Достъп до MongoDB с Node.js

I. Въведение

Node.js е междуплатформена среда за изпълнение на JavaScript с отворен код, която работи с двигателя V8 и изпълнява JavaScript код без участието на Web браузър. V8 е двигател за изпълнение на JavaScript, който първоначално е създаден за Google Chrome през 2008 г. Началният вариант на Node.js е написан от Райън Дал (Ryan Dahl) през 2009 г. Преди проектът Node.js се управляваше от фондация Node.js, а сега от OpenJS Foundation. През 2019 г. фондациите JS Foundation и Node.js Foundation се сляха, за да създадат OpenJS Foundation.

Node.js позволява създаването на Web сървъри и мрежови услуги с помощта на JavaScript и набор от **модули**, които управляват различни основни функционалности. Предвидени са модули за входно/изходни операции на файловата система, работа в мрежа (DNS, HTTP, TCP, UDP и TLS/SSL комуникации), криптографски функции, работа с потоци от данни и др. Модулите на Node.js използват API, предназначен да намали сложността на писането на сървърни приложения. Модулите в Node js са начин за капсулиране на код в отделна логическа единица. На пазара се предлагат много готови модули, които могат да се използват в Node js. По-долу са изброени някои от популярните Node.js модули:

- Express - Express е малка по размер и гъвкава рамка за Web приложения, която предоставя солиден набор от функции за Web и мобилни приложения.
- Socket.io - Socket.io позволява двупосочна комуникация в реално време, базирана на събития.
- Jade - Jade е високопроизводителна машина за шаблони, силно повлияна от Haml и реализирана с JavaScript за Node.js и браузъри.
- MongoDB Node.js драйвер - Драйверът MongoDB Node.js е официално поддържаният драйвер за достъп до MongoDB бази данни.
- Restify - Олекотена програмна рамка за изграждане на семантично коректни RESTful уеб услуги. Restify оптимизира производителността и се използва в някои от най-големите внедрявания на Node.js. Restify умишлено е заимстван в голяма степен от Express, тъй като това е API за писане на Web приложения над Node. js.
- Bluebird - Bluebird е JS е напълно функционална библиотека за работа с обекти-обещания (promises). Най-силната характеристика на Bluebird е, че тя позволява да "обещавате" други модули на Node.js, за да ги използвате асинхронно. Promisify е концепция, която се прилага към функциите за обратно извикване (callback).

За да използвате модули в приложение на Node.js, те първо трябва да бъдат инсталирани чрез мениджъра на пакети на Node.js – **npm**. Например, за да инсталирате модул Express трябва да изпълните следната команда:

```
npm install express
```

Тази команда ще изтегли необходимите файлове, които изграждат Express. След като модулът е инсталиран, за да го използвате в приложение на Node.js, трябва да използвате ключовата дума "require". Тази ключова дума е начин, който Node.js използва, за да включи функционалността на даден модул в дадено приложение, например:

```
const express = require('express');  // зареждане на модула
const app = express(); // създаване на обект за достъп до Express
```

Node.js позволява на разработчиците да използват JavaScript за писане на код от страна на сървъра (backend) - изпълнение на скриптове от страна на сървъра за създаване на динамично съдържание на Web страници, преди страницата да бъде изпратена към браузъра на потребителя. Следователно, Node.js позволява разработването на Web приложения чрез един-единствен език за програмиране, а не чрез различни езици от страна на сървъра и от страна на клиента (full-stack програмиране).

