

# Build an API!

Jay Hilton  
Joyce Yoon

frontend



backend



weblab staff  
starting to teach backend

students learning  
about async and react



# Making our endpoints in our server

```
// import libraries needed for the webserver to work!
const express = require("express"); // backend framework for our node server.
const path = require("path"); // provide utilities for working with file and directory paths

// create a new express server
const app = express();

// allow us to make post requests
app.use(express.json());
```

```
app.get("/dog/test", (req, res) => {
  res.send({ message: "Wow I made my first dog endpoint!" });
});
```

/dog

```
app.get("/dog/breeds", (req, res) => {
  res.send({ breeds: ["lab", "dalmation", "corgi"] });
});
```

```
app.get("/cat/test", (req, res) => {
  res.send({ message: "Wow I made my first cat endpoint!" });
});
```

/cat

```
app.get("/cat/breeds", (req, res) => {
  res.send({ breeds: ["siamese", "maine coon", "tabby"] });
});
```

```
//many more endpoints below
```

Imagine our server had many many endpoints...

Is there a clear way to organize our endpoints?

Organize by /dog and /cat

# Making our endpoints in our server

## app


Represents your overall server  
(main-application)

## router

Isolated group of API endpoints  
(mini-application)

### app in server.js

```
...  
get /dog/test endpoint  
get /dog/breeds endpoint  
get /cat/test endpoint  
get /cat/breeds endpoint  
...
```



Having all our  
endpoints in our  
server can be  
cluttered

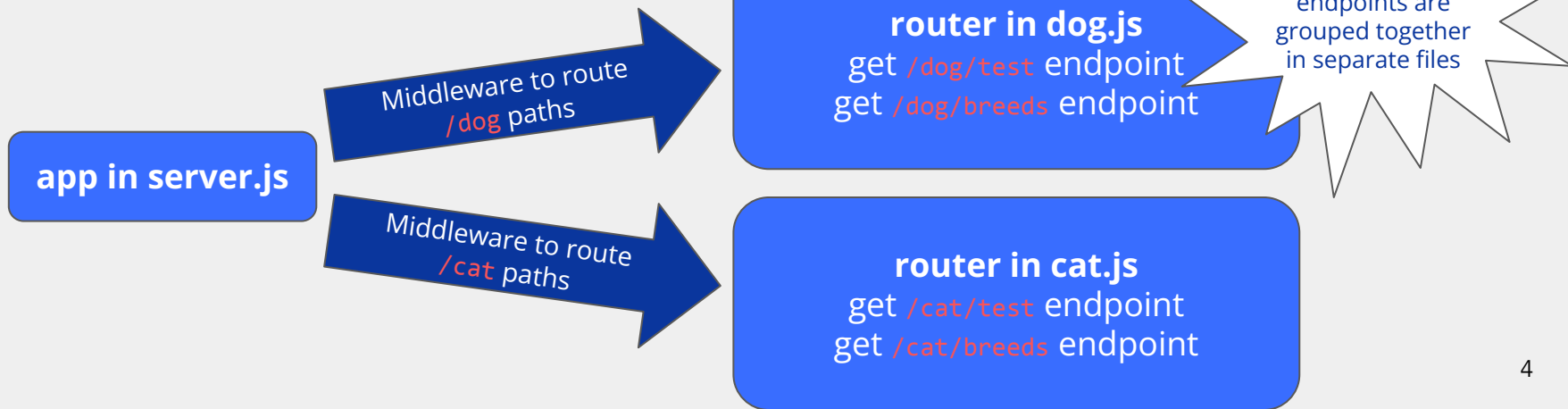
# Making our endpoints in our router

## app

Represents your overall server  
(main-application)

## router

Isolated group of API endpoints  
(mini-application)



# App vs. Router

**app** (`server.js`)

Represents your overall server  
(main-application)

**router** (`cat.js`, `dog.js`, `api.js`, etc.)

Isolated group of API endpoints  
(mini-application)

## Why use routers?

- **Simplicity!** De-clutter our server and organize our endpoints
- **Modularity!** We can structure endpoints in isolation without having to care about the logic of the server or other endpoints

# Some more words

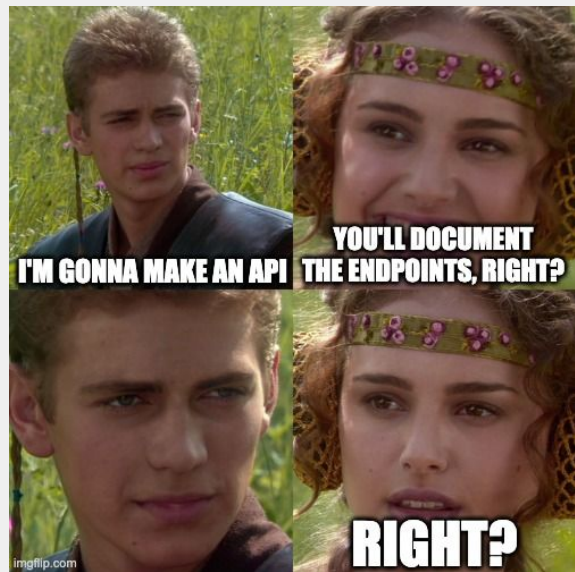
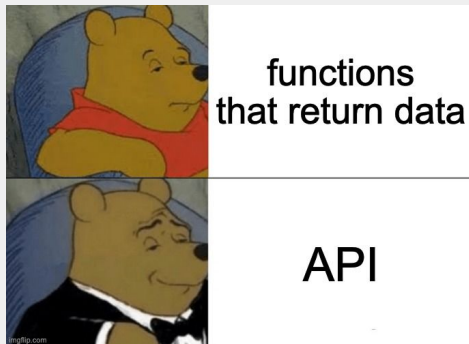
API endpoint - a part of the server that performs some specific function

