NODE.JS*, JAVASCRIPT*, API*, БЛОГ КОМПАНИИ RUVDS.COM

Node.js, Express и MongoDB: API за полчаса

ПЕРЕВОД

ru_vds 3 февраля 2017 в 15:07 90,5k

Оригинал: Scott Domes

Начинающему программисту разработка для Node.js может показаться сущим кошмаром. Виной всему – гибкость этой платформы и отсутствие чётких руководств. Но, на самом деле, всё не так уж и страшно.

Вот, например, типичная задача: разработка REST API, серверной части некоего приложения. Обилие собственных возможностей Node и множество дополнительных модулей, которые способны

помочь в решении этой задачи, способны завести новичка в тупик, вызванный богатством выбора. Основные вопросы здесь заключаются в подборе компонентов и в настройке их совместной работы.

Один из способов создания серверной части приложения заключается в применении связки из Node.js, фреймворка Express и СУБД MongoDB. Собственно говоря, сегодня я расскажу о том, как создать рабочий макет API, который может служить основой для практически любого приложения. Здесь мы реализуем основные маршруты REST, будем взаимодействовать с API по HTTP и использовать простые варианты работы с базой данных.

Для того, чтобы успешно освоить этот материал, вам надо понимать, что такое REST API, иметь представление об операциях CRUD и обладать базовыми знаниями в области JavaScript. Здесь я использую ES6, ничего особенно сложного, в основном – стрелочные функции.

Мы разработаем скелет серверной части приложения для создания заметок, похожего на Google Keep. При этом с заметками можно будет выполнять все четыре CRUD-действия, а именно – создание (create), чтение (read), обновление (update), и удаление (delete).

Предварительная подготовка

Если Node у вас пока нет, самое время его установить. После установки создайте папку и выполните в ней команду

инициализации нового проекта:

```
npm init
```

В ходе инициализации ответьте на вопросы, в частности, дайте приложению имя «notable» (или, если хотите, любое другое).

Теперь в папке должен появиться файл *package.json*. Это означает, что можно начать устанавливать дополнительные пакеты, от которых зависит проект.

В качестве фреймворка мы планируем использовать Express. Системой управления базами данных будет MongoDB. Кроме того, в качестве вспомогательного средства для работы с JSON, используем пакет body-parser. Установим всё это:

```
npm install --save express mongodb body-parser
```

Ещё, я очень рекомендую установить Nodemon как dev-зависимость. Это простой маленький пакет, который, при изменении файлов, автоматически перезапускает сервер.

Для установки этого пакета выполните команду:

```
npm install --save-dev nodemon
```

Затем можно добавить следующий скрипт в файл *package.json*:

```
// package.json
  "scripts": {
    "dev": "nodemon server.js"
  },
```

Готовый *package.json* будет выглядеть примерно так:

```
// package.json
{
  "name": "notable",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "dev": "nodemon server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.15.2",
    "express": "^4.14.0",
    "mongodb": "^2.2.16"
  },
  "devDependencies": {
    "nodemon": "^1.11.0"
  }
}
```

Теперь создадим файл *server.js* и приступим к работе над API.

Сервер

Начнём с подключения зависимостей в файле *server.js*.

```
// server.js
const express        = require('express');
const MongoClient    = require('mongodb').MongoClient;
```

```
const bodyParser     = require('body-parser');
const app            = express();
```

MongoClient будем использовать для взаимодействия с базой данных. Кроме того, здесь мы инициализируем константу `app`, символизирующую наше приложение, экземпляром фреймворка Express. Чтобы сервер заработал, осталось лишь указать приложению на то, чтобы оно начало прослушивать HTTP-запросы.

Тут укажем порт и запустим прослушивание следующим образом:

```
// server.js
const port = 8000;
app.listen(port, () => {
  console.log('We are live on ' + port);
});
```

Теперь, если выполнить команду `npm run dev` (или – `node server.js`, если вы не устанавливали Nodemon), в терминале должно появиться сообщение: «We are live on port 8000».

Итак, сервер работает. Но сейчас он не делает совершенно ничего полезного. Давайте с этим разберёмся.

Маршруты, ориентированные на CRUD-операции

Мы планируем создать 4 маршрута. А именно:

- CREATE – создание заметок.

- READ –чтение заметок.
- UPDATE –обновление заметок.
- DELETE –удаление заметок.

Освоив эту схему, вы сможете понять, как, с помощью Node, организовать практически любой необходимый REST-маршрут.

Для того, чтобы протестировать API, понадобится нечто, способное имитировать запросы клиентской части приложения. Решить эту задачу нам поможет отличная программа, которая называется Postman. Она позволяет выполнять простые HTTP-запросы с заданным телом и параметрами.

Установите Postman. Теперь всё готово к настройке маршрутов.

О структуре проекта

В большинстве руководств по Node.js (и во множестве реальных приложений) все маршруты размещают в одном большом файле *route.js*. Мне такой подход не очень нравится. Если разложить файлы по разным папкам, это улучшит читаемость кода, приложением будет легче управлять.

