

Building Larger Applications with IBM Worklight

Andrew Ferrier

andrew.ferrier@uk.ibm.com

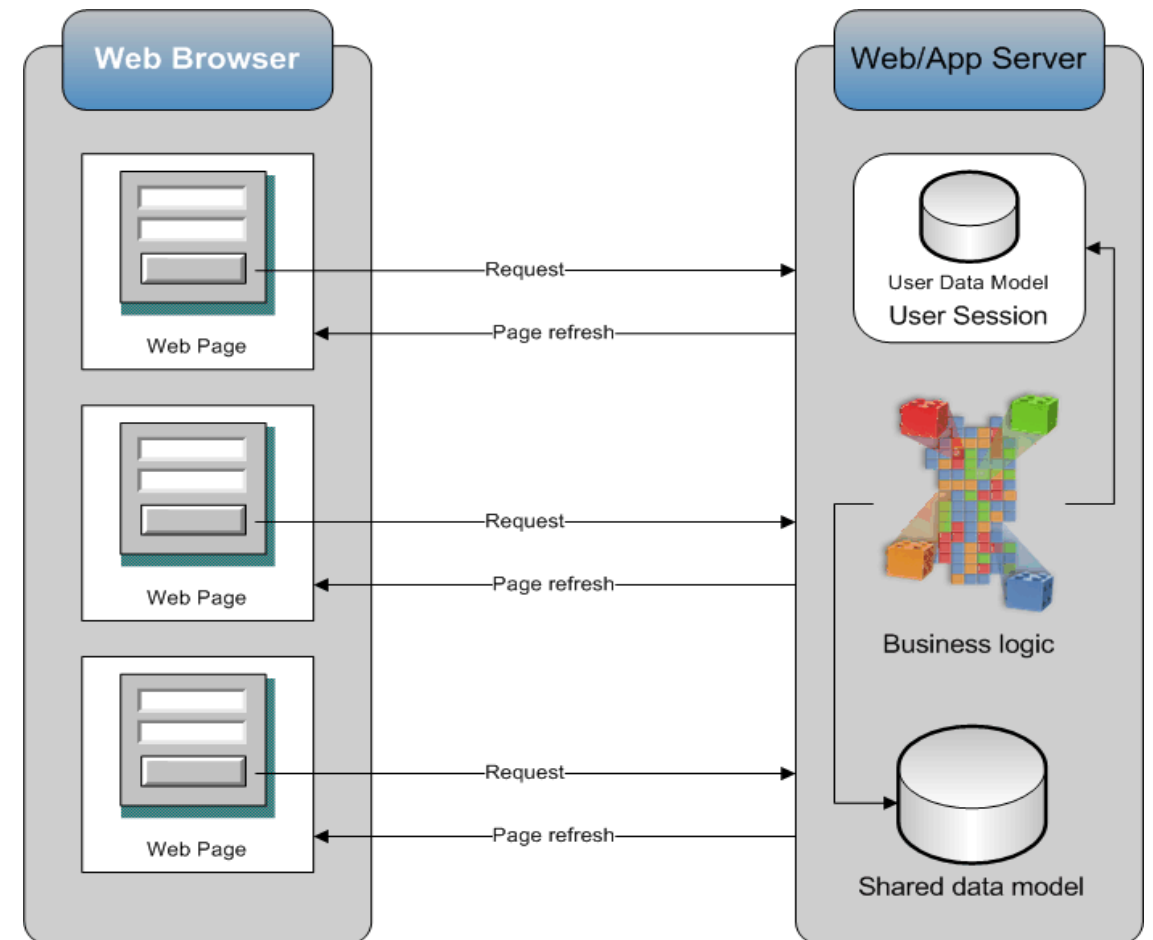
Agenda

- Recap - Web, Mobile, and Worklight
- **Development Time**
 - Toolkits and Frameworks
 - Structuring Code
- RESTful **Services** and Worklight **Adapters**
- **Lifecycle** - Library Systems, Builds / Testing / Deployment
- **Other** Tips - Client-side, Server-side, and Updating

Recap - How Has The Web Changed?

