Node/Express Wrapup

Springboard

Express

Serving Static Files

Can serve static HTML, CSS, images, etc:

```
// serve files in `/js` directory as `/js/___`

app.use("/js", express.static('js'));
```

Templating HTML

Pug is a popular template system

Unlike Jinja/Nunjucks, you don't write HTML — you write simpler text:

```
doctype html
html(lang="en")
  head
    title= pageTitle
  body
    h1 Pug - node template engine
    #container.col
      if youAreUsingPug
        p You are amazing
      else
        p Get on it!
```

Common Security Fixes

Helmet

Provides tools for dealing with CSRF and other concerns

Authentication/Login

Passport.js

Provides common pattern for authentication

Also provides login via Facebook, Twitter, etc

Dealing with Cookies

```
const cookieParser = require('cookie-parser')

app.use(cookieParser())

app.get('/', function(req, res, next) {
  console.log('Cookies: ', req.cookies)
})
```

Can also sign cookies, to make tamper-free cookies

Other Node Web Frameworks

Koa2

Koa2

- Written by original author of Node
- A bit more modern & opinionated
- Not as popular as Express — yet!

Sails

Sails

- Larger, more opinionated framework
- Similar to Django or Ruby on Rails
- Includes ORM, ***Waterline***

Node

Popular Library: Moment

Moment.js

Convenient functions for date manipulation & conversion

Provides "humanized" dates, like "a few minutes ago", "yesterday"

Popular Library: Validator.js

Validator.js

Popular library of string validators:

- is all uppercase?
- is email?
- is URL?
- and so on

Popular Library: Lodash

Lodash

Useful set of small utility functions for common actions on arrays, objects, functions

Grouping, filtering, transforming, and more!

npm Scripts

package.json can define scripts to run:

```
{
  "scripts" :
  {
    "test": "jest",
    "debug": "nodemon --inspect server.js",
  }
}
```

Can then run like `npm test`

Other Common Data Stores

MongoDB

- A non-relational database *(often called NoSQL)*
  - Stores data as objects, not in tables
- Useful for unstructured data or recursive data
- More difficult to enforce integrity and join data together
- Good for large-scaling data where there isn't much interconnectedness

Note: MongoDB Blog Post

Check out Michael's blog post to get up and running with MongoDB.

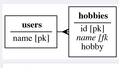
Redis

- "Key/Value" store
  - Like a simple 2-column table
- Can be extremely fast and easy to scale
- Doesn't have much security, transactions, integrity… by design
  - This helps make it fast & scalable
- Often used for "server-side caching"
  - Sometimes in front of a more traditional database

PostgreSQL

- Nice try — we already know PostgreSQL!
- Oh, but there's so many awesome things left!

Querying Relationships




```
CREATE TABLE users (name TEXT PRIMARY KEY);

CREATE TABLE hobbies (id SERIAL PRIMARY KEY,
                      user_name TEXT REFERENCES users,
                      hobby TEXT);

INSERT INTO users VALUES ('elie'), ('matt');

INSERT INTO hobbies (user_name, hobby) VALUES
    ('elie', 'dancing'),
    ('elie', 'javascript'),
    ('matt', 'math'),
    ('matt', 'cooking');
```

If we want `{name, hobbies: [hobby, ...]}` …

- You could write a query and make the nested JSON in JS
- Or you could tell PostgreSQL to do it!

```
SELECT name, json_agg(hobby) AS hobbies
FROM users AS u
  JOIN hobbies AS h ON (u.name = h.user_name)
GROUP BY name;
```

name	hobbies
elie	["dancing", "javascript"]
matt	["math", "cooking"]

Websockets

- We've used Node/Express to deal with HTTP requests
- It can also serve HTTPS
  - Though, typically, that's handled elsewhere by DevOps
- It can also serve "websocket" protocol
- HTTP is a pretty wordy, heavy protocol
  - So many things in headers!
- HTTP is stateless
  - Ask for answer, get answer, hang up connection
- Websockets are tiny and stateful — they stay connected!
  - They're often used for "tell the browser something has changed"

In Client

```
const ws = new WebSocket('ws://localhost:3000/chat');

ws.onopen = function(evt) {
  // called when browser connects to server
};

ws.onmessage = function(evt) {
  // called when browser receives a "message"
  console.log("got", evt.data);

ws.onclose = function(evt) {
  // called when server closes connection
}
```

to send a message to server

```
ws.send("this is a message from browser");
```

In Server

Library ***express-ws*** makes Websockets act like other routes

app.js

```
const wsExpress = require("express-ws")(app);

app.ws("/chat", function (ws, req, next) {
  ws.on("open", function () {
    // called when connection is opened
  });

  ws.on('message', function (data) {
    // called when message is received from browser
  });

  ws.on('close', function () {
    // called when browser closes connection
  });
});
```

to send a message to client

```
ws.send("this is a message from server");
```

Goodbye, Node?

Nope

This is the end of our time with backend JS

But we'll see that React apps are often made using Node — to setup project, run tests, run dev server, etc