Koa.JS

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

Koa.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is an open source framework developed and maintained by the creators of Express.js, the most popular node web framework.

Audience

This tutorial has been created for those who have basic knowledge of HTML, JavaScript(ES6) and how the client-servers work. After completing this tutorial, you'll be able to build moderately complex websites and backends for mobile applications.

Prerequisites

You should have basic knowledge of JavaScript(ES6) and HTML. If you are not acquainted with these, we'll suggest you to go through their tutorials first. Some knowledge of how HTTP works will be quite helpful (not necessary) for you to understand this tutorial. Having basic knowledge on MongoDB will help you with the Database chapter.

Copyright & Disclaimer

Table of Contents

1. KOA.JS – OVERVIEW

A web application framework provides you with a simple API to build websites, web apps, and backend. You need not worry about low level protocols, processes, etc.

What is Koa?

Koa provides a minimal interface to build applications. It is a very small framework (600 LoC) which provides the required tools to build apps and is quite flexible. There are numerous modules available on npm for Koa, which can be directly plugged into it. Koa can be thought of as the core of express.js without all the bells and whistles.

Why Koa?

Koa has a small footprint (600 LoC) and is a very thin layer of abstraction over the node to create server side apps. It is completely pluggable and has a huge community. This also allows us to easily extend Koa and use it according to our needs. It is built using the bleeding edge technology (ES6) which gives it an edge over older frameworks such as express.

Pug

Pug (earlier known as Jade) is a terse language for writing HTML templates.

- Produces HTML
- Supports dynamic code
- Supports reusability (DRY)

It is one of the most popular templating language used with Koa.

MongoDB and Mongoose

MongoDB is an open-source, document database designed for ease of development and scaling. We'll use this database to store data.

Mongoose is a client API for node.js which makes it easy to access our database from our Koa application.

2. KOA.JS – ENVIRONMENT

To get started with developing using the Koa framework, you need to have Node and npm (node package manager) installed. If you don't already have these, head over to Node setup to install node on your local system. Confirm that node and npm are installed by running the following commands in your terminal.

```
$ node --version
$ npm --version
```

You should receive an output similar to:

```
v5.0.0
3.5.2
```

Please ensure your node version is above 6.5.0. Now that we have Node and npm set up, let us understand what npm is and how to use it.

Node Package Manager (npm)

npm is the package manager for node. The npm Registry is a public collection of packages of open-source code for Node.js, front-end web apps, mobile apps, robots, routers, and countless other needs of the JavaScript community. npm allows us to access all these packages and install them locally. You can browse through the list of packages available on npm at npmJS.

How to Use npm?

There are two ways to install a package using npm: globally and locally.

Globally: This method is generally used to install development tools and CLI based packages. To install a package globally, use the following command.

```
$ npm install -g <package-name>
```

Locally: This method is generally used to install frameworks and libraries. A locally installed package can be used only within the directory it is installed. To install a package locally, use the same command as above without the **-g** flag.

```
$ npm install <package-name>
```

Whenever we create a project using npm, we need to provide a package.json file, which has all the details about our project. npm makes it easy for us to set up this file. Let us set up our development project.

Step 1: Fire up your terminal/cmd, create a new folder named hello-world and cd into it:

```
ayushgp@dell:~$ mkdir hello-world
ayushgp@dell:~$ cd hello-world/
ayushgp@dell:~/hello-world$
```

Step 2: Now to create the package.json file using npm, use the following.

```
npm init
```

It'll ask you for the following information:

```
Press ^C at any time to quit.
name: (hello-world)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author: Ayush Gupta
license: (ISC)
About to write to /home/ayushgp/hello-world/package.json:

{
  "name": "hello-world",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Ayush Gupta",
  "license": "ISC"
}


Is this ok? (yes) yes
ayushgp@dell:~/hello-world$
```

Just keep pressing enter, and enter your name in the "author name" field.

Step 3: Now we have our package.json file set up, we'll install Koa. To install Koa and add it in our package.json file, use the following command.

```
$ npm install --save koa
```

To confirm Koa installed correctly, run the following command.

```
$ ls node_modules #(dir node_modules for windows)
```

Tip: The **--save** flag can be replaced by **-S** flag. This flag ensures that Koa is added as a dependency to our package.json file. This has an advantage, the next time we need to install all the dependencies of our project, we just need to run the command npm install and it'll find the dependencies in this file and install them for us.