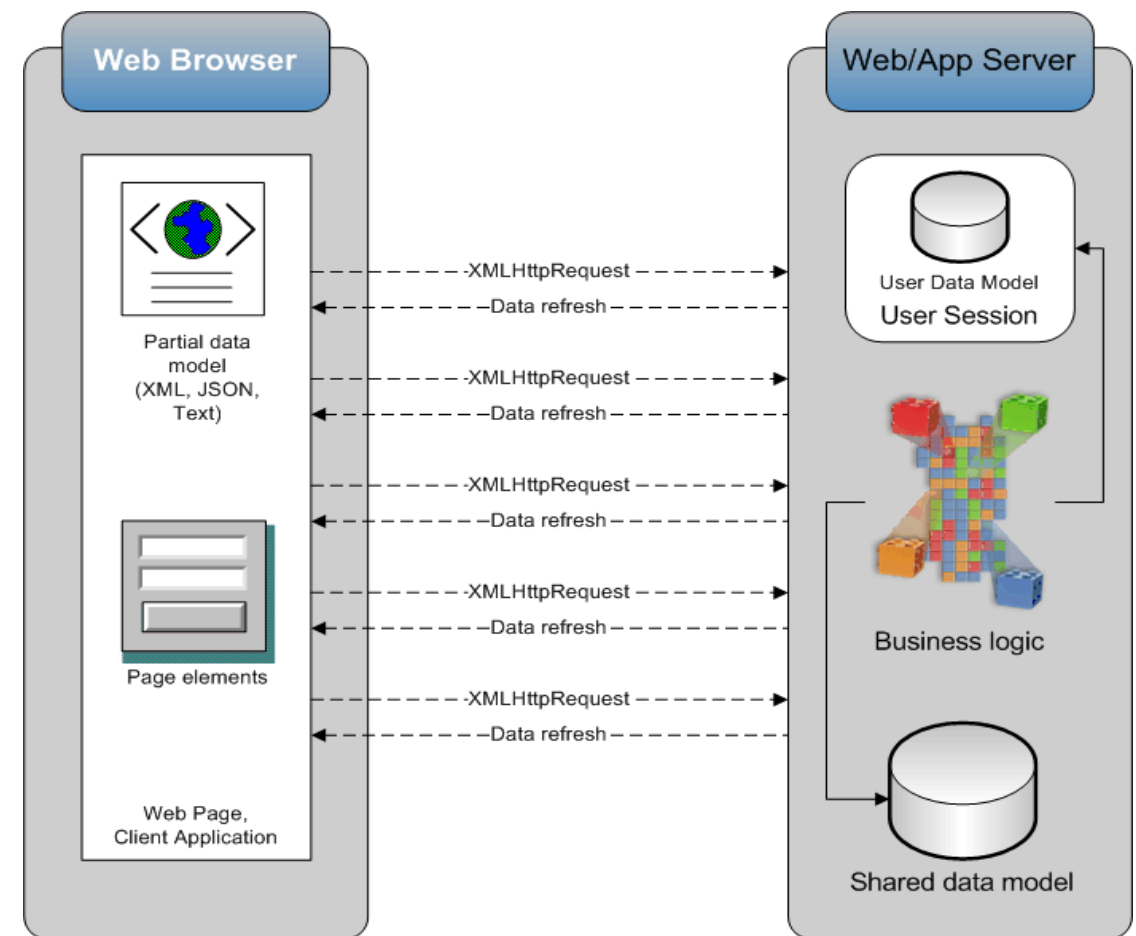
Web 1.0 Model

- Static HTML content, little-to-no-dynamicity
- Most folks know this already
- Server-side-driven content
 - Perhaps with a small amount of JavaScript for effects or form validation
 - Traditionally written with a variety of technologies – Servlets, JSPs, PHP, etc.

