Google
Developers

Upgrading to a Chrome Packaged App

This presentation on Youtube: http://www.youtube.com/watch?v=e0W2szZ2qhg

Joe Marini - Chrome Developer Advocate

Google I/O

About Me

Joe Marini

Developer Advocate - Google Chrome

http://plus.ly/joemarini

@joemarini

https://github.com/joemarini

Agenda

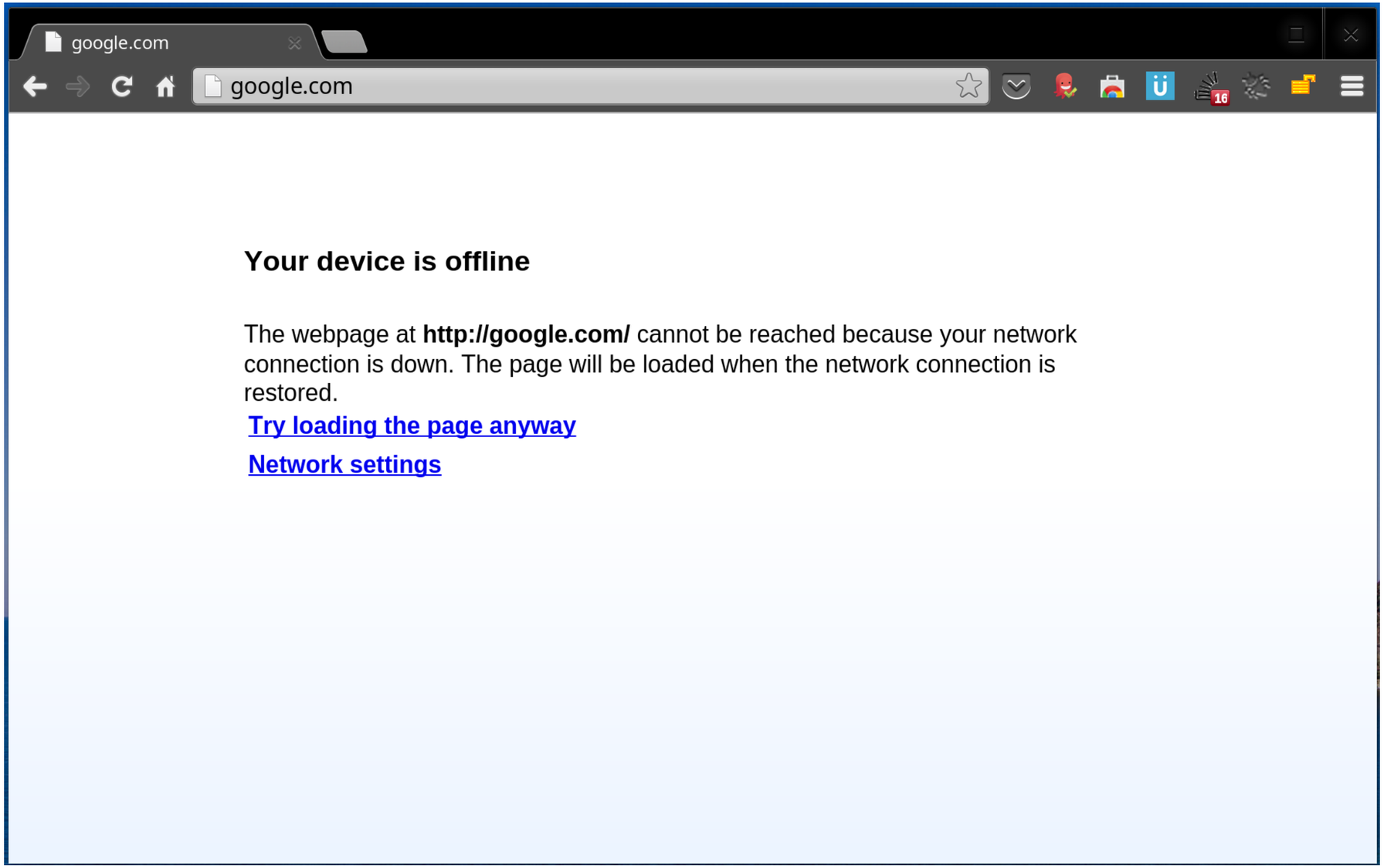
- Why build a packaged app?
- Structure of a packaged app
- Things you will encounter when upgrading to a Packaged App
  - Working offline
  - Using the cloud
  - Rich platform capabilities
  - Immersive UX
  - Application security
- Summary

Demos!

