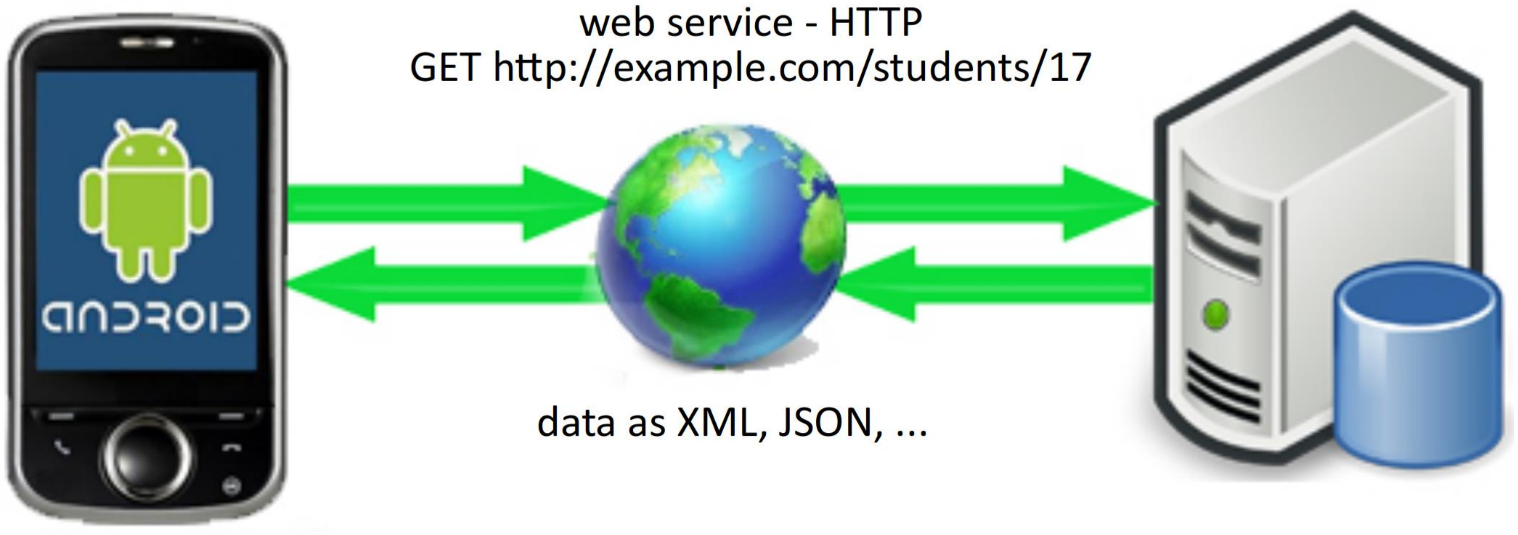
CSC 4210
Intro to Mobile App Dev.

Week of 4/3/18

Lecture 13

Using Web Services to access data

- Many apps access data through a web layer.
- **Client** (app) makes queries by contacting certain specific URLs.
- **Server** (web URL) sends the appropriate database data back.
- Client parses the data, displays it, etc.