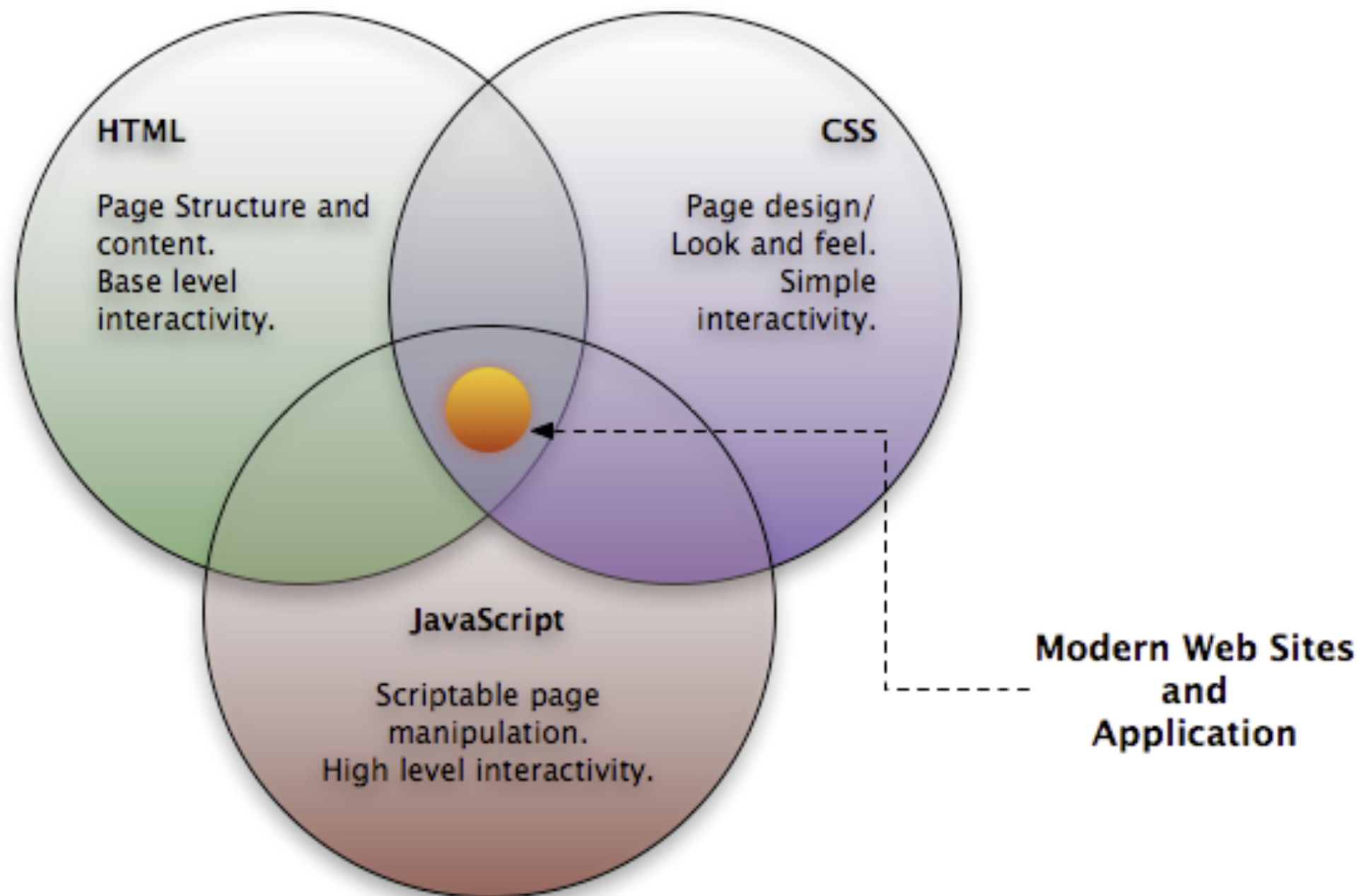


Web 2.0 Model

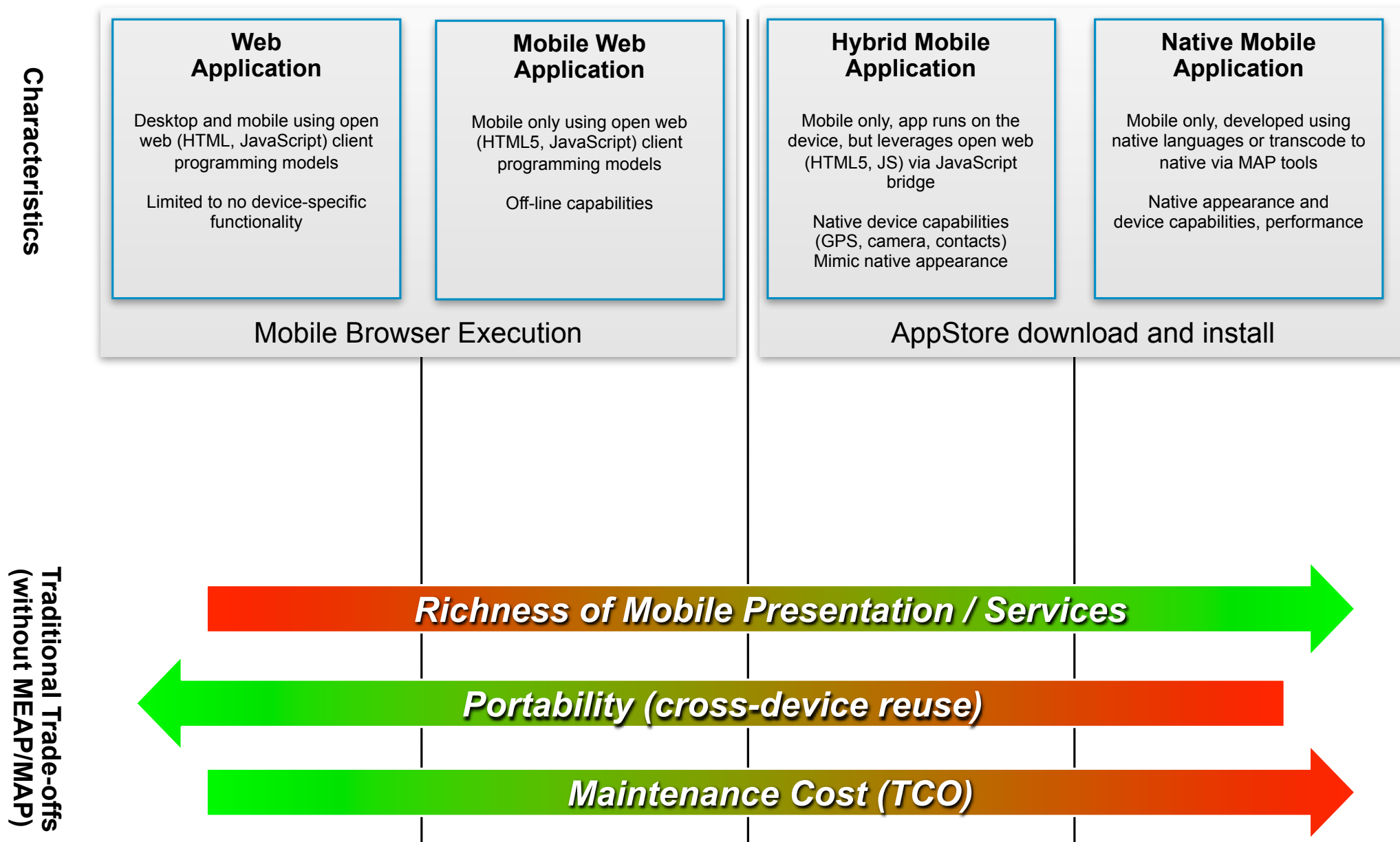
- Browser using AJAX/XHR to communicate with server
- Lightweight RESTful Services (often using JSON data)
- Service Gateway or other technology to proxy all service invocations



