WEB SERVICES

Practical Definition:

- Web Service - web call that returns data for programmatic use
- Endpoint - the URL for that call

SERVICES AREN'T PAGES

The difference is purely in how it is used.

Not a code difference.

Pages return HTML intended for the browser.

Other endpoints return CSS, JS, images, media.

Service endpoints can return HTML fragments, text, JSON, XML, YAML, etc

Services used by frontend and/or backend

SERVICE CONVENTIONS

- What do you send?
  - Url
  - Parameters
  - Headers
- How do you send it?
  - HTTP Method
  - Body/URL Parameters
- What do you get back?
  - Body
  - Status
  - Headers

ANYTHING GOES

- Have an endpoint
- Send params to define a command and what to do
- Return a 200 status code
- Return a message body that contains results or error message

This is common…but not very good

Why?

SOAP

A more defined version of anything goes, but with a lot of XML

- benefits of XML (schemas)
- drawbacks of XML
- real control at the data assembly/parsing, not at the HTTP layer

GRAPHQL

- Send a "query" using a query language
- describe what data you want
- describe what format (data structure) you want

REST

Nuanced system, often done partially

Three Basic rules:

- URL represents a "resource" (to interact with)
- HTTP Methods are the interaction
  - GET: Read
  - POST: Create
  - PUT: Replace
  - DELETE: Delete
  - PATCH: Update
- HTTP Status code represents the RESOURCE

MODERN DAY

- SOAP - rare
- GraphQL - new and rapidly growing
- REST - most common
- JSON most common data format

We will do REST for the class

We will return JSON

BASIC REST EXAMPLE

```
const people = {};

app.get('/people/', (req, res) => {
  res.json(Object.keys(people));
});

app.get('/people/:name', (req, res) => {
  const name = req.params.name;
  if(people[name]) {
    res.json(people[name]);
  } else {
    res.status(404).json({ error: `Unknown user: ${name}`});
  }
});
```

`.json()` does `JSON.stringify()` AND sets the `content-type` header

MORE REST EXAMPLE

```
app.post('/people/', express.json(), (req, res) => {
  const name = req.body.name;
  if(!name) {
    res.status(400).json({ error: "'name' property required" });
  } else if(people[name]) {
    res.status(409).json({ error: `already exists: ${name}`});
  } else {
    people[name] = req.body;
    res.sendStatus(200);
  }
});
```

`express.json()` middleware requires `content-type` of `application/json` on INCOMING requests, populates `req.body`

No `content-type`, no `body` value

CONSIDERATIONS

- Query params for filters
- No body in GET, so all parameters in query
- When POST data doesn't contain identifier, need to return unique identifier
- How to manage security? (more later)
- Long running requests need to have a polling setup
- Identifying errors well-enough
- Versioning of services!