Moodle an Hochschulen

Community-Workshop "Moodle Plugin-Entwicklung" 15.9.2022

Agenda

1 Entwickeln für Moodle – grundlegende Kategorien

- Moodle Core Contributions (https://moodledev.io/general/development/process)
  - perspektivisch diskutieren: wie bekommen wir Code für die offenen Issues
- Moodle Plugins (Thema dieses Workshops)
- Moodle Schnittstellen (insbesondere LTI)
- Moodle Core Hacks (Don't try this at home!)
  - wenn, dann mit Git, um Upgrade-Workflows nicht zu zerstören oder sogar ein Plugin entwickeln statt eines Core-Hacks
- Neuer Einstieg für Entwickler-Doku und Release notes:
  - https://moodledev.io/
  - https://moodledev.io/general/releases
  - aber: noch nicht alles migriert vom alten Ort: https://docs.moodle.org/dev/Main_Page
  - auch anschauen: https://moodle.academy/

2 Die verschiedenen Plugin-Typen in Moodle und deren zielgerichteter Einsatz

- https://moodledev.io/docs/apis/plugintypes/
- Aktivitäts-Plugins
  - mod_*
  - Achtung: "mod" steht _nicht_ für ein allgemeines Plugin
  - https://moodledev.io/docs/apis/plugintypes/mod
- Blöcke
- block_*
- Achtung: Der Einsatz von Blöcken war zeitweise discouraged vom Moodle HQ. Der Einsatz will gut bedacht sein.
  - aber: guter Einstieg für erste Erfahrungen in der Moodle-Programmierung, in der alten Doku:
    - https://docs.moodle.org/dev/Blocks
    - https://docs.moodle.org/dev/Blocks_Advanced
- Blöcke funktionieren in der App nur mit Zusatzcode.
- https://moodledev.io/docs/apis/plugintypes/blocks
  - wichtig: Einstiegspunkt bei Block-Plugins ist die Datei "block_pluginname.php"
- Question Types
  - qtype_*
  - https://docs.moodle.org/dev/Question_types
  - Im Zweifelsfall (wie bei anderen Plugins) an vorhandenen (Core-)Plugins orientieren, um eventuelle Lücken in der Doku zu füllen.
- Filter
  - filter_*
  - https://moodledev.io/docs/apis/plugintypes/filter
- Kursformate
  - format_*
  - https://moodledev.io/docs/apis/plugintypes/format
- Themes
  - theme_*
  - Themes sind auch 'nur' Plugins, das ist aber ein Thema für einen anderen Workshop
  - https://docs.moodle.org/dev/Themes
- Es gibt viele Stellen an denen man in Moodle mit einem Plugin einhaken kann
  - Core Hacks sehen vielleicht wie die schnelle Lösung aus, sind aber häufig nicht notwendig und vom langfristigem Aufwand (Berücksichtigung bei Updates...) nicht nachhaltig.
- Plugin Typen nicht mit Core API Typen (https://docs.moodle.org/dev/Core_APIs) verwechseln
- Diskussion: Moodle plugin skeleton generator (https://moodle.org/plugins/tool_pluginskel) vs. anschauen was es gibt bzw. etwas das vor Kurzem überarbeitet wurde (Forum und BBB sind vor kurzem überarbeitet worden und damit tendentiell gute Anschauungsbeispiele)
- Moodle Developer-Chat:
  - https://docs.moodle.org/dev/Chat
  - https://telegram.me/moodledev

3 Das local-Plugin als "Allzweckwaffe"

- Abgrenzung zwischen local_* und tool_*
  - local Plugins waren vorher da, Admin Tools kamen deutlich später
  - Admin tools sind semantisch auf administrative oder konfigurierende Ziele ausgerichtet und haben in den allermeisten Fällen auch irgendeine GUI
  - Local Plugins können alles was Admin tools können, aber fungieren darüber hinaus noch als Resterampe: In einem local Plugin kann man praktisch alles verpacken, was woanders keinen richtigen Platz hat
  - https://moodledev.io/docs/apis/plugintypes/local und https://github.com/moodle/moodle/blob/master/local/readme.txt
  - https://docs.moodle.org/dev/Admin_tools
- Harmoniert auch prima mit Event Listenern, Callbacks, Hooks, ...
  - Themen für weitere Workshops