Example: the stories endpoint

API route - the name you use to access that endpoint

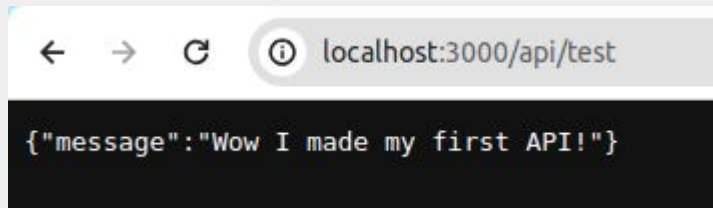
Example: `"/api/stories/"`

These are equivalent for our purposes



# Workshop 5

So far, your Catbook has a single accessible API endpoint: /api/test



Now it's time we lay the groundwork for more useful endpoints, ones that can dynamically store and retrieve data!

# Let's get started...

```
git reset --hard
```

```
git fetch
```

```
git checkout w5-starter
```



```
git reset --hard  
git fetch  
git checkout w5-starter
```

## Step 1

Move endpoints from our  
server to our router

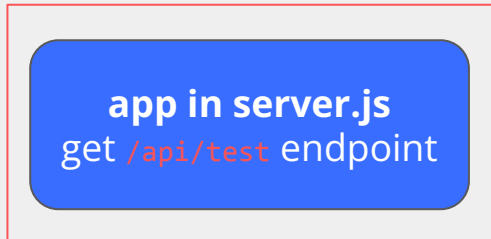
# Moving our endpoints from our server to our router

## app (server.js)

Represents your overall server  
(main-application)

## router (api.js)

Isolated group of API endpoints  
(mini-application)



What we start with in starter

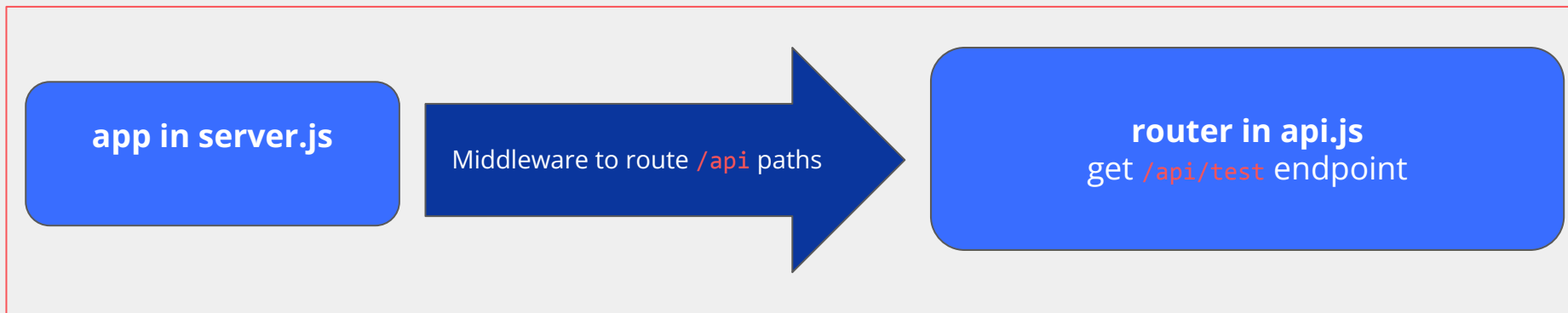
# Moving our endpoints from our server to our router

app (server.js)

Represents your overall server  
(main-application)

router (api.js)

Isolated group of API endpoints  
(mini-application)



What we want by the end of Step 1

# Moving our endpoints from our server to our router

Steps:

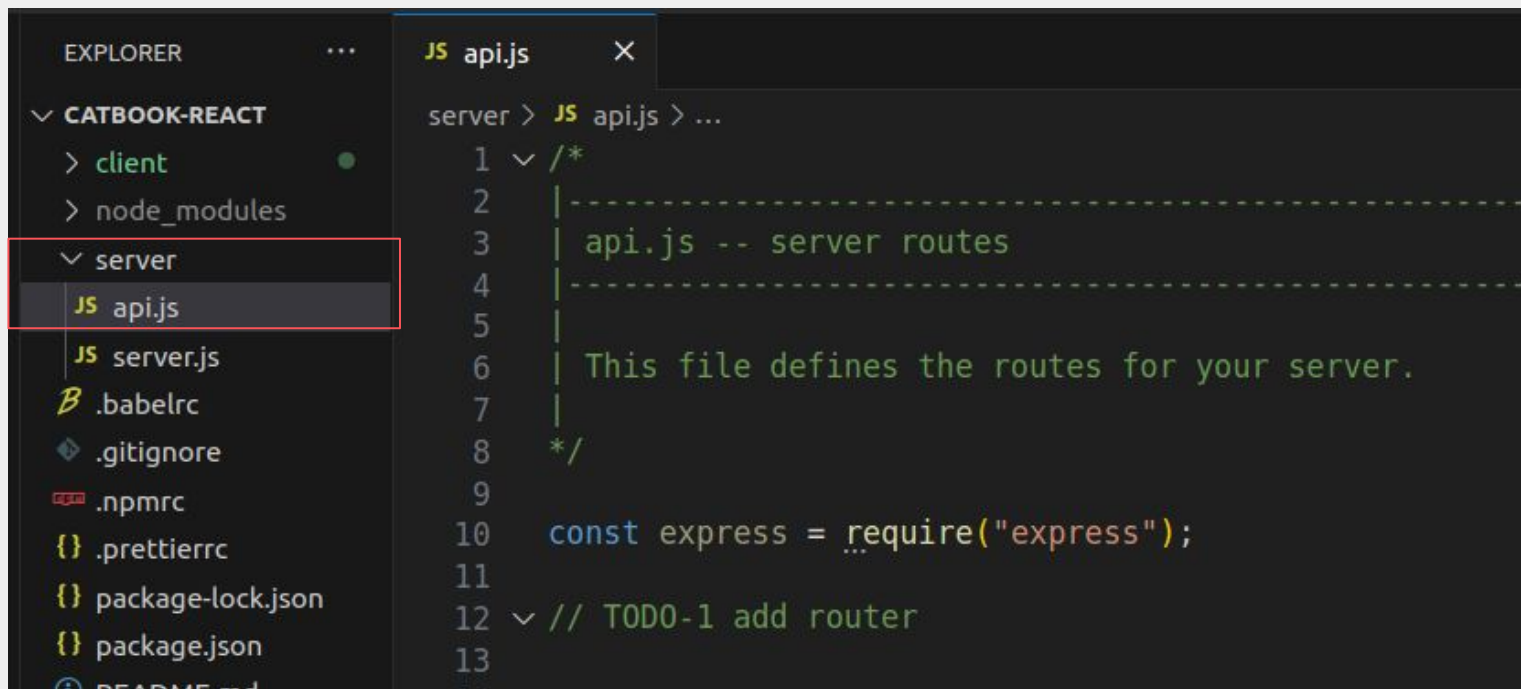
1. Create a separate file (we have already done this by making **api.js**)

app in server.js  
get **/api/test** endpoint

api.js

# Use **api** Route for Requests

Open `api.js` from the `./server` directory.



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure for 'CATBOOK-REACT'. The 'server' directory is expanded, and 'api.js' is selected, highlighted with a red rectangle. The main editor area shows the content of 'api.js', which includes a JSDoc comment and a code snippet for setting up Express.js.

```
server > JS api.js > ...
1  ▾ /*
2    |-----
3    |  api.js -- server routes
4    |-----
5    |
6    |  This file defines the routes for your server.
7    |
8    */
9
10  const express = require("express");
11
12  ▾ // TODO-1 add router
13
```

# Moving our endpoints from our server to our router

Steps:

1. Create a separate file (we have already done this by making `api.js`)
2. **Define a router for `/api` in `api.js`**

**app in server.js**  
get `/api/test` endpoint

**router in api.js**

# TODO-1&2: Connect `api.js` to our main server

In `api.js`:

```
JS server.js M JS api.js M X
server > JS api.js > ...
1  /*
2  |-----
3  | api.js -- server routes
4  |-----
5  |
6  | This file defines the routes for your server.
7  |
8  */
9
10 const express = require("express");
11
12 // TODO-1 add router
13
14
15 // TODO-5 migrate api/test endpoint from server.js
16
17
18 // TODO-2 export router
19
20
```

Reminder:  
Ever confused how we know what to code?  
Look up the documentation:  
Ex. [expressjs.com/en/guide/routing.html](https://expressjs.com/en/guide/routing.html)

# TODO-1&2: Connect `api.js` to our main server

In `api.js`:

```
JS server.js M JS api.js M X
server > JS api.js > ...
1  /*
2  |-----
3  | api.js -- server routes
4  |-----
5  |
6  | This file defines the routes for your server.
7  |
8  */
9
10 const express = require("express");
11
12 // TODO-1 add router
13 const router = express.Router();
14
15 // TODO-5 migrate api/test endpoint from server.js
16
17
18 // TODO-2 export router
19 module.exports = router;
20
```

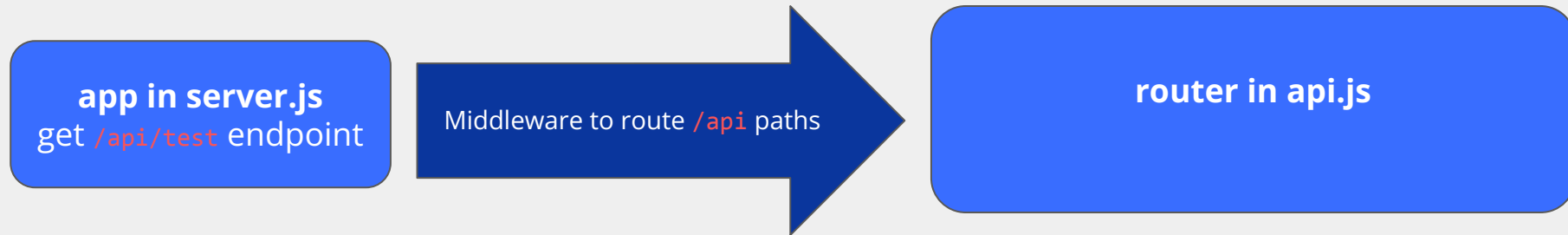
Reminder:  
Ever confused how we know what to code?  
Look up the documentation:  
Ex. [expressjs.com/en/guide/routing.html](https://expressjs.com/en/guide/routing.html)



# Moving our endpoints from our server to our router

Steps:

1. Create a separate file (we have already done this by making `api.js`)
2. Define a router for `/api` (use `express.Router()`) in `api.js`
3. **Define middleware to route any api paths prefixed with `/api` to `api.js`**



