jQuery and AJAX for Responsive Web Sites

Lecture 5
APIs and Security

Objectives

- Recap
- AJAX and History
- APIs
- Security

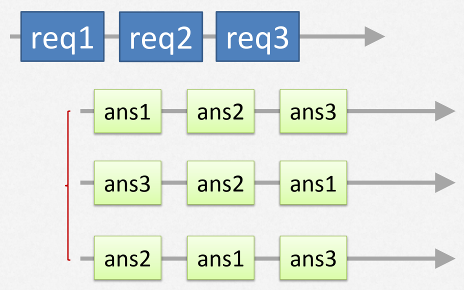
Previously on WEBLAMP 441…

Same Origin Policy

$\underbrace{\text{http://}}\underbrace{\text{niekdev.foo.com}}:\underbrace{\text{8080}}$

- Security
- JSON-P
- XMLHttpRequest2
- Proxy Server

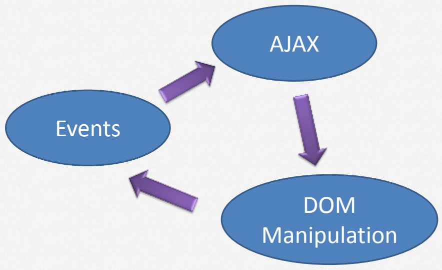
Asynchronous Call Flows



AJAX Failure Handling

- Retry
- Track unsaved
- Failure callback

Feedback Loop



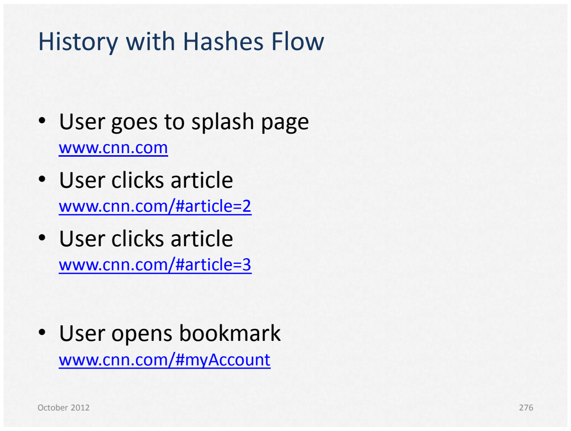
paradise?

paradise?

AJAX +
Browser History

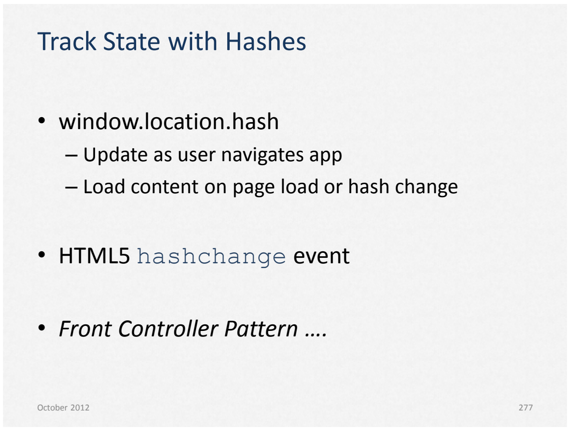
< Broken Back Demo >

< Hash Mark Demo >



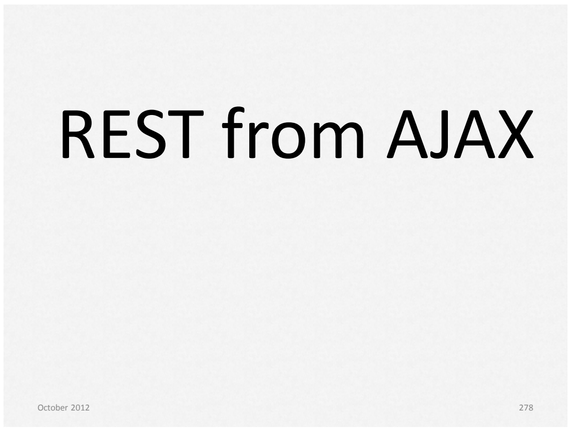
History with Hashes Flow

- User goes to splash page
  www.cnn.com
- User clicks article
  www.cnn.com/#article=2
- User clicks article
  www.cnn.com/#article=3
- User opens bookmark
  www.cnn.com/#myAccount

