6.1040 Rec 5

Node, Sessions, & Cookies 🍪

Plan for today

1. Review yesterday’s lecture (Express, routes)
2. Middleware in Express
3. Sessions and cookies
4. Demo and exercises!

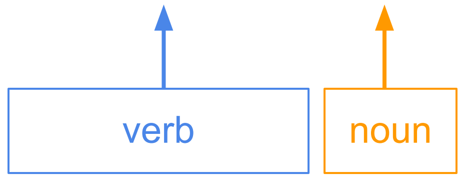
Moving to Node.js

- JavaScript environment built for web dev. Lots of packages so you don’t have to rebuild generic concepts
- Node vs Jekyll: *dynamic* vs **static**

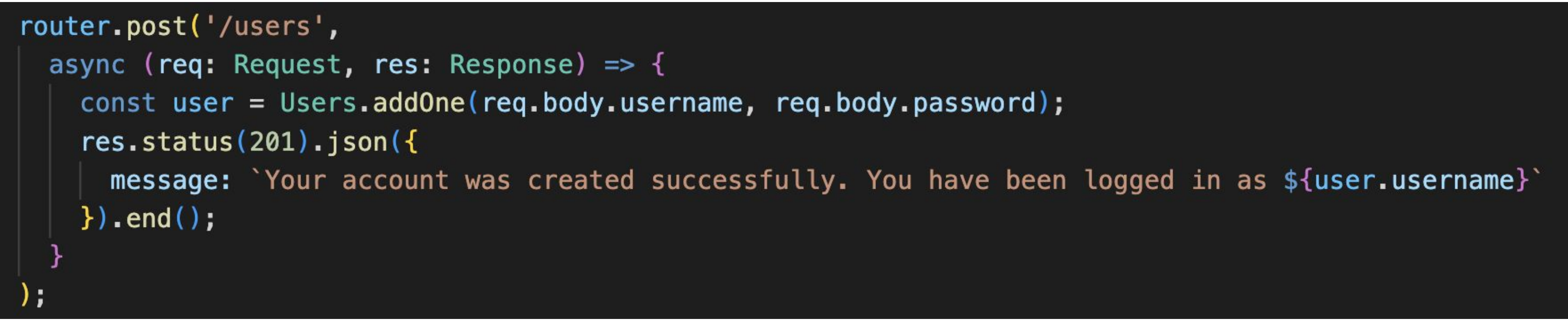
Static	**Dynamic**
Site is the same for every user	Site is different depending on the user
Site is pre-generated	Site is generated on the fly
Faster to load	Slower to load

Review of Express

- Express: web serving framework built on Node.js
- Main purpose: routing
  - Secondary purpose: adding even more packages
- Routing: `router.method(“/route”, (req, res) => {     })`



Review of routing syntax



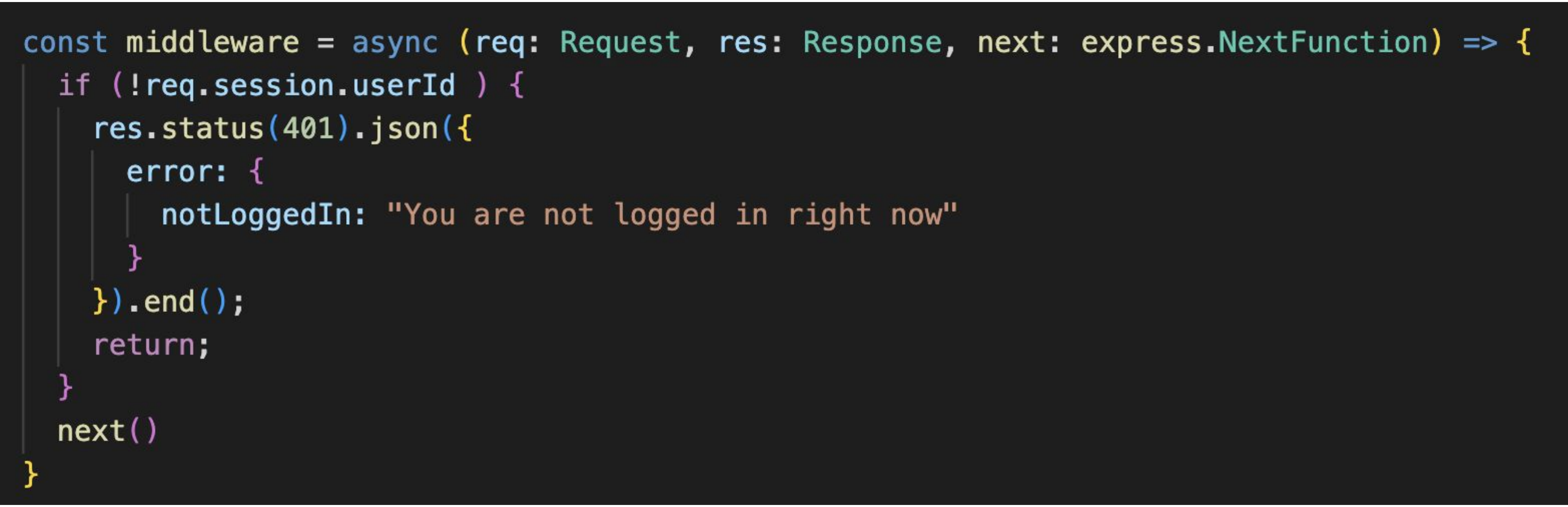
```
router.post('/users',
  async (req: Request, res: Response) => {
    const user = Users.addOne(req.body.username, req.body.password);
    res.status(201).json({
      message: `Your account was created successfully. You have been logged in as ${user.username}`
    }).end();
  }
);
```