## TODO-3: Connect `api.js` to our main server

In `server.js`:

```
server > JS server.js > ...
```

```
14
15 // import libraries needed for the webserver to work!
16 const express = require("express"); // backend framework for our node server.
17 const path = require("path"); // provide utilities for working with file and directory paths
18
19 // TODO-3 import the router from our API file
20
21
22 // create a new express server
23 const app = express();
24
```

## TODO-3: Connect `api.js` to our main server

In `server.js`:

```
server > JS server.js > ...
```

```
14
15 // import libraries needed for the webserver to work!
16 const express = require("express"); // backend framework for our node server.
17 const path = require("path"); // provide utilities for working with file and directory paths
18
19 // TODO-3 import the router from our API file
20 const api = require("./api.js");
21
22 // create a new express server
23 const app = express();
24
```

## TODO-4: Connect `api.js` to our main server

In `server.js`:

```
19 // import the router from our API file
20 const api = require("./api.js");
21
22 // create a new express server
23 const app = express();
24
25 // allow us to make post requests
26 app.use(express.json());
27
28 // connect API routes from api.js
29
```

## TODO-4: Connect `api.js` to our main server

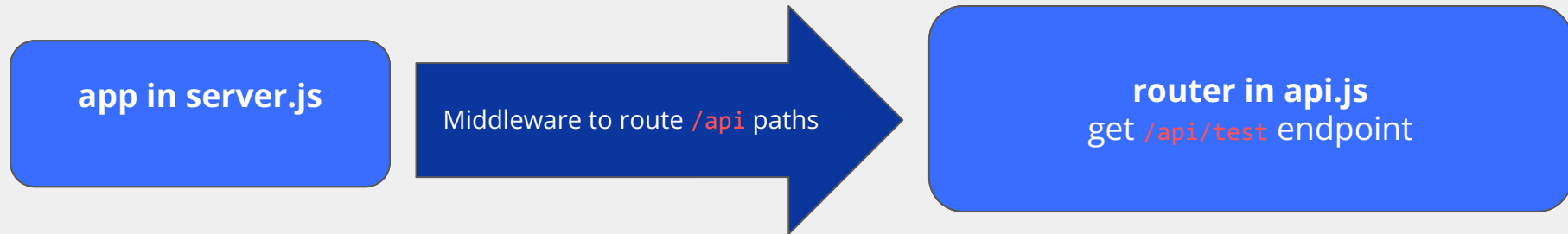
In `server.js`:

```
19 // import the router from our API file
20 const api = require("./api.js");
21
22 // create a new express server
23 const app = express();
24
25 // allow us to make post requests
26 app.use(express.json());
27
28 // connect API routes from api.js
29 app.use("/api", api);
```

# Moving our endpoints from our server to our router

Steps:

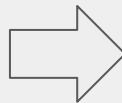

1. Create a separate file (we have already done this by making `api.js`)
2. Define a router for `/api` (use `express.Router()`) in `api.js`
3. Define middleware to route any api paths prefixed with `/api` to `api.js`
4. **Move our `/api/test` endpoint from `server.js` to `api.js`**



## TODO-5: Move our route into `api.js`

In `server.js`:

```
22 // create a new express server
23 const app = express();
24
25 // allow us to make post requests
26 app.use(express.json());
27
28 // TODO-4 connect API routes from api.js
29 app.use("/api", api);
30
31 // TODO-5 migrate endpoint to api.js
32 app.get("/api/test", (req, res) => {
33   res.send("I made my first API!");
34 });
```



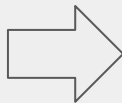

In `api.js`:

```
10 const express = require("express");
11
12 // TODO-1 add router
13 const router = express.Router();
14
15 // TODO-5 migrate api/test endpoint from server.js
16
17
18
19
20 // TODO-2 export router
21 module.exports = router;
```

## TODO-5: Move our route into `api.js`

In `server.js`:

```
22 // create a new express server
23 const app = express();
24
25 // allow us to make post requests
26 app.use(express.json());
27
28 // TODO-4 connect API routes from api.js
29 app.use("/api", api);
30
31 // TODO-5 migrate endpoint to api.js
32 app.get("/api/test", (req, res) => {
33   res.send({ message: "Wow I made my first API!" });
34 });
```



In `api.js`:

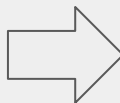
```
10 const express = require("express");
11
12 // TODO-1 add router
13 const router = express.Router();
14
15 // TODO-5 migrate api/test endpoint from server.js
16 router.get("/test", (req, res) => {
17   res.send({ message: "Wow I made my first API!" });
18 });
19
20 // TODO-2 export router
21 module.exports = router;
```



## TODO-5: Move our route into `api.js`

In `server.js`:

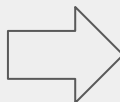
```
22 // create a new express server
23 const app = express();
24
25 // allow us to make post requests
26 app.use(express.json());
27
28 // TODO-4 connect API routes from api.js
29 app.use("/api", api);
30
31 // TODO-5 migrate endpoint to api.js
32 app.get("/api/test", (req, res) => {
33   res.send({ message: "Wow I made my first API!" });
34 });
```



In `api.js`:

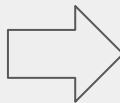
```
10 const express = require("express");
11
12 // TODO-1 add router
13 const router = express.Router();
14
15 // TODO-5 migrate api/test endpoint from server.js
16 router.get("/test", (req, res) => {
17   res.send({ message: "Wow I made my first API!" });
18 });
19
20 // TODO-2 export router
21 module.exports = router;
```

