Remote addons support in the Mobile app

Issue: https://tracker.moodle.org/browse/MDL-49279

DOCS: https://docs.moodle.org/dev/Mobile_addons_support_in_Moodle_plugins

General ideas

We plan to support pulling app addons (AngularJS modules) from the remote site and lazy load them into the app. This will happen every time a user logs in into a site (or the remote site information is refreshed).

We need to find a way to make Moodle plugins able to declare that they "export" Moodle Mobile app addons.

The list of available app addons in a site should be exported via the core_site_get_site_info external function, so we need a way to retrieve a list of app addons provided by a Moodle site.

Any type of Moodle plugin must support app plugins (mod, local, admin tools, etc...) so we need to use a generic mechanism

Please, notice that this is only intended for Moodle additional (contributed) plugins, although it will also support core plugins. Core moodle plugins support will be implemented natively in the app.

Declaring app addons in Moodle

There are several options, we can:

- Use the db/services.php file
- Use a new db/mobile.php file
- Create a mobile/ directory in the plugin

db/services.php
Pros
Existing file, already included

Cons
It's not directly related to mobile, it's for exporting functions and services

db/mobile.php
Pros
Very easy to detect plugins that support this file using the get_plugin_list_with_file() function (that's how the db/cache.php definitions file works)

Cons
May need to create a new type of file

mobile/ directory in the plugin
Pros
No need to include new files
Cons
It will require iterate through all the plugins, it's not declarative

My view (Juan)
I think that it makes more sense a db/mobile.php file, it could be used for future features.

Storing and serving the app addons

Two options:

A - The AngularJS module packaged in zip format
Pros
It doesn't require to implement anything in Moodle, the file is served as is
Very easy to create a hash to check for new versions

Cons
Requires to handle the path to files (because the fonts or images or css or lang files would be stored in the sdcard)

B - A single file
Pros
Does not require to unzip in the app, supported in browsers without requiring zip libraries

Cons
Very hard to create a single file including all the required assets (css, images encoded, language files), should be created by Moodle? by the user?
It will also require to handle the lang files and any other information that will be stored in the sdcard, and to solve the paths too

My view (Juan):
I think option A makes a lot of sense right now for the SCORM player we currently support, because it unzips both in browser and app, and the app can be run in a browser because it now supports the file system.
The path could be very easily treated if we use a similar approach to what Moodle does using the @@PLUGINFILE@@ placeholder; we can use placeholders that will be replaced with the final location of the file once unzipped.

Caching addons

The addons information will be returned by the WS get_site_info, a new list containing the addons information:

```
addons [
 {
  component: “mod_certificate”, // The Moodle component
  addon: “mod_certificate”, // The addon name in the app
  package: “mod_certificate.zip” // The zip package name
  hash: “adf39fee…..” // The hash of the package file
  size: // the package size in bytes
  dependencies: list of app addons this addon depends on
  version: (the Moodle plugin version number)
 }
]
```

Since the zipped plugins should be very small, I think we could be consistent with what Moodle does and create a hash of the file.
The hash of the file can be calculated every time the file is served or when the plugin is upgraded.

If we only want to calculate the hash when the plugin is upgraded we would have some issues, for example:
- We'd have to create new tables in the database to store the packages information including the hashed value
- We'd need to add new code in the plugins install/upgrade process

We shouldn't use Etags neither because it will require a PHP file to serve the plugin hash files instead a direct download. Since the Web Service get_site_info will return the list of mobile addons including the hashes I don't see necessary to use tags.

We could use the Moodle cache for caching the hashes of the files, using a key like: component + addon + package + version

My opinion (Juan), I think that the simple implementation would be just calculate at run-time the file hashes since they MUST be very small and we can use the Moodle cache. I think that we could event limit the size of the allowed addons.

Multiple addons per plugin.

Should we allow multiple addons per Moodle plugin? Or should be allow just one since a mobile addon support multiple handles?

My opinion (Juan) - Just one addon per plugin, if the app requires multiple plugin the programmer can package them into local plugins and use the dependencies setting

Implementation

My proposed implementation (Juan)

Addons declaration:

plugin/db/mobile.php

Example

```
<?php

$addon = array(
    "name": "mod_certificate", (addon name in the app, mod_certificate, progress_bar, ….)
    "package": "mod_certificate.zip", (the package file name in the plugin/mobile directory)
    "dependencies": array("addonname1","addonname2")
```

);

web service get_site_info

Returns the addon structure after searching plugins implementing db/mobile.php via get_plugin_list_with_file:

```
addons [
 {
  component: “mod_certificate”, // The Moodle component
  addon: “mod_certificate”, // The addon name in the app
  package: “mod_certificate.zip” // The zip package name
  hash: “adf39fee…..” // The hash of the package file
  size: // the package size in bytes
  dependencies: list of app addons this addon depends on
  version: (the Moodle plugin version number)
 }
]
```

The hash will be retrieved from cache (using a cache key like component + addon + package + version) and if’s not possible, re-generated at run-time

In this initial phase, do not implement any UI in Moodle to list the plugins supporting mobile plugins (because we want to migrate all the mobile settings to a new admin tool in Moodle 3.2)

Angular side

Fred: Imagining I access two sites, both have hotpot but both sites are on different Moodle/Addon versions. When I access the first site we’ll load the hotpot addon which may override some existing services, etc… when I switch to the other site, how do you handle the overridden addons, and how do you handle both instances of the same addon having been loaded?