Express

Express is an incredibly flexible framework. Express essentially sits on top of `http.createServer`, but it gives us a lot of extra flexibility to do what we want, without clogging up our code with a ton of callbacks. Below is a very basic express server:

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

const app = express()

app.use((req, res) => {
  const method = req.method;
  const route = req.url;

  console.log("Method: " + method)
  console.log("Route Path: " + route)

  const returnObject = {
    method,
    route,
    message: "You wanted to " + method + " to " + route
  }

  res.send(returnObject);
})

const port = 3000;
app.listen(port)
console.log("Now listening on port " + port);
```

In the previous lecture, we talked about basic routing, that is, we switched on the method and route:

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

const app = express()

app.use((req, res) => {
  const method = req.method;
```

```
  const route = req.url;

  if (route === "/" && method === "GET") {
    const someObject = {
      hi: "there",
      whats: "up"
    }
    res.send(someObject);
  } else if (route === "/some-path" && method === "GET") {
    const otherObject = {
      not: "much"
    }
    res.send(otherObject);
  }
  res.end("Where were you going?");
})

const port = 3000;
app.listen(port)
console.log("Now listening on port " + port);
```

While creating a server that switches on on route and method like above is completely reasonable and fine, there are better ways to do it!

Let's first start with some hard coded data to retrieve. In creating a new application, we'll have our index be a basic express app, and a couple of other files. Our project folder will look like:

```
.
├── index.js
├── office.js
├── package.json
└── parksNRec.js
```

We'll want to create a couple of data files where for now, we'll just hard code our data as JSON:

`office.js`

```
module.exports = {
  "Michael Scott": "Steve Carell",
  "Dwight Schrute": "Rainn Wilson",
  "Jim Halpert": "John Krasinski",
  "Pam Beesly": "Jenna Fischer",
```

```
  "Ryan Howard": "B.J. Novak",
  "Andy Bernard": "Ed Helms",
  "Robert California": "James Spader",
  "Stanley Hudson": "Leslie David Baker",
  "Kevin Malone": "Brian Baumgartner",
  "Meredith Palmer": "Kate Flannery",
  "Angela Martin": "Angela Kinsey",
  "Oscar Martinez": "Oscar Nunez",
  "Phyllis Lapin": "Phyllis Smith",
  "Roy Anderson": "David Denman",
  "Jan Levinson": "Melora Hardin",
  "Kelly Kapoor": "Mindy Kaling",
  "Toby Flenderson": "Paul Lieberstein",
  "Creed Bratton": "Creed Bratton",
  "Darryl Philbin": "Craig Robinson",
  "Erin Hannon": "Ellie Kemper",
  "Gabe Lewis": "Zach Woods",
  "Holly Flax": "Amy Ryan",
};
```

`parksNRect.js`

```
module.exports = {
  "Leslie Knope": "Amy Poehler",
  "Ann Perkins": "Rashida Jones",
  "Mark Brendanawicz": "Paul Schneider",
  "Tom Haverford": "Aziz Ansari",
  "Ron Swanson": "Nick Offerman",
  "April Ludgate": "Aubrey Plaza",
  "Andy Dwyer": "Chris Pratt",
  "Ben Wyatt": "Adam Scott",
  "Chris Traeger": "Rob Lowe",
  "Jerry Gergich": "Jim O'Heir",
  "Donna Meagle": "Retta",
  "Craig Middlebrooks": "Billy Eichner",
};
```

GREAT! Now we've got some data. Let's return that data with our main file, `index.js`:

```
const express = require("express");
const officeCharacters = require("./office");
const parksAndRecCharacters = require("./parksNRec");

```

```
const app = express();

app.use((req, res) => {
  res.send({
    parksAndRecCharacters,
    officeCharacters,
  });
});

const port = 3000;
app.listen(port);
console.log("Now listening on port " + port);
```