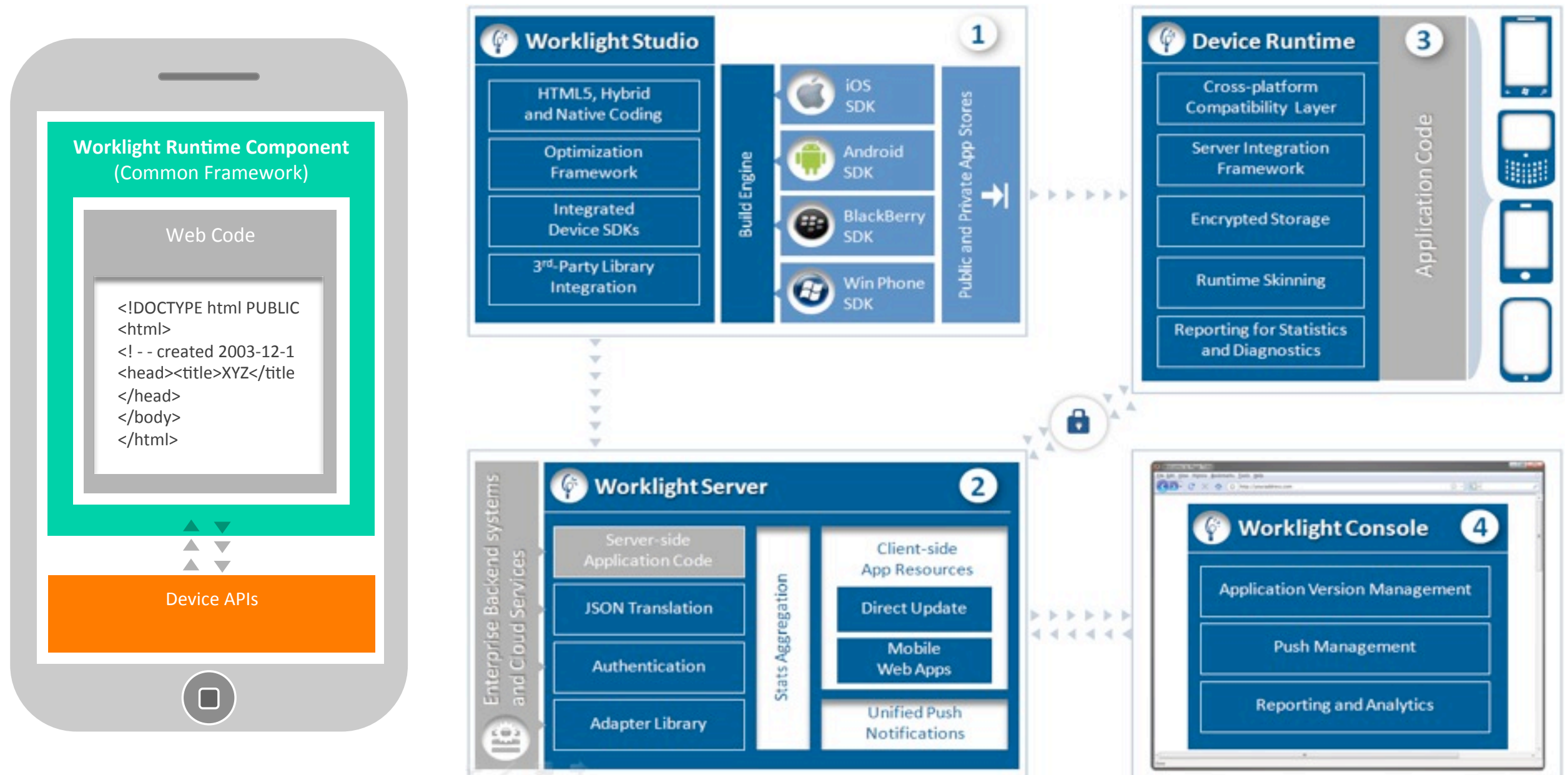
The Programming Model



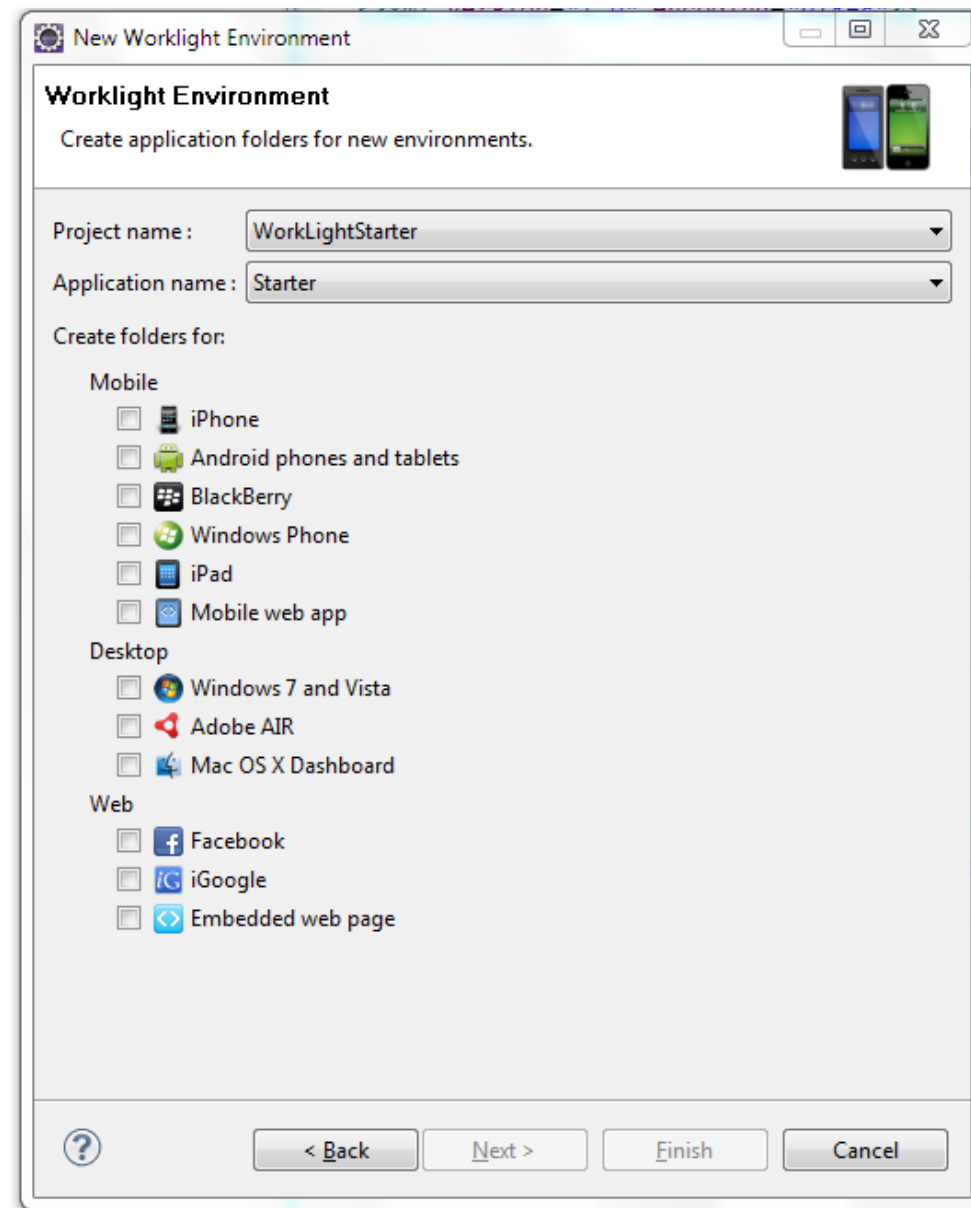
What does the mobile landscape look like?



