Creating RESTful APIs with NodeJS and MongoDB Tutorial

This exercise is about creating a RESTful API using Node.js (Express.js) and MongoDB (mongoose)! You will learn how to install and use each component individually and then proceed to create a RESTful API.

1 What is a RESTful API?

REST stands for Representational State Transfer. It is an architecture that allows `client-server` communication through a uniform interface. REST is *stateless*, *cachable* and has property called *idempotence*; the side effect of identical requests have the same side-effect as a single request.

HTTP RESTful API's are composed of:

HTTP methods — e.g., GET, PUT, DELETE, PATCH, POST, ...

Base URI — e.g., `http://www.dcs.bbk.ac.uk`

URL path — e.g., `/path/creating-a-restful-api-tutorial-with-nodejs-and-mongodb/`

Media type — e.g., `html`, `JSON`, `XML`, `Microformats`, `Atom`, `Images`, ...

Here is a summary what we want to implement:

Resource (URI)	POST (create)	GET (read)	PUT (update)	DELETE (destroy)
/todos	create new task	list tasks	N/A (update all)	N/A (destroy all)
/todos/1	error	show task ID 1	update task ID 1	destroy task ID 1

NOTE for this tutorial:

- The format will be JSON.
- Bulk updates and bulk destroys are not safe, so we will not be implementing those.
- **CRUD** functionality: POST == **C**REATE, GET == **R**EAD, PUT == **U**PDATE, DELETE == **D**ELETE.

2 Installing the MEAN Stack backend

In this section, we are going to install the main backend components of the MEAN stack: MongoDB, NodeJS and ExpressJS.

2.1 Installing MongoDB

MongoDB is a document-oriented NoSQL database (Big Data ready). It stores data in JSON-like format and allows users to perform SQL-like queries against it.

You can install MongoDB following these instructions.

If you have a **Mac** and brew it's just:

```
brew install mongodb && mongod
```

In **Ubuntu**:

```
sudo apt-get -y install mongodb
```

In Windows you need to simply double-click the downloaded `msi` file.

After you have installed MongoDB you should then check the version:

```
mongod --version

# => db version v2.6.4
# => 2014-10-01T19:07:26.649-0400 git version: nogitversion
```

2.2 Installing NodeJS

The Node official description is:

> Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.
>
> **Node.js Website**nodejs.org

In short, NodeJS allows you to run Javascript outside the browser, in this case, on the web server. NPM allows you to install/publish node packages with ease.

To install it, you can go to the NodeJS Website.

Since Node versions changes very often. You can use the NVM (Node Version Manager) with:

```
# *nix version - Windows is slightly different (see the distribution download page for details)
# download NPM
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.4/install.sh | bash

# load NPM
export NVM_DIR="$HOME/.nvm"
```

```
[ -s "$NVM_DIR/nvm.sh" ] && . "$NVM_DIR/nvm.sh" # This loads nvm

# Install latest stable version
nvm install stable
```

Check out `https://github.com/creationix/nvm` for more details.

On the **Mac** with Homebrew you can do:

```
brew install nodejs
```

After you have it installed, check the version of node and npm (node package manager):

```
node -v
# => v6.2.2
npm -v
# => 3.9.5
```

or something similar.

2.3 Installing ExpressJS

ExpressJS is a web application framework that runs on NodeJS. It allows you to build web applications and API endpoints. (more details on this later).

We are going to create a project folder `first`, and then add `express` as a dependency. Let's use NPM init command to get us started.

```
# create project folder
mkdir todo-app

# move to the folder and initialise the project
cd todo-app
npm init .

# press enter multiple times to accept all defaults
# install express and save it as dependency
npm install express --save
```

Notice that after the last command, `express` should be added to `package.json` with the version the appropriate version information.

3 Using MongoDB with Mongoose

Mongoose is an NPM package that allows you to interact with MongoDB. You can install it as follows:

```
npm install mongoose --save
```

If you followed the previous steps, you should have all the software you need to complete this tutorial. We are going to build an API that allow users to CRUD (Create-Read-Update-Delete) Todo tasks from a database.