This is all we need to start development using the Koa framework. To make our development process a lot easier, we will install a tool from npm, nodemon. What this tool does is, it restarts our server as soon as we make a change in any of our files, otherwise we need to

restart the server manually after each file modification. To install nodemon, use the following command.

```
$ npm install -g nodemon
```

Now we are all ready to dive into Koa!

3. KOA.JS – HELLO WORLD

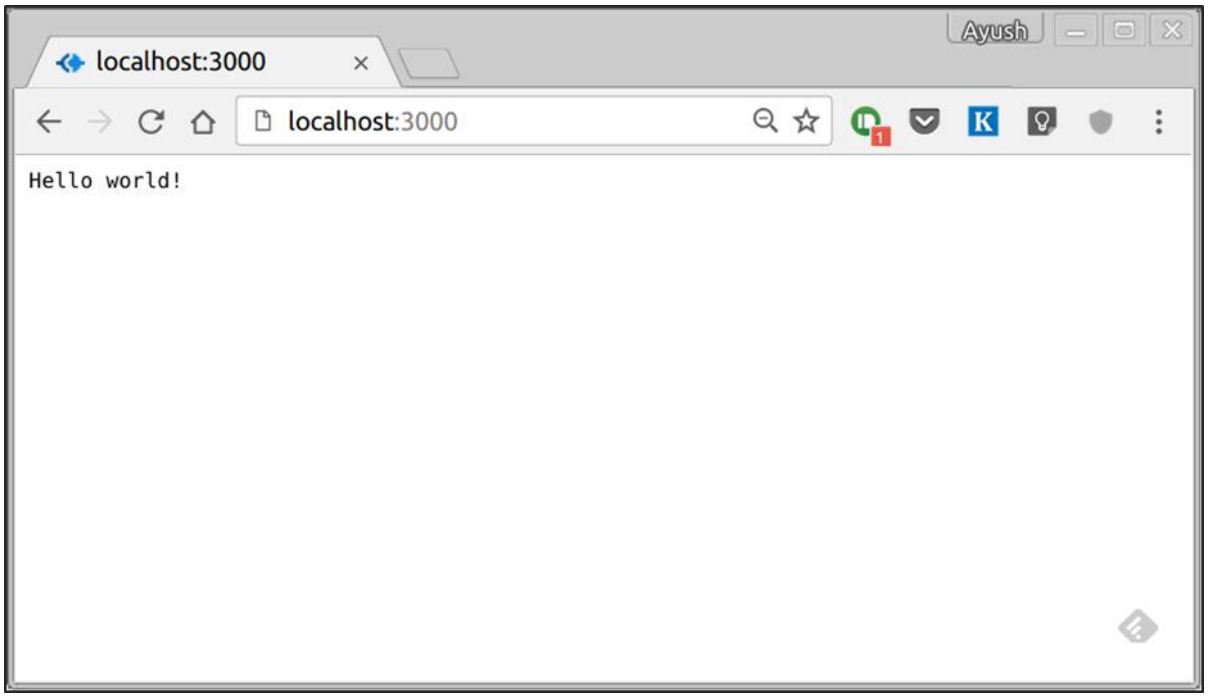
Once we have set up the development, it is time to start developing our first app using Koa. Create a new file called **app.js** and type the following in it.

```
var koa = require('koa');
var app = koa();

app.use(function* (){
    this.body = 'Hello world!';
});

app.listen(3000, function(){
    console.log('Server running on https://localhost:3000')
});
```

Save the file, go to your terminal and type.

```
$ nodemon app.js
```

This will start the server. To test this app, open your browser and go to https://localhost:3000 and you should receive the following message.



How This App Works?

The first line imports Koa in our file. We have access to its API through the variable Koa. We use it to create an application and assign it to var app.

app.use(function) - This function is a middleware, which gets called whenever our server gets a request. We'll learn more about middleware in the subsequent chapters. The callback function is a generator, which we'll see in the next chapter. The context of this generator is called context in Koa. This context is used to access and modify the request and response objects. We are setting the body of this response to be **Hello world!**.

app.listen(port, function) - This function binds and listens for connections on the specified port. Port is the only required parameter here. The callback function is executed, if the app runs successfully.

