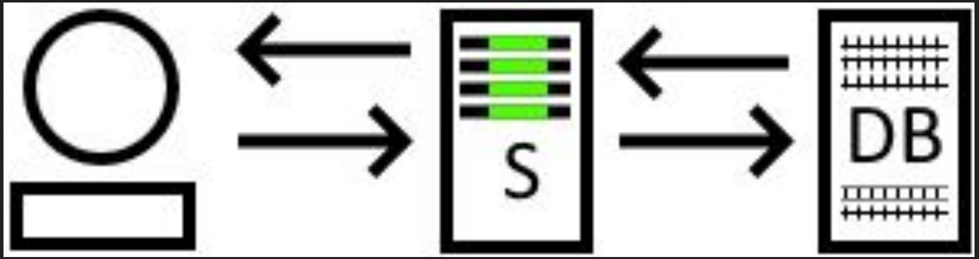
Week 3: Routes

The Purpose of Routes



When a user makes a *request*, they make it to the server. The server then does a particular action, usually interacting with the database, and gives the user a *response*. The user never interacts with the database directly - only the server.

Types of Routes

There are many, but the ones you’ll focus on include:

GET - used only to request data; an example would be trying to read a user by id

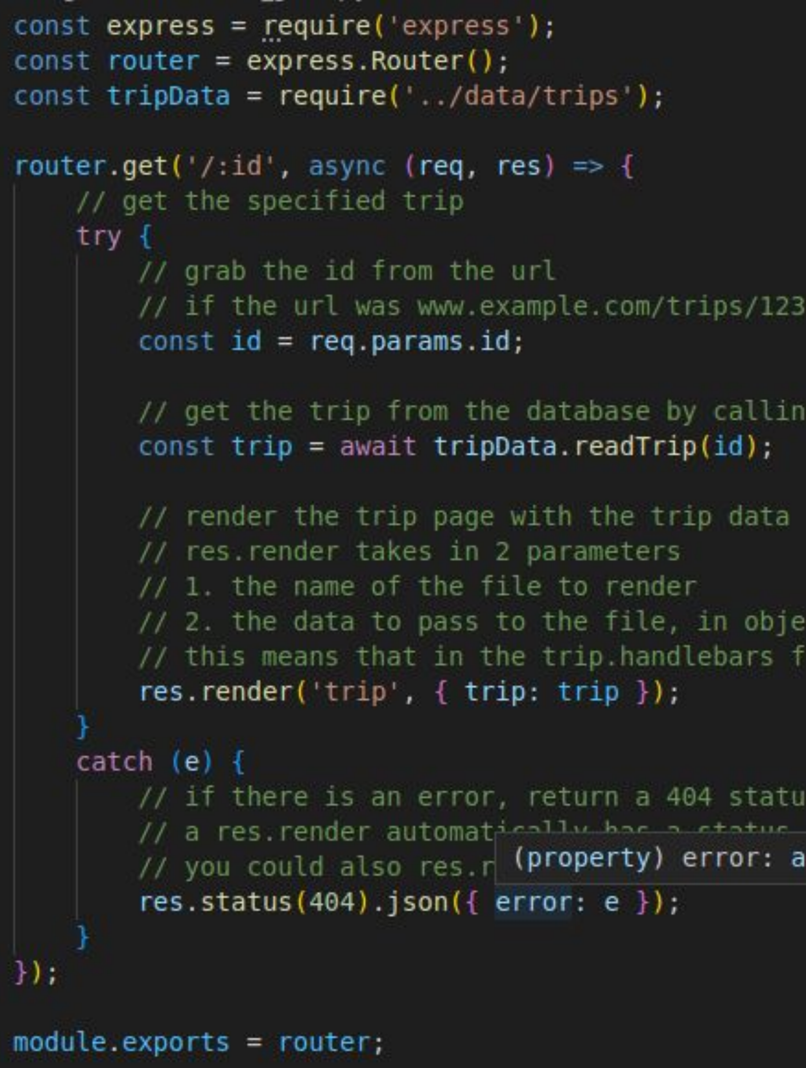
POST - submits some data and expects the server to do something with the given information; will be used for all the non-get interactions

Other types include **PUT**, **PATCH**, **DELETE**, but you don’t really need to use them since they’re functionally doing the same thing as **POST**

GET Route

The get route will get the specified trip

- From the URL, we can grab the id of the trip we're looking for
  - www.example.com/trips/1234 should only show information about that trip
- Once we have it, we can run our database function using the id
- Now that we have the trip, we *render* a page back to the user and give the trip to that page so that it can use it



```
const express = require('express');
const router = express.Router();
const tripData = require('../data/trips');

router.get('/:id', async (req, res) => {
    // get the specified trip
    try {
        // grab the id from the url
        // if the url was www.example.com/trips/123
        const id = req.params.id;

        // get the trip from the database by callin
        const trip = await tripData.readTrip(id);

        // render the trip page with the trip data
        // res.render takes in 2 parameters
        // 1. the name of the file to render
        // 2. the data to pass to the file, in obje
        // this means that in the trip.handlebars f
        res.render('trip', { trip: trip });
    }
    catch (e) {
        // if there is an error, return a 404 statu
        // a res.render automat[illegible]
        // you could also res.r (property) error: a
        res.status(404).json({ error: e });
    }
});

module.exports = router;
```

Why Not Always GET Routes?

- URL parameters are good for specific things, like ids or a username
- They should not hold sensitive information, such as a password
- If the user wants to change 5 things about their trip, directing them to www.example.com/change/these/five/things/about/my/trip gets boring quickly
- Requests that want to send a lot of information, or that hold sensitive information, should use a POST route instead

POST Route

(the code for this is too horizontally long to paste here, just view 6_postTrip.js)

This post route will create a new trip when given some information

- Get all of the information from the *body* of the request
  - a **GET** request doesn't have a useful body - we can modify the body of **POST** requests
- Once we have it, we can run our database function to create a trip which needs a user id, destination, start date, and end date
- Now that we created the trip, we can *redirect* the user to the page for the newly created trip

When to Use Routes

- You should have a route for each of your database functions
- On the frontend, you’ll be able to call each of your routes, and in turn then be able to (indirectly) call each of your database functions
- This is why we planned our database structure first, then go in this order of construction where we build our database functions, then build our routes which use those database functions, and then build our frontend which uses those routes
- Because we’re using **Handlebars.js**, routes are also used to *render* frontend pages, which is what the next lecture covers