3.1 Mongoose CRUD

CRUD == **C**reate-**R**ead-**U**pdate-**D**elete

We are going to create, read, update and delete data from MongoDB using Mongoose/Node. First, you need to have `mongodb` up and running:

```
# run mongo daemon
mongod
```

Keep mongo running in a terminal window and while in the folder `todoApp` type `node` to enter the node command line interpreter (CLI). Then type:

```
// Load mongoose package
var mongoose = require('mongoose');

// Connect to MongoDB and create/use database called todoApp
mongoose.connect('mongodb://localhost/todoApp');

// Create a schema
var TodoSchema = new mongoose.Schema({
  name: String,
  completed: Boolean,
  note: String,
  updated_at: { type: Date, default: Date.now }
});

// Create a model based on the schema
var Todo = mongoose.model('Todo', TodoSchema);
```

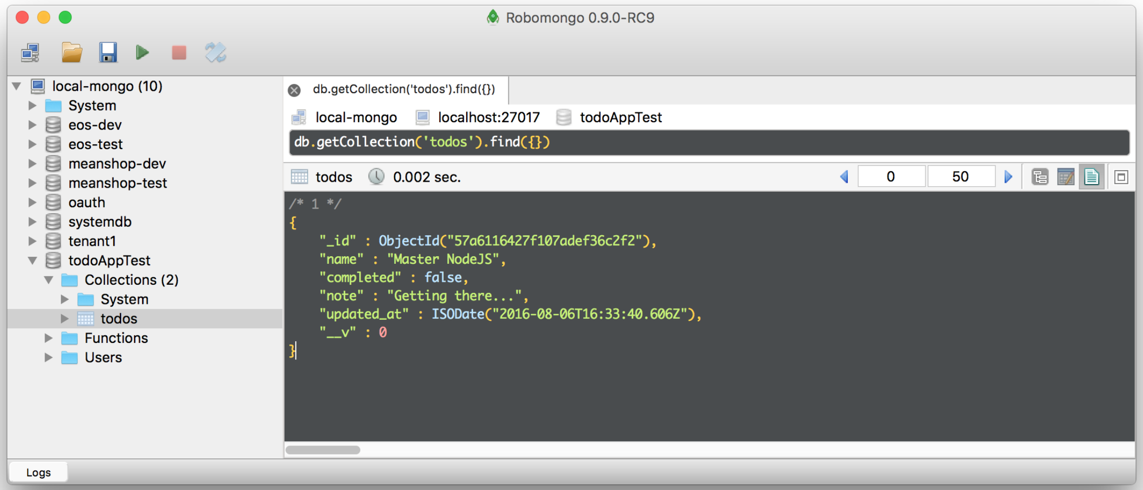
Now, let's test that we can save and edit data.

3.2 Mongoose Create

```
// Create a todo in memory
var todo = new Todo({name: 'Master NodeJS', completed: false,
  note: 'Getting there...'});

// Save it to the database
todo.save(function(err){
  if(err)
    console.log(err);
  else
    console.log(todo);
});
```

If you take a look at Mongo you will notice that we just created an entry. You can easily visualise the data using a tool such as Robomongo.



```
db.getCollection('todos').find({})
```

```
/* 1 */
{
    "_id" : ObjectId("57a6116427f107adef36c2f2"),
    "name" : "Master NodeJS",
    "completed" : false,
    "note" : "Getting there...",
    "updated_at" : ISODate("2016-08-06T16:33:40.606Z"),
    "__v" : 0
}
```

You can also build the object and save it in one step using `create`:

```
Todo.create({name: 'Create something with Mongoose', completed: true,
  note: 'this is one'}, function(err, todo){
    if(err) console.log(err);
    else console.log(todo);
});
```

3.3 Mongoose Read and Query

So far we have been able to save data, now we are going explore how to query the information. There are multiple options for reading/querying data:

- `Model.find(conditions, {[}fields{]}, {[}options{]}, {[}callback{]})`
- `Model.findById(id, {[}fields{]}, {[}options{]}, {[}callback{]})`
- `Model.findOne(conditions, {[}fields{]}, {[}options{]}, {[}callback{]})`

