</talentlabs>

Express Lecture 9

Express Restful API Error Handling

</talentlabs>

Agenda

- HTTP Status Code
- Demonstrate a 5xx Error
- 4xx Errors: Intro to Validation

HTTP Status Code

</talentlabs>

HTTP Status Code

A HTTP Response Message should contain a HTTP status code.

- With the HTTP status code, the client doesn't need to read the entire message to know the overall HTTP request result

HTTP defines these standard status codes that can be used to convey the results of a client's request. The status codes are divided into the five categories.

- 1xx: Informational – Communicates transfer protocol-level information.
- **2xx: Success – Indicates that the client's request was accepted successfully.**
- 3xx: Redirection – Indicates that the client must take some additional action in order to complete their request.
- **4xx: Client Error – This category of error status codes points the finger at clients.**
- **5xx: Server Error – The server takes responsibility for these error status codes.**

We have been using 2xx so far because we want to indicate the HTTP request is handled successfully.

But today we want to dive into the errors.

Client Error vs Server Error

- Missing a parameter,
- Password is used
- Wrong format
- etc.

- 4xx: Client Error – This category of error status codes **points the finger at clients**.
- 5xx: Server Error – The **server takes responsibility** for these error status codes.

- Server is down
- Server app has a bug
- etc

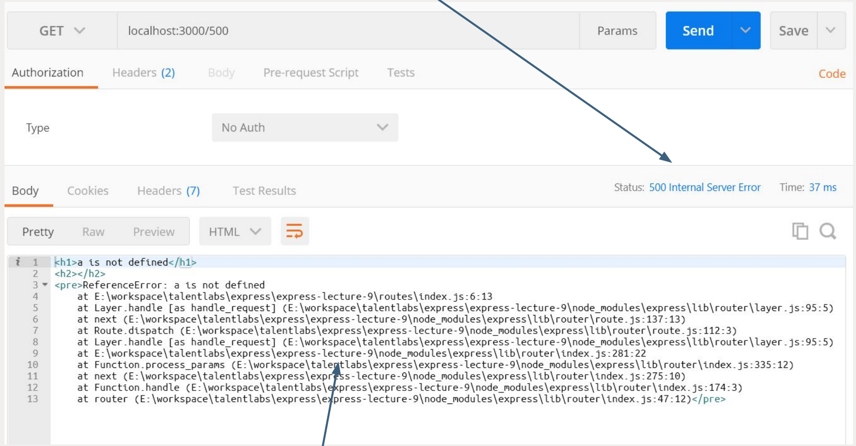
Demonstrate a 5xx Error

Server Error

We are trying to use an undefined variable “a”.

500 Error

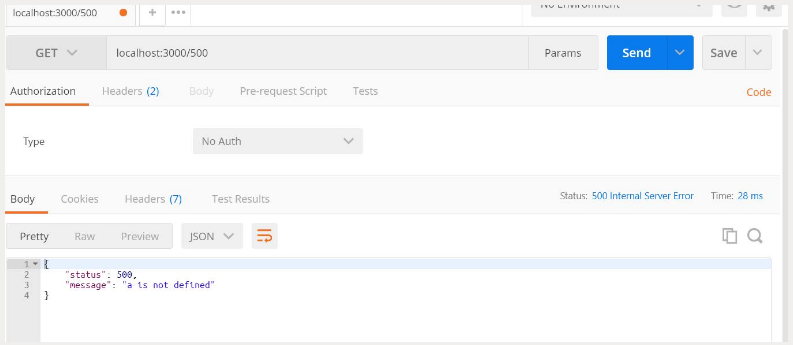
```
router.get('/500', function(req, res, next) {
  res.json({
    result: a,
  });
});
```



```
<h1>a is not defined</h1>
<h2></h2>
<pre>ReferenceError: a is not defined
    at E:\workspace\talentlabs\express\express-lecture-9\routes\index.js:6:13
    at Layer.handle [as handle_request] (E:\workspace\talentlabs\express\express-lecture-9\node_modules\express\lib\router\layer.js:95:5)
    at next (E:\workspace\talentlabs\express\express-lecture-9\node_modules\express\lib\router\route.js:137:13)
    at Route.dispatch (E:\workspace\talentlabs\express\express-lecture-9\node_modules\express\lib\router\route.js:112:3)
    at Layer.handle [as handle_request] (E:\workspace\talentlabs\express\express-lecture-9\node_modules\express\lib\router\layer.js:95:5)
    at E:\workspace\talentlabs\express\express-lecture-9\node_modules\express\lib\router\index.js:281:22
    at Function.process_params (E:\workspace\talentlabs\express\express-lecture-9\node_modules\express\lib\router\index.js:335:12)
    at next (E:\workspace\talentlabs\express\express-lecture-9\node_modules\express\lib\router\index.js:275:10)
    at Function.handle (E:\workspace\talentlabs\express\express-lecture-9\node_modules\express\lib\router\index.js:174:3)
    at router (E:\workspace\talentlabs\express\express-lecture-9\node_modules\express\lib\router\index.js:47:12)</pre>
```

