University of Washington iOS Development

Ted Neward
Neward & Associates
http://www.tedneward.com | ted@tedneward.com

10 Nov 2015

Networking

Goals

Today we're going to...

... talk about HTTP (the lingua franca of the Internet)

... examine JSON (one of the data lingos of the Internet)

... examine XML (the other data lingo of the Internet)

... show how to use them from iOS

HTTP

Who, what, why...?

HTTP

- Part of the World Wide Web experiments
- Designed to be:
  - **simple**
  - **content-agnostic**
  - **platform-agnostic**
  - **referral (to another server) capability**
  - **minimal to zero administrative overhead**
  - **format negotiation**
  - **operate over standard infrastructure (TCP/IP)**

HTTP

HTTP has seen few revisions

- HTTP 0.9: Original proposal (1991) by TBL
- HTTP 1.0: 1991 - 1995 saw rapid growth/adoption
- HTTP 1.1: Internet standard status (95 - 99)
- numerous proposals to enhance 1.1 since

  nothing officially adopted (or even taking root)

HTTP

HTTP is now the de facto standard protocol of the Internet

- for better or for worse
- to the point of supplanting other more specific protocols
  - **FTP replaced by file transfer over HTTP**
  - **email protocols replaced by web clients and transfer**
  - **bidirectional sockets replaced by WebSockets (!)**

HTTP

HTTP is the protocol part of a REST system

- designed by Roy Fielding as his Doctoral dissertation
- explicit architectural goals
- attempts to shoehorn additional features into HTTP meet with mixed success

  and with Fielding's frustration and disdain

HTTP

For more on HTTP concepts

- see Fielding's dissertation

  http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- see Architecture of the World Wide Web

  http://www.w3.org/TR/webarch/
- both are highly recommended reads

HTTP/1.1 Protocol Details

Jumping into the Web pool

HTTP Protocol

HTTP/1.1: RFC 2616

- "application-level protocol for distributed, collaborative, hypermedia information systems"
- generic, stateless protocol
- allows for very easy extension
- typing and negotiation of data representation

HTTP Protocol

Dependencies

- TCP/IP, DNS

  underlying communication infrastructure
- URL and URI (RFCs 1738, 1630, 1808, 2396)

  target server, port and resource to request
- MIME (RFCs 2045, 2046, 2047)

  description of content formats
- TLS (SSL)

  secure transmission

HTTP Protocol

Basic details

- server listens on well-known port (80)
- client initiates communication
- client sends request packet, server sends response packet
- connection is closed after each send/receive cycle

  no state retained across cycles

HTTP Protocol

Quick note: stateless

- HTTP explicitly holds no server fidelity
  any server can answer any request
- this is what allows HTTP to scale so well
  the ubiquitous "web farm"
- browser cookies are NOT(!) part of the HTTP spec
  in fact, HTTP authors disdain the use of cookies
- making HTTP stateful in some way usually fails miserably

HTTP Protocol

Basic protocol notes

- all text is in "7-bit ASCII clean" format
  - **in other words, nothing above ASCII value 127**
- all text uses CRLF pairs to denote EOL
- client/server request/resonse protocol
  - **client always initiates**
  - **client blocks until server responds**
- packets are always single-line plus header/value pairs
  - **and optional content body**

HTTP Protocol

Request packet

```
GET / HTTP/1.1
Host: www.newardassociates.com
Accept: */*
```

HTTP Protocol

Response packet

```
200 OK HTTP/1.1
Content-Type: text/html
Content-Length: 32

<html><body>Howdy!</body></html>
```

HTTP Protocol

Request packet

- Request-Line: Method Request-URI HTTP-Version CRLF
  - **Method: the "verb"**
  - **Request-URI: the resource**
  - **HTTP-Version: "HTTP/1.1" or "HTTP/1.0"**

    other versions possible, never used
- (Optional) Header: Value CRLF
- CRLF
- (Optional) Content body

HTTP Protocol

Request methods

- GET: retrieve resource idempotently

  should have no side effects (cacheable results)
- POST: accept this sent relevant data
- PUT: store this data as the resource
- DELETE: remove the resources

HTTP Protocol

Request methods

- OPTIONS: describe verbs supported for the resource
- HEAD: GET without content body
- TRACE: diagnostic trace
- CONNECT: for use with a tunneling proxy