Why build a packaged app?

Web Apps are Great!

… Until they're not



Web Games are Great!

... Until they're not

Why build a packaged app?

Packaged Apps run offline by default

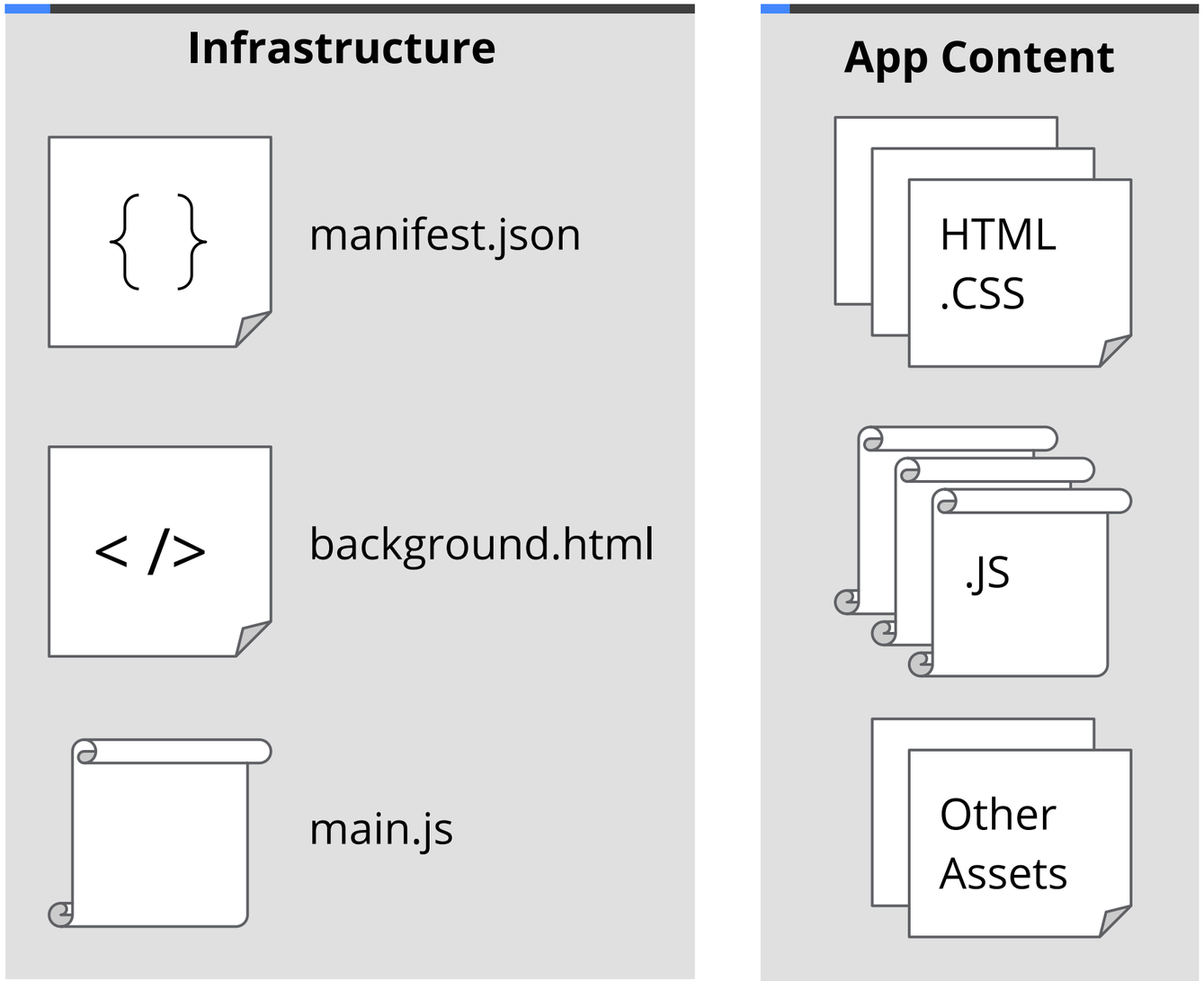
Access to platform capabilities and hardware

Rich, immersive user experience

Distribution and updates via the Chrome Web Store

From Web App to Packaged App

Packaged App Structure



Offline by default

All app resources are stored locally, and your app can be launched at any time

Implications:

- Disconnected experience, including first-run - resources (except video/audio) must be local
- Determining disconnected features
- Handling stale data and synchronization
- Storing data locally - window.localStorage not supported
    - *Synchronous APIs in general not supported - they block the UX thread*

Offline by default

Plan for a great offline experience:

- Factor your app to store resources locally (separate JS, CSS, HTML)
- Figure out what your app's offline features will be
- Use navigator.onLine and related events to update the UI
- Store data with `chrome.storage.local` or IndexedDB
- Use the HTML5 Filesystem API
- Set the `offline_enabled` flag in your manifest file

Example: chrome.storage API

JS

```
function saveChanges() {
  // Get a value saved in a form.
  var theValue = textarea.value;

  // Save it using the Chrome extension storage API.
  chrome.storage.local.set({'value': theValue}, function() {
    // Notify that we saved.
    message('Settings saved');
  });
}
```

Cloud by default

Users increasingly expect their data to be everywhere, and your apps need to address that expectation

Implications:

- Users will expect that their settings and data will sync
- Your app is responsible for handling conflicts
- Your app may be used across multiple OSes / form factors

Cloud by default

Leverage Chrome's cloud features:

- Use the `chrome.storage.sync` API to sync smaller data items
- Use the SyncFilesystem API to sync larger data files
  - Uses the Google Drive API as the backend, but is extensible
- Use the Push Messaging API to send messages from your server
- Use the Identity API to authenticate users
  - API specifically for Google services, another for 3rd party sites

Packaged App Cloud APIs

JS

```
chrome.syncFileSystem.setConflictResolutionPolicy('last_write_win'); // or 'manual'

chrome.syncFileSystem.requestFileSystem(function (fs) {
  if (chrome.runtime.lastError) {
      // handle any error
  }
  onFileSystemOpened(fs, true);
});
```

JS

```
chrome.experimental.identity.getAuthToken({ 'interactive': true }, onGetAuthToken);

chrome.experimental.identity.launchWebAuthFlow(
  {'url': '<url-to-do-auth>', 'interactive': true},
  function(redirect_url) { /* Extract token from redirect_url */ });
```

Immersive user experience

Packaged Apps live outside the browser, and can thus have richer user interfaces and experiences

Implications:

- Your app is responsible for things that it wasn't before
- Users have different expectations of apps vs. web sites

Immersive user experience

Build a great user experience:

- Use the windowing API to manage your app's windows
    - Use the screen size to determine initial window size/position
    - Remember window location/size for the next time the app is run
- Your app can control whether the default OS title bar is shown
- Use "`-webkit-app-region: drag`" to define custom drag regions
- Use chrome.contextMenus API to implement context menus

Access to platform and hardware

Chrome Packaged Apps are able to get access to the native hardware platform - files, USB, Bluetooth, Sockets

Implications:

- Privacy and security are even more important
- Be clear to the user about when you are using platform resources
- Remember to release resources that you are done with

Application security and CSP

Chrome Apps implement Content Security Policy, which has a direct impact on common Web app patterns

- Don't be `eval()`
- You can't use `new Function()`
- All JavaScript code must be in separate .js files
- No inline event handlers or embedding content in <iframe> tags
- You need to declare where your content comes from, if not local

Example: Use separate JS files and event handlers

HTML

```
<script>
function onWindowLoad() { ... }
</script>
<body onload="onWindowLoad">
</body>
```

JS

```
<script src="myjscode.js"></script>
window.addEventListener("load", onWindowLoad)
function onWindowLoad(evt) {
  // perform some initialization
}
```

Application Security and CSP

Embedding Web content

- Use the <webview> control to embed content

Accessing Remote Resources

- Fetch remote resources with XHR, then use `blob:`, `data:`, `filesystem:` URLs

Using Templating Libraries

- Use libraries that precompile templates
- Use a sandboxed page to host code that uses eval or new function()

Summary

Offline	• Factor your app's features • Identify offline capabilities • Store resources locally
Cloud	• Use chrome's sync features • SyncFilesystem, chrome.storage.sync
Immersive UX	• Your app controls its own windows and UX • Remember your window position, app state, etc
Platform Access	• A Real API for USB, Bluetooth, Sockets
Security	• Understand the Content Security Policy • Separate JS from HTML, no inline event handlers • Use Sandboxed pages for unsafe operations

<Thank You!>

Please remember to fill out session evaluation forms!

This presentation on Youtube: http://www.youtube.com/watch?v=e0W2szZ2qhg

http://plus.ly/joemarini

@joemarini

https://github.com/joemarini

Google
Developers