It is in HTML format! Let’s make sure the Express App return JSON error.

Error Response in JSON format



localhost:3000/500

GET localhost:3000/500

Type: No Auth

Status: 500 Internal Server Error Time: 28 ms

JSON

```
{
    "status": 500,
    "message": "a is not defined"
}
```

Update the ErrorHandler in the app.js

```
// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') ===
'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});
```

```
app.use(function(err, req, res, next) {
  // render the error page
  res.status(err.status || 500);
  res.json({
    status: err.status || 500,
    message: err.message,
  });
});
```

</talentlabs>

Introduction to Validation

</talentlabs>

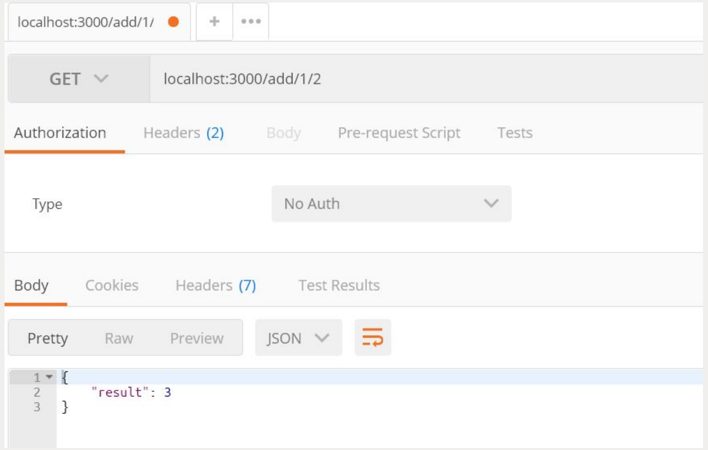
What is Validation

```
router.get("/add/:a/:b", function (req, res, next) {
  res.json({
    result: parseFloat(req.params["a"]) + parseFloat(req.params["b"]),
  });
});
```

Result is strange here because the parameters are not as expected.

We want to **validate the parameters before handling the request. It should be a client error.**