`app`



`router`

`"/api/test"`



`"/test"`

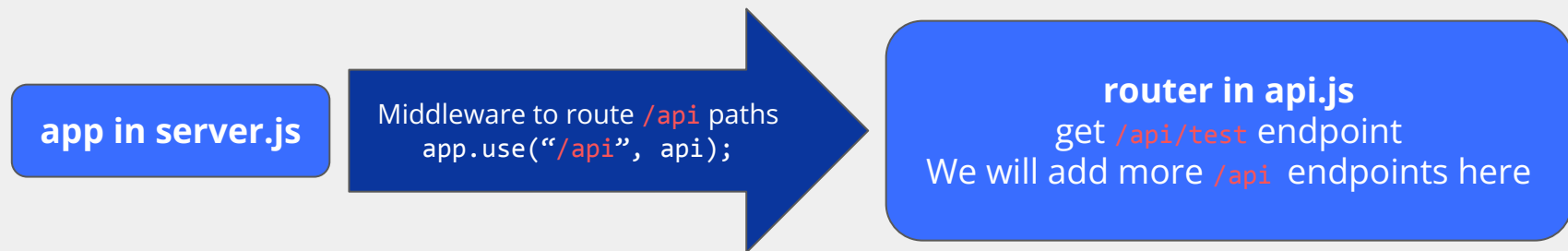
# Making our endpoints in our router

**app (server.js)**

Represents your overall server  
(main-application)

**router (api.js)**

Isolated group of API endpoints  
(mini-application)



## Test it out

Your `/api/test` route should work just like before, but now it has been moved into a separate file using a router!

Run `npm start`, then go to `localhost:3000/api/test` in your browser

# Test it out

Your `/api/test` route should work just like before, but now it has been moved into a separate file!

Run `npm start`, then go to `localhost:3000/api/test` in your browser

Have a bug but want to test the solution? Feel free to checkout the next step then test `/api/test`

```
git reset --hard  
git fetch  
git checkout w5-step1
```

# All together now...

```
git reset --hard
```

```
git fetch
```

```
git checkout w5-step1
```

## Step 2

Catbook GET routes!

# How to store our data?

For now, just declared an object called `data` in `api.js`

Stories are in `data.stories`

Comments are in `data.comments`

TODO-1: Add a story to `data.stories`

```
const data = {  
  stories: [  
    {  
      _id: 0,  
      creator_name: "Tony Cui",  
      content: "Send it or blend it?",  
    },  
    {  
      _id: 1,  
      creator_name: "Andrew Liu",  
      content: "web.labling with Tony <3",  
    }  
  ],  
  comments: [  
    {  
      _id: 0,  
      creator_name: "Stanley Zhao",  
      parent: 0,  
      content: "Both!",  
    },  
  ],  
};  
  
api.js
```

## TODO-2: GET Stories



Remember *these*??

```
// an example GET route
router.get("/test", (req, res) => {
  res.send({ message: "Wow I made my first API!" });
});
```

# req and res

req is the incoming request

...

req.query

req.body

req.params

Custom keys (ie. req.user)

...

res is your server's response

...

res.status(<status code>)

res.send(<object>)

...

## TODO-2: GET `/api/stories`

What does this route need to do?

Send back all the stories to the frontend!

How can we access all the stories?

`data.stories`

## TODO-2: GET /api/stories solution

```
38 // an example GET route
39 router.get("/test", (req, res) => {
40 |   res.send({ message: "Wow I made my first API!" });
41 | });
42
43 // TODO (step1) Add get stories endpoint
44 router.get("/stories", (req, res) => {
45 |   // send back all of the stories!
46 |   res.send(data.stories);
47 | });
```

# Let's test it out!

In one terminal:

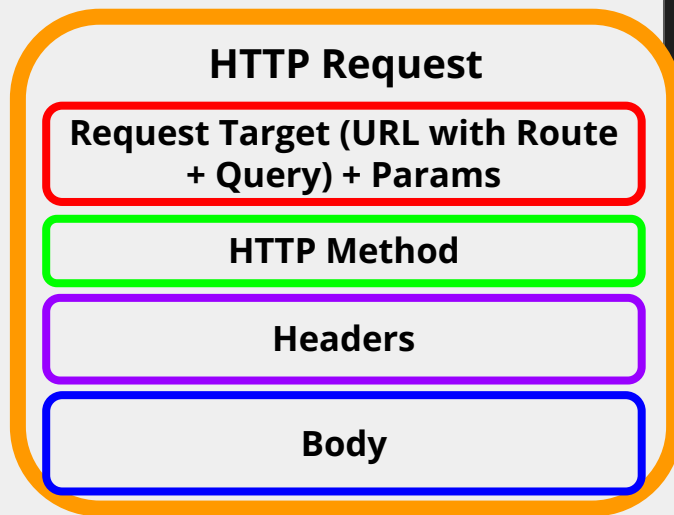
```
npm start
```

In *another* terminal:

```
npm run hotloader
```

... and go check localhost:5050 in your browser!