4. KOA.JS – GENERATORS

One of the most exciting new features of JavaScript ES6 is a new breed of function, called a generator. Before generators, the whole script was used to usually execute in a top to bottom order, without an easy way to stop code execution and resuming with the same stack later. Generators are functions which can be exited and later re-entered. Their context (variable bindings) will be saved across re-entrances.

Generators allow us to stop code execution in between. Hence, let's take a look at a simple generator.

```
var generator_func = function* (){
     yield 1;
     yield 2;
};

var itr = generator_func();
console.log(itr.next());
console.log(itr.next());
console.log(itr.next());
```

When running the above code, following will be the result.

```
{ value: 1, done: false }
{ value: 2, done: false }
{ value: undefined, done: true }
```

Let's look inside the above code. We first create a generator called **generator_func()**. We created an instance of this weird looking function and assigned it to **itr**. Then we started calling **next()** on this itr variable.

Calling next() starts the generator and it runs until it hits a yield. Then it returns the object with value and done, where the value has the expression value. This expression can be anything. At this point, it pauses execution. Again when we call this function(next), the generator resumes execution from the last yield point with the function state being the same at the time of pause, till the next yield point. This is done till there are no more yield points in the code.

Generators in Koa

So why are we discussing generators in this tutorial. As you might remember from the hello world program, we used a **function* ()** notation to pass a callback to app.use(). Koa is an

object, which contains an array of middleware generator functions, all of which are composed and executed in a stack-like manner upon each request. Koa also implements downstreaming followed by upstreaming of control flow.

Take a look at the following example to understand this in a better way.

```
var koa = require('koa');
var app = koa();

app.use(function* (next) {
    //do something before yielding to next generator function
    //in line which will be 1st event in downstream

    console.log("1");
    yield next;

    // do something when the execution returns upstream,
    //this will be last event in upstream
    console.log("2");
});

app.use(function* (next) {
    // This shall be 2nd event downstream

    console.log("3");
    yield next;

    // This would be 2nd event upstream
    console.log("4");
});

app.use(function* () {
     // Here it would be last function downstream
    console.log("5");

    // Set response body
    this.body = "Hello Generators";
```

```
    // First event of upstream (from the last to first)
    console.log("6");
});
app.listen(3000);
```

When running the above code and navigating to https://localhost:3000/ we get the following output on our console.

```
1
3
5
6
4
2
```

This is essentially how Koa uses generators. It allows us to create compact middleware using this property and write code for both upstream and downstream functionalities, thus saving us from callbacks.

5. KOA.JS – ROUTING

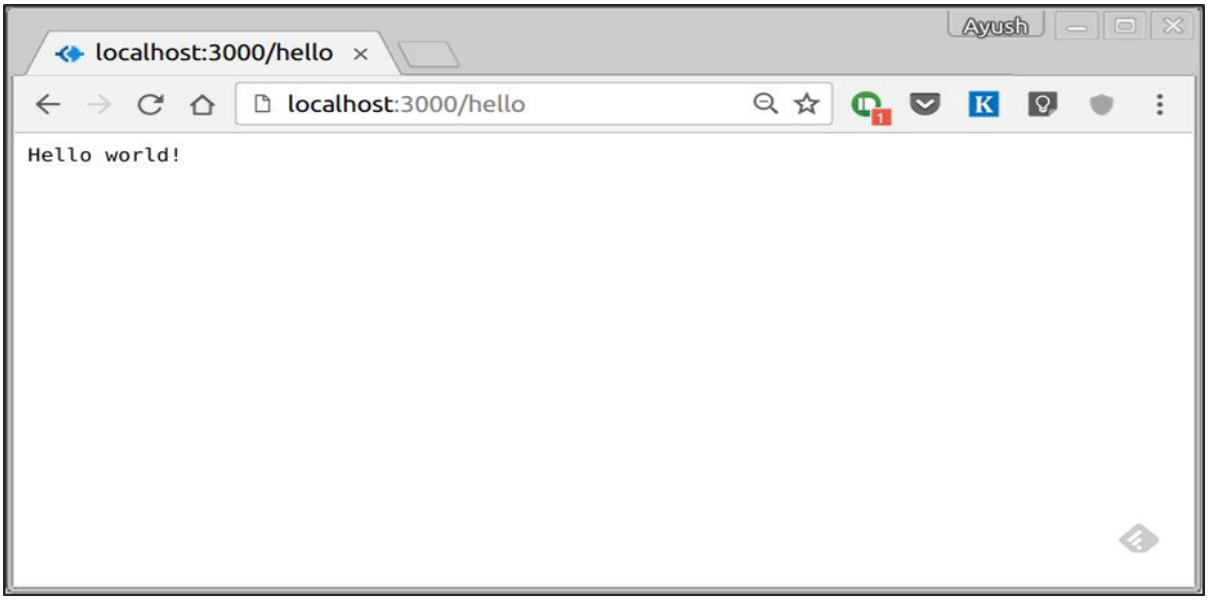
Web frameworks provide resources such as HTML pages, scripts, images, etc. at different routes. Koa does not support routes in the core module. We need to use the Koa-router module to easily create routes in Koa. Install this module using the following command.

```
npm install --save koa-router
```