Some examples:

```
// Find all data in the Todo collection
Todo.find(function (err, todos) {
  if (err) return console.error(err);
  console.log(todos)
});
```

The result is something like this:

```
[ { _id: 57a6116427f107adef36c2f2,
  name: 'Master NodeJS',
  completed: false,
  note: 'Getting there...',
  __v: 0,
  updated_at: 2016-08-06T16:33:40.606Z },
  { _id: 57a6142127f107adef36c2f3,
  name: 'Create something with Mongoose',
  completed: true,
  note: 'this is one',    __v: 0,
  updated_at: 2016-08-06T16:45:21.143Z } ]
```

You can also add queries:

```
// callback function to avoid duplicating it all over
var callback = function (err, data) {
  if (err) { return console.error(err); }
  else { console.log(data); }}

  // Get ONLY completed tasks
  Todo.find({completed: true }, callback);

  // Get all tasks ending with ‘JS‘
  Todo.find({name: /JS$/ }, callback);
```

You can chain multiple queries, e.g.:

```
var oneYearAgo = new Date();
oneYearAgo.setYear(oneYearAgo.getFullYear() - 1);

// Get all tasks staring with ‘Master‘, completed
Todo.find({name: /^Master/, completed: true }, callback);

// Get all tasks staring with ‘Master‘, not completed and created within the last year
Todo.find({name: /^Master/, completed: false })
  .where('updated_at')
  .gt(oneYearAgo)
  .exec(callback);
```

The MongoDB query language is very powerful. We can combine regular expressions, date comparison and more.

3.4 Mongoose Update

Moving on, we are now going to explore how to update data.

Each model has an `update` method which accepts multiple updates (for batch updates, because it doesn't return an array with data).

- `Model.update(conditions, update, {[}options{]}, {[}callback{]})`
- `Model.findByIdAndUpdate(id, {[}update{]}, {[}options{]}, {[}callback{]})`
- `Model.findOneAndUpdate({[}conditions{]}, {[}update{]}, {[}options{]}, {[}callback{]})`

Alternatively, the method `findOneAndUpdate` could be used to update just one and return an object.

```
// Model.update(conditions, update, [options], [callback])
// update ‘multiple tasks from complete false to true

Todo.update({ name: /master/i }, { completed: true }, { multi: true }, callback);

//Model.findOneAndUpdate([conditions], [update], [options], [callback])
Todo.findOneAndUpdate({name: /JS$/ }, {completed: false}, callback);
```

As you might have noticed the batch updates (`multi: true`) doesn't show the data, but instead displays the number of fields that were modified.

```
{ ok: 1, nModified: 1, n: 1 }
```

Here is what those fields mean:

- `n` means the number of records that matches the query
- `nModified` represents the number of documents that were modified with update query.

3.5 Mongoose Delete

`update` and `remove` mongoose API are identical, the only difference it is that no elements are returned. Try it on your own ;)