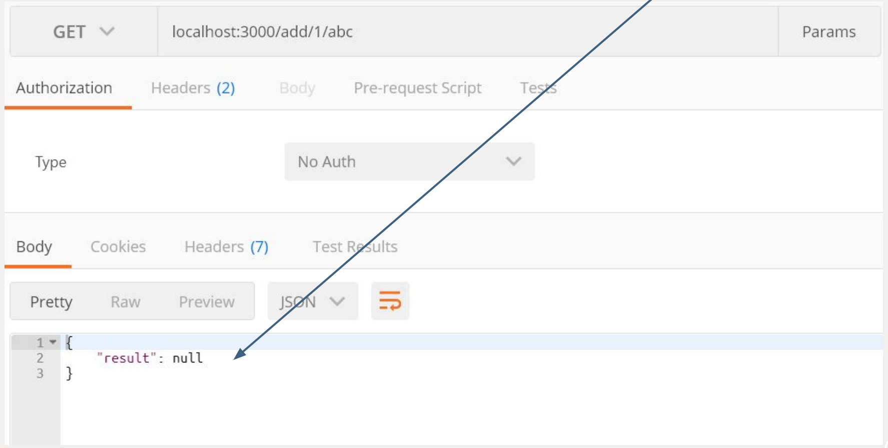
Valid parameters



GET localhost:3000/add/1/2

```
{
    "result": 3
}
```

Invalid parameters



GET localhost:3000/add/1/abc

```
{
    "result": null
}
```

Express Validator

https://express-validator.github.io/docs/

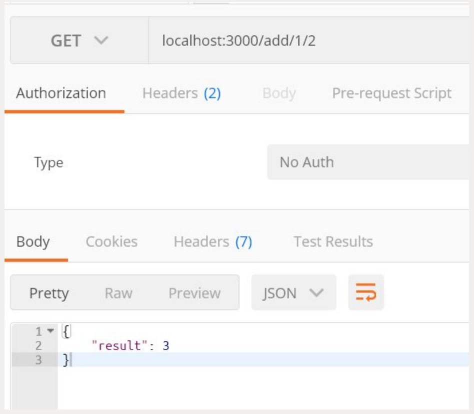
```
npm install --save express-validator
```

Express Validator

```
const { check, validationResult} = require("express-validator");

router.get("/add/:a/:b",
  check("a").isFloat(),
  check("b").isFloat(),
  function (req, res, next) {
  // Validate
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    // error response
    return res.status(400).json({ errors: errors.array() });
  }

  res.json({
    result: parseFloat(req.params["a"]) + parseFloat(req.params["b"]),
  });
});
```



GET localhost:3000/add/1/2

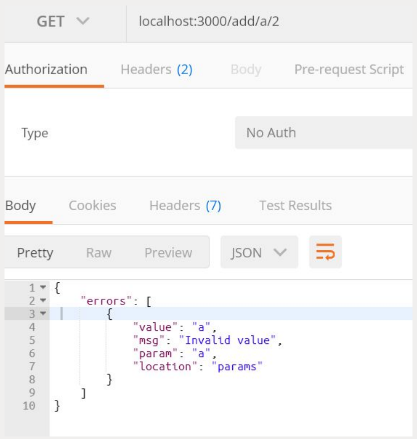
Authorization · Headers (2) · Body · Pre-request Script

Type: No Auth

Body · Cookies · Headers (7) · Test Results

Pretty · Raw · Preview · JSON

```
{
    "result": 3
}
```



GET localhost:3000/add/a/2

Authorization · Headers (2) · Body · Pre-request Script

Type: No Auth

Body · Cookies · Headers (7) · Test Results

Pretty · Raw · Preview · JSON

```
{
    "errors": [
        {
            "value": "a",
            "msg": "Invalid value",
            "param": "a",
            "location": "params"
        }
    ]
}
```

Express Validator

```
const { check, validationResult} = require("express-validator");

router.get("/add/:a/:b",
  check("a").isFloat(),
  check("b").isFloat(),
  function (req, res, next) {
  // Validate
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    // error response
    return res.status(400).json({ errors: errors.array() });
  }

  res.json({
    result: parseFloat(req.params["a"]) + parseFloat(req.params["b"]),
  });
});
```

