Использование Node.js для работы с СУБД MongoDB.

Ниже представлены примеры работы скриптов Node.js для работы с данными MongoDB. Первоначальный недостаток знаний по языку Node.js можно устранить самостоятельно на сайте https://metanit.com/web/nodejs/.

! Выделенным шрифтом даны задания, которые в обязательном порядке нужно выполнить самостоятельно.

1. Введение.

В этой лабораторной работе рассмотрим не примерах, как можно взаимодействовать с СУБД MongoDB из среды Node.js, начиная от вывода данных в окно терминала до создания полноценного клиент-серверного web-приложения. Для выполнения работы необходимо иметь на клиентском компьютере среду Node.js версии >20, удобный редактор для работы с JavaScrypt (например, *VS Code*), браузер с возможностью отладки JavaScript (например, *Google Chrome*), клиент *MongoDB Compass* (либо *Mongosh*), программу *Postman* для отладки API-приложений (не обязательно), а также иметь логин и пароль для доступа к серверу MongoDB.

2. Простое приложение Node.js, извлекающее данные из MongoDB.

Пошаговая инструкция.

1) Создайте папку проекта в удобном для работы месте (ни в коем случае не на сетевом диске, лучше всего в папке «Мои документы»), зайдите в созданную папку из редактора VS Code, и запустите внутри редактора системный терминал (в компьютерных классах используйте вызов оболочки *cmd*, а не *PowerShell*). В терминале введите строку `npm init`, которая инициализирует новый проект Node.js. Во время инициализации вы можете ответить на предлагаемые вопросы либо оставить ответы по умолчанию. В представленных далее примерах все действия выполняются от лица пользователя с логином *std-01*.

2) Установите локально MongoDB драйвер командой `npm install mongodb`.

3) Посмотрите содержание файла package.json. Оно пока может быть произвольным и напоминать показанное ниже:

```
{
 "name": "mongo-console-app",
 "version": "1.0.0",
 "description": "node.js + mingoDB",
 "license": "ISC",
 "author": "std-01",
 "type": "commonjs",
 "main": "app.js",
 "scripts": {
  "start": "node app.js",
  "test": "echo \"Без тестов\" && exit 1"
```

```
 },
 "dependencies": {
  "mongodb": "^5.0.0"
 }
}
```

4) Создадим в папке проекта файл конфигурации проекта **config.js**, в котором разместим строки подключения к базе данных:

```
const DBConfig = {
  uri: "mongodb://std-01:пароль@172.20.1.176:27017/?authSource=std-01",
  dbName: "std-01",
  collectionName: "testusers"
};
module.exports = DBConfig;
```

В вашем случае строка подключения uri будет такой, какую вы используете при подключении *MongoDB Compass* к вашей сетевой базе данных. Везде *std-01* заменяете на свой логин.

5) Создаём тестовую коллекцию данных в MongoDB. Для этого открываем *MongoDB Compass*, подключаемся к серверу баз данных, открываем окно *Mongosh*, в нём переходим в свою базу данных и добавляем тестовые данные так, как примерно представлено ниже:

```
db.testusers.insertMany([
    {
        name: "Иван Иванов",
        age: 25,
        email: "ivan@example.com",
        city: "Москва",
        createdAt: new Date()
    },
    {
        name: "Мария Петрова",
        age: 30,
        email: "maria@example.com",
        city: "Санкт-Петербург",
        hobbies: ["чтение", "путешествия"],
        createdAt: new Date()
    },
    {
        name: "Алексей Сидоров",
        age: 22,
        email: "alex@example.com",
        city: "Новосибирск",
        isStudent: true,
        createdAt: new Date()
    },
    {
        name: "Екатерина Волкова",
        age: 28,
        email: "ekaterina@example.com",
        city: "Екатеринбург",
        skills: ["JavaScript", "Node.js", "MongoDB"],
        createdAt: new Date()
    },
    {
        name: "Дмитрий Козлов",
```

```
        age: 35,
        email: "dmitry@example.com",
        city: "Казань",
        salary: 50000,
        createdAt: new Date()
    }
]);
```

Проверим, что данные добавлены, с помощью запроса: `db.testusers.find()`.