- Line 1: RESTful verb and noun
- Line 2: request and response
- Line 3: accessing body of request
- Lines 4-6: sending HTTP status code and message in response

Middleware

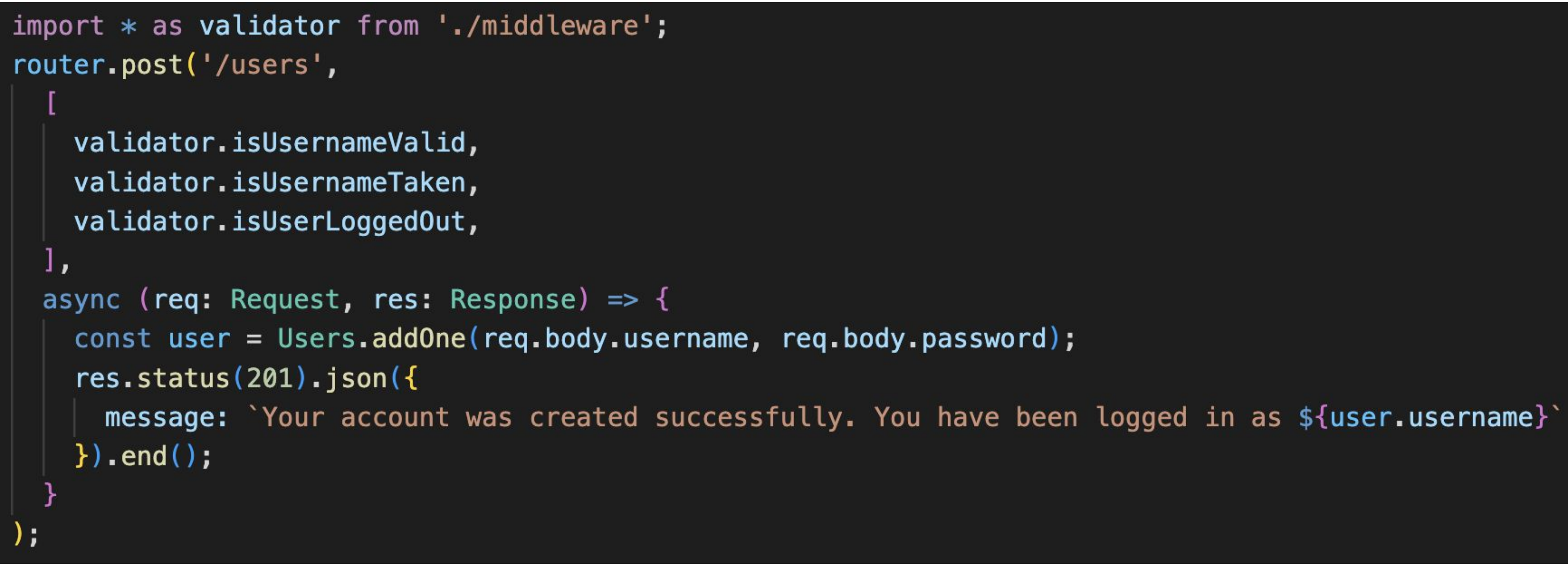
- Anything that happens between the request and the response
- Things you want to do before you even start the response
- Things you want to do on a lot of routes
- Often used for validation. Common examples
  - Checking if a user is logged in
  - Checking if a route’s parameters are correctly formatted
- Note: the UI also provides validation, but good to have both to prevent hackers