## TODO-3: GET Comments



**req.query**

For **GET** requests:

Use `req.query`

E.g. `req.query.content`

**req.body**

For **POST** requests:

Use `req.body`

E.g. `req.body.content`

## req.query

Use req.query for GET requests:

```
// Get request in frontend
const query = {
  content: "web development class and competition"
};
get("/api/weblab", query).then((comment) => {
  // ...
});
```

OR

localhost:3000/api/weblab?content="web development class and competition"

```
// Get request in backend
router.get("/weblab", (req, res) => {
  console.log(req.query.content);
  // "web development class and competition"
});
```

## req.body

Use req.body for POST request:

```
// Post request in frontend
const body = {
  content: "web development class and competition"
};
post("/api/weblab", body).then((comment) => {
  // ...
});
```



```
// express.json() middleware parses request body
app.use(express.json());

// Post request in backend
router.post("/weblab", (req, res) => {
  console.log(req.body.content);
  // "web development class and competition"
});
```



# req.query vs. req.body

Use req.query for GET requests:

```
// Get request in frontend
const query = {
  content: "web development class and competition"
};
get("/api/weblab", query).then((comment) => {
  // ...
});
```



localhost:3000/api/weblab?content="web development class and competition"



```
// Get request in backend
router.get("/weblab", (req, res) => {
  console.log(req.query.content);
  //"web development class and competition"
});
```

Use req.body for POST request:

```
// Post request in frontend
const body = {
  content: "web development class and competition"
};
post("/api/weblab", body).then((comment) => {
  // ...
});
```



```
// express.json() middleware parses request body
app.use(express.json());
```

```
// Post request in backend
router.post("/weblab", (req, res) => {
  console.log(req.body.content);
  //"web development class and competition"
});
```

## Remember this?

We included the **parent** story's **\_id** prop when we made the GET from the frontend!

```
16  const Card = (props) => {  
17    const [comments, setComments] = useState([]);  
18  
19    useEffect(() => {  
20      get("/api/comment", { parent: props._id }).then((comments) => {  
21        setComments(comments);  
22      });  
23    }, []);  
  }
```

Card.js

How can we access this from the backend?

`req.query.parent`

## TODO-3: GET /api/comment

What does this route need to do?

1. Figure out what story we need the comments from (hint: `req.query`)
2. Filter out only the comments that are children of that story
3. Send back those comments to the frontend!

## TODO-3: Your turn! GET /api/comment

Add an API route that correctly responds to GET requests to /api/comment.

1. Figure out what story we need the comments from (hint: `req.query`)
2. Filter out only the comments that are children of that story
3. Send back those comments to the frontend!

Hint: It's a lot like /api/stories, but with a bit more logic.

Hint 2: You might want the `array.filter((item) => return <condition>);` function from the JavaScript (cont'd) on Tuesday!

Hint 3: `req.query` returns a string. Use `==` to compare values of different data types.

## TODO-3: GET /api/comment

What does this route need to do?

1. Figure out what story we need the comments from
  - The parent story\_id contained within `req.query`
2. Filter out only the comments that are children of that story
  - We'll use JavaScript's `filter()` function on our `data.comments`
3. Send back those comments to the frontend!
  - Use `res.send()` just like our other endpoints

## TODO-3: GET /api/comment solution

```
const filteredComments =  
  data.comments.filter(  
    (comment) => comment.parent == req.query.parent  
  );
```

Or in English...

get just the comments whose parent is equal to req.query.parent

## TODO-3: Expanded GET /api/comment solution

```
// TODO (step1) Add get comments endpoint
router.get("/comment", (req, res) =>
{
  const filteredComments = data.comments.filter((comment) => comment.parent == req.query.parent);
  res.send(filteredComments);
})
```

# Let's test it out with comments!

In one terminal:

```
npm start
```

In *another* terminal:

```
npm run hotloader
```

... and go check localhost:5050 in your browser!



# Now we see our stories and comments from data!

Catbook Home Profile

Andrew Liu

weblabing with Tony <3

Tony Cui

Send it or blend it?

Stanley Zhao | Both!

Feel free to checkout the solution if you have a bug

```
git reset --hard  
git fetch  
git checkout w5-step2
```

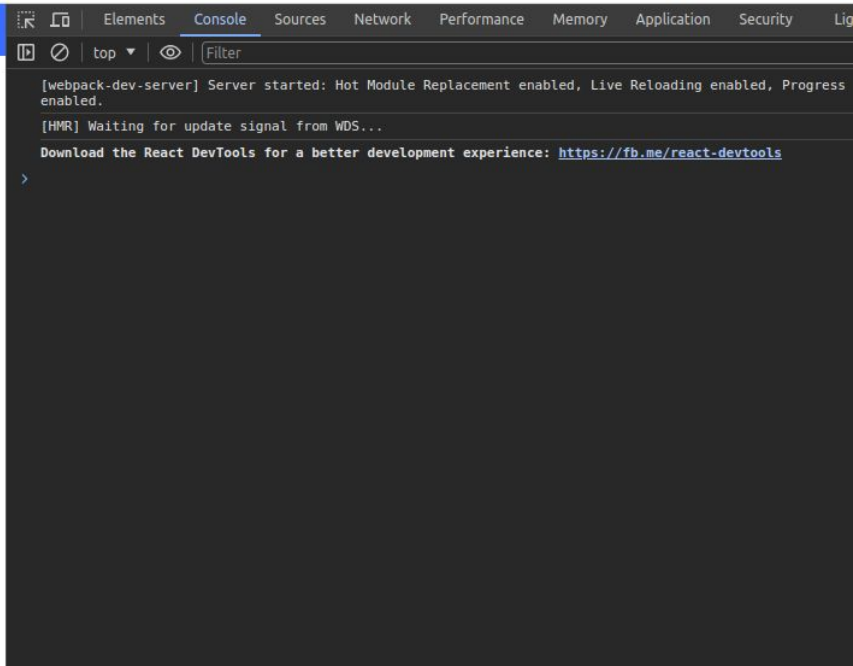
# Inspect to see details

Catbook [Home](#) [Profile](#)

