Getting “MEAN”

- A Practical Workshop

Express JS Framework

Express Overview:

Express is a **minimal** and **flexible Node.js web application framework** that provides a robust set of features for web and mobile applications.

It facilitates the rapid development of Node-based Web applications.

Following are some of the core features of Express framework:

- Allows to set up middlewares to respond to HTTP Requests.

- Defines a routing table which is used to perform different actions based on HTTP Method and URL.

-Allows to dynamically render HTML Pages based on passing arguments to templates.

Installing Express:

Assuming you have already installed Node.js, create a directory to hold your application, make that your working directory.

```
$ mkdir myApp
$ cd myApp
```

Use the npm init command to create a package.json file for your application.

For more information on how package.json works , see
(https://docs.npmjs.com/files/package.json)

```
$ npm init
```

This command prompts you for a number of things, such as the name and version of your application. For now, you can hit ENTER key to accept the defaults for most of them, with the following **exception**.

```
entry point: (index.js)
```
Enter app.js, or whatever you want the name of the **main file** to be.

If you want it to be index.js, hit RETURN(Enter key) to accept the suggested default file name.

Now install Express in the myApp directory and save it in the dependencies list.

For example:
$ npm install express --save

To install Express temporarily and not add it to the dependencies list, omit the **--save** option:

$ npm install express

Note : Node modules installed with the **--save** option are added to the dependencies list in the **package.json** file. Afterwards, running **npm install** in the app directory will automatically install modules in the dependencies list.

Express Application generator :

Use the application generator tool, **express-generator** , to quickly create an application skeleton.

Install express-generator with the following command.

$npm install express-generator -g

After express-generator installation, you can go ahead to install express based app.

$ express myApp

```
  create : myApp
  create : myApp/package.json
  create : myApp/app.js
  create : myApp/public
  create : myApp/routes
  create : myApp/routes/index.js
  create : myApp/routes/users.js
  create : myApp/views
  create : myApp/views/index.jade
```

```
   create : myApp/views/layout.jade
   create : myApp/views/error.jade
   create : myApp/bin
   create : myApp/bin/www
   create : myApp/public/javascripts
   create : myApp/public/images
   create : myApp/public/stylesheets
   create : myApp/public/stylesheets/style.css

   install dependencies:
     $ cd myApp && npm install

   run the app:
     $ DEBUG=myApp:* npm start
```