What is Worklight?



Build Once, Run Anywhere...



Development Time

Toolkits vs. Frameworks

- **Toolkits** - JavaScript-based libraries used on top of JavaScript itself
- Smooth out the rough edges of JavaScript
- Add additional features, UI controls etc.



Toolkits vs. Frameworks

- **Frameworks** - used on top of toolkits
- Structure applications
- Provide large-application functionality

Toolkit Options

- The largest players in the market are



- Generally, IBM prefers Dojo

Why Dojo?

- Enterprise-grade toolkit and feature set
- Stronger support for structuring large applications
 - e.g. Class system (`dojo/declare`)
- Better focus on internationalization, accessibility, etc.
- But **jQuery** is a supported choice too for Worklight

Do we need a framework?

- Coding without JS **toolkit** in 2013 is like entering the program in binary
- You can code without a **framework**, but you lose:
 - Endpoint management (stubbing)
 - State / session management
 - *(other application-level stuff)*

Generally use **views** for Mobile...

- Rather than multiple `.html` pages, have one
- Less page startup pain for mobile web
- Dynamically insert views (HTML) into DOM
- Dojo Mobile has this concept built in - `dojox.mobile.view`
- Reuse this concept for Hybrid too

Framework Options

- For Dojo:
 - `dojox/app`
 - `issw.mobile/issw.pocMobile`
 - Your own custom framework
 - Not as bad an idea as it sounds!
- For jQuery:
 - `mustache`, `RequireJS`, `Knockout JS`, `Backbone`, etc...

dojox/app

- Can define “page controllers” for different views in the application
- Manages loading of views and associated page controllers via configuration
- Also allows for declarative MVC framework where needed (working with dojox/mvc)
- No endpoint management, etc...

ISSW Offerings

- ISSW has offerings e.g. `issw.pocMobile`. Includes:
 - Extra `dojox.mobile.*` widgets
 - Easy lazily-loaded views
 - Worklight 'mocking' to use project outside of WL
 - Abstraction of endpoints / adapters / services
 - etc...



Structuring Code

- Whatever framework you use, follow code structuring practices:
 - 1:1 mapping between View ('page') and programmatic controller class for that page
 - Dynamically load views into the DOM on-demand to avoid overloading it