6) Создаём основное приложение. Для этого в текстовом редакторе создаём файл **app.js** в каталоге проекта и записываем в него следующий код на языке Node.js:

```
const { MongoClient } = require('mongodb');
const config = require('./config.js');

async function mongoReader() {

  console.log('Подключаемся к MongoDB...\n');
  const client = new MongoClient(config.uri);

  try {
    await client.connect();
    console.log('✅ Подключение успешно\n');

    const db = client.db(config.dbName);
    const collection = db.collection(config.collectionName);

    // Формулируем запрос к базе данных и получаем курсор
    const documents = await collection.find({}).toArray();

    console.log(`Найдено ${documents.length} документов в коллекции "$
{config.collectionName}":\n`);

    // Вывод информации в консоль: обрабатываем курсор
    documents.forEach((doc, index) => {
      console.log(`📄 Документ ${index + 1}:`);
      console.log(`   ID: ${doc._id}`);
      console.log(`   Имя: ${doc.name || 'Не указано'}`);
      console.log(`   Возраст: ${doc.age || 'Не указан'}`);
      console.log(`   Город: ${doc.city || 'Не указан'}`);
      console.log(`   Email: ${doc.email || 'Не указан'}`);
      // обрабатываем массивы
      if (doc.hobbies) {
        console.log(`   Хобби: ${doc.hobbies.join(', ')}`);
      }
      if (doc.skills) {
        console.log(`   Навыки: ${doc.skills.join(', ')}`);
      }

      console.log('---');
    });
```

```
  } catch (error) {
    console.error('❌ Ошибка:', error.message);
  } finally {
    await client.close();
    console.log('\n Соединение закрыто.');
  }
}

// Тестируем функцию
mongoReader();
```

7) Запускаем приложение из терминала командой `node app.js`. Должен получиться вывод, похожий на представленный ниже:

```
Подключаемся к MongoDB...

✅ Подключение успешно

Найдено 5 документов в коллекции "testusers":

📄 Документ 1:
   ID: 697ccb145f8227500a2a116e
   Имя: Иван Иванов
   Возраст: 25
   Город: Москва
   Email: ivan@example.com
---
📄 Документ 2:
   ID: 697ccb145f8227500a2a116f
   Имя: Мария Петрова
   Возраст: 30
   Город: Санкт-Петербург
   Email: maria@example.com
   Хобби: чтение, путешествия
---
📄 Документ 3:
   ID: 697ccb145f8227500a2a1170
   Имя: Алексей Сидоров
   Возраст: 22
   Город: Новосибирск
   Email: alex@example.com
---
📄 Документ 4:
   ID: 697ccb145f8227500a2a1171
   Имя: Екатерина Волкова
   Возраст: 28
   Город: Екатеринбург
   Email: ekaterina@example.com
   Навыки: JavaScript, Node.js, MongoDB
---
📄 Документ 5:
   ID: 697ccb145f8227500a2a1172
   Имя: Дмитрий Козлов
   Возраст: 35
   Город: Казань
   Email: dmitry@example.com
---

 Соединение закрыто.
```

Попробуйте получить то же самое, выполнив команду `npm start`. Задача выполнена.

3. Задание для самостоятельной работы.

! В базе данных *sample_mflix* на сервере MongoDB в коллекции *movies* выберите 3 кинокартины, созданные в 200X году, где X — номер вашего варианта, которые получили максимальное количество наград (*awards/wins*). Напишите приложение на Node.js, которое выведет полученную информацию в консоль не в виде таблицы, а в виде текста приблизительно такого содержания: «Из фильмов 2011 года тонким ценителям киноискусства мы советовали бы посмотреть прежде всего картину 'The Artist' (режиссёр Michel Hazanavicius), получившую 161 премию, а также фильм 'The Tree of Life' (режиссёр Nicolas Winding Refn, 88 премий), ну и, возможно, фильм 'Drive' (режиссёр Nicolas Winding Refn, 88 премий).» Проверьте работу вашей программы для других лет.

4. Создаём web-сервер.

Создадим сначала несколько простых проектов, чтобы понять основные принципы разработки web-сервера с использованием Node.js.

4.1. Простейший web-сервер на «чистом» Node.js (без использования фреймворков).

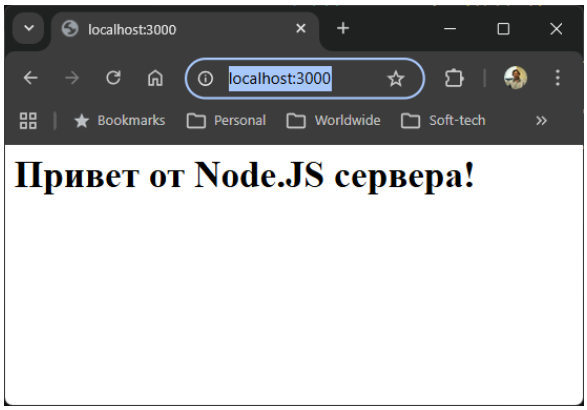
- Создайте папку проекта в удобном для работы месте, зайдите в эту папку из редактора VS Code, и запустите внутри редактора системный терминал. В терминале введите строку `npm init`, которая инициализирует новый проект Node.js. Во время инициализации вы можете ответить на предлагаемые вопросы либо оставить ответы по умолчанию.
- Создайте файл с именем **app1.js** следующего содержания:

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html; charset=utf-8');
  res.end('<h1>Привет от Node.JS сервера!</h1>');
});

server.listen(3000, () => {
  console.log('Сервер запущен на http://localhost:3000');
});
```

Для организации работы web-сервера в Node.js используется модуль (класс) *http*. Для создания сервера используется его метод *http.createServer()*. Параметром является функция-обработчик (оформленная в виде лямбда-функции), принимающая два параметра: *req[uest]* хранит информацию о запросе к серверу со стороны клиента, *res[ponse]* управляет отправкой ответа от сервера к клиенту. Чтобы сервер мог прослушивать и обрабатывать входящие подключения, у объекта сервера необходимо вызвать метод *listen()*, у которого в качестве

первого параметра используется номер порта. Запускается сервер командой **node app1.js** из командной строки терминала. После этого сервер остаётся в запущенном состоянии, выводя сообщения в консоль с помощью метода *console.log()*, что удобно использовать для поиска багов. Для остановки сервера достаточно нажать клавиши Ctrl-C в окне терминала, из которого был запущен сервер.
Подробное объяснение последующих кодов заняло бы слишком много места в данном практическом руководстве, поэтому для более углублённого разбора кода слушайте лекции по соответствующей теме и самостоятельно изучайте Node.js, например, по этому онлайн-пособию: https://metanit.com/web/nodejs/.

- После того, как сервер будет запущен, откройте в Интернет-браузере адрес http://localhost:3000. Вы увидите следующий результат:



4.2. Использование фреймворка Express при разработке web-сервера.

Express представляет собой популярный веб-фреймворк, написанный на JavaScript и работающий внутри среды исполнения Node.js. Он упрощает создание серверных приложений и веб-сайтов, работая поверх встроенного модуля http. Ключевые особенности фреймворка: определение поведения приложения на основе URL и HTTP-методов и применение функций, обрабатывающих запросы до передачи их маршрутам.

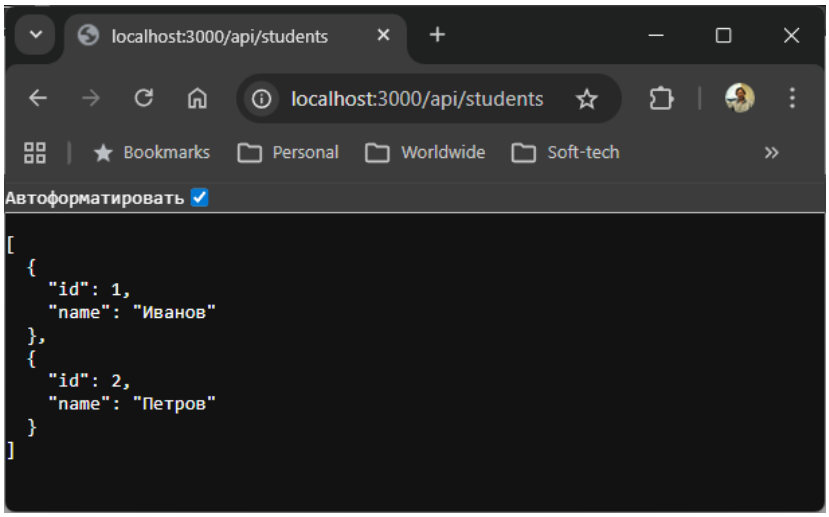
- Будем использовать каталог проекта из предыдущего примера. В окне терминала с помощью команды **npm install express** установим фреймворк.
- Создадим файл **app2.js** следующего содержания:

```
const express = require('express');
const app = express();

studData=[{id: 1, name: 'Иванов'}, {id: 2, name: 'Петров'}];

app.get('/', (req, res) => res.send('Здесь может быть ваша web-страница'));
app.get('/api/students', (req, res) => res.json(studData));