Наше приложение большим не назовёшь, но предлагаю сделать всё как надо, учитывая, всё же, его скромные масштабы. Создайте следующие папки: папку *app*, а внутри неё – *routes*. В папке *routes* создайте файлы *index.js* и *note_routes.js*. Другими словами, структура проекта будет выглядеть так: *root* > *app* > *routes* >

index.js и *note_routes.js*.

```
mkdir app
cd app
mkdir routes
cd routes
touch index.js
touch note_routes.js
```

Такая структура, для маленького приложения, может показаться избыточной, но она окажется очень кстати в более крупной системе, построенной на базе нашего примера. К тому же, любые проекты лучше всего начинать, используя лучшие из существующих наработок.

Создание заметок: маршрут CREATE

Начнём с маршрута CREATE. Для этого ответим на вопрос: «Как создать заметку?».
Прежде чем приступить к созданию заметок, нам понадобится расширить инфраструктуру приложения. В Express маршруты оборачивают в функцию, которая принимает экземпляр Express и базу данных как аргументы.

Выглядеть это может так:

```
// routes/note_routes.js
module.exports = function(app, db) {
};
```

Теперь можно экспортировать эту функцию через *index.js*:

```
// routes/index.js
const noteRoutes = require('./note_routes');
module.exports = function(app, db) {
  noteRoutes(app, db);
  // Тут, позже, будут и другие обработчики маршрутов
};
```

Импортируем то, что получилось, в *server.js*:

```
// server.js
const express        = require('express');
const MongoClient    = require('mongodb').MongoClient;
const bodyParser     = require('body-parser');
const app            = express();
const port = 8000;
require('./app/routes')(app, {});
app.listen(port, () => {
  console.log('We are live on ' + port);
});
```

Обратите внимание на то, что так как базу данных мы пока не настроили, вторым аргументом передаётся пустой объект.

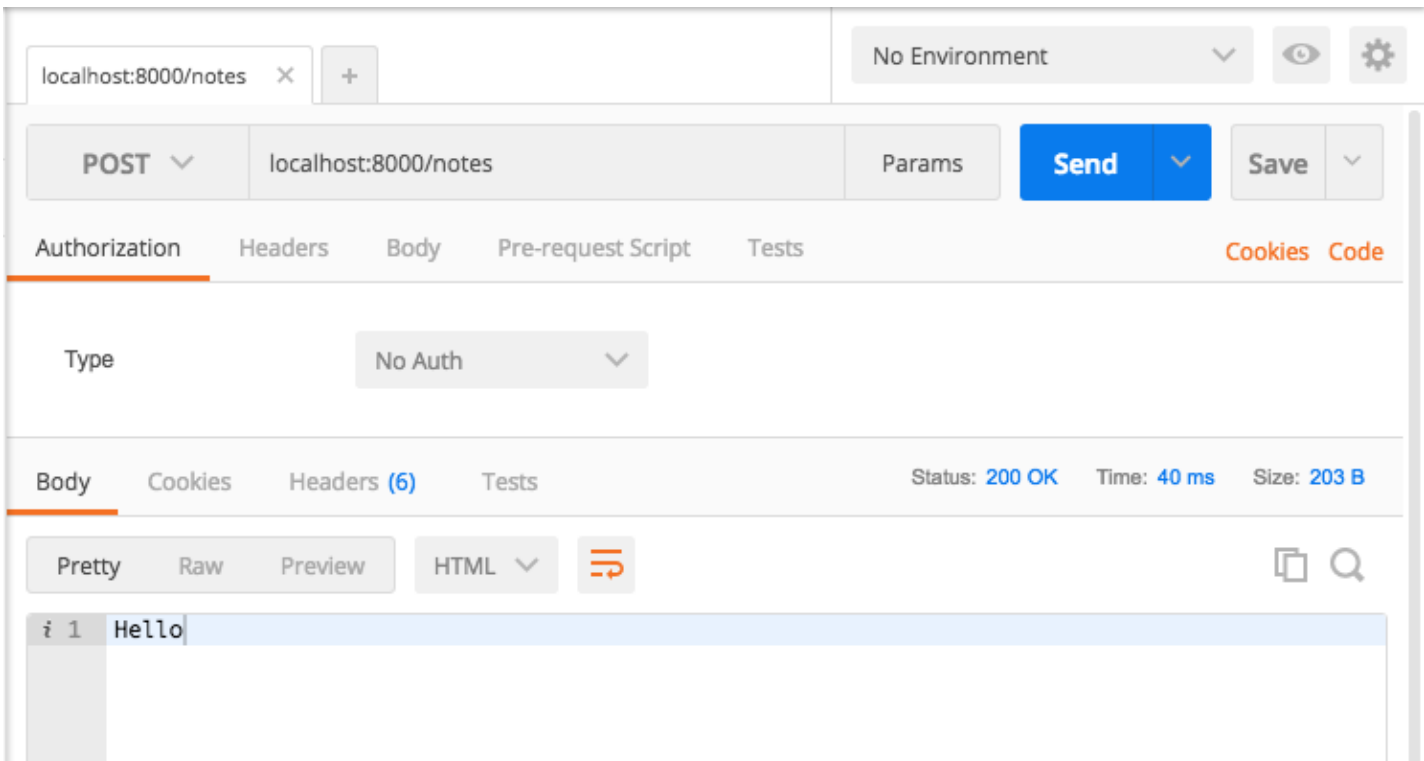
Теперь создаём маршрут CREATE. Синтаксис здесь довольно простой:

```
module.exports = function(app, db) {
  app.post('/notes', (req, res) => {
    // Здесь будем создавать заметку.
    res.send('Hello')
  });
};
```

