

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЧНОЇ
СИНХРОНІЗАЦІЇ КОРИСТУВАЧІВ У LMS MOODLE**

**DEVELOPMENT OF A SOFTWARE MODULE FOR AUTOMATIC USER
SYNCHRONIZATION IN THE MOODLE LMS**

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Васильчук Андрій Андрійович

Керівник:
Гульчук Юрій Миколайович

Кваліфікаційну роботу
допущено до захисту
«__» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

«__» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Васильчуку Андрію Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка програмного модуля автоматичної синхронізації користувачів у LMS Moodle»

Керівник роботи: асистент Гульчук Ю. М.

затверджені наказом закладу вищої освіти від «11» березня 2026 р. № 116/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра «29» травня 2026 р.

3. Вихідні дані до роботи: PHP 8.0+, Moodle 4.0.4+, REST API, curl, Drupal JSON API, Cron Scheduled Tasks, Moodle Web Services, Розробка програмного модуля.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): аналіз сучасного стану проблем інтеграції LMS-систем, постановка завдання, визначення вимог до модуля синхронізації, обґрунтування вибору технологій та архітектурних рішень, практична реалізація плагіну local_mlsync, опис структури та логіки класів, тестування та налагодження.

5. Перелік графічного матеріалу: 10 рисунків, 6 таблиця, 3 додатки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	Гульчук Ю. М.		
Специфікація вимог до розробленої системи	Гульчук Ю. М.		
Розробка об'єкта проектування	Гульчук Ю. М.		
Нормоконтроль	Суринович О. М.		
Гарант ОП	Ліщина Н. М.		
Показник запозичень тексту	_____ %		
Академічна доброчесність	Гульчук Ю. М.		

7. Дата видачі завдання «11» березня 2026 р.

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 11.03.2026 р.	
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 14.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 19.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об'єкта проектування	до 14.04.2026 р.	
5	Практична реалізація плагіну local_mlsync	до 20.04.2026 р.	
6	Тестування та налагодження плагіну	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі	до 29.05.2026 р.	

Здобувач вищої освіти _____

Андрій ВАСИЛЬЧУК

Керівник кваліфікаційної роботи _____

Юрій ГУЛЬЧУК

АНОТАЦІЯ

Васильчук А. А. Розробка програмного модуля автоматичної синхронізації користувачів у LMS Moodle. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз актуальності проблеми ручного управління обліковими записами у системах електронного навчання та сформульовано завдання. В другому розділі визначено вимоги до розроблюваного програмного модуля і вибір засобів, методів та алгоритмів для вирішення задачі автоматизації синхронізації. У третьому розділі представлено практичну реалізацію плагіну `local_mlsync` для Moodle 4.0.4+, описано структуру та логіку класів, реалізовано механізми підключення до ML-платформи (`ml.lntu.edu.ua`) та Moodle Web Services, розглянуто питання безпеки, масштабованості та тестування. У висновках узагальнено отримані результати, підсумовано досягнення поставлених цілей та окреслено перспективи подальшого розвитку системи.

Ключові слова: Moodle, LMS, плагін, Web Services, PHP, REST API, Drupal, cron, когорта, синхронізація, автоматизація, local plugin, scheduled task.

ABSTRACT

Vasylchuk A. A. Development of a Software Module for Automatic User Synchronization in the Moodle LMS. Manuscript. Qualification work for bachelor's degree in Software Engineering, specialty “Software Engineering”. Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of references and appendices.

The first chapter analyzes the relevance of the problem of manual management of user accounts in e-learning systems and formulates the tasks. The second chapter defines the requirements for the software module to be developed and the choice of tools, methods and algorithms for solving the synchronization automation task. The third chapter presents the practical implementation of the local_mlsync plugin for Moodle 4.0.4+, describes the structure and logic of classes, implements the mechanisms for connecting to the ML platform and Moodle Web Services, and considers security, scalability and testing issues. The conclusions summarize the results obtained, summarize the achievement of the set goals, and outline the prospects for further development of the system.