Then install dependencies :

```
$ cd myApp
$ npm install
```

Then to run the application, use the below command :

```
$npm start
```

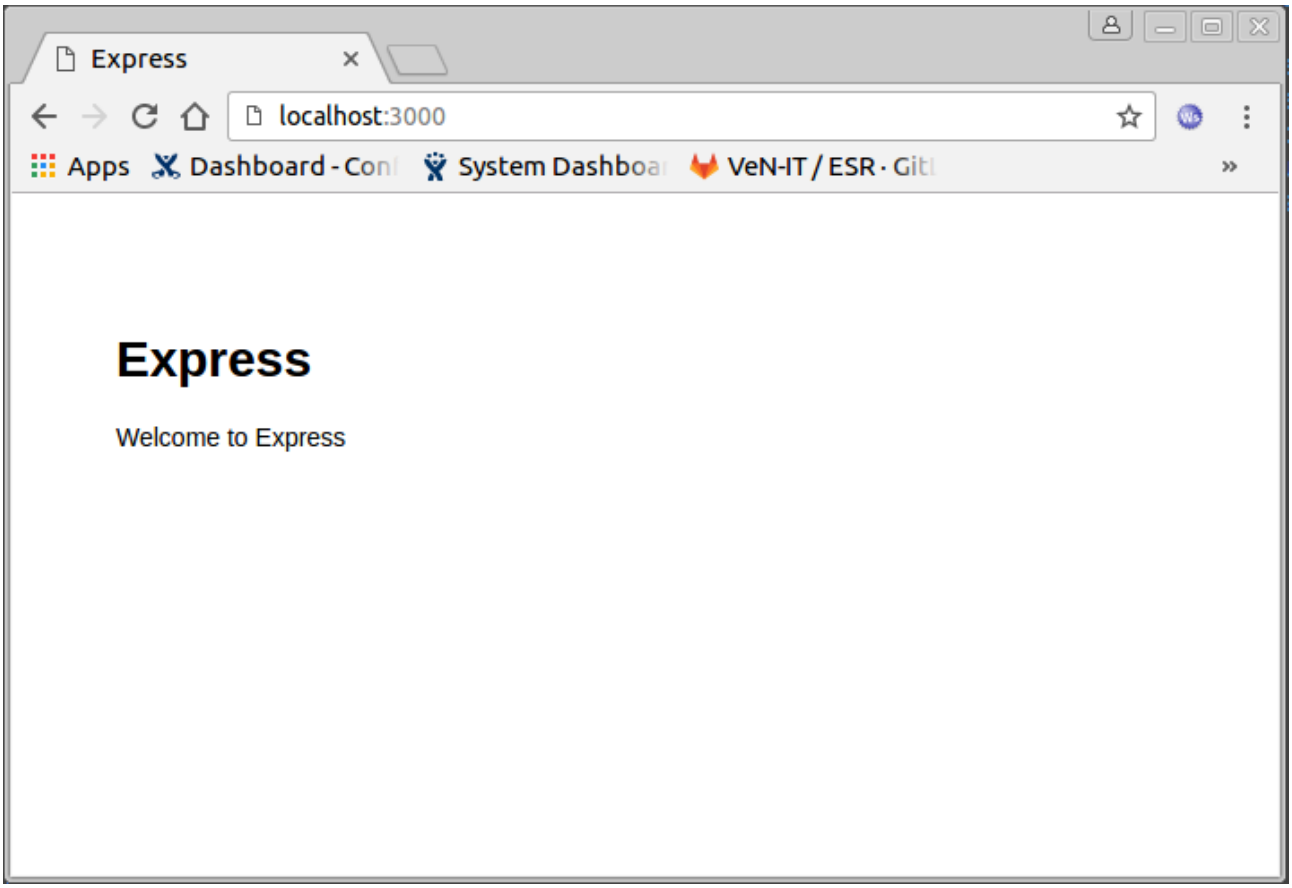
Verify the output :

```
sharif@world:~/mean/express/myApp$ npm start

> myApp@0.0.0 start /home/sharif/mean/express/myApp
> node ./bin/www
```

Open http://localhost:3000 any browser to check the application is running :

Verify the output at server side :

```
> myApp@0.0.0 start /home/sharif/mean/express/myApp
> node ./bin/www

GET / 304 308.310 ms - -
GET /stylesheets/style.css 200 8.828 ms - 111
```



The generated app has the following directory structure :

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.jade
    ├── index.jade
    └── layout.jade