app.listen(3000, () => console.log('API запущено!'));
```

С помощью Express легко создавать *Application Programming Interface* (API), определяя endpoints (конечные точки) – конкретные URL-адреса, по которым клиентское приложение обращается к серверу для выполнения определенной операции: получения, создания,

обновления или удаления данных. В данном случае мы обслуживаем наиболее часто используемый метод GET протокола *http*.

- **! Запустите сервер app2.js и обратитесь к нему по соответствующим точкам входа:**



5. Создаём web-сервер с html-клиентом и полноценной поддержкой СУБД MongoDB.

Следующее приложение будет устроено более сложным образом. После запуска примера внимательно изучите представленный код, чтобы взять его за основу для выполнения итогового задания по данной теме. В этом проекте не используется фреймворк Express, поскольку оно поддерживает лишь несколько примитивных операций и вполне может быть реализовано базовыми средствами Node.js. Сложная структура каталогов в проекте также будет избыточной, все файлы будут находиться в одном каталоге.

Пошаговая инструкция.

1) Создайте папку проекта в удобном для работы месте, зайдите в эту папку из редактора VS Code, и запустите внутри редактора системный терминал. В терминале введите строку `npm init -y`, которая инициализирует новый проект Node.js. Будет создан файл *package.json*, содержимое которого мы отредактируем позже. В представленных примерах, как обычно, все действия выполняются от лица пользователя с логином *std-01*.

2) Установите локально MongoDB драйвер командой `npm install mongodb`. Также установите модуль *dotenv*, используемый для безопасного хранения конфиденциальных данных, командой `npm install dotenv`. Все настройки проекта (в первую очередь логин и пароль базы данных) теперь будет удобно хранить в «секретном» файле *.env*.

3) Создайте файл с именем *.env* с содержанием, подобным следующему:

```
PORT=3000
MONGODB_URL=mongodb://std-01:пароль@172.20.1.176:27017/?authSource=std-01
DB_NAME=std-01
```

4) Используя MongoDB Compass, создайте в вашей базе данных коллекцию students приблизительно таким образом:

```
db.students.insertMany([
 {
  name: "Иванов Иван Иванович",
  age: 20,
  email: "ivan@example.com",
  course: "Информатика",
  gpa: 4.5,
  enrollmentDate: new Date("2023-09-01"),
  active: true
 },
 {
  name: "Петрова Мария Сергеевна",
  age: 21,
  email: "maria@example.com",
  course: "Математика",
  gpa: 4.7,
  enrollmentDate: new Date("2023-09-01"),
  active: true
 },
 {
  name: "Сидоров Трофим Сергеевич",
  age: 22,
  email: "trofim@example.com",
  course: "Физика",
  gpa: 4.2,
  enrollmentDate: new Date("2023-09-01"),
  active: false
 },
 {
  name: "Шакалова Зверолюба Игнатьевна",
  age: 19,
  email: "luba@example.com",
  course: "Химия",
  gpa: 4.8,
  enrollmentDate: new Date("2023-09-01"),
  hobbies: ["чтение", "спорт"],
  active: true
 },
 {
  name: "Козлов Дмитрий Сергеевич",
  age: 23,
  email: "dmitry@example.com",
  course: "Биология",
  gpa: 3.9,
  enrollmentDate: new Date("2023-09-01"),
  active: true
 }
]);
```

5) Создаём файл *server.js*, в котором будет находиться весь серверный код. Разбор этого файла будет проведён на лекции. Ниже дано содержание этого файла:

```
// сервер node.js

const http = require('http');
const url = require('url');
const fs = require('fs');
const { MongoClient, ObjectId } = require('mongodb');
require('dotenv').config();

// Конфиденциальные данные
const PORT = process.env.PORT;
const MONGODB_URL = process.env.MONGODB_URL;
const DB = process.env.DB_NAME;

// Подключаемся к MongoDB (при старте сервера)
let db;
let studentsCollection;

async function connectToMongoDB() {
  const client = new MongoClient(MONGODB_URL);
  await client.connect();
  db = client.db(DB);
  studentsCollection = db.collection('students');
  console.log('Подключились к MongoDB');
}

// Создаем HTTP сервер
const server = http.createServer(async (req, res) => {
  const parsedUrl = url.parse(req.url, true);

  // Настраиваем CORS заголовки (чтобы работало из браузера)
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader('Access-Control-Allow-Methods', 'PUT, GET, POST, DELETE,
OPTIONS');
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type');

  // Обрабатываем OPTIONS запрос (для механизма CORS)
  if (req.method === 'OPTIONS') {
    res.writeHead(200);
    res.end();
    return;
  }

  // Создаём и обрабатываем все маршруты API

  if (req.method === 'GET' && parsedUrl.pathname === '/api/students') {
    // Получить данные о всех студентах
    const students = await studentsCollection.find({}).toArray();
```