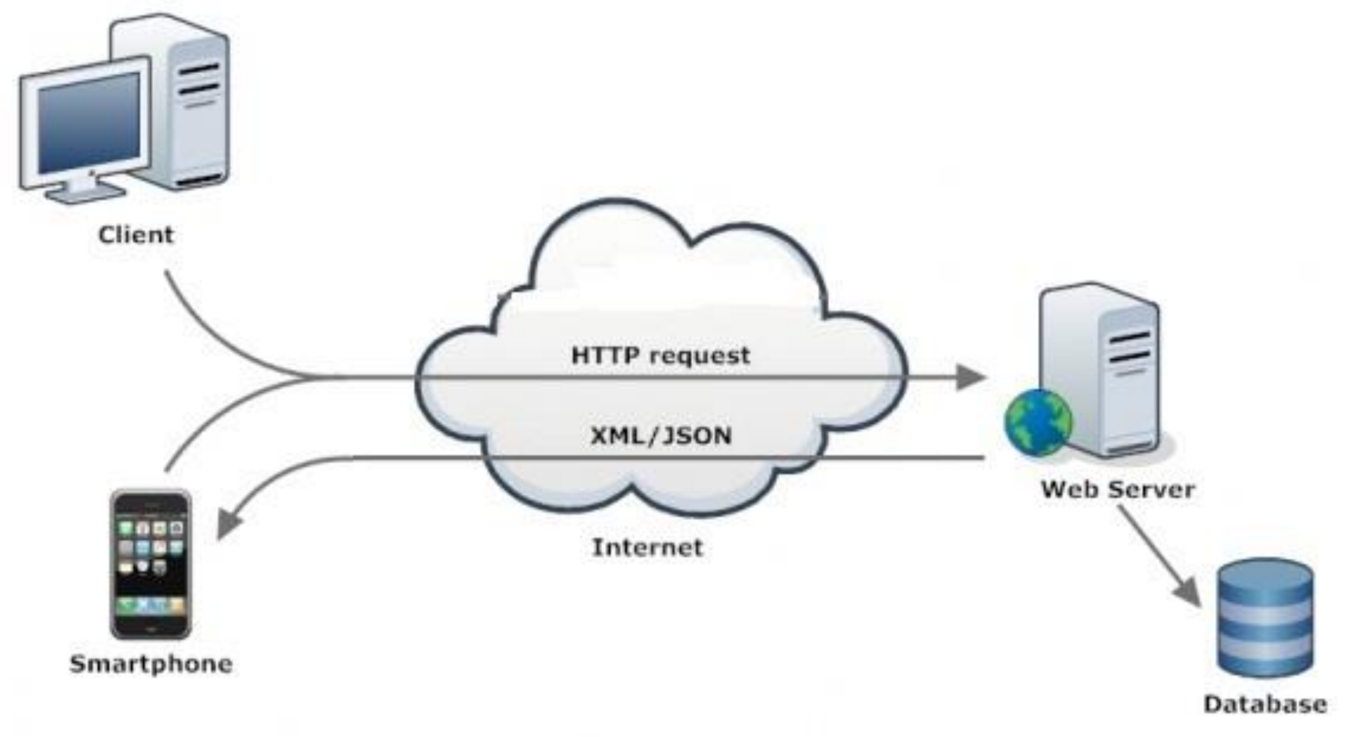


Web Service

- **web service:** a set of functionality offered over a server using the web, but not web pages / HTMLUse the web's HTTP protocol to connect and transfer data.
- Client connects to specific URLs to request specific data, which is then sent back in some documented format such as XML or JSON.
- **REST:** Representational State Transfer. Common style of web services.
  - "RESTful web services" or "RESTful APIs"

Web Service

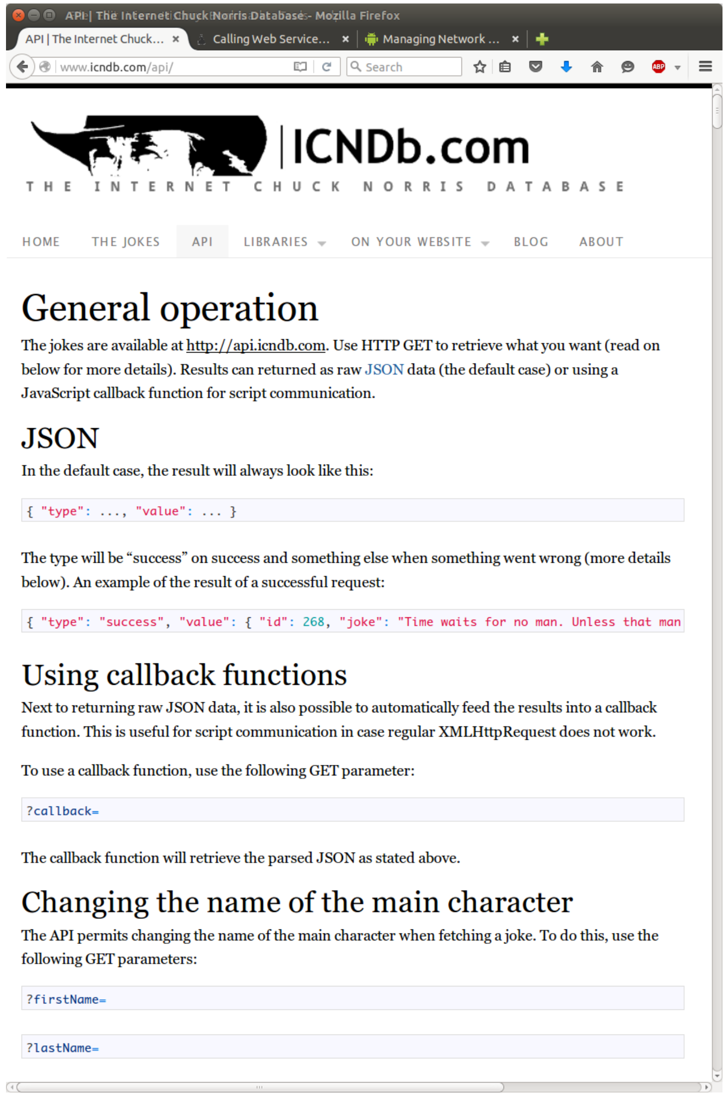
- Web services are a bit like remote function calls where you can request data via URLs with parameters and get the data returned as a response.



