RESTful APIs

- We’re implementing what is called a RESTful API
- ReST stands for “representational state transfer”
- The term was coined in 2000 by Roy Fielding, who at the time was PhD student at UCI.

Some basic REST ideas:
- ✗ Client (browser here) needs to know only a single URL to access the resource (photoBooth app); further interactions are learned as it goes along
- ✗ Server does not need to know or remember anything about state of client

Browser’s view

- Browser
  - HTTP request: URL of page on server
  - First thread
  - Runs HTML, CSS, Javascript until either waiting for user input or needs more server data...

Which will it be for photoBooth?

Browser’s view

- Server at that URL
  - HTTP response: First Web page

Browser’s view

- Server with URL
  - HTTP request: First Web page
  - HTTP response: JSON data

Browser’s view

- Needs more server data... sends request, sets callback and goes to sleep

Browser’s view

- Server with URL
Browser's view

When will it wake back up?

ZZZ...

Server with URL

Browser's view

Callback wakes up and runs, changing DOM, until ready for user input, then goes to sleep.

HTTP response containing JSON

Second thread

Browser's view

Button action called, maybe when user deletes a label

URL requesting label delete

Third thread

Browser's view

... requests deletion, sets callback, goes back to sleep

Third thread

Browser's view

ZZZ...

Server with URL

When will it wake back up?
The world is full of potential clients.

Server's view

Server sends back response object that is self-addressed to browser.

Server's view

The server remembers nothing about the transaction.
Server's view

HTTP request for JSON data

Server

Database

Browser

Server's view

Puts together DB request, sets up callback and calls db.all()

Second thread

Server's view

ZZZ...

The server callback remembers response object

Browser

Database

Server's view

Callback wakes up, gets data from DB, puts it into response object body...

DB response

Third thread

Server's view

Sends-off HTTP response

Third thread

Server's view

ZZZ...

Server remembers nothing about transaction

Browser

Database
Last time

- DB operations using SQL
- One server thread might fire off multiple DB operations
- Order in which DB operations complete is not necessarily the order in which they are issued.
- When calling two DB operations in the same thread you can serialize them using `db.serialize()`.
- We'll see a more typical approach in a bit.

DB operations in different threads

- Can't be serialized.
- On a production server, many HTTP requests might be in process at once.

Production server

- Many callback-HTTP response object pairs waiting for their DB responses

Database state

- No way to order these operations.
- Best we can hope for is to ensure that database is always in some meaningful valid state, e.g. we don't have half a label written when it is read by someone else.
- Database systems work hard to ensure this.

Debugging – see database

```bash
amanto@cs189h$ sqlite3
...Use "open FILENAME" to reopen on a persistent database.
sqlite> .open photos.db
sqlite> select * from PhotoLabels;
  > hula.jpg|Dance, Event, Hula, Folk Dance|0
  > eagle.jpg|0
  > redwoods.jpg|0
sqlite> .quit
```
- You can also change items using "update", etc.

Debugging

- SQLite3 is nasty to debug, run-time errors just crash, do not tell you where it failed
- Always handle errors and print something on the error callbacks, gives you a fighting chance at figuring out what is going on!
Editing the labels string

- Need to read it, then write it.
- Chain together the two operations, using their callbacks to guarantee sequence

Parse request, set up "getCallback", construct DB command to get current labels

SELECT – get labels

DB response containing data

Run "getCallback", change string, ...
...set up update callback, request DB updates.

DB update command

Database

Editing the labels string

Editing the labels string

...set up update callback, request DB updates.

DB update command

Database

Editing the labels string

Editing the labels string

Fill in response object, tell browser all OK.

Response all OK

Database

...set up update callback, request DB updates.

DB update command

Database

Editing the labels string

Editing the labels string

Fill in response object, tell browser all OK.

Response all OK

Database

Editing the labels string

Editing the labels string

Fill in response object, tell browser all OK.

Response all OK

Database

Editing the labels string

Editing the labels string

Fill in response object, tell browser all OK.

Response all OK

Database

Editing the labels string

Editing the labels string

Fill in response object, tell browser all OK.

Response all OK

Database

Parsing the query

var querystring = require('querystring');
function answer(query, response) {
    queryObj = querystring.parse(query);
    if (queryObj.op == "add") {
        var newLabel = queryObj.label;
        var imageFile = queryObj.img;
        if (newLabel && imageFile) {
            ...
        }
    } else {
        HTTP response all OK
    }
Call DB to get current labels

```javascript
db.get( "SELECT labels FROM photoLabels WHERE fileName = ?", [imageFile],
getCallback);

□ Using “fill in the blanks” Node.js SQLite3 syntax
```

First callback

```javascript
□ Defined inside “answer” so it has access to all local variables – because it’s in the closure of “answer”!

function getCallback(err,data {
    ...handle error...
    db.run("UPDATE photoLabels SET labels = ? WHERE fileName = ?", [data.labels +", "+newLabel, imageFile],
    updateCallback); })
```

Second callback

```javascript
□ Also defined inside “answer”

function updateCallback(err) {
    ...handle error...
    response.status(200);
    response.type("text/plain");
    response.send("added label "+newLabel+" to "+imageFile); }

□ Uses Express syntax, could also have used Node
```

Module for queries

```javascript
□ Putting all the server code in one file will get messy
□ The code for answering queries will get big
□ Let’s put it in it’s own module
□ To make a file into a module add lines at the end to tell it to export stuff

// function answer visible from outside as “answer”
exports.answer = answer;
```

Using the module in server file

```javascript
□ Put pathname to file containing module as name of module

var queries = require("./queries");

□ Function to answer queries is now visible in tripleThreatServer as:
queries.answer(request,response);
```