If we make a call to our server, now, we'll receive a JSON payload that has a combination of both of our files!

```
{
  "parksAndRecCharacters": {
    "Leslie Knope": "Amy Poehler",
    "Ann Perkins": "Rashida Jones",
    "Mark Brendanawicz": "Paul Schneider",
    "Tom Haverford": "Aziz Ansari",
    "Ron Swanson": "Nick Offerman",
    "April Ludgate": "Aubrey Plaza",
    "Andy Dwyer": "Chris Pratt",
    "Ben Wyatt": "Adam Scott",
    "Chris Traeger": "Rob Lowe",
    "Jerry Gergich": "Jim O'Heir",
    "Donna Meagle": "Retta",
    "Craig Middlebrooks": "Billy Eichner"
  },
  "officeCharacters": {
    "Michael Scott": "Steve Carell",
    "Dwight Schrute": "Rainn Wilson",
    "Jim Halpert": "John Krasinski",
    "Pam Beesly": "Jenna Fischer",
    "Ryan Howard": "B.J. Novak",
    "Andy Bernard": "Ed Helms",
    "Robert California": "James Spader",
    "Stanley Hudson": "Leslie David Baker",
    "Kevin Malone": "Brian Baumgartner",
    "Meredith Palmer": "Kate Flannery",
    "Angela Martin": "Angela Kinsey",
    "Oscar Martinez": "Oscar Nunez",
    "Phyllis Lapin": "Phyllis Smith",
```

```
    "Roy Anderson": "David Denman",
    "Jan Levinson": "Melora Hardin",
    "Kelly Kapoor": "Mindy Kaling",
    "Toby Flenderson": "Paul Lieberstein",
    "Creed Bratton": "Creed Bratton",
    "Darryl Philbin": "Craig Robinson",
    "Erin Hannon": "Ellie Kemper",
    "Gabe Lewis": "Zach Woods",
    "Holly Flax": "Amy Ryan"
  }
}
```

This is fun and great, but what if we wanted to have an endpoint for each show? We'd need to create a route for our `index.js`:

```
const express = require("express");
const officeCharacters = require("./office");
const parksAndRecCharacters = require("./parksNRec");

const app = express();

app.use((req, res) => {
 const route = req.method + ' ' + req.url

 if(route === 'GET /office') {
   res.send(officeCharacters)
 } else if (route === 'GET /parksAndRec'){
   res.send(parksAndRecCharacters)
 } else {
   res.send({ message: "No matching route found :( Four oh Four..."})
 }
});

const port = 3000;
app.listen(port);
console.log("Now listening on port " + port);
```

Now, if we make a request to `localhost:3000/parksAndRec`, we'll wind up with a response of only those characters:

```
{
"Leslie Knope": "Amy Poehler",
```

```
"Ann Perkins": "Rashida Jones",
"Mark Brendanawicz": "Paul Schneider",
"Tom Haverford": "Aziz Ansari",
"Ron Swanson": "Nick Offerman",
"April Ludgate": "Aubrey Plaza",
"Andy Dwyer": "Chris Pratt",
"Ben Wyatt": "Adam Scott",
"Chris Traeger": "Rob Lowe",
"Jerry Gergich": "Jim O'Heir",
"Donna Meagle": "Retta",
"Craig Middlebrooks": "Billy Eichner"
}
```

A More Efficient (and Cleaner) Way!

Instead of just using res.send inside of our if/else, let's pull that out, and write these as functions:

```
const express = require("express");
const officeCharacters = require("./office");
const parksAndRecCharacters = require("./parksNRec");

const app = express();

const getOfficeCharacters = (req,res) => {
  res.send(officeCharacters)
}

const getParksAndRecCharacters = (req,res) => {
  res.send(parksAndRecCharacters)
}

app.use((req, res) => {
 const route = req.method + ' ' + req.url

 if(route === 'GET /office') {
  getOfficeCharacters(req,res)
 } else if (route === 'GET /parksAndRec'){
  getParksAndRecCharacters(req,res)
 } else {
   res.send({ message: "No matching route found :( Four oh Four..."})
 }
});
```

```
26
27 const port = 3000;
28 app.listen(port);
29 console.log("Now listening on port " + port);
```

So, now we've got our functions, what do we do with the if/else? We'll create a dictionary to match on the route as its keys, and then use the methods we created as values!

`index.js`

```
1  const express = require("express");
2  const officeCharacters = require("./office");
3  const parksAndRecCharacters = require("./parksNRec");
4
5  const app = express();
6
7  const getOfficeCharacters = (req, res) => {
8    res.send(officeCharacters);
9  };
10
11 const getParksAndRecCharacters = (req, res) => {
12   res.send(parksAndRecCharacters);
13 };
14
15 const routes = {
16   "GET /office": getOfficeCharacters,
17   "GET /parksAndRec": getParksAndRecCharacters,
18 };
19
20 app.use((req, res) => {
21   const requestedRoute = req.method + " " + req.url;
22   const routeHandler = routes[requestedRoute];
23
24   if (routeHandler !== undefined) {
25     routeHandler(req, res);
26   } else {
27     res.send({ message: "No matching route found :( Four oh Four..." });
28   }
29 });
30
31 const port = 3000;
32 app.listen(port);
33 console.log("Now listening on port " + port);
34
```