7 directories, 9 files
```

Basic Routing

Basic Routing :

- Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

- Each route can have one or more handler functions, which are executed when the route is matched.

- Route definition takes the following structure:

app.METHOD(PATH, HANDLER)

where :

app – is an instance of express

METHOD – is an HTTP request Method, (in lowercase)

PATH – is a path on the server

HANDLER – is the function executed when the route is matched.

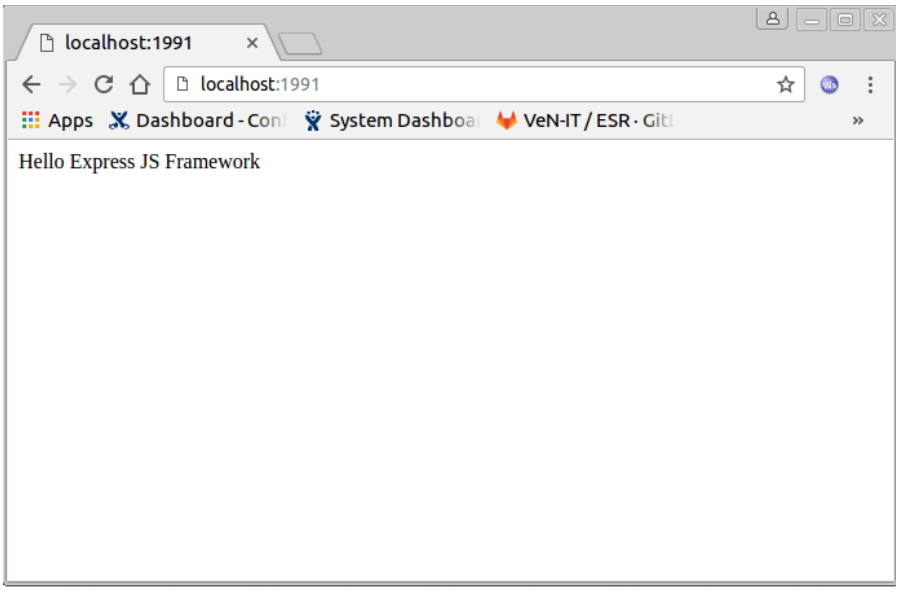
Example 1 : Hello Express Js Framework
You can download the source code (https://github.com/virtualSharif/gettingMEAN/blob/master/express/examples/example1.js) or create example1.js file which have contents as the following :

```
1 var express = require('express');
2 var app = express();
3
4 app.get('/', function (req, res) {
5   res.send('Hello Express JS Framework');
6 });
7
8 app.listen(1991, function () {
9   console.log('Example app listening on port 1991!');
10 });
```

To run the application : $ node example1.js

Verify the output :
Example app listening on port 1991!

Then , load http://localhost:1991/ in any browser to see the output:



Example 2 : Multiple HTTP methods at same URL
You can download the source code
(https://github.com/virtualSharif/gettingMEAN/blob/master/express/examples/example2.js)

or create **example2.js** file which have contents as the following :

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

//Respond with Hello World! on the homepage:
app.get('/', function (req, res) {
  res.send('Hello Express JS Framework!');
});

//Respond to GET request on the /user route
app.get('/user', function (req, res) {
  res.send('Got a Get request at /user');
});

//Respond to POST request to the /user route

app.post('/user', function (req, res) {
  res.send('Got a POST request at /user');
});
```

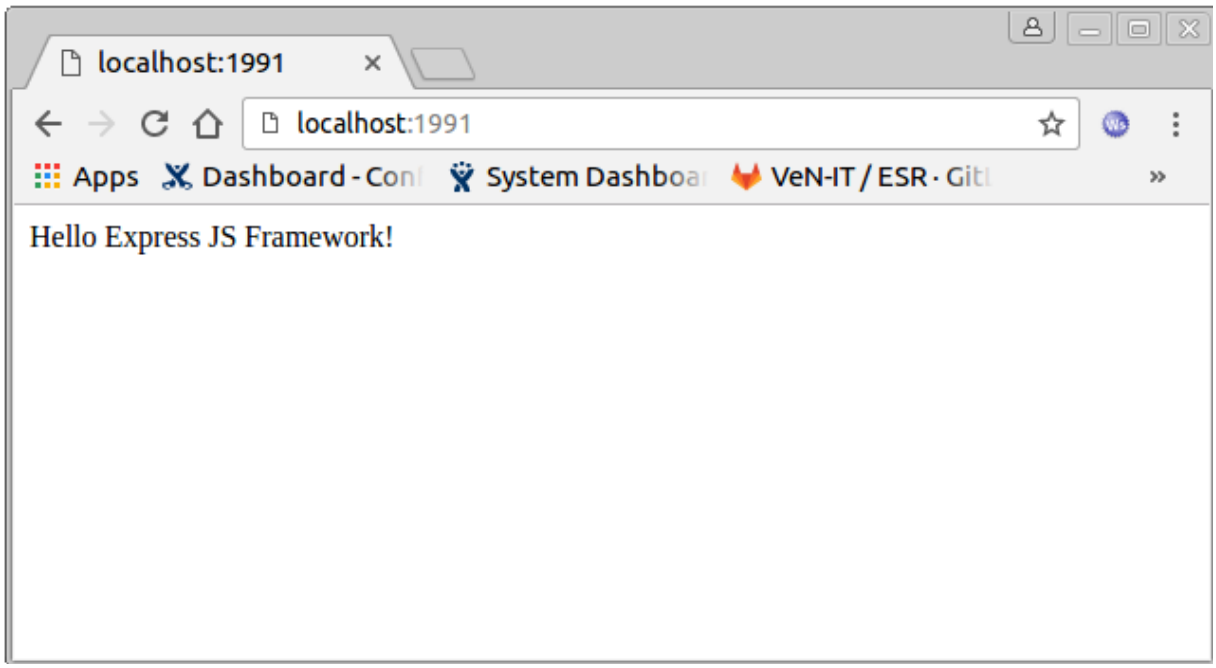
```
19
20 //Respond to a PUT request to the /user route:
21
22 app.put('/user', function (req, res) {
23   res.send('Got a PUT request at /user');
24 });
25
26 //Respond to a DELETE request to the /user route:
27
28 app.delete('/user', function (req, res) {
29   res.send('Got a DELETE request at /user');
30 });
31
32 app.listen(1991, function () {
33   console.log('Example app listening on port 1991!');
34 });
```

