





Mobile != Independent

- * Data sources for Mobile Apps
 - * Mapping, Google Now, email, weather etc.
- * Central data repository format
 - * Web-service
 - * Not for browsers
 - * Content – no mark-up or styles
 - * Deployment from...
 - * Corporate website
 - * Web Hosting
 - * Cloud

Most mobile apps which have some value are connected to a central data source
 Think of mapping, Google Now, email, weather etc.
 The app gets data that is delivered from some central resource

What form does a central data repository take?

Typically a web-service

i.e. a web presence that is not aimed at a browser

Typically provides content without mark-up, style or functionality – e.g.
 Google Finance provides raw data on share prices, weather sources
 provide very condensed weather info.

This can be deployed from a corporate website, a hosted site, a PC with a
 permanent web connection or a cloud service

Given that, many web apps are simple presentations of data-feeds

Weather app depicts location, temperature, rainfall probability and
 barometric pressure as a graphic of some weather condition



Web Servers and Web Services

- **Web Server**
 - A machine that serves web pages or data
- **Formats**
 - URL → HTML files
 - URL+Parameters → Active Server Pages
 - REST → URLs map to resources
 - Operations: Create, Retrieve, Update, Delete
- **Web Services**
 - Access points to online data systems
 - URL → represents a data record
 - For **machine access**
 - a website for automations
 - don't care about presentation or style

The diagram illustrates an Android client (labeled 'Android Client') connected via a green arrow to a cloud icon representing a 'Web Service'. Below the cloud, a server icon is shown with the text 'Application Client Service (Application Data)'.

A web server is a computer with a web connection that is configured to provide a response to web requests

A number of formats are available

Simplest – a URL maps to a single file (not very interesting)

More likely for a service – a URL + parameters accesses a server-side script which uses the parameters to look-up or compile some data to return

REST – each URL maps to a specific resource, which can be a file, or a *virtual resource* (e.g. info extracted from a database) REpresentational

State Transfer – each URL acts as a look-up key to a particular item of **state** – i.e. a specific piece of data)

Web Services are access points to online data systems

URL indicates the item (or collection) of data

Operation (in SOAP or REST format) indicates what is to be done (i.e. retrieve specified data, update specified data, add new data, delete existing data) – CRUD

Web services are for **machine access** – a website for automations that don't care about presentation or style

e.g. <http://mcm-share-prices-hrd.appspot.com/getprice?tickers=GOOG,AAPL>

This returned ['750.73', '449.83'] on 29th January 2013. The data is formatted as JSON (Javascript Object Notation) so can be accessed from a web-app

hosted at the same domain.



A mobile device can do almost everything a PC on the corporate network can

Acts as a terminal for accessing corporate data

e.g. Customer records for mobile sales staff

Instant job estimations – competitive quotes

Can upload data to add to corporate database

Instant invoicing

Reduced duplication of effort

More accurate information

The risks – security – is this a problem?

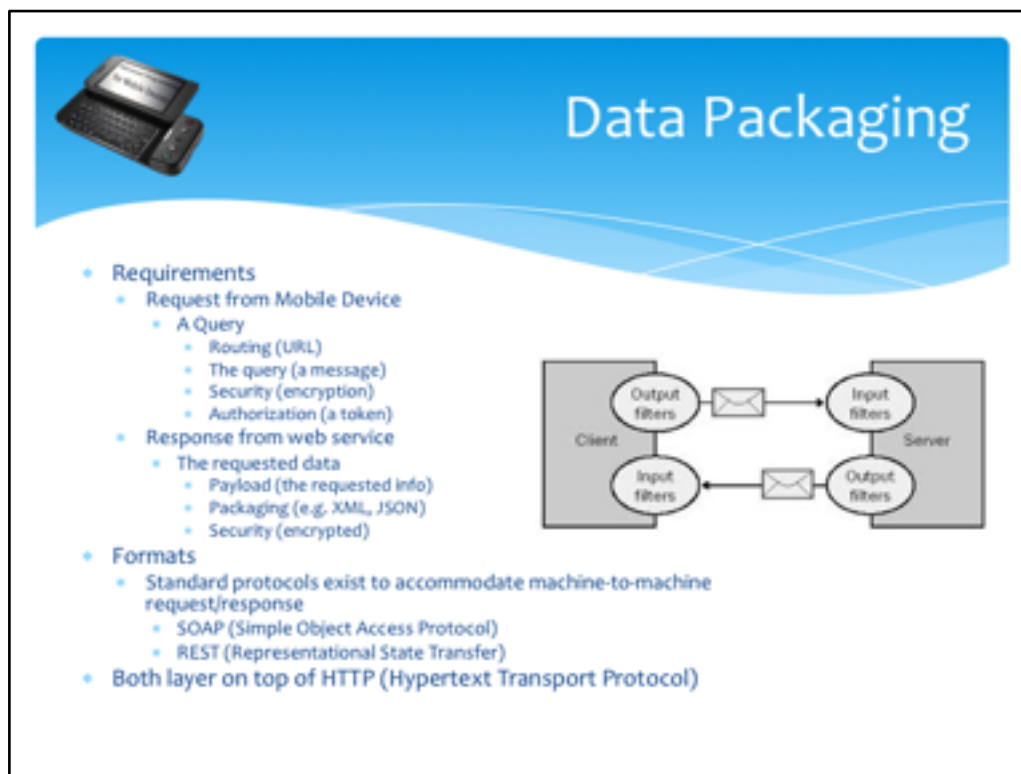
Only if the mobile access system is the same as the corporate LAN