Когда приложение получает POST-запрос по пути ‘/notes’, оно

исполнит код внутри функции обратного вызова, передав ей объект запроса (который содержит параметры запроса или JSON-данные) и объект ответа (который, понятно, используется для ответа).

То, что у нас получилось, уже можно протестировать. Отправим, с помощью Postman, POST-запрос по адресу **localhost:8000/notes**.



В ответ на запрос должно прийти «Hello»

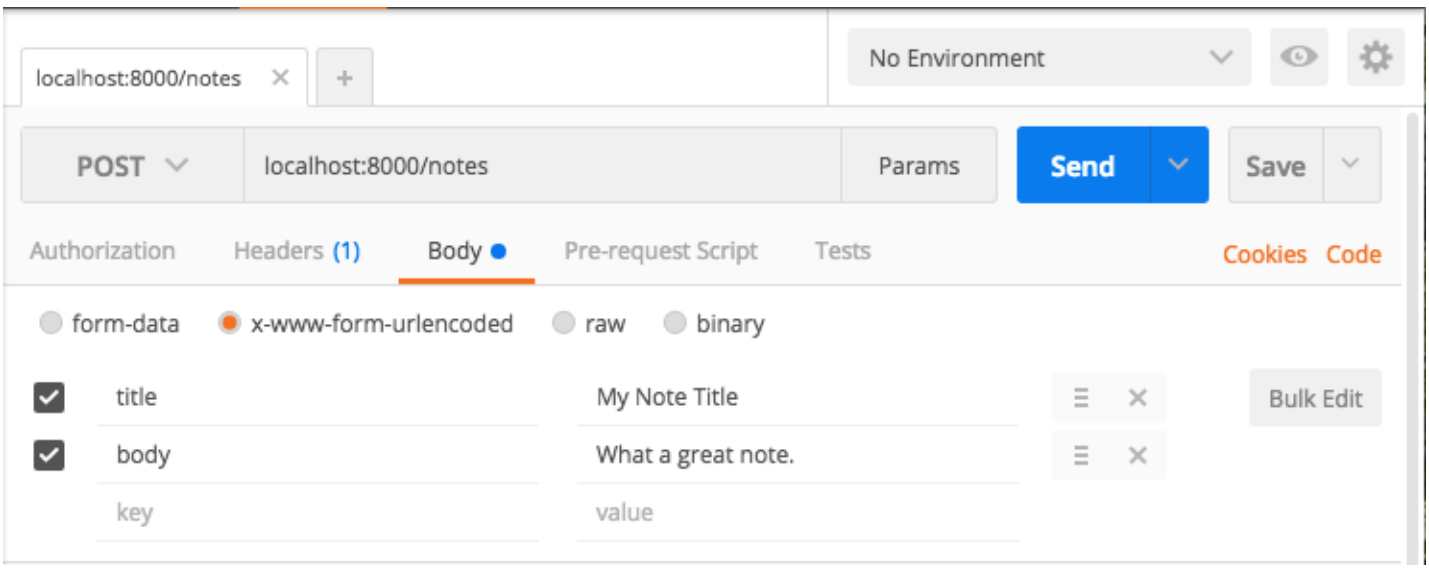
Отлично. Первый маршрут создан. Следующий шаг – добавление к запросу параметров, обработка их в API, и, наконец – сохранение заметки в базе данных.

Параметры запроса

В Postman перейдите на вкладку **Body** и добавьте несколько пар ключ-значение, выбрав радиокнопку **x-www-form-urlencoded**. А

именно, первым ключом будет **title**, его значение – **My Note Title**. Второй ключ – **body**, его значение – **What a great note**.

Это добавит к запросу закодированные данные, которые можно будет обработать в API.



Заголовок моей заметки, да и она сама – очень просты, а вы тут можете проявить фантазию

В файле *note_route.js*, просто выведем тело заметки в консоль.

```
// note_routes.js
module.exports = function(app, db) {
  app.post('/notes', (req, res) => {
    console.log(req.body)
    res.send('Hello')
  });
};
```

Попробуйте отправить запрос с помощью Postman, и вы увидите… `undefined`.