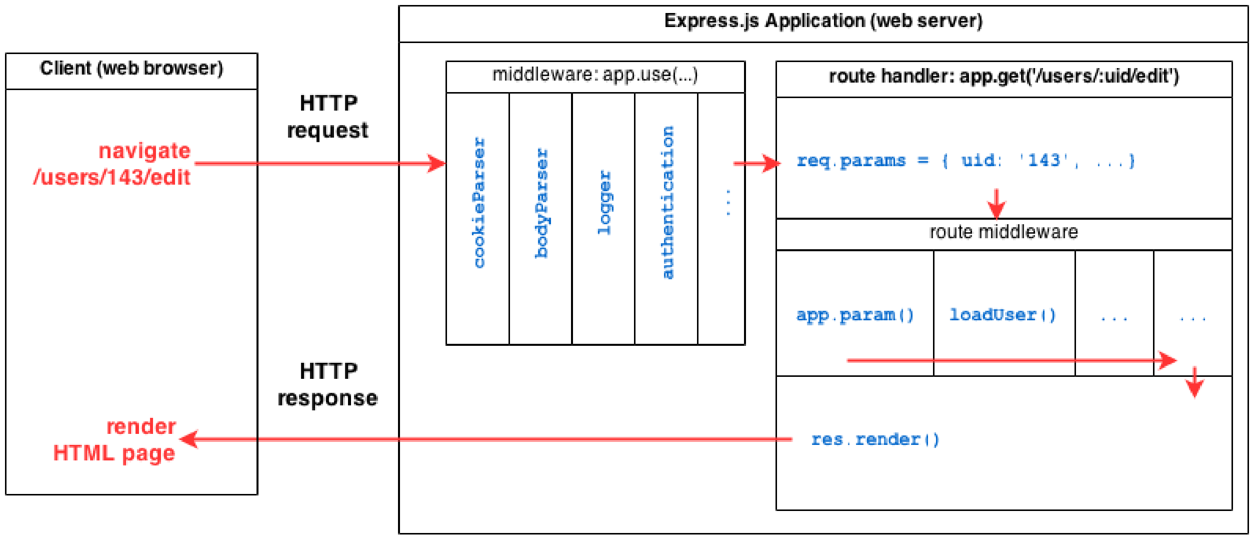
- `Model.remove(conditions, {[}callback{]})`
- `Model.findByIdAndRemove(id, {[}options{]}, {[}callback{]})`
- `Model.findOneAndRemove(conditions, {[}options{]}, {[}callback{]})`

4 ExpressJS and Middleware

ExpressJS is a complete web framework solution. It has HTML template solutions (jade, ejs, handlebars, hogan.js) and CSS pre-compilers (less, stylus, compass). Through middleware layers, it handles: cookies, sessions, caching, Cross-Site Request Forgery (CSRF), compression, and much more.

Middleware in this context are pluggable processors that runs on each request made to the server. You can have any number of middleware that will process the request one by one in a serial fashion. Some middleware might alter the request input. Others, might create log outputs, add data and pass it to the `next()` middleware in the chain.

We can use the middleware using `app.use`. That will apply for all request. If you want to be more specific, you can use `app.`*verb*. For instance: `app.get`, `app.delete`, `app.post`, `app.update`, . . .



Here are some examples of middleware layers to illustrate this point. Say you want to log the IP of the client on each request:

```
app.use(function (req, res, next) {
  var ip = req.headers['x-forwarded-for'] || req.connection.remoteAddress;
  console.log('Client IP:', ip);  next();
});
```

Notice that each middleware layer has three parameters:

- `req`: contain all the requests objects like URLs, path, ...
- `res`: is the response object where we can send the reply back to the client.
- `next`: continue with the next middleware in the chain.

You can also specify a path that you want the middleware to activate on, in this case the middleware layer is on "/todos/:id" and logs the request method:

```
app.use('/todos/:id', function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});
```

And finally you can use `app.get` to catch GET requests with matching routes, reply the request with a `response.send` and end the middleware chain. Let's use what we learned previously concerning reading using mongoose to reply with the user's data that matches the `id`.

```
app.get('/todos/:id', function (req, res, next) {
  Todo.findById(req.params.id, function(err, todo){
    if(err) res.send(err);
    res.json(todo);
  });
});
```

Notice that all previous middleware layers called `next()` except this last one, because it sends a response (in JSON) to the client with the requested `todo` data.

Hopefully, you don't have to develop a bunch of middleware besides routes, since ExpressJS has several middleware layers available.

4.1 Default Express Middleware

- morgan: logger
- body-parser: parse the body so you can access parameters in requests in `req.body`. e.g. `req.body.name`.
- cookie-parser: parse the cookies so you can access parameters in cookies `req.cookies`. e.g. `req.cookies.name`.
- serve-favicon: exactly that, serve favicon from route `/favicon.ico`. Should be call on the top before any other routing/middleware takes place to avoids unnecessary parsing.