4 Die allgemeine Ordner- & Dateistruktur von Moodle

- Blick auf https://github.com/moodle/moodle
- upgrade.txt dokumentiert Änderungen aus jeweiligen Major-Releases, z.B. "=== 3.9. ==="
- Ganz klar: Gewachsene Struktur
  - git hilft, die Struktur und deren Wachstum nachzuvollziehen, bspw. History/Blame
- Man sieht aber auch Unterordner, die den Namen von Plugintypen entsprechen
- Jeder Plugintyp hat seinen festgelegten Speicherort (https://github.com/moodle/moodle/blob/master/lib/components.json)
- Jedes Plugin kann Subplugins definieren (db/subplugins.json), z.B. für mod_quiz:

```
{
    "plugintypes": {
        "quiz": "mod\/quiz\/report",AwZ
        "quizaccess": "mod\/quiz\/accessrule"
    }
}
```

5 Typische Dateien, die man in jedem Plugin findet, und ihr Zweck

- https://moodledev.io/docs/apis/commonfiles
- version.php
  - Ohne diese Datei ist es kein Plugin - Voraussetzung für Funktionieren und Installation
  - enthält Metadaten
  - Pflichtangaben und optionale Angaben
  - Minimalbeispiel: https://github.com/moodle/moodle/blob/master/mod/forum/version.php
  - Maximalausbau: https://github.com/moodle-an-hochschulen/moodle-theme_boost_union/blob/master/version.php
  - Versionsnummer für tägliches Entwickeln sehr relevant (bspw. Update von Sprachdateien nur bei neuer Version)

    - Versionsnummer-Schema i.d.R.: JahreszahlMonatTag, z.B. 20220915 bzw. 2022091500
    - component: "Frankenstyle", wird genutzt zur Referenzierung an verschiedenen Stellen, Beispiel: mod_forum
    - supported: Hinweise, unter welchen Versionen das Plugin lauffähig ist - auch wichtig fürs automatisierte Veröffentlichen auf moodle.org
    - Versionsnummer der Moodle-Versionen unter: https://moodledev.io/general/releases
    - Zum Thema dependency auch interessant: Kommunikation zwischen verschiedenen Komponenten - was ist guter Stil (fortgeschrittenes Thema): https://moodledev.io/general/development/policies/component-communication
- lang/en/pluginname.php
    - Das Sprachpaket bzw. die Sprachpakete
- lib.php und locallib.php
    - Funktionen und Bibliotheken
- classes Ordner
    - Klassen und Bibliotheken
- settings.php
    - Admin Einstellungen des Plugins.
- db Ordner ("_d_ie _b_etriebsinformationen")
    - install.xml
        - Eigene Tabellen des Plugins
        - Hinweis: nicht immer eigene DB-Tabellen notwendig
            - User Preferences, Aktivitätseinstellungen, Favoriten
    - install.php und uninstall.php (optional)
        - Installationsscript und Uninstall Script
    - upgrade.php (optional)
        - Upgrade Script: https://moodledev.io/docs/guides/upgrade/#dbupgradephp
    - access.php
    - Capabilities (siehe https://moodledev.io/docs/apis/subsystems/roles)
    - tasks.php
        - Scheduled Tasks
        - Hier werden die Tasks nur registriert, die Implementierung liegt dann in classes/task
        - https://moodledev.io/docs/apis/commonfiles/db-tasks.php
        - Beispiel: https://github.com/moodle/moodle/blob/master/mod/forum/db/tasks.php
    - events.php
        - Event observer / event listener des Plugins
        - https://docs.moodle.org/dev/Events_API#Event_observers
        - Beispiel: https://github.com/moodle/moodle/blob/master/mod/forum/db/events.php

