Node.js: search by date in MongoDB with Mongoose

In this tutorial we will see how to search by date on MongoDB in Node.js using Mongoose.

The simplest use case is to search by exact date, i.e. by entering the year, month and day. Keep in mind that the search parameters that will make up the final `Date` object must be validated.

A day obviously consists of 24 hours, so in order to get the entire time frame we will have to use two dates, the first with the time set to `00:00:00` and the second with the time set to `23:59:59`, both with the same year, month and day.

The two dates will need to be handled using methods of the `Date` object which use the UTC format. If we didn't, we would encounter discrepancies due to the local environment in which Node.js runs.

Mongoose gives us the operators `$gte` (greater-than-equal) and `$lte` (less-than-equal) which operate on dates both when values are passed as strings and when they are instances of the `Date` object. Using them we can find those documents whose field of type `ISODate` falls within the specified time interval.

First we define two helper methods for validating the parameters passed to the ExpressJS API.

```
module.exports = {
    validYear(str) {
        if(!/^2\d{3}$/.test(str)) {
            return false;
        }
```

```
        const year = new Date().getFullYear();
        const min = year - 4; // Minimum year
        const value = parseInt(str, 10);

        return (value >= min && value <= year);
    },
    validValue(str, type = 'month') {
        if(!/^\d{1,2}$/.test(str)) {
            return false;
        }
        const value = parseInt(str, 10);
        let valid = true;
        switch (type) {
            case 'month':
                if(value < 0 || value > 11) {
                    valid = false;
                }
                break;
            case 'day':
                if(value < 1 || value > 31) {
                    valid = false;
                }
                break;
            default:
                break;
        }

        return valid;
    }
}
```

At this point we can define the API route that will search by date.

```
'use strict';

const express = require('express');
const router = express.Router();
const stats = require('../models/stats');
const utils = require('../lib/utils');

router.post('/search', async (req, res, next) => {
    const body = req.body;

    if(typeof body.year === 'undefined' || typeof
body.month === 'undefined' || typeof body.day ===
'undefined') {
        return res.json({
            error: 'Missing required parameters.'
        });
    }

    const { year, month, day } = body;


    if(!utils.validYear(year)) {
        return res.json({
            error: 'Invalid year parameter.'
        });
    }

    if(!utils.validValue(month, 'month')) {
        return res.json({
            error: 'Invalid month parameter.'
        });
    }

    if(!utils.validValue(day, 'day')) {
```

```
        return res.json({
            error: 'Invalid day parameter.'
        });
    }

    const dateStart = new Date();

    dateStart.setUTCFullYear(parseInt(year, 10));
    dateStart.setUTCMonth(parseInt(month, 10));
    dateStart.setUTCDate(parseInt(day, 10));

    dateStart.setUTCHours(0, 0, 0);

    const dateMax = new Date();

    dateMax.setUTCFullYear(parseInt(year, 10));
    dateMax.setUTCMonth(parseInt(month, 10));
    dateMax.setUTCDate(parseInt(day, 10));
    dateMax.setUTCHours(23, 59, 59);

    try {
        const query = { date: {
                $gte: dateStart, $lte: dateMax
        } };
        const results = await stats.find(query);
        res.json(results);
    } catch (err) {
        res.json(err);
    }
});

module.exports = router;
```

After validating the three parameters of the POST request, we create the two time intervals using these parameters. To the `find()` method of the chosen collection an object is passed that defines our query to the database.

The key to getting consistent results is to use the UTC format for dates. By doing so, we are certain that the local configuration of Node.js will not be used to perform these calculations.