

Fundamentals of Web Development

Third Edition by Randy Connolly and Ricardo Hoar



Chapter 7

Working with Databases

Part 2: CRUD

Which HTTP method to use: What is CRUD?

- In HTTP we have different request methods that should be used for the corresponding database operation:

Database operation	HTTP method	express function
C reate	POST is used to create new data in the backend (in the database for example).	app.post()
R ead	By default, GET is used, GET is used to get a resource from the server (webpage, css file, image, data, etc.)	app.get()
U ppdate	PUT is used to update existing data in the backend.	app.put()
D eleate	DELETE is used to delete data	app.delete()

Our case study

- We will start by creating the structure of a complete express MVC application.
- The goal is to create:
 - a form to add articles to the database.
 - A table to show retrieved data.
 - a button that allows us to delete an item.
 - An update button on the form to update data.

The screenshot shows a web browser at `localhost:3000/articles/add`. The page has an orange header with the title 'EKSTRA'. Below the header is a dark navigation bar with three links: 'Home' (highlighted in green), 'Add Article', and 'Search Articles'. The main content area contains a form titled 'Add Article'. The form has three input fields: 'Name' (containing 'name'), 'Code' (containing 'CODE'), and 'Description' (containing 'add a description here....'). Below these fields is a 'Submit' button. At the bottom of the page, there is an orange footer with a 'Home' link.

Design the API endpoints

Operation	Route	HTTP Method
To open the home page To open the form page	/ /v1/ articles /page	GET
To retrieve articles	/v1/ articles /v1/ articles /id/:id /v1/ articles /category/:category	GET
To add a new article	/v1/ articles /code/:code/name/:name/desc/:desc	POST
To update an article	/v1/ articles /id/:id/code/:code/name/:name/desc/:desc	PUT
To delete an article	/v1/ articles /id/:id	DELETE

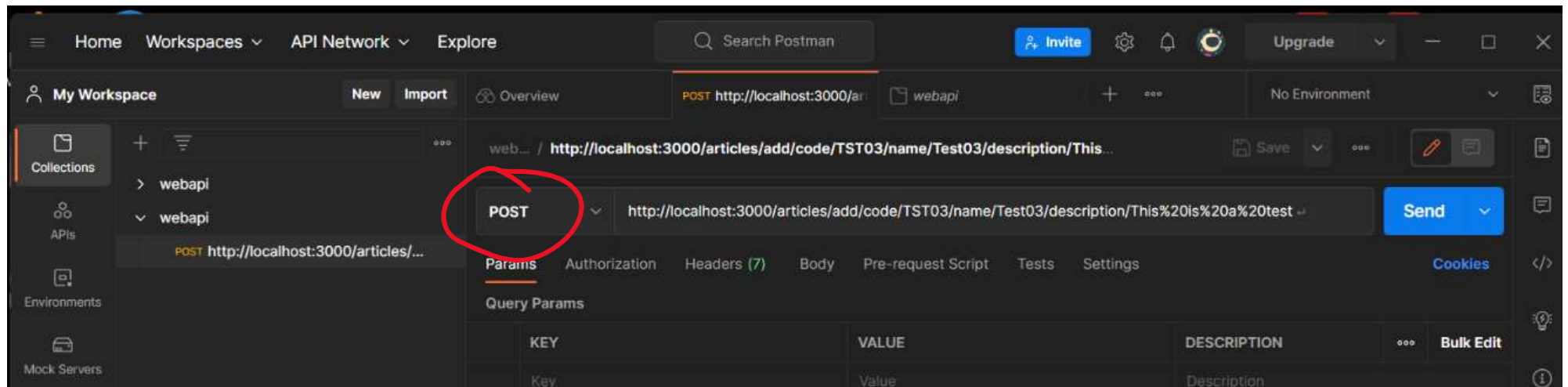
These two routes return web pages

These routes return json data only

- By using /v1/**articles** with all operations related to articles, we can create a router that handles only articles operations -> better modularity in our code

How to test a web API

- Using the browser we can send GET requests through the address bar.
- How to test POST, PUT, DELETE requests?
- Postman is a web API testing tool



Add a new article

- First, we need a form with method="post" and the correct route as the form action.
- We can add basic HTML5 validation.

```
<form method="post" action="/v1/articles/">
  <fieldset class="add_article_form">
    <legend>Add Article</legend>
    <p><label>Name</label>
      <input type="text" name="name" placeholder="name" required>
    </p>
    <p><label>Code</label>
      <input type="text" name="code" placeholder="CODE" required>
    </p>
    <p><label>Description</label>
      <input type="text" name="description" placeholder="add a description here....">
    </p>
    <input type="submit">
  </fieldset>
</form>
```

The screenshot shows a web application interface for adding a new article. The header is yellow with the text 'EKSTRA'. Below the header is a dark navigation bar with links: 'Home' (highlighted in green), 'Add Article', and 'Search Articles'. The main content area is white and contains a form titled 'Add Article'. The form has three input fields: 'Name' (with placeholder 'name'), 'Code' (with placeholder 'CODE'), and 'Description' (with placeholder 'add a description here....'). The 'Code' field is highlighted with a red border and a tooltip that says 'Please fill out this field.' indicating it is a required field. At the bottom of the form is a 'Submit' button. The footer is yellow and contains a 'Home' link.