К сожалению, Express не может самостоятельно обрабатывать

формы в URL-кодировке. Тут нам на помощь придёт ранее установленный пакет body-parser.

```
// server.js
const express        = require('express');
const MongoClient    = require('mongodb').MongoClient;
const bodyParser     = require('body-parser');
const app            = express();
const port = 8000;
app.use(bodyParser.urlencoded({ extended: true }));
require('./app/routes')(app, {});
app.listen(port, () => {
  console.log('We are live on ' + port);
});
```

Теперь, после выполнения POST-запроса, его тело можно будет увидеть в терминале в виде объекта.

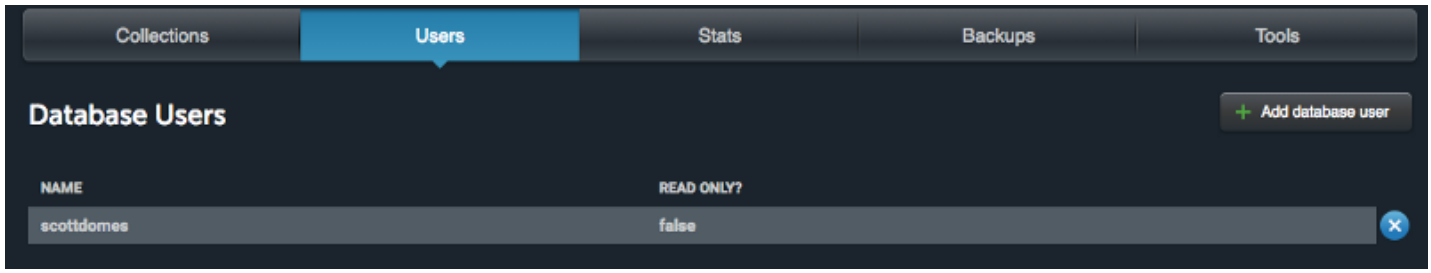
```
{ title: 'My Note Title', body: 'What a great note.' }
```

Чтобы первый маршрут полностью заработал, осталось лишь настроить базу данных и добавить в неё заметку.

Для быстрого создания и настройки базы данных воспользуемся сервисом mLab. Работать с ним легко, для маленьких объёмов информации он бесплатен.

Создайте учётную запись на сайте mLab и разверните новую базу данных MongoDB. Для этого нажмите на кнопку **Create New** в разделе **MongoDB Deployments**, в появившемся окне, в разделе **Plan**, выберите **Single-node**. В списке **Standard Line**, выберите **Sandbox** и дайте базе данных имя. Далее, в окне управления

базой, перейдите на вкладку **Users** и добавьте пользователя базы данных, задав имя и пароль.



Новый пользователь базы данных

Скопируйте с той же страницы второй URL – строку подключения к базе данных.

URL для подключения к базе данных

В корень проекта добавьте директорию *config*, создайте в ней файл *db.js*.

```
mkdir config
cd config
touch db.js
```

В файл *db.js* добавьте следующее:

```
module.exports = {
  url : здесь будет ваш URL
};
```

Не забудьте добавить в URL имя пользователя и пароль (не те, что от учётной записи в mLab, а те, что создавали для базы данных). Если вы размещаете проект на Github, не забудьте включить в него файл *.gitignore* (вроде этого). Так вы не сделаете всеобщим достоянием имя и пароль для работы с базой.

Теперь, в *server.js*, можно использовать MongoClient для подключения к базе данных и обернуть в функцию, которая передаётся ему при создании, настройки приложения:

```
// server.js
const express        = require('express');
const MongoClient    = require('mongodb').MongoClient;
const bodyParser     = require('body-parser');
const db             = require('./config/db');
const app            = express();
const port = 8000;
app.use(bodyParser.urlencoded({ extended: true }));
MongoClient.connect(db.url, (err, database) => {
  if (err) return console.log(err)
  require('./app/routes')(app, database);
  app.listen(port, () => {
    console.log('We are live on ' + port);
  });
})
```

На этом подготовка инфраструктуры закончена. С этого момента будем заниматься исключительно путями.

Добавление записей в базу данных

MongoDB хранит данные в коллекциях (collections), которые полностью оправдывают своё название. В нашем случае заметки

будут храниться в коллекции, которая, как несложно догадаться, будет называться `notes`.

В ходе настройки клиента, ему была передана строка подключения к базе данных, аргумент `db`. В коде маршрутов доступ к базе можно получить так:

```
db.collection('notes')
```

Создание заметки в базе равносильно вызову команды `insert` для коллекции `notes`:

```
const note = { text: req.body.body, title: req.body.title}
  db.collection('notes').insert(note, (err, results) => {
}
```