Now that we have Koa-router installed, let's look at a simple GET route example.

```
var koa = require('koa');
var router = require('koa-router');
var app = koa();

var _ = router();              // Instantiate the router
_.get('/hello', getMessage);   // Define routes

function *getMessage(){
     this.body = "Hello world!";
};

app.use(_.routes());           // Use the routes defined using the router
app.listen(3000);
```

If we run our application and go to localhost:3000/hello, the server receives a get request at route "/hello". Our Koa app executes the callback function attached to this route and sends "Hello World!" as the response.



We can also have multiple different methods at the same route. For example,

```
var koa = require('koa');
var router = require('koa-router');
var app = koa();

var _ = router();        // Instantiate the router

_.get('/hello', getMessage);
_.post('/hello', postMessage);

function *getMessage(){
     this.body = "Hello world!";
};

function *postMessage(){
     this.body = "You just called the post method at '/hello'!\n";
};

app.use(_.routes());     // Use the routes defined using the router
app.listen(3000);
```

To test this request, open your terminal and use cURL to execute the following request

```
curl -X POST "https://localhost:3000/hello"
```

```
ayushgp@swaggy:~/hello-world$ curl -X POST "http://localhost:3000/hello"
You just called the post method at '/hello'!
ayushgp@swaggy:~/hello-world$
```

A special method, **all**, is provided by express to handle all types of http methods at a particular route using the same function. To use this method, try the following:

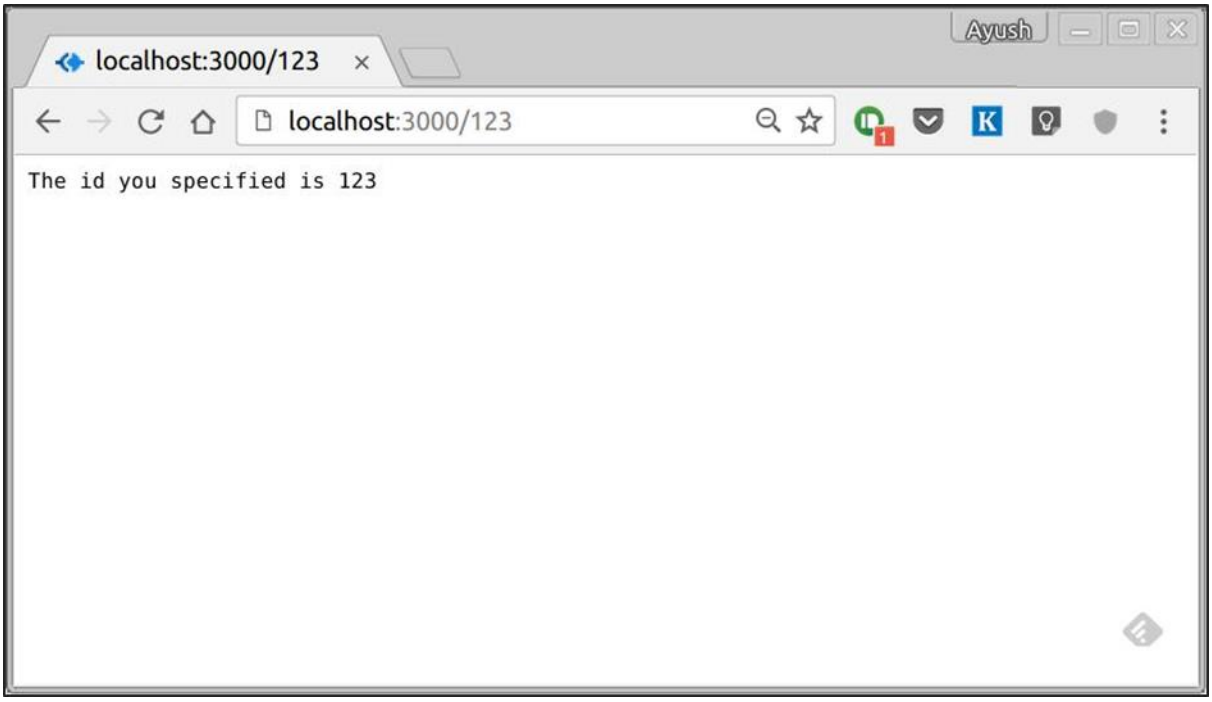
```
_.all('/test', allMessage);


function *allMessage(){
     this.body = "All HTTP calls regardless of the verb will get this response";
};
```