HTTP Protocol

Request-URI

- URL minus TCP/IP-related parts
  - no scheme (http://)
  - no server (www.google.com)
  - no port (:80)
- used to identify resource requested
- absolute resource path
  - not always a filesystem resource
  - ... though early and simple webservers do map URL paths to filesystem paths
  - doing so is dangerous: beware relative paths ("../../../etc/passwd")

HTTP Protocol

HTTP-Version

- this is the version the client wishes to use
- server will respond with its version in response
- client can either upgrade or downgrade as necessary
- in practice, this is almost always "HTTP/1.1"
  - **ten years ago, negotation between 1.0 and 1.1 was common**
  - **if we ever see an HTTP/2.0, negotiation will become important**

HTTP Protocol

Header: Value lines

- more on headers later
- each header line must be ended with CRLF
- each header describes one annotation/extension/ adaptation to the request
- request and response use same sets of headers
  - **a few are client- or server-specific, but not many**

HTTP Protocol

CRLF (empty line)

- empty line is mandatory
- server will block until it receives this second CRLF!
- separates Request-Line/Headers from Content body
- must be present, even with no headers or content body

HTTP Protocol

Content body

- entirely opaque to HTTP protocol
- we use headers to describe the content body

  Content-Type, Content-Length most common
- recipient then to read exactly that many bytes (no more, no less)

  failure to do this is a security hole!

HTTP Protocol

Response packet

- Response-Line: Status-Code Reason-Phrase HTTP-Version CRLF
- (Optional) Header: Value CRLF
- CRLF
- (Optional) Content body

HTTP Protocol

Status-Code

- quick integer description of server's results
- 1xx: Informational
- 2xx: Sucess
- 3xx: Redirect
- 4xx: Client error
- 5xx: Server error

HTTP Protocol

Reason-Phrase

- textual description of status-code

  usually purely for human consumption
- these are not standardized except de facto in a few cases
- 200: OK
- 404: Resource not found
- 401: Requires authorization
- 500: Internal server error

HTTP Protocol

HTTP-Version

- this is the version the client wishes to use
- server will respond with its version in response
- client can either upgrade or downgrade as necessary
- in practice, this is almost always "HTTP/1.1"
  - **ten years ago, negotation between 1.0 and 1.1 was common**
  - **if we ever see an HTTP/2.0, negotiation will become important**

HTTP Protocol

Header: Value lines

- more on headers later
- each header line must be ended with CRLF
- each header describes one annotation/extension/ adaptation to the request
- request and response use same sets of headers
  - **a few are client- or server-specific, but not many**

HTTP Protocol

CRLF (empty line)

- empty line is mandatory
- server will block until it receives this second CRLF!
- separates Request-Line/Headers from Content body
- must be present, even with no headers or content body

HTTP Protocol

Content body

- entirely opaque to HTTP protocol
- we use headers to describe the content body

  Content-Type, Content-Length most common
- recipient then to read exactly that many bytes (no more, no less)

  failure to do this is a security hole!

HTTP Protocol

Common headers

- Host: the host (and optional port) requested
  - **required by 1.1 for multitenant server scenarios**
- User-Agent: description field describing the client
  - **not required, but almost always included**
  - **this is how we determine client capabilities**

HTTP Protocol

Common headers

- Content-Type: MIME type of content being sent
- Content-Length: size (in octets/bytes) of content

  these two are required if content body is present

HTTP Protocol

Common headers

- Accept: specify certain media types to be acceptable
  - **comma-delimited list of MIME types**
- Accept-Encoding: describes content encodings
  - **used to allow for request/response gzip compression**
- Authorization: client sends to authenticate to server
  - **"Authorization: {credentials}"**
  - **credentials: scheme credential-data**
  - **where scheme is Basic, Digest, or others**

HTTP Protocol

Common headers

- Connection: state of the connection
  - **server most often sends "close" in 1.1 exchanges**
- WWW-Authenticate: required in all 401 responses
  - **contains a challenge that indicates the authentication scheme**
  - **"WWW-Authenticate: {challenge}"**
  - **challenge is typically either Basic or Digest**

HTTP Protocol

For more information

- Consult RFC 2616 for any remaining details

  official: https://www.ietf.org/rfc/rfc2616.txt
- Consult RFC 2324 for details on how to extend HTTP

  •Hyper Text Coffee Pot Control Protocol
  •official: https://www.ietf.org/rfc/rfc2324.txt