```
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify(students));
  }
  else if (req.method === 'POST' && parsedUrl.pathname === '/api/students') {
    // Добавить нового студента
    let body = '';
    req.on('data', chunk => body += chunk);
    req.on('end', async () => {
      const newStudent = JSON.parse(body);
      await studentsCollection.insertOne(newStudent);
      res.writeHead(201, { 'Content-Type': 'application/json' });
      res.end(JSON.stringify({ message: 'Студент добавлен!' }));
    });
  }
  else if (req.method === 'DELETE' &&
parsedUrl.pathname.startsWith('/api/students/')) {
    // Корректно удалить студента из коллекции
    const pathParts = parsedUrl.pathname.split('/');
    const id = pathParts[3];
    try {
      const result = await studentsCollection.deleteOne({ _id: new ObjectId(id)});
       if (result.deletedCount === 1) {
         res.writeHead(200, { 'Content-Type': 'application/json' });
         res.end(JSON.stringify({deleted: true}));
       } else {
         res.writeHead(404, { 'Content-Type': 'application/json' });
         res.end(JSON.stringify({deleted: false}));
       }
    } catch (err) {
      res.writeHead(500, { 'Content-Type': 'application/json' });
      res.end(JSON.stringify({ error: 'Ошибка сервера!' }));
    }
  }
  else if (req.method === 'GET' && parsedUrl.pathname === '/api/students/search')
{
    // Поиск студентов
    const query = parsedUrl.query.q || '';
    const students = await studentsCollection.find({
      $or: [
        { name: { $regex: query, $options: 'i' } },
        { course: { $regex: query, $options: 'i' } }
      ]
    }).toArray();
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify(students));
  }
  else if (req.method === 'GET' && parsedUrl.pathname === '/api/stats') {
    // Простая статистика
    const total = await studentsCollection.countDocuments();
    const courses = await studentsCollection.distinct('course');
```

```
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({ total, courses }));
  }
  else if (req.method === 'GET' && parsedUrl.pathname === '/') {
    // Стартовая страница сервера, можно было бы сразу вызвать index.html
    res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
    res.end('<h1>Работает Node.js сервер.</h1><p>Доступна страница: <a
href="/index.html">index.html</a></p>');
  }
  else if (req.method === 'GET' && parsedUrl.pathname === '/index.html') {
    // Читаем файл index.html и отдаем его
    const fs = require('fs');
    fs.readFile('index.html', 'utf8', (err, data) => {
      if (err) {
        res.writeHead(500);
        res.end('Ошибка загрузки страницы');
      } else {
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.end(data);
      }
    });
  }
  else if (req.url === '/styles.css') {
    // Подгрузка стилей для index.html
    fs.readFile('styles.css', 'utf8', (err, data) => {
      if (err) {
        res.writeHead(500);
        res.end('Ошибка загрузки css-файла');
      } else {
        res.writeHead(200, { 'Content-Type': 'text/css' });
        res.end(data);
      }
    });
  }
  else if (req.url === '/client.js') {
    // Подгрузка клиентского JavaScript для index.html
    fs.readFile('client.js', 'utf8', (err, data) => {
      if (err) {
        res.writeHead(500);
        res.end('Ошибка загрузки JS-файла');
      } else {
        res.writeHead(200, { 'Content-Type': 'application/javascript' });
        res.end(data);
      }
    });
  }
  else {
    // 404 - не найдено
    res.writeHead(404, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({ error: 'Маршрут не найден!' }));
```

```
  }
});

// Запускаем сервер

async function startServer() {
  await connectToMongoDB();
  server.listen(PORT, () => {
    console.log(`Сервер запущен на http://localhost:${PORT}`);
    console.log(`API доступно по адресу http://localhost:${PORT}/api/students`);
  });
}

