This work is licensed under CC BY 4.0

Create Read Update Delete

How to think about CRUD operations regarding routes and Mongoose



CRUD, Verbs, and Actions

- A route provides a mapping between HTTP verbs and URLs to controller actions.
- Each action maps to a specific CRUD operation in a database.
- Route parameters, such as :id, are named URL segments that are used to capture the values specified at their position.

HTTP Verb	Path	Controller#Action	Used for
GET	/tasks	TaskController#index	display a list of tasks
GET	/tasks/create	TaskController#create	return an HTML form for creating a new task
POST	/tasks/create	TaskController#createPost	create a new task
GET	/tasks/:id/update	TaskController#update	return an HTML form for editing a task
POST	/tasks/:id/update	TaskController#updatePost	update a specific task
GET	/tasks/:id/delete	TaskController#delete	return an HTML form for removing a task
POST	/tasks/:id/delete	TaskController#deletePost	delete a specific task

Routes Mapping to the TaskController

```
// src/routes/taskRouter.js
import express from 'express'
import { TaskController } from '../controllers/TaskController.js'

export const router = express.Router()

const controller = new TaskController()

// Provide req.doc to the route if :id is present in the route path.
router.param('id', (req, res, next, id) => controller.loadTaskDocument(req, res, next, id))

// Map HTTP verbs and route paths to controller action methods.
router.get('/', (req, res, next) => controller.index(req, res, next))

router.get('/create', (req, res, next) => controller.create(req, res, next))
router.post('/create', (req, res, next) => controller.createPost(req, res, next))

router.get('/:id/update', (req, res, next) => controller.update(req, res, next))
router.post('/:id/update', (req, res, next) => controller.updatePost(req, res, next))

router.get('/:id/delete', (req, res, next) => controller.delete(req, res, next))
router.post('/:id/delete', (req, res, next) => controller.deletePost(req, res, next))
```

```
import express from 'express'
import { router as homeRouter } from './homeRouter.js'
import { router as taskRouter } from './taskRouter.js'

export const router = express.Router()

router.use('/', homeRouter)
router.use('/tasks', taskRouter)

router.use('*', (req, res, next) => {
  const error = new Error('Not Found')
  error.status = 404
  next(error)
})
```

- ✓ 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.
- ✓ If `:id` is present in a route path, task document loading logic is mapped to automatically provide `req.doc` to the route.

HTTP Verb	Path	Controller#Action	Used for
GET	/tasks	TaskController#index	display a list of tasks
GET	/tasks/create	TaskController#create	return an HTML form for c
POST	/tasks/create	TaskController#createPost	create a new task
GET	/tasks/:id/update	TaskController#update	return an HTML form for ed
POST	/tasks/:id/update	TaskController#updatePost	update a specific task
GET	/tasks/:id/delete	TaskController#delete	return an HTML form for ren
POST	/tasks/:id/delete	TaskController#deletePost	delete a specific task

Loading the task document

```
// src/controllers/TaskController.js
import { TaskModel } from '../models/TaskModel.js'

/**
 * Provide req.doc to the route if :id is present.
 *
 * @param {object} req - Express request object.
 * @param {object} res - Express response object.
 * @param {Function} next - Express next middleware function.
 * @param {string} id - The value of the id for the task to load.
 */
async loadTaskDocument (req, res, next, id) {
  try {
    // Get the task document.
    const taskDoc = await TaskModel.findById(id)

    // If the task document is not found, throw an error.
    if (!taskDoc) {
      const error = new Error('The task you requested does not exist.')
      error.status = 404
      throw error
    }

    // Provide the task document to req.
    req.doc = taskDoc

    // Next middleware.
    next()
  } catch (error) {
    next(error)
  }
}
```

- ✓ When the route path contains an :id parameter, `loadTaskDocument` is invoked. The fourth parameter stores the :id value. Using `router.param` for document-loading offers advantages:
  - **Code Reusability**: Centralizing the document-loading process in a middleware function enables code reuse across multiple routes that require the same document, reducing duplication and simplifying maintenance.
  - **Separation of Concerns**: Dedicated middleware allows route handlers to focus on specific tasks.
  - **Error Handling**: Streamlines error management and ensures consistent error responses.
  - **Improved Performance**: Fetching the document once in middleware enhances performance.
  - **Consistency**: Guarantees a uniform document-loading process, avoiding issues and enhancing readability.