To run the application :
$ node example1.js

Verify the output :

Example app listening on port 1991!

Then , load http://localhost:1991/ in any browser to see the output:



But now, to check the other URL you need to have some application which can send different type of HTTP request to our server.

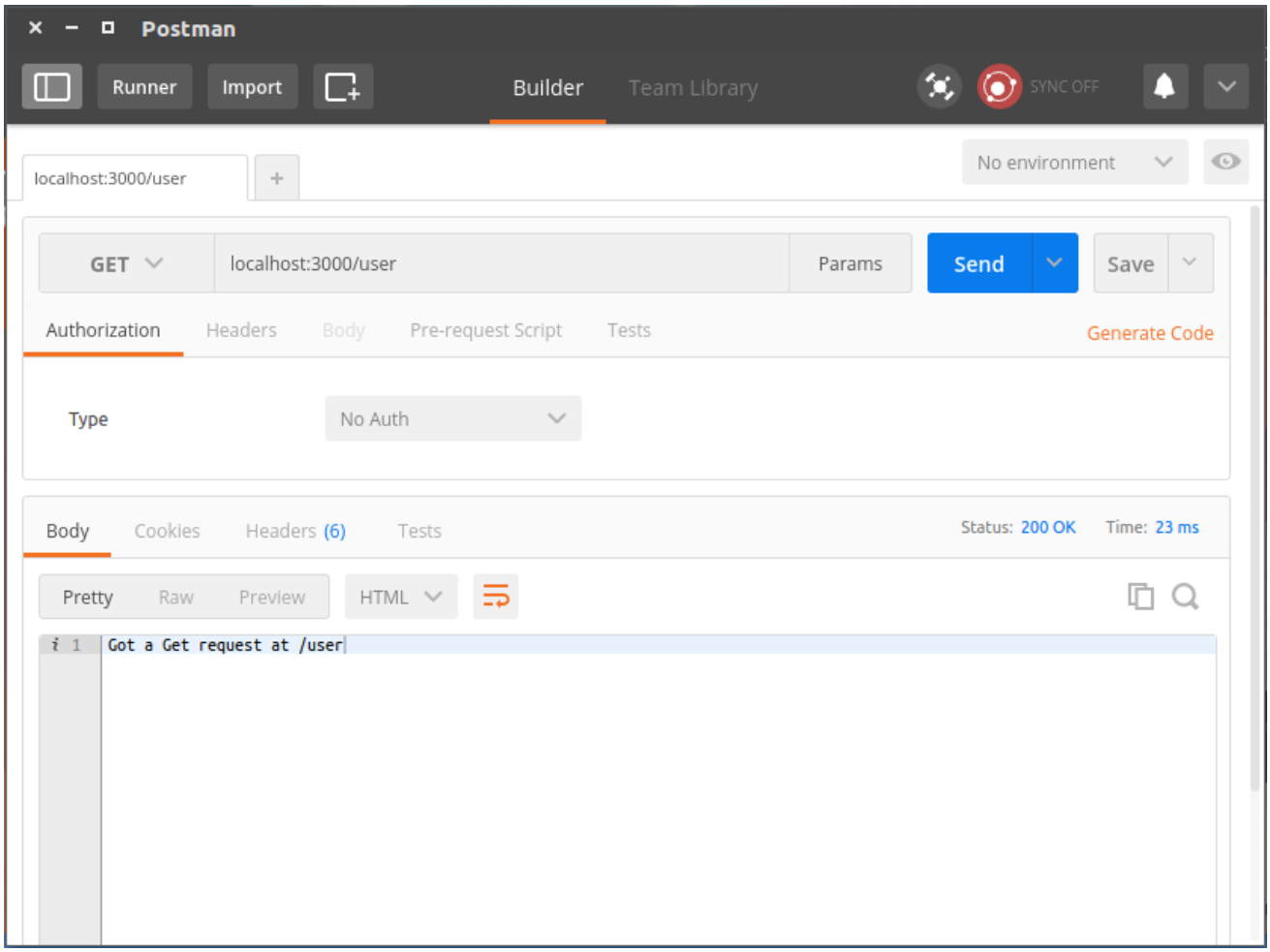
POSTMAN Chrome Extenstion :

