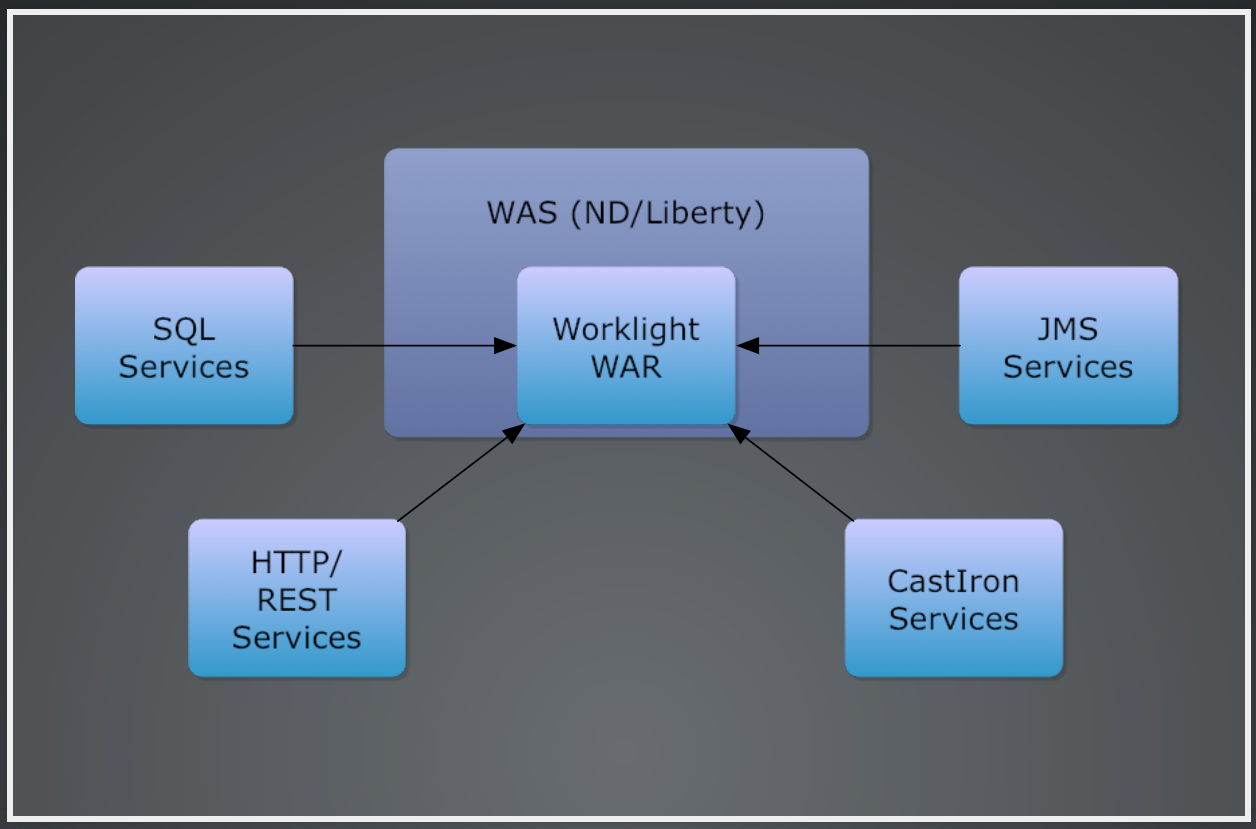
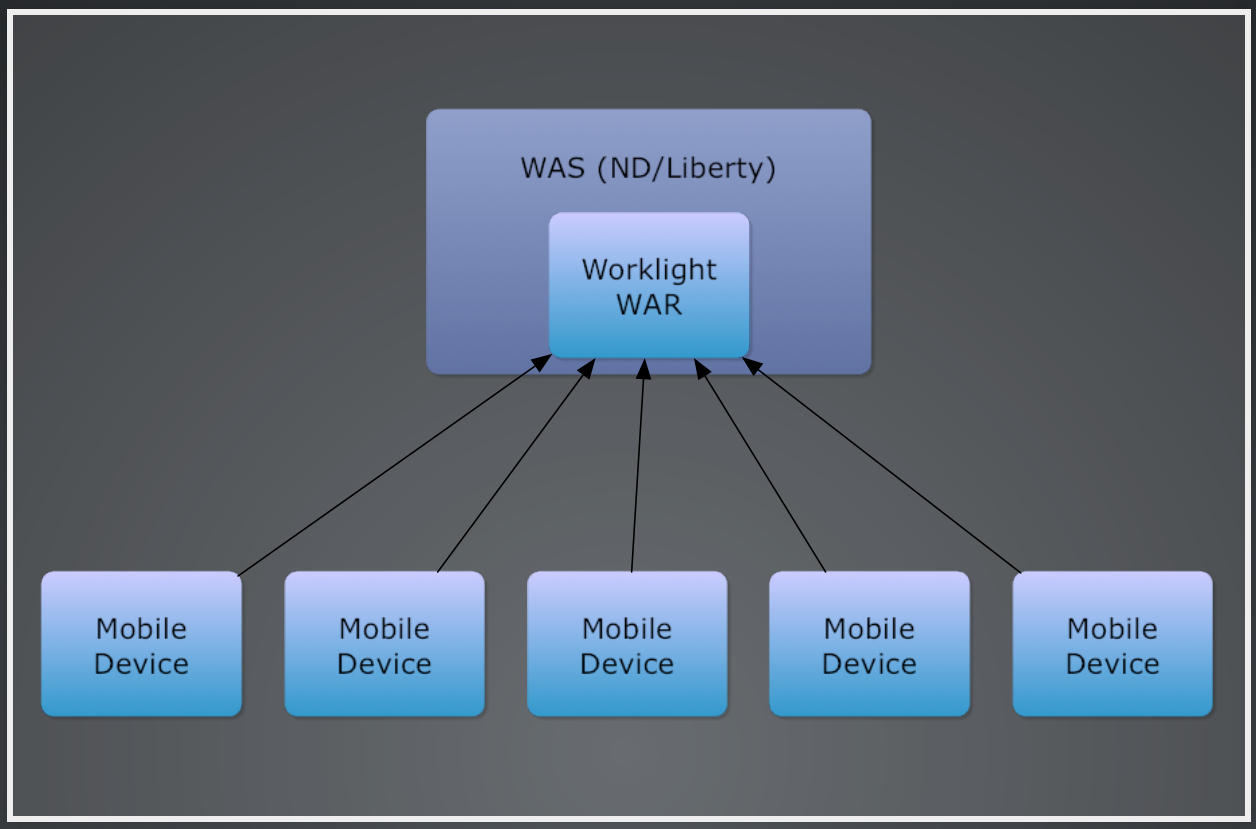
# LESSONS LEARNT IMPLEMENTING A WORK-LIGHT-BASED ECOMMERCE MOBILE SOLUTION

