

- * Get request = **Read**
- * Post request = **Create**
- * Put & Patch request = **Update**
 - Sending entire entry to replace previous one**
 - only sending piece of data which needs to be updated**
- * delete = **delete**
- * **HTTP 3T**
- * Rest : **Representational State Transfer** is an architectural style for designing API's

* To Setup Server

- Create new Directory called **Wiki-API**
- Initialise NPM and install **body-parser**, **mongoose**, **ejs** and **express**
- Create a new file called **app.js**
- Inside **app.js** add **server code** (**Write/Copy**)
- Setup **MongoDB**:
 - DB name is **wikiDB**
 - Collection name is **articles**
 - Document has 2 fields: **title** and **content**

```
//jshint esversion:6
const express = require("express");
const bodyParser = require("body-parser");
const ejs = require("ejs");
const mongoose = require('mongoose');

const app = express();

app.set('view engine', 'ejs');

app.use(bodyParser.urlencoded({
  extended: true
}));
app.use(express.static("public"));

mongoose.connect("mongodb://localhost:27017/wikiDB", {useNewUrlParser: true});

const articleSchema = {
  title: String,
  content: String
};

const Article = mongoose.model("Article", articleSchema);

//TODO

app.listen(3000, function() {
  console.log("Server started on port 3000");
});
```

```
<ModelName>.find(conditions, function(err, results){
    //Use the found results docs.
});
```

```
const Article = mongoose.model("Article", articleSchema);

app.get("/articles", function(req, res){
    Article.find(function(err, foundArticles){
        if (!err) {
            res.send(foundArticles); Now we are finding everything so first parameter is not passed
        } else {
            res.send(err);
        }
    });
});
```

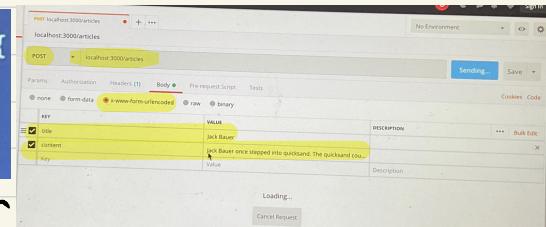
* now when run browser
localhost:3000/articles
output

```
[{"_id": "5c18f35cd40ab6cc551cd60", "title": "Sobek is the crocodile god", "content": "Sobek's name is often used for Representational State Transfer. It's an architectural style for designing APIs."}, {"_id": "5c18f35cd40ab6cc551cd61", "title": "React.js", "content": "React's 'JSX' stands for Application Programming Interface. It is a set of subroutines, definitions, communication protocols, and tools for building software."}, {"_id": "5c18f35cd40ab6cc551cd62", "title": "MongoDB", "content": "MongoDB is a framework developed by Twitter that contains pre-made front-end templates for web design."}, {"_id": "5c18f35cd40ab6cc551cd63", "title": "Jack Bauer", "content": "Jack Bauer once stepped into quicksand. The quicksand could..."}]
```

we created these in Robo 3T under articles

```
app.post("/articles", function(req, res){
    console.log(req.body.title);
    console.log(req.body.content);
});
```

Here we are sending data through Postman



Key	Value	Type
(1) ObjectID("5c18e1892998bd3b3d355bf")	{ 3 fields }	Object
(2) ObjectID("5c139771d79ac8eac11e754a")	{ 3 fields }	Object
(3) ObjectID("5c1398aad79ac8eac11e7561")	{ 3 fields }	Object
(4) ObjectID("5c1398ecd79ac8eac11e7567")	{ 3 fields }	Object

MongoDB Database

```
app.post("/articles", function(req, res){
    console.log();
    console.log();

    const newArticle = new Article({
        title: req.body.title,
        content: req.body.content
    });

    newArticle.save();
});
```

