Learning Objectives

Learners will be able to...

- **create a simple HTTP server for several endpoint paths,**
- **process GET requests for their server,**
- **extract parts of the URL that represent the entities**
- **assemble requests to the server using curl and the JS client.**

info

Make Sure You Know

This assignment assumes a basic knowledge of JavaScript, HTML and CSS and everything that has been covered in the previous assignments.

Limitations

We will not explain simple programming concepts, it is assumed that the user is familiar with these concepts.

Simplest HTTP Server

Express

One of the most popular server-side Javascript libraries for the REST APIs is the `express.js` framework. Express facilitates creating handlers for the different **HTTP** "verbs" such as GET, PUT, POST. This framework is simple yet flexible and there are many compatible libraries that handle the details related to web development. You can access the list of maintained libraries here.

Simple example

The simplest `express.js` app is in the code window on the upper left, 01-hello-world.js. This app always sends the string "Hello world!" for the URL path '/'.

`express()` creates an instance of the framework.

- `app.listen(port)` actually starts our app.
- The server starts accepting requests on the TCP (Transmission Control Protocol) port that we pass to the `app.listen` method.
- By default it listens on all available network interfaces meaning all IP (Internet Protocol) addresses of the current computer, including the IP of localhost.
- TCP/IP is a set of communication protocols used to connect computer systems in a network.
- Hypertext Transfer Protocol (HTTP) is a communication protocol that is one of the TCP/IP set.
- More than one user process accessing data on a server can use TCP, port numbers are used to identify the data associated with each process.
- `app.get` adds the handler for the requests with the GET method. It also specifies the accepted path name of the URL.
- The first parameter (`req` in our example) provides access to various properties of the HTTP request.
- The second parameter (`response`) allows us to modify the HTTP response before sending it.

Something must be sent, at least an empty body, or a string in this case, otherwise the request will hang and the client won't get anything in response.

important

Installing the node modules

The first time you run something in each of the assignments in this course you need to make sure that you have all the proper node modules installed. To do that run the commands below in the terminal.

```
cd examples
npm i
```

Starting the App

To start the app enter the following in the terminal. To stop the app use Ctrl-C in the terminal. You must be in the `examples` directory to run the command.

```
nodejs 01-hello-world.js
```

Make a GET request

The default method for a curl command is GET.

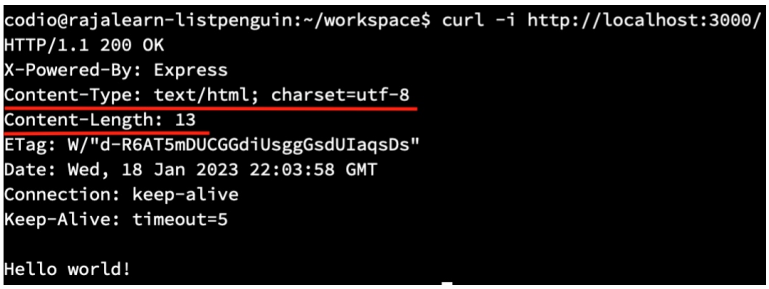
Open up a new terminal window and paste command below. You will see it return the expected string.

```
curl http://localhost:3000/
```

View the headers

Let's explore what `express.js` does in addition to just sending "Hello world!" here. The `-i` parameter in the curl command below tells it to display headers. Paste the command below in the new terminal window you created above.

```
curl -i http://localhost:3000/
```



Header output from the curl command

- The first line indicates the successful status “200 OK”.
- The “Content-Length” value is required and it indicates the length of the response.
- Notice the “Content-Type” value. We talked about this in the previous module when we discussed content negotiation. In this case the type is “text/html”.

Getting the response in a different format

More commonly we need the response in another format such as JSON for REST APIs. As JSON is often used in the Web services, there is a built-in function in express.js for that.

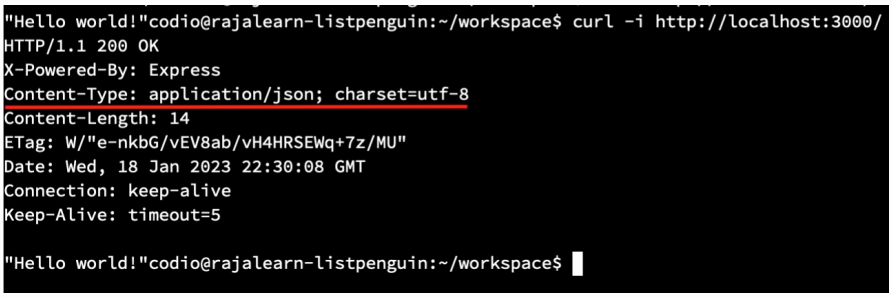
We’ll change the method used for the response: (`send` to `json`) you can see the changed code in 02-hello-world-json.js.

Restart the app using the second file we looked at. You must be in the `examples` directory to run the command.

```
nodejs 02-hello-world-json.js
```

Run curl again to see the changed headers

```
curl -i http://localhost:3000/
```



Now the “Content-Type” is `application/json; charset=utf-8`.