To check the different HTTP calls, use postman chrome app.
(https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=en)

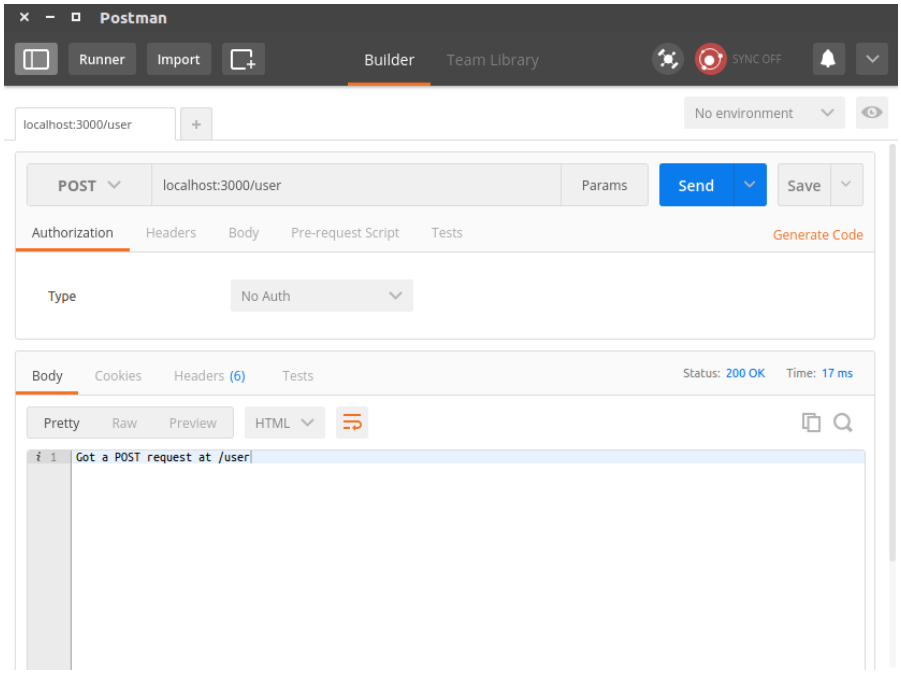
After installation of POSTMAN chrome extension:

You can send different types of HTTP requests to our server.