Keywords: Moodle, LMS, plugin, Web Services, PHP, REST API, Drupal, cron, cohort, synchronization, automation, local plugin, scheduled task.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМ УПРАВЛІННЯ КОРИСТУВАЧАМИ У LMS Moodle ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ	9
1.1 Аналіз сучасного стану проблеми	9
1.2 Постановка завдання на кваліфікаційну роботу бакалавра	11
Висновки до розділу 1	12
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ.....	13
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення	13
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання	17
Висновки до розділу 2	18
РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЧНОЇ СИНХРОНІЗАЦІЇ КОРИСТУВАЧІВ У LMS MOODLE.....	20
3.1 Практична реалізація об'єкта проектування.....	20
3.2 Розробка компонентів програмного модуля	23
3.3 Захист інформаційно-комп'ютерної системи	31
3.4 Тестування та налагодження інформаційно-комп'ютерної системи	32
3.5 Модульне розширення та масштабування системи.....	36
Висновки до розділу 3	38
ВИСНОВКИ.....	39
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	41
ДОДАТКИ.....	43

ВСТУП

Актуальність теми. У сучасному освітньому просторі системи управління навчанням (Learning Management Systems, LMS) відіграють ключову роль в організації навчального процесу у вищих навчальних закладах. Moodle є однією з найпоширеніших відкритих LMS у світі, яка активно використовується в українських університетах, зокрема в Луцькому національному технічному університеті. Паралельно з Moodle університет експлуатує окрему ML-платформу (ml.lntu.edu.ua) на базі Drupal, де зберігається актуальна інформація про студентів та їхню приналежність до академічних груп.

Ручне управління обліковими записами студентів у Moodle є одним із найбільш трудомістких адміністративних процесів: щороку необхідно реєструвати сотні нових студентів, розподіляти їх по академічних групах (когортах) та переводити між ними при зміні групи. Помилки при ручному введенні даних призводять до неправильного розподілу студентів, проблем з доступом до курсів та додаткового навантаження на адміністраторів. Відсутність автоматизованого механізму керування користувачами у системі управління навчання Moodle формує технічні затримки із появою нових облікових записів у системі, що негативно впливає на ефективність навчального процесу.

Мета роботи полягає в проектуванні та розробці програмного модуля у вигляді плагіну для Moodle, який автоматично синхронізує облікові записи студентів між ML-платформою та Moodle, забезпечуючи правильний розподіл по когортах без участі адміністратора.

Об'єктом роботи є процес автоматизованої інтеграції та синхронізації даних між двома різними платформами:

- Moodle (mdl.lntu.edu.ua);
- Drupal (ml.lntu.edu.ua).

Предметом роботи є архітектурні та технічні рішення розробки плагіну local_mlsync для LMS Moodle 4.0.4+ з використанням PHP, Moodle Web Services API та REST API ML-платформи.

Завдання на кваліфікаційну роботу:

- проаналізувати сучасний стан проблем управління користувачами у LMS Moodle;
- дослідити існуючі підходи та інструменти інтеграції Moodle з зовнішніми системами;
- сформулювати вимоги до розроблюваного програмного модуля;
- обґрунтувати вибір технологій та архітектурних рішень;
- практично реалізувати плагін `local_mlsync`;
- описати структуру класів та логіку роботи модуля;
- забезпечити захист даних та стійкість до помилок;
- протестувати та налагодити програмний модуль.

РОЗДІЛ 1

АНАЛІЗ ПРОБЛЕМ УПРАВЛІННЯ КОРИСТУВАЧАМИ У LMS Moodle ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Системи управління навчанням набули масового поширення у вищій освіті упродовж останніх двох десятиліть. За даними аналітичних досліджень, понад 30 мільйонів користувачів у більш ніж 230 країнах використовують Moodle – найпоширенішу відкриту LMS у світі [15]. Ключовою перевагою Moodle є її розширюваність через систему плагінів, відкритий вихідний код та сильна спільнота розробників. На рисунку 1.1 зображено статистику використання LMS Moodle у всьому світі.