**Andrew Liu**  
weblabing with Tony <3

**Tony Cui**  
Send it or blend it?

**Stanley Zhao** | Both!



# Back on the same page!

```
git reset --hard
```

```
git fetch
```

```
git checkout w5-step2
```

# Making a new story and inspect Console + Network

The image shows a web application interface on the left and its DevTools console on the right. The web application has a blue header with 'Catbook', 'Home', and 'Profile' links. Below the header, there's a 'New Story' form with a red prompt 'Try typing here and making a new story!' and a 'Submit' button. The form contains the text 'Andrew Liu' and 'weblabing with Tony <3'. Below this is a 'New Comment' form with a 'Submit' button. Further down, there's a section for 'Tony Cui' with the text 'Send it or blend it?'. At the bottom, there's a section for 'Stanley Zhao | Both!' with a 'New Comment' form and a 'Submit' button. The DevTools console on the right has tabs for 'Elements', 'Console', 'Sources', and 'Network'. The 'Console' tab is active, showing several log messages. The first two are informational: '[webpack-dev-server] Server started: Hot Module Replacement enabled, Live Reloading enabled, Progress disabled, Overlay enabled.' and '[HMR] Waiting for update signal from WDS...'. The third is a warning: 'Download the React DevTools for a better development experience: https://fb.me/react-devtools'. The fourth is an error: 'POST http://localhost:5050/api/story 404 (Not Found)'. The fifth is an uncaught promise error: 'Uncaught (in promise) POST request to /api/story failed with error: API request failed with response status 404 and text: Not Found'. The 'Network' tab is also visible, showing the failed POST request to 'http://localhost:5050/api/story'.

Catbook Home Profile

New Story Try typing here and making a new story! Submit

Andrew Liu

weblabing with Tony <3

New Comment Submit

Tony Cui

Send it or blend it?

Stanley Zhao | Both!

New Comment Submit

Elements Console Sources Network Performance Memory Application Security Lighthouse >> 2 3

[webpack-dev-server] Server started: Hot Module Replacement enabled, Live Reloading enabled, Progress disabled, Overlay enabled. bootstrap:24

[HMR] Waiting for update signal from WDS... bootstrap:24

Download the React DevTools for a better development experience: <https://fb.me/react-devtools> bootstrap:24

POST http://localhost:5050/api/story 404 (Not Found) utilities.js:53 @

Uncaught (in promise) POST request to /api/story failed with error: localhost:1

API request failed with response status 404 and text: Not Found

Inspect what is outputted in Console and Network when you make a new story/comment

# What happened?

404 error :(

```
✖ ▶ POST http://localhost:5050/api/comment 404 (Not Found) utilities.js:53 ⚙  
✖ ▶ Uncaught (in promise) POST request to /api/comment failed with error: localhost/:1  
API request failed with response status 404 and text: Not Found
```



## TODO-1: Handling missing API routes

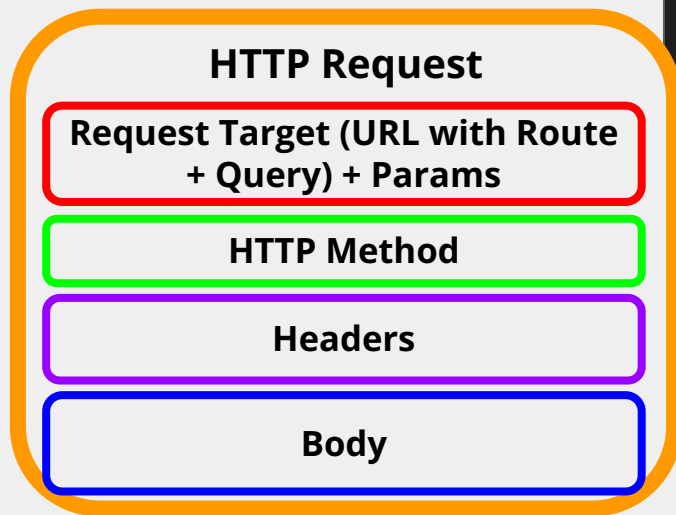
In `api.js`:

```
78 // any requests that don't have a defined route go to this case
79 router.all("*", (req, res) => {
80   console.log(`API Route not found: ${req.method} ${req.url}`);
81   res.status(404).send({ message: "API Route not found" });
82 });
83
84 module.exports = router;
```

**IMPORTANT:** Put this route below all other API routes or their requests will go to this route!

## Step 3

Catbook POST routes!



`req.query`

For GET requests:

Use `req.query`

E.g. `req.query.content`

`req.body`

For **POST** requests:

Use `req.body`

E.g. `req.body.content`



## req.query

Use req.query for GET requests:

```
// Get request in frontend
const query = {
  content: "web development class and competition"
};
get("/api/weblab", query).then((comment) => {
  // ...
});
```



localhost:3000/api/weblab?content="web development class and competition"



```
// Get request in backend
router.get("/weblab", (req, res) => {
  console.log(req.query.content);
  //"web development class and competition"
});
```

## req.body

Use req.body for POST request:

```
// Post request in frontend
const body = {
  content: "web development class and competition"
};
post("/api/weblab", body).then((comment) => {
  // ...
});
```



```
// express.json() middleware parses request body
app.use(express.json());
```

```
// Post request in backend
router.post("/weblab", (req, res) => {
  console.log(req.body.content);
  //"web development class and competition"
});
```

## TODO-2: POST Stories

# What do our POST endpoints need to handle?

1. Figure out what data needs to be saved
2. Put it in a unified structure
3. Add it to our **data** object

Hint 1: What a new story needs (from **data.stories**):

```
stories: [  
  {  
    _id: 0,  
    creator_name: "Tony Cui",  
    content: "Send it or blend it?",  
  },  
]
```