Sean Bedford / @bedfordsean / SEANBEDFORD@uk.ibm.com

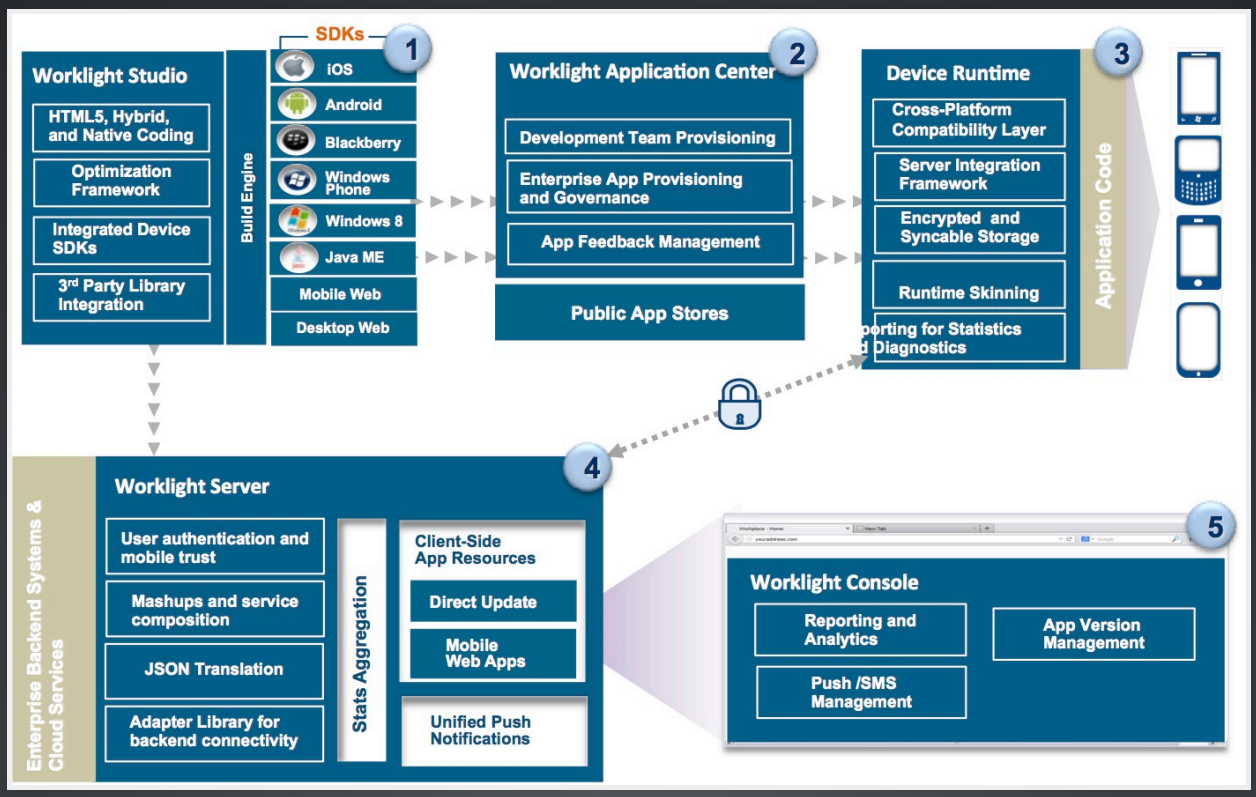
# WHAT IS WORKLIGHT? - SERVER SIDE



# WHAT IS WORKLIGHT? - CLIENT SIDE



# THE BIGGER PICTURE



# WHY IS IT GOOD?

- Out-of-box support for app analytics (Device, OS, code version)
- Direct update provides a powerful way to quickly patch issues
- App versioning scheme allows operational management of multiple deployed versions
- Security model provides easy control of back end resources
- Not "just Cordova (PhoneGap)" - native APIs for iOS, Android, and devices running JavaME

# CORDOVA PRIMER

- API layer between web container and native functionality
- You can write a new Cordova plugin with native implementations on each platform you want to support
- This plugin can then be wrapped in to JavaScript to allow for calling from a web container

# CLIENT SIDE CODE

## MVC APPROACH

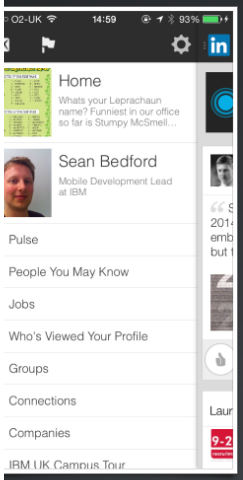
- How do you store data?
- How do you display data?
- How do you handle the business rules?

Pick a framework that supports MVC such as **jQuery Mobile, AngularJS, or Dojo**. It will make your life MUCH easier :-)

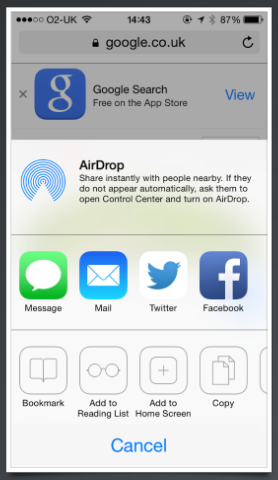
# THINK OF THE USER

- Think about how the user is going to interact with your app
- Consider controls, text sizes, readability. Apple recommends >= 44px * 44px tappable area and 17pt+ font size
- Design, prototype, and test with actual end-users
- Don't be afraid to scrap a design and start again