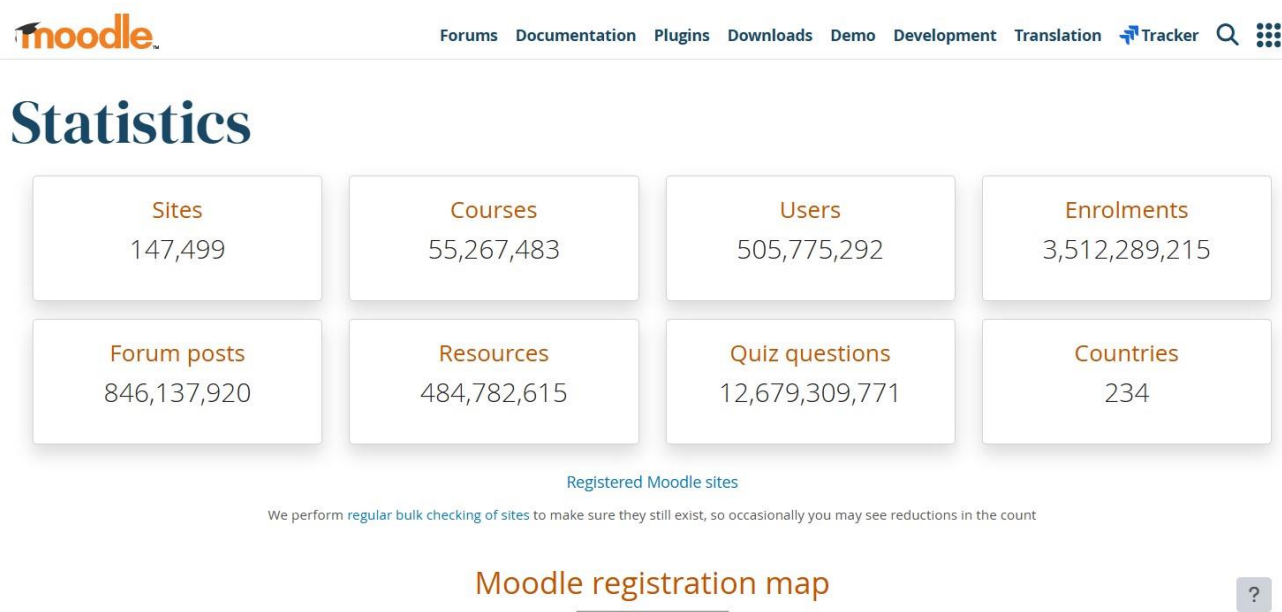


Рисунок 1.1 – Статистика використання LMS Moodle

Водночас сучасні університети, як правило, не обмежуються однією інформаційною системою. Типова інфраструктура включає систему управління студентами (Student Information System, SIS), портал для дистанційного навчання (LMS), електронну бібліотеку, систему електронного документообігу тощо. Кожна з цих систем накопичує власну базу даних користувачів, що породжує

проблему синхронізації: одні й ті самі студенти мають записи в кількох різних системах, які не завжди відповідають одна одній.

Інтеграція Moodle із зовнішніми системами може здійснюватися кількома способами. Найпоширенішим є використання протоколів єдиного входу (Single Sign-On, SSO): LDAP, SAML 2.0, CAS [5]. Ці протоколи вирішують лише задачу автентифікації, але не синхронізацію даних профілів та розподіл по групах [10]. Провізіонування користувачів через LDAP або Active Directory є стандартним рішенням у корпоративних середовищах, проте вимагає відповідної інфраструктури та не підходить для платформ з власною користувацькою базою, як-от ML-платформа з ядром Drupal.

