

<b>ДИСЦИПЛИНА</b>	Фронтенд и бэкенд разработка
<b>ИНСТИТУТ</b>	Институт перспективных технологий и индустриального программирования
<b>КАФЕДРА</b>	Кафедра индустриального программирования
<b>ВИД УЧЕБНОГО МАТЕРИАЛА</b>	Методические указания к практическим занятиям
<b>ПРЕПОДАВАТЕЛЬ</b>	Астафьев Рустам Уралович
<b>СЕМЕСТР</b>	4 семестр, 2023-2024 учебный год

## Практическое занятие №8

### Применение MongoDB

Обратите внимание, что базовым примером для этого занятия является выполнение текущей инструкции.

Итак, для изучения интеграции MongoDB в проект на Node.js у вас должен быть запущен сервер баз данных.

При подключении и взаимодействии с БД в MongoDB можно выделить следующие этапы:

- Подключение к серверу
- Получение объекта базы данных на сервере
- Получение объекта коллекции в базе данных
- Взаимодействие с коллекцией (добавление, удаление, получение, изменение данных)

1. Подключение к серверу баз данных. Создадим новый проект Node.js. Для этого создадим в папке пользователя новую папку, которая будет называться `mongoapp`. Далее определим в этой папке новый файл `package.json`:

```
{
  "name": "mongoapp",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.17.0",
    "mongodb": "^5.4.0"
  }
}
```

2. Откроем командную строку и выполним последовательно 2 команды:

```
cd mongoapp
npm install
```

```
C:\Users\Alexander>cd mongoapp
C:\Users\Alexander\mongoapp>npm install
added 73 packages, and audited 74 packages in 18s
7 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
C:\Users\Alexander\mongoapp>
```

3. Ключевым объектом для работы с MongoDB является объект MongoClient, и через него будет идти все взаимодействия с хранилищем данных. Соответственно вначале мы должны получить возможность работать с MongoClient. Для этого в папке mongoapp создадим пустой файл app.js и добавим в него следующий код, необходимый нам для подключения к БД

```
const MongoClient = require("mongodb").MongoClient;
const url = "mongodb://127.0.0.1:27017/";
const mongoClient = new MongoClient(url);
async function run() {
  try {
    await mongoClient.connect();
    const db = mongoClient.db("usersdb");
    const collection = db.collection("users");
    const count = await collection.countDocuments();
    console.log(`В коллекции users ${count} документа/ов`);
  } catch(err) {
    console.log(err);
  } finally {
    await mongoClient.close();
  }
}
run().catch(console.error);
```

В качестве базы данных здесь используется "usersdb". При этом не важно, что по умолчанию на сервере MongoDB нет подобной базы данных. При первом к ней обращении сервер автоматически ее создаст.

После подключения мы обращаемся к коллекции "users". Опять же неважно, что такой коллекции по умолчанию нет в бд usersdb, она также будет создана при первом обращении.

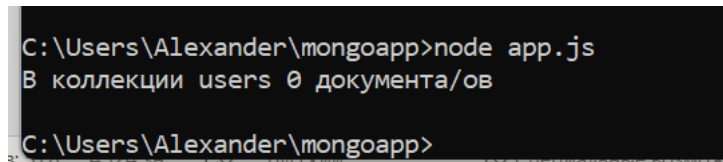
Получив коллекцию, мы можем использовать ее методы. В данном случае для простоты применяется метод countDocuments(), который получает

количество документов коллекции. Этот метод возвращает Promise, из которого можно получить собственно результат операции - количество документов.

Для запуска нашего app.js приложения выполним в командной строке:

```
node app.js
```

Результат должен быть такой:



```
C:\Users\Alexander\mongoapp>node app.js
В коллекции users 0 документа/ов
C:\Users\Alexander\mongoapp>
```

База данных пустая, поэтому количество документов – нулевое

4. Добавление, изменение, удаление данных в БД MongoDB производится с помощью функций, которые были рассмотрены нами в предыдущий раз.

