Build an API!

Noah Raby

App vs. Router

Why? Simplicity! Modularity!

server.js is already over 60 lines and it doesn't even do anything useful yet...

`app`

Represents your overall web application

`router`

Isolated group of API endpoints (mini-application)

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 (there is a 1:1 mapping)

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
```

Use `api` Route for Requests

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

Part 1: Create the router

server > JS api.js > ...

```
/*
|-------------------------------------------------------------------------
| api.js -- server routes
|-------------------------------------------------------------------------
|
| This file defines the routes for your server.
|
*/

const express = require("express");

const router = express.Router();

```

Part 2: Connect `api.js` to our main server

In `api.js`:

server > JS api.js > ...

```
/*
|-----------------------------------------------------------------------
| api.js -- server routes
|-----------------------------------------------------------------------
|
| This file defines the routes for your server.
|
*/

const express = require("express");

const router = express.Router();

module.exports = router;
```

Part 2: Connect `api.js` to our main server

In `server.js`:

```
20
21  // import libraries needed for the webserver to work!
22  const express = require("express"); // backend framework for our node server.
23  const path = require("path"); // provide utilities for working with file and directory paths
24
25  // import the router from our API file
26  const api = require("./api.js");
27
28  // create a new express server
29  const app = express();
30  app.use(validator.checkRoutes);
```

Part 2: Connect `api.js` to our main server

In `server.js`:

```
24
25  // import the router from our API file
26  const api = require("./api.js");
27
28  // create a new express server
29  const app = express();
30  app.use(validator.checkRoutes);
31
32  // allow us to parse POST request data using middleware
33  app.use(express.json());
34
35  // connect API routes from api.js
36  app.use("/api", api);
37
```

Part 3: Move our route into `api.js`

In `server.js`:

```
28  // create a new express server
29  const app = express();
30  app.use(validator.checkRoutes);
31
32  // allow us to parse POST request data
33  app.use(express.json());
34
35  // connect API routes from api.js
36  app.use("/api", api);
37
38  // an example GET route
39  app.get("/test", (req, res) => {
40    res.send({ message: "it works" });
41  });
42
```

In `api.js`:

```
35  const router = express.Router();
36
37  // an example GET route
38  router.get("/test", (req, res) => {
39    res.send({ message: "it works" });
40  });
41
42  module.exports = router;
43
```

Part 3: Move our route into `api.js`

```
34
35  const router = express.Router();
36
37  // an example GET route
38  router.get("/test", (req, res) => {
39    res.send({ message: "it works" });
40  });
41
42  module.exports = router;
43
```

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

All together now...

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

Catbook routes!

How to store our data?

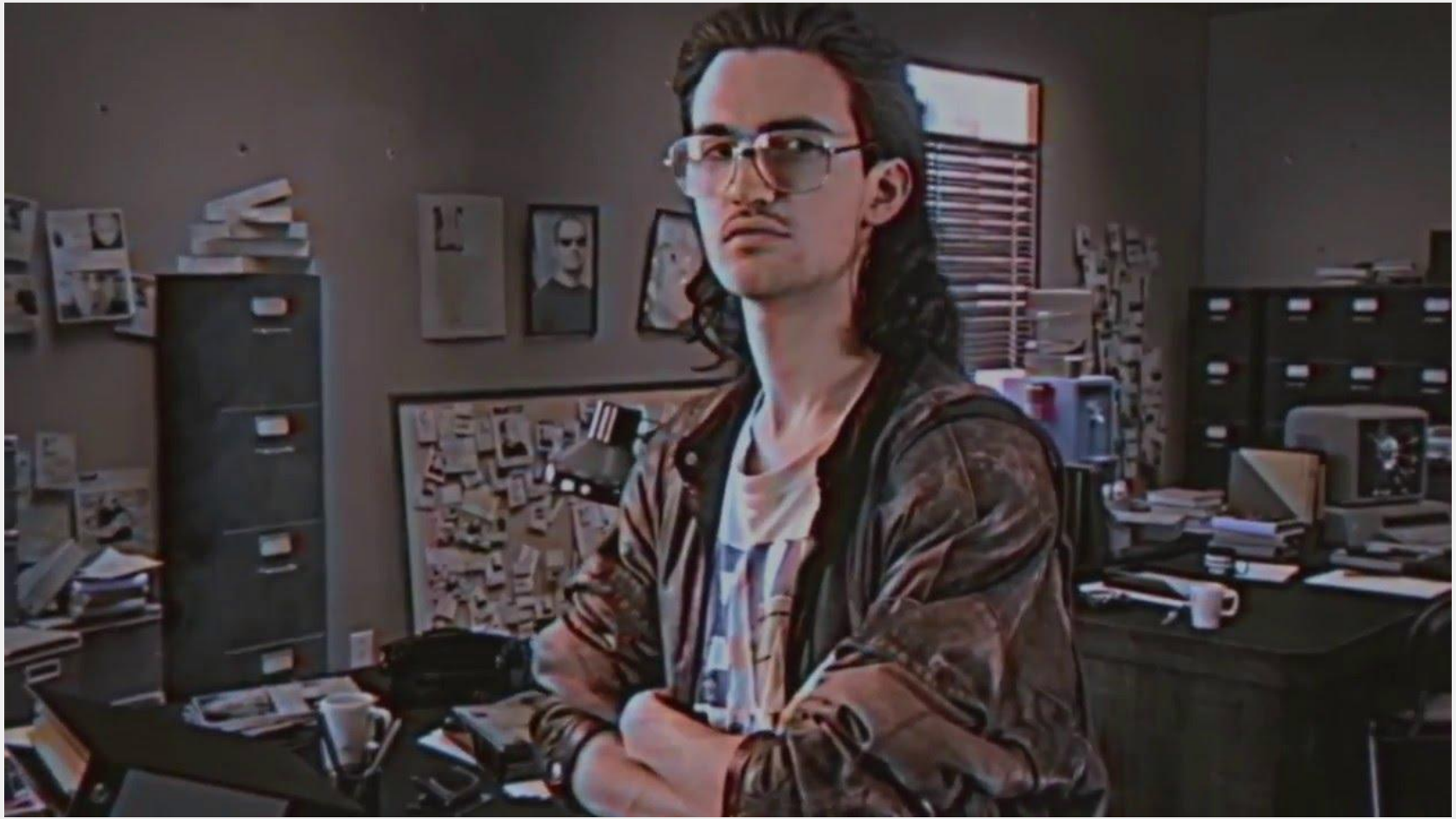
For now, just declare an object called `data`

Stories are in `data.stories`

Comments are in `data.comments`