Moodle Web Services (WS) – це вбудований механізм, що дозволяє зовнішнім системам взаємодіяти з Moodle через HTTP-запити [16]. Web Services підтримують формати REST, SOAP та XML-RPC і надають доступ до широкого набору функцій: управління користувачами, курсами, когортами тощо [13]. Цей механізм є найбільш гнучким інструментом для реалізації програмної інтеграції без втручання у ядро Moodle.

Серед наявних готових рішень варто відзначити плагін «Automatic enrolment in groups» (рис. 1.2) та різні реалізації провізіонування через CLI-скрипти. Проте більшість з них або розраховані на LDAP/AD, або є несумісними з Moodle 4.x, або не підтримують автоматичне управління когортами. Плагін типу local з scheduled task є найбільш гнучким і підтримуваним підходом для реалізації власної логіки синхронізації [9].

Plugins / Administration tools / Automatic enrolment in groups / Description



Automatic enrolment in groups

Administration tools :: tool_groupautoenrol

Maintained by Pascal Maury, Gemma Lesterhuis, Luuk Verhoeven

Plugin to auto enrol users in Moodle courses groups

Latest release: 8 мес. 321 sites 76 downloads 32 fans

Description

Versions

Stats

Translations

Рисунок 1.2 – Плагін для автоматичної реєстрації користувачів у групах курсів Moodle

Окремою проблемою є синхронізація з платформами на базі Drupal. Drupal – це відкрита CMS/CMF, яка, як і Moodle, широко використовується в освітніх установах. Drupal пропонує власний REST API (модуль JSON:API або Views REST), що дозволяє отримувати дані про користувачів у форматі JSON. Однак механізм авторизації в Drupal відрізняється від стандартних OAuth-потоків: для сесійної авторизації використовується CSRF-токен та cookie-сесія, що потребує спеціальної обробки на стороні клієнта [3].

Аналіз показав відсутність готових рішень для автоматичної двосторонньої синхронізації між Drupal-платформою та Moodle з підтримкою управління когортами в Moodle 4.0.4+. Розробка власного плагіну є обґрунтованим та економічно доцільним рішенням.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою дослідження є розробка програмного модуля `local_mlsync` у вигляді плагіну для Moodle 4.0.4+, який автоматично синхронізує облікові записи

студентів з ML-платформи (Drupal) у Moodle та забезпечує їхній правильний розподіл по когортах (академічних групах).

Для досягнення поставленої мети було сформовано такі завдання:

- проаналізувати сучасний стан проблем управління користувачами у LMS Moodle та існуючі підходи до інтеграції;
- дослідити архітектуру плагінів Moodle 4.0.4+ та механізми `scheduled tasks`;
- сформулювати функціональні та нефункціональні вимоги до плагіну;
- обґрунтувати вибір архітектурних рішень та технологій;
- розробити структуру класів та реалізувати плагін `local_mlsync`;
- реалізувати безпечне підключення до ML-платформи та Moodle Web Services;
- забезпечити захист даних та стійкість до помилок;
- провести тестування та налагодження плагіну.

Висновки до розділу 1

У першому розділі здійснено комплексний аналіз проблематики управління обліковими записами у системах електронного навчання. Встановлено, що автоматизація процесу управління та синхронізації обліковими записами в LMS Moodle є актуальною задачею для університетів, які використовують кілька інформаційних платформ одночасно. Проаналізовано наявні підходи до інтеграції Moodle з зовнішніми системами (LDAP, Active Directory, Web Services), виявлено їхні обмеження стосовно завдання синхронізації з Drupal-платформою та управління когортами.

Визначено, що розробка спеціалізованого плагіну з використанням Moodle Web Services та `scheduled tasks` є оптимальним підходом для вирішення поставленого завдання [4]. Визначено мету та завдання кваліфікаційної роботи, що стали основою для подальшого проектування та реалізації плагіну `local_mlsync`.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Розробка програмного модуля автоматичної синхронізації вимагає ретельного аналізу предметної області та чіткого формулювання вимог. Модуль функціонуватиме на перетині двох систем – ML-платформи (Drupal) та LMS Moodle, тому необхідно враховувати специфіку обох платформ: їхні API, механізми авторизації, формати даних та обмеження.