Key	Value	Type
(1) ObjectID("5c18e1892998bd3b3d355bf")	{ 3 fields }	Object
(2) ObjectID("5c139771d79ac8eac11e754a")	{ 3 fields }	Object
(3) ObjectID("5c1398aad79ac8eac11e7561")	{ 3 fields }	Object
(4) ObjectID("5c1398ecd79ac8eac11e7567")	{ 3 fields }	Object
(5) ObjectID("5c18f35cd40ab6cc551cd60")	ObjectID("5c18f35cd40ab6cc551cd60") String String - Int32	Object

after sending from postman

```

app.post("/articles", function(req, res){
  const newArticle = new Article({
    title: req.body.title,
    content: req.body.content
  });

  newArticle.save(function(err){
    if (!err){
      res.send("Successfully added a new article.");
    } else {
      res.send(err);
    }
  });
});

```

HTTP Verbs	/articles	/articles/jack-bauer
GET	Fetches all the articles ✓	Fetches the article on jack-bauer
POST	Creates one new article ✓	-
PUT	-	Updates the article on jack-bauer
PATCH	-	Updates the article on jack-bauer
DELETE	in next page Deletes all the articles ✓	Deletes the article on jack-bauer

app.delete (route, function (req, res) {})
 (in express)

#<Model name>.deleteMany () (in mongoose)
 {conditions} //empty if want to delete everything
 functions (err) {}
);

```

app.delete("/articles", function(req, res){
  Article.deleteMany(function(err){
    if (!err){
      res.send("Successfully deleted all articles.");
    } else {
      res.send(err);
    }
  });
});

```

now when you send a delete request from postman to localhost:3000/articles output: Successfully deleted all articles.

and now when get request on localhost:3000/articles output: [] empty array

```
app.route('/book')
  .get(function (req, res) {
    res.send('Get a random book')
  })
  .post(function (req, res) {
    res.send('Add a book')
  })
  .put(function (req, res) {
    res.send('Update the book')
  })
```

to create chainable route
handlers for a route
Sequence is important here

```
app.route("/articles")

.get(function(req, res){
  Article.find(function(err, foundArticles){
    if (!err) {
      res.send(foundArticles);
    } else {
      res.send(err);
    }
  });
}) no semicolon as we do not want
      to end here
.post(function(req, res){

  const newArticle = new Article({
    title: req.body.title,
    content: req.body.content
  });

  newArticle.save(function(err){
    if (!err){
      res.send("Successfully added a new article.");
    } else {
      res.send(err);
    }
  );
}) no semicolon, we do not want to
      end here
.delete(function(req, res){
  Article.deleteMany(function(err){
    if (!err){
      res.send("Successfully deleted all articles.");
    } else {
      res.send(err);
    }
  );
});
```

Requests targeting Specific articles

Route parameters

Route parameters are named URL segments that are used to capture the values specified at their position in the URL. The captured values are populated in the `req.params` object, with the name of the route parameter specified in the path as their respective keys.

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

To define routes with route parameters, simply specify the route parameters in the path of the route as shown below.

```
app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params)
})
```

```
app.route("/articles/:articleTitle")
```

```
.get(function(req, res){
  Article.findOne({title: req.params.articleTitle}, function(err, foundArticle){
    if (foundArticle) {
      res.send(foundArticle);
    } else {
      res.send("No articles matching that title was found.");
    }
});
```

HTTP Verbs	/articles	/articles/jack-bauer
GET	Fetches all the articles ✓	Fetches the article on jack-bauer ✓
POST	Creates one new article ✓	
PUT		Updates the article on jack-bauer
PATCH		Updates the article on jack-bauer
DELETE	Deletes all the articles ✓	Deletes the article on jack-bauer