Приведем некоторые примеры использования этих функций.

5. Для добавления мы можем использовать различные методы. Если нужно добавить один объект, то применяется метод insertOne(). При добавлении набора объектов можно использовать метод insertMany().

Для добавления одного документа применяется метод insertOne(). В качестве параметра он принимает добавляемый документ. Рассмотрим на примере. Заменим весь код файла app.js следующим:

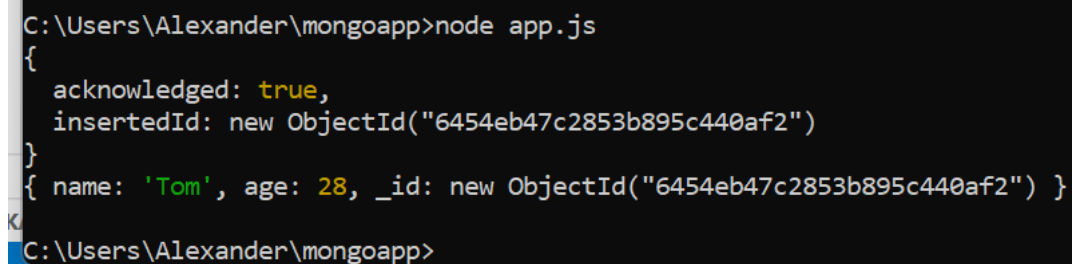
```
const MongoClient = require("mongodb").MongoClient;
const url = "mongodb://127.0.0.1:27017/";
const mongoClient = new MongoClient(url);
async function run() {
  try {
    await mongoClient.connect();
    const db = mongoClient.db("usersdb");
    const collection = db.collection("users");
    const user = {name: "Tom", age: 28};
    const result = await collection.insertOne(user);
    console.log(result);
    console.log(user);
  } catch(err) {
    console.log(err);
  } finally {
    await mongoClient.close();
  }
}
run().catch(console.error);
```

Добавляемый документ представляет объект. В данном случае это объект {name: "Tom", age: 28};

То есть в коллекцию users будет добавляться документ, который имеет для свойства: name и age, которые хранят соответственно строку "Tom" и число 23.

После добавления получаем результат операции и выводим его на консоль (после выполнения команды node app.js). Также выводим на консоль объект добавленный user, чтобы посмотреть какие изменения произошли с ним после добавления в бд.

Результат работы программы:



```
C:\Users\Alexander\mongoapp>node app.js
{
  acknowledged: true,
  insertedId: new ObjectId("6454eb47c2853b895c440af2")
}
{ name: 'Tom', age: 28, _id: new ObjectId("6454eb47c2853b895c440af2") }
C:\Users\Alexander\mongoapp>
```

Здесь мы можем увидеть результат операции, в частности, с помощью свойства result. insertedId получить идентификатор добавленного объекта. Кроме того, мы видим, кроме начальных свойств документ user после добавления в БД получили дополнительное свойство \_id - это уникальный идентификатор документа, который присваивается сервером при добавлении.

6. Добавление нескольких объектов. Теперь используем метод insertMany().

Добавим набор объектов и для этого изменим файл приложения app.js:

```
const MongoClient = require("mongodb").MongoClient;
const url = "mongodb://127.0.0.1:27017/";
const mongoClient = new MongoClient(url);
const users = [{name: "Bob", age: 35} , {name: "Alice", age: 21}, {name:
"Tom", age: 45}];
async function run() {
  try {
    await mongoClient.connect();
    const db = mongoClient.db("usersdb");
    const collection = db.collection("users");
    const results = await collection.insertMany(users);
    console.log(results);
    console.log(users);
  } catch(err) {
    console.log(err);
  } finally {
    await mongoClient.close();
  }
}
run().catch(console.error);
```

Как и `insertOne`, метод `insertMany()` в качестве параметра принимает добавляемые данные - массив объектов, а в качестве результата возвращает результат операции. При удачном добавлении добавленные объекты из массива `users` аналогичным образом получают свойство `_id`.