Web services tend to be designed with security in mind

No direct access to data – access to data gateways

System design incorporates authorization

Can be short term, based on business model (e.g. authorize to generate a single quote)



In any exchange with a web server, there is a request (from the client) and a response (from the server)

Request needs – address, query, security, authorization (for a secure service)

Response needs to provide – the requested data in some agreed format (e.g. XML)

May include encryption (depends on sensitivity of information)

Web Services are defined in terms of *protocols* - essentially message format and addressing schemes.

Standard formats are SOAP and REST

SOAP – exchange of XML messages between client and web server

A SOAP package includes a message and an envelope

Envelope = Sender, Receiver, indication of end-points (URLs)

Can incorporate security (although HTTPS can do this for it)

Now losing favour due to data size inefficiency

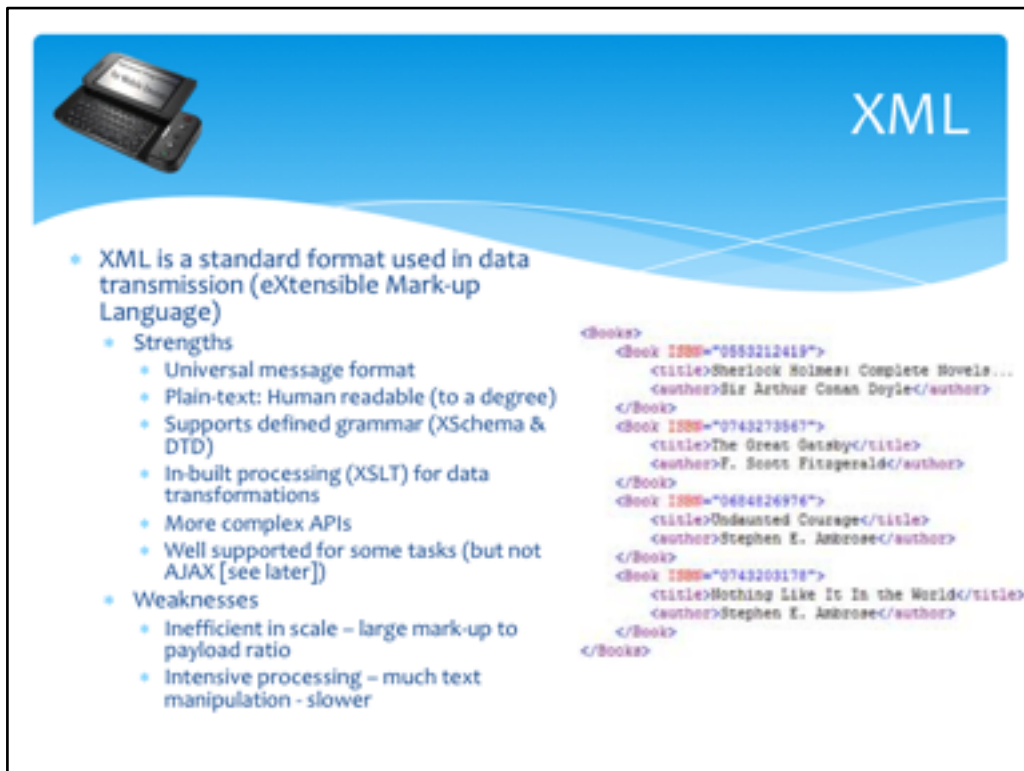
REST – URL includes all request data

service address (e.g. <http://corporate-site.com/datastore/>)