Rest WebService

Locating and Using webAPI’s

- **Locate** them online
  - Google for phrases like "<*company*> REST API" or "<*service*> free API"
- **Sign up** for an account
  - Many web APIs require a login or API key
  - Register to receive key or account
- Read the online **documentation** to find out how the API works
  - APIs are not standardized; each one is completely unique
  - Need documentation to learn the available services, parameters, etc.



General operation

The jokes are available at http://api.icndb.com. Use HTTP GET to retrieve what you want (read on below for more details). Results can returned as raw JSON data (the default case) or using a JavaScript callback function for script communication.

JSON

In the default case, the result will always look like this:

```
{ "type": ..., "value": ... }
```

The type will be “success” on success and something else when something went wrong (more details below). An example of the result of a successful request:

```
{ "type": "success", "value": { "id": 268, "joke": "Time waits for no man. Unless that man
```

Using callback functions

Next to returning raw JSON data, it is also possible to automatically feed the results into a callback function. This is useful for script communication in case regular XMLHttpRequest does not work.

To use a callback function, use the following GET parameter:

```
?callback=
```

The callback function will retrieve the parsed JSON as stated above.

Changing the name of the main character

The API permits changing the name of the main character when fetching a joke. To do this, use the following GET parameters:

```
?firstName=
```

```
?lastName=
```

Data Formats

- Most web APIs return their data in one of these formats:
  - JSON: JavaScript Object NotationData is a JavaScript object literal.
  - JS objects are basically maps from keys to values.
  - All values in the data are the fields of the object.
  - Object can contain sub-objects, lists, strings, numbers, etc.
  - Slightly less capable than XML, but simpler to read, write, parse.
  - Currently most popular web data interchange format for most apps.
- XML: Extensible Markup LanguageData is a nested tree of tags and attributes.
  - More structured, but bulkier/harder to parse.
  - Very popular 5-10 years ago but being superseded by JSON.
- Some web APIs use other data formats:
  - YAML: Yet Another Markup Language. Popular in Ruby/Rails community.
  - plain text

JSON example

```
{
  "private": "true",
  "from": "Alice Smith (alice@example.com)",
  "to": [
    "Robert Jones (roberto@example.com)",
    "Charles Dodd (cdodd@example.com)"
  ],
  "subject": "Tomorrow's \"Birthday Bash\" event!",

  "message": {
    "language": "english",
    "text": "Hey guys, don't forget to call me this weekend!"
  }
}
```

JSON example

```
 ↙ {...} = object document
{   ↓ key / ↓ value pairs
  "private": "true",   ← boolean
  "from": "Alice Smith (alice@example.com)",   ← string
  "to": [   ← [] denotes an array
    "Robert Jones (roberto@example.com)",   ← array element 0
    "Charles Dodd (cdodd@example.com)"      ← array element 1
  ],
  "subject": "Tomorrow's \"Birthday Bash\" event!",

  "message": {   ← {...} = a nested object
    "language": "english",
    "text": "Hey guys, don't forget to call me this weekend!"
  }
}
```

Chuck Norris REST API

- fetches random Chuck Norris quotes and "Facts" in JSON format
  - http://www.icndb.com/api/
  - figure login/key required? NO
- API: http://api.icndb.com/ __________
  - /jokes/random - fetch a random joke
    - { "type": "success", "value": { "id": 194, "joke": "Chuck Norris kicked cancer.", "categories": [] } }
  - /jokes/random/N - fetch multiple random jokes
    - { "type": "success", "value": [{ "id": 417, "joke": "..." }, { "id": 505, "joke": "...", "categories": ["nerdy"] }, { "id": 291, "joke": "...", "categories": [] }] }