Запустим приложение:

```
C:\Users\Alexander\mongoapp>node app.js
{
  acknowledged: true,
  insertedCount: 3,
  insertedIds: {
    '0': new ObjectId("6454ebdb2a595e6ec1b65ad8"),
    '1': new ObjectId("6454ebdb2a595e6ec1b65ad9"),
    '2': new ObjectId("6454ebdb2a595e6ec1b65ada")
  }
}
[
  {
    name: 'Bob',
    age: 35,
    _id: new ObjectId("6454ebdb2a595e6ec1b65ad8")
  },
  {
    name: 'Alice',
    age: 21,
    _id: new ObjectId("6454ebdb2a595e6ec1b65ad9")
  },
  {
    name: 'Tom',
    age: 45,
    _id: new ObjectId("6454ebdb2a595e6ec1b65ada")
  }
]
C:\Users\Alexander\mongoapp>
```

7. Получение данных в MongoDB. Для получения данных из коллекции применяется метод `find()`. Для его демонстрации заменим `app.js` следующим кодом

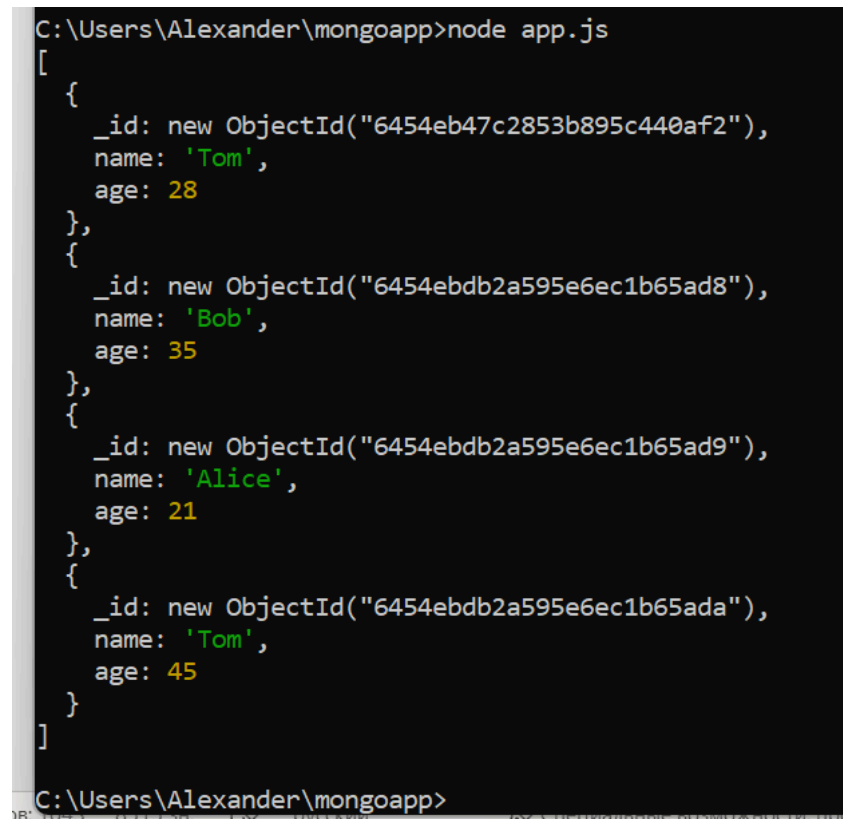
```
const MongoClient = require("mongodb").MongoClient;
const url = "mongodb://127.0.0.1:27017/";
const mongoClient = new MongoClient(url);
async function run() {
  try {
    await mongoClient.connect();
    const db = mongoClient.db("usersdb");
    const collection = db.collection("users");
    const results = await collection.find().toArray();
    console.log(results);
  } catch (err) {
```

```

        console.log(err);
    } finally {
        await mongoClient.close();
    }
}
run().catch(console.error);