indication of requested resource (e.g. [/customers/1234567](#))

indication of operation (GET, PUT, POST, DELETE)

All operations implicit in HTTP



XML

- XML is a standard format used in data transmission (eXtensible Mark-up Language)
 - Strengths
 - Universal message format
 - Plain-text: Human readable (to a degree)
 - Supports defined grammar (XSchema & DTD)
 - In-built processing (XSLT) for data transformations
 - More complex APIs
 - Well supported for some tasks (but not AJAX [see later])
 - Weaknesses
 - Inefficient in scale – large mark-up to payload ratio
 - Intensive processing – much text manipulation - slower

```

<Books>
  <Book ISBN="0553212419">
    <title>Sherlock Holmes: Complete Novels...</title>
    <author>Sir Arthur Conan Doyle</author>
  </Book>
  <Book ISBN="0743273867">
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
  </Book>
  <Book ISBN="048626976">
    <title>Undaunted Courage</title>
    <author>Stephen E. Ambrose</author>
  </Book>
  <Book ISBN="0743203178">
    <title>Nothing Like It In the World</title>
    <author>Stephen E. Ambrose</author>
  </Book>
</Books>

```

XML is based on very simple rules...

A Document starts with the XML header...

```
<?xml version="1.0"?>
```

An XML Document is a tree of nodes with a SINGLE root element (e.g. Books)

An element can have any number of sub-elements, each can have sub-elements etc.

An Element has an opening (<Book>) and closing (</Book>) tag.

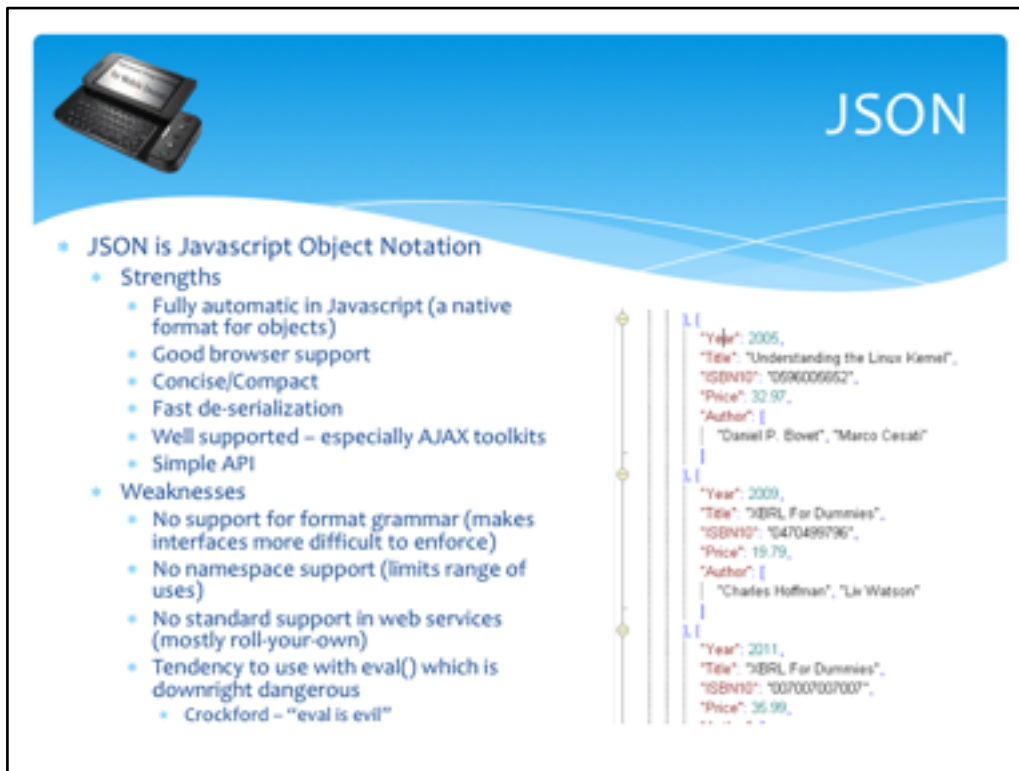
An element can contain other elements, or text (CDATA – Character Data)

An element can also contain Attributes. These are inside the opening tag, and give more information on the element (e.g. ISBN="0123455677").

