GDPR For Plugin Developers

Moodle's Privacy API

Andrew Nicols

Senior Analyst, Developer, and Integrator

@_andrewrn_

#mootes18

Reminder about GDPR... (in case you missed it)

Grants some key rights, including:

- Right of access
- Right to erasure
- Right to data portability

Also:

- Privacy by Design

Moodle and GDPR

- There is no magic one-click solution
    - Developers *will* have to make changes
    - Administrators *will* have to make changes
- GDPR is probably only the first - expect others!

Privacy API - Fundamentals

As a developer you must:

- Determine what data you hold
- Think about how much to export
- Think about how you can delete data

Privacy API - Fundamentals

Every plugin must:

- Create a `provider`

Use the provider to:

- Describe the kind of data the plugin holds
- Export any personal data
- Delete any personal data

Privacy API - Fundamentals

Implemented as a set of PHP Interfaces

- Strict contract with plugin developers
- Allows for easy deprecation
- Allows use of multiple interfaces

Privacy API - Fundamentals

Two sides to the privacy API:

- Metadata
    - Information about *what* data is stored
- Request
    - Subject Access Request
    - Right to be forgotten
    - Privacy by Design

Privacy API - Fundamentals

You must create a `provider` class

The class must:

- Have a namespace `plugin_frankenstyle\privacy`
- Be named `provider`
- Implement one `metadata` provider
- Implement zero or more `request` providers

Provider overview

There are two main categories:

- Those which do not store user data
- Those which store user data

Plugins which do not store user data

- Most plugins
- Easiest to implement
  - Metadata `null_provider`
  - Language string
- No other changes required

```php
<?php

namespace atto_undo\privacy;

class provider implements \core_privacy\local\metadata\null_provider {

    /**
     * Get the language string identifier with the component's language
     * file to explain why this plugin stores no data.
     *
     * @return  string
     */
    public static function get_reason() : string {
        return 'privacy:metadata';
    }
}
```

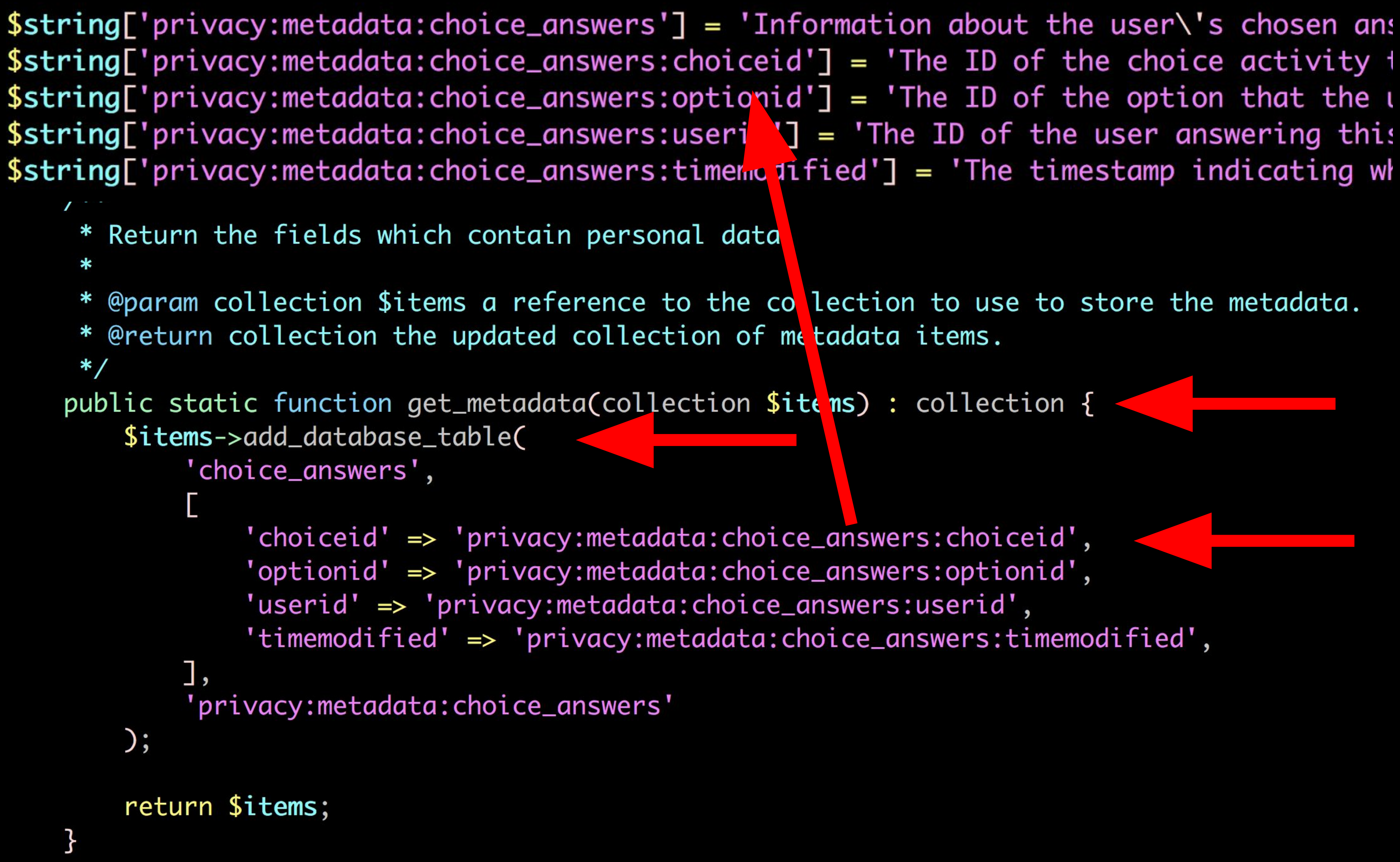
```php
<?php

$string['pluginname'] = 'Undo/Redo';
$string['privacy:metadata'] = 'The atto_undo plugin does not store any personal data.';
```