Node.js се **управлява от събития**. Всички комуникации (входно-изходни операции) по подразбиране са асинхронни и следователно са неблокиращи. Този избор на дизайн има за цел да оптимизира пропускателната способност и мащабируемостта на Web приложения с много входно/изходни операции, както и за Web приложения, които работят в реално време. Node.js използва една програмна нишка за изпълнение на JavaScript модули, използвайки неблокиращи I/O извиквания. Това позволява да се поддържат десетки хиляди едновременни комуникационни канали (връзки). При използване на Node.js програмистът не трябва да има познания в областта на thread-safe програмирането. За да приспособи еднонишковия цикъл на събитията, Node.js използва библиотеката **libuv**. Това е мултиплатформена библиотека на C, която осигурява поддръжка на асинхронни входно-изходни операции, базирани на цикли от събития. Библиотеката libuv използва пул от нишки с фиксиран размер, който обработва някои от неблокиращите асинхронни I/O операции. Пулът от нишки се грижи за изпълнението на паралелни задачи в Node.js. Извикването на функцията на главната нишка изпраща задачи в обща опашка за задачи, които нишките в пула от нишки изтеглят и изпълняват. Неблокиращите по същността си системни функции, като например мрежовите, се превеждат към неблокиращи сокети от страна на ядрото, докато блокиращите по същността си системни функции, като например файловите входно-изходни операции, се изпълняват блокиращо чрез собствени нишки. Когато някоя нишка в пула от нишки завърши своята задача, тя информира за това главната нишка, която на свой ред се събужда и изпълнява регистрираното обратно извикване (callback функция).

Node.js поддържа **WebAssembly**. WebAssembly (Wasm) е отворен стандарт, който определя преносим формат на двоичен код за изпълними програми и съответен текстов формат, например JavaScript, както и интерфейси за улесняване на взаимодействието между такива програми и тяхната хост среда. WebAssembly и WebAssembly JavaScript Interface са препоръка на World Wide Web Consortium от 5.12.2019 г,. а от Node,js v.14 има експериментална поддръжка на WASI (WebAssembly System Interface) - системния интерфейс на WebAssembly. Node.js предоставя начин за създаване на добавки (**addons**) чрез N-API (API, базиран на програмния език C), който може да се използва за създаване на зареждащи се Node.js модули от изходен код, написан на C или C++.

II. Необходими инсталации

Сървърите за управление на MongoDB бази данни позволяват използване на Node.js за достъп до базите чрез Node.js драйвер. Официалният драйвер MongoDB Node.js позволява на Node.js приложенията да се свързват с MongoDB сървъра по унифициран начин, характерен за всички програмни езици, които MongoDB поддържа. Драйверът разполага с асинхронен API, който позволява да се получи достъп до върнатите стойности (след изпълнение на заявки) чрез JavaScript обещания (promises) или чрез callback функции.