6. KOA.JS – URL BUILDING

We can now define routes; they are either static or fixed. To use dynamic routes, we need to provide different types of routes. Using dynamic routes allow us to pass parameters and process based on them. Following is an example of a dynamic route.

```
var koa = require('koa');
var router = require('koa-router');
var app = koa();

var _ = router();

_.get('/:id', sendID);

function *sendID(){
    this.body = 'The id you specified is ' + this.params.id;
}

app.use(_.routes());

app.listen(3000);
```

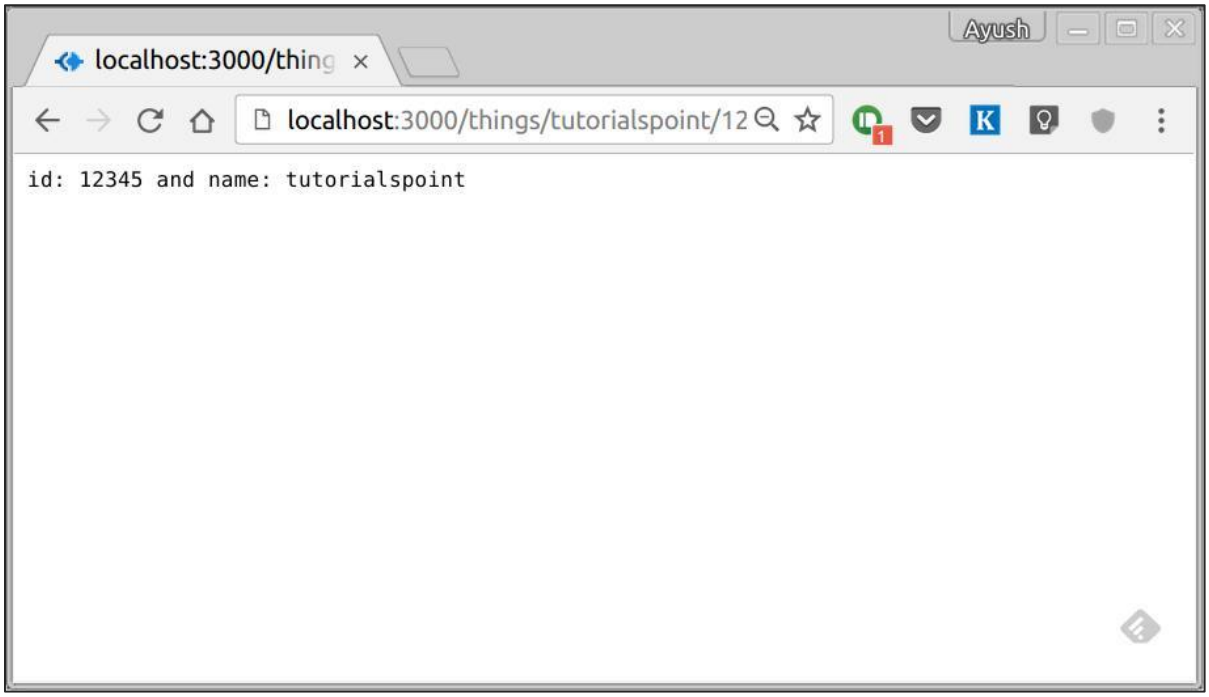
To test this go to https://localhost:3000/123. You will get the following response.