Plugins with user data

Metadata:

- Metadata `provider`
- Describe data
- Language strings



```
$string['privacy:metadata:choice_answers'] = 'Information about the user\'s chosen ans
$string['privacy:metadata:choice_answers:choiceid'] = 'The ID of the choice activity t
$string['privacy:metadata:choice_answers:optionid'] = 'The ID of the option that the u
$string['privacy:metadata:choice_answers:userid'] = 'The ID of the user answering this
$string['privacy:metadata:choice_answers:timemodified'] = 'The timestamp indicating wh
```

```
    /**
     * Return the fields which contain personal data.
     *
     * @param collection $items a reference to the collection to use to store the metadata.
     * @return collection the updated collection of metadata items.
     */
    public static function get_metadata(collection $items) : collection {
        $items->add_database_table(
            'choice_answers',
            [
                'choiceid' => 'privacy:metadata:choice_answers:choiceid',
                'optionid' => 'privacy:metadata:choice_answers:optionid',
                'userid' => 'privacy:metadata:choice_answers:userid',
                'timemodified' => 'privacy:metadata:choice_answers:timemodified',
            ],
            'privacy:metadata:choice_answers'
        );

        return $items;
    }
```

```
// There are several user preferences.
$items->add_user_preference('maildigest', 'privacy:metadata:preference:maildigest');
$items->add_user_preference('autosubscribe', 'privacy:metadata:preference:autosubscribe');
$items->add_user_preference('trackforums', 'privacy:metadata:preference:trackforums');
```

```
$collection->add_external_location_link('googledrive', [
    'params' => 'privacy:metadata:fileconverter_googledrive:params',
    'filecontent' => 'privacy:metadata:fileconverter_googledrive:filecontent',
    'filemimetype' => 'privacy:metadata:fileconverter_googledrive:filemimetype',
], 'privacy:metadata:fileconverter_googledrive:externalpurpose');
```

```
// The quiz links to the 'core_question' subsystem for all question functionality.
$items->add_subsystem_link('core_question', [], 'privacy:metadata:core_question');

// The quiz has two subplugins..
$items->add_plugintype_link('quiz', [], 'privacy:metadata:quiz');
$items->add_plugintype_link('quizaccess', [], 'privacy:metadata:quizaccess');
```

Plugins with user data

Request providers:

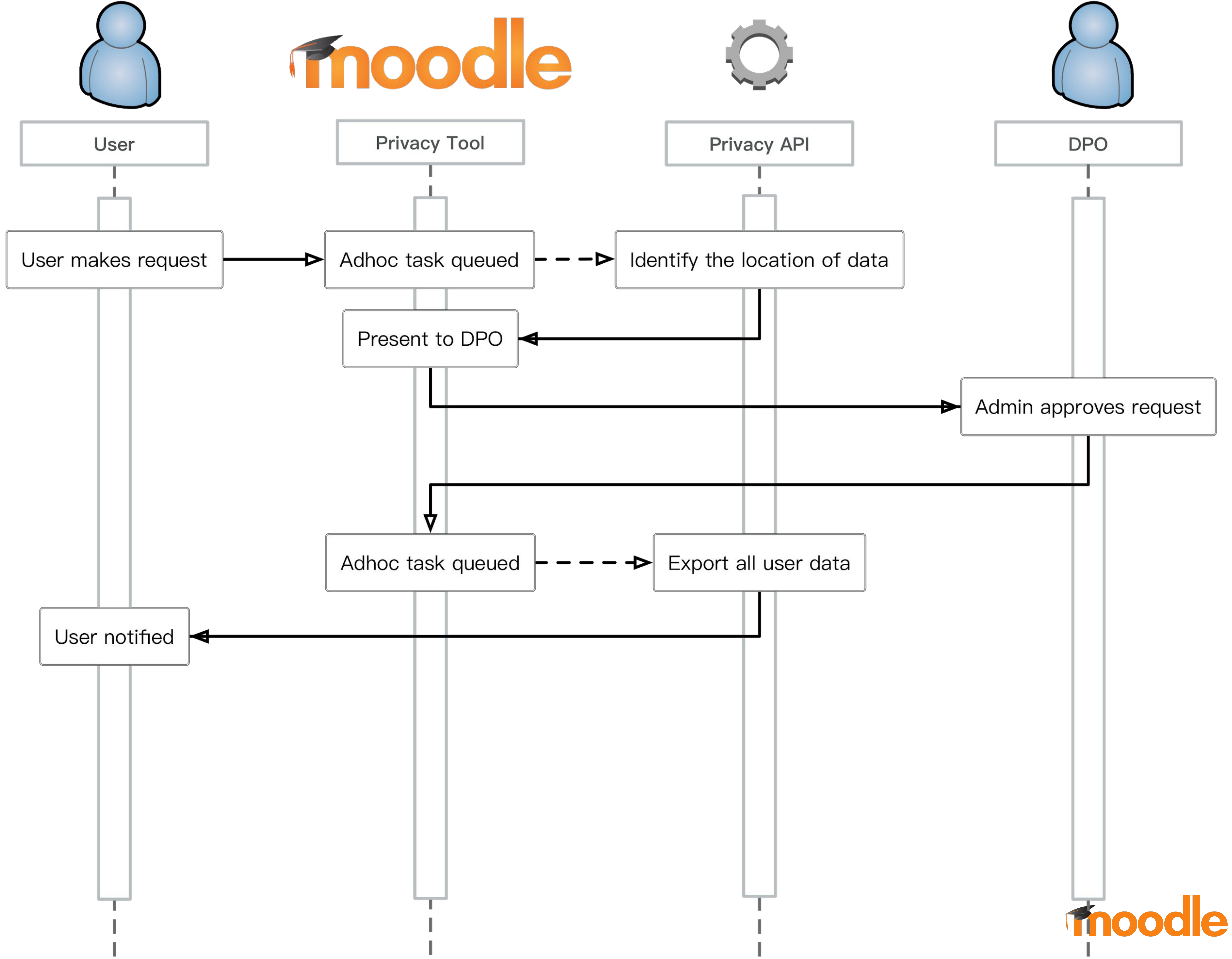
- User preference provider
- Plugin provider
- Subsystem plugins (Portfolio, Plagiarism)
- Subplugins

Privacy API - good examples

- User preferences:
  - `tool_usertours`
  - `theme_boost`
- Activities:
  - `mod_survey`
  - `mod_choice`

Privacy API - good examples

- Blocks:
  - `block_comments`
- Subplugins:
  - `assignsubmission_file`
- Others:
  - `repository_googledocs`


User
Privacy Tool
Privacy API
DPO
User makes request
Adhoc task queued
Identify the location of data
Present to DPO
Admin approves request
Adhoc task queued
Export all user data
User notified