Additional, useful information

- Fielding's dissertation

  http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Architecture of the World Wide Web

  http://www.w3.org/TR/webarch/

Networking in iOS

How to reach out and ping somebody

iOS Networking

"It's just TCP/IP"

- true at most levels
- but there are a few caveats to consider
  - **power requirements**
  - **cellular network unreliability**
  - **data plan limitations**

iOS Networking

There's a couple of ways to approach this

- High-level: NSURLSession (preferred) or NSURLDownload
  - **high-level, URL-based**
  - **the workhorse of the lot**
- Low-level: Sockets/streaming
  - **choice between streaming (TCP) and packets (UDP)**
  - **each has their uses**
  - **both require a lot more work implementing the protocol**

iOS Networking

NSURLSession

- natively supports several schemes
  data, file, fto, http and https
- transparent support for proxies/gateways
- leverages user preferences settings

iOS Networking

NSURLSession workflow

- Create the NSURLSession
  - **one of four different types of session**
- Create a "task"/request
  - **requires an NSURL of some form**
- Execute the request

iOS Networking

Simple HTTP GET: Establishing the session

```
func httpGet(request: NSURLRequest!, callback: (String, String?) ->
Void) {
    let session = NSURLSession.sharedSession()
    let task = session.dataTaskWithRequest(request) {
         (data, response, error) -> Void in
         if error != nil {
             callback("", error!.localizedDescription)
         } else {
             let result = NSString(data: data!, encoding:
                                   NSASCIIStringEncoding)!
             callback(result as String, nil)
         }
    }
    task.resume()
}
```

iOS Networking

Simple HTTP GET: Executing

```
let request = NSMutableURLRequest(URL: NSURL(string: "http://
www.google.com")!)
httpGet(request) {
    (data, error) -> Void in
    if error != nil {
        print(error)
    } else {
        print(data)
    }
}
```

iOS Networking

HTTP POST w/authorization headers: Setup

```
let request = NSMutableURLRequest(URL: NSURL(string: "http://
localhost:4567/login")!)
let session = NSURLSession.sharedSession()
request.HTTPMethod = "POST"

let params = ["username":"fred", "password":"password"] as
Dictionary<String, String>

request.HTTPBody = try!
NSJSONSerialization.dataWithJSONObject(params, options:
NSJSONWritingOptions(rawValue: 0))
request.addValue("application/json", forHTTPHeaderField: "Content-
Type")
request.addValue("application/json", forHTTPHeaderField: "Accept")
```

iOS Networking

HTTP POST w/authorization headers: Executing

```
let task = session.dataTaskWithRequest(request, completionHandler:
{data, response, error -> Void in
        var strData = NSString(data: data!, encoding:
NSUTF8StringEncoding)
        do {
                let json =
                        try
NSJSONSerialization.JSONObjectWithData(data!,
options: .MutableLeaves)
                                        as? NSDictionary
```

iOS Networking

HTTP POST w/authorization headers: Validating

```
                // The JSONObjectWithData constructor didn't return
an error. But, we should still
                // check and make sure that json has a value using
optional binding.
                if let parseJSON = json {
                        // Okay, the parsedJSON is here, let's get
the value for 'success' out of it
                        var success = parseJSON["success"] as? Int
                        print("Succes: \(success)")
                }
                else {
                        // Woa, okay the json object was nil,
something went worng. Maybe the server isn't running?
                        let jsonStr = NSString(data: data!,
encoding: NSUTF8StringEncoding)
                        print("Error could not parse JSON: \
(jsonStr)")
                }
// {{## BEGIN usage-2 ##}}

// {{## BEGIN usage-3 ##}}
        }
        catch {
```

iOS Networking

HTTP POST w/authorization headers: Validating

```
        }
        catch {
                let jsonStr = NSString(data: data!, encoding:
NSUTF8StringEncoding)
                print("Error could not parse JSON: '\(jsonStr)'")
        }
})
// {{## BEGIN usage-3 ##}}

task.resume()
NSThread.sleepForTimeInterval(5.0)
print("End")
```

iOS Networking

Four NSURLSessionConfiguration options

- singleton shared session object
  - **basic requests**
  - **NSURLSessionConfiguration.sharedSession()**
- default sessions
  - **data obtained incrementally using delegate**
  - **NSURLSessionConfiguration.defaultSessionConfiguration()**
- ephemeral essions
  - **do not write caches, cookies, or credentials to disk**
  - **NSURLSessionConfiguration.ephemeralSessionConfiguration()**
- background sessions
  - **uploads/downloads while app is not running**
  - **NSURLSessionConfiguration.backgroundSessionConfiguration()**

iOS Networking

Three "task types":

