**Technical Product Document: Moodle Student Data Aggregation Plugin**

**1. Introduction**

**Project:** Develop a Moodle plugin: Student Data Aggregator

**Purpose:** This plugin will aggregate student data from various sources within Moodle and present it in a centralized, user-friendly dashboard.

**Target Audience:** Moodle students

**2. System Architecture**

* The plugin will reside within the Moodle platform.
* It will leverage Moodle's API to retrieve data from various Moodle functionalities (Courses, Users, Assignments, etc.).
* The plugin will store and manage user-specific aggregated data locally within Moodle's database.

**3. Moodle Installation Process (Summary)**

**Moodle Download and Unpack**

1. Download the complete Moodle install package (.zip) from the Moodle Windows download page: [link to windows download](https://download.moodle.org/windows/).
2. Extract the downloaded zip file to your desired location on your computer. This will create a folder named "server" and some additional files.

**Start the Webserver**

1. Run the "Start Moodle.exe" file located in the main directory. **Important:** Keep this program running throughout Moodle usage. Use "Stop Moodle.exe" to shut it down.

**Moodle Installation via Web Browser**

1. Open a web browser and navigate to:
   * http://localhost (for local testing)
   * http://192.168.1.1/moodle (for testing on a local network using the server's IP address)

**Moodle Path Configuration**

* If using for local testing, accept the default paths displayed.
* For other setups, enter the appropriate web address based on your needs:
  + Local Testing: http://localhost
  + Local Network Testing: http://your\_server\_IP/moodle (replace with your server's IP)
  + Public Installation: http://your\_public\_IP/moodle (replace with your public IP) or your domain name.

**Database Configuration**

1. Configure the database settings. It's recommended to set a username and password (don't forget them!).
2. **Security Warning:** Avoid using the "root" user without a password.

**Installation Process**

1. Click "Continue" to proceed with the installation.
2. The installer will check server requirements. Ensure they are met for successful installation.
3. Click "Continue" again to proceed.

**Installation Modules and Blocks**

1. You may encounter a series of screens with "Continue" buttons. Click through them.
2. The process will end with the Admin user profile setup.

**Admin User Profile**

1. Fill in the required fields for the admin user account, including a strong password.
2. Click "Update profile" to save your information.

**Site Name and Settings**

1. Enter your desired site name and other settings.
2. Click "Save changes" to finalize the installation.

**4. Development Process**

***4.1 Generate the Plugin Skeleton***

1. Create the Plugin Directory: In your Moodle's blocks directory, create a new directory for your plugin, student\_dash.

2. Create the Essential Files: Inside your plugin directory, create the following files:

(Ref: [https://moodledev.io/docs/apis/commonfiles](https://moodledev.io/docs/apis/core/dml))

- version.php: Contains version information and metadata about the plugin.

- student\_dash.php: The main PHP file that will define the block's behavior.

- lang/en/local\_ student\_dash.php: Contains language strings for English.

***4.2 Define the Plugin's Metadata (version.php)***

This file describes your plugin to Moodle, including version, required Moodle version, and component name. Example content:

php

defined('MOODLE\_INTERNAL') || die();

$plugin->component = 'block\_motivation\_block'; // Full name of the plugin.

$plugin->version = 2022040100; // The current plugin version (Date: YYYYMMDDXX).

$plugin->requires = 2020110900; // Requires this Moodle version.

$plugin->maturity = MATURITY\_STABLE;

$plugin->release = 'v1.0';

***4.3 Implement the Plugin's Core Functionality*** ***(local\_student\_dash.php)***

This file will define the plugin’s class and methods. At a minimum, you need to implement the init(), get\_content() methods:

php

class block\_motivation\_block extends block\_base {

public function init() {

$this->title = get\_string('pluginname', 'block\_motivation\_block');

}

public function get\_content() {

if ($this->content !== null) {

return $this->content;

}

$this->content = new stdClass;

$quotes = [

"Don’t Let Yesterday Take Up Too Much Of Today.",

"You Learn More From Failure Than From Success.",

"We May Encounter Many Defeats But We Must Not Be Defeated.",

];

$random\_key = array\_rand($quotes, 1);

$this->content->text = $quotes[$random\_key];

$this->content->footer = '';

return $this->content;

}

}

***4.4 Add Language Strings (lang/en/local\_student\_dash.php)***

Define the plugin name and any other strings you'll use:

php

$string['pluginname'] = 'Motivation Block';

***4.5 Install and Test Your Plugin***

- Place your plugin code in the local directory of your Moodle installation.