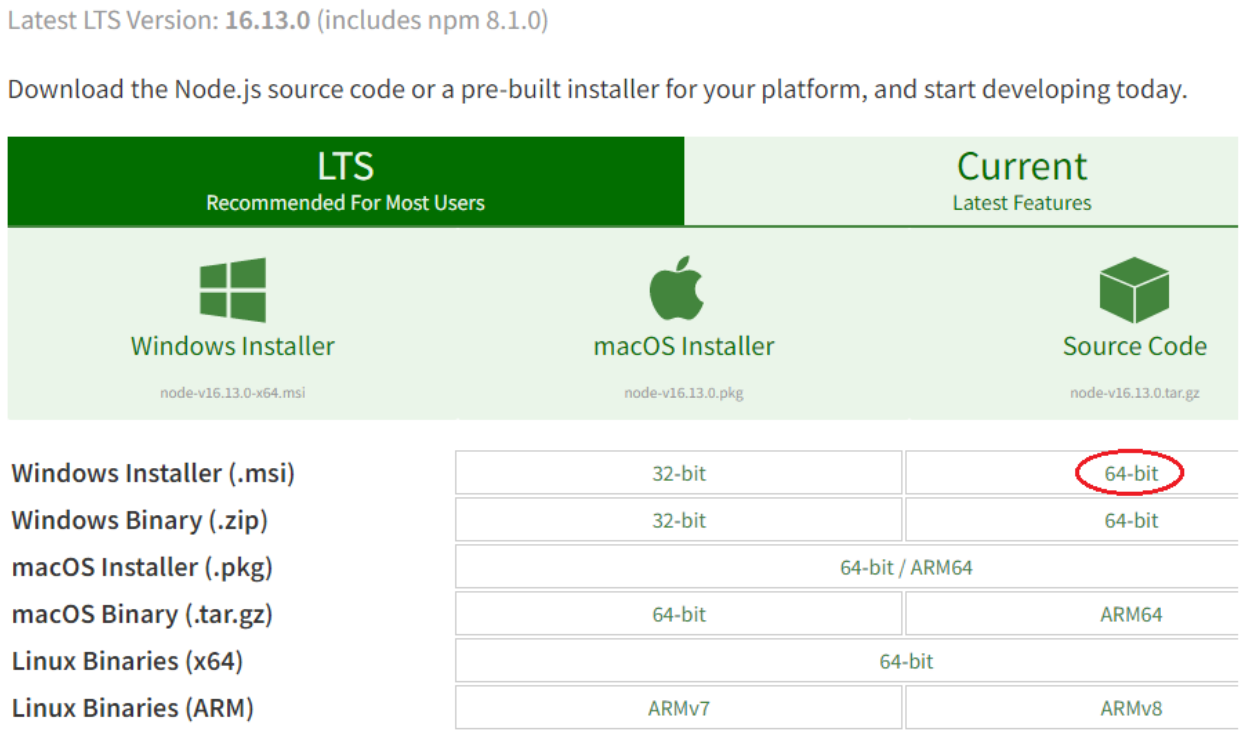
Инсталиране на Node.js

Може да свалите необходимия инсталатор за Node.js от следния адрес:

https://nodejs.org/en/download/

Изберете инсталатор за вашата операционна система. При ОС Windows е най-добре да свалите съответния msi инсталатор, например:

node-v14.17.3-x86.msi

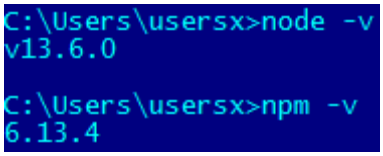


С тази инсталация ще получите съответната версия на Node.js, както и версия на NMP (Node Package Manager). Приложението NPM е по подразбиране мениджър за инсталиране на Node.js програмни пакети. Състои се от клиент за командния ред, също наречен NPM, и онлайн база данни с публични и платени частни пакети, наречена NPM регистър. Достъпът до регистъра се осъществява чрез клиента, а наличните пакети могат да бъдат разглеждани и търсени чрез уебсайта на NPM Inc.

Направете проверка дали инсталацията е успешна. За целта от командния ред изпълнете следните команди:

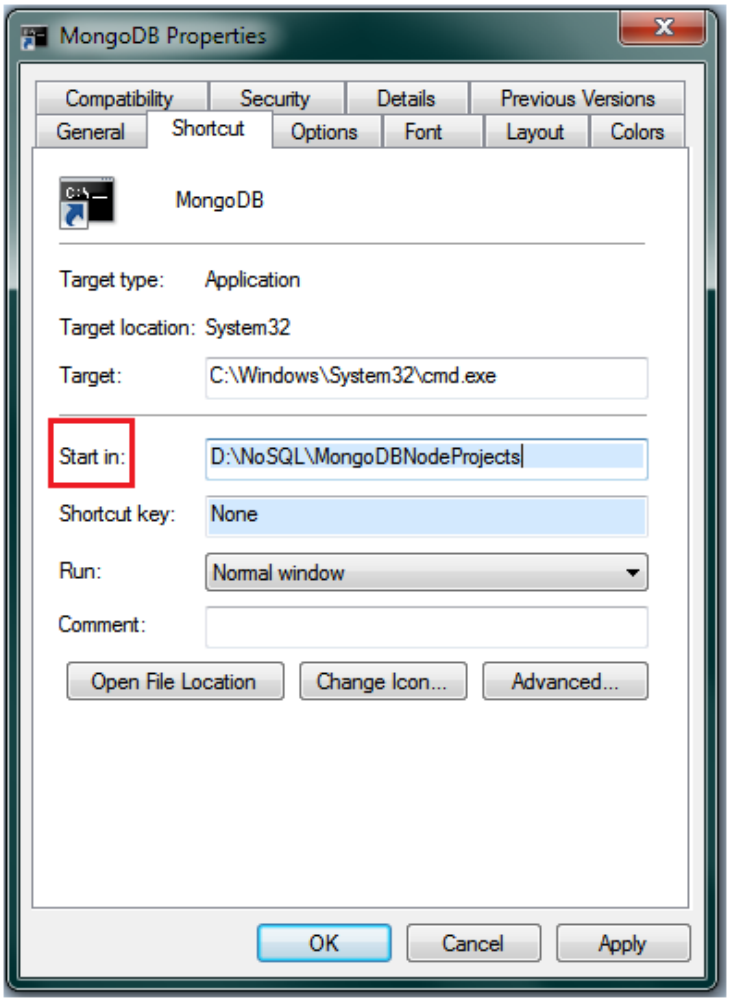
node –v
npm –v

Като резултат трябва да получите версиите на инсталираното програмно осигуряване:



Инсталиране на MongoDB Node.js драйвер

Създайте папка (например MongoDBNodeProjects) в която ще бъдат вашите MongoDB Node.js проекти. За да имате бърз достъп от командно ниво до тази папка, направете чрез командния интерпретатор на Windows (cmd.exe) препратка към Desktop (виж Фиг. 1). За целта изберете с десния бутон на мишката приложението cmd.exe и изберете “Desktop (create shortcut)”. Къде се намира cmd.exe се вижда от поле “Target”.



Фиг. 1 Задаване на желана папка да бъде активна след стартиране на cmd.exe

След избор на тази препратка, вашата папка ще бъде текущата папка. От командния ред стартирайте следната команда:

npm install mongodb –save

Тази команда ще изтегли MongoDB Node.js драйвера и ще добави запис за зависимост във вашия файл “package.json”. Проверете дали драйверът е инсталиран коректно:

npm view mongodb version

След като имате наличен драйвер, може да го използвате като модул към вашите MongoDB Node.js проекти.

```
C:\Users\usersx>npm view mongodb version
4.2.0
```

III. Технологични стекове за разработване на Node.js Web услуги

Технологичният стек, известен също като програмен стек, е екосистема от данни, която включва основните програмни подсистеми или компоненти, необходими за създаването на цялостна услуга, така че да не е необходим допълнителен софтуер за поддръжка на приложенията. Например Facebook използва стек, който включва PHP, React, GraphQL, Cassandra, Hadoop, Swift и множество други рамки.

Конкурентоспособността и жизнеспособността на всяка услуга зависят от технологичния стек, който е избран. Изборът на стек има голямо значение за разработката на вашето уеб приложение, тъй като от него зависи:

- Как работи вашето приложение.
- Как ще се държи в бъдеще.
- Мащабируемостта на услугата.
- Изборът ви на място за съхранение на данни и приложения (в облак или локално).
- Капацитетът на вашите сървърни системи.

Като анализирате силните и слабите страни на популярните програмни стекове, можете да започнете да разработвате услугата си с по-голяма увереност, тъй като ще се запознаете с начините за намаляване на рисковете при разработката.

