EXPRESS.JS

Routes & REST

BUT FIRST...

POP QUIZ

CLIENT

Something that makes (HTTP) requests

SERVER

Something that responds to (HTTP) requests

REQUEST

A formatted message sent over the network by a client.
Contains VERB, URI (route), headers, and body.

RESPONSE

A server's reply to a request (formatted message).
Contains headers, payload, and status.

REQUEST-RESPONSE CYCLE

*The client **always** initiates by sending a request, and the server completes it by sending **exactly one** response*

EXPRESS MIDDLEWARE

A function that receives the request and response objects of an HTTP request/response cycle.

EXPRESS MIDDLEWARE

A function that receives the request and response objects of an HTTP request/response cycle.

function(req, res, next){…}

EXPRESS MIDDLEWARE CAN…

- Execute any code (such as logging) **then** move to the **next** middleware function in the chain
- Modify the request and the response objects **then** pass them to the **next** middleware function in the chain
- End the request-response cycle (E.g. **res.send**)

EXPRESS ROUTER

EXPRESS ROUTER

- **Express provides a Router middleware to create modular, mountable route handlers.**
- **Think of it as a “mini-app” that nests within an existing app.**
- **It lets you break up the major parts of your application into separate modules.**

App.js

```
const express = require("express");
const morgan = require("morgan");
const client = require("./db");
const postList = require("./views/postList");
const postDetails = require("./views/postDetails");

const app = express();

app.use(morgan("dev"));
app.use(express.static(__dirname + "/public"));

app.get("/", async (req, res) => {
  const data = await client.query("SELECT...");
  res.send(postList(data.rows));
});

app.get("/posts/:id", async (req, res) => {
  const data = await client.query("SELECT ...);
  const post = data.rows[0];
  res.send(postDetails(post));
});

const PORT = 1337;

app.listen(PORT, () => {

  console.log(`App listening in port ${PORT}`);

});
```

App.js

```
const express = require("express");
const morgan = require("morgan");


const routes = require("./routes");

const app = express();

app.use(morgan("dev"));
app.use(express.static(__dirname + "/public"));
app.use(routes);

const PORT = 1337;


app.listen(PORT, () => {
  console.log(`App listening in port ${PORT}`);
});
```

routes.js

```
const express = require('express');
const router = express.Router();
const client = require("./db");
const postList = require("./views/postList");
const postDetails = require("./views/postDetails");

app.get("/", async (req, res) => {
  const data = await client.query("SELECT...");
  res.send(postList(data.rows));
});

app.get("/posts/:id", async (req, res) => {
  const data = await client.query("SELECT ...);
  const post = data.rows[0];
  res.send(postDetails(post));
});
module.exports = router;
```

App.js

```
const express = require("express");
const morgan = require("morgan");


const routes = require("./routes");

const app = express();

app.use(morgan("dev"));
app.use(express.static(__dirname + "/public"));
app.use(routes);

const PORT = 1337;
app.listen(PORT, () => {
  console.log(`App listening in port ${PORT}`);
});
```

routes.js

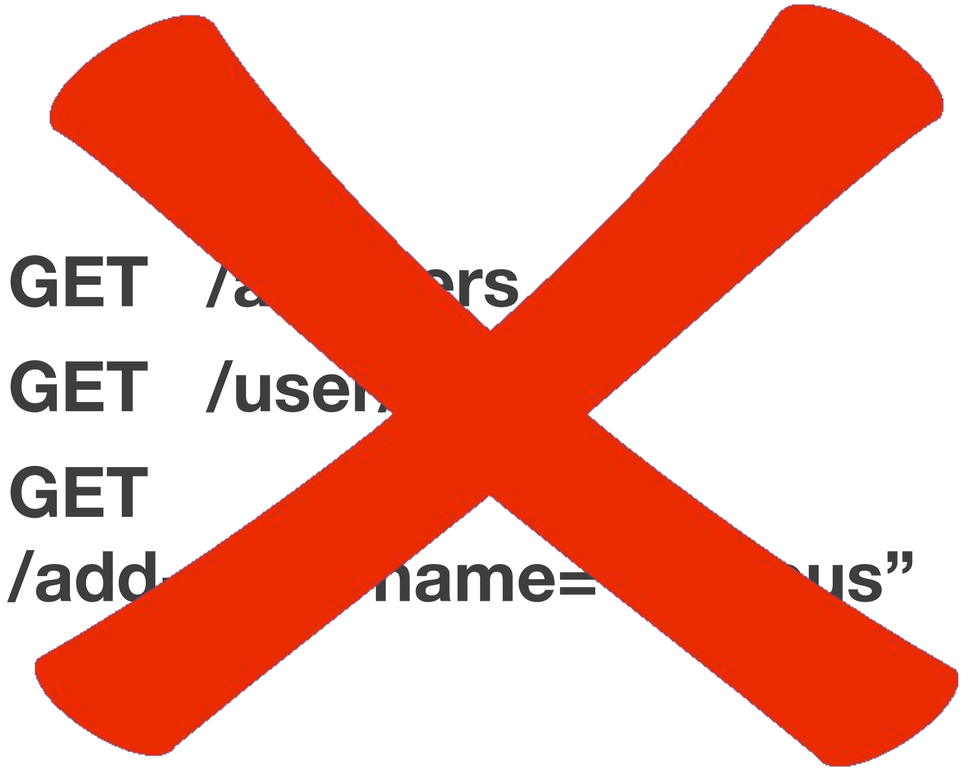
```
const express = require('express');
const router = express.Router();
const client = require("./db");
const postList = require("./views/postList");
const postDetails = require("./views/postDetails");

router.get("/", async (req, res) => {
  const data = await client.query("SELECT...");
  res.send(postList(data.rows));
});

router.get(“/posts/:id", async (req, res) => {
  const data = await client.query("SELECT ...);
  const post = data.rows[0];
  res.send(postDetails(post));
});
module.exports = router;
```

REST

REST

- **Architecture style for designing backend applications.**
- **Helps answer the question on how to organize routes and how to map functionality to URIs and Methods:**
  - Paths represent "nouns" or *resources*
  - HTTP “verbs” map to data operations

REST - RESOURCES



REST - RESOURCES

GET	/users	Show all users
GET	/users/4	Show a single user (whose ID=4 in the db)
POST	/users	Create a new user in the DB
PATCH	/users/4	Update user 4 in the db
DELETE	/users/4	Delete user 4 from the db

App.js

```
const express = require("express");
const app = express();
app.use(morgan("dev"));
app.use(express.static(__dirname + "/public"));

app.use('/posts', require('./routes/posts'));
app.use('/users', require('./routes/users'));

const PORT = 1337;


app.listen(PORT, () => {
  console.log(`App listening in port ${PORT}`);
});
```

posts.js

users.js

```
const express = require('express');
const router = express.Router();
const client = require("./db");

router.get("/", async (req, res) => {
  const data = await client.query("SELECT...");
  res.send(postList(data.rows));
});

router.get("/:id", async (req, res) => {
  const data = await client.query("SELECT ...);
  const post = data.rows[0];
  res.send(postDetails(post));
});
module.exports = router;
```

REQUEST BODY & BODY PARSER

- **POST, PUT (and PATCH) HTTP requests can contain information in the body**
- **The request body is streamed and frequently compressed**
- **Express comes with a built-in middleware that automatically parses incoming request bodies and makes the data available under** req.body

BODY PARSER

verb route

headers

In express…

```
request.body = {bookId:12345, author: 'Nimit'}
```

body

BODY PARSER

```
const express = require('express');

app.use(express.urlencoded({ extended: false }));
```