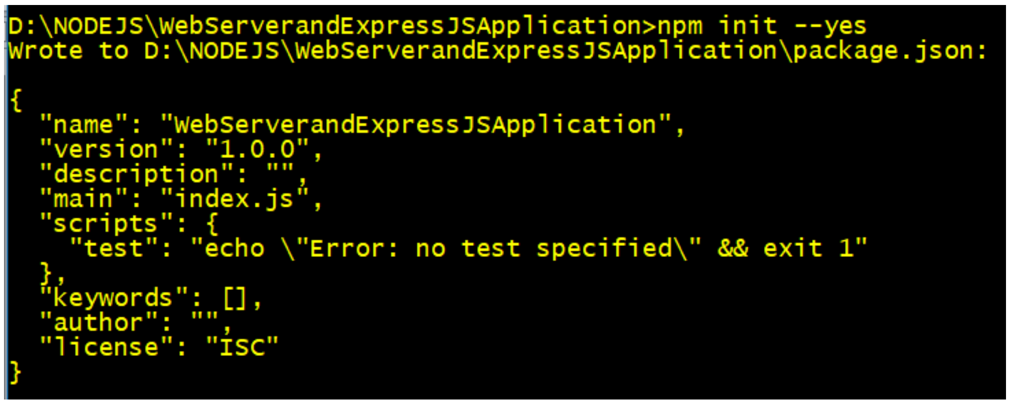
1. **Installing Express framework: https://www.npmjs.com/package/express**
2. Create a new folder named express-demo
3. Open the folder in Visual Studio Code
4. Generate **package.json** file



```
D:\NODEJS\WebServerandExpressJSApplication>npm init --yes
Wrote to D:\NODEJS\WebServerandExpressJSApplication\package.json:

{
  "name": "WebServerandExpressJSApplication",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

5. Install Express using command: **npm install express**

Building Web Server

1. Create a file named **index.js** in the application
2. Load the express module and instantiate object to handle HTTP request from the client application

```
// express module returns a function express
// express() is a top-level function exported by the express module
const express = require('express');

// invoke the function that returns an objectx
const app = express()
```

3. Register the port for express to listen requests.

```
// listen on export
app.listen(3000, () => {
    console.log('Listening on port 3000...');
});
```

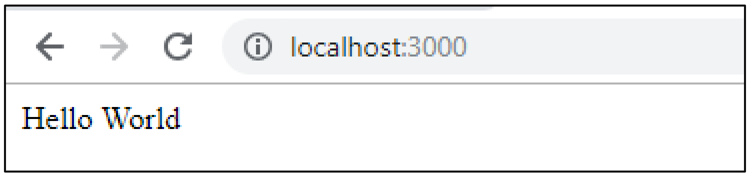
4. Write the handler for the HTTP GET request on URL '/'

5.

```
// app object has useful methods such as get(), post(), put(), delete()
// get(path or url, callback)
// callback function acts like a route handler
app.get('/', (req, res) => {
    res.send('Hello World');
});
```

6. Run index.js: **node index.js**

```
D:\NODEJS\WebServerandExpressJSApplication>node index.js
Listening on port 3000...
```

7. 

localhost:3000

Hello World

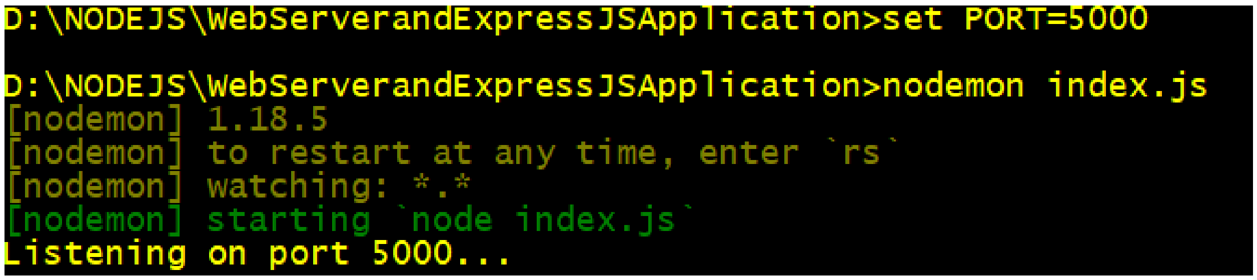
8. Add another get request for the url – **'/api/customers'** and return an array of customers back in the response

Environment Variables

1. Set the PORT environment variable to set the port dynamically.

