* get request = Read

* Post request = Create

* Put & Patch request = Update

  - Put: sending entire entry to replace previous one
  - Patch: only sending piece of data which needs to be updated

* delete = delete

* POSTMAN

* 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

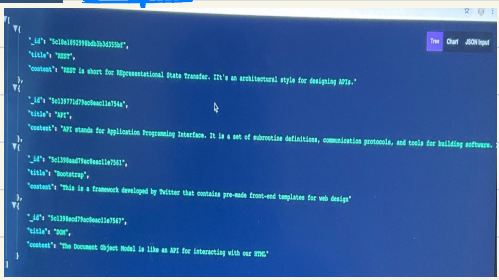
```
//jshint esversion:6

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", {use NewUrlParser: 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.
});
```

* now when in browser localhost:3000/articles output

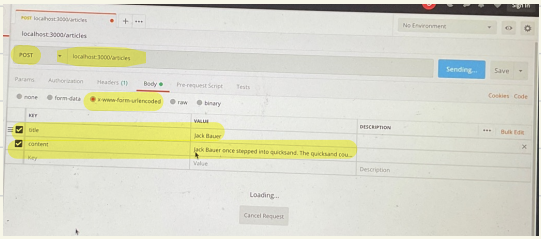
```
const Article = mongoose.model("Article", articleSchema);

app.get("/articles", function(req, res){
  Article.find(function(err, foundArticles){
    if (!err) {
      res.send(foundArticles);
    } else {
      res.send(err);
    }
  });
});
```

Here we are finding everything so first parameter is not passed

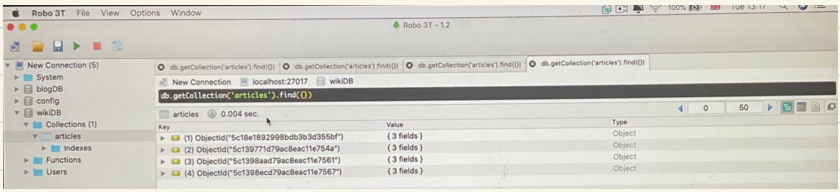


we created these in Robo 3T under articles

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



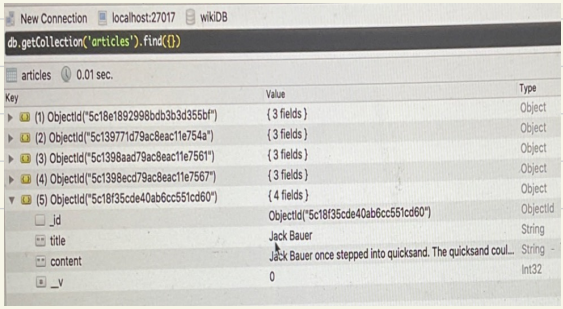
Here we are sending data through Postman



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();
});
```

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
  function (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 on 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)
})
```

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

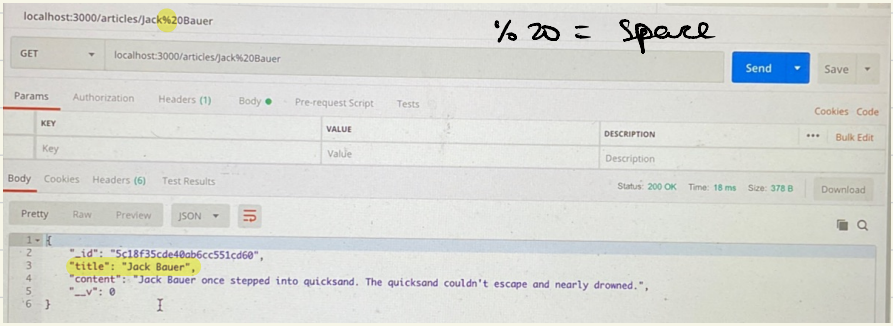
```
app.route("/articles/:articleTitle")

.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.");
    }
  });
});
```

// first : wikiDB < Collections < articles < insert Document
paste all the deleted data from previous page & save

now in Postman : get request : localhost:3000/articles/Rest

output :

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



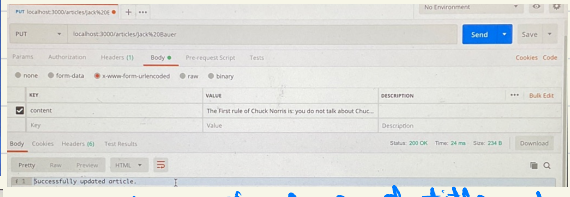
```
<ModelName>.update(
    {conditions},
    {updates},
    {overwrite: true},
    function(err, results){}
);
```

* 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},
    function(err){
      if(!err){
        res.send("Successfully updated article.");
      }
    }
  );
});
```

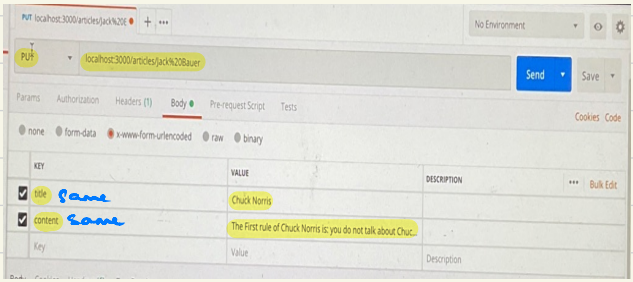
* we will send a put request from postman

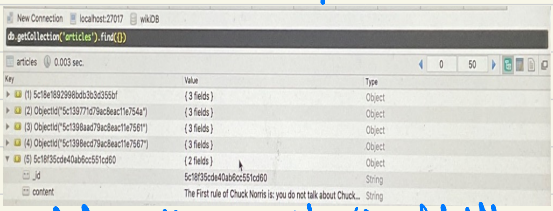
Important



Even if we dont send title it will be a successful response

⇓ output





* Problem: updates full document

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

#Patch : Update particular document

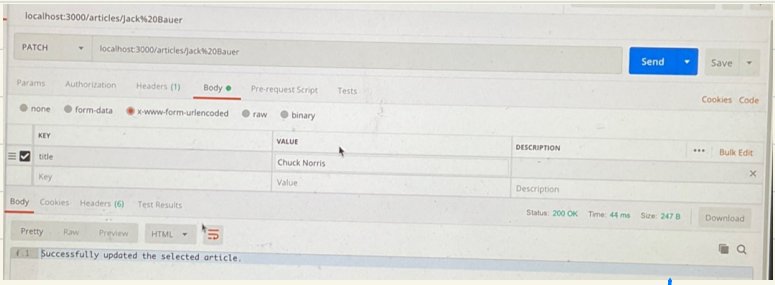
UPDATE

```
<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 is 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

ok



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);
      }
    }
  );
});
```