Аналіз предметної області виявив ключові функціональні сценарії. Основним є автоматичне отримання переліку студентів з ML-платформи з їхніми поточними статусами навчання та створення облікових записів у Moodle (якщо ще не існують) та синхронізація приналежності до когорти (академічної групи). Особливо важливим є обробка трьох ситуацій: студент вперше з'являється в середовищі Moodle, студент вже існує в Moodle, студент переводиться з однієї групи в іншу.

На рисунку 2.1 наведено блок-схему алгоритму роботи програмного модуля синхронізації. Алгоритм передбачає три сценарії обробки: створення нового облікового запису студента у Moodle (ситуація 1), підтвердження коректності поточної когорти для вже існуючого користувача (ситуація 2) та переведення студента з однієї когорти до іншої при зміні академічної групи (ситуація 3). Цикл повторюється для кожного запису, отриманого з ML-платформи, після чого задача завершує роботу.

Окремої уваги заслуговує аналіз нефункціональних вимог до модуля. Оскільки синхронізація виконується в автоматичному режимі за розкладом (cron), система повинна забезпечувати стійкість до тимчасової недоступності зовнішнього API, коректно обробляти часткові відповіді та не допускати дублювання облікових записів у разі повторного запуску.

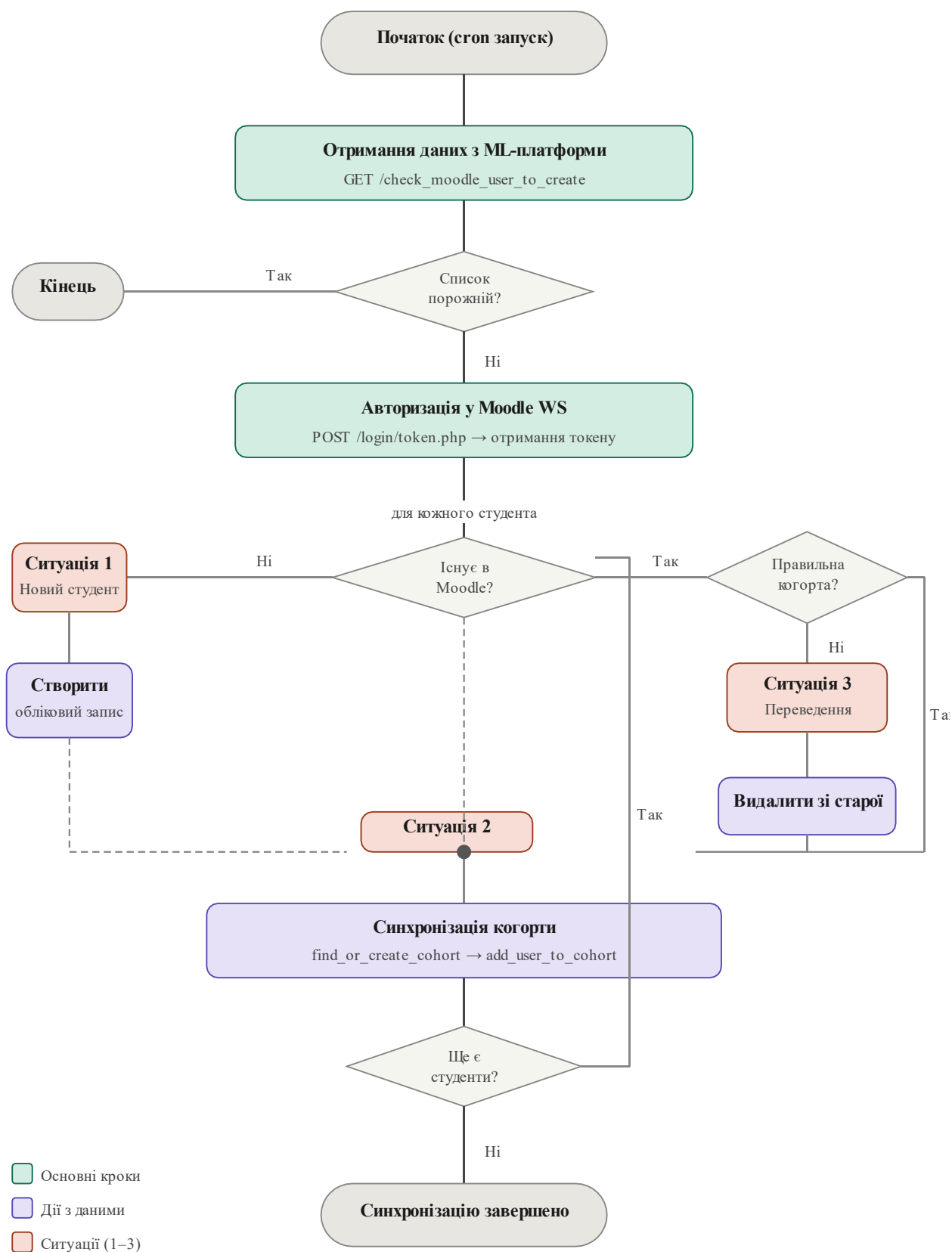


Рисунок 2.1 – Блок-схема алгоритму автоматичної синхронізації користувачів між ML-платформою та LMS Moodle

Когорта (cohort) у Moodle – це іменована група користувачів на рівні системи, що є аналогом академічної групи. Когорти можна прив'язувати до курсів, що забезпечує автоматичний запис всіх членів когорти на відповідні дисципліни. Логіка «один студент – одна когорта» є бізнес-вимогою, оскільки студент може бути лише в одній академічній групі одночасно [11].

Для моделювання взаємодії між компонентами системи розроблено схему потоку даних. Cron-задача виступає оркестратором: отримує дані з ML-платформи, передає їх до клієнта Moodle WS, який створює користувачів та синхронізує когорти. Усі компоненти є незалежними та тестованими окремо.