What did we do? We used a dictionary construct to key value pairing where the route and method are the key, while the value is the desired response method. If that method's key value pairing exists, then we can invoked it like at line 25, otherwise, we response with a route not found, which we also write a special function for:

```
1  const express = require("express");
2  const officeCharacters = require("./office");
3  const parksAndRecCharacters = require("./parksNRec");
4
5  const app = express();
6
7  const getOfficeCharacters = (req, res) => {
8    res.send(officeCharacters);
9  };
10
11 const getParksAndRecCharacters = (req, res) => {
12   res.send(parksAndRecCharacters);
13 };
14
15 const routes = {
16   "GET /office": getOfficeCharacters,
17   "GET /parksAndRec": getParksAndRecCharacters,
18 };
19
20 const fourOhFour = (req, res) => {
21   res.send({
22     status: 404,
23     message: "No matching route found :( Four oh Four...",
24   });
25 };
26
27 app.use((req, res) => {
28   const requestedRoute = req.method + " " + req.url;
29   const routeHandler = routes[requestedRoute] || fourOhFour;
30
31   if (routeHandler !== undefined) {
32     routeHandler(req, res);
33   } else {
34     res.send({ message: "No matching route found :( Four oh Four..." });
35   }
36 });
37
38 const port = 3000;
39 app.listen(port);
40 console.log("Now listening on port " + port);
```

Express definitely understands what's going on, so let's extract our callback at line 27 to its own function and pass that into `app.use` instead:

```
1  const express = require("express");
2  const officeCharacters = require("./office");
3  const parksAndRecCharacters = require("./parksNRec");
4
5  const app = express();
6
7  const getOfficeCharacters = (req, res) => {
8    res.send(officeCharacters);
9  };
10
11 const getParksAndRecCharacters = (req, res) => {
12   res.send(parksAndRecCharacters);
13 };
14
15 const routes = {
16   "GET /office": getOfficeCharacters,
17   "GET /parksAndRec": getParksAndRecCharacters,
18 };
19
20 const fourOhFour = (req, res) => {
21   res.send({
22     status: 404,
23     message: "No matching route found :( Four oh Four...",
24   });
25 };
26
27
28 const router = (req, res) => {
29   const requestedRoute = req.method + " " + req.url;
30   const routeHandler = routes[requestedRoute] || fourOhFour;
31
32   if (routeHandler !== undefined) {
33     routeHandler(req, res);
34   } else {
35     res.send({ message: "No matching route found :( Four oh Four..." });
36   }
37 }
38
39 app.use(router);
40
41 const port = 3000;
42 app.listen(port);
43 console.log("Now listening on port " + port);
```

An Even Cleaner Way: Express Router

Ultimately speaking, this is all routing is! It's a relatively simple idea but it can become pretty complex pretty quickly! Luckily for us, even though we've spent a lot of time writing all of this code, Express has a feature that does all of this for us! It's called `express.Router`.

Express router lets us create routes with a single line:

```
const express = require("express");
const officeCharacters = require("./office");
const parksAndRecCharacters = require("./parksNRec");

const app = express();

const getOfficeCharacters = (req, res) => {
  res.send(officeCharacters);
};

const getParksAndRecCharacters = (req, res) => {
  res.send(parksAndRecCharacters);
};

const fourOhFour = (req, res) => {
  res.send({
    status: 404,
    message: "No matching route found :( Four oh Four...",
  });
};

const router = express.Router();
router.get("/office", getOfficeCharacters);
router.get("/parksAndRec", getParksAndRecCharacters);

app.use(router);

const port = 3000;
app.listen(port);
console.log("Now listening on port " + port);
```

Now, instead of having a more difficult construction with creating routes and a lookup dictionary, all we need to do is use `router.<method>`, and pass in our path along with the function that we want to response with!

Query Parameters?