Основните програмни стекове за разработка на Web услуги чрез Node.js са следните:

1. Стек MEAN

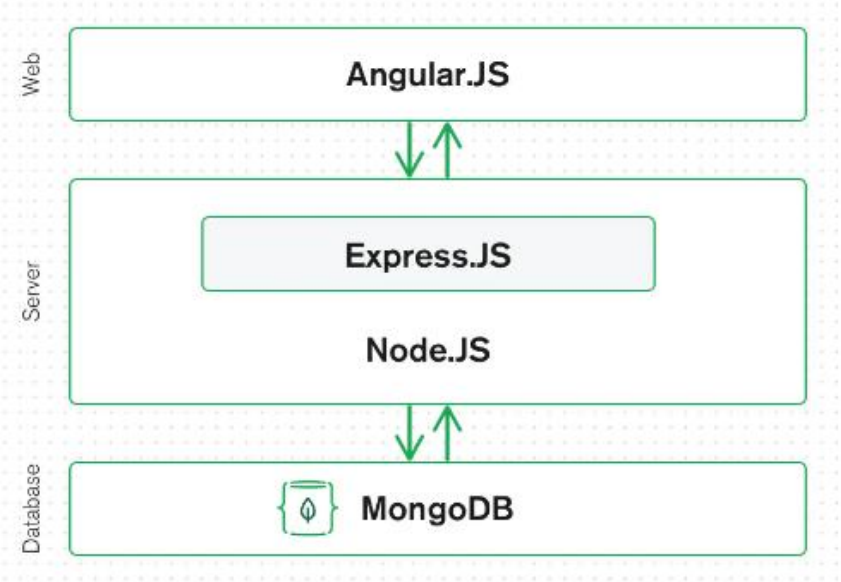
Стекът MEAN (MongoDB, Express.js, AngularJS и Node.js) е един от най-популярните технологични стекове за 2021 г. Предназначението на технологиите от този стек е следното:

MongoDB: Това е междуплатформена, ориентирана към документи мащабируема NoSQL база данни..

Express.js: Това е програмен слой над Node.js. Поддържа рамката на бекенда и улеснява писането на кода на бекенда. Тази модулна рамка поддържа и междинни програми и насърчава повторната използваемост на кода, за да улесни процеса на писане на код. Тя е специално проектирана за изграждане на опростени Web приложения.

Angular.js: Фронт-енд JavaScript рамка с отворен код за изграждане на потребителския интерфейс.

Node.js: Това е среда за изпълнение на JavaScript, която позволява на потребителите да изпълняват код от страна на сървъра. Тя поддържа Node Pack Manager (npm), който представлява комбинация от хиляди публични и частни модули за Node.js, които са безплатни за изтегляне.



Тъй като MEAN е стек от край до край на JavaScript, трябва да използвате един-единствен език в рамките на целия стек. По този начин имате възможност да използвате повторно кода в цялото приложение. Друго предимство на използването на MEAN е, че всички технологии са безплатни и с отворен код с подкрепата на активна общност. MEAN-базираните приложения и услуги са идеални за облачен хостинг, тъй като са гъвкави, мащабируеми и разширяеми. Стекът MEAN може да бъде разгърнат лесно, тъй като включва собствен Web сървър. Базата данни може да се мащабира при поискване. MEAN ви позволява да използвате един екип от разработчици на JavaScript, които могат да работят адаптивно.

Използването на MEAN стек е подходящ избор за ускоряване на процеса на разработване на Web или мобилни приложения. Основните предимства на MEAN стека са следните:

- Разработчиците, използващи MEAN стек обхващат целия процес на разработка от фронтенда до бекенда на приложението, като използват JavaScript.
- Услугите са най-често с MVC (Model-View-Controller) архитектура и с цел изолация на бизнес логиката от кода за формиране на потребителския интерфейс.
- Работи се с първокласни технологии, т.е. използват MongoDB и Express.js за работа на бекенда, докато Angular.js и Node.js за работа на фронтенда.
- Позволява използването на популярни плъгини и намалява времето, необходимо за администриране на системата..