Функціональні вимоги до плагіну сформовано на основі аналізу бізнес-процесів:

- автентифікація на ML-платформі (Drupal) через REST API з CSRF-токеном та cookie-сесією;
- отримання переліку студентів для синхронізації з Drupal Views endpoint;
- перевірка існування кожного студента у Moodle за email;
- масове створення нових студентів через Moodle WS (core_user_create_users);
- визначення поточної когорти студента та порівняння з цільовою;
- автоматичне переведення студента між когортами при необхідності;
- автоматичне створення когорти якщо вона не існує;
- захист від дублювання даних (перевірка за email та username);
- примусова зміна пароля при першому вході нового студента;
- ведення журналу виконання для моніторингу та діагностики.

Нефункціональні вимоги:

- ідемпотентність – повторний запуск не спричиняє дублювання даних;
- стійкість до помилок – помилка синхронізації одного студента не зупиняє обробку інших;
- продуктивність – batch-запити замість N окремих запитів для N студентів;

- відповідність стандартам розробки плагінів Moodle 4.0.4+;
- GDPR-сумісність – відсутність збереження персональних даних у власних таблицях;
- конфігурованість – параметри підключення легко змінюються.

Архітектурне проектування системи здійснено з урахуванням принципів розмежування повноважень (Separation of Concerns) та єдиної відповідальності (Single Responsibility Principle). Система складається з трьох незалежних компонентів: клієнта ML-платформи, клієнта Moodle WS та задачі-оркестратора. Такий поділ забезпечує легкість тестування та можливість незалежної заміни кожного компоненту.

Схема взаємодії компонентів системи відображена у таблиці 2.1.

Таблиця 2.1 – Взаємодія компонентів системи local_mlsync

Крок	Компонент	Дія
1	Moodle cron runner	Запускає задачу sync_users за розкладом
2	task\sync_users	Створює ml_client, викликає fetch_users()
3	ml_client	Авторизація