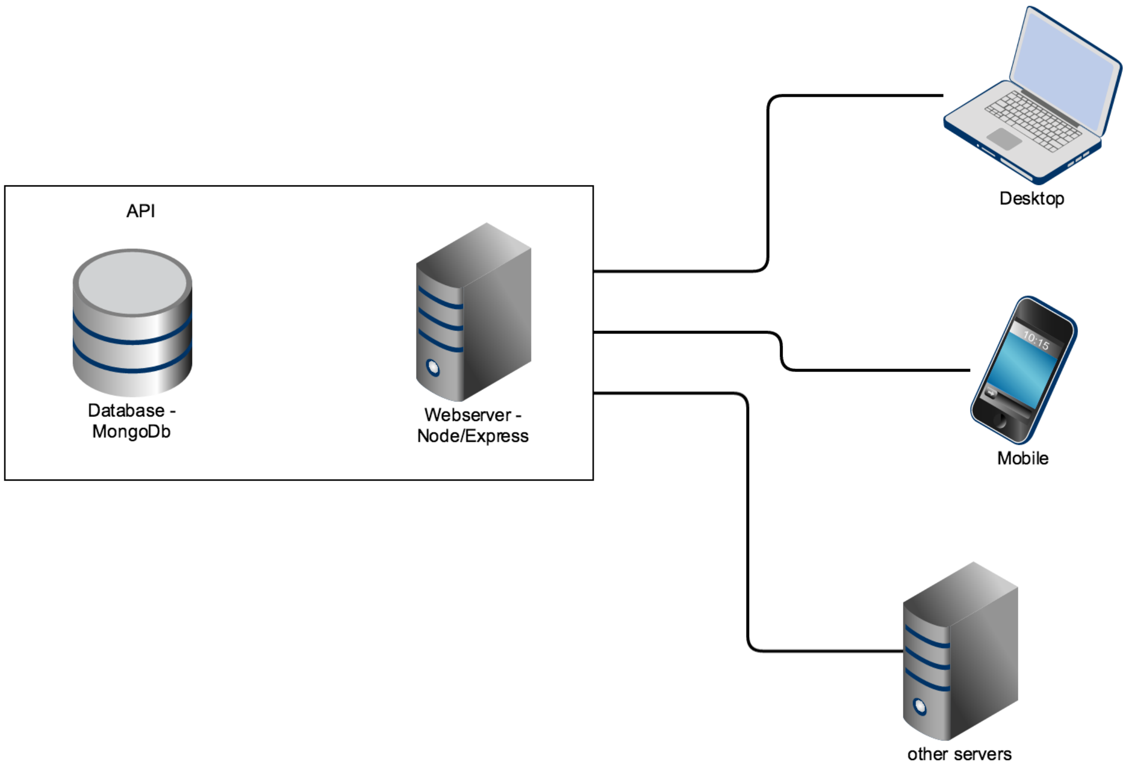
4.2 Other ExpressJS Middleware

The following middleware are not added by default, but it's nice to know they exist at least:

- compression: compress all request. e.g. `app.use(compression())`
- session: create sessions. e.g. `app.use(session({secret: Secr3t}))`
- method-override: `app.use(methodOverride(_method))` Override methods to the one specified on the `_method` param. e.g. `GET /resource/1?_method=DELETE` will become `DELETE /resource/1`.
- response-time: `app.use(responseTime())` adds `X-Response-Time` header to responses.
- errorhandler: Aid development, by sending full error stack traces to the client when an error occurs. `app.use(errorhandler())`. It is good practice to surround it with an if statement to check `process.env.NODE_ENV === development`.
- vhost: Allows you to use different stack of middleware depending on the request `hostname`. e.g.
  `app.use(vhost(*.user.local, userapp))` and
  `app.use(vhost(assets-*.example.com, staticapp))`
  where `userapp` and `staticapp` are different express instances with different middleware layers.
- csurf: Adds a **C**ross-**s**ite **r**equest **f**orgery (CSRF) protection by adding a token to responds either via `session` or `cookie-parser` middleware. `app.use(csrf());`
- timeout: halt execution if it takes more that a given time. e.g. `app.use(timeout(5s));`. However you need to check by yourself under every request with a middleware that checks `if (!req.timedout) next();`.