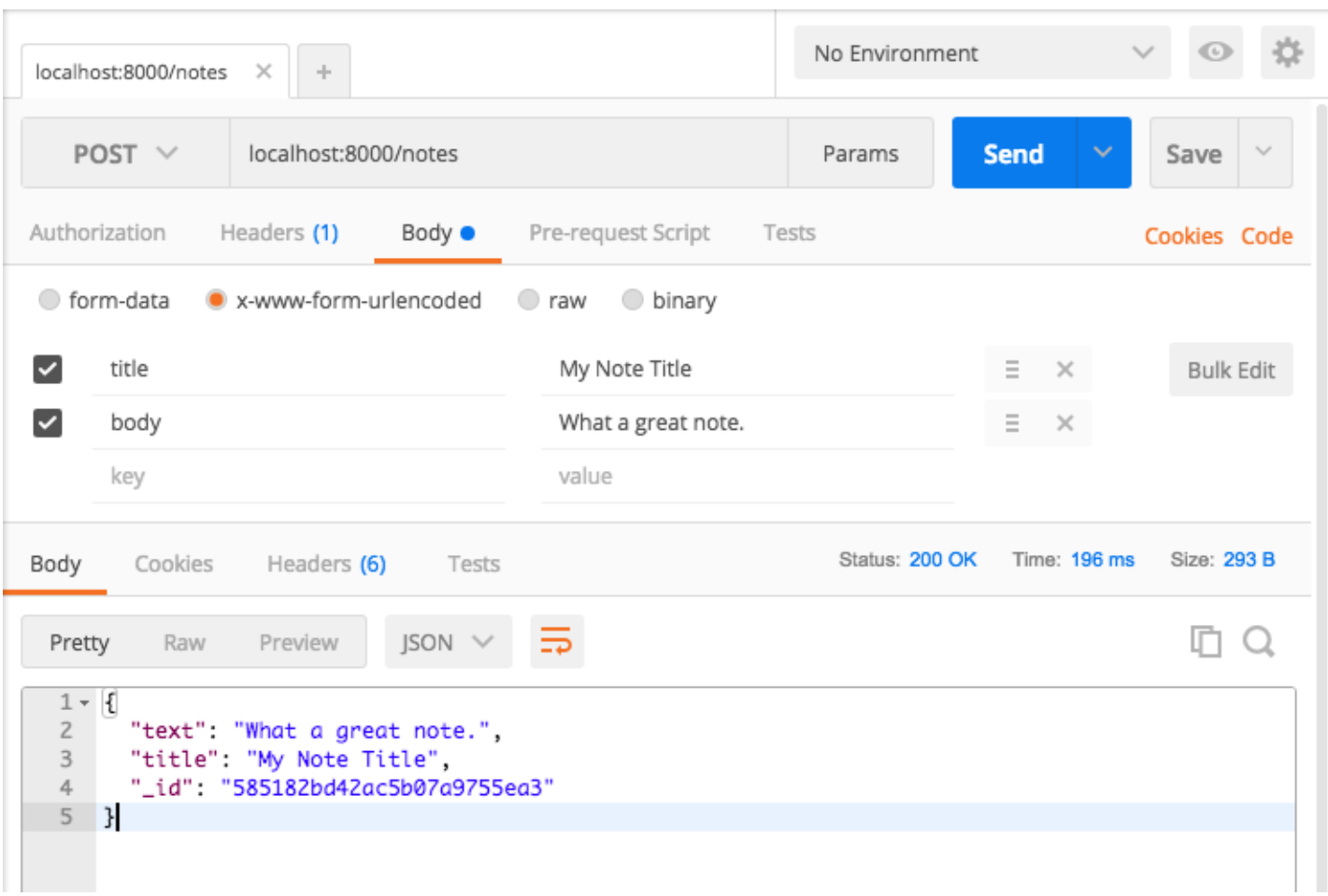
После успешного завершения команды (или после того, как она, по какой-нибудь причине, не сможет выполниться), нужно либо отправить в ответ только что созданный объект заметки, либо – сообщение об ошибке. Вот код *note_routes.js*, дополненный с учётом этих рассуждений:

```
// note_routes.js
module.exports = function(app, db) {
  app.post('/notes', (req, res) => {
    const note = { text: req.body.body, title: req.body.title };
    db.collection('notes').insert(note, (err, result) => {
      if (err) {
        res.send({ 'error': 'An error has occurred' });
      } else {
        res.send(result.ops[0]);
      }
    });
```

```
  });
};
```

Испытайте то, что получилось. Отправьте из Postman POST-запрос (с флагом **x-www-form-urlencoded**), задав на вкладке **Body** значения полей **title** и **body**.

Ответ должен выглядеть примерно так:



Успешное добавление записи в базу

Если теперь посмотреть на базу, авторизовавшись на mLab, в ней можно будет найти только что созданную заметку.

Чтение заметок: маршрут READ

Инфраструктура, которую мы подготовили выше, подходит для всех маршрутов, поэтому теперь дело пойдёт быстрее.

Итак, мы собираемся запросить только что созданную заметку, пройдя по пути **localhost:8000/notes/{id заметки}**. В нашем случае путь будет выглядеть так: **localhost:8000/notes/585182bd42ac5b07a9755ea3**.

Если ID одной из уже созданных заметок у вас нет, можете заглянуть в базу на mLab и найти его там, или создать новую заметку и скопировать её идентификатор.