- "data" tasks
  - **short, often interactive tasks**
- upload tasks
  - **support background uploads**
- download tasks
  - **retrieve data in the form of a file**

iOS Networking

NSURLSession

- works with your code one of two ways:
  - **a completion handler invoked on end**
  - **delegate methods invoked as incidents occur**

iOS Networking

NSURLSessionDelegate

- URLSession:downloadTask:didResumeAtOffset:expectedTotalBytes:
- URLSession:downloadTask:didWriteData:totalBytesWritten:totalBytesExpectedToWrite:
- URLSession:downloadTask:didFinishDownloadingToURL:

  required

iOS Networking

NSURLSessionTaskDelegate

- URLSession:task:didCompleteWithError:
- URLSession:task:didReceiveChallenge:completionHandler:
- URLSession:task:didSendBodyData:totalBytesSent:totalBytesExpectedToSend:
- URLSession:task:needNewBodyStream:
- URLSession:task:willPerformHTTPRedirection:newRequest:completionHandler:

iOS Networking

Particular case: NSURLDownload

- specifically for downloading a file to disk
- (since replaced by NSURLSession)
- does not wake the app up on download completion
- does not download in the background either

JavaScript Object Notation

Why, exactly...?

JSON Concepts

JSON: Data in JavaScript (RFC 4627)

- with the rise of JavaScript's acceptance came a desire to make it easy to send data to a JavaScript client
- JavaScript has an "object literal" notation, so we started making use of that
- this came to be known as "JSON"
- JSON has since become a format used in multiple contexts

  including databases (CouchDB, MongoDB, RavenDB, and more)

JSON Concepts

JSON benefits

- simple (for the most part)
- highly portable
- extensible
- (relatively) compact
  compared to XML

JSON Concepts

JSON warnings

- data only!
  - **JSON has no provision for including code**
- data is plain-text format
- data is limited to JavaScript types
  - **strings and numbers (and arrays of) for the most part**
  - **binary is not going to transmit well in particular**
- never use client-side eval() to parse a JSON document
  - **injection attacks await those who do**
- does not follow object references/graphs well
  - **becomes more of a hierarchical format than an object one**

JSON Format Details

What's legit JSON?

JSON Format

JSON file format

- always a plain-text format
- governed under ECMA 404 specification
- easily described

JSON Format

JSON Document

- a single object literal denoted by { and }

JSON Format

JSON Object

- object is a comma-delimited list of string : value pairs
  - **string is "-quoted field name**
  - **":" separates name from value**
  - **value is any JSON-acceptable value**

JSON Format

JSON value

- acceptable JSON values:
  - **strings**
  - **numbers**
  - **objects**
  - **arrays**
  - **true, false (boolean literals)**
  - **null**

JSON Format

JSON strings

- always "-quoted
- any UNICODE character (except ", \ or control characters)
- \ escapes sensitive characters

  \", \\, \/, \b, \f, \n, \r, \t, or \uXXXX

JSON Format

JSON numbers

- decimal formats only
- floating-point, exponential formats allowed
  - 1
  - 1.0
  - 1.7e14
  - negative forms of the above

JSON Format

JSON arrays

- denoted by [and]
- contain comma-delimited list of values

JSON Format

For more information

- see RFC 4627

  http://www.ietf.org/rfc/rfc4627.txt
- see JSON website (includes list of JSON parser libraries)

  http://json.org
- see ECMA 404 specification

  http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
- see JSON Lint (JSON validator)

  http://jsonlint.com/

JSON in iOS

Parsing and producing JSON on iOS

JSON in iOS

NSJSONSerialization

- convert Foundation objects to JSON
- convert JSON to Foundation objects
- ... assuming certain conventions
  - **top-level object is NSArray or NSDictionary**
  - **all objects are one of NSString, NSNumber, NSArray, NSDictionary or NSNull**
  - **all dictionary keys are NSString**
  - **Numbers are not NaN or infinity**