Plugins which store data

- Must identify where data is stored for a user (`context`)
- Must export any user data, plus any additional data to give that data relevance
- Must delete user data in a specific context *where it does not affect other users*
- Must delete all user data for a specific context

```
/**
 * Get the list of contexts that contain user information for the specified user.
 *
 * @param   int           $userid     The user to search.
 * @return  contextlist   $contextlist  The contextlist containing the list of contexts used in this plugin.
 */
public static function get_contexts_for_userid(int $userid) : \core_privacy\local\request\contextlist {
    $sql = '...';
    $params = [];

    $contextlist = new \core_privacy\local\request\contextlist();
    $contextlist->add_from_sql($sql, $params);

    return $contextlist;
}
```

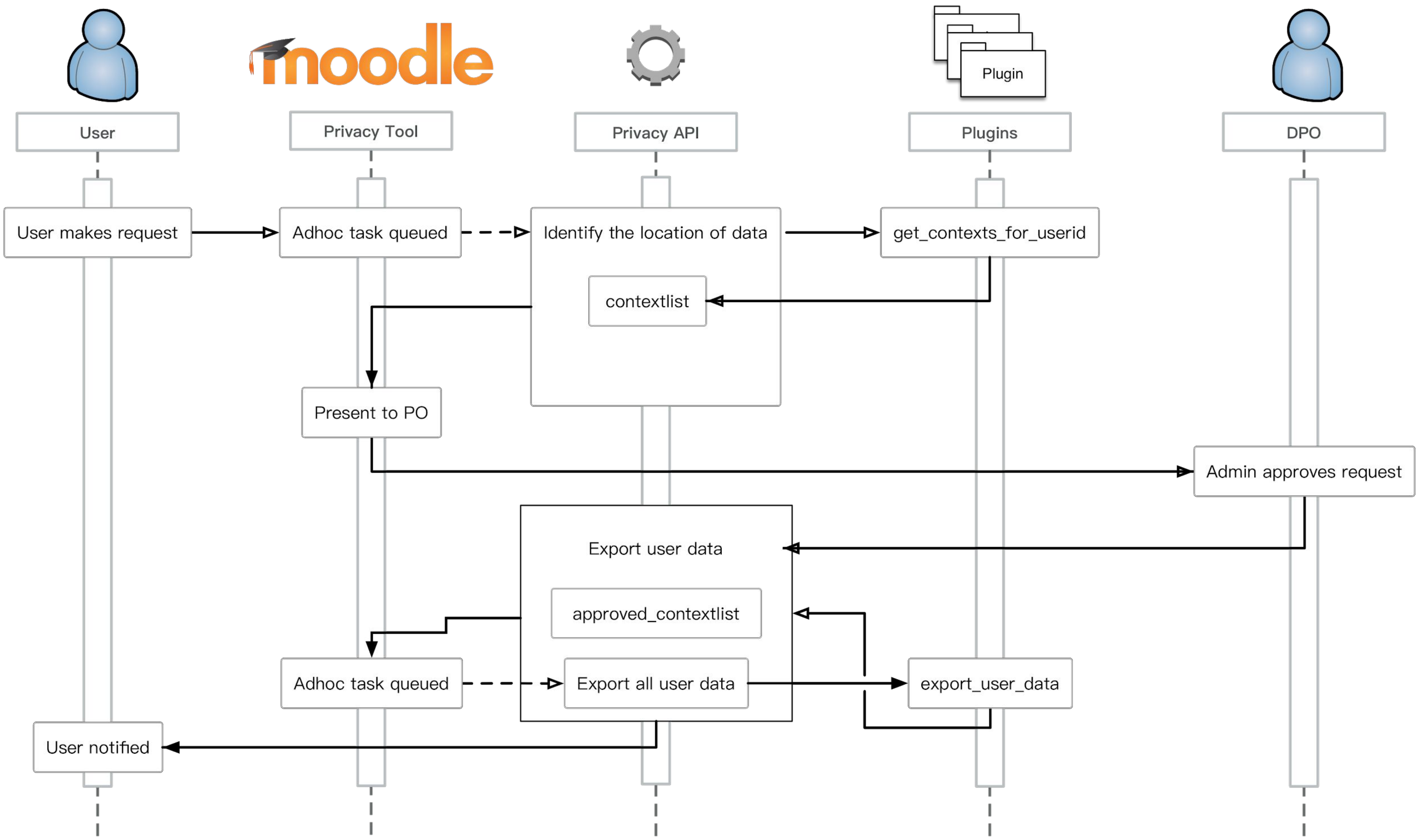
Privacy API - Identifying location of data

Why SQL?