```
// to set the port dynamically
const port = process.env.port || 3000;
app.listen(port, () => {
    console.log(`Listening on port ${port}...`);
});
```

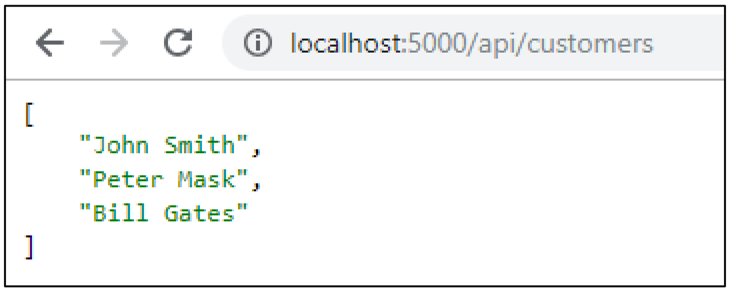
2. Now, on command prompt, we need to set an environment variable PORT using **set** command



```
D:\NODEJS\WebServerandExpressJSApplication>set PORT=5000

D:\NODEJS\WebServerandExpressJSApplication>nodemon index.js
[nodemon] 1.18.5
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node index.js`
Listening on port 5000...
```

3. Execute URL on 5000 in the Web browser



localhost:5000/api/customers

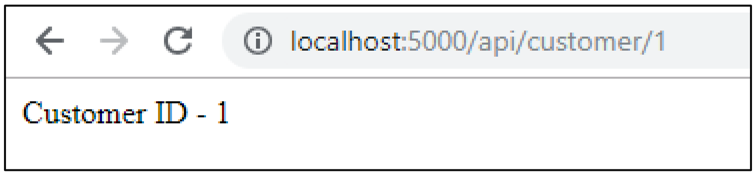
```
[
    "John Smith",
    "Peter Mask",
    "Bill Gates"
]
```

Route Parameters

1. Add the get with the route: '/api/customer/:id'

```
        app.get('/api/customer/:id', (req, res) =>{
          // to read the parameter use params property in request object
              res.send('Customer ID - ' + req.params.id);
            res.end();
        });
```

2. Check in browser

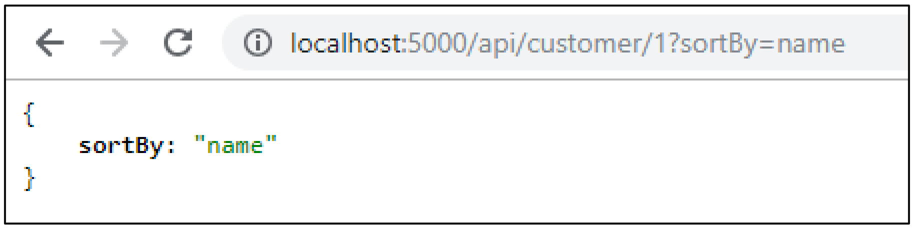


localhost:5000/api/customer/1

Customer ID - 1

3. Send multiple parameters in the GET request

4. Pass the query parameters to the request



localhost:5000/api/customer/1?sortBy=name

```
{
    sortBy: "name"
}
```

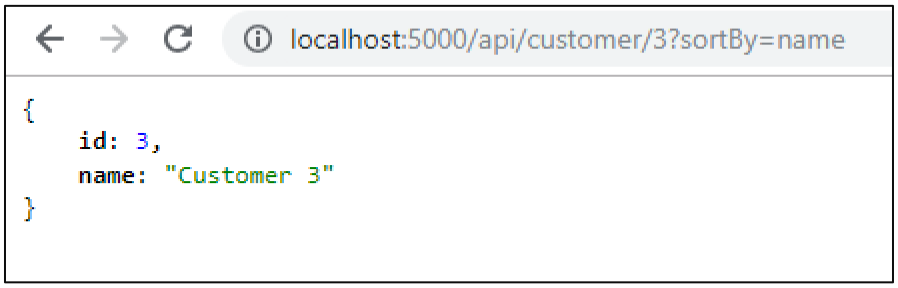
Search Based on Customer ID

1. Add customer array with few customer objects

```
 const customers = [
        {id: 1, name: 'Customer 1'},
        {id: 2, name: 'Customer 2'},
        {id: 3, name: 'Customer 3'},
 ];

app.get('/api/customer/:id', (req, res) =>{
     let customer = customers.find(c => c.id ===
                                parseInt(req.params.i  d));
   res.send(customer);
});
```

2. Check in browser



localhost:5000/api/customer/3?sortBy=name