Уменията, които трябва да притежава разработчикът на MEAN стек, са следните:

- Да познава облачните технологии, MVC архитектурата, Web проектирането и интеграцията.
- Да има познания за HTML5 и CSS3.
- Разбиране на SDLC (Systems Development Life Cycle) и Agile (гъвкавите методологии са набор от методологии и програмни техники при програмиране, софтуерна разработка и управление на софтуерни проекти.)
- Да има достъп до множество рамки за създаване и разработване на Web сайтове.

2. Стек MERN

MERN стекът е често срещана алтернатива на MEAN стека, при която традиционният MERN е акроним, съставен от набор от технологии, а именно MongoDB, Express.js, React.js и Node.js. Angular.js е заменен с библиотека React.js, което прави процеса на разработка много по- лесен.

React.js: Това е front-end рамка, която се използва за създаване на потребителския интерфейс. Тя се справя добре с бързо променящите се данни, като опростява разработването на SPA (Single Page App) приложения, както и мобилни приложения чрез React Native. Тази JavaScript библиотека позволява на разработчиците да създават компоненти на потребителския интерфейс и интерфейса на SPA Web приложения. Тя също така им позволява да редактират и опресняват съдържанието на страницата, без да я презареждат.

React е най-харесваната уеб рамка, като 74,5% от анкетираните разработчици (2021 г.) предпочитат да разработват с нея. Съответната цифра за Angular е 57,6%. В сравнение с Angular.js, React.js е най-добър за абстракция на слоя на потребителския интерфейс. Тя изпълнява бързо разработката на код благодарение на колекцията си от динамични потребителски интерфейси, които са лесно достъпни. MERN действа най-добре при

контролирането и актуализирането на големи динамични JSON данни, които могат да се движат безпроблемно между frontend и backend на услугата.

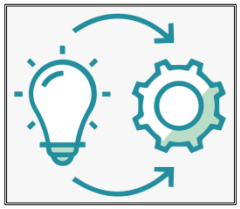
Основни разлики между MEAN и MERN

MEAN Stack	MERN Stack
Използва рамка AngularJS	Използва библиотека ReactJS
Подкрепя се от Google Inc.	Подкрепя се от Facebook Inc.
Изисква по-продължително обучение	Много по-лесно е да се научи
Големи различия между версии 1 и 2	React.js се представя по-добре от Angular.js при поддържане на визуализирането на потребителския интерфейс
MEAN поддържа абстракция в кодирането и също така управлява файлове.	MERN позволява разработването на код с по-бързи темпове.
MEAN стекът е за предпочитане при създаване на големи корпоративни проекти	MERN е предпочитан за бърза разработка на по-малки по сложност приложения
Притежава готови за използване функции за поддръжка на множество библиотеки на трети страни	React.js се нуждае от допълнителни библиотеки, за да поддържа такива извиквания.
Предлага добра производителност на разработчиците.	React предлага по-ниска производителност на разработчиците.
Потокът от данни е двупосочен - когато потребителският интерфейс се промени, състоянието на модела се променя автоматично.	Потокът от данни е еднопосочен – потребителският интерфейс може да се променя само след промяна на състоянието на модела.

Недостатъци на MEAN и MERN стека

Двата стека се базират на JavaScript, а този програмен език не може да гарантира възможността за разработване на бърза и сигурна бизнес логика. Вероятността за получаване на спагети код е голяма. Програмният код от страна на сървъра може да има лоша изолация.

IV.Задачи за изпълнение



Задача 1: *Създайте Node.js приложение, което разпечатва на конзолата стойността на показанията на определен тип сензори (за температура или влажност на въздуха) от база данни "my-sensors", колекция "data". Да се обработват само данните, получени преди дата 15.06.2021. Резултатът да се сортира в зависимост от стойността на идентификатора на сензорите.*