```
12  // we haven't set up user login yet, so just
13  // use a hardcoded name for now
14  // TODO change to a unique name for workshop
15  const MY_NAME = "Anonymous User";
16
17  let data = {
18    stories: [
19      {
20        _id: 0,
21        creator_name: "Shannen Wu",
22        content: "I love corgis!",
23      },
24    ],
25    comments: [
26      {
27        _id: 0,
28        creator_name: "Jessica Tang",
29        parent: 0,
30        content: "Wow! Me too!",
31      },
32    ],
33  };
```

api.js



Part 1: Stories

But what are *those??*

```
// an example GET route
router.get("/test", (req, res) => {
  res.send({ message: "it works" });
});
```

req and res

req is the incoming request

```
req.query
req.body
...
```

res is your server's response

```
res.send(<object>)
res.status(<status code>)
...
```

Part 1: `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`

Part 1: GET /api/stories

```
41
42  router.get("/stories", (req, res) => {
43    // just send back all of the stories!
44    res.send(data.stories);
45  });
46
```

Part 2: Comments

Remember this?

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

```
useEffect(() => {
  get("/api/comment", { parent: props._id }).then((comments) => {
    setComments(comments);
  });
}, []);
```

Part 2: `GET /api/comments`

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!

Part 2: `GET /api/comment`

Now it's your turn....

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

1. Figure out what story we need the comments from
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 Justin's Intro to JavaScript on Tuesday!

Part 2: `GET /api/comment`

What does this route need to do?

1. Figure out what story we need the comments from
   - The _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

Part 2: `GET /api/comments` 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

Let's test it out!

In one terminal:

```
npm start
```

In *another* terminal:

```
npm run hotloader
```

... and go check `localhost:5000` in your browser!

News Feed - Google Chrome

News Feed

localhost:5000

Catbook Home Profile

New Story Submit

Shannen Wu

I love corgis!

Jessica Tang | Wow! Me too!

New Comment Submit

News Feed - Google Chrome

News Feed

localhost:5000

Catbook Home Profile

New Story Submit

Shannen Wu

I love corgis!

Jessica Tang | Wow! Me too!

I want to write a new comment too!!! Submit

Elements Console Sources Network Performance

top Filter Default levels

Console was cleared VM174:1

undefined

News Feed - Google Chrome

News Feed

localhost:5000

Catbook Home Profile

New Story Submit

Shannen Wu

I love corgis!

Jessica Tang | Wow! Me too!

New Comment Submit

Elements Console Sources Network

top Filter Default levels

```
Console was cleared                                                    VM174:1
undefined
POST http://localhost:5000/api/comment 404 (Not Found)                 utilities.js:53
Uncaught (in promise) POST request to /api/comment failed              localhost/:1
with error:
API request failed with response status 404 and text: Not Found
```

```
<· undefined
POST http://localhost:5000/api/comment 404 (Not Found)        utilities.js:53
Uncaught (in promise) POST request to /api/comment failed      localhost/:1
with error:
API request failed with response status 404 and text: Not Found
>
```

Back on the same page!

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

Part 1: Handling missing API routes

```
62
63  // anything else falls to this "not found" case
64  router.all("*", (req, res) => {
65    console.log(`API route not found: ${req.method} ${req.url}`);
66    res.status(404).send({ msg: "API route not found" });
67  });
68
69  module.exports = router;
70
```

The POST body

From `NewPostInput.js`:

```
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**

```
{"parent":0,"content":"I want to write a new comment too!!!"}
```

`req.query` vs. `req.body`

For `GET` requests:

Use `req.query`

E.g. `req.query.content`

For `POST` requests:

Use `req.body`

E.g. `req.body.content`

Part 1: 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

Part 1: POST /api/story

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

Part 1: POST /api/story

```
router.post("/story", (req, res) => {
  const newStory = {
    _id: data.stories.length,
    creator_name: MY_NAME,
    content: req.body.content,
  };

  data.stories.push(newStory);
  res.send(newStory);
});
```

Part 2: Comments

Part 2: `POST /api/comment`

Now it's your turn....

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

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

Part 2: POST /api/comment solution

```
63
64  router.post("/comment", (req, res) => {
65    const newComment = {
66      _id: data.comments.length,
67      creator_name: MY_NAME,
68      parent: req.body.parent,
69      content: req.body.content,
70    };
71
72    data.comments.push(newComment);
73    res.send(newComment);
74  });
75
```

One last test

In one terminal:

```
npm start
```

In *another* terminal:

```
npm run hotloader
```

... and go check `localhost:5000` in your browser - posting stories and comments should now work!

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 :(

Next: Intro to Databases

GET MONGODB WORKING YOU FILTHY ANIMALS (weblab.to/homework1)