Вот как это выглядит в *note_route.js*:

```
// note_routes.js
module.exports = function(app, db) {
  app.get('/notes/:id', (req, res) => {

  });
  app.post('/notes', (req, res) => {
    const note = { text: req.body.body, title: req.body.title };
    db.collection('notes').insert(note, (err, result) => {
      if (err) {
        res.send({ 'error': 'An error has occurred' });
      } else {
        res.send(result.ops[0]);
      }
    });
  });
};
```

Так же, как и раньше, мы собираемся вызвать некую команду для коллекции базы данных заметок. Применим для этого метод

findOne.

```
// note_routes.js
module.exports = function(app, db) {
  app.get('/notes/:id', (req, res) => {
    const details = { '_id': <ТУТ БУДЕТ ID> };
    db.collection('notes').findOne(details, (err, item) => {
      if (err) {
        res.send({'error':'An error has occurred'});
      } else {
        res.send(item);
      }
    });
  });
app.post('/notes', (req, res) => {
    const note = { text: req.body.body, title: req.body.title };
    db.collection('notes').insert(note, (err, result) => {
      if (err) {
        res.send({ 'error': 'An error has occurred' });
      } else {
        res.send(result.ops[0]);
      }
    });
  });
};
```

Идентификатор из параметров URL можно вытащить с помощью конструкции `req.params.id`. Однако если просто вставить строку вместо <<>> из кода выше, работать это не будет.

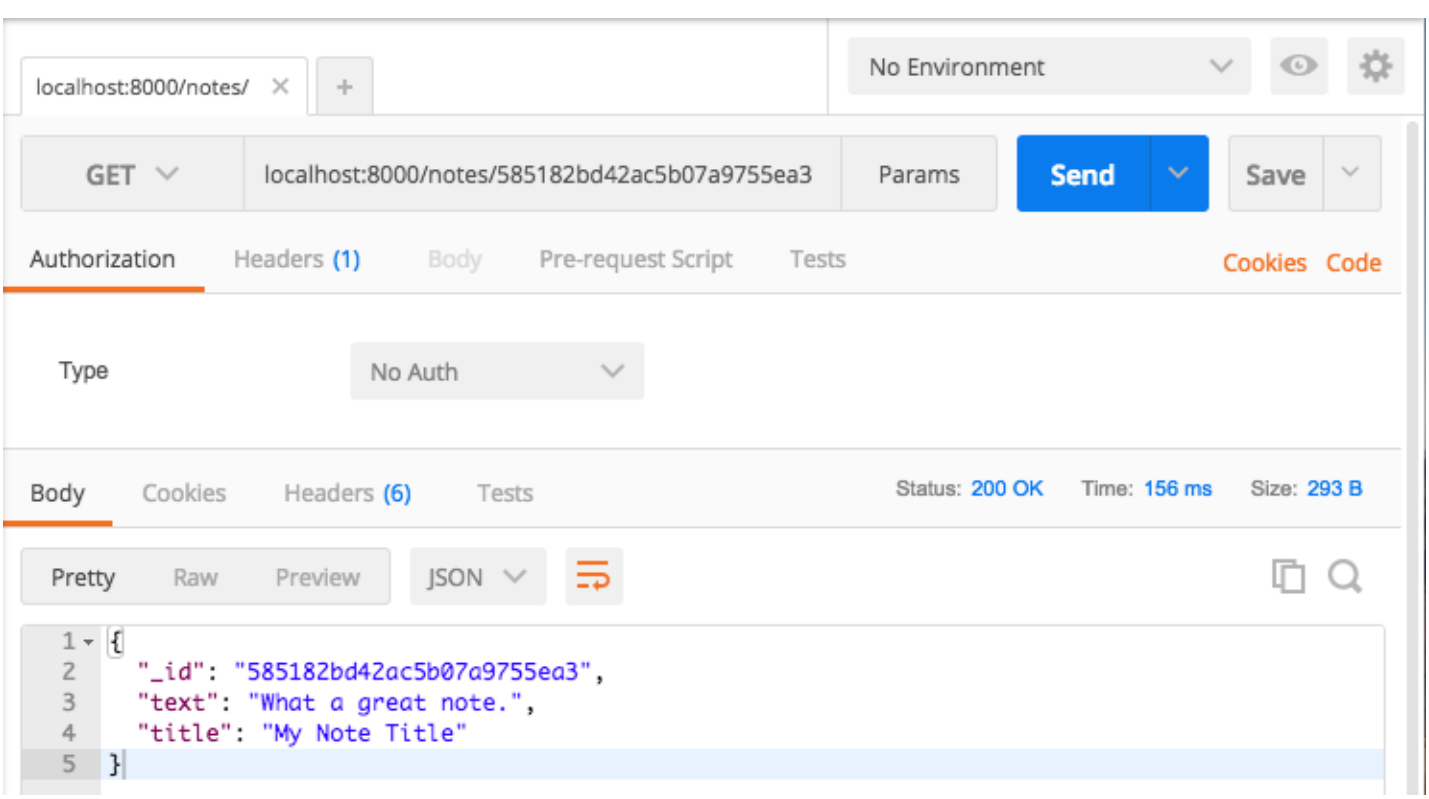
MongoDB требуется ID не в виде строки, а в виде специального объекта. Он называется **ObjectID**.

