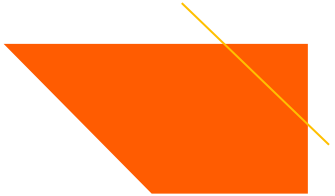
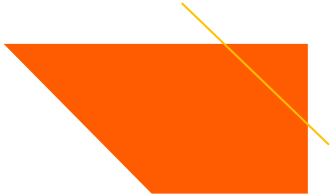
Web Application Development

COSI 152A



Main Point Preview

- In this lesson, you will learn to:
  - Set up a different start script.
  - Handle errors by logging errors and responding with error pages.
  - Set up Express.js to serve static assets.



Start Script

- The package.json file is created when you initialize a Node.js application
  - but you've changed hardly any of its values manually.
- How do you start your project?

```
node <main application file>
```

```
node index.js
```

- You can modify package.json file to set up a new start script

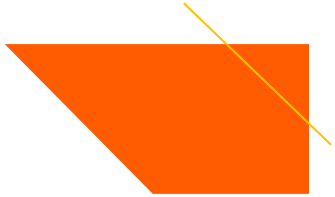
Setting Up a New Start Script

- To set up a new start script:
  - Make a copy of your express_templates application folder
  - In your package.json file, locate the scripts property.

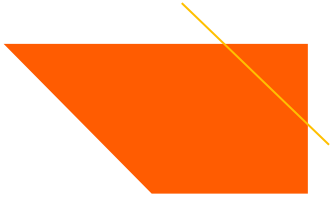
    - You should see a placeholder for a test script.
  - Add a comma to the end of that test script and add `"start": "node main.js"`
  - This script allows you to run npm start to start your application.
  - It abstracts the need to know the name of your main application file.

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node main.js"
},
```

Add a start script to package.json.

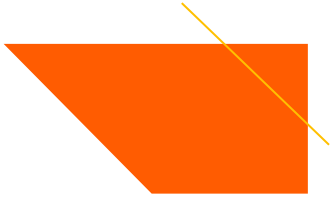


Handling Errors



Handling Errors with Express.js

- When you make a request to a path where there's no route to handle that request, you see an unfriendly Cannot GET / in your browser.
- How do you handle this?

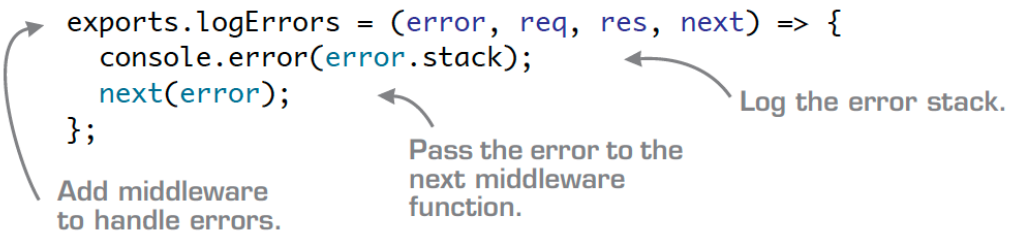


Handling Errors with Express.js

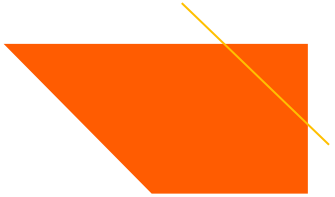
- There are few approaches to error handling with Express.js.
- The first approach is logging to your console whenever an error occurs.
  - You can log errors the same way that you logged the requested path previously.
- It is recommended that you create a new controller while handling errors

Handling Errors with Express.js

- Create errorController.js in your controllers folder, and add the function



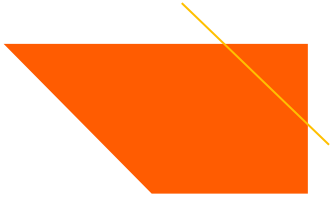
- This contains one more argument than the normal middleware function.
  - If error exists in the request-response cycle, it appears as the first argument.
  - use console.error to log the error object's stack property.



Handling Errors with Express.js

- Next, require the errorController in your main file.
- Tell Express.js to use this middleware function:

```
app.use(errorController.logErrors)
```



Handling Errors with Express.js

- Invoke an error by commenting out the line that defines the paramsName variable in the respondWithName function.
- Then, when you visit http:// localhost:3000/name/jon, your logErrors function will run.
- Remember to uncomment that line when you’re done.

Default Error Handling in Express

- Express.js handles any errors at the end of processing a request by default.
- If you want to respond with a custom message:
  - add a catch-all route at the end of your routes to respond with a:

    -404 status code if the page is not found

    -500 status code if your application got an error in the process.

Default Error Handling in Express

- That code in errorController.js should look like:

```
const httpStatus = require("http-status-codes");

exports.respondNoResourceFound = (req, res) => {
  let errorCode = httpStatus.NOT_FOUND;
  res.status(errorCode);
  res.send(`${errorCode} | The page does not exist!`);
};
exports.respondInternalError = (error, req, res, next) => {
  let errorCode = httpStatus.INTERNAL_SERVER_ERROR;
  console.log(`ERROR occurred: ${error.stack}`)
  res.status(errorCode);
  res.send(`${errorCode} | Sorry, our application is
➥experiencing a problem!`);
};
```

Respond with a 404 status code.

Catch all errors and respond with a 500 status code.

Default Error Handling in Express

- Add these middleware functions to main.js, as shown:

```
app.use(errorController.respondNoResourceFound);
app.use(errorController.respondInternalError);
```

Add error-handling middleware to main.js.

- In main.js, order matters:
  - respondNoResourceFound will catch requests made with no matching routes
  - respondInternalError will catch any requests where errors occurred.

Customized Error Pages

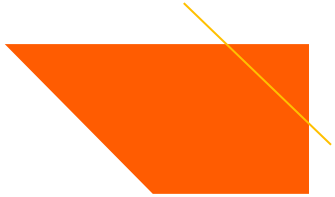
- If you want to customize your error pages, you can add a 404.html and a 500.html page in your public folder with basic HTML.
  - Instead of responding with a plain-text, you can respond with these files.
  - These files won't use your templating engine to process the response.
- The respondNoResourceFound function in error controller looks like:

```
exports.respondNoResourceFound = (req, res) => {
  let errorCode = httpStatus.NOT_FOUND;
  res.status(errorCode);
  res.sendFile(`./public/${errorCode}.html`, {
    root: "./"
  });
};
```

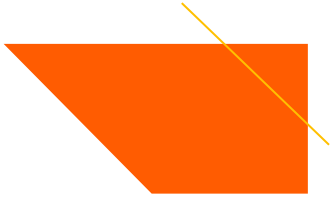
Respond with a custom error page.

Send content in 404.html.

  - res.sendFile allows you to specify an absolute path to your error page
  - it is helpful if your normal templating renderer isn't working.



Serving Static Files

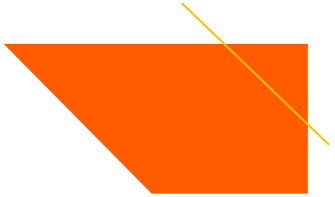


Serving Static Files

- Serving different types of static files and assets would require hundreds of lines of code.
- With Express.js, these file types are accounted for automatically.
- The only thing you need to do is tell Express.js where to find these static files:

```
app.use(express.static("public"))
```

- This line tells your application to use the public folder to serve static files.

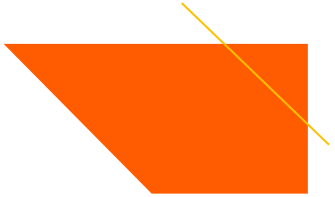


Serving Static Files

- Now you can visit http:// localhost:3000/404.html directly.

- You can also place an image or another static asset in your public folder and access it by filename after the main domain in your URL.

http:// localhost: 3000/ images/cat.jpg



Thank You!