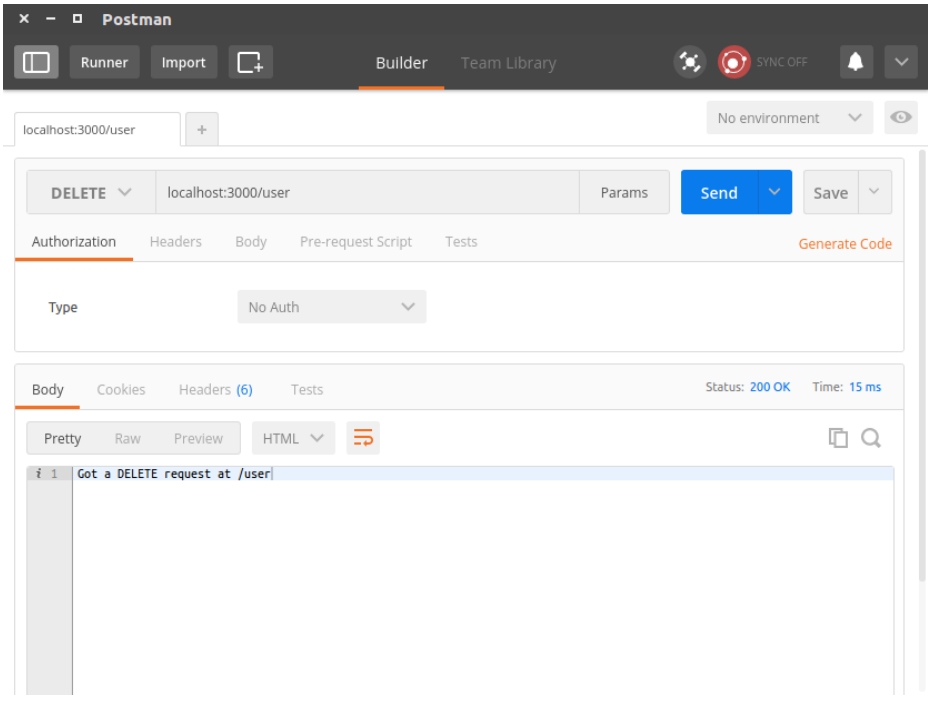
i. HTTP GET Method



ii. HTTP POST Method



iii. HTTP DELETE Method



To know more about the postman chrome extension, please go through the link (https://www.getpostman.com/)

RESTful API

What is REST Architecture?

REST stands for **RE**presentational **S**tate **T**ransfer.

REST is a web standard based architecture that uses HTTP Protocol.

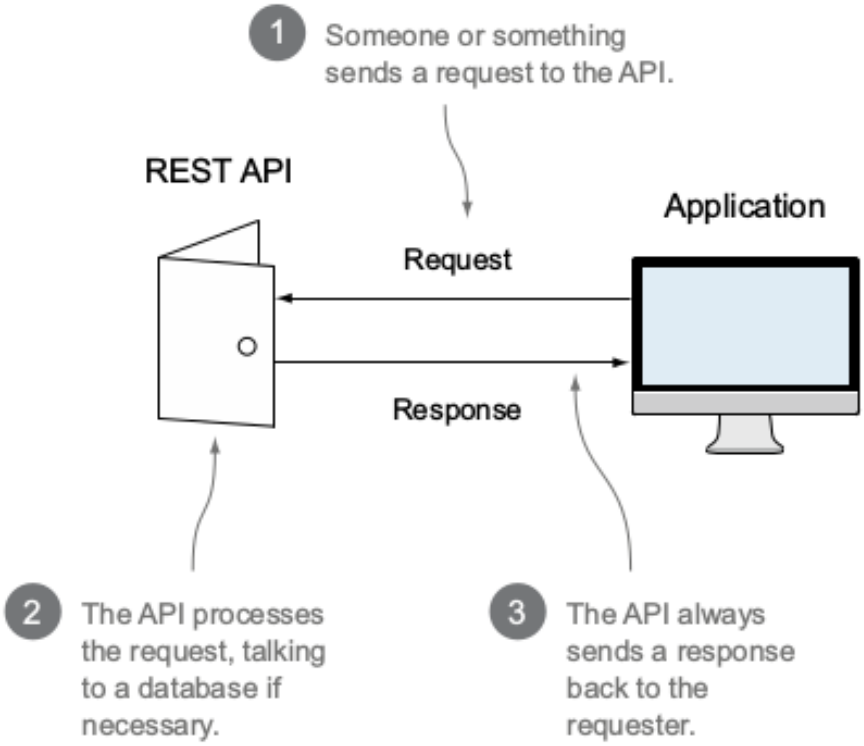
It revolves around resources where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