Routing is great! But what about query parameters? We all know that a very typical construction of a website allows for query parameters, so how do we work with those in our routes? Let's create another route to get a specific actor from the office by their character name:

```
const express = require("express");
const officeCharacters = require("./office");
const parksAndRecCharacters = require("./parksNRec");

const app = express();

const getOfficeCharacters = (req, res) => {
  res.send(officeCharacters);
};

const getOfficeCharacter = (req, res) => {
  const characterName = req.params.characterName;
  res.send({
    characterName: characterName,
    actorName: officeCharacters[characterName],
  });
};

const getParksAndRecCharacters = (req, res) => {
  res.send(parksAndRecCharacters);
};

const fourOhFour = (req, res) => {
  res.send({
    status: 404,
    message: "No matching route found :( Four oh Four...",
  });
};

const router = express.Router();
router.get("/office", getOfficeCharacters);
router.get("/office/:characterName", getOfficeCharacter);
router.get("/parksAndRec", getParksAndRecCharacters);

```

```
35 app.use(router);
36
37 const port = 3000;
38 app.listen(port);
39 console.log("Now listening on port " + port);
40
```

What we did above at line 32 is state that along the path of `/office` we will have someone pass in a parameter, which we can then access as `req.params.whateverWeNamedOurQueryParameter`. Let's do the same for parks and rec:

```
1  const express = require("express");
2  const officeCharacters = require("./office");
3  const parksAndRecCharacters = require("./parksNRec");
4
5  const app = express();
6
7  const getOfficeCharacters = (req, res) => {
8    res.send(officeCharacters);
9  };
10
11 const getOfficeCharacter = (req, res) => {
12   const characterName = req.params.characterName;
13   res.send({
14     characterName: characterName,
15     actorName: officeCharacters[characterName],
16   });
17 };
18
19 const getParksAndRecCharacters = (req, res) => {
20   res.send(parksAndRecCharacters);
21 };
22
23 const getParksAndRecCharacter = (req, res) => {
24   const characterName = req.params.characterName;
25   res.send({
26     characterName: characterName,
27     actorName: parksAndRecCharacters[characterName],
28   });
29 };
30
31 const fourOhFour = (req, res) => {
32   res.send({
33     status: 404,
34     message: "No matching route found :( Four oh Four...",
```

```
  });
};

const router = express.Router();
router.get("/office", getOfficeCharacters);
router.get("/office/:characterName", getOfficeCharacter);
router.get("/parksAndRec", getParksAndRecCharacters);
router.get("/parksAndRec/:characterName", getParksAndRecCharacter);

app.use(router);

const port = 3000;
app.listen(port);
console.log("Now listening on port " + port);

```

Cleaning Up: Using Routers to Route Routers

Our file's beginning to look a little big. First, instead of having to type `/office` or `/parksAndRec` before every single route that we make, we can create a router for each (and assign those routers to a route!):

```
const express = require("express");
const officeCharacters = require("./office");
const parksAndRecCharacters = require("./parksNRec");

const app = express();

const getOfficeCharacters = (req, res) => {
  res.send(officeCharacters);
};

const getOfficeCharacter = (req, res) => {
  const characterName = req.params.characterName;
  res.send({
    characterName: characterName,
    actorName: officeCharacters[characterName],
  });
};

const getParksAndRecCharacters = (req, res) => {
  res.send(parksAndRecCharacters);
```

```
};

const getParksAndRecCharacter = (req, res) => {
  const characterName = req.params.characterName;
  res.send({
    characterName: characterName,
    actorName: parksAndRecCharacters[characterName],
  });
};

const fourOhFour = (req, res) => {
  res.send({
    status: 404,
    message: "No matching route found :( Four oh Four...",
  });
};

const officeRouter = express.Router();
const parksAndRecRouter = express.Router();

officeRouter.get("/", getOfficeCharacters);
officeRouter.get(":characterName", getOfficeCharacter);
parksAndRecRouter.get("/", getParksAndRecCharacters);
parksAndRecRouter.get(":characterName", getParksAndRecCharacter);

app.use("/office", officeRouter);
app.use("/parksAndRec", parksAndRecRouter);

const port = 3000;
app.listen(port);
console.log("Now listening on port " + port);

```

Now that we've got our routes set up, let's set up some special files for each of our shows:

```
.
├── index.js
├── package.json
└── routes
    ├── office
    |   ├── office.js
    |   └── officeRoute.js
    └── parksAndRec
        ├── parksAndRecRoute.js
        └── parksNRec.js
```

The `office.js` and `parksNRect.js` files can just be moved entirely. We don't need to do anything special with them. We DO however have to extract our routing into their own separate files:

`officeRoute.js`

```
const express = require("express");
const officeCharacters = require("./office");

const getOfficeCharacters = (req, res) => {
  res.send(officeCharacters);
};

const getOfficeCharacter = (req, res) => {
  const characterName = req.params.characterName;
  res.send({
    characterName: characterName,
    actorName: officeCharacters[characterName],
  });
};

const officeRouter = express.Router();

officeRouter.get("/", getOfficeCharacters);
officeRouter.get("/:characterName", getOfficeCharacter);

module.exports = officeRouter;
```

`parksAndRecRoute.js`

```
const express = require("express");
const parksAndRecCharacters = require("./parksNRec");

const getParksAndRecCharacters = (req, res) => {
  res.send(parksAndRecCharacters);
```

```
};

const getParksAndRecCharacter = (req, res) => {
  const characterName = req.params.characterName;
  res.send({
    characterName: characterName,
    actorName: parksAndRecCharacters[characterName],
  });
};

const parksAndRecRouter = express.Router();

parksAndRecRouter.get("/", getParksAndRecCharacters);
parksAndRecRouter.get("/:characterName", getParksAndRecCharacter);

module.exports = parksAndRecRouter;

```

And finally, we can clean up our extremely dirty `index.js` to be incredibly clean:

```
const express = require("express");
const officeRouter = require("./routes/office/officeRoute");
const parksAndRecRouter = require("./routes/parksAndRec/parksAndRecRoute");

const app = express();

app.use("/office", officeRouter);
app.use("/parksAndRec", parksAndRecRouter);

const port = 3000;
app.listen(port);
console.log("Now listening on port " + port);

```

When you keep your routes separate like this, you only need to create more files, and you won't find yourself with 300 line routing files!

Posting Data with a Body:

Now that we have a relatively clean application to work with, let's try to create an endpoint to grab more than one character! Before we go any further, what we'll have to do is build our own special parser. The reason we have to do that is because the body of the request itself is a stream object. Let's see this data come in for `parksAndRecRoute.js`:

```
...

const getMultipleCharacters = (req, res) => {
  const characterNames = req.body;

  req.on('data', () => console.log("We're receiving data!"))
  req.on('end', () => console.log("We got all the data!"))

  console.log(req);
};

...
```

What's happening is that we're sending our data one chunk at a time. That doesn't really help us much, since we need the whole body to work with it. Let's write a body parser and put it in its new spot!

```
.
├── index.js
├── lib
|   └── bodyParser.js
├── package-lock.json
├── package.json
└── routes
    ├── office
    |   ├── office.js
    |   └── officeRoute.js
    └── parksAndRec
        ├── parksAndRecRoute.js
        └── parksNRec.js
```

`bodyParser.js`

```
const bodyParser = (req) =>
  new Promise((resolve) => {
    let chunks = [];
    req.on("data", (chunk) => {
      console.log("Got chunk: ", chunk.toString());
      chunks.push(chunk);
```

```
    });
    req.on("end", () => {
      console.log("got everything!");
      resolve(Buffer.concat(chunks));
    });
  });

module.exports = bodyParser;

```

In our `parksAndRecRoute.js` file, we can now user our body parser and use our body to find what we want! So let's bring that into our parks and rec:

```
const bodyParser = require("../../lib/bodyParser");
const express = require("express");
const parksAndRecCharacters = require("./parksNRec");

const getParksAndRecCharacters = (req, res) => {
  res.send(parksAndRecCharacters);
};

const getParksAndRecCharacter = (req, res) => {
  const characterName = req.params.characterName;
  res.send({
    characterName: characterName,
    actorName: parksAndRecCharacters[characterName],
  });
};

const getMultipleCharacters = async (req, res) => {
  console.log("Getting body: ");

  const body = await bodyParser(req);
  const parsedBody = JSON.parse(body);

  console.log("God Body", parsedBody);

  const response = parsedBody.map((characterName) => ({
    character: characterName,
    actor: parksAndRecCharacters[characterName],
  }));

  res.send(response);
};
```

```
32
33 const parksAndRecRouter = express.Router();
34
35 parksAndRecRouter.get("/", getParksAndRecCharacters);
36 parksAndRecRouter.get("/:characterName", getParksAndRecCharacter);
37 parksAndRecRouter.post("/", getMultipleCharacters);
38
39 module.exports = parksAndRecRouter;
40
```

Now, when we send a body in a post (or any other HTTP method with a body), we wind up getting to work with the material inside the actual files, as opposed to a readable stream! Try sending a `POST` request to `localhost:3001/parksAndRec/`:

```
[
  "Leslie Knope",
  "Tom Haverford"
]
```

Try doing the same for the office!