Node.js and Express.js

Spotfiy playlist

6.1040 Recitation 9/19/2024

Walk through prep solution

Node.js

- Server-side JavaScript runtime environment
- Built for web development
- Comes with lots of packages

Express.js

- Popular Node.js backend web application framework
- Handles routing

Express & Web APIs

- You may have seen code like this for a GET request to the “`/users/:username`” endpoint (e.g., `/users/max`):

```
router.get("/users/:username", async (req: Request, res: Response) => {
    const users = await User.getUsers(req.params.username);
    res.status(200).json({
        message: “Found user”,
        user: users[0],
    });
})
```

Express & Web APIs

- In the framework given in the starter code, here's how we might write a GET request to the `/users/:username` endpoint to get a single user:

```
@Router.get("/users/:username")
async getUser(username: string) {
    const user = await Authing.getUserByUsername(username);
    return { msg: "Found user", user: user }
};
```

(`src/framework/router.ts` handles the status part)

Parameters

- Path params: “eecs.mit.edu/people/max-goldman/”
- Query: “student.mit.edu/catalog/search.cgi?search=6.1040”
- Body: not in URL, can be a JSON

More review:
https://web.mit.edu/6.102/www/sp23/classes/18-message-passing-networking/#web_apis

Parameters

- Params example: `@Router.get("/posts/:postId)`
  - `GET /posts/65012cf9be58f89b0c6d096b`

- Query example: `getUsers(username: string)`
  - `GET /users?username=dnj`

- Body example: `createPost(content: string)`
  - `POST body { content: “This is my post” }`

Parameters

In our framework:

- If the parameter name is “session”, it’s read as express-session (which we pass into Sessioning)
- If the parameter name is “params”, “query”, or “body”, it is read from req.params, req.query, or req.body, respectively
- Otherwise, the parameter is searched for in req.params, req.query, and req.body, in that order

```
function handleRequest(session) {
    // session is now the express-session object
}
```

```
function handleRequest(params, query, body) {
    // params is req.params
    // query is req.query
    // body is req.body
}
```

```
function handleRequest(id) {
    // First, checks req.params.id
    // If not found, checks req.query.id
    // If still not found, checks req.body.id
}
```

Parameters

- If the parameter name is not “session”, “params”, “query”, or “body”. the parameter is searched for in req.params, req.query, and req.body, in that order
- For this example:

```
@Router.post("/posts")
async createPost(session: SessionDoc, content: string) {
… }
```

We will send “content” in the request body. When interpreting the parameters here, the router will look first at req.params (but it won’t find content because we have no URL params), will then look at req.query, and since it won’t find it there since we sent it in body, it will finally look in req.body.

Validation

- What if we want to throw an exception if something is invalid?
- In this example, `User.addUser` should be written to throw an exception when there is already a user with that username, so we don't need any if/else logic here

```
@Router.post("/users")
async addUser(username: string, password: string) {
    await User.addUser(username, password);
    return { msg: “Successfully added user” }
};
```

Walk through starter code

Exercise: Validation

Exercise

- **Clone https://github.com/61040-fa24/rec-3-exercise**
- Run `npm install`
- Right now, you can start a session with a user who isn't registered yet
- **Working with a partner, add a check that makes sure the user is registered before they can start a session**
- Add a new function that throws an error if the username hasn't been registered yet
- Use this function to validate that the user is registered before allowing login

Walk through solution