RESTful Services and WL Adapters

RESTful Services

- The world (at least UIs) are moving to simpler services

- A RESTful style - plain HTTP GET, PUT, POST, DELETE →

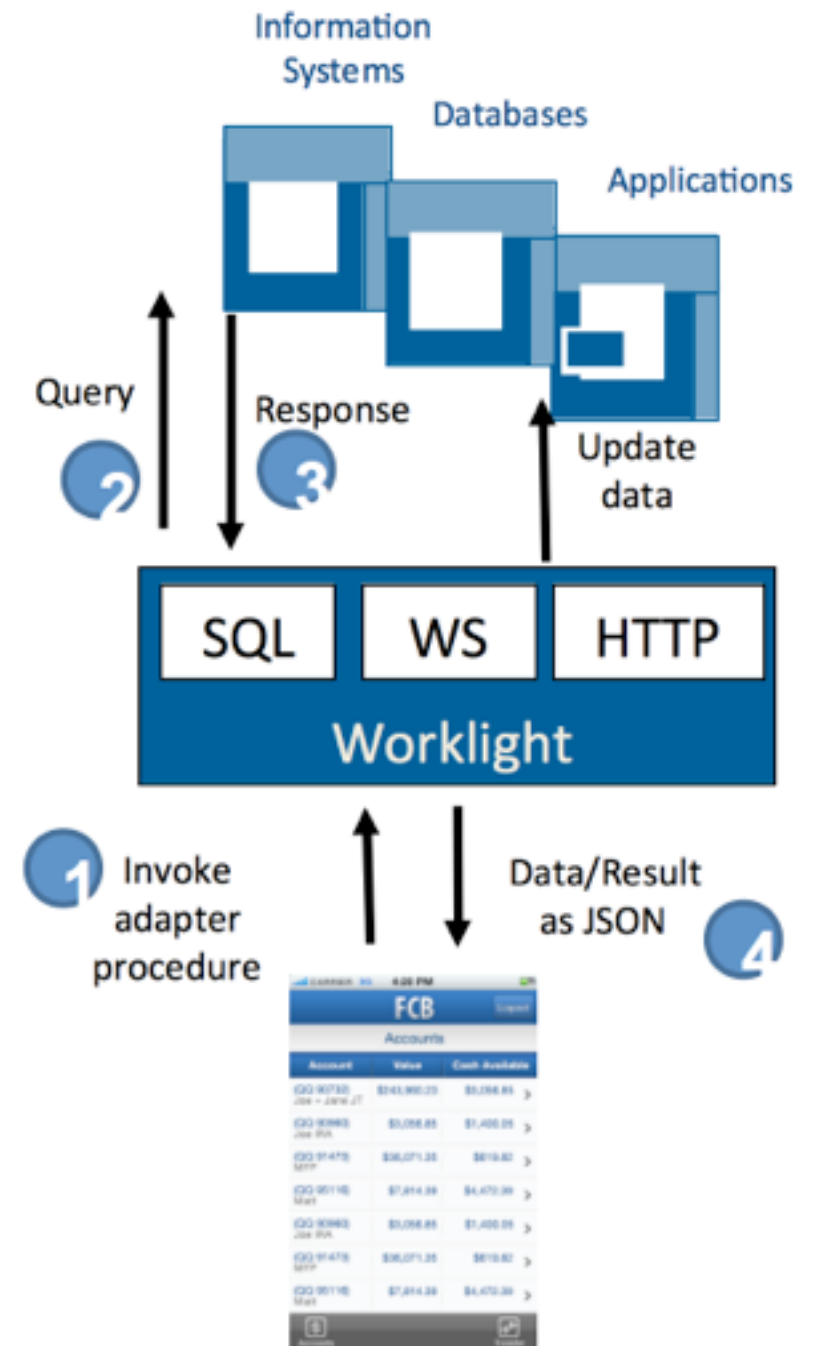
```
GET http://mycorp.com/customer/1234
```

```
{  
  "name": "Fred Bloggs",  
  "address": "123 Anytown"  
}
```

- JSON as the data format
- Practically mandatory for consumption by Web 2.0 clients

WL Adapters

- WL adds adapter framework
- Customized on server with server-side JS
- Supports HTTP, JMS, SQL, and Cast Iron adapter types
- Most common use is HTTP adapter to integrate with JSON/REST or SOAP/HTTP



WL Adapters - REST & HTTP

- You could use RESTful services directly from WL container with conventional XHRs, but you lose:
 - The ability to use the WL server as a “choke point”
 - WL’s authentication mechanism for services
 - WL Logging/Auditing

WL HTTP Adapter and REST

- Even for services already exposed over REST, we would re-expose them using the WL HTTP Adapter.
- This is comparatively straightforward to do.
- You can also use SOAP services from WL
 - Abilities are limited at the moment so for more sophisticated scenarios, consider an ESB.

Lifecycle

Library Systems

- WL can work with most version control systems that integrate with Eclipse
- Common choices:
 - **Rational Team Concert**
 - **Git**
 - **Subversion**

Library Systems 2

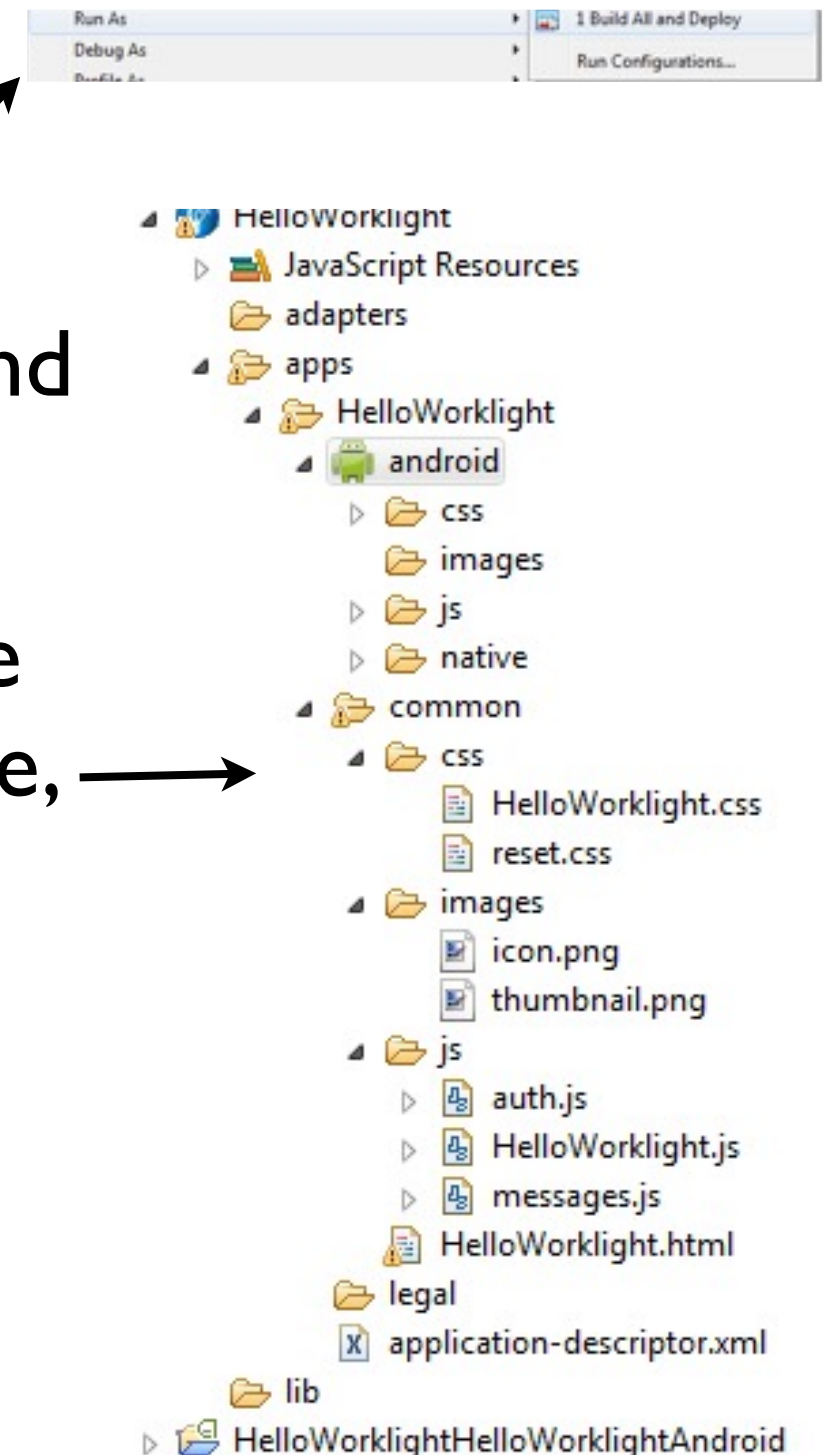
- There are files that must be excluded as they are part of WL generated resources, see here:

- http://pic.dhe.ibm.com/infocenter/wrklight/v5r0m5/index.jsp?topic=%2Fcom.ibm.worklight.help.doc%2Fdevref%2Fr_integrating_with_source_control.html

```
MyProject
  adapters
  apps
    myApp
      android
        css
        images
        js
        native
          assets
          www
          wliclient.properties
          other user files
        bin
        gen
        nativeResources
        res
        libs
        res
        src
        AndroidManifest.xml
        default.properties
      blackberry
        css
        images
        js
        native
        ext
        www
        config.xml
        icon.png
        splash.png
        common
      ipad / iphone
        css
        images
        js
        native
          build
          Classes
          Cordova.framework
          <application-name>.xcodeproj
          Plugins
          Resources
          WorklightSDK
          www
          Entitlements.plist
          main.m
          Cordova.plist
          <application-name>_Prefix.pch
          <application-name>-Info.plist
          README.txt
          worklight.plist
        nativeResources
        Resources
      package
    dojo
    server
    conf
```

Building

- You will want to automate the build.
- WL provides the `<app-builder>` and `<adapter-builder>` ANT tasks
- Only builds the Server portion of the projects - the **.war** customisation file, the **.wlapp** file, and the **.adapter** files.
- You will need to build the **.apk** and **.ipa** files using platform-native process.



Building

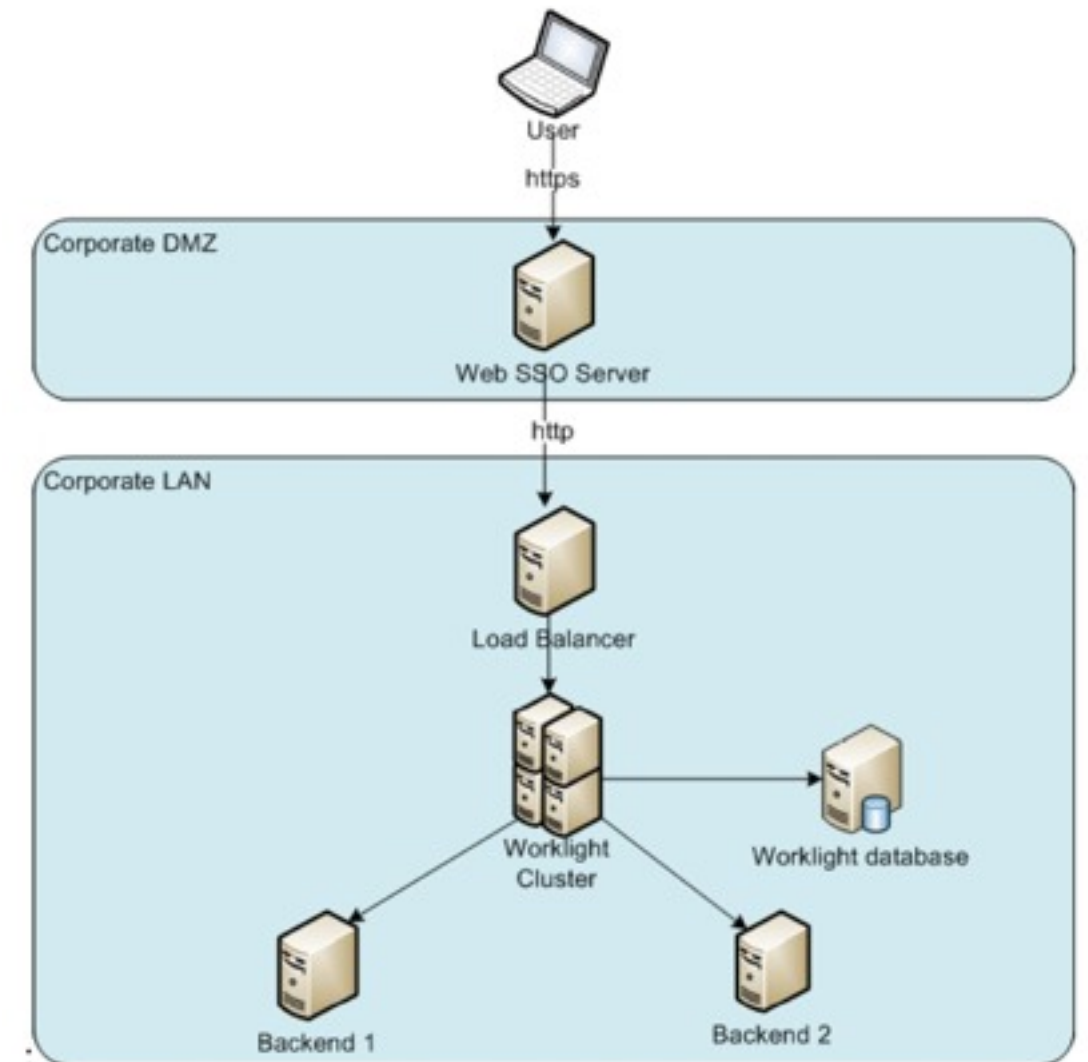
- During build, externalise certain things:
 - `worklightServerRootUrl` in `application-descriptor.xml`
 - `server/conf/worklight.properties`
 - `maxConcurrentConnectionsPerNode` for adapters