JSON in iOS

Parsing JSON

- NSJSONSerialization.JSONObjectWithData()
  - **options: NSJSONReadingOptions**
  - **(optional) error: (ObjC only)**
    in Swift, throws an exception instead
  - **returns an AnyObject (or nil on error)**
- NSJSONSerialization.JSONObjectFromStream()
  - **stream: NSInputStream (for direct access)**
  - **other parameters identical to above**

JSON in iOS

Producing JSON

- NSJSONSerialization.dataWithJSONObject()
  - **options: NSJSONWritingOptions**
  - **(optional) error: error object**
    in Swift, throws an exception instead
  - **retuns an NSData (or nil on error)**
- NSJSONSerialization.writeJSONObject()
  - **toStream: NSOutputStream to which JSON is written**
  - **other parameters identical to above**

JSON in iOS

Avoiding errors

- NSJSONSerialization:isValidJSONObject

  can object be converted to JSON

eXtensible Markup Language

Why, exactly...?

XML Concepts

XML

- domain-independent markup language
  - **simplified form of SGML**
  - **inspired by HTML's success in the late 90s**
  - **intended to be generic data markup language**
- since fallen a little out of favor

  but expectations were a little overinflated, too

XML Concepts

XML has/had 10 design goals:

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.

XML Concepts

XML has/had 10 design goals:

- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.

XML Concepts

XML is conceptually a multilayered concept

- the abstract data model is called the Infoset
  - **governed by its own spec (XML Infoset Spec)**
- the usual concrete representation is similar to HTML
  - **governed by the XML Specification (1.0, 1.1)**
- numerous attempts at a binary representation
  - **BinaryXML, EXI, Fast Infoset, ...**

XML Concepts

XML data model

- fundamentally, XML is a strictly hierarchical data model
  - **data is in nodes called "elements"**
  - **one root node ("document element")**
  - **nodes can contian name/value pairs ("attributes")**
  - **nodes can contain child nodes**
  - **nodes can contain raw text (character data)**

XML Concepts

XML levels of correctness

- an XML document that can be parsed correctly is called "well-formed"

  basically, it obeys the formatting specification
- an XML document that cannot be parsed is "ill-formed"
- an XML document that can be validated is called "valid"

  variety of different validation schemes; not baked in to XML itself
- documents can be well-formed but invalid

  it can be syntactically parsed, but fails semantic validation

XML Concepts

Elements

- elements are named nodes in the XML document
  - **names may be scoped using XML Namespaces**
- elements always have a parent element
  - **exception: the root element, a.k.a. document element**
- elements have 0..many child elements
  - **raw data will appear as child "text elements"**
- elements have 0..many attributes

XML Concepts

Attributes

- attributes are name/value pairs
- attributes always apply to a given element
  - **cannot have attributes without an element**
- attributes typically modify/annotate the element in some way

XML Concepts

Comments

- comments are like comments in any language
  - **they do not appear in the abstract data model**

XML Concepts

For more information

- See the XML 1.0 Specification

  http://www.w3.org/TR/REC-xml/
- See the XML 1.1 Draft Specification

  http://www.w3.org/TR/xml11/
- See the XML Infoset Specification

  http://www.w3.org/TR/xml-infoset/

XML 1.0 details

What makes an XML document legal?

XML Details

XML documents

- XML data is always considered to be a "document"
  - **even if it's never stored in a file or is never human-consumed**
- uses text for canonical representation
  - **alternative representations (binary) are possible, but not standardized**
- default representation looks very similar to HTML
  - **XML is a simplified SGML, and HTML is an SGML-defined markup language**

XML Details

XML basic rules

- markup start-tags and end-tags are angle-brackets ("<" and ">")
- certain characters are not legal except as markup characters
  - **ecape using "entity" syntax:**
  - **< => <**
  - **> => >**
  - **& => &**

XML Details

XML elements

- elements are named tags

  <foobar> is a "foobar" element
- names must be C-style identifiers

  alphanumeric + underscore, no whitespace, etc
- elements can either have explicit start and end tags...