startServer();
```

6) Протестируйте работу сервера, набрав в терминале **node server.js**. Откройте браузер, по адресу *http://localhost:3000/api/students* вы должны получить список студентов из коллекции *students* в формате JSON. Сервер поддерживает 5 endpoints; некоторые из них вы также можете легко проверить:

GET /api/students	- Получить данные всех студентов
POST /api/students	- Добавить студента (требуется JSON-строка)
DELETE /api/students/...	- Удалить одного студента (с указанным _id)
GET /api/students/search?q=…	- Поиск студентов
GET /api/stats	- Вывод статистики.

Остановим web-сервер и будем далее писать клиентскую часть.

7) Будем писать одностраничное клиентское web-приложение, в котором динамические данные подгружаются на страницу *index.html* в виде JSON-кода без перезагрузки самой страницы. Создайте файл *index.html* и добавьте в него следующее содержание:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Практикум по MongoDB, Интернет-доступ к данным</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>📚 Студенты (учебная коллекция)</h1>

    <div id="message" class="success"></div>
    <div id="error" class="error"></div>

    <div class="stats">
      <h3>📊 Статистика:</h3>
```

```
      <p>Всего студентов: <span id="totalCount">0</span></p>
      <p>Курсы: <span id="coursesList">загрузка...</span></p>
    </div>

    <div class="app-container">
      <!-- Левая панель: Добавление студента -->
      <div class="left-panel">
        <h2>➕ Добавить нового студента</h2>
        <form id="addForm">
          <div class="form-group">
            <label>ФИО:</label>
            <input type="text" id="name" placeholder="Иван Иванов" required>
          </div>

          <div class="form-group">
            <label>Возраст:</label>
            <input type="number" id="age" placeholder="20" min="16" max="60">
          </div>

          <div class="form-group">
            <label>Email:</label>
            <input type="email" id="email" placeholder="ivan@example.com">
          </div>

          <div class="form-group">
            <label>Курс:</label>
            <select id="course">
              <option value="Информатика">Информатика</option>
              <option value="Математика">Математика</option>
              <option value="Физика">Физика</option>
              <option value="Химия">Химия</option>
              <option value="Биология">Биология</option>
            </select>
          </div>

          <button type="submit">Добавить студента</button>
        </form>

        <div class="search-box">
          <input type="text" id="searchInput" placeholder="Поиск по имени или
курсу...">
          <button onclick="searchStudents()">🔍 Найти</button>
        </div>

        <h3>🔍 Результаты поиска:</h3>
        <div id="searchResults"></div>
      </div>

      <!-- Правая панель: Список студентов -->
      <div class="right-panel">
```

```
        <h2>👥 Список студентов</h2>
        <button onclick="loadStudents()" style="margin-bottom: 10px; background:
#2196F3;">🔄 Обновить список</button>

        <div id="studentsTable">
          <div class="loading">Загрузка студентов...</div>
        </div>
      </div>
    </div>
  </div>
  <script src="client.js"></script>
</body>
</html>
```

Этот код не будет работать без стилевого оформления и клиентских функций, осуществляющих подгрузку и обработку данных. Создадим сначала файл *styles.css* со следущим содержанием:

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: Arial, sans-serif;
}

body {
  background: #f5f5f5;
  padding: 20px;
  max-width: 1200px;
  margin: 0 auto;
}

.container {
  background: white;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
}

h1 {
  color: #333;
  margin-bottom: 20px;
  text-align: center;
}

h2 {
  color: #555;
  margin: 20px 0 10px 0;
}
```

```css
.app-container {
  display: flex;
  gap: 20px;
}

.left-panel, .right-panel {
  flex: 1;
}

.form-group {
  margin-bottom: 15px;
}

label {
  display: block;
  margin-bottom: 5px;
  color: #666;
  font-weight: bold;
}

input, select {
  width: 100%;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 5px;
  font-size: 16px;
}

button {
  background: #4CAF50;
  color: white;
  border: none;
  padding: 12px 20px;
  border-radius: 5px;
  cursor: pointer;
  font-size: 16px;
  font-weight: bold;
  transition: background 0.3s;
  width: 100%;
}

button:hover {
  background: #45a049;
}

.search-box {
  display: flex;
  gap: 10px;
  margin: 20px 0;
}
```

```css
.search-box input {
  flex: 1;
}

.search-box button {
  width: auto;
  background: #2196F3;
}
.search-box button:hover {
  background: #0b7dda;
}

.stats {
  background: #e3f2fd;
  padding: 15px;
  border-radius: 5px;
  margin-bottom: 20px;
}

table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 10px;
}

th {
  background: #4CAF50;
  color: white;
  padding: 12px;
  text-align: left;
}

td {
  padding: 10px;
  border-bottom: 1px solid #ddd;
}

tr:hover {
  background: #f5f5f5;
}

.delete-btn {
  background: #f44336;
  padding: 5px 10px;
  font-size: 14px;
  width: auto;
}

.delete-btn:hover {
```

```
  background: #d32f2f;
}

.success {
  background: #d4edda;
  color: #155724;
  padding: 10px;
  border-radius: 5px;
  margin: 10px 0;
  display: none;
}

.error {
  background: #f8d7da;
  color: #721c24;
  padding: 10px;
  border-radius: 5px;
  margin: 10px 0;
  display: none;
}

.loading {
  text-align: center;
  padding: 20px;
  color: #666;
}
```

И, наконец, создадим файл *client.js* , подключённый к *index.html*, со следующим содержанием:

```
// Базовый URL API
const API_URL = 'http://localhost:3000/api';

// Показываем сообщение
function showMessage(text, isError = false) {
  const messageDiv = document.getElementById(isError ? 'error' : 'message');
  messageDiv.textContent = text;
  messageDiv.style.display = 'block';
  setTimeout(() => {
    messageDiv.style.display = 'none';
  }, 3000);
}