Attribute values must be in quotes double "" or single '')

Tabs must be properly nested (<a>, not <a>)

Using these rules, document of any complexity can be built, and are inherently machine readable.



JSON

- JSON is Javascript Object Notation
 - Strengths
 - Fully automatic in Javascript (a native format for objects)
 - Good browser support
 - Concise/Compact
 - Fast de-serialization
 - Well supported – especially AJAX toolkits
 - Simple API
 - Weaknesses
 - No support for format grammar (makes interfaces more difficult to enforce)
 - No namespace support (limits range of uses)
 - No standard support in web services (mostly roll-your-own)
 - Tendency to use with eval() which is downright dangerous
 - Crockford – “eval is evil”

```

{
  "Year": 2005,
  "Title": "Understanding the Linux Kernel",
  "ISBN10": "0596006652",
  "Price": 32.97,
  "Author": [
    "Daniel P. Bovet", "Marco Cesati"
  ]
}

{
  "Year": 2009,
  "Title": "XBRL For Dummies",
  "ISBN10": "0470489796",
  "Price": 19.79,
  "Author": [
    "Charles Hoffman", "Li Watson"
  ]
}

{
  "Year": 2011,
  "Title": "XBRL For Dummies",
  "ISBN10": "007007007000",
  "Price": 35.99,
  ...
}

```

JSON is a simple notation based on standard Javascript data elements and collections.

A JSON document is organized as a literal object definition within a string...

```
{ "name" : "Fred Bloggs", "dob": "1970-08-22", "occupation" : "Programmer",
  "skills": ["Drinking Coffee", "Coding in Java", "Sleeping"] }
```

Attributes are name-value pairs.

Note that the last attribute here is an array.

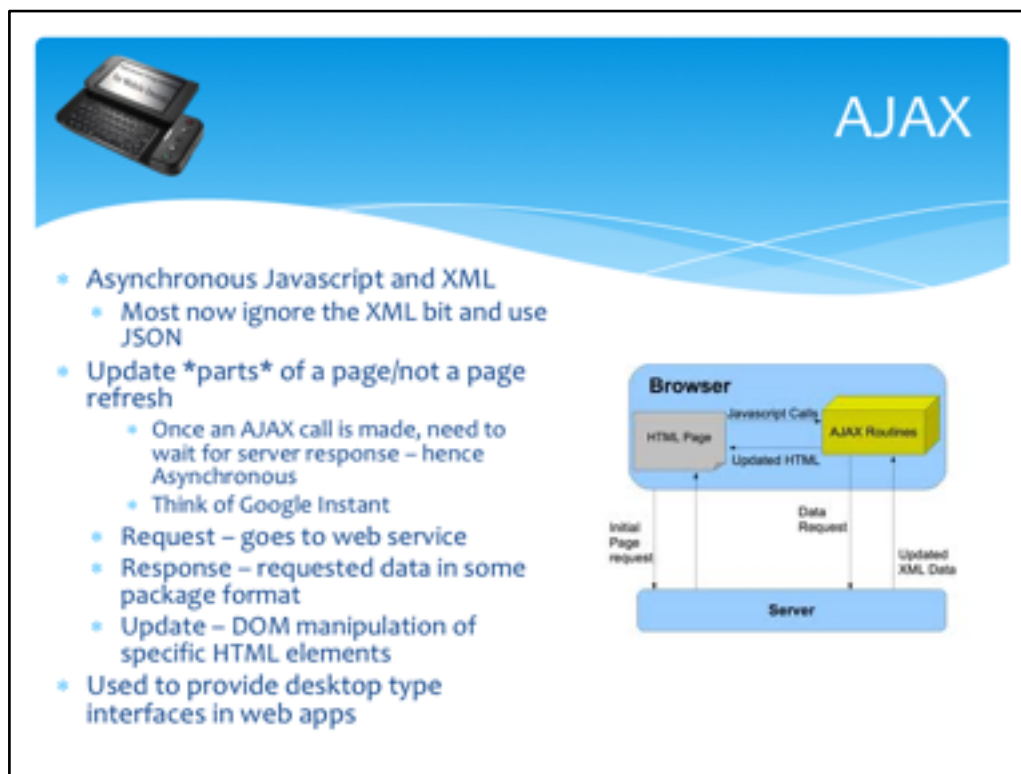
A JSON document can be 'stringified' – i.e. inserted into a string...

```
'{ "name" : "Fred Bloggs", "dob": "1970-08-22", "occupation" : "Programmer",
  "skills": ["Drinking Coffee", "Coding in Java", "Sleeping"] }'
```

