Moodle Development

Bei der Entwicklung von Code für Moodle sind die Ressourcen vom Moodle HQ sehr hilfreich, zuvorderst sind hier das "Developer Resource centre" und die "Moodle Academy" (Developer learning pathway) zu nennen. Zudem bietet der Community-Workshop „Moodle Plugin-Entwicklung“ einen guten Überblick.

Getting Started

Als zentraler Einstieg mit Verweis auf viele weitere Themen, wie z.B. Developer FAQ und Moodle Debugging, eignet sich Getting Started auf moodledev.io. Empfohlen werden der Moodle PHP CodeSniffer zur Prüfung auf die Coding-Standards und PHPdoc check plugin zur Einhaltung der Dokumentationsstandards. Zudem wird zur Nutzung der Community ermutigt, z.B. das Developer Forum oder der Matrix-Kanal.

Lokales Moodle

Entwickelter Code, wie z.B. Plugins, sollten auf einem lokalen Moodle-System getestet werden. Die einfachste Möglichkeit Moodle lokal aufzusetzen ist es, sich den Windows Moodle Installer in der gewünschten Version herunterzuladen und dann "Start Moodle.exe" auszuführen. Im Anschluss ist ein betriebsbereites Moodle mit MySQL, PHP und den Moodle-files unter localhost bereitgestellt. Dies hat den Vorteil, dass der Code lokal per IDE bearbeitet werden kann und nicht auf einem Server liegt. Die PHP Error-logs liegen dann unter "server/apache/logs/error.log".

Die andere Möglichkeit ist es Docker zu benutzen. Hierfür kann man sich auf moodle-docker auf github ein fertiges Set an Containern herunterladen. Darüber ist es auch möglich, mehrere verschiedene Moodle-Versionen parallel zu betreiben.
Ein weiterer Vorteil: **Behat- und PHPUnit-Tests** kann man mit diesem Docker-Setup sehr einfach laufen lassen.
Für Windows nutzt man aus Performancegründen am besten das WSL (Windows Subsystem für Linux).

Die Academy bietet zudem einen Kurs "Set up your Moodle Development Environment".

Plugin-Typen

Plugin-Entwicklung ist die gängigste Entwicklung in der Community. Hier ist zuerst zu entscheiden, was für ein Plugin entwickelt werden soll. Davon ist abhängig, wie das Plugin aufgebaut werden

muss. Admin-Tools unterscheiden sich z.B. grundlegend von Blöcken oder Activity-Plugins. Eine Auflistung aller Plugin-Typen findet sich unter Plugin types (auf moodledev.io). Fast jedes Moodle-Plugin benutzt die gleichen, standardisierten Dateitypen und -namen. So wird z.B. in der "version.php" die aktuelle Version des Plugins gespeichert. Durch eine Erhöhung der Version kann zum Beispiel ein Datenbank-Update forciert werden.

Die Moodle Docs bieten auch eine Seite zum Start in die Moodle-Entwicklung:
https://docs.moodle.org/dev/Tutorial

Plugin-Entwicklungsprinzipien

- Moodle-APIs statt eigener Lösungen verwenden
- Capabilities definieren
- Sprachstrings nutzen
- Upgrade-Skripte verwenden

Plugin-Struktur

Beispielhafter Aufbau eines Plugins, siehe auch Kapitel 5 Community-Workshop "Moodle Plugin-Entwicklung":
"Typische Dateien, die man in jedem Plugin findet, und ihr Zweck".

> local_myplugin/
> ├── version.php
> ├── settings.php
> ├── db/
> │ ├── access.php
> │ ├── install.xml
> │ └── events.php
> ├── classes/
> ├── lang/
> └── lib.php

AG Plugins

Die AG Plugins ist ein guter Ansprechpartner und sollte bei Plugin-Entwicklungen konsultiert werden, vor allem hinsichtlich der Möglichkeiten der "Plugin-Werkstatt" (siehe auch Vereins-Strategie, Kaptiel "2.2 Plug-ins").

Coding-Standards

Besonders für Beiträge zum Moodle-Core ist es wichtig, sich an die Coding styles zu halten. Aber auch für Plugin-Entwicklungen ist ein einheitlicher, gut lesbarer und wartbarer Code, der die Zusammenarbeit und Code-Reviews erleichtert, hilfreich. Grundlage bilden die PHP-Standards PSR-1 und PSR-12. Die Richtlinien behandeln Konsistenz, Lesbarkeit, Namenskonventionen, Sicherheit, Automatische Prüfung (Code checker).