  <foobar> ... </foobar>
- ... or use "shorthand" syntax if they are childless

  <foobar />

XML Details

MXL elements

- can have any number of child elements
  - **child elements started must also end nested inside this element**
  - **child elements are "owned" by this element**
  - **all elements have a parent element**
- can have any amount of raw data embedded inside element declaration
  - **technically these are "text elements" of no name and containing text values**
  - **whitespace may or may not be included, depending on parser details**

XML Details

XML document element

- one element is the root of the entire document
- this is called the document element
- only element that has no parent
- only one document element per XML document

  in essence, this is a singly-rooted tree

XML Details

Legal XML Document

```
<data />
```

Legal XML Document

```
<data><child1 /><child2>Mommy!</child2></data>
```

XML Details

XML attributes

- attributes are name/value pairs attached to an element
- attribute names are case-sensitive identifiers
  - **no whitespace in the name**
- attributes values are contained in quotes (single or double)
- attributes are not children of the element
- attribute names must be unique on the given element
- any number of attributes on a given element

XML Details

Legal XML Document

```
<data date="2/15/2015"><child1 /><child2>Mommy!</child2></data>
```

XML Details

XML comments

- comment nodes begin with <!-- and end with -->
- comments are entirely ignored by XML processing agents
- comments cannot appear inside an element tag
- comments may not nest
- comments may not include "--" anywhere in their body

  this is the most annoying restriction of all time

XML Details

Legal XML Document

```
<!-- Hello, XML data -->
<data date="2/15/2015">
  <!-- Ths is a child node -->
  <child1 />
  <!-- This is another child node -->
  <child2>Mommy!</child2>
</data>
```

XML Details

XML Processing Instructions

- PIs are "special" nodes as hints to the XML processing agents
- PIs do not appear in the abstract data model
- most common PI is the "XML declaration" at the top of an XML file

  declares XML version and character encoding, language, etc
- different PIs may appear for particular processing scenarios

  they will be entirely processing-agent-specific

XML Details

Legal XML Document

```
<?xml version="1.0" ?>
<!-- Hello, XML data -->
<data date="2/15/2015">
  <!-- Ths is a child node -->
  <child1 />
  <!-- This is another child node -->
  <child2>Mommy!</child2>
</data>
```

XML Details

XML CDATA sections

- escaping sensitive characters can get ugly in certain scenarios
  - **example: including XML inside XML (such as example code)**
- the CDATA section begins with <[CDATA[and ends with]]>
  - **anything contained inside here is treated as raw data and never markup**

XML Concepts

For more information

- See the XML 1.0 Specification

  http://www.w3.org/TR/REC-xml/
- See the XML 1.1 Draft Specification

  http://www.w3.org/TR/xml11/

XML in iOS

Producing and consuming XML on iOS

XML in iOS

Parsing XML in iOS: NSXMLParser

- this is a SAX-style parser
- designate a delegate to receive event methods
- fast for simple parsing
- complicated for more detailed XML documents

XML in iOS

Using NSXMLParser

- instantiate around the document to be parsed
- designate an NSXMLParserDelegate
- call parse()

XML in iOS

NSXMLParserDelegate: four methods of interest

- parser:didStartElement:...
  an opening element tag was parsed
- parser:didEndElement:...
  a closing element tag was parsed
- parser:foundCharacters:...
  character data was parsed
- parser:parseErrorOccurred:...
  something wrong

XML in iOS

iOS lacks a DOM-style parser

- several OSS options available
- SWXMLHash

  https://github.com/drmohundro/SWXMLHash
- AEXML

  https://github.com/tadija/AEXML
- GlimpseXML

  https://github.com/glimpseio/GlimpseXML

iQuiz

Part 3

iQuiz

iQuiz: A Multiple-Choice Q-and-A application

- users can choose from a collection of quizzes
- each quiz has a number (1-to-many) of questions
- each question is a multiple-choice answer
- users progress through each question one at a time
- app will track their answers
- app could upload their scores
- quizzes are updated from a server

iQuiz

Part 3: Network and storage

- all quizzes/questions should come from online
- store the quizzes/questions to local storage
- if offline, use locally-stored data
- initial URL to use:

  http://tednewardsandbox.site44.com/questions.json
- Settings should be a popover
  - **include a URL field to use instead of the above**
  - **include a "check now" button to retrieve**

iQuiz

Grading: 5 points

- download JSON from site: 1 pt
- store JSON to local storage: 1 pt
- use local storage offline: 2 pt
- refresh on Settings "check now": 1 pt

Extra credit:

- implement "pull to refresh": 1 pt
- central score storage:
  - **upload scores to central storage: 1 pt**
  - **add new toolbar button and popover for scores: 1 pt**
  - **change icon in quiz list to reflect score: 1 pt**

DUE: 1 week