Chuck Norris REST API

- /jokes/random/limitTo=[categories] - limit categories of joke
- /jokes/random/exclude=[categories] - exclude categories of joke
- /jokes/N - fetch a specific joke with ID #N
  - { "type": "success", "value": { "id": 194, "joke": "Chuck Norris kicked cancer.", "categories": [] } }
- /jokes/count - fetch total number of jokes
  - { "type": "success", "value": 549 }
- /categories - fetch names of all categories of jokes
  - { "type": "success", "value": ["nerdy", "explicit", "chuck norris", "bruce schneier"] }

Parsing JSON data

```
{
  "private": "true",
  "from": "Alice (alice@ex.com)",
  "subject": "Today's event",

  "to": [
    "Robert (roberto@ex.com)",
    "Charles (cdodd@ex.com)"
  ],

  "message": {
    "lang": "english",
    "text": "Call this weekend!"
  }
}
```

```
private void processData(String data) {
try {
  // extract the information from JSON data
  JSONObject json = new JSONObject(data);
→ boolean private = json.getBoolean("private");
→ String from = json.getString("from");
→ String subject = json.getString("subject");

→ JSONArray a = json.getJSONArray("to");
→ String to1 = a.getString(0);
→ String to2 = a.getString(1);


→ JSONObject msg =
→              json.getJSONObject("message");
→ String lang = msg.getString("lang");
  String text = msg.getString("text");

} catch (JSONException e) {
    Log.wtf("json", e);
}
}
```

Getting web data

```
public void fetchData(String urlString) {
Thread thread = new Thread(new Runnable() {
public void run() {
try {
URL url = new URL(urlString); // connect to the site
HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
conn.setConnectTimeout(30000); // milliseconds
conn.setReadTimeout(10000);
conn.setRequestMethod("GET"); conn.connect();
int responseCode = conn.getResponseCode(); // HTTP result
codes; 200=success
if (responseCode == HttpURLConnection.HTTP_OK) {
```

Getting web data (cont.)

```
InputStream input = conn.getInputStream(); // read data from URL
to string
StringBuilder sb = new StringBuilder();
 while (true) {
 int ch = input.read();
if (ch == -1) break;
sb.append((char) ch); }
 String text = sb.toString();
processData(text); // you write this! }
else {
Log.d("url", "HTTP fail, code " + responseCode); // request failed
} }
catch (IOException ioe) { Log.wtf("url", ioe); } } });
thread.start();
```

Updating widgets in thread

- Code in a **background thread** cannot modify UI widgets.
  - The code will throw an exception.
  - Widgets must be updated in Android's UI thread.
- Simplest way to update a widget (without libraries): call post method, pass a Runnable containing code to run:

```
// update a UI widget in a background thread
textView.post(new Runnable() {
public void run() {
textView.setText(joke);
}
 });
```

Getting web data, Ion library

```
// fetch REST API data in background with Ion library
public void fetchData(String urlString) {
Ion.with(context)
.load("urlString")
.asString()
.setCallback(new FutureCallback<String>() {
public void onCompleted(Exception e, String data) {
 // process the data or error
JSONObject json = new JSONObject(data); processData(json);
// you write this! } }); }
```

The cat API

- facts/photos of cats in **XML** / HTML
  - http://thecatapi.com/docs.html
  - login/key required? **OPTIONAL**
- API: **http://thecatapi.com/ ________**
  - **/api/images/get?***param***=***val***&***param***=***val* - fetch a random cat picture
    - **image_id=***ID* specific image ID
    - **format=***format* format of data to return: xml, html, or src [default xml]
    - **results_per_page=***N* number of images to send back [default 1]
    - **category=***cat* category of images [default none]
    - **size=***size* size: small, med, or full [default full]
    - example: http://thecatapi.com/api/images/get?format=xml&size=med&results_per_page=3