Middleware in Express



```
const middleware = async (req: Request, res: Response, next: express.NextFunction) => {
  if (!req.session.userId ) {
    res.status(401).json({
      error: {
        notLoggedIn: "You are not logged in right now"
      }
    }).end();
    return;
  }
  next()
}
```

Middleware + Routing

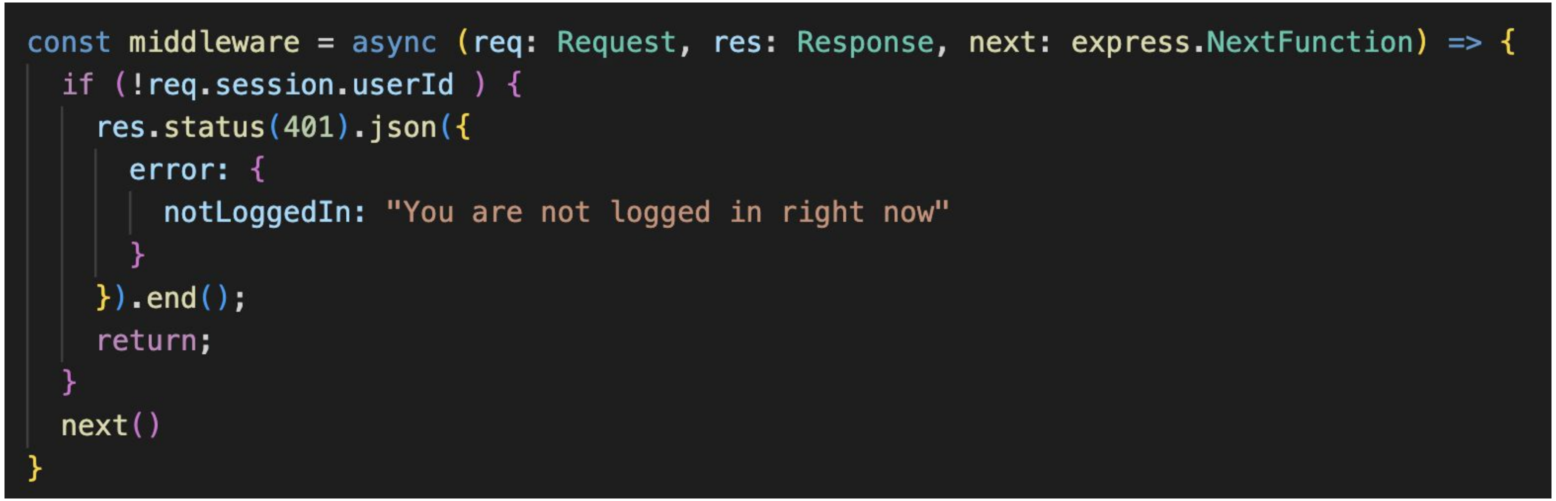


```
import * as validator from './middleware';
router.post('/users',
  [
    validator.isUsernameValid,
    validator.isUsernameTaken,
    validator.isUserLoggedOut,
  ],
  async (req: Request, res: Response) => {
    const user = Users.addOne(req.body.username, req.body.password);
    res.status(201).json({
      message: `Your account was created successfully. You have been logged in as ${user.username}`
    }).end();
  }
);
```