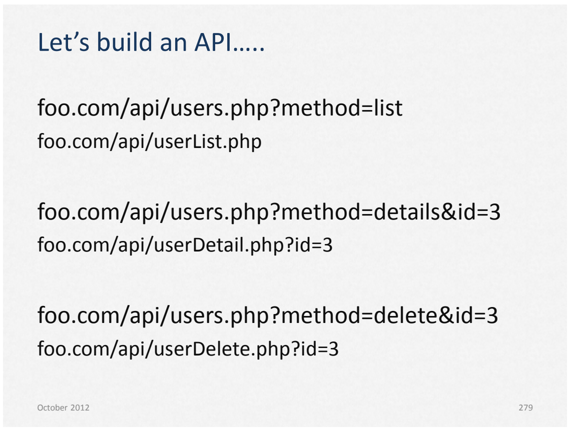


Track State with Hashes

- window.location.hash
  – Update as user navigates app
  – Load content on page load or hash change
- HTML5 `hashchange` event
- *Front Controller Pattern ….*



REST from AJAX

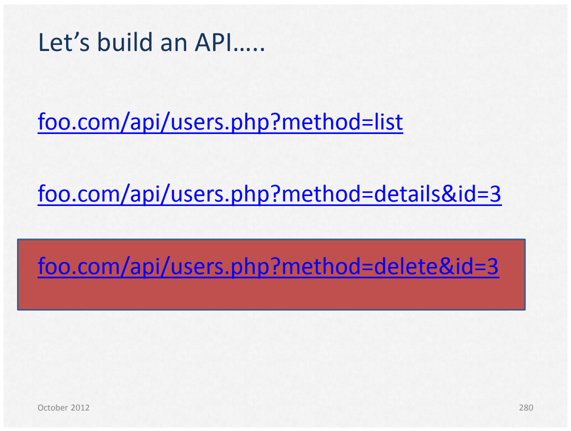


Let's build an API…..

foo.com/api/users.php?method=list
foo.com/api/userList.php

foo.com/api/users.php?method=details&id=3
foo.com/api/userDetail.php?id=3

foo.com/api/users.php?method=delete&id=3
foo.com/api/userDelete.php?id=3

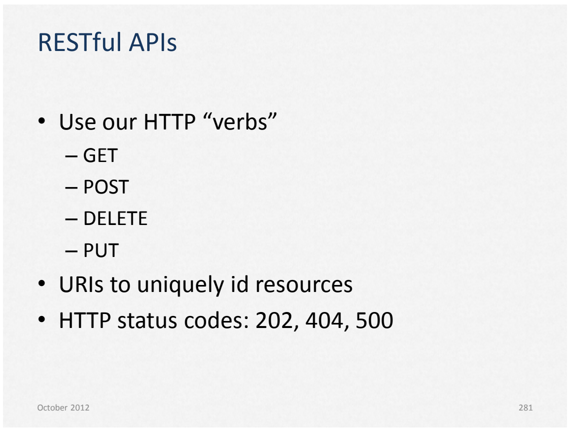


Let's build an API…..

foo.com/api/users.php?method=list

foo.com/api/users.php?method=details&id=3

foo.com/api/users.php?method=delete&id=3



RESTful APIs

- Use our HTTP "verbs"
  – GET
  – POST
  – DELETE
  – PUT
- URIs to uniquely id resources
- HTTP status codes: 202, 404, 500

GET vs. POST

GET	POST
• Does not mutate state • Browser may cache • Passed via URL • Size limits	• MUTATES state • Browser may **not** cache • Passed in Body

Let's build an API…..

foo.com/api/users.php?method=list
GET foo.com/api/users/

foo.com/api/users.php?method=details&id=3
GET foo.com/api/users/3/

foo.com/api/users.php?method=delete&id=3
DELETE foo.com/api/users/3/

< Github Demo >

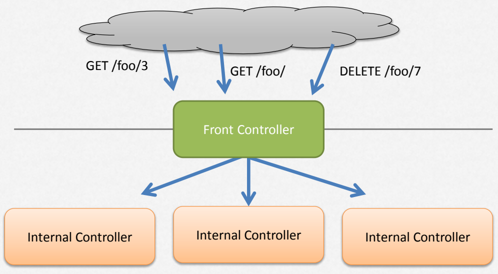
RESTful API Upside

- Uniformity
  - Easy to learn new APIs
  - Easy to write general purpose API libs
- Proper semantics with browser caching
  - POST for state mutations

RESTful API Downside

- Limited to four verbs
  - Semantic mismatch
  - "turn on printer"
- HTTP error codes
  - Semantic mismatch