5 Wiring up the MEAN Stack

In the following sections we are going to put together everything that we learnt above and build an API. The API can be consumed by browsers, mobile apps, and even other servers.



5.1 Bootstrapping ExpressJS

After a detour in the land of Node, MongoDB, Mongoose, and middleware, we are back to our express `todoApp` application. This time we will create the routes and finalise our RESTful API.

Express has a separate package called `express-generator`, which can help us to get started with out API.

```
# install it globally using -g
npm install express-generator -g

# create todo-app API with EJS views (instead the default Jade)
express todo-api -e

#   create : todo-api
#   create : todo-api/package.json
#   create : todo-api/app.js
#   create : todo-api/public
#   create : todo-api/public/javascripts
#   create : todo-api/routes
#   create : todo-api/routes/index.js
#   create : todo-api/routes/users.js
#   create : todo-api/public/stylesheets
#   create : todo-api/public/stylesheets/style.css
#   create : todo-api/views
#   create : todo-api/views/index.ejs
#   create : todo-api/views/layout.ejs
#   create : todo-api/views/error.ejs
#   create : todo-api/public/images
#   create : todo-api/bin
#   create : todo-api/bin/www
#
#   install dependencies:
#     $ cd todo-api && npm install
#
#   run the app:
#     $ DEBUG=todo-app:* npm start
```

This will create a new folder called `todo-api`. Let's go ahead and install the dependencies and run the app:

```
# install dependencies
cd todo-api && npm install
# run the app
npm start
```

Use your browser to go to `http://0.0.0.0:3000`, and you should see a message

```
  Welcome to Express.
```

5.2 Connect ExpressJS to MongoDB

In this section we are going to access MongoDB using our newly created express app. Hopefully, you have installed MongoDB in the setup section, and you can start it by typing (if you haven't yet):

```
mongod
```

Install the MongoDB driver for NodeJS called mongoose:

```
npm install mongoose --save
```

Notice `--save`. It will add it to the `todo-api/package.json`

Next, you need to require mongoose in the `todo-api/app.js`

```
// load mongoose package
var mongoose = require('mongoose');
// Use native Node promises
```

```
mongoose.Promise = global.Promise;
// connect to MongoDB
mongoose.connect('mongodb://localhost/todo-api')
  .then(() =>  console.log('connection succesful'))
  .catch((err) => console.error(err));
```

Now, when you run `npm start` or `./bin/www`, you will (hopefully) see the message `connection successful`.

5.3 Creating the Todo model with Mongoose

Create a `models` directory and a `Todo.js` model:

```
mkdir models
touch models/Todo.js
```

In the `models/Todo.js`:

```
var mongoose = require('mongoose');
var TodoSchema = new mongoose.Schema({
  name: String,
  completed: Boolean,
  note: String,
  updated_at: { type: Date, default: Date.now },});

module.exports = mongoose.model('Todo', TodoSchema);
```

You may well ask what this is about as MongoDB is supposed to be a schemaless database system? Well, it is schemaless and flexible however, very often we want validation and constraints in our API and therefore enforcing a schema keeps the structure consistent; Mongoose does that for us. You can use the following types:

- `String`
- `Boolean`
- `Date`
- `Array`
- `Number`
- `ObjectId`
- `Mixed`
- `Buffer`