You can replace '123' in the URL with anything else and it'll be reflected in the response. Following is a complex example of the above.

```
var koa = require('koa');
var router = require('koa-router');
var app = koa();

var _ = router();

_.get('/things/:name/:id', sendIdAndName);

function *sendIdAndName(){
    this.body = 'id: ' + this.params.id + ' and name: ' + this.params.name;
};

app.use(_.routes());

app.listen(3000);
```

To test this, go to https://localhost:3000/things/tutorialspoint/12345



You can use the ***this.params*** object to access all the parameters you pass in the URL. Note that the above two have different paths. They will never overlap. Also if you want to execute the code when you get '/things', then you need to define it separately.

Pattern Matched Routes

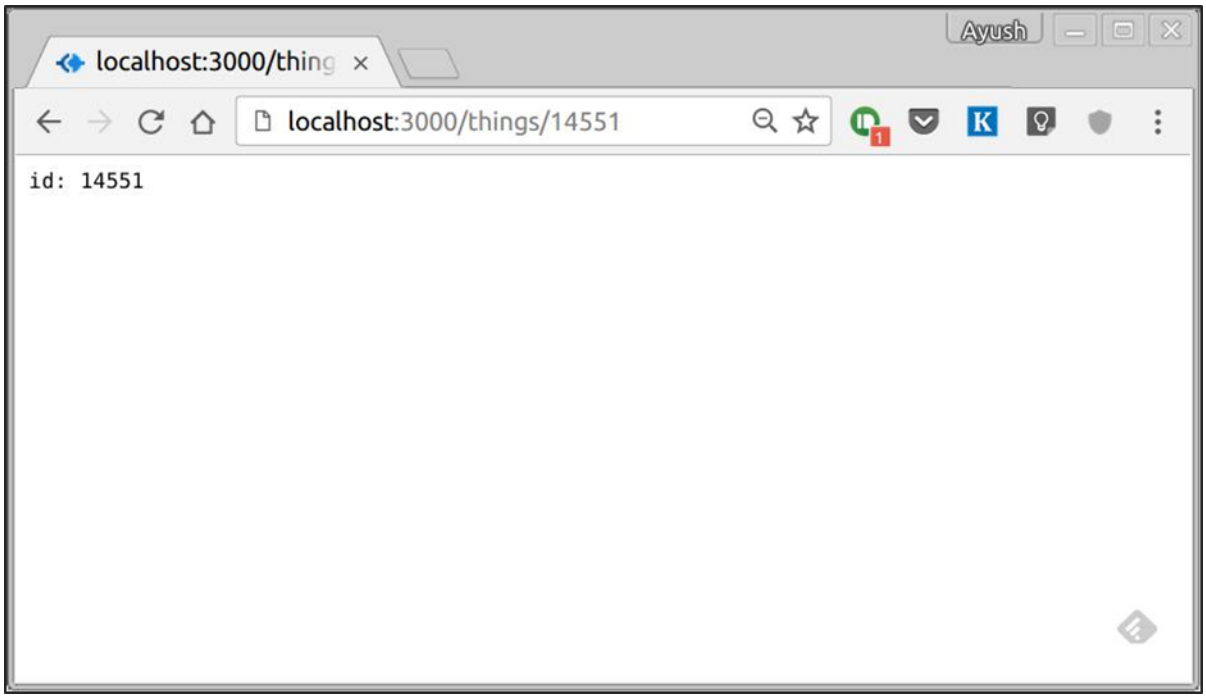
You can also use regex to restrict URL parameter matching. Let's say you need the id to be five digits long number. You can use the following route definition.

```
var koa = require('koa');
var router = require('koa-router');
var app = koa();

var _ = router();

_.get('/things/:id([0-9]{5})', sendID);

function *sendID(){
     this.body = 'id: ' + this.params.id;
}

app.use(_.routes());
```

```
app.listen(3000);
```

Note that this will **only** match the requests that have a 5-digit long id. You can use more complex regexes to match/validate your routes. If none of your routes match the request, you'll get a Not found message as response.

For example, if we define the same routes as above, on requesting with a valid URL, we get:



7. KOA.JS – HTTP METHODS

The HTTP method is supplied in the request and specifies the operation that the client has requested. The following table summarizes the commonly used HTTP methods.

Method	Description
GET	The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
POST	The POST method requests that the server accept the data enclosed in the request as a new object/entity of the resource identified by the URI.
PUT	The PUT method requests that the server accept the data enclosed in the request as a modification to the existing object identified by the URI. If it does not exist, then PUT method should create one.
DELETE	The DELETE method requests that the server delete the specified resource.