Вот что, после небольших изменений, у нас получилось:

```
// note_routes.js
var ObjectID = require('mongodb').ObjectID;
module.exports = function(app, db) {
```

```
  app.get('/notes/:id', (req, res) => {
    const id = req.params.id;
    const details = { '_id': new ObjectID(id) };
    db.collection('notes').findOne(details, (err, item) => {
      if (err) {
        res.send({'error':'An error has occurred'});
      } else {
        res.send(item);
      }
    });
  });
app.post('/notes', (req, res) => {
    const note = { text: req.body.body, title: req.body.title };
    db.collection('notes').insert(note, (err, result) => {
      if (err) {
        res.send({ 'error': 'An error has occurred' });
      } else {
        res.send(result.ops[0]);
      }
    });
  });
};
```

Испытайте это с одним из идентификаторов заметок, имеющихся в базе данных. Ответ в Postman должен выглядеть так:



Успешный запрос заметки из базы

Удаление заметок: маршрут DELETE

Удаление объектов – это практически то же самое, что их поиск в базе. Только вместо функции `findOne` используется функция `remove`. Вот полный код соответствующего пути. Здесь выделено то, что отличается от кода уже существующего метода, обрабатывающего запрос GET.

```
// note_routes.js
// ...
  app.delete('/notes/:id', (req, res) => {
    const id = req.params.id;
    const details = { '_id': new ObjectID(id) };
    db.collection('notes').remove(details, (err, item) => {
      if (err) {
        res.send({'error':'An error has occurred'});
      } else {
        res.send('Note ' + id + ' deleted!');
      }
    });
  });
// ...
```