// first : wikiDB < collections < articles < insert Document
Paste all the deleted data from previous page & save
now in Postman : get request : localhost:3000/articles/keit
output :

```
  "_id": "5c18e1892998bdb3b3d355bf",
  "title": "REST",
  "content": "REST is short for REpresentational State Transfer. It's an architectural style for designing APIs."
}
```

localhost:3000/articles/jack%20Bauer % 20 = Space

GET localhost:3000/articles/Jack%20Bauer

Params Authorization Headers (1) Body Pre-request Script Tests

KEY	VALUE	DESCRIPTION	Code
Key	Value	Description	Bulk Edit

Status: 200 OK Time: 18 ms Size: 378 B Download

Pretty Raw Preview JSON

```
1 - [
2   "_id": "5c18f35cde40db6cc551cd60",
3   "title": "Jack Bauer",
4   "content": "Jack Bauer once stepped into quicksand. The quicksand couldn't escape and nearly drowned.",
5   "__v": 0
6 }
```

```

95
<ModelName>.update(
  {conditions},
  {updates},
  {overwrite: true},
  function(err, results){}
);

```

* chaining to app route (" /articles/:articleTitle ");

```

.put(function(req, res){
  Article.update(
    {title: req.params.articleTitle},
    {title: req.body.title, content: req.body.content},
    {overwrite: true}, same
    function(err){
      if(!err){
        res.send("Successfully updated article.");
      }
    });
}); I

```

* we will send a put request from postman

Important

PUT localhost:3000/articles/jack%20Bauer

Params: Authorization: Headers (1) Body (1) Pre-request Script: Tests: None

Body (x-www-form-urlencoded)

Key	Value
content	The First rule of Chuck Norris is: you do not talk about Chuck Norris.

Status: 200 OK Time: 24 ms Size: 234 B

Even if we don't send title it will be a successful response
↓ output

PUT localhost:3000/articles/jack%20Bauer

Params: Authorization: Headers (1) Body (1) Pre-request Script: Tests: None

Body (x-www-form-urlencoded)

Key	Value	Description
title	same	Chuck Norris
content	same	The First rule of Chuck Norris is: you do not talk about Chuck Norris.

articles (1) 0.03 sec.

Key	Value	Type
0	{ "_id": "5c18e1b922980d03a020355bf", "title": "Jack Bauer", "content": "The First rule of Chuck Norris is: you do not talk about Chuck Norris."}	Object

update will overwrite the default mongo DB behaviour and will only apply the changes which are sent

* problem: updates full document

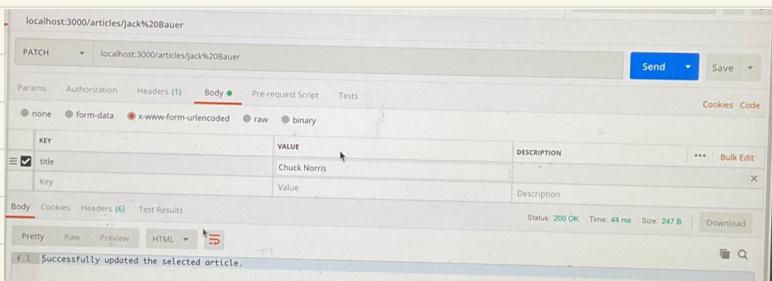
Patch: Update particular document

UPDATE

```
<ModelName>.update(  
  {conditions},  
  {$set: updates},  
  function(err, results){}  
);
```

```
.patch(function(req, res){  
  Article.update(  
    {title: req.params.articleTitle}, // condition  
    {$set: req.body}, will only update what we sent  
    function(err){  
      if(!err){  
        res.send("Successfully updated article.")  
      } else {  
        res.send(err);  
      }  
    }  
  );  
});
```

/we are only updating title and
rest will be as it is.



will send
req.body = {
title: "Chuck Norris"
}

Delete specific document

 Sending delete request

```
+ app.delete('/route', Function (req, res){ {} }) //for express
```

DELETE

for MongoDB

```
<ModelName>.deleteOne(  
  {conditions},  
  function(err){}  
);
```

```
.delete(function(req, res){  
  Article.deleteOne(  
    {title: req.params.articleTitle},  
    function(err){  
      if (!err){  
        res.send("Successfully deleted the corresponding article.");  
      } else {  
        res.send(err);  
      }  
    }  
  );  
});
```