6 API clients (Browser, Postman and curl)

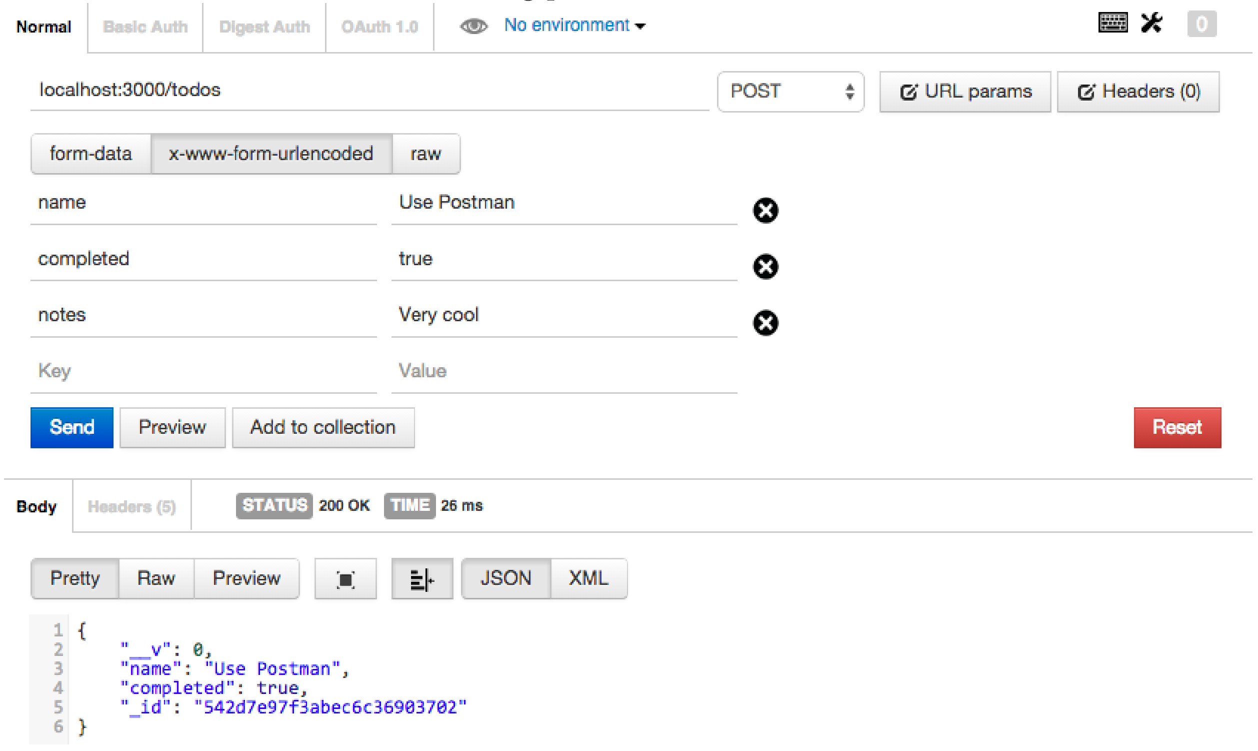
A slight aside now. Here are some tools which may prove useful in allowing easy retrieve, change and delete data operations from your future API.

6.1 Curl

```
# Create task
curl -XPOST http://localhost:3000/todos -d 'name=Master%20Routes&completed=false&note=soon...'
# List tasks
curl -XGET http://localhost:3000/todos
```

6.2 Browser and Postman

If you open your browser and type `localhost:3000/todos` you will see all the tasks (when you implement it). However, you cannot do post commands by default. For further testing let's use a Chrome plugin called Postman (there is also a standalone app for those of you who prefer another browser option). It allows you to use all the HTTP VERBS easily and check `x-www-form-urlencoded` for adding parameters.



localhost:3000/todos POST

name	Use Postman
completed	true
notes	Very cool

STATUS 200 OK TIME 26 ms

```
{
    "__v": 0,
    "name": "Use Postman",
    "completed": true,
    "_id": "542d7e97f3abec6c36903702"
}
```

Don't forget to "tick" `x-www-form-urlencoded` or it won't work!

6.3 Websites and Mobile Apps

Probably these are the main consumers of APIs. You can interact with RESTful APIs using jQuery's `$ajax` and its wrappers, BackboneJS's Collections/models, AngularJS's `$http` or

`$resource`, among many other libraries/frameworks and mobile clients.