Server-side input validation

- We need to validate the input coming from the client before executing the insertion operation. For example:
 - name is a string of at least 3 characters
 - email is a real email
 - age is a number, between 0 and 110
- In Express, we can use the [express-validator](#) module:
- `npm install express-validator` // to install it

Then, we need to import it :

```
const { check, validationResult } = require('express-validator');
```

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

app.use(express.json());

app.post('/form', (req, res) => {
    const name = req.body.name
    const email = req.body.email
    const age = req.body.age
})
```

Server-side input validation (2)

- We pass an array of check() calls as the second argument of the post()
- call. Every check() call accepts the parameter name as argument. Then we
- call validationResult() to verify there were no validation errors.

```
app.post('/form', [ check('name').isLength({ min: 3 }),
                    check('email').isEmail(),
                    check('age').isNumeric()
                  ],
  (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(422).json({ errors: errors.array() });
    }
    const name = req.body.name;
    const email = req.body.email;
    const age = req.body.age;
    ....
  });
```


Validation methods

- In the previous example, we used `isEmail`, `isNumeric`, etc. These are defined in **validator.js** which has been imported behind the scenes by **express-validator**
- Here are other methods from `validator.js`:
- All those checks can be combined:

```
check('name').isAlpha().isLength({ min: 10 })
```

<code>contains()</code>	<code>equals()</code>	<code>isAlphanumeric()</code>
<code>isBoolean()</code>	<code>isCurrency()</code>	<code>isDecimal()</code>
<code>isEmpty()</code>	<code>isFloat()</code>	<code>isIP()</code>
<code>isInt()</code>	<code>isJSON()</code>	<code>isLowercase()</code>
<code>isPostalCode()</code>	<code>isURL()</code>	<code>isUppercase()</code>

Validating our form input

- In the router, we have to pass check calls as a middleware to the post function.
- Here, we consider that the name needs to contain at least 3 characters, the code at least two, but the description is not validated.

```
// in articlesRoutes.js
// First we only import the check function
const { check } = require('express-validator');

// Then, we pass two checks as a middleware to the post method
router.post('/add', [ check('name').isLength({ min: 3 }),
                     check('code').isLength({ min: 2 }) ],
             , articlesController.add_article
             );
```

- Then, if we try to add an article with a name 'DF', we will get this response:

```
{"errors":[{"value":"DF","msg":"Invalid value","param":"name","location":"body"}]}
```



Validating our form input (2)

- In the controller, we check the results of the validation first.

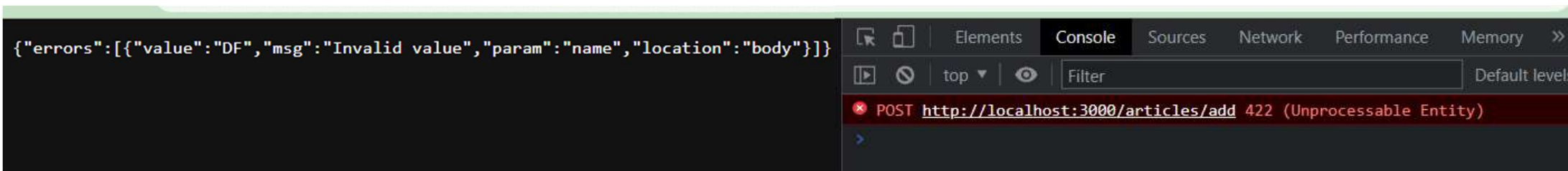
```
const add_article = (request, response) => {
  const errors = validationResult(request);
  if (!errors.isEmpty()) {
    return response.status(422).json({ errors: errors.array() });
  }

  let name = request.body.name;
  let code = request.body.code;
  let description = request.body.description;

  let art = new Article({ name: name, code: code, description: description });
  art.save()
    .then((data) => {
      console.log(`Article saved to database: id -> ${data._id}`);
      response.redirect(`/v1/articles/add`); // redirect to the same page (empty form)
    })
    .catch((err) => { console.log(err) });
};
```

Why return a 422 HTTP error message?

- The 422 HTTP error message means that the server understands the content type of the request entity, and the syntax of the request entity is correct (thus a 400 Bad Request status code is inappropriate) but was unable to process the contained instructions.



Search articles form

- We want to search for articles that have the name or code containing the given string with case insensitivity.

Search for Articles

Name contains:

name

Code Contains:

CUT_290

Search

Id	Name	Code	Description
63754285d4f42b348f1cd34c	Cutter	CUT_290	A paper cutter

