Best Practices for APIs in REDCap

Statement from REDCap Developer's at Vanderbilt:
Direct database back-end access in REDCap should not be allowed. Not only does back-end access bypass REDCap's built-in logging abilities (thus creating compliance issues), but it is likely to cause permanent damage to data. REDCap’s data model is very complex and is always changing, so a simple mistake could cause great damage to valuable data. Additionally, some back-end queries that might be correct now might no longer be correct in a future version of REDCap as the data model changes with every new version. For all these reasons, we (the REDCap developers at Vanderbilt) only recommend using front-end methods in the application (website and API) for extracting data. Any other method is not recommended.

The only method of interfacing data for the UT Southwestern Instant of REDCap that is supported is the built-in API functionality using the secure API key. This is the only way to comply with the requirements of the Health Insurance Portability and Accountability Act of 1996 (Title 45, Parts 160 and 164, subparts A and E of the Code of Federal Regulation (the “Privacy Rule”). The following is some best practice guidelines for writing APIs for REDCap.

1. Use the API Playground if possible to generate your code. Why struggle when a computer can struggle for you?
2. API Export and Import use the same data structure formats. If you have trouble having API Import's data to be formatted properly, do an API Export. Make the data structure viewable in some form via whatever language you want (json_encode for PHP, JSON.stringify for JavaScript, etc.). Then modify the data values logically to form a new JSON to put into an API Import call.
3. Put the data into a JSON. Do not leave it in its native structure. For some languages, like JavaScript, this is not a big deal. For other languages that do not use JSON-compatible data structures (like PHP, Objective-C, Swift, etc.), putting the native structure into the POST command dooms you to failure. Just throw it into a JSON-encoded structure (which most likely will be a string in standard JSON format), and put it into the data field. I promise, it'll work.
4. API JSON data structures are in arrays ([{ ... }]). If you leave it as a hash ({ ... }), your call will not succeed. Easy to say, easy to forget.
5. API Import calls have a data parameter; API export calls do not. If the content is the same, they are probably the same except for that data parameter. Save yourself a little time.
6. Decode JSONs in PHP using json_decode(json, true). If you leave off the parameter with the true, PHP will attempt to decode the JSON as a hash; since it is not a hash, it will fail and leave you wondering like you've been stood up on a date. Always, always put the true as the second parameter when dealing with REDCap.
7. Keep in mind memory limitations when the API will be communicating with a phone or tablet. You may need to change the synchronization process to proceed one record at a time. Sometimes you have to break up API Import or Export calls.
8. Any mobile calls are almost certainly cross domain. Turn this on if it is available on your program (as it was available with my on JavaScript's AJAX calls). Read up on JSONP if you wish.
9. JSONs easier to handle than XML or CSV. To some extent, this is a personal preference, but JSON parsers seem more friendly than XML or CSV to me for computational processing. Of course, this depends on your application.
10. When using the same basic API code for a new project, duplicate it, then just change the PID and the API Token and you should be good to go.