Front Controller

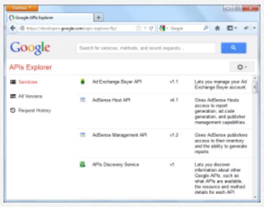


Front Controller

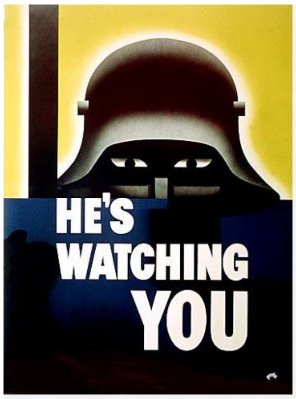
- Lets you break URI -> filesystem relation
- mod_rewrite
- Can steal similar idea for JavaScript and history management

Frontend API Resources

Programmable Web
http://tinyurl.com/nykjl



Google APIs Explorer
http://tinyurl.com/ctes4c2



security

Injection Attacks

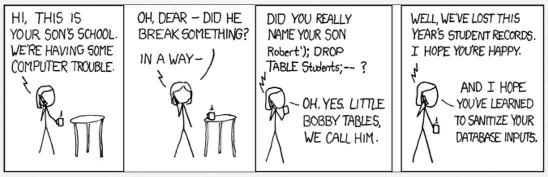

HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Image from XKCD

SQL Injection (1)

```
function addStudent( stuName ) {
    sql = “INSERT INTO students (name) VALUES (“ +
            stuName + “)”;
    execute( sql );
}
```

SQL Injection (2)

stuName:	Bill
SQL:	INSERT INTO students (name) VALUES (‘Bill’);

```
function addStudent( stuName ) {
    sql = “INSERT INTO students (name) VALUES (‘“ +
            stuName + “’)”;
    execute( sql );
}
```

SQL Injection (3)

stuName: Robert'); drop table students; --
SQL: INSERT INTO students (name)
VALUES ('Robert');
DROP TABLE students; --')

```
function addStudent( stuName ) {
      sql = "INSERT INTO students (name) VALUES ('" +
            stuName + "')";
      execute( sql );
}
```

SQL Injection (4)

- SQL parameter binding
- Whitelisting
- Escaping functions

SQL Injection - PHP

~~mysql_foo~~

Use mysqli or PDO instead.

More SQL Injection!

```
function getUserDetails( $myId ) {

      $sql = "SELECT name, email
             FROM students
             WHERE id=:id";

      $stmt = Db::getDb()->prepare( $sql );
      $stmt->bindParam( ':id', $myId, \PDO::PARAM_INT );
      $stmt->execute();

      ……
}
```

http://api.foo.com/myAccount?id=100

Image from Wikipedia

http://api.foo.com/myAccount?id=101

Image from Wikipedia

< StackOverflow User Demo >

SQL Injection

- Binding Syntax
- Whitelisting
- Business logic, sanity checks

JavaScript Injection!

User: here is my user/pass
Server: storage and id for session
Server: set id as cookie in response

User: requesting page, here is cookie
Server: yummy! Here is response
or….
Server: cookie invalid/expired! Login!

< document.cookie Demo >

< Steal Cookie Demo >

Client Side – Incoming Validation

- Don’t allow JavaScript
- Injection vectors…..
  - URLs
  - User-submitted content
- Scrubbing server side
  - HTML Purify
- Scrubbing client side
  - $(‘<li></li>’).text(userContent);

Client Side - Outgoing Validation

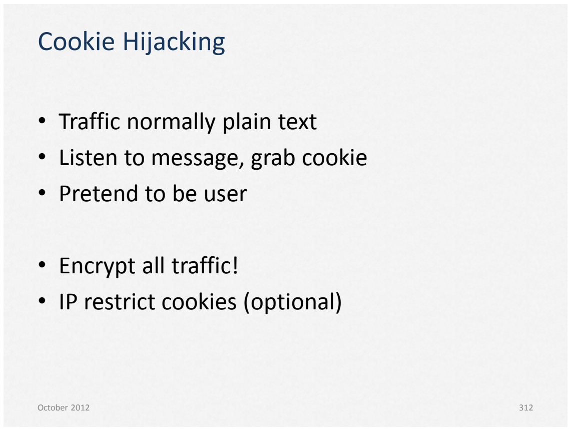
- Account creation
  - Is email address taken?
  - Is password strong enough?
- Improve UX, not security
  - Avoid round-tripping
  - < Twitter signup example >

Client Side - Misc

- HttpOnly Cookies

http://www.php.net/manual/en/function.setcookie.php

?



Cookie Hijacking

- Traffic normally plain text
- Listen to message, grab cookie
- Pretend to be user

- Encrypt all traffic!
- IP restrict cookies (optional)

October 2012 312