Create – the C in CRUD

```
// src/controllers/TaskController.js
import { TaskModel } from '../models/TaskModel.js'

  /**
   * Creates a new task.
   *
   * @param {object} req - Express request object.
   * @param {object} res - Express response object.
   */
  async createPost (req, res) {
    try {
      const { description, done } = req.body

      await TaskModel.create({
        description,
        done: done === 'on'
      })

      req.session.flash = { type: 'success', text: 'The task was created successfully.' }
      res.redirect('.')
    } catch (error) {
      req.session.flash = { type: 'danger', text: error.message }
      res.redirect('./create')
    }
  }
```

- ✓ Create a new instance of the model and save the document to the database.
  - `TaskModel.create()` is a shortcut for saving one or more documents to the database. The shortcut is instead of `new TaskModel({…}).save()`.

    Easy peasy!

Read – the R in CRUD

```
// src/controllers/TaskController.js
/**
 * Displays a list of tasks.
 *
 * @param {object} req - Express request object.
 * @param {object} res - Express response object.
 * @param {Function} next - Express next middleware function.
 */
async index (req, res, next) {
  try {
    const viewData = {
      tasks: (await TaskModel.find())
        .map(taskDoc => taskDoc.toObject())
    }

    res.render('tasks/index', { viewData })
  } catch (error) {
    next(error)
  }
}
```

- ✓ Use `find()` to find all documents in a collection.
- ✓ It is good practice to transform the documents into plain old JavaScript objects before passing the data to the view.

  An easy way of doing it is to use the `toObject()` method defined in the base schema.

```
// src/model/baseSchema.js
const convertOptions = Object.freeze({
  …
  transform: (doc, ret) => {
    delete ret._id
    return ret
  }
})

const baseSchema = new mongoose.Schema({}, {
  …
  toObject: convertOptions,
  toJSON: convertOptions,
  …
})

export const BASE_SCHEMA = Object.freeze(baseSchema)
```

Delete `_id` (the virtual `id` property , a string, is generated by default).

Update – the U in CRUD

```
// src/controllers/TaskController.js
/**
 * Updates a specific task.
 *
 * @param {object} req - Express request object.
 * @param {object} res - Express response object.
 */
async updatePost (req, res) {
  try {
    if ('description' in req.body) req.doc.description = req.body.description
    if ('done' in req.body) req.doc.done = req.body.done === 'on'

    if (req.doc.isModified()) {
      await req.doc.save()
      req.session.flash = { type: 'success', text: 'The task was updated successfully.' }
    } else {
      req.session.flash = {
        type: 'info',
        text: 'The task was not updated because there was nothing to update.'
      }
    }
    res.redirect('..')
  } catch (error) {
    req.session.flash = { type: 'danger', text: error.message }
    res.redirect('./update')
  }
}
```

- ✓ Query first approach (pretty much the same in all languages and frameworks).
  1. Find the document by its id using `findById()`.
     - Which is precisely what `loadTaskDocument` already done.
  2. If the document was found `updatePost` will be called, and…
     a) Modify the appropriate properties.
     b) Check if the document is modified.
     c) Make a call to the `save()` method.
- ✓ You can also use `findByIdAndUpdate()`, or `updateOne()` , or …, to update a document in the database.
  - BUT #1, the document is already loaded.
  - BUT #2, if you're planning to implement concurrency control, stick with the save method.

Delete – the D in CRUD

```
// src/controllers/TaskController.js
/**
 * Deletes the specified task.
 *
 * @param {object} req - Express request object.
 * @param {object} res - Express response object.
 */
async deletePost (req, res) {
  try {
    await req.doc.deleteOne()

    req.session.flash = { type: 'success', text: 'The task was deleted successfully.' }
    res.redirect('..')
  } catch (error) {
    req.session.flash = { type: 'danger', text: error.message }
    res.redirect('./delete')
  }
}
```

- ✓ Use `deleteOne()` to remove the document loaded by `loadTaskDocument()`.