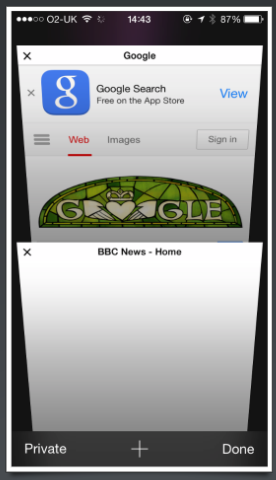
# COMMON UI PATTERNS



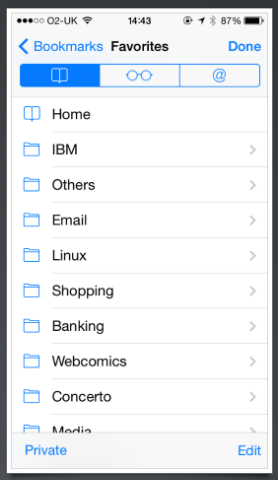

Navigation



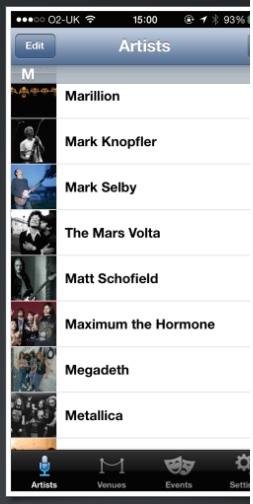

Actions




Custom




Modals, Lists, Scope

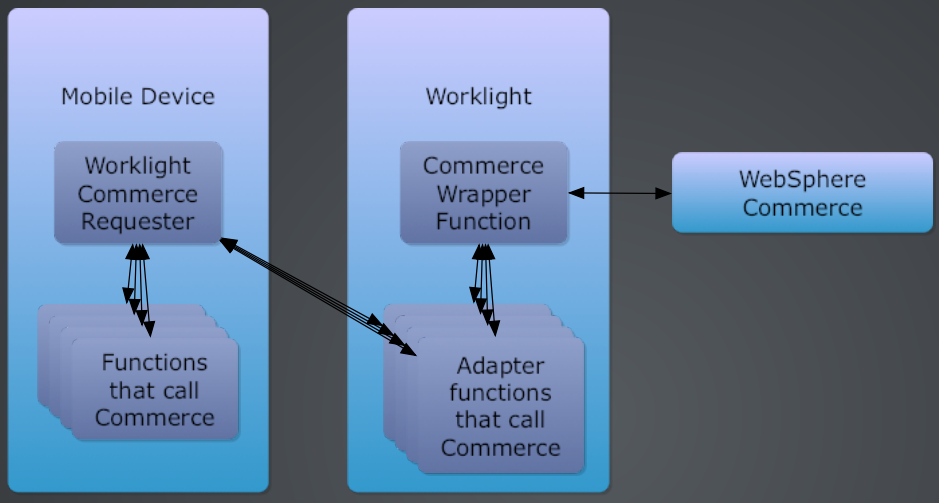



Tabs

# SERVER SIDE CODE

## CONSISTENT ENDPOINT MANAGEMENT

- Common problem: How to handle errors, timeouts, bad responses consistently?
- Solution: Provide a consistent client- and server-side endpoint