// Загружаем статистику
async function loadStats() {
  try {
    const response = await fetch(`${API_URL}/stats`);
    const data = await response.json();
    document.getElementById('totalCount').textContent = data.total;
```

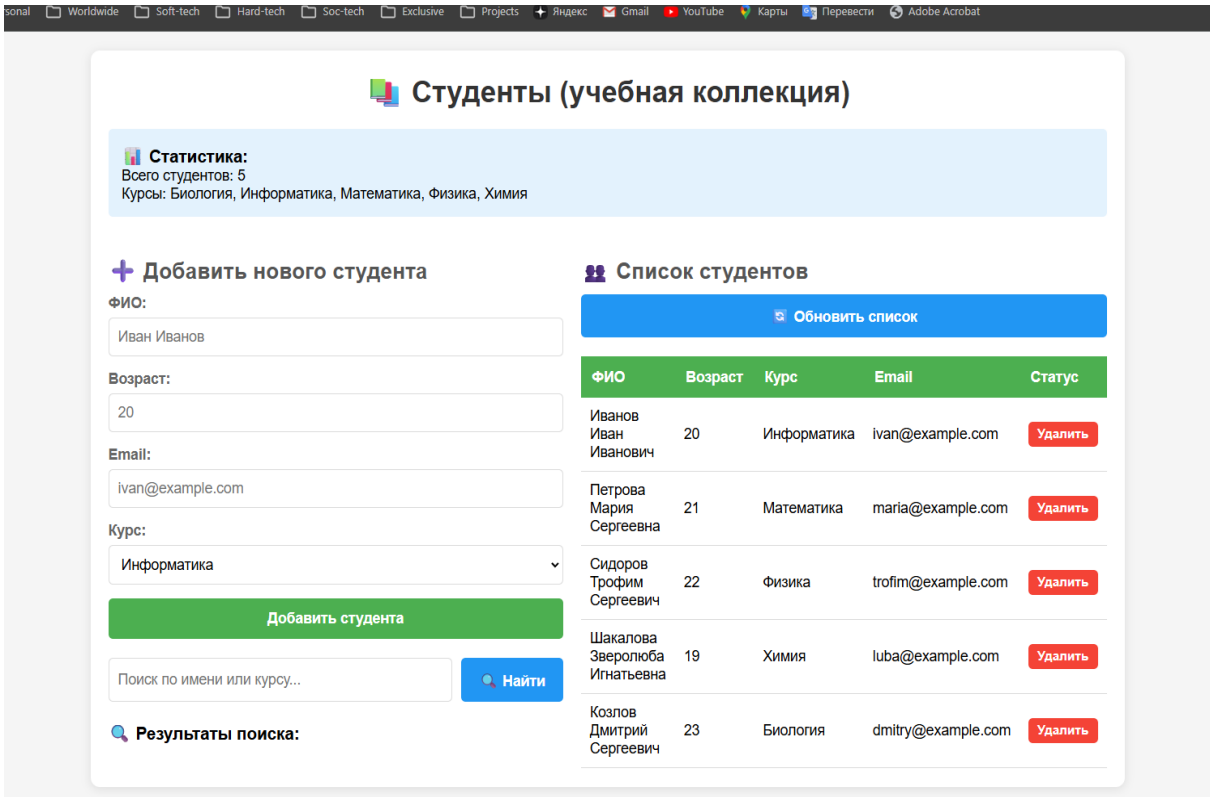
```
    document.getElementById('coursesList').textContent = data.courses.join(', ');
  } catch (err) {
    console.log('Ошибка загрузки статистики:', err);
  }
}

// Загружаем список студентов
async function loadStudents() {
  const tableDiv = document.getElementById('studentsTable');
  tableDiv.innerHTML = '<div class="loading">Загрузка студентов...</div>';

  try {
    const response = await fetch(`${API_URL}/students`);
    const students = await response.json();

    if (students.length === 0) {
      tableDiv.innerHTML = '<p>Нет студентов в базе данных</p>';
      return;
    }

    let html = `
      <table>
        <thead>
          <tr>
            <th>ФИО</th>
            <th>Возраст</th>
            <th>Курс</th>
            <th>Email</th>
            <th>Статус</th>
          </tr>
        </thead>
        <tbody>
    `;

    students.forEach(student => {
      html += `
        <tr>
          <td>${student.name || '-'}</td>
          <td>${student.age || '-'}</td>
          <td>${student.course || '-'}</td>
          <td>${student.email || '-'}</td>
          <td><button class="delete-btn" onclick="deleteStudent('$
{student._id}')">Удалить</button></td>
        </tr>
      `;
    });

    html += '</tbody></table>';
    tableDiv.innerHTML = html;
```

```
  } catch (err) {
    tableDiv.innerHTML = '<p class="error">Ошибка загрузки: ' + err.message +
'</p>';
  }
}