    - messages.php
        - Nachrichten, die das Plugins verschickt.
        - https://docs.moodle.org/dev/Message_API
        - Beispiel: https://github.com/moodle/moodle/blob/master/mod/forum/db/messages.php
    - services.php
        - Webservices, die das Plugin implementiert
        - https://docs.moodle.org/dev/Adding_a_web_service_to_a_plugin
        - Beispiel: https://github.com/moodle/moodle/blob/master/mod/forum/db/services.php
        - Ist insbesondere für die Mobile App relevant, aber das ist ein anderer Workshop
- amd
    - JavaScript Module
    - Wird mit Grunt kompiliert, es gibt also src und build Unterordner
    - https://moodledev.io/docs/guides/javascript
    - Beispiel: https://github.com/moodle/moodle/tree/master/mod/forum/amd
    - In legacy Plugins gibt es auch noch YUI, das ist aber eine Altlast -> bitte kein YUI mehr verwenden!
- backup
    - Backup Routinen, falls das Plugin diese nutzen möchte. Relevant insbesonders für Aktivitäten. Muss definiert werden, wenn Plugin für Backups berücksichtigt werden soll.
    - https://docs.moodle.org/dev/Backup_2.0_for_developers
    - Beispiel: https://github.com/moodle/moodle/tree/master/mod/forum/backup/moodle2
- styles.css
    - CSS Datei, die pluginspezifische CSS Regeln zum ausgelieferten Theme hinzufügt.
    - Unterstützt aktuell nur CSS, kein SCSS, wird also nicht kompiliert sondern nur konkateniert
    - https://docs.moodle.org/dev/Plugin_contribution_checklist#CSS_styles
    - Beispiel: https://github.com/moodle/moodle/blob/master/mod/forum/styles.css
- pix
    - Ordner für Bilder
    - Für Bilder wie Aktivitätsicons gibt es vorgegebene Konventionen, ansonsten Ablage frei Schnauze
    - Beispiel: https://github.com/moodle/moodle/tree/master/mod/forum/pix
- templates
    - Ordner für mustache-Templates mit deren Hilfe Daten in HTML gesetzt werden
    - https://moodledev.io/docs/guides/templates
    - Vortrag von David Mudrak auf der MoodleDach 2019 zum Thema "Outputting HTML in Moodle":
        - https://www.youtube.com/watch?v=xDrjK1mcu0Y
- Man findet diese typischen Dateien nicht nur innerhalb von Plugins, sondern auch teilweise im Ordner lib für den Moodle Core als Ganzes

6 Definition und Nutzung von Admin Settings

- Beispiel: https://github.com/moodle-an-hochschulen/moodle-block_cohortspecifichtml/blob/master/settings.php
  - Achtung beim Namen, unter dem eine Einstellung gespeichert wird, auf den Slash im String achten: [componenten_name]/[einstellungs_name]
- Vordefinierte admin_setting_* Klassen für diverse Admin-Settings-Typen, der Entwickler muss sich eigentlich nur überlegen _was_ oder unter welchem Namen er speichern will
  - beim Ergänzen neuer Settings muss die version in der version.php hochgesetzt werden
- Abfrage der Einstellungen im Plugincode mit get_config()
- https://docs.moodle.org/dev/Admin_settings
- Falls es keine passende admin_setting Klasse gibt, kann man eigene schreiben und nutzen, Beispiel https://github.com/moodle-an-hochschulen/moodle-local_staticpage/blob/master/classes/admin_setting_staticpagestoredfile.php
- Settings.php wird, genauso wie lib.php bei jedem Seitenaufruf mit eingebunden, die Abfragen if ($ADMIN->fulltree) und if ($hassiteconfig) sind daher performance-relevant
  - ohne diese Checks würden für Admins/Manger die Settings immer vollständig gelanden werden, auch beim normalen surfen in Moodle außerhalb der Website-Administration
- Einschub: defined('MOODLE_INTERNAL') || die;
  - Moodle erwartet die Einbindung der config.php, wenn Seiten über den Browser aufgerufen werden sollen
  - Dateien wie die settings.php werden intern benutzt und sollen nicht direkt aufgerufen werden, hier ist defined('MOODLE_INTERNAL') || die; zu verwenden

