LESSON COMMENTS (0)

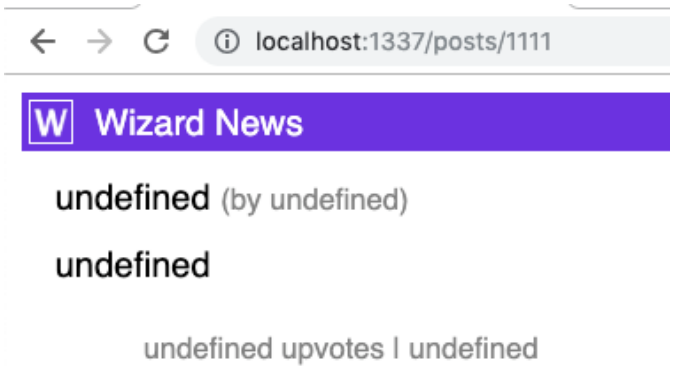
Workshop
UNIV Web - Wizard News
0%

UNIV Web - Wizard News > Route handlers

# Custom Error Handler

We're in good shape! We can list all posts. When we click on a post, it takes us to the detail view of that post.

But let's suppose that a user mistypes the URL for a post. Go to http://localhost:1337/posts/1111 and you'll see something like this:




It would be better if users were given a friendly "404 Not Found" page. We've got a couple different ways to accomplish this.

- **Option 1:** Check to see if

`postBank.find()` returned an actual post and if not, send them a Not Found page instead of the post detail HTML.

- **Option 2:** Check to see if `postBank.find()` returned an actual post and if not, *throw an error*, to be caught by an Express error handler.
- **Option 3:** Check to see if `postBank.find()` returned an actual post and if not, *create an error*, and pass that error to the `next` callback to be handled by an Express error handler.

Option 1 would look something like this:

```
app.get('/posts/:id', (req, res, ne
  const id = req.params.id
  const post = postBank.find(id)
  if (!post.id) {
    // If the post wasn't found, se
    res.status(404)
    const html = `
    <!DOCTYPE html>
    <html>
    <head>
      <title>Wizard News</title>
      <link rel="stylesheet" href="
    </head>
    <body>
      <header><img src="/logo.png"/
      <div class="not-found">
```

```
        <p>404: Page Not Found</p>
      </div>
    </body>
    </html>`
    res.send(html)
  } else {
  // ... Otherwise, send the regula
```

This approach works just fine. But it might get repetitive to have to handle errors separately for each individual route.

Option 2 would look something like this:

```
app.get('/posts/:id', (req, res) =>
  const id = req.params.id
  const post = find(id)
  if (!post.id) {
    // If the post wasn't found, ju
    throw new Error('Not Found')
  }
  // ... Otherwise, send the regula
```

Try this out yourself. You'll notice a few things:

1. The server didn't shut down. It's still listening for requests. Phew!
2. The error was logged in the terminal, including a stack trace. That's useful.

3. The error is displayed in the browser, including a stack trace. That's... not so great.

   We certainly don't want to send the server's stack traces to the browser. Not only does it make for a bad user experience, it may also pose a security vulnerabilitity.

   This is the built-in Express error handler at work. It's good to have some default error handling middleware built in to Express. Otherwise a single bad request could shut down the server. But we may want to provide our *own* error handler.

   Create an error handler, placed somewhere after all the other routes (e.g. just above `app.listen()`). It should respond with a 404 status code and some kind of friendly "Not Found" page.

   Remember that error handlers are Express middleware, much like `morgan` or `express.static`. But they're special in that they take *four* parameters: `(err, req, res, next)`. Read the docs for some guidance.

   Option 3 is particularly useful for catching asynchronous errors. We don't have any asynchronous code yet, since all of our data is stored in memory. We'll get a

chance to use that `next` callback in a future workshop.

## MOVE ON WHEN...

- [ ] Visiting a post that doesn't exist displays a friendly Not Found page.

PREVIOUS LESSON
Add a Single-Post Route

NEXT LESSON
Deploying Your App: Deployment