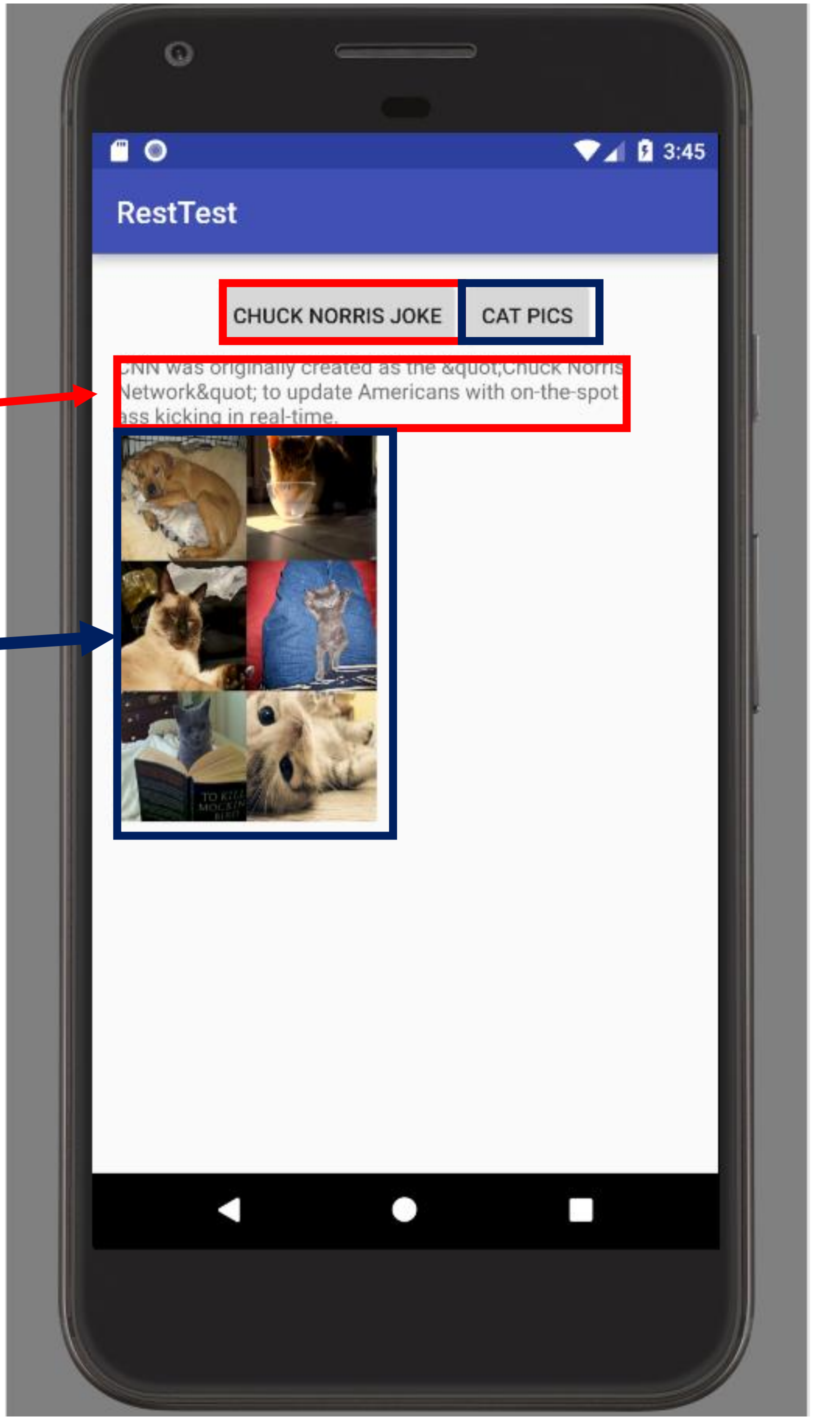
The cat API (cont)

- **/api/images/vote** - score an image from 1-10
  - **api_key=***key* API key (required)
  - **image_id=***ID* ID of image to vote on (required)
  - **score=***N* score from 1-10 (required)
  - example: http://thecatapi.com/api/images/vote?api_key=xxxxx&image_id=bC24&score=8
- **/api/images/getvotes** - return all votes made by your API key
  - **api_key=***key* API key (required)
- **/api/categories/list** - get list of all active image categories
  - { "type": "success", "value": 549 }

Cat/CN APP (**DUE 4/11**)

The app should have 2 buttons

1. Button prints random jokes about Chuck Norris (http://www.icndb.com/api/)
2. Button prints random cat pics to a grid (http://thecatapi.com/)



CNN was originally created as the &quot;Chuck Norris Network&quot; to update Americans with on-the-spot ass kicking in real-time.

Resources

- https://restfulapi.net/
- https://developer.android.com/reference/org/json/JSONObject.html
- https://developers.google.com/api-client-library/java/google-http-java-client/json
- https://developer.android.com/training/basics/network-ops/xml.html