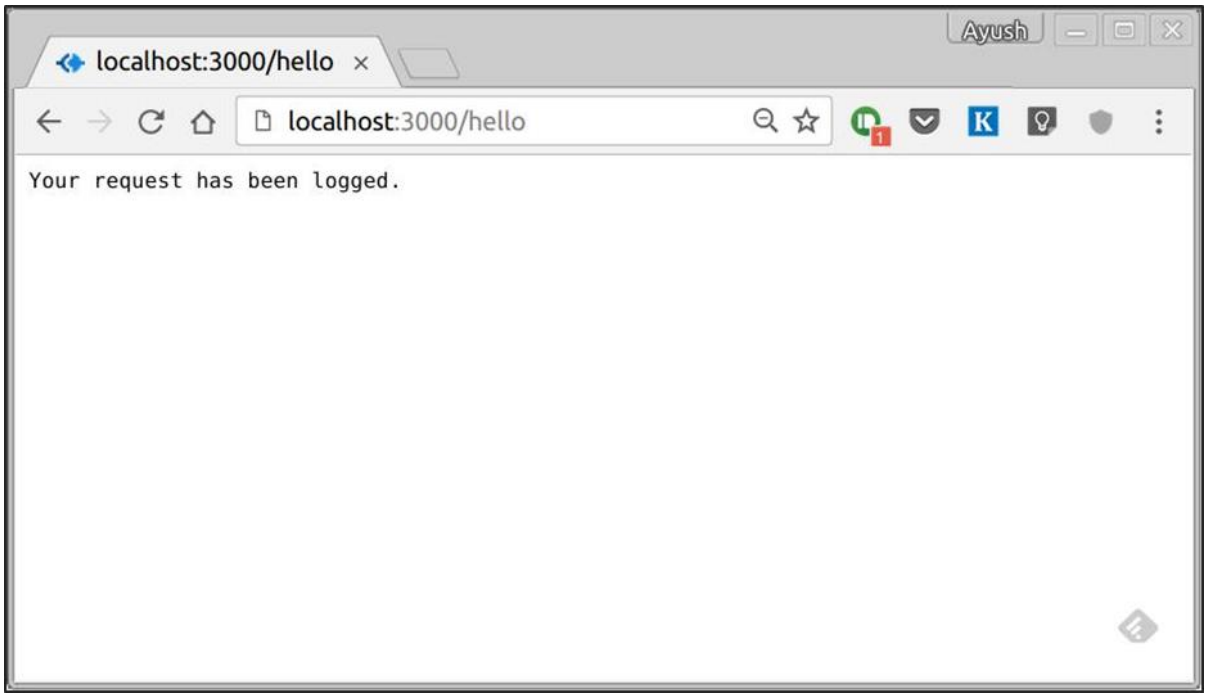
These are the most common HTTP methods. To learn more about them, head over to https://www.tutorialspoint.com/http/http_methods.htm.

8. KOA.JS – REQUEST OBJECT

A Koa Request object is an abstraction on top of node's vanilla request object, providing additional functionality that is useful for everyday HTTP server development. The Koa request object is embedded in the context object, **this**. Let's log out the request object whenever we get a request.

```
var koa = require('koa');
var router = require('koa-router');
var app = koa();

var _ = router();
_.get('/hello', getMessage);

function *getMessage(){
    console.log(this.request);
    this.body = 'Your request has been logged.';
}

app.use(_.routes());
app.listen(3000);
```

When you run this code and navigate to https://localhost:3000/hello, then you will receive the following response.



On your console, you'll get the request object logged out.

```
{
    method: 'GET',
    url: '/hello/',
    header:
    {
        host: 'localhost:3000',
        connection: 'keep-alive',
        'upgrade-insecure-requests': '1',
        'user-agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/52.0.2743.116 Safari/537.36',
        accept:
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
        dnt: '1',
        'accept-encoding': 'gzip, deflate, sdch',
        'accept-language': 'en-US,en;q=0.8'
    }
}
```

We have access to many useful properties of the request using this object. Let us look at some examples.

request.header

Provides all the request headers.

request.method

Provides the request method(GET, POST, etc.)

request.href

Provides the full request URL.

request.path

Provides the path of the request. Without query string and base url.

request.query

Gives the parsed query string. For example, if we log this on a request such as https://localhost:3000/hello/?name=Ayush&age=20&country=India, then we'll get the following object.

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**