REST was first introduced by Roy Fielding in 2000.

A REST Server simply provides access to resources and a REST client accesses and modifies the resources using HTTP protocol.

Here each resource is identified by URIs/ global Ids.

REST uses various representation to represent a resource, for example, text, JSON, XML, but JSON is the most popular one.



A REST API takes incoming HTTP requests, does some processing, and returns HTTP responses.

HTTP Request methods :
The following four HTTP methods are commonly used in REST based architecture.

- GET - This is used to provide a read-only access to a resource.
- POST - This is used to create a new resource.
- DELETE - This is used to remove a resource.
- PUT - This is used to update an existing resource.

The rules of REST API :

HTTP requests can have different methods that essentially tell the server what type of action to take.

Four basic Request methods used in a REST API

Request Method	Use	Response
POST	Create new data in DB	New data object as seen in DB
GET	Read data from the DB	Data object as seen in DB
PUT	Update data in DB	Update data object as seen in DB
DELETE	Delete data from the DB	Null

URL paths and parameters for an API to the Users; all have the same base path, and several have the same userId parameter.

Action	URL Path	Para-meters	Example
Create new User	/users		http://localhost:1991/api/users
Return list of Users	/users		http://localhost:1991/api/users
Return a specific user	/users	userId	http://localhost:1991/api/users/123
Update a specific user	/users	userId	http://localhost:1991/api/users/123
Delete a specific user	/users	userId	http://localhost:1991/api/users/123

Conclusion :

Request method is used to link the URL to the desired actions, enabling the API to use the same URL for different actions.

Action	Method	URL Path	Parameters
Create new User	POST	/users	
Return list of Users	GET	/users	
Return a specific user	GET	/users	userId
Update a specific user	PUT	/users	userId
Delete a specific user	DELETE	/users	userId

HTTP Status Code :

Most popular HTTP status code and how they might be used when sending responses to an API request.

Status Code	Name	Use Case
200	OK	A successful GET or PUT request
201	Created	A successful POST request
204	No content	A successful DELETE request
400	Bad request	An unsuccessful GET, POST, or PUT request, due to invalid content
401	Unauthorized	Requesting a restricted URL with incorrect credentials
403	Forbidden	Making a request that isn’t allowed
404	Not Found	Unsuccessful request due to an incorrect parameter in the URL
405	Method not allowed	Request Method not allowed for the given URL
409	Conflict	Unsuccesful POST request when another object already exists with the same data
500	Internal Server Error	Problem with your server or the database server

Example 3 :

Now we will modulize the application.
For example, we will separate routes and controllers and build application.

We will also use **module.exports** to exports our function and make the application modular.

Download the source code
(https://github.com/virtualSharif/gettingMEAN/tree/master/express/examples/example3)

where you can see, two directories controllers and node_modules. And one file named as app.js

i. Directory : controllers : will contain all the controllers

Currently it has two files ;
a. index.js : which will have all controllers registration list
b. user.js : dedicated to have user related APIs

ii. Directory : node_modules : will contain node_modules installed locally

iii. File : app.js : main entry for the server side application