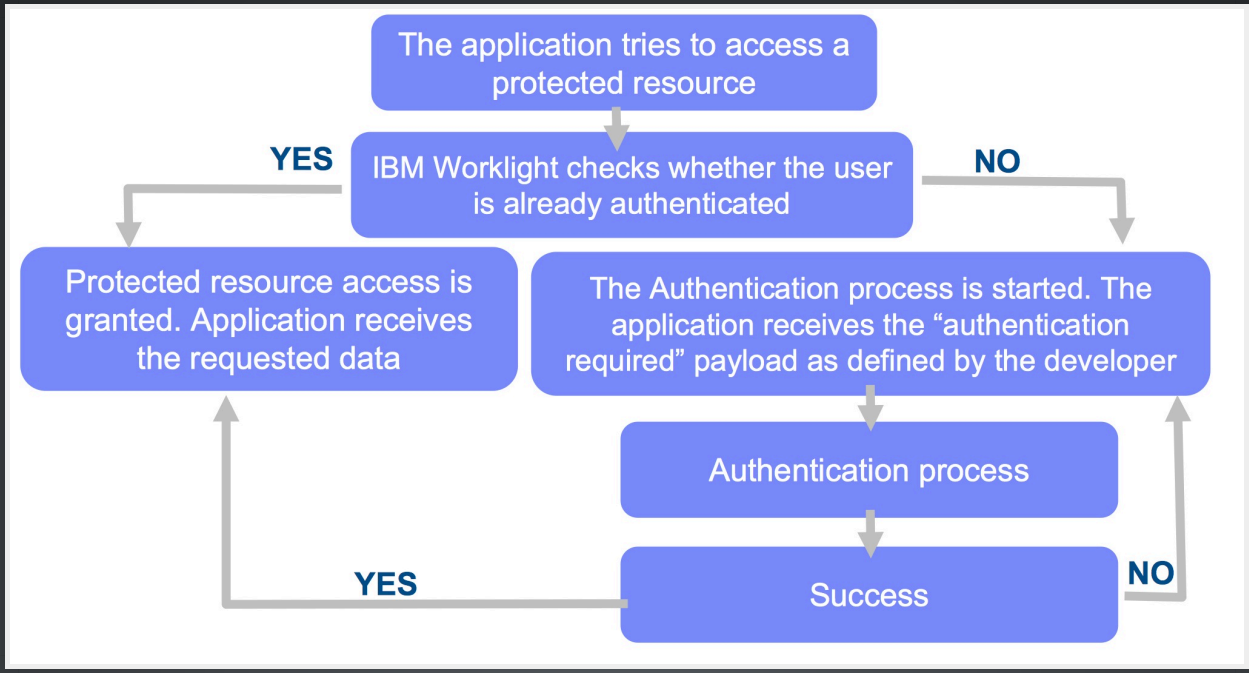
# GO-LIVE CONSIDERATIONS

- Think about end-end versioning - does your back end support multiple live versions?
- Think about security - validation in the front end only is never enough
- Think about how the application will load and service frequent requests (startup, browse, login...)
- Think about operational management (logging, reporting, information for support)

# BASIC WORKLIGHT USER AUTHENTICATION

- A key feature of Worklight is it's security model
- This allows the securing of a back end call with a single line of code (once it is set up)

# AUTHENTICATION FLOW



# EXAMPLE - SET UP

- Create a challenge handler in your code. This will need to issue a username/password, or some other authentication toekn
- Set up a realm and security test in authenticationConfig.xml. In this security test, state you want to use adapter functions to authenticate the user

```
<customsecuritytest name="MySecurityTest">
<test isinternaluserid="true" realm="MySecureRealm" step="1">
</test></customsecuritytest>
...
<realm loginmodule="StrongDummy" name="MySecureRealm">
<classname>com.worklight.integration.auth.AdapterAuthenticator</classname>
<parameter name="login-function" value="AuthenticationAdapter.onAuthRequired">
<parameter name="logout-function" value="AuthenticationAdapter.onLogout">
</parameter></parameter></realm>
```

# EXAMPLE - SECURING A FUNCTION

- In your adapter's XML file, add a security test to a function

```
<procedure name="myUnsecuredProcedure"></procedure>
<procedure name="mySecuredProcedure" securitytest="MySecurityTest"></procedure>
```

- This will trigger the security test against the realm you've defined in authenticationConfig.xml
- This in turn will issue a challenge for "MySecureRealm" if there is no user logged in for that realm
- At this point, your client code can handle the challenge, by providing authentication details, tokens, or similar
- After authenticating, Worklight can re-issue the original request and return a result from the back end service

# QUESTIONS?