Note that the outer quotes are different from the inner ones (doesn't matter whether we do '{"zx" : "y"}' or '{"x': 'y}')

jQuery's AJAX functionality (\$.ajax()) takes care of JSON automatically, converting Javascript object to JSON format for POST and PUT operations, and converting JSON responses into Javascript objects. If you implement raw AJAX (using the XMLHttpRequest object), you will need to transform POST and PUT data into JSON

using `JSON.stringify()` and retrieve Javascript objects from any response using `JSON.parse()`.



AJAX is now the standard methods for in-web-page updates.

Well supported by Javascript libraries (no need for it in other languages)
jQuery in particular makes a good job of it – also Dojo

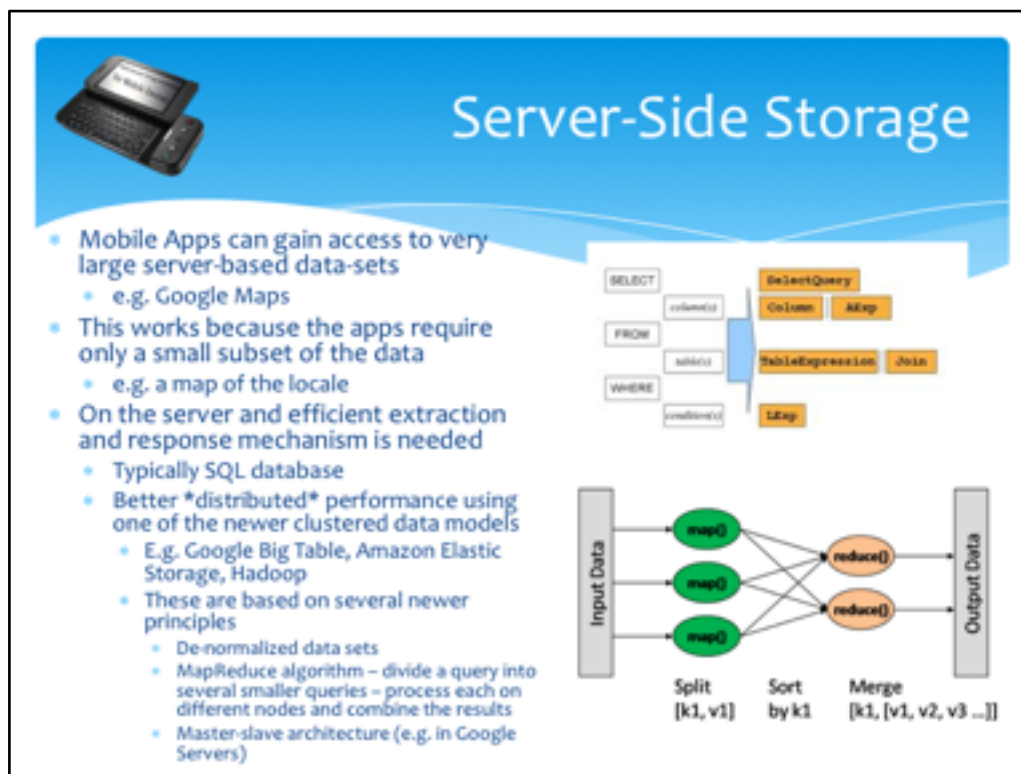
The term Ajax was coined by Jesse James Garret in “Ajax: A New Approach To Web Applications” in 1996 (discussing Google Applications)

Ajax (Asynchronous Javascript and XML) now rarely uses XML, which is considered inefficient for web communications. Instead, JSON is becoming the preferred data packaging method. Since Ajax calls are always made to collect data for display in a browser, JSON is as convenient format.