Also notice that there are now the quotes (`"Hello world!"` instead of `Hello world!`) because it is a valid JSON string.

challenge

Try this variation

- Change the path that is sent to `app.get`
- Restart the app
- What is different about the header?

Query parameters

In the previous module we created requests that contained query parameters. Now we will look at how query parameters are handled on the server side.

The example on the left illustrates the server echoing of the query text.

express.js puts the query parameters of the request into the `query` property. You can have multiple query parameters, the query string is parsed and each `key=value` parameter is added to a map of names to values, a JavaScript object.

Checking for text in a query and echoing that text

All query parameters are optional for express, it is up to the application developer to raise an error if a parameter that is required for the API service is missing.

A **400 Bad Request** should be sent if the server cannot process the request due to the client error.
As illustrated in this example, a missing parameter is a client error so we send the status `400` and then stop the handler function from continuing.

Starting the App

To start the app enter the following in the terminal. To stop the app use Ctrl-C in the terminal.

```
cd examples
nodejs 03-echo-query.js
```

Trying these test cases, the first line of the output shows the error:

- `curl -i "http://localhost:3000/"`
- `curl -i "http://localhost:3000/?text"`
- `curl -i "http://localhost:3000/?text=cat%20facts"`
  This is an example of two query parameters, separated by an `&`.
- `curl -i "http://localhost:3000/?text=cat%20facts&text=dogs"`

▼ What's the %20?
As we discussed in the previous module, a space character is a special character and as such must be "percent" encoded if there is one in the query string.

Specifying number of times to echo the request

We can add the number of times that the echo repeats, this is an optional parameter for the logic. It is important to validate the query parameters because it is untrusted user input. Otherwise this may lead to security problems or denial of service.

Sample that handles the optional `times` parameter.

In this example you can see what is done to make sure that times is set to a valid number.
- `parseInt(request.query.times, 10) || 1` the output of `parseInt` is "OR"ed with 1 in case it returns NaN (Not a Number)
- The `Math.max` portion of the code makes sure that `1` is the minimum number

Run this App

To start the app enter the following in the terminal. To stop the app use Ctrl-C in the terminal. You must be in the `examples` directory to run the command.

```
nodejs 04-echo-query-2.js
```

Trying the test cases below to see that invalid parameters are handled properly:

- `curl -i "http://localhost:3000/?text=cat%20facts"`
- `curl -i "http://localhost:3000/?times=-1&text=cat%20facts"`
- `curl -i "http://localhost:3000/?times=5&text=cat%20facts"`

Routes and path parameters

As we learned in the previous module, at a minimum the URI in a REST API must identify the resource. The path parameters allow you to identify a resource uniquely. Each separate path can be referred to as an endpoint or a route.

It is also possible for the same resource to be accessed via multiple URLs. For example, in a social network service the current user can be accessed both as `/users/id1234` and `/users/me`. Another example would be a document that can be accessed in multiple ways: both `/versions/2023-02-07` and `/versions/latest`.

REST APIs allow an unlimited number of resources but a limited number of "verbs" (GET, POST, PUT, DELETE, PATCH) on those resources. Your path has to be specific to what you want because you don't have unique methods to identify the data. For example, you only have `app.get` and not `app.getpets` so you need to specify the route - '/pets'. In the RPC model the addressable units are the procedures and "endpoints" are hidden from the user.

Example for a pet store:

```
app.get('/pets', (req, res) => {
  // ...
});

app.get('/orders', (req, res) => {
  // ...
});

app.get('/customers', (req, res) => {
  // ...
};
```

The routes in express are matched in order, i.e. if there are several handlers for the same path, only the first one will run unless specified explicitly.

Get a specific resource - not a collection

Usually that means that an identifier (also known as a descriptor) of a resource is part of the URI path. The identifier may be a number from the database or a descriptive string.

info

Starting the App

To start the app enter the following in the terminal. To stop the app use Ctrl-C in the terminal.

```
cd examples
nodejs 05-routes-and-paths.js
```

Try out a request

The example on the left is connected to a database with the information about pets. Notice that the get commands dispatch SQL to the database.
The following curl command returns the representation of the `pets` resource with an identifier `5`.

```
curl -s http://localhost:3000/pets/5
```

Resource hierarchy

As stated before, the resources of web APIs are usually organized into hierarchies.

While it is possible to store all the pet checkups extending from a single root, you would then need to filter via query parameters.

The pet checkups are not independent resources and it make mores sense to access them in a hierarchical fashion by petId. The request to this endpoint would fetch the list of checkups associated with a particular pet.

You would select a specific checkup using the `checkupId`.

It is also possible to specify multiple route parameters, for example:

```
app.get('/pets/:petId/images/:imageId, (req, res) => {
  console.log('Pet ID is ', req.params.petId);
  console.log('Image ID is ', req.params.imageId);
  // ... load the particular image of the pet from the data
storage
});
```

This request would fetch a checkup with id `1` for pet 5.

```
curl -s http://localhost:3000/pets/5/checkups/1
```

You can find out more about routing here.