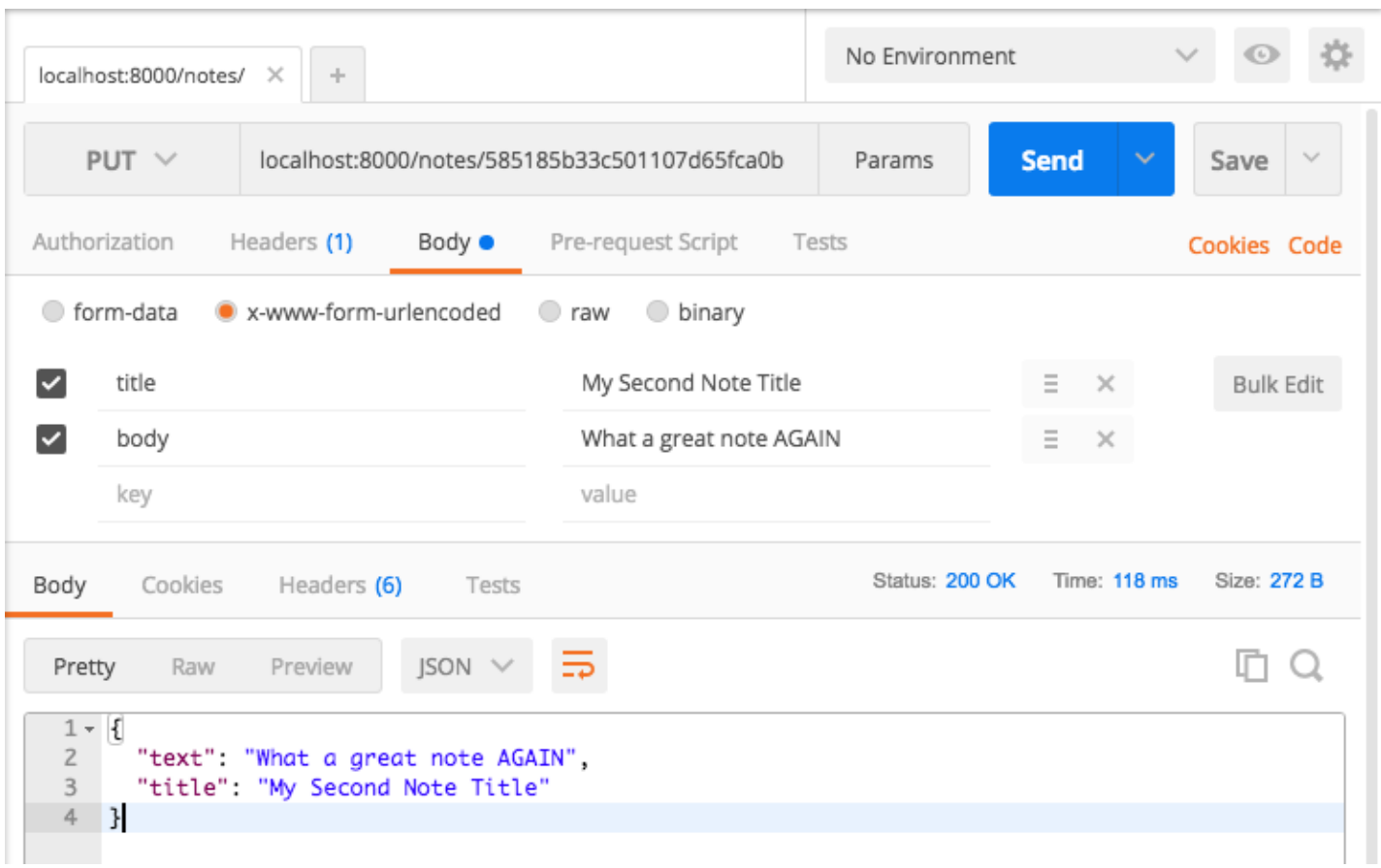
Обновление заметок: маршрут UPDATE

А вот и последний маршрут. Обработка запроса PUT – это, по сути, гибрид операций READ и CREATE. Сначала надо найти объект, потом – обновить его в соответствии с поступившими в запросе данными. Сейчас, если вы, испытывая предыдущий фрагмент кода, удалили свою единственную заметку, создайте ещё одну.

Вот код маршрута обновления заметок:

```
// note_routes.js
// ...
  app.put ('/notes/:id', (req, res) => {
    const id = req.params.id;
    const details = { '_id': new ObjectID(id) };
    const note = { text: req.body.body, title: req.body.title };
    db.collection('notes').update(details, note, (err, result) => {
      if (err) {
          res.send({'error':'An error has occurred'});
      } else {
          res.send(note);
      }
    });
  });
// ...
```

Теперь любую заметку можно редактировать. Вот, как это выглядит:



Успешное обновление заметки

Обратите внимание на недостаток нашего примера. Если в PUT-запросе не будет тела или заголовка заметки, соответствующие поля в базе будут просто очищены.

Я не нагружал пример дополнительными проверками. Если хотите, можете сами доработать операцию обновления заметок, внося в базу новые данные только в том случае, если запрос сформирован верно.

Итоги

Теперь у вас есть рабочее Node API, которое поддерживает четыре основные операции CRUD. Серверная часть приложения умеет, реагируя на HTTP-запросы клиента, создавать в базе данных заметки, находить их, удалять и редактировать.

Основная цель моего рассказа – познакомить всех желающих со связкой Node + Express + MongoDB и с методикой разработки серверных приложений. Конечно, если сегодня состоялось ваше первое знакомство с этими инструментами, для того, чтобы во всё лучше вникнуть, вам понадобится почитать документацию. Однако, понимание происходящего позволит вам быстро восполнить пробелы в знаниях и приступить к работе над собственными проектами, использовав, в качестве отправной точки, приложение, которым мы с вами тут занимались.

Если у вас есть опыт работы с Node.js, Express и MongoDB в

реальных проектах, может быть, вы посоветуете что-нибудь полезное новичкам? А если вы только что всё это впервые попробовали – ждём ваших впечатлений.

Проголосовать:

+21

Поделиться:

Сохранить:

Комментарии (27)

Похожие публикации

Веб-камера, Node.js и OpenCV: делаем систему распознавания лиц

6

ПЕРЕВОД

ru_vds • 17 августа 2017 в 12:00

Аутентификация в Node.js. Учебные руководства и возможные ошибки

3

ПЕРЕВОД

ru_vds • 11 августа 2017 в 15:11

Пишем симпатичные Node.js-API с использованием async/await и базы данных Firebase

6

ПЕРЕВОД

ru_vds • 10 июля 2017 в 14:19

Популярное за сутки

Яндекс открывает Алису для всех разработчиков. Платформа Яндекс.Диалоги (бета)

69

BarakAdama • вчера в 10:52

Почему следует игнорировать истории основателей успешных стартапов

20

ПЕРЕВОД

m1rko • вчера в 10:44

Как получить телефон (почти) любой красотки в Москве, или интересная особенность MT_FREE

24

ИЗ ПЕСОЧНИЦЫ

cab404 • вчера в 20:27

Java и Project Reactor

10

zealot_and_frenzy • вчера в 10:56

Пользовательские агрегатные и оконные функции в PostgreSQL и Oracle

6

erogov • вчера в 12:46

Лучшее на Geektimes

Как фермеры Дикого Запада организовали телефонную сеть на колючей проволоке 31

NAGru • вчера в 10:10

Энтузиаст сделал новую материнскую плату для ThinkPad X200s 49

alizar • вчера в 15:32

Кто-то посылает секс-игрушки с Amazon незнакомцам. Amazon не знает, как их остановить 85

Pochtoycom • вчера в 13:06

Илон Маск продолжает убеждать в необходимости создания колонии людей на Марсе 139

marks • вчера в 14:19

Дела шпионские (часть 1) 16

TashaFridrih • вчера в 13:16