Hello world Example(1/4)

Let's create our first application using Express.

Include this library:

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

Now we can create an application:

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

The application is a web server that will run locally on port 4000:

```
const port = 4000;
```

Let's define a wildcard route (*) with app.get() function:

```
app.get('*', function(req, res){  res.end('Hello World');  });
```

The app.get() function above accepts regular expressions of the URL patterns in a string format. In our example, we're processing all URLs with the wildcard * character.

The second parameter to the app.get() is a request handler. A typical Express.js request handler is similar to the one we pass as a callback to the native/core Node.js http.createServer() method.

Lastly, we start the Express.js web server and output a user-friendly terminal message in a callback:

```
app.listen(port, function() {
  console.log('The server is running, ' +
      ' please, open your browser at http://localhost:%s',
      port);
});
```

Hello world Example(2/4)

The full code of the index.js file

```
const express = require('express');
const app = express();
const port = 4000;
app.get('*', function(req, res){
    res.end('Hello World');
});
app.listen(port, function(){
  console.log('The server is running, ' +
      ' please, open your browser at http://localhost:%s',
      port);
});
```

To run the script, we execute node index.js from the project folder:

```
$ node index
```

Now, if you open your browser at http://localhost:4000 (same as http://127.0.0.1:4000), you should see the “Hello World” message

Hello world Example(3/4)

We can make our example a little more interactive by echoing the name that we provide to the server along with the “Hello” phrase. To do so, add the following route before the all-encompassing

route from the previous example.

```
app.get('/name/:user_name', function(req,res) {
  res.status(200);
  res.set('Content-type', 'text/html');
  res.send('<html><body>' +
  '<h1>Hello ' + req.params.user_name + '</h1>' +
  '</body></html>'
  );
});
```

Inside of the /name/:name_route route, we set the proper HTTP status code (200 means okay), HTTP response headers and wrap our dynamic text in HTML body and h1 tags.

res.send() is a special Express.js method that conveniently goes beyond what our old friend from core HTTP module res.end() does. For example, the former automatically adds a Content-Length HTTP header for us. It also augments Content-Type based on the data provided to it.

Hello world Example(4/4)

The full source code of the index.js file:

```
const express = require('express');
const app = express();
const port = 4000;
app.get('/name/:user_name', function(req,res) {
  res.status(200);
  res.set('Content-type', 'text/html');
  res.send('<html><body>' +
  '<h1>Hello ' + req.params.user_name + '</h1>' +
  '</body></html>'
  );
});

app.get('*', function(req, res){
    res.end('Hello World');
```

```
});
app.listen(port, function(){
  console.log('The server is running, ' +
      ' please, open your browser at http://localhost:%s',
      port);
});
```

After shutting down the previous server and launching the index.js script, you'll be able to see the dynamic response, e.g., by entering http://localhost:4000/name/Gomycode in your browser yields:

Previous next