Final Checkpoint Study Guide: EXPRESS

Express Docs: https://expressjs.com/en/4x/api.html#app

EXPRESS : INTIAL SETUP

Making middleware match (verb & URI)
App.use, app.all
Importing and syncing the db
Sending static files
Error handling

```
const express = require('express');
const path = require('path');
const morgan = require('morgan'); //logging the requests and responses
const bodyParser = require('body-parser'); //parsing the req/res

//require in the main db file
const db = require('./db')

const app = express(); //creating the link to express

//starting up the server
const server = app.listen(8080, ()=> console.log('listening on port 8080'));
// app.listen() takes the port number and a callback function

//calling sync on db will allow for the connection to be made.
db.sync().then() => console.log("database is sycned")

//app.use means for all incoming requests use these routes

app.use(morgan('dev')) // logging the middleware

//how to serve up static files
app.use(express.static(path.join(__dirname, '..', 'node_modules')))

// body parsing middleware
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// 'API' routes all backend routes sends to the index.js in the api folder.
app.use('/api', require('./api'));
```

```
31
32 // 404 middleware
33 app.use((req, res, next) =>
34   path.extname(req.path).length > 0 ?
35     res.status(404).send('Not found') :
36     next()
37 );
38
39 // error handling endware
40 app.use((err, req, res, next) =>
41   res.status(err.status || 500).send(err.message || 'Internal server error.')
42 );
43
```

EXPRESS-SEQUELIZE : ACCESS INFORMATION FROM REQUEST

GET VS PUT VS DELETE VS POST
req.body vs. req.params vs req.query
Only PUT and POST have a req.body GET and DELETE do not!
req.params : part of our URI in the request and in our middleware (:postId)
req.query: part of the URI but not in middleware (?dks:dlks);
sending status and sending responses
error handling

```
1  const router = require('express').Router();
2  module.exports = router;
3
4  //for redirecting to another file with / routes use:
5
6  router.use('/nextroute', require('./nextroute')
7
8  //router.VERB('/URI', callback function)
9  router.get('/', function(req,res,next)){
10   Model.findAll()
11   //gets the information and sends a response using json
12   .then(posts => res.json(posts))
13   .catch(next);
14 }
15
16 //GET = retriving information
17 router.get('/posts/:postId', function(req,res,next)){
18   const postId = req.params.postId
19   //the id is coming from the req that was sent in
20   Model.findById(postId)
21   .then(post => res.json(post))
22   .catch(next);
23 }
24
25 //PUT = updating information
```

```
router.put('/post/:postId', function(req, res, next)){
    const postId = req.params.postId
    Model.findById(postId)
    .then(message => message.update(req.body))
    //after finding the post to edit you can call update on the instance and add in the req.body
    .catch(next)
}

//DELETE = deleting a row of information
router.delete('/post/:postId', function(req,res,next)){
  //find the instance first then delete it
  Model.findById(postId)
  .then(foundPost => {
    return foundPost.destroy();
  })
  .catch(next);
}

//can also delete this way

router.delete('/people/:peopleId', function(req,res,next)){
  //find the instance first then delete it
  Model.destroy({where: {id:peopleId}})
  //send a status of 202 if it worked
  .then( ()=> { res.sendStatus(202); } )
  //send a message of error when it fails
  .catch(err => { res.status(err.status).send(err.message)});
}

//POST = creating a new row
router.post('/posts', function(req,res,next){
  Model.create(req.body)
  .then( post => res.json(post).status(201).end() )
  //if doing just res.status() need a .end() after
  .catch(next);
} )
```