1 Validation Rules

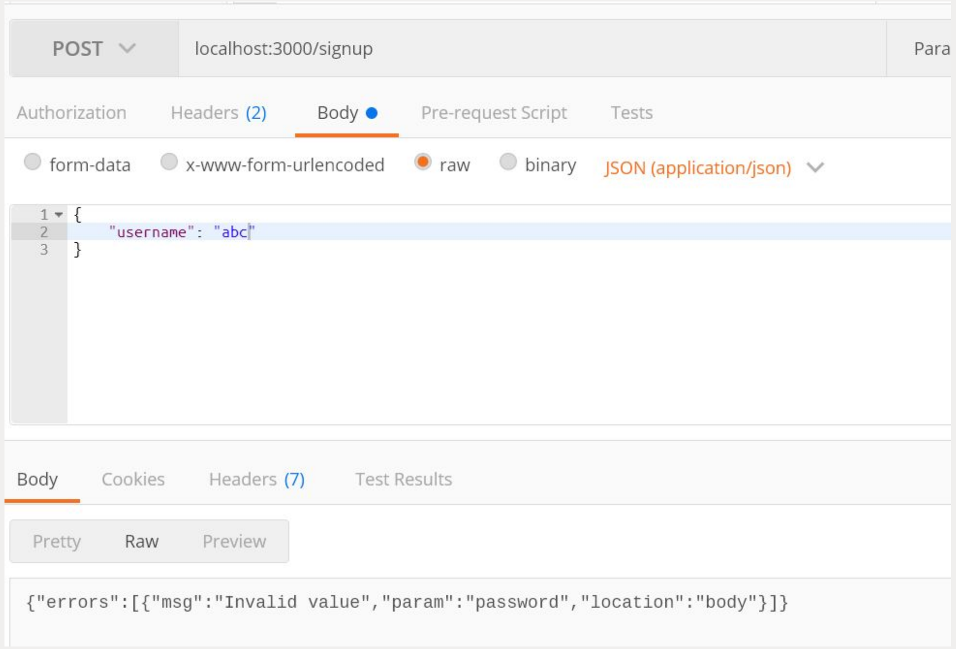
2 Validate

3 If there are validation errors, return an error response. The status code is 400 (client error)

Common Validations

Not Empty

```
router.post("/signup",
  check("username").notEmpty(),
  check("password").notEmpty(),
  function(req, res, next) {
    // Validate
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      // error response
      return res.status(400).json({ errors: errors.array() });
    }
    return res.json({})
  }
)
```



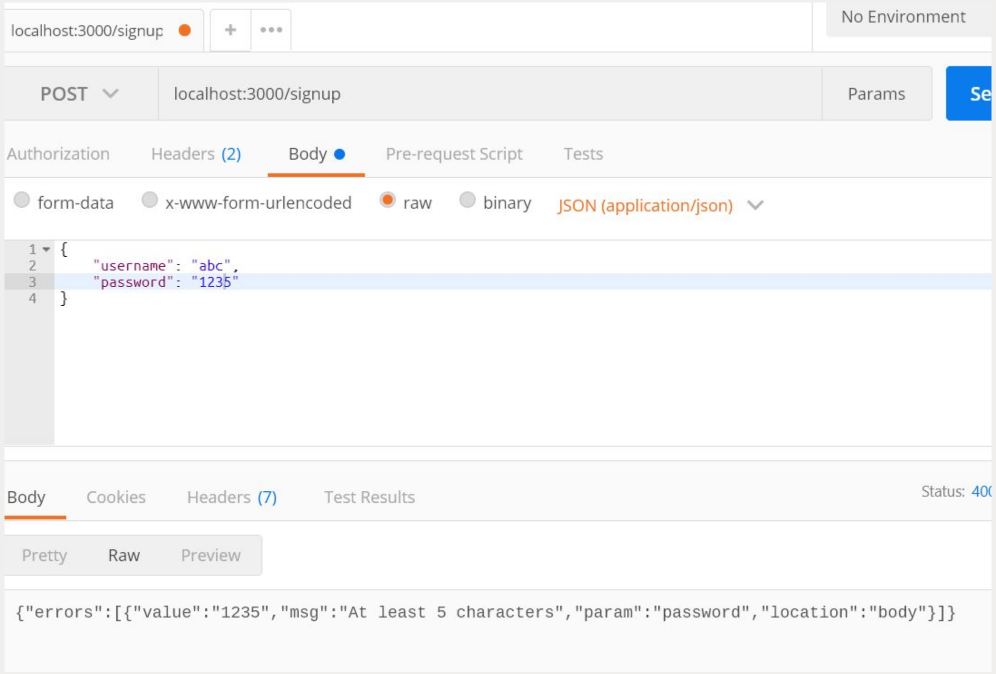
```
{
    "username": "abc"
}
```

```
{"errors":[{"msg":"Invalid value","param":"password","location":"body"}]}
```