- Forces developers to think about how they structure their provider code
- Limits use of APIs (which is normally a bad thing)
  - Many APIs perform capability checks
  - Many APIs are not good at fetching data in bulk
- Allows for future restructuring

Plugins which store data

- ~~Must identify where data is stored for a user (`context`)~~
- Must export any user data, plus any additional data to give that data relevance
- Must delete user data in a specific context *where it does not affect other users*
- Must delete all user data for a specific context



Plugin
User
Privacy Tool
Privacy API
Plugins
DPO
User makes request
Adhoc task queued
Identify the location of data
get_contexts_for_userid
contextlist
Present to PO
Admin approves request
Export user data
approved_contextlist
Adhoc task queued
Export all user data
export_user_data
User notified

```
/**
 * Export all user data for the specified user, in the specified contexts.
 *
 * @param   approved_contextlist    $contextlist    The approved contexts to export information for.
 */
public static function export_user_data(approved_contextlist $contextlist) {
    global $DB;

    $user = $contextlist->get_user();

    list($contextsql, $contextparams) = $DB->get_in_or_equal($contextlist->get_contextids(), SQL_PARAMS_NAMED);
    $contextparams['userid'] = $contextlist->get_user()->id;
```

```
foreach ($autosaves as $autosave) {
    $context = \context::instance_by_id($autosave->contextid);
    $subcontext = [get_string('autosaves', 'editor_atto'), $autosave->id];

    $html = writer::with_context($context)
        ->rewrite_pluginfile_urls($subcontext, 'user', 'draft', $autosave->draftid, $autosave->draftte

    $data = (object) [
        'drafttext' => format_text($html, FORMAT_HTML, static::get_filter_options()),
        'timemodified' => \core_privacy\local\request\transform::datetime($autosave->timemodified),
    ];

    if ($autosave->userid != $user->id) {
        $data->author = \core_privacy\local\request\transform::user($autosave->userid);
    }

    writer::with_context($context)
        ->export_data($subcontext, $data)
        ->export_area_files($subcontext, 'user', 'draft', $autosave->draftid);
```

```
foreach ($autosaves as $autosave) {
    $context = \context::instance_by_id($autosave->contextid);
    $subcontext = [get_string('autosaves', 'editor_atto'), $autosave->id];

    $html = writer::with_context($context)
        ->rewrite_pluginfile_urls($subcontext, 'user', 'draft', $autosave->draftid, $autosave->draftte

    $data = (object) [
        'drafttext' => format_text($html, FORMAT_HTML, static::get_filter_options()),
        'timemodified' => \core_privacy\local\request\transform::datetime($autosave->timemodified),
    ];

    if ($autosave->userid != $user->id) {
        $data->author = \core_privacy\local\request\transform::user($autosave->userid);
    }

    writer::with_context($context)
        ->export_data($subcontext, $data)
        ->export_area_files($subcontext, 'user', 'draft', $autosave->draftid);
```

Privacy API - Exporting data

Why `writer`?

- The current exporter is a basic exporter
- We plan to add an HTML exporter
- Writer is mocked during testing
- Forces structure in exports

```
$discussiondata = (object) [
    'name' => format_string($discussion->name, true),
    'pinned' => transform::yesno((bool) $discussion->pinned),
    'timemodified' => transform::datetime($discussion->timemodified),
    'usermodified' => transform::datetime($discussion->usermodified),
    'creator_was_you' => transform::yesno($discussion->userid == $userid),
];
```

Privacy API - Export transformations

Why transform?

- Data should be meaningful and (theoretically) machine readable
- Sometimes we need to provide data in dual formats
- Allows for changes in future (possibly dependant upon the writer format)

Privacy API - What else..?

- Everything is strongly typed (yay PHP 7!)
  - Don’t forget about PHP 5.6 - if you need to support it, check out the `legacy_polyfill`
- Unit tests are awesome - easiest way to catch mistakes in your provider

Privacy API - Where next?

Docs: https://tinyurl.com/privacy-api

Utilities: https://tinyurl.com/privacy-utilities

Follow-up issues: MDL-62331

Screencast: https://youtu.be/jcVKzq2qimQ moodle

Empowering educators to improve our world