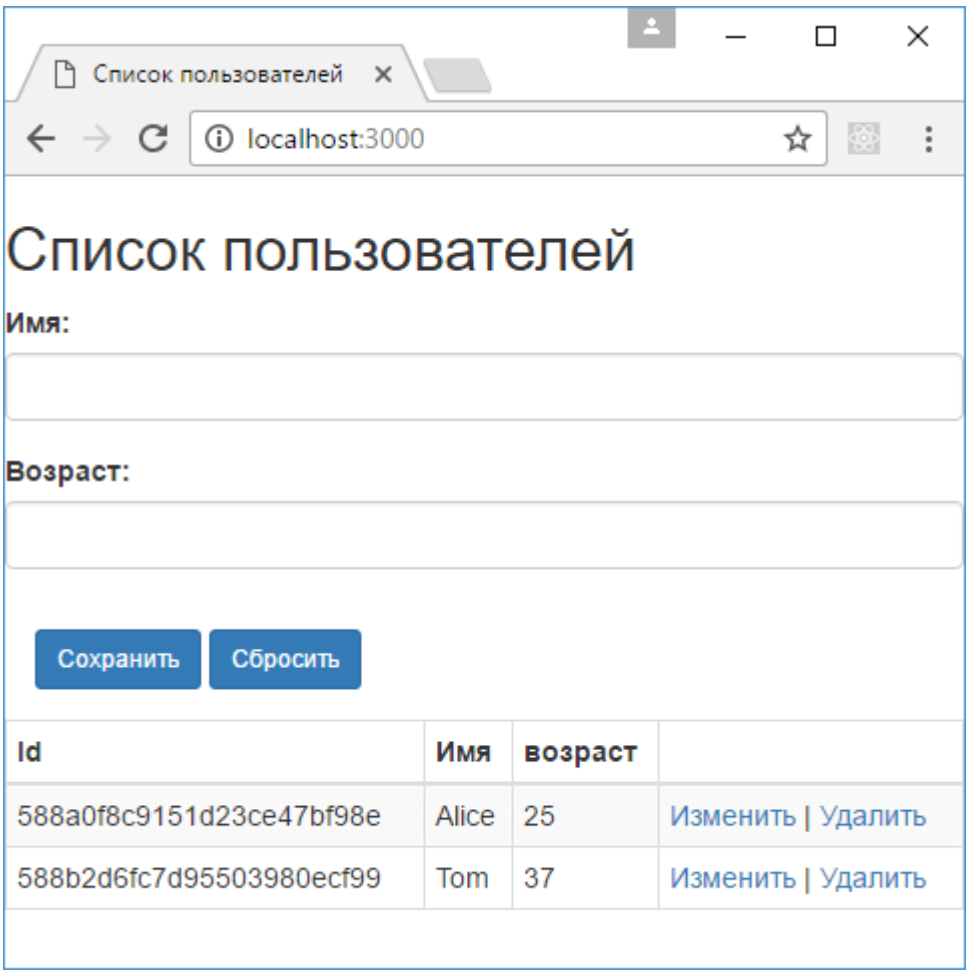
```
            tr.append(ageTd);

            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").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);
        });
```

```
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 отримає даний файл.

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



Назад Зміст Вперед