Min Length

```
router.post(
  "/signup",
  check("username").notEmpty(),
  check("password").notEmpty(),
  check("password", "At least 5 characters").isLength({ min: 5 }),
  function (req, res, next) {
    // Validate
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      // error response
      return res.status(400).json({ errors: errors.array() });
    }

    return res.json({});
  }
);
```



POST localhost:3000/signup

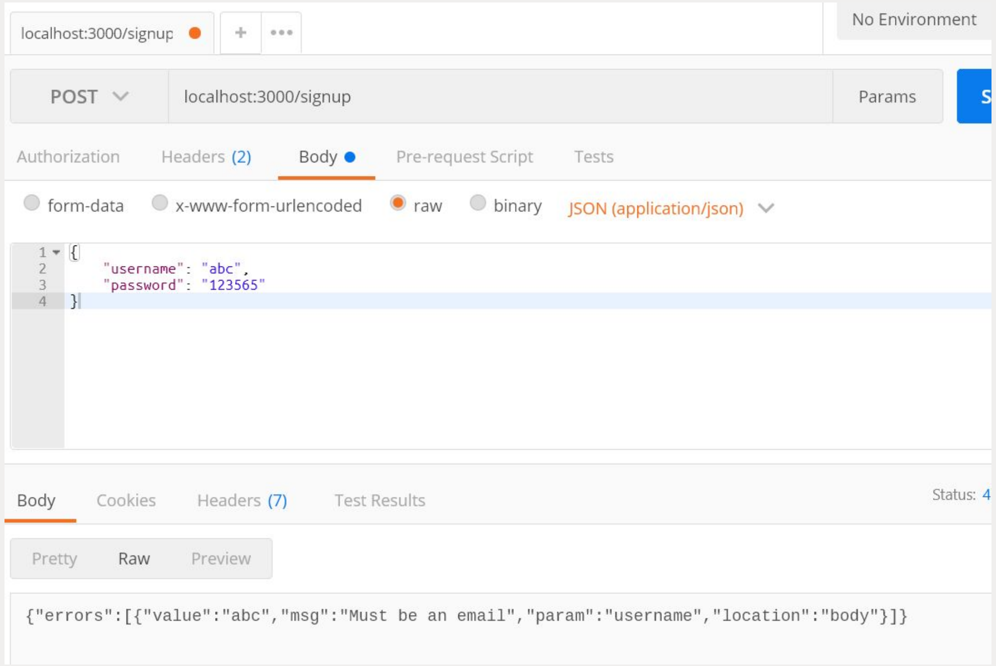
```
{
    "username": "abc",
    "password": "1235"
}
```

```
{"errors":[{"value":"1235","msg":"At least 5 characters","param":"password","location":"body"}]}
```

Email

```
router.post(
  "/signup",
  check("username").notEmpty(),
  check("username", "Must be an email").isEmail(),
  check("password").notEmpty(),
  check("password", "At least 5 characters").isLength({ min: 5 }),
  function (req, res, next) {
    // Validate
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      // error response
      return res.status(400).json({ errors: errors.array() });
    }

    return res.json({});
  }
);
```



POST localhost:3000/signup

Body: raw [✓] JSON (application/json)

```
{
    "username": "abc",
    "password": "123565"
}
```

Raw

```
{"errors":[{"value":"abc","msg":"Must be an email","param":"username","location":"body"}]}
```

Summary

- Return Errors in JSON format
- Server Error
- Client Error
    - Validation

More Validators

https://github.com/validatorjs/validator.js#validators