```

Метод `find` возвращает специальный объект `FindCursor`, и чтобы получить все данные у этого объекта вызывается метод `toArray()`. И если мы запустим приложение, то увидим все ранее добавленные данные:



```

C:\Users\Alexander\mongoapp>node app.js
[
  {
    _id: new ObjectId("6454eb47c2853b895c440af2"),
    name: 'Tom',
    age: 28
  },
  {
    _id: new ObjectId("6454ebdb2a595e6ec1b65ad8"),
    name: 'Bob',
    age: 35
  },
  {
    _id: new ObjectId("6454ebdb2a595e6ec1b65ad9"),
    name: 'Alice',
    age: 21
  },
  {
    _id: new ObjectId("6454ebdb2a595e6ec1b65ada"),
    name: 'Tom',
    age: 45
  }
]
C:\Users\Alexander\mongoapp>

```

8. С помощью метода `find()` мы можем дополнительно отфильтровать извлекаемые документы. Например, нам надо найти всех пользователей, у которых имя – Том. Заменяем код `app.js` следующим кодом:

```

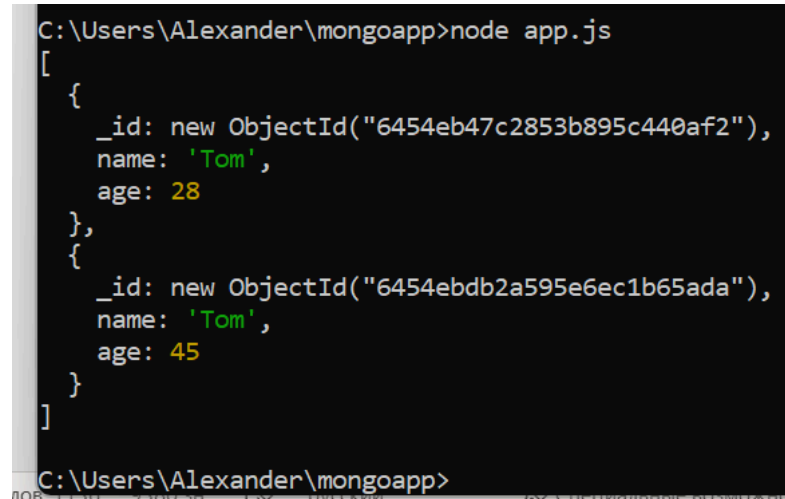
const MongoClient = require("mongodb").MongoClient;
const url = "mongodb://127.0.0.1:27017/";
const mongoClient = new MongoClient(url);

async function run() {
  try {
    await mongoClient.connect();
    const db = mongoClient.db("usersdb");
    const collection = db.collection("users");
    const results = await collection.find({name: "Tom"}).toArray();
    console.log(results);
  } catch(err) {
    console.log(err);
  } finally {
    await mongoClient.close();
  }
}

```

```
}  
run().catch(console.error);
```

В качестве параметра в `find` передается объект, который устанавливает параметры фильтрации. В частности, что свойство `name` должно быть равно "Tom"

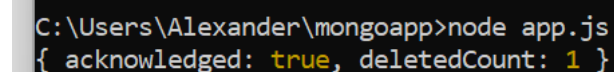


```
C:\Users\Alexander\mongoapp>node app.js  
[  
  {  
    _id: new ObjectId("6454eb47c2853b895c440af2"),  
    name: 'Tom',  
    age: 28  
  },  
  {  
    _id: new ObjectId("6454ebdb2a595e6ec1b65ada"),  
    name: 'Tom',  
    age: 45  
  }  
]  
C:\Users\Alexander\mongoapp>
```

9. Удалять документы в MongoDB можно различными способами. Метод `deleteOne()` удаляет только объект. Изменим код `app.js`

```
const MongoClient = require("mongodb").MongoClient;  
  
const url = "mongodb://127.0.0.1:27017/";  
const mongoClient = new MongoClient(url);  
async function run() {  
  try {  
    await mongoClient.connect();  
    const db = mongoClient.db("usersdb");  
    const collection = db.collection("users");  
    const result = await collection.deleteOne({name: "Bob"});  
    console.log(result);  
  
  } catch(err) {  
    console.log(err);  
  } finally {  
    await mongoClient.close();  
  }  
}  
run().catch(console.error);
```

