REST

INFO 253A: Frontend Web Architecture

Kay Ashaolu

REST

Representational State Transfer

Good news, everyone!

- You already know REST
- Representations (HTML! but also XML, JSON)
- State
- Transfer

History

- Roy Fielding co-wrote HTTP specs
- Defined REST in his 2000 PhD dissertations
- Defined core set of constraints and why they were important

Constraints

- **Client-server:** Two separate systems talk to each other through a well defined interface
- **Stateless:** No context is stored between requests
- **Cacheable:** Clients or intermediaries can cache results, and requests and results can specify caching information

Constraints

- **Layered:** Requests can go through intermediaries (proxies)
- **Uniform Interface:** The protocol between client and server follows the same rules regardless of the specific application

Client-Server

Pros

- Browsers don't care what web server is providing representations, or which database is holding data
- Servers don't care which clients are connecting

Cons

- Overhead of transferring data
- Fewer, simpler failure modes

Stateless

Pros

- Simplifies server design and storage
- Simplifies request grammar
- Improves scalability, error recovery

Cons

- Overhead of transferring client state
- Not convenient for interactivity at protocol level

Cacheable

Pros

- Browsers can store CSS and JavaScript
- Businesses can cache responses, even from external sites
- Servers can specify how long things should be cached for

Cons

- Cache invalidation is hard
- Can't rely on updated resources updating "everywhere"

Uniform Interface

Pros

- Client and server know how to interact regardless of application hosted
- Pinterest uses same interface as Yelp
- Wider variety of clients that can handle multiple applications

Cons

- For really unique applications, must jam into old paradigms
- Difficult to optimize for performance of single application

REST is not HTTP

- Remember HTTP is a transport protocol: a tube!
- REST is a set of constraints on how to use that tube
- We could use other tubes, like FTP, SMTP

Web is RESTful

- Web is build on these ideas
- Better leverage attained by embracing REST
- Flexibility, scalability, visibility, simplicity

How to Spot RESTfulness

- Should think through constraints, but here are some heuristics

Uses HTTP

- REST is the underlying architectural principle of the web
- The web primarily uses the HTTP protocol
- The way browsers interface with web servers is inherently RESTFul if you think about it

Uses HTTP Commands

- GET, POST, PUT...
- vs using POST for everything

Uses HTTP response codes

- 404 Not Found, 200 OK
- vs. always responding with 200 OK but has an error message

URLs point to resources

- /blog, /api/messages/34
- vs. URLs pointing to commands: /api/createBlog, /api/getMessage/34

Representation links

- A representation links to new possible actions
- Client only needs to have representation
- Hypermedia as the engine of application state (HATEOAS)

Example

```
{
        "business": "http://yelp.com/biz/27",
        "user": "http://yelp.com/user/5",
        "review_text": "..."
}
```

Counter Example

```
data = {
        "business_id": 27,
        "user_id": 5
        "review_text": "..."
}

fetch(`http://yelp.com/biz/${data["business_id"]}`).then((response) => {
  ...
});
```

Uses headers for meta data

- Content-Type XML or JSON
- vs. response has extra meta data in XML

Questions?