Създайте нов JavaScript файл в папката с Node.js проекти. Първо ще декларираме всички необходими константи (виж Фиг. 2). Зареждането на желан клас от програмен модул се реализира чрез метод **require**. При конкретния пример от модул "mongodb" създаваме обект от клас MongoClient. Този обект ще е необходим при свързване с базата данни.

```
const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017';
const dbName = 'my-sensors';
const collectionName = "data";
const options = {
    serverSelectionTimeoutMS: 3000,
    connectTimeoutMS: 3000,
    socketTimeoutMS: 3000,
    useUnifiedTopology: true,
};
const sensorType = 'humidity';
```

Фиг. 2 Деклариране на константи

Останалите константи, които приложението ще използва, са следните:

- url – низ, съдържат спецификацията за връзка с локално инсталиран MongoDB сървър.
- collectionName – име на колекцията с която ще работим.
- options – JSON обект, който се използва от метод connect чрез който се реализира свързване със MongoDB сървъра. Използвани са полета чрез които се задава времена на изчакване при свързване с MongoDB.
- sensorType – кой тип сензор се интересуваме (“temperature” или “humidity”).

Ще създадем функция **startQuery**, която съдържа програмния код за свързване с базата данни, изпращане на необходимата заявка и парсване на резултата. Тъй като Noje.js работи по подразбиране асинхронно, след като се извика една функция не се чака да се получи резултат, а се вика следващата функция. Следователно, трябва ни механизъм чрез който да знаем кога една функция е върнала резултат. Има два основни подхода за това:

- Задаване чрез ключова дума **await**, че трябва да се изчака изпълнението на текущата функция.
- Използване на JavaScript обект-обещание (**promise**) чрез който може да разберем кога една функция е върнала резултат чрез викане на наша callback функция.

И двата подхода могат да бъдат използвани, но за предпочитане е да се работи с обещания. Чрез тях може да синхронизирате работата на своите функции, да регистрирате callback функция при завършване на една или няколко функции. Освен това, използването на обещания позволява много по-добра обработка на възникналите изключения.

При конкретния пример ще използваме ключова дума await. За да използвате await от тялото на дадена функция трябва функцията да се декларира като **async**. Например, декларацията на функция startQuery (виж Фиг. 3) започва с ключова дума async. Това е необходимо, тъй като ще използваме ключова дума await за функцията за свързване с базата данни (connect) и функцията за изпращане на заявка (find).

Тъй като реализираме комуникация в мрежова среда, вероятността за получаване на изключения е висока. Обработката им става чрез клаузи **try-catch-finally**. Започване с получаване на обект “client” чрез който ще изградим комуникационен канал между клиента и сървъра. За целта се използва метод **connect**. При проблем се изпълнява

програмния код от тялото на клауза catch – печата се съобщение за грешката и се прекратява програмата. Чрез методи **time** и **timeEnd** се изчислява за колко време се изгражда връзката.

```
async function startQuery() {

    let client;
    console.time('Connect time ');
    try {
        client = await MongoClient.connect(url, options);
    }
    catch(error) {
        console.log(`Timeout:\n${error.message}`);
        process.exit();
    }
    finally {
        console.timeEnd('Connect time ');
    }

    const collection = client.db(dbName).collection(collectionName);
    console.time('Find time: ');
    try {
        await collection
            .find(
                {
                    type: sensorType,
                    timestamp: { $lt: new Date('2021-06-15') }
                },
                {
                    projection: {_id: 0, value: 1, sensorId: 1}
                }
            ).sort(
                {
                    sensorId: 1
                }
            ).toArray(function(error, results) {
                if (error) throw error;
                let units = sensorType === 'temperature' ? '\u00B0C' : '%';
                results.forEach(result => {
                    let id = result.sensorId;
                    let value = result.value;
                    console.log(`Sensor ${id}: ${sensorType} ${value}${units}`);
                });
                client.close();
            });
    }
    catch(error) {
        console.log(error.message);
    }
    finally {
        console.timeEnd('Find time: ');
    }
}
```