// Добавляем нового студента
document.getElementById('addForm').onsubmit = async function(e) {
  e.preventDefault();

  const student = {
    name: document.getElementById('name').value,
    age: parseInt(document.getElementById('age').value) || 20,
    email: document.getElementById('email').value || '',
    course: document.getElementById('course').value,
    createdAt: new Date()
  };

  try {
    const response = await fetch(`${API_URL}/students`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(student)
    });

    const result = await response.json();
    showMessage('Студент успешно добавлен!');

    // Очищаем форму
    document.getElementById('addForm').reset();

    // Обновляем список и статистику
    loadStudents();
    loadStats();

  } catch (err) {
    showMessage('Ошибка: ' + err.message, true);
  }
};

// Ищем студентов
async function searchStudents() {
  const query = document.getElementById('searchInput').value;
  const resultsDiv = document.getElementById('searchResults');

  if (!query) {
    resultsDiv.innerHTML = '<p>Введите поисковый запрос</p>';
    return;
```

```
  }

  resultsDiv.innerHTML = '<div class="loading">Поиск...</div>';

  try {
    const response = await fetch(`${API_URL}/students/search?q=${encodeURIComponent(query)}`);
    const students = await response.json();

    if (students.length === 0) {
      resultsDiv.innerHTML = '<p>Студенты не найдены</p>';
      return;
    }

    let html = '<div style="background: #f9f9f9; padding: 10px; border-radius: 5px;">';
    students.forEach(student => {
      html += `<p><strong>${student.name}</strong> - ${student.course} (${student.age || '?'} лет)</p>`;
    });
    html += '</div>';

    resultsDiv.innerHTML = html;

  } catch (err) {
    resultsDiv.innerHTML = '<p class="error">Ошибка поиска</p>';
  }
}

// Удаляем студента
async function deleteStudent(id) {
  if (!confirm('Удалить этого студента?')) return;
  try {
    const response = await fetch(`${API_URL}/students/${id}`, {
      method: 'DELETE'
    });
    const result = await response.json();
    if (response.ok) {
      showMessage('Студент успешно удален!');
      loadStudents();
      loadStats();
    } else {
      showMessage('Ошибка: ' + (result.error || 'Не удалось удалить'), true);
    }
  } catch (err) {
    showMessage('Ошибка соединения с сервером', true);
  }
}

// Загружаем данные при старте
```

```
window.onload = function() {
  loadStudents();
  loadStats();
};
```

8) Протестируем созданный проект. Запустите сервер и обратитесь к нему через браузер по адресу *http://localhost:3000/index.html*. Должна появиться страница следующего вида:



Проверьте работу всех кнопок на представленной странице и убедитесь, что работают поиск, добавление и удаление студентов.

9) Изменим файл описания проекта:

```
{
 "name": "Easy-mongo-app",
 "version": "1.0.0",
 "description": "Простое приложение Node.js + MongoDB",
 "main": "server.js",
 "scripts": {
  "start": "node server.js",
  "test": "echo \"Error: no test specified\" && exit 1"
 },
 "keywords": [],
 "author": "???",
 "license": "ISC",
 "type": "commonjs",
 "dependencies": {
  "dotenv": "^17.2.4",
```

```
    "mongodb": "^4.0.0"
  }
}
```

Теперь, помимо эстетических улучшений, вы сможете запускать сервер «абстрактной» командой **npm start**.

Задача выполнена.

10) **! В коллекции students в каждом документе есть ещё несколько незадействованных полей. Измените код клиентской и серверной части представленного проекта таким образом, чтобы в таблице** (только в таблице!) **присутствовали поля «Средний балл» (gpa) и «Статус» (active). Сделайте так, чтобы список студентов всегда выводился по алфавиту (по фамилиям), а в области статистики выводился средний балл по всем студентам в коллекции.**

6. Итоговое творческое задание.

! Примените полученный опыт для создания аналогичного приложения для работы с коллекцией *movies*, ранее скопированной в вашу базу данных. Рекомендуется сделать только просмотр данных в виде таблицы либо списка (не делать ни изменения, ни удаления данных), но при этом реализовать на web-странице хотя бы минимальный выбор условий для отображения данных (по году выпуска фильма, по режиссёру, актёру и т. п.). Поскольку выборка случайно может оказаться очень и очень большой, в запросах сразу поставьте ограничение на количество выводимых документов.