Ajax relies on the XMLHttpRequest object, which is now provided as standard in most browsers. The very first Ajax-like performance was implemented using Microsoft’s ActiveX HTTP-Request object – Mozilla (Firefox) soon followed suit by introducing the XMLHttpRequest object in browsers.

The Asynchronous part of the name comes from the inevitable response delay after making a web request. This is always unpredictable since the request needs to get to the server, the server needs to respond (possibly having to perform some complex processing) and the result has to get back to the browser. Performance at the browser would be very jerky if every time a request was made, the browser had to

wait for a response. Instead, when a request for data is made to the server, a callback function is provided so that when the result is returned, this function will be used to process it typically inserting it into the DOM tree). This is asynchronous behaviour – send out a request and don't wait for the response.



Implementation depends on:

The size of the data set being accessed

e.g. a table of employees in a small company, or a distributed database of map tiles for route calculation

The distribution of data servers

e.g. a single DBMS in a server machine, a server cluster, an international distributed data-set

The nature of the query

does it access a single constrained set of data (employees) of a complex, general database (e.g. Google Search)?

The query algorithm used

e.g. SQL SELECT, or something more complex

Many VERY LARGE data-sets are based on distributed systems with in-built redundancy (e.g. Amazon, Google)

- The results of a query might come from any or many of a large set of semi-independent data sets
- MapReduce makes it possible to search a large number of data-sets in parallel very quickly
- The MapReduce algorithm uses two functions (Map and Reduce, natch)
- Map() takes a key value pair (e.g. q=Distributed+Data+Set) and return a list of matching sub-queries (e.g. q=Distributed, q=Data, q=Set) These are applied in

parallel to a set of database nodes, the responses are gathered by the master node.

- The Reduce() function concentrates the set of responses into a prioritized list Note
 - Google are very secretive about the actual reduce algorithm they use

The overall result of MapReduce in BigTable, EC2 or Hadoop is that a massive distributed data set can be searched efficiently

Each node does only a small amount of work.

Throughput for a lot of queries can be very high.



Web Server Costs

- What does it cost to manage a web server
 - Permanent internet connection
 - 24x7 up-time
 - Costs if there is a break
 - SLAs = Service Level Agreements
 - Maintenance of server data, web pages, services
 - Provision of redundancy (to limit down-time)
- If you are an Application Provider, these costs come on top of development costs
 - Infrastructure for building is low-cost
 - Infrastructure for operation is high-cost



The point of this slide is to indicate the potential costs of having to host a web app. It is not suggesting that you should never host a mobile app by providing a web server etc., but simply that it is a fairly expensive burden for a single web app. If you happen to have a ready-configured server lying around, by all means use it. However, if you don't, there are many cheaper (and probably more effective) options.



Hosted Solutions

- * Absolve yourself of the costs of managing a server but...
 - * Accept limitations of host
 - * Charges – possibly steep for a good SLA
 - * Limitations – may not provide the format (server-side languages, database formats) you develop in
 - * 3rd Party – do they care about your data? (hence the high cost of SLAs)
 - * How expert are they in data management, security, redundancy etc.
 - * What if the infrastructure doesn't work?



There are loads of web hosting companies out there. It is worth asking some tough questions of any company before you decide to use its services. The little cartoon at the bottom is a good indication of what you could end up with “No special skills Required” is hardly the qualification you would want for a host for your app.

Bring in issues of payment processing, service level agreement (guaranteed uptime or a refund), multi-format hosting, secure back-ups, high availability etc., you will probably want to use a reputable host or one of the big players (not suggesting Google, Microsoft or Amazon aren't reputable).