```
<form id="search-form" method="get" action="/v1/articles/">
  <fieldset class="form-article">
    <legend>Search for Articles</legend>
    <p><label>Name contains:</label>
      <input type="text" name="name" placeholder="name" ></p>
    <p><label>Code Contains:</label>
      <input type="text" name="code" placeholder="CODE" ></p>
    <button type="submit" form="search-form" value="Submit">Search</button>
  </fieldset>
</form>
```

Search articles form

```
const find_articles = (request, response) => {
  let name = request.query.name;
  let code = request.query.code;
  let nameRegex = (name.length > 0)? new RegExp(name, 'i') : null;
  let codeRegex = (code.length > 0)? new RegExp(code, 'i') : null;
  let search = {};
  if( nameRegex != null && codeRegex != null){
    search = { $or: [ { 'name': { "$regex": nameRegex } },
                      { 'code': { "$regex": codeRegex } }
                    ]};
  } else if(codeRegex != null){
    search = { 'code': { "$regex": codeRegex } };
  } else if(nameRegex != null){
    search = { 'name': { "$regex": nameRegex } };
  } else { response.status(204).end(); } // no data, send no data HTTP code
  Article.find(search)
    .then((data) => {
      if(response._closed == false) // response was not already sent
        response.render('search_articles', {title: "Search Articles", articles: data});
    })
    .catch((err) => { console.log(err) });
};
```

Delete an article

We need to send a fetch request to delete an element:

Home Add Article Search Articles

Search for Articles

Name contains:

Code Contains:

Search

Why return a 422 HTTP error mess...

Id	Name	Code	Description	
63754285d4f42b348f1cd34c	Cutter	CUT_290	A paper cutter	Delete
637543aafbc023f52ed3d937	White Paint Exterior	WHT_PNT_EXT	A paint of white color for exterior walls	Delete
637543d73ded9861ddb81bbe	Black paint exterior	BLK_PNT_EXT	A black paint for exterior surfaces	Delete
63792f1a4d0bc335949ba6af	TRTH	GJH_098		Delete

```
<article id="articles_table">
  <% if( articles != undefined && articles != null ){ %>
    <script>
      const deleteArt = (e) => {
        let delete_promise = fetch('/v1/articles/' + e.id, { method: 'DELETE'})
        delete_promise.then(response => response.json())
          .then( data => {
            window.location = data.redirect;
          })
        .catch((reject) => {
          document.getElementById("message_delete").textContent = reject;
        });
      }
    </script>
    <table>
      <thead>
        <tr><th>Id<th>Name<th>Code<th>Description<th></th>
      <tbody>
        <% if(articles.length > 0){ %>
          <% articles.forEach((article) => { %>
            <tr>
              <td><%= article.id %>
              <td><%= article.name %>
              <td><%= article.code %>
              <td><%= article.description %>
              <td><button id="<%= article.id %>" onclick="deleteArt(this)" data>
                Delete</button></td>
            <% }) %>
          </tr>
        <% }} %>
      </tbody>
    </table>
  <% }} %>
</article>
```

Delete an article (2)

- On the server-side, we need to handle the delete request by the router.
- Then, in the controller, we need to use the findByIdAndDelete.
- Finally, we cannot redirect from the server with the fetch call, so we can send back a redirect string in a json object, and the client-side fetch will redirect to the given route.

```
// in the routes.js
router.delete('/:id', articlesController.delete_article);

// in the controller.js
const delete_article = (request, response) => {
  let id = request.params.id;

  Article.findByIdAndDelete(id)
    .then((result) => {
      response.redirect(`/v1/articles/search`);
    })
    .catch((err) => { console.log(err) });
};
```


Update an article

- To update an article, we should use the `findOneAndUpdate` mongoose function. The function returns the old copy of the object if the update succeed:

```
router.put('/id/:id/code/:code/name/:name/desc/:desc', articlesController.update_article);
```

```
const update_article = (request, response) => {  
  const filter = { id: request.params.id };  
  const update = { code: request.params.code,  
                  name: request.params.name,  
                  desc: request.params.desc  
                };  
  
  Article.findOneAndUpdate(filter, update)  
    .then((result) => {  
      response.redirect('/v1/articles/search');  
    })  
    .catch((err) => { console.log(err) });  
};
```