7 Definition und Nutzung von Capabilities

- Definition in access.php
- https://moodledev.io/docs/apis/subsystems/access und https://docs.moodle.org/dev/NEWMODULE_Adding_capabilities
- Beispiel: https://github.com/moodle/moodle/blob/master/mod/forum/db/access.php
- Capabilities haben mindestens einen Namen, einen captype und einen contextlevel
- Context-Level: https://docs.moodle.org/dev/Roles_and_modules#Context
- Capabilities werden im Code mit has_capability() abgefragt
- Man kann damit aber auch Scripte als ganzes schützen mit require_capability()
- Je nach Plugintyp gibt es capabilites, die vorhanden sein müssen, damit ein Plugin korrekt funktioniert
  - bspw. "addinstance" bei mod plugins
  - welche capabilities dies sind steht (hoffentlich) in der Doku, ggf. bei Core-Plugins nachschauen

8 Definition und Nutzung von eigenen Datenbanktabellen

- install.xml kann es definieren
- Wird mit XMLDB-Editor gebaut, besser nicht von Hand editieren
- https://docs.moodle.org/dev/Using_XMLDB

- Daten abrufen: https://docs.moodle.org/dev/Data_manipulation_API
- Beispiel: https://github.com/moodle/moodle/blob/master/mod/forum/db/install.xml
- ACHTUNG: Fremdschlüssel sind in Moodle nicht implementiert, d.h. bspw. bei mod-Plugins auch die Delete-Instance Methode implementieren und sich selbst um die abhängigen, nicht mehr benötigten Daten kümmern
- admin/cli/check_database_schema.php
  - regelmäßig nutzen, um zu überprüfen, ob die XMLDB-Definitionen auch in der DB angekommen sind
- Index-Definitionen nicht vernachlässigen, um Performanceeinbrüche bei großen Tabellen zu vermeiden
- siehe auch XMLDB-Editor: https://docs.moodle.org/dev/XMLDB_editor (prüfen / check von Indizes, Fremdschlüsseln u.a.)
- Mit der DB kommunizieren über das $DB Objekt, die Data manipulation API
  - https://moodledev.io/docs/apis/core/dml

9 Installations- und Upgrade-Skripte von Plugins

- Install / Uninstall
  - Viele Dinge werden automatisch installiert und auch wieder entfernt (wie Datenbanktabellen aus install.xml), manches muss man aber auch von Hand in diesen Dateien erledigen
- Upgrade
- Achtung: Änderungen an der Tabellenstruktur in install.xml müssen für existierende Installationen hier in PHP dupliziert werden
  - code für upgrade.php über den xmldb-editor generieren
- https://docs.moodle.org/dev/Upgrade_API
- Beispiel: https://github.com/moodle/moodle/blob/master/mod/forum/db/upgrade.php
  - upgrade.php kann auch genutzt werden um _wichtige_ Informationen an Admins auszugeben, ggf. notwendige manuelle post-upgrade Schritte.

10 Libraries vs. Classes -> Wo liegt eigentlich unsere Funktionalität?

- Libraries
  - lib.php wird bei jedem Seitenaufruf mit eingebunden, sollte daher nur wirklich relevante Funktionen, die der Moodle Core dort erwartet, enthalten
  - -> keine eigenen Funktionen, nur Implementierungen von Funktionen, die der Moodle Core auffruft
  - https://moodledev.io/docs/apis/commonfiles#libphp
  - Performance impact!
  - locallib.php enthält dann alle weiteren, pluginspezifischen Funktionen. locallib.php muss von Hand im Plugin required werden. Neuer Code sollte aber eigentlich gar keine locallib.php mehr nutzen, sondern mit classes arbeiten
  - Beispiel: https://github.com/moodle-an-hochschulen/moodle-block_cohortspecifichtml/blob/master/lib.php und https://github.com/moodle-an-hochschulen/moodle-block_cohortspecifichtml/blob/master/locallib.php

- Classes
  - privacy Ordner
    - https://moodledev.io/docs/apis/subsystems/privacy/
  - Diverse Ordner für weitere Moodle Core APIs
  - Eigene Ordner und Klassendateien für pluginspezifische Klassen
    - https://docs.moodle.org/dev/Automatic_class_loading
    - bspw im classes/local Ordner

11 Sprachpakete

- Englisch ist Pflicht, andere Sprachen optional (und idealerweise eigentlich über AMOS)
- Datei mit Key-Value Paaren
- Wird mit get_string() im Moodle Plugin ausgelesen
- Es gibt mehrere obligatorische Strings wie "pluginname" oder die Beschreibung für Capabilities sowie auch magische Strings wie *_help
- https://docs.moodle.org/dev/String_API
- Beispiel: https://github.com/moodle/moodle/blob/master/mod/forum/lang/en/forum.php