Hint 2: What is sent from the frontend (find post req in **NewPostInput.js**):

```
const addStory = (value) => {  
  const body = { content: value };  
  post("/api/story", body).then((story) => {  
    // display this story on the screen  
    props.addNewStory(story);  
  });  
};
```

## TODO-2: POST /api/story

In `api.js`:

```
const newStory = {  
  _id: data.stories.length,  
  creator_name: myName,  
  content: req.body.content,  
};
```

## TODO-2: POST /api/story solution

```
43 // TODO-2 (step1) Add get stories endpoint
44 router.post("/story", (req, res) => {
45   // create a new story object
46   const newStory = {
47     _id: data.stories.length,
48     creator_name: myName,
49     content: req.body.content,
50   };
51
52   // add it to our backend data
53   data.stories.push(newStory);
54   // send it to the frontend
55   res.send(newStory);
56 });
```

# Let's test adding a story!

In one terminal:

```
npm start
```

In *another* terminal:

```
npm run hotloader
```

... and go check localhost:5050 in your browser - posting stories and comments should now work!

## TODO-3: POST Comments

## TODO-3: Your turn! **POST** `/api/comment`

Add an API route that correctly handles **POST** requests to `/api/comment`.



## TODO-3: Your turn! `POST /api/comment`

Add an API route that correctly handles `POST` requests to `/api/comment`.

Hint 1: It's a lot like `POST /api/story`.

Hint 2: What a new comment needs (from `data.comments`):

```
comments: [  
  {  
    _id: 0,  
    creator_name: "Stanley Zhao",  
    parent: 0,  
    content: "Both!",  
  },  
],
```

Hint 3: What is sent from the frontend (in `NewPostInput.js`):

```
57 const NewComment = (props) => {  
58   const addComment = (value) => {  
59     const body = { parent: props.storyId, content: value };  
60     post("/api/comment", body).then((comment) => {  
61       // display this comment on the screen  
62       props.addNewComment(comment);  
63     });  
64   };  
};
```

▼ Request Payload [view source](#)

```
▼ {parent: 1, content: "Congrats on 1st Place!"}  
  content: "Congrats on 1st Place!"  
  parent: 1
```

## TODO-3: POST /api/comment solution

```
69 // TODO-3 (step2) Add post comment endpoint
70 router.post("/comment", (req, res) => {
71   // create a new comment object
72   const newComment = {
73     _id: data.comments.length,
74     creator_name: myName,
75     parent: req.body.parent,
76     content: req.body.content,
77   };
78
79   // add it to our backend data
80   data.comments.push(newComment);
81   // send it to the frontend
82   res.send(newComment);
83 });
```

# One last test

In one terminal:

```
npm start
```

In *another* terminal:

```
npm run hotloader
```

... and go check localhost:5050 in your browser - posting stories and comments should now work!

← → ↺ 🏠 localhost:5050 ☆ 📧 👤 📺 3 📍 📄 📌 ☰

Catbook Home Profile

Andrew Liu

web.labing with Tony <3

Tony Cui

Send it or blend it?

Stanley Zhao | Both!

Feel free to checkout the solution if you have a bug

```
git reset --hard
git fetch
git checkout w5-complete
```

← → ↺ 🏠 🛡️ 📄 localhost:5050 ☆ 📧 👤 📱 🔵 📄 📁 📌 ☰

Catbook Home Profile

Andrew Liu

web.labing with Tony <3

Web Lab 2024 Staff | Congrats on 1st Place!

Tony Cui

Send it or blend it?

Stanley Zhao | Both!

Web Lab 2024 Staff | That's the theme!

Feel free to checkout the solution if you have a bug

```
git reset --hard
git fetch
git checkout w5-complete
```

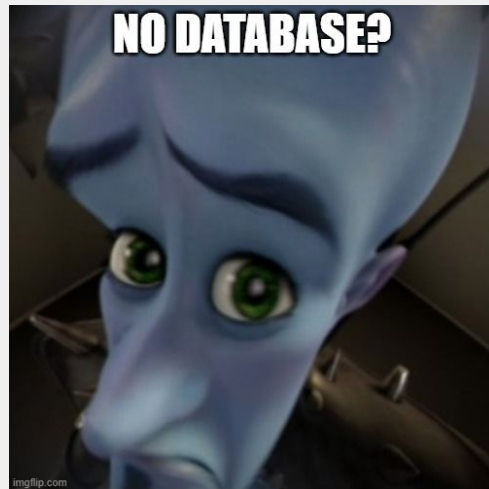
# What's next?

Change something in `server.js` or `api.js`

`nodemon` will notice the change and reload the server

.... and all of the new posts and comments are gone!

Since `data` is just defined at the top of our server file, it only lasts as long as the server stays running :(







# Tomorrow: Intro to Databases

Set up MongoDB Atlas before tomorrow's  
workshop!

**<http://weblab.is/mongo-setup>**

# Announcements

- HW2 Due today for workshops tmr! (Setup MongoDB Acc)
- Milestone 0 deadline extended to 5:30pm today!
- Milestone 1 (Project Pitch) signups out! Time slots are for Sat and Sun 1-5pm.
  - [weblab.is/milestone1](https://weblab.is/milestone1)
- Fill out [weblab.is/feedback](https://weblab.is/feedback)! Note that the “What day is it” question is the day you’re giving feedback for, not the current day.
-     ([weblab.is/milkandcookies](https://weblab.is/milkandcookies)) uwu
- Still need a team? Stay after lecture today!
- Informal OH now, tomorrow 3-5 (after lecture)