Deploying

- Deploy the **.war** using relevant application server method
- Deploy the **.wlapp** and **.adapter** server-side portions of the application using `<app-deployer>` and `<adapter-deployer>` ANT tasks.

Deployment Topology

- Options include:
 - WebSphere Application Server - familiar
 - WAS Liberty Profile - simpler
 - Tomcat
- Consider **HTTPS**, **load spraying**



Deploying to Phones

- You still need to get the native application (.ipa, .apk, etc.) onto your user's phones.
- **Testing lifecycle:** AppCenter - comes with WL server editions
- **B2C:** public App Stores (Apple App Store, Google Play Store)
- **B2E:** Tivoli Endpoint Manager or similar

Testing

- Typically you'll want to test:
 - **Manual UI** on physical phones
 - Coverage across devices
 - **Automated UI** - mocking framework and automated test tool (e.g. Selenium)
 - **Adapters** - load / performance / functional tests - just HTTP

Other Tips

Client-side

- Don't optimize for size of the client like you would do for Mobile Web
- Nevertheless, there is still a browser control underneath
- Use `WL.Logger` . { `debug` , `error` } API, logging in development environment is customizable, & log the username on errors

Client-side

- Understand handling errors on client-side, in particular adapter invocations:
 - http://www.ibm.com/developerworks/websphere/techjournal/1212_paris/1212_paris.html?ca=drs-
- Use `connectOnStartup: false`, with `WL.Client.Connect()` after startup - gives more startup control
- Write as little native code as possible

Server-side

- Again, understand how to handle errors from adapter invocations (same article).
- Again, use `WL.Logger` API - has various levels of logging, can be configured on server. Log the username on errors.

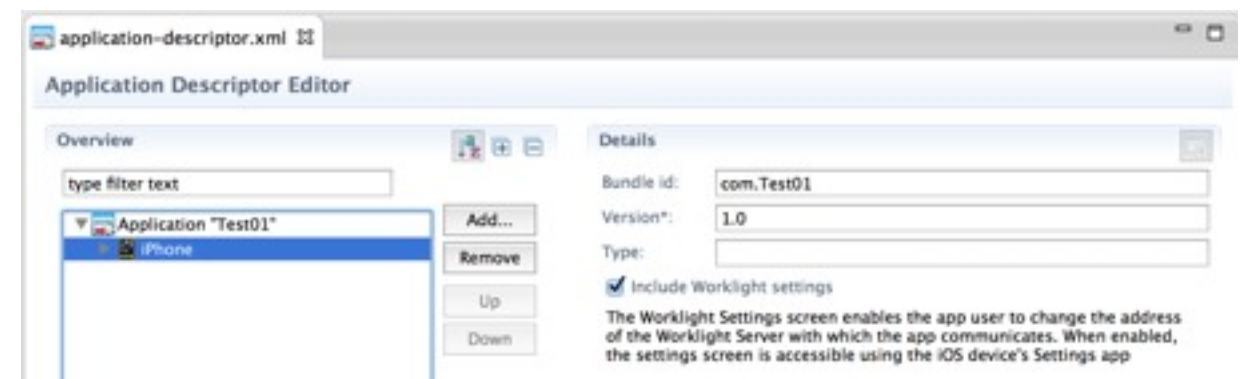
Two Ways to Update - Method I

- Update your web code only
- **Don't** change the version number of the application
- Redeploy **.wlapp** only
- Implicitly encourages a “Direct Update” next time client connects.



Two Ways to Update - Method 2

- Method 2:
 - Update web code and/or custom native code
 - **Do** update the application version number
 - Re-release via binary method (App Store, etc.)



Updating Worklight

- Re-release an app using method 2
- Gets new Device Runtime onto end-users' phones
- But end-users can continue using old app; wire protocol is backward-compatible

Summary

- **Development Time**
 - Toolkits and Frameworks
 - Structuring Code
- RESTful **Services** and Worklight **Adapters**
- **Build Time** - Library Systems, Builds / Testing / Deployment
- Other Tips