- Visit the Moodle site as an admin to trigger the installation automatically.

- Add the plugin to a course or dashboard and verify it displays the motivational quotes correctly.

**5. Data Required for Implementation**

For a Moodle plugin focusing on enhancing the student dashboard with detailed user data, course information, grades, assignments, and additional resources, let's delve into specific Moodle APIs you would use and how to apply them effectively. These APIs and functions allow for a structured and secure way to access and manipulate Moodle's data.

(Ref: <https://moodledev.io/docs/apis/core/dml>

<https://moodledev.io/docs/apis>)

**5.1.** **User Data**`

-**Global $USER Object**: Directly access logged-in user information.

php

global $USER;

echo $USER->email; // Example: Accessing user's email address

- \*Profile Fields\*: For accessing custom profile fields, use:

php

$customfields = profile\_user\_record($USER->id);

echo $customfields->department; // Assuming 'department' is a custom field

**5.2. Grades & Courses**

- **Grades API** **(grade\_get\_grades())**: Retrieve a user's grades for a specific course.

php

$grades = grade\_get\_grades($courseid, 'mod/assign', 'submission', null, $userid);

// Process $grades object for display

- \*Courses\*: Use Moodle's course-related functions to fetch course data.

- \*Fetching course by ID\*:

php

$course = get\_course($courseid);

echo $course->fullname; // Display the full name of the course

**5.3.** **Assignments & Quizzes**

- **Assignments**: Accessing assignment details and submissions.

- *List assignments for a course*:

php

$assignments = $DB->get\_records('assign', ['course' => $courseid]);

foreach ($assignments as $assignment) {

// Process each assignment

}

- **Quizzes**: Retrieving quiz attempts and information.

- *Quiz Attempts*:

php

$attempts = $DB->get\_records('quiz\_attempts', ['userid' => $USER->id, 'quiz' => $quizid]);

foreach ($attempts as $attempt) {

// Process each attempt

}

**5.4.** **Database Access ($DB)**

Moodle's database abstraction layer $DB is crucial for custom queries not covered by specific APIs.

- **Example of a Custom SQL Query**:

php

$sql = "SELECT \* FROM {course} WHERE category = ?";

$courses = $DB->get\_records\_sql($sql, [$categoryid]);

foreach ($courses as $course) {

// Process each course

}

**5.5.** **File API**

For accessing files, like course syllabus, Moodle's File API is used.

- **Retrieve a File**:

php

$context = context\_course::instance($courseid);

$fs = get\_file\_storage();

$files = $fs->get\_area\_files($context->id, 'mod\_resource', 'content', 0, 'sortorder DESC, id ASC', false);

foreach ($files as $file) {

// Process each file (e.g., display a link to download the file)

}

**5.6.** **Zoom Recordings (If applicable)**

For plugins like Zoom, you'd need to consult the specific plugin's documentation for API functions. If storing data in Moodle's database, you'd access it similarly to other custom queries or through specific functions provided by the plugin.

php

// Hypothetical example - actual implementation will vary based on the plugin

$recordings = $DB->get\_records('zoom\_recordings', ['courseid' => $courseid]);

**Using These APIs**

When using these APIs:

* Check for user permissions before displaying sensitive information.
* Utilize Moodle's capabilities system to ensure only authorized users perform certain actions.
* Performance and security - use parameterized queries $DB->get\_records\_sql() to avoid SQL injection attacks.

**6. Data Storage**

* The plugin will store minimal user data locally (name for display purposes).
* Links to access existing Moodle data will be used whenever possible.
* User-added tasks will be stored in a dedicated table within the Moodle database with appropriate fields for task descriptions and course associations.

**7. API Reference**

* Refer to the official Moodle API documentation for detailed information on the specific API calls used to retrieve the required data.