Фиг. 3 Метод startQuery - Задача 1

Получаването на обект за работа с базата данни се реализира чрез метод **db**, а обект за работа с конкретна колекция – метод **collection**. В нов try блок се изпълнява програмния код за изпращане на заявката и обработка на резултата. Заявката се изпраща чрез метод **find**. Неговият синтаксис е идентичен със синтаксиса на метод find от конзолата на сървъра MongoDB. В конкретния пример метод find получава два JSON обекта. Първият описва филтъра (ограничения за полета “type” и “timestamp”, а втория задава чрез поле “projection” какви полета да останат в резултата. Зададено е възходящо сортиране на резултата (sensorId: 1). Чрез метод **toArray** резултатът се конвертира до списък от

масиви. Всеки елемент от списъка е един JSON документ. Обхождането на документите в обекта за резултата “results” се реализира чрез метод forEach и използване на ламбда функция (оператор =>). От тялото на функцията се извлича и разпечатва идентификатора на сензора (sensorId) и стойността на температурата или влажността на въздуха.

Не забравяйте да стартирате метод startQuery!

След като въведете програмния код, запишете JavaScript файла в папката с вашите Node.js проекти. Стартирайте приложението от командния ред чрез следната номада:

node QUERY_8_1.js

Трябва да получите резултат, подобен на следния:

```
Connect time : 46.494ms
Find time: : 5.39ms
Sensor 15: humidity 57%
Sensor 20: humidity 73%
```

Задача 2: *Създайте Node.js приложение, което разпечатва на конзолата средната стойност на показанията на всички сензори, групирани по техния тип (температура или влажност на въздуха) от база данни “my-sensors”, колекция “data”. Да се обработват само данните, получени преди дата 20.06.2021.*

Създайте нов JavaScript файл в папката с Node.js проекти, например QUERY_8_2.js. В този случай, ще използваме функцията за агрегиране на съдържание, тъй като трябва да получим средна стойност. В Node.js тази функция е с име **aggregate**. На Фиг. 4 е показан програмния код който вика метод aggregate.

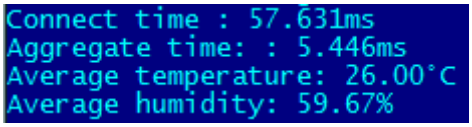
```
    try {
        await collection
            .aggregate(
                [
                    {
                        $match:
                        {
                            timestamp: { $lt: new Date('2021-06-20') }
                        }

                    },
                    {
                        $group:
                        {
                            _id : '$type',
                            avg : { $avg : '$value' }
                        }
                    }
                ]
            ).toArray(function(error, results) {
                if (error) throw error;
                results.forEach(result => {
                    let type = result._id;
                    let units = result._id === 'temperature' ? '\u00B0C' : '%';
                    console.log(`Average ${type}:
                                ${result.avg.toFixed(2)}${units}`);
                });
                client.close();
            });
    }
```

Фиг. 4 Програмен код на метод aggregate - Задача 2

Следователно трябва да замените метод find с метод aggregate. Чрез оператор $match ще зададем нужния филтър (поле “timestamp”). Чрез оператор $group ще обединим всички файлове с еднакъв тип на сензора и ще въведем в изходния документ ново поле с име “avg”, чиято стойност ще получим чрез оператор $avg приложен към стойността на показанията на сензора – поле “value”.

Трябва да получите следния резултат:



Задача 3: *Тествайте програмния код от Задача 1 и Задача 2 но с бази данни, разположени в MongoDB Atlas. Направете анализ на времето за свързване със сървъра и времето, необходимо за изпълнение на заявките.*

Използвайте следният синтаксис за обект url:

```
"mongodb://username:password@cluster0-shard-00-00.shlgp.mongodb.net:27017,cluster0-shard-00-01.shlgp.mongodb.net:27017,cluster0-shard-00-02.shlgp.mongodb.net:27017/DATABASE?ssl=true&replicaSet=atlas-lhm95z-shard-0&authSource=admin&retryWrites=true&w=majority"
```