# Creating Web Services

# Node HTTP Library

# HTTP Parser

A C program that is wrapped in JavaScript and brought into Node.js as a part of its core functionality via the http module.

It parses HTTP requests and responses - breaks the data apart and allows us to handle the individual aspects of a request or response.

# http.createServer()

The createServer() function on Node's HTTP module is how we create our own web services. This will allow us to write code that other developers can use.

The requestListener parameter is a callback that allows us to listen for incoming requests and provide those requests with a response.

# requestListener

The listener that hooks into the request event is provided with two parameters, an object that represents the incoming request and an object that represents the outgoing response.

The request event is emitted each time there is an incoming request to the server. There may be multiple requests per connection.

# Incoming Message

Can view information associated with the request being made via the request parameter.

Implements the Readable Stream interface, while also implementing specific events, methods and parameters.

# Server Response

Can construct the response being sent back to the client via the response parameter.

Implements the Writable Stream interface without inheriting from Writable, and implements some additional events as well.

# http.Server

The create server function returns an instance of this server. This class inherits from the server class that exists in the network module, and like other classes that we've looked at today, adds functionality on top of what it inherits from.

# net.Server

Creates either a TCP or local server.

Inherits from Event Emitter.

# Creating a Web Server

1. Call the create function and define the request listener callback.
2. Specify the port number and address the server should be listening for.
3. Create the response headers.
4. Create the body and send the response.

# Running a Web Server

Unlike previous Node programs we've written, when we run this server code, it doesn't simply execute and return to the terminal window.

The code will run continuously, listening for incoming requests until we actively kill the process.

If any changes are made to the source code for the server that is currently running, the process will have to be killed and run again for those changes to be present.

# Request Headers

The HTTP parser built into Node will separate out the request's headers for us, and gives us an easy way to view them.

# JSON Responses

In order to send JSON responses, we have to do a bit of tweaking.

# Routing

# Routing

Mapping HTTP requests to content, such as files that exist on the server, or determining if logic has be run to provide data to send back as the response.

We need to look at what URL is passed into the request and provide different content in the response based on what the URL is asking for.

# Examining the URL

We can add endpoints to our API by examining the URL, which is made available to use by the HTTP parser as a property on the request object.

# Content and URL

Different URLs can be programmed to provide different types of content in the response.

Remember, the response object behaves as if it were a Writable stream.

# Query Parameters

If we want to send additional data as input to do additional logic a GET request, we can include that data in the url as query parameters, and examine them using Node's built in URL library.

Query parameters help with GET requests since those by definition do not include a body.

# Robust URL Handling

We need to make sure we're handling requests that come in for an endpoint we've not explicitly defined, as well as make sure that our code doesn't fall into any other successes that might happen before we end the response stream.

# Handling HTTP Methods

Like examining the url, we can inspect the request object to see what type of request is being made, and execute different code based on that type.

# Testing Web Services

# Browser

You can navigate to endpoints in your browser to test GET methods.

# curl

Command line tool that lets you build various types of requests and supply headers and data via text.

```
curl -X GET localhost:5000
curl -X POST -H 'Content-Type: application/json' -d '{"firstname" : "Sarah"}' localhost:5000/person
curl -i -X POST -H 'Content-Type: application/json' -d '{"firstname" : "Sarah"}' localhost:5000/person
```

# Postman

GUI for building requests and viewing responses.