REST API

Class 41

API

- acronym API: application programming interface
- a contract or specification for a connection between two programs

- very similar to a function prototype in C++:
  `string& foo(const vector<string>& bar, unsigned bam);`

- this prototype tells you that to call foo:
  1. foo must receive a vector of strings (which won't be changed) and an unsigned integer
  2. foo will provide a reference to a string

API

- an API system is a server that provides resources
- a system that uses the API is a client and requests resources from the server

- a major purpose of an API is to hide all implementation details
- it exposes only the interface details
- even if the internal implementation later changes, the interface will remain the same
- using an API reduces the coupling between the server and its client

Common APIs

- google maps: `https://developers.google.com/maps/documentation/javascript/`
- IBM Watson: `http://developer.ibm.com/watson`
  an AI engine that can do amazing stuff
- twitter: `http://dev.twitter.com/`
- facebook: `http://developers.facebook.com/`

REST

- acronym REST: representational state transfer
- Roy Fielding, 2000, an API for web applications
- meaning: a conversation between client and server involves the transfer of a representation of the state of a resource
- the state representation is almost always in JSON format

- when the client wishes a resource, it must give the server two pieces of information
  1. the unique identifier of the resource; this is a URL (uniform resource locator)
  2. the operation you wish to perform on the resource; an HTTP action: GET, POST, PUT, DELETE

GET

- I have written an API to allow you to interface with the words system
- it maintains a collection of words

- if you want to know the current state of a particular word, you might send a GET request to the URL: `.../api/abate_verb`
- the server would respond with the JSON object

```
{
  "word": "abate",
  "part": "verb",
  "definition": "to put an end to"
}
```

PUT and POST

- if you want to create a new resource, you might send a PUT request to the URL: `.../api/abacus_noun`
- and send the JSON object

```
{
  "definition": "a calculating device"
}
```

- or a POST request to `.../api/` with the JSON

```
{
  "word": "abacus",
  "part": "noun",
  "definition": "a calculating device"
}
```

PUT vs POST

- POST creates a new resource and adds it to a collection
- if an identical resource already exists, then a duplicate is created

- PUT creates a new resource if it doesn't exist
- but if it does exist, then PUT updates it

DELETE

- if you want to delete a particular word from the collection, you would send a DELETE request to the URL: `.../api/abate` with no JSON

- all these requests assume you have logged in or otherwise have permissions to perform the requested action

URL Rewrite

- the problem is that `.../api/abacus_noun` does not exist

- instead there's a program such as `.../api/add_word.php`
- we need to route the incoming request to the correct program to handle it
- RESTful web frameworks build routing tables automatically during compilation

- but we can easily rewrite the incoming URL manually

URL Rewrite

.htaccess

```
RewriteEngine On

# if the method is GET, redirect to get_words.php
RewriteCond %{REQUEST_METHOD} =GET
RewriteRule . get_words.php [L]

# if the method is DELETE, redirect to delete_word.php
RewriteCond %{REQUEST_METHOD} =DELETE
RewriteRule . delete_word.php [L]

# if the method is PUT, redirect to update_word.php
RewriteCond %{REQUEST_METHOD} =PUT
RewriteRule . add_word.php [L]

# everything else is an error
RewriteRule . error.php
```