В качестве параметра метод `deleteOne()` принимает фильтр документов, которые надо удалить. Возвращает метод результат операции удаления. При этом результат удаления будет представлять объект, в котором с помощью свойства `deletedCount` можно увидеть количество удаленных документов.



```
C:\Users\Alexander\mongoapp>node app.js  
{ acknowledged: true, deletedCount: 1 }
```



10.Метод `deleteMany()` работает аналогичным образом, только удаляет все документы, которые соответствуют фильтру. Например, удалить всех пользователей, у которых имя "Tom", можно командой `const result = await collection.deleteMany({name: "Tom"});`

11.Метод `drop()` удаляет всю коллекцию.

12.Метод `updateMany()` позволяет обновить все документы из коллекции, которые соответствуют критерию фильтрации:

```
const MongoClient = require("mongodb").MongoClient;
const url = "mongodb://127.0.0.1:27017/";
const mongoClient = new MongoClient(url);
async function run() {
  try {
    await mongoClient.connect();
    const db = mongoClient.db("usersdb");
    const collection = db.collection("users");
    const result = await collection.updateMany({name: "Tom"}, { $set:
{name: "Bob"}});
    console.log(result);

  } catch(err) {
    console.log(err);
  } finally {
    await mongoClient.close();
  }
}
run().catch(console.error);
```

Во всех документах, где есть имя «Том», оно заменяется на «Боб»

13. Рассмотрим возможности использования базы данных MongoDB в экосистеме Node.js. Для этого объединим в одном приложении обработку запросов с помощью Express и работу с данными в MongoDB. Пример, приведенный далее, является достаточно сложным и в комплексе описывает работу с различными запросами к бэкенду. Изменим файл приложения `app.js`:

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

const app = express();
const bodyParser = 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:
{age: userAge, name: userName}},
        {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();
});

```

Проанализируем данный код.

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 и только после открытия подключения запускаем прослушивание входящих запросов. 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:

```

(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);
    }
  }) ();

```

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.

```

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:
{age:userAge, name: userName}},
            {returnDocument: "after" });

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

```

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

```

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);
    }
});

```

Теперь создадим в папке проекта новую папку "public" и создадим в этой папке файл index.html. В файл index.html добавим 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/bootstrap.min.css"
rel="stylesheet" />
</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</button>
      <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></thead>
  <tbody>
  </tbody>
</table>

<script>
// Получение всех пользователей
async function getUsers() {
  // отправляет запрос и получаем ответ
  const response = await fetch("/api/users", {
    method: "GET",
    headers: { "Аccept": "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":
"application/json" },
        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":
"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"];
        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);
    });

    // загрузка пользователей
    getUsers();
</script>
</body>
</html>

```

Этот файл представляет собой форму для ввода и редактирования данных (элемент `<form>`), а также таблицу (элемент `<table>`), в которой отображаются данные, хранящиеся в базе данных. Кроме того, значительную часть файла занимает скрипт. Данный скрипт представляет собой API (описание способа взаимодействия страницы `index.html` с сервером Node.js и базой данных MongoDB).

Поскольку Express в качестве хранилища статических файлов использует папку `public`, то при обращении к приложению по корневому маршруту `http://localhost:3000` клиент получит данный файл (т.е. автоматически в браузере отобразится файл `index.html`).

Запустим приложение, обратимся к приложению по адресу `http://localhost:3000` и мы сможем управлять пользователями, которые хранятся в базе данных MongoDB:

Id	Имя	возраст	Изменить	Удалить
6454eb47c2853b895c440af2	Bob	28	Изменить	Удалить
6454ebdb2a595e6ec1b65ad9	Alice	21	Изменить	Удалить
6454ebdb2a595e6ec1b65ada	Bob	45	Изменить	Удалить