```
{
    id: 3,
    name: "Customer 3"
}
```

Post Request

1. In index.js, use middleware to use json object

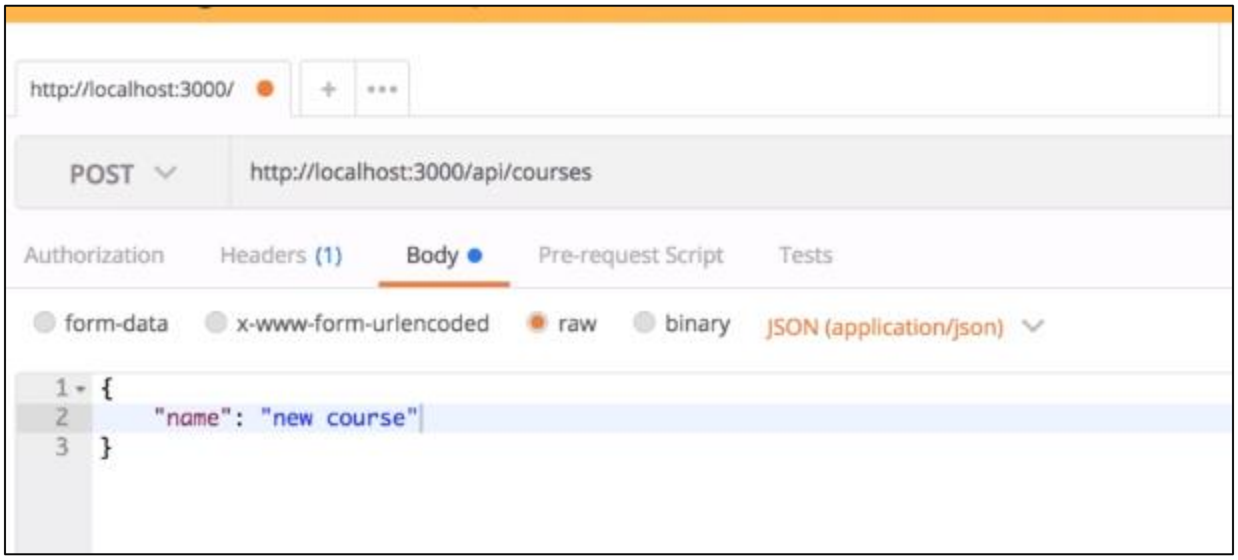
```
          // Add a middleware
          app.use(express.json());
```

Add

2. Add the post request to create a new customer resource on the server

```
// post request
app.post('/api/customer', (req, res) => {
    const customer ={
        id: customers.length + 1,
        name: req.body.name
    };
    customers.push(customer);
    res.send(customer);
});
```

3. Use postman client to check the post request



http://localhost:3000/api/courses

```
{
    "name": "new course"
}
```

Input Validation

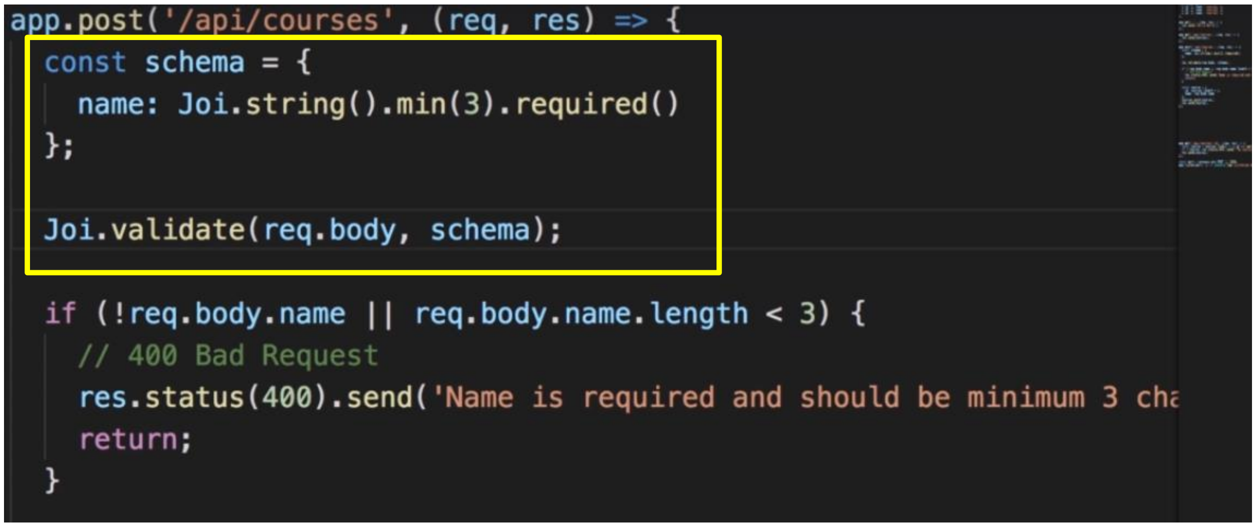
1. Input data needs to validated before server sync up the data in the services.
2. To achieve this:

```
// post request
app.post('/api/customer', (req, res) => {

    if(!req.body.name || req.name.length < 3)
    {
        // 400 Bad request
        res.status(400).send('Name is required and length needs to be greater than 3');
        return;
    }
    const customer ={
        id: customers.length + 1,
        name: req.body.name
    };
    customers.push(customer);
    res.send(customer);
});
```

3. Use joi module - Object schema description language and validator for JavaScript objects.
4. Command: npm I joi

```
// joi module returns a class
const Joi = require('joi');
```



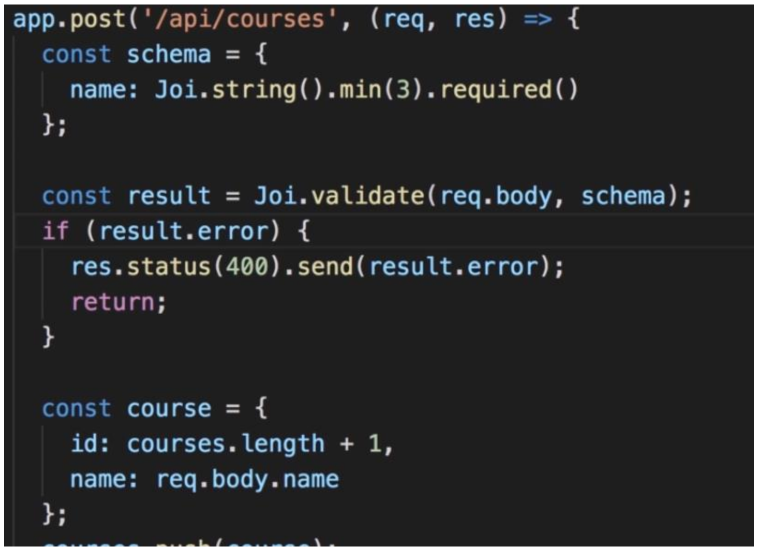
```
app.post('/api/courses', (req, res) => {
  const schema = {
    name: Joi.string().min(3).required()
  };

  Joi.validate(req.body, schema);

  if (!req.body.name || req.body.name.length < 3) {
    // 400 Bad Request
    res.status(400).send('Name is required and should be minimum 3 cha
    return;
  }
```

Now, if we pass and empty object from client, then server throws validation exception on console.

To handle error using validator:

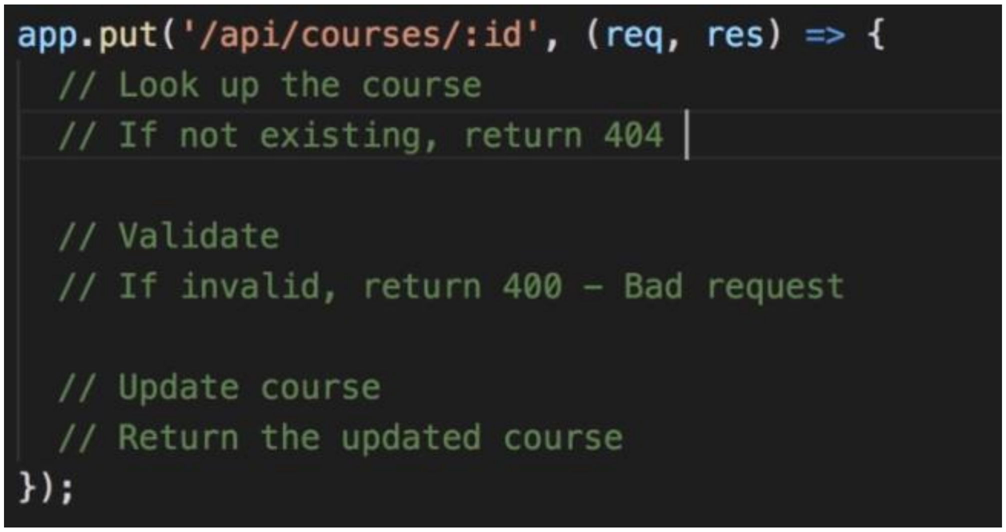


```
app.post('/api/courses', (req, res) => {
  const schema = {
    name: Joi.string().min(3).required()
  };

  const result = Joi.validate(req.body, schema);
  if (result.error) {
    res.status(400).send(result.error);
    return;
  }

  const course = {
    id: courses.length + 1,
    name: req.body.name
  };
```

PUT Request



```
app.put('/api/courses/:id', (req, res) => {
  // Look up the course
  // If not existing, return 404

  // Validate
  // If invalid, return 400 - Bad request

  // Update course
  // Return the updated course
});
```