Exercise

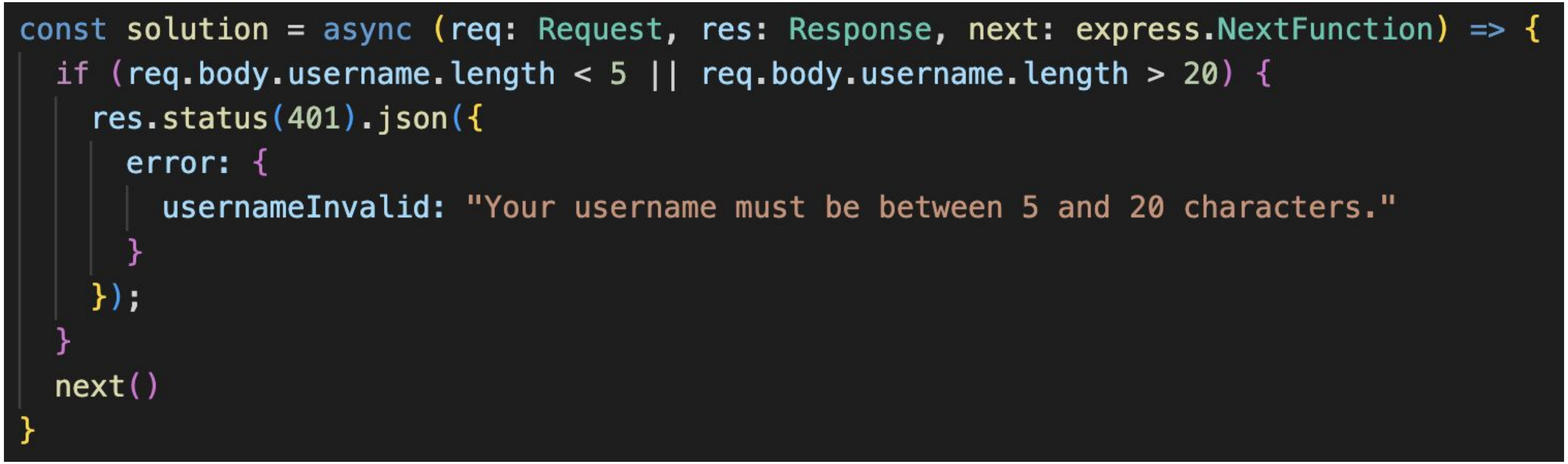
Exercise: write a piece of middleware that checks if a username is valid (you can decide validity!)

Previous middleware for reference:



```
const middleware = async (req: Request, res: Response, next: express.NextFunction) => {
  if (!req.session.userId ) {
    res.status(401).json({
      error: {
        notLoggedIn: "You are not logged in right now"
      }
    }).end();
    return;
  }
  next()
}
```

Example solution



```
const solution = async (req: Request, res: Response, next: express.NextFunction) => {
  if (req.body.username.length < 5 || req.body.username.length > 20) {
    res.status(401).json({
      error: {
        usernameInvalid: "Your username must be between 5 and 20 characters."
      }
    });
  }
  next()
}
```

*Note - there is actually a mistake in this solution! There should be a return inside the if statement (after line 7)

Sessions

- A **session** = a set of requests from the same client in a given time period
  - Same client = same person? Not always
- Why do we care?
  - HTTP is **stateless** → need sessions for **persistence**
- How long should the time period be? How many requests are allowed? Out of scope :)

Cookies

- Cookies are how servers keep track of sessions
- Unique identifier for the client for the duration of the session
- Also how servers keep track of a lot of other things
    - Your preferences (good)
    - Your preferences (bad)

Example repository

Repo: https://github.com/61040-fa22/rec5

Installation instructions:

- Git clone using the URL under “Code” on the top right:

- Move to the cloned directory in the command line
  - Run `npm install` (just the first time after you clone the repo)
  - Run `npm start` (every time you want to start the site)
  - Go to localhost:3000/ in your browser

Exercises

1. Add a variable to sessions to count the number of times the user has viewed the page and report it back to them in the response
2. Your choice! Extend the implementation. Some ideas:
    a. Add middleware that checks if the user’s password is correct and if their account exists, and don’t let them log in if not
    b. Add middleware that checks if the user is logged in/logged out, and don’t let them log out/log in otherwise
    c. Add middleware that checks if the user’s username is taken and don’t let them create an account if it is

Resources

- Official Express docs (pretty helpful!):
  - Routing in Express: https://expressjs.com/en/guide/routing.html
  - Middleware in Express: https://expressjs.com/en/guide/using-middleware.html
- Multi-part Express tutorial: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs (multi-part tutorial)
- express-session:
  - https://www.npmjs.com/package/express-session (official docs, lots of info but harder to follow)
  - https://www.tutorialspoint.com/expressjs/expressjs_sessions.htm (unofficial tutorial, not guaranteed to be up to date but more helpful overall)