7 ExpressJS Routes

We want to achieve the following:

Resource (URI)	POST (create)	GET (read)	PUT (update)	DELETE (destroy)
/todos	create new task	list tasks	error	error
/todos/:id	error	show task :id	update task :id	destroy task ID 1

Let's setup the routes. We'll create a new route called `todos.js` in the `routes` folder or rename `users.js`.

```
mv routes/users.js routes/todos.js
```

In `app.js` add new `todos` route, or just replace `./routes/users` for `./routes/todos`

```
var todos = require('./routes/todos');
app.use('/todos', todos);
```

Edit the `routes/todos.js` file.

7.1 List: GET /todos

Remember the mongoose query api? Here's how to use it in this context:

```
var express = require('express');
var router = express.Router();
var mongoose = require('mongoose');
var Todo = require('../models/Todo.js');

/* GET /todos listing. */
router.get('/', function(req, res, next) {
  Todo.find(function (err, todos) {
    if (err) return next(err);
    res.json(todos);
  });
});

module.exports = router;
```

We don't have any tasks in the database at the moment but at least we verify it is working:

```
# Start database
mongod

# Start Webserver (in another terminal tab)
npm start
```

```
# Test API (in another terminal tab)
curl localhost:3000/todos
# => []%
```

If it returns an empty array `[]` you are all set. If you get errors, try going back and making sure you didn't forget anything.

7.2 Create: POST /todos

Back in `routes/todos.js`, we are going to add the ability to create using mongoose create. Can you make it work before looking at the next example?

```
/* POST /todos */
router.post('/', function(req, res, next) {
  Todo.create(req.body, function (err, post) {
    if (err) return next(err);
    res.json(post);
  });
});
```

A few things are worth mentioning here:

- We are using the `router.post` instead of `router.get`.
- You have to stop and run the server again: `npm start`.

Every time you change a file you have to stop and start the web server. Let's fix that using `nodemon` to refresh automatically:

```
# install nodemon globally
npm install nodemon -g

# Run web server with nodemon
nodemon
```

7.3 Show: GET /todos/:id

This is straightforward with `Todo.findById` and `req.params`. Notice that `params` matches the placeholder name we set while defining the route. `:id` in this case.

```
/* GET /todos/id */
router.get('/:id', function(req, res, next) {
  Todo.findById(req.params.id, function (err, post) {
    if (err) return next(err);
    res.json(post);
  });
});
```

Let's test what we have so far!

```
# Start Web Server
nodemon

# Create a todo using the API
curl -XPOST http://localhost:3000/todos -d
  'name=Master%20Routes&completed=false&note=soon...'
# => {"__v":0,"name":"Master Routes","completed":false,"note":"soon...","_id":"..."}%

# Get todo by ID (use the _id from the previous request)
curl -XGET http://localhost:3000/todos/
  57a655997d2241695585ecf8{"_id":"57a655997d2241695585ecf8","name":"Master
  Routes","completed":false,"note":"soon...","__v":0}%

# Get all elements (notice it is an array)
curl -XGET http://localhost:3000/todos[{"_id":"57a655997d2241695585ecf8",
  "name":"Master Routes","completed":false,"note":"soon...","__v":0}]%
```

7.4 Update: PUT /todos/:id

Back in `routes/todos.js`, we are going to update tasks.

```
/* PUT /todos/:id */
router.put('/:id', function(req, res, next) {
  Todo.findByIdAndUpdate(req.params.id, req.body, function (err, post) {
    if (err) return next(err);
    res.json(post);
  });
});
```

Using curl:

```
# Use the ID from the todo, in my case 57a655997d2241695585ecf8
curl -XPUT http://localhost:3000/todos/57a655997d2241695585ecf8 -d "note=hola"
# => {"_id":"57a655997d2241695585ecf8",
  "name":"Master Routes","completed":true,"note":"hola","__v":0}%
```

7.5 Destroy: DELETE /todos/:id

Almost identical to `update`, use `findByIdAndRemove`.

```
/* DELETE /todos/:id */
router.delete('/:id', function(req, res, next) {
  Todo.findByIdAndRemove(req.params.id, req.body, function (err, post) {
    if (err) return next(err);
    res.json(post);
  });
});
```

Acknowledgements

This document is based upon an article by Adrian Mejia `http://adrianmejia.com`.