RESTful APIs with Express

In this lab, you will build upon the "todo" application created in the previous lab and create a REST API alongside the web interface.

# Objectives

In this lab, you will learn to

* define RESTful routes in Express, and
* implement behavior for each route.

# Set things up

You have been provided a working Express application to start the lab but first we need to install the required modules.

## Install dependencies

X. Run npm install in the lab directory. This will install the dependencies for our application from the package.json file. In this case, our only dependency is express, an MVC web framework for Node.js.

While not required, you probably want to install nodemon (npm –g nodemon). nodemon automatically restarts your application when a file system change is detected and will save time so you don’t have to manually restart Node to view your edits.

## Specify the findAllJson route

X. Open routes/index.js in your editor and define a HTTP GET route for the REST path to get all todos, /api/todos, by using app.get. The first argument to app.get is the URL path for the matching route, and the second argument is the callback function that will handle the request. In the spirit of a layered architecture, our controller layer concerning todos will be placed in module todo-controller.js located in the same directory. The service layer will be placed in the services directory. You can see that as your application grows, this allows your routes/index.js to remain easy to read and keeps the controllers separate. Use todoController.findAllJson, which we'll write in the next step, as the second argument for your route.

Our REST API will be prefixed with /api so there is no confusion about which routes belong to the web layer and which belong to our API.

## Create the REST endpoint to get all todos

X. Open routes/todo-controller.js and provide an implementation for the function findAllJson(req, res, next). In your implementation, delegate to the service.findAll function, and pass to it a callback of the form function onFoundJson(err, todos). The service.findAll function has already been implemented from the previous lab. In the callback, use the res.json(todos) function to send a JSON response containing all todos. If there is an error, return it with res.json({message : err.message}).

response.json() is identical to response.send() when an object or array is passed. It may also be used for explicit JSON conversion of non-objects (null, undefined, etc.).

In the event of an error, be sure to always send a JSON response containing your error message and any other important information (stack trace, error code, etc) rather than sending a response code indicating failure like 500. Sending error information as a JSON response is a best practice to give consumers of your API as much help as possible to diagnose a problem.

## Start the application and test the REST endpoint to get all todos

X. Run node app.js (or nodemon app.js). Open a separate terminal window and issue the command curl -i http://localhost:3000/api/todos. If everything is working correctly, you should see a JSON response with three sample todo items.

If you are able to run Google Chrome on your lab machine and would prefer to use a graphical REST client, please refer to Appendix A.

## Create the REST endpoint to create a todo

X. Open routes/index.js and define a HTTP POST route for path /api/todos and function addTodoJson(req, res, next) in routes/todo-controller.js.

REST conventions dictate that the HTTP method GET is used for reading resources, POST for creating, PUT for updating, and DELETE for deleting.

The addTodoJson function in routes/todo-controller.js should call the service layer's addTodo function and pass req.body.content, along with a callback of the form function onAddedJson(err, todo). In the callback, use res.json(todo) to send a JSON response containing the newly added todo item. If there is an error, return it with res.json({message : err.message}).

## Test the creation endpoint

X. Test the addTodoJson route by issuing a command in the form curl -i -X POST -H "Content-Type: application/json" -d '{ "content" : "Get the milk" }' http://localhost:3000/api/todos. The new todo item should be returned in a JSON response. You can also invoke the REST endpoint GET '/api/todos' to see the entire list with your new item added.

## Create the REST endpoint to get a single todo by id

X. Open routes/index.js and define a HTTP GET route for /api/todos/:id and corresponding controller function findOneJson in routes/todo-controller.js. The token :id in the route is a named route parameter that allows the id parameter to be accessed via request.params.id. The findOneJson function should call the service layer's findOne and pass the todo's id and a callback of the form function onFoundJson(err, todo). The service layer function findOne has already been provided for you. In the callback, use res.json(todo) to send the JSON response. If there is an error, return it with res.json({message : err.message}).

## Test the REST endpoint to get a single todo

X. Test the findOneJson route by issuing a command in the form curl -i http://localhost:3000/api/todos/508d4a36-3c7c-7bb3-c5ee-1c4c3fb440f6. The todo item specified by the id should be returned.

## Create the REST endpoint to update a todo

X. Open routes/index.js and define a HTTP PUT route for /api/todos/:id and controller function updateTodoJson in routes/todo-controller.js. Like the findOneJson route, updateTodoJson should use a named route parameter for the todo id. The updateTodoJson function should call the service layer's updateTodo function and pass the updated todo's id and content for the item being changed, and a callback of the form function onUpdatedJson(err, todo). In the callback, use res.json() to return the updated todo's id. If there is an error, return it with res.json({message : err.message}).

## Test the REST endpoint to update a todo

X. Test the updateTodoJson route by updating the "Get the mail" todo to be "Pick up pizza" instead. Issue the command curl -i -X PUT -H "Content-Type: application/json" -d '{ "content" : "Pick up pizza" }' http://localhost:3000/api/todos/508d4a36-3c7c-7bb3-c5ee-1c4c3fb440f6. A JSON response of the updated todo's id should be returned.

## Create the REST endpoint to delete a todo

X. Open routes/index.js and define a HTTP DELETE route for /api/todos/:id and controller function deleteTodoJson in routes/todo-controller.js. Like the findOneJson and updateTodoJson routes, we need to use a named route parameter for the id corresponding to the specific todo item we wish to remove. Call the service layer deleteTodo function, pass in the id of the item to be deleted, and a callback of the form function onDeletedJson(err). If there is no error, use res.json() to return the id of the deleted item. If there is an error, return it with res.json({message : err.message}).

## Test the REST endpoint to delete a todo

X. Test the deleteTodoJson route by issuing a command in the form curl -i -X DELETE http://localhost:3000/api/todos/508d4a36-3c7c-7bb3-c5ee-1c4c3fb440f6. If the command is successful, the id of the deleted item should be returned.

Once everything is working properly, this lab is now complete!

# Appendix A

## Google Chrome

If your lab machine is equipped with Google Chrome, you may find it easier to work with a GUI based REST client rather than the curl command line utility. There are a number to choose from in the Chrome Web Store. One client that has worked well for the lab authors is Postman: <http://www.getpostman.com>.