Coding style (auf moodledev.io)

Codesniffer

Der Codesniffer analysiert den PHP Code auf Probleme und Kompatibilität mit den Moodle Coding Guidelines.
Manche Fehler kann er auch automatisch beheben.

https://moodledev.io/general/development/tools/phpcs

phpcs mit Composer installieren

```
composer global config minimum-stability dev
```

```
composer global require moodlehq/moodle-cs
```

phpcs für ein Plugin laufen lassen

```
~/.composer/vendor/bin/phpcs /path/to/moodle/mod/foobar/
```

Auto-fix durchführen

```
~/.composer/vendor/bin/phpcbf /path/to/moodle/mod/foobar/
```

ESLint

ESLint ist das Äquivalent für Javascript zu Codesniffer für PHP.

https://moodledev.io/general/development/tools/nodejs#running-grunt

https://docs.moodle.org/dev/Linting#Javascript_(ESLint)

ESLint für in einem Plugin Verzeichnis laufen lassen

```
grunt eslint --show-lint-warnings
```

Auto-fix durchführen

```
grunt eslint --fix
```

API Guide

Moodle bietet viele Programmierschnittstellen (APIs). Diese können bei Entwicklungen z.B. für Standardaufgaben genutzt werden und sind nachhaltiger und kompatibler als eigenen Code zu entwickeln. Häufig verwendete APIs sind die **Access API** (Rechte und Rollen), **Data Manipulation API (DML)** (Datenbankzugriffe), **File API** (Dateiverwaltung), **Form API** (Formulare), **Events API** (Ereignisse) und **External Functions API** (Webservices). Die Nutzung von Events, Hooks und weiteren Core-APIs verbessert die Wartbarkeit.

API Guides (auf moodledev.io)

Moodle Database Schema

Einen Überblick über das Moodle Database Schema inklusive der Beziehungen zwischen den Tabellen bietet

https://www.examulator.com/er/output/index.html.

Security

Durch Implementierung von Schutzmaßnahmen sollten Benutzer:innen, Daten und das Moodle-System vor unbefugtem Zugriff, Datenverlust und Angriffen geschützt werden. Zentrale Elemente hierbei sind Berechtigungen, Prüfung von Nutzereingaben, SQL-Injection verhindern durch Nutzung der Moodle Data Manipulation API (DML), Cross-Site-Scripting (XSS) vermeiden, Ändernde Aktionen absichern, Sensible Informationen schützen, Risiken von Capabilities dokumentieren. Sicherheit sollte fester Bestandteil des Entwicklungsprozesses sein.

Security (auf moodledev.io)

Testing

Durch manuelle und automatische Tests sollen frühzeitig Fehler erkannt und die Stabilität über mehrere Versionen hinweg gesichert werden. Bereits während der Entwicklung sollten Tests geschrieben und regelmäßig ausgeführt werden. Moodle unterstützt u. a. **PHPUnit** für Unit-Tests und **Behat** für Akzeptanztests der Benutzeroberfläche.

Continuous Integration (CI)

Moodle Plugin CI ist das von Moodle empfohlene Werkzeug, um Plugins automatisiert zu prüfen. Es bündelt die wichtigsten Tests und Qualitätsprüfungen und lässt sich in CI-Systeme wie **GitHub Actions** integrieren. Dadurch werden Änderungen bei jedem Commit oder Pull Request automatisch validiert und Fehler frühzeitig erkannt. U.a. **PHPUnit**, **Behat**, **Moodle Code Checker**, **PHPDoc Check**, **Mustache Linting** und **PHP Linting** können automatisiert ausgeführt werden.

Moodle an Hochschulen Plugin Guidelines

Maintaining: https://github.com/moodle-an-hochschulen/moodle-plugin-maintaining/wiki

Readme-Vorlage: https://github.com/moodle-an-hochschulen/moodle-readmetemplate

---

Autor: Klaus Steitz, Technische Universität Darmstadt

Version #8
Erstellt: 2026-07-16 13:53:24 UTC von Klaus Steitz
Zuletzt aktualisiert: 2026-07-21 09:36:18 UTC von Melanie Treitinger