Cloud Infrastructure

- Several models here, but typically...
 - You rent virtual machine images in the provider's farm
 - You configure the machine to meet your needs
 - Install database (e.g. MySQL, SQL Server...)
 - Install software support (e.g. PHP, .NET framework...)
 - Manage the web server (IIS or Apache)
 - In the best cases, standard machine images are available – you simply add your own data and code
 - You pay for machine images – need more machines, buy more images
- This is the Amazon, Windows Azure approach
- No server maintenance, usually good SLAs (the host simply sees machine images – easy to back up, deploy, duplicate etc.)
- Then there's Google App Engine...
 - A different structure
 - You develop an app for the Google Framework
 - Big Table database built-in
 - Simple access to data through language bindings (Java, Python or Go)
 - Google Hosts your app across its servers
 - Benefits of big data access etc., Google distribution etc.

The most important feature of *proper* cloud hosting is that there should be no problem dealing with big and rapid changes in the number of connections that your site needs to serve. Cloud providers operate from server farms where scalability is built-in and available through a simple console setting. Google, Amazon, Microsoft, Citrix etc. all make it very easy to increase the amount of processing power at your disposal (at a cost, of course)

Interestingly, Google seems to be the only one that provides application tuning for response to their non-paying customers. The App Engine provides several ways for you to tweak your application to make it respond faster – warmup requests, memcache and other measures are available even to customers using only their free quota. While these features can add to the amount of processing you need to pay for once you've gone beyond the free quota, you still need to use a lot before fees start coming in.



Google App Engine

- Basically this is your application running on the Google infrastructure
 - The same one that handles search, maps etc.
- Develop in Python, Java or Google Go
 - Or use Zoho Creator (for a price)
- Develop and test on your own machine (Windows, Mac, Linux)
- When ready, deploy to Google servers
- Pricing
 - First 5,000,000 page hits per month free – up to 10 apps
 - That's a continuous 2 hits per second
 - Apps may not scale up too well
 - Beyond that, \$9.00 per app/month includes a SLA and infinite scaling
- Not surprisingly, this is ideal in a teaching environment, and what I suggest you should use for project work



Google App Engine is built on Google's WSGI (Web Service Gateway Interface) API. It is rather like working with a web server, in that you define classes which contain functions that respond to web requests. Using classes is important, since we may want a page to be able to respond to many simultaneous requests, and defining a class allows the server to create a response instance for each incoming request.

A typical GAE response class (in Python, 'cos it's the easiest to follow) is:

class **MainPage(webapp.RequestHandler):**

def **get(self):**

```
username = self.request.get("username")
self.response.headers['Content-Type'] = 'text/plain'
self.response.out.write('Hello %s, welcome to web-app world! % (username))
```

The code above is most of what you would need for a basic HelloWorld app that responds to Fred's request with the message:

Hello Fred, welcome to web-app world!

The single function within the MainPage class responds to a HTTP GET request. In this case the request comes with a request variable – username (the web mechanism

for this is that the URL making the request ends in ?username=<some user name>). The response is sent via the MainPage class's **response** member which deals with sending data back to the requesting app or browser. In this case, the user's name is embedded in the rest of the message using a standard python formatted string (%s is a format element, that will be substituted by a value inserted after the % at the end of the string). Before the actual response is returned, the get() function writes a header to the response which simply indicates the type of data the rest of the response will be composed of (plain text here).

The App Engine supports three languages at present – python (my preference), Java (most other people's preference) and Google Go (not very widely used, but an interesting language optimized for this purpose).

In addition, you can use the Zoho Creator to build app-engine apps, a drag & drop programming environment (with an embedded language, Deluge Script [yuk!] for adding complex behaviour). See <https://www.zoho.com/creator/> for details. This can be used for free to create up to 3 apps by up to 2 developers. This is perfectly suitable for your project work.



Initially a Google App Engine app, Firebase is now a fully supported Google online database system


- It is NoSQL – i.e. a Firebase DB is a set of key:value elements in JSON format
- Go to <https://www.firebase.com/> and start the tutorial (press the “Get Started” button) to get an idea of what it is for
- A Firebase DB can act as an online data source for a Mobile App
- Firebase hosting is available for serving your application from a related web domain (firebaseapp.com)
- If you intend to use Firebase for your project, be aware I will expect an app to be fully secured, served from a Firebase hosting platform and your app will need to be able to store and retrieve data from it.
- More on this later in the module.