12 Ausgabe von Text und Inhalt in der GUI

- Viele Plugin Typen haben dafür vorgesehene Dateien zur Implementierung der Pluginansicht, die dann auch pluginspezifisch implementiert werden müssen und pluginspezifisch ausgewertet werden
  - Beispiel für einen Block: https://github.com/moodle-an-hochschulen/moodle-block_people/blob/master/block_people.php
  - Beispiel für eine Aktivität: https://github.com/moodle/moodle/blob/master/mod/forum/view.php
- format_text() und format_string()
  - https://moodledev.io/docs/apis/subsystems/output#format_text
- Mustache Templates
  - https://moodledev.io/docs/guides/templates

13 Veröffentlichung eines Plugins auf Github

- Github ist der De-facto-Standard für die Veröffentlichung von Moodle Plugins für die Community. Einen richtigen technischen Grund oder eine Policy gibt es dafür aber nicht.
- Für öffentliche Plugins genügt die kostenlose Ausgabe von Github vollkommen (der Verein hat auch keine Bezahl-Version)
- Neues Repo anlegen:
  - Namensgebung: moodle-plugintyp_pluginname, also zum Beispiel moodle-local_boostnavigation
  - Idealerweise eine nette Beschreibung und Tags wie 'moodle' und 'moodle-plugin' vergeben
  - Idealerweise die GPL als Datei mitliefern

- Entscheidung für Branch Strategie:
  - Ein Entwicklungsstrang im master-Branch
  - Mehrere Entwicklungsstränge in MOODLE_xxx_STABLE Branches, dabei dann https://docs.moodle.org/dev/Moodle_versions beachten
- Der Plugin Code wird ohne das Pluginverzeichnis und vor allem ohne den Moodle Core auf oberster Ebene committed
- README.md sinnvoll befüllen, da sie auf der Startseite des Repos dargestellt wird
- Beispiel: https://github.com/moodle-an-hochschulen/moodle-local_boostnavigation
- Tags können genutzt werden für die autom. Veröffentlichung auf moodle.org - in Zusammenspiel mit Release-Attribut in version.php
- Releases können mit Tags markiert werden (und darüber dann nach moodle.org/plugins gezogen werden), richtige Github Releases sind nett aber optional

14 Github Actions (ein erster Blick auf Moodle-Plugin-CI)

- Moodle Plugin CI ist die Standardlösung für automatisiertes testen / continuous integration von Moodle Plugins
- https://moodlehq.github.io/moodle-plugin-ci/
- Arbeitet mit Github Actions oder Travis CI. Ersteres ist für öffentliche Plugins komplett kostenlos nutzbar.
- Auch wenn man selbst keine automatisierten (Behat / PHPUnit) Tests im Plugin hat, lohnt sich die Nutzung von Moodle Plugin CI, denn
  - Es testet ob das Plugin auf einem nackten Moodle installierbar ist
  - Es testet ob die Coding Guidelines eingehalten werden
  - Es testet auf Wunsch mehrere Plugin- und PHP- und DB-Versionen in einem Rutsch
- Quick Start:
  - Github Actions im Github Repo aktivieren
  - Konfigurationsdatei https://github.com/moodlehq/moodle-plugin-ci/blob/master/gha.dist.yml in den Unterordner .github/workflows legen und gegebenenfalls anpassen
  - Neuen Commit pushen oder PR stellen
  - Ergebnis im Github Repo im Actions-Tab anschauen

15 Further Reading

- https://moodledev.io/ (die zukünftige Doku)
- https://docs.moodle.org/dev/Main_Page (die bisherige Doku)
- Readme-Dateien im jeweiligen Plugin-Root
- https://moodle.academy/
- Doku für Entwicklungsprozesse im Rahmen der Vereins-Plugins https://github.com/moodle-an-hochschulen/moodle-plugin-maintaining/wiki
  - zum Thema Qualitätsstandards siehe auch https://github.com/moodle-an-hochschulen/moodle-plugin-maintaining/wiki/Principles-and-approaches-for-building-our-Moodle-plugins#coding-principles