Firebase is fairly new and has been getting a lot of attention from app developers who don't want to go to the effort of building, managing and maintaining a web service using more normal tools (e.g. App Engine) and only need data storage. Its major selling points are:

- It can be used form a variety of application types, supporting Android and iOS apps, Windows apps, Web apps and provides a standard REST interface for any other type of client
- The web-app framework (a single Javascript file called firebase.js) takes care of all of the complexities of upload, download, updates and security – no need to use AJAX since it does it for you
- Security is strong, and is a very simple declarative process – i.e. you create a settings 'file' in a Firebase control panel that specifies your security needs. From there, the client libraries handle registration (if needed) and log-in transparently. You can use Google, Facebook, Twitter or GitHub authentication, which makes the security stronger and easier to manage for developers (very little coding) and clients (using your Google log-in saves you remembering yet another password)
- The Web interface is event-based, which suits Javascript perfectly. Uploads are done by two simple functions (Firebase.set() and Firebase.push()) depending on whether you want to supply a key (set()) or create a new one (push()). Changes to a Firebase data-store are automatically sent as events to every web subscriber currently running the Firebase library. Events signal records added, changed or

deleted.

- It is a NoSQL data-store, which works well since the JSON format reflects the data members of Javascript objects automatically.




Node.js

If you have a server to manage, or are prepared to have a site that has limited availability

- Node.js is...
 - The Google V8 Javascript Engine
 - An extensive library of modules (networking, http, data management, database binding etc.)
 - Javascript extensions for code modularity
- Run node at the command line and immediately start serving pages
 - To localhost – ideal for testing
 - To your machines IP address
 - Other machines on the network can access
- A list of hosts who provide Node.js server instances is at...
<https://github.com/joyent/node/wiki/Node-Hosting>

```
var http = require('http');
http.createServer( function(req, resp) {
  resp.writeHead(200,
    {'Content-Type': 'text/plain'});
  resp.end('Hello World!\n');
}).listen(8888, "127.0.0.1");
```

node? JS? WTF??



Often a mobile app will have very simple servicing needs – a shopping-list app, for example, may only need some simple unstructured web-storage so that the data can be synched between multiple devices. In that case, Node.js is a very good solution. Node.js is a lightweight HTTP server which can be used to create very simple request/response web apps. Typically, you would run the Node executable on a PC with a permanent web connection, although there are a good few web hosts that provide it as a service. Its strongest features are:

- You write Node.js code in Javascript – the Google V8 Engine is embedded in the Node.js executable and so you are able to create server code using exactly the same language as you use to create client code.
- It includes a library of code modules that cover every need for web apps – from HTTP request/response frameworks and simple text file handling to MySQL database access.
- The language includes a very simple but powerful module mechanism – `require()` is a function that you use to include a library module into your code.

You can write a full featured web server with enough power to handle the needs of just about any mobile client app in about 25 lines of Javascript using Node.js. You should view Ryan Dahl's video introduction to Node.js at http://www.youtube.com/watch?v=jo_B4LTHi3I



Your Coursework

- The project you do for this module will involve creating a mobile app AND a site for distribution and back-end services
- You can use any of the methods discussed here (or others – just check with me first)
- Best option – create a service using the tech that most closely matches your past experience
- If you're having difficulty choosing, go for Google App Engine and Python
 - Easiest from a standing start
 - Good Google tutorials
 - Some of my examples may help get you started

The only rules you need to follow in this coursework are:

- Projects are to be your own work. You can include other people's code provided you attribute it but you should not try to pass off someone else's work as your own, and your project submission can not be a complete application built by someone else. The UWS rules on plagiarism are available from the library website
- Your project must include client and server parts. The client part must be a built for some form of mobile device specifically (not just a web app that will run in a mobile browser). The server part can be built in any of the technologies discussed in the module. If you want to incorporate server technology not mentioned in the module (e.g. ColdFusion), you need to discuss this with me (Alistair). You can use a Firebase database as the online service, but since this is a lot less work to set up, I will insist on any Firebase database as being properly set up with security and user-account control – without these, I won't mark the service side of your project at all
- You will need to fully deploy your application using UWS web servers available in the labs, a cloud host (such as Google App Engine or Zoho) or your own server – e.g. you can demonstrate your app to me using Apache + PHP in your laptop or provide access to a web server running on your home PC. Localhost servers are available in the E116 labs (IIS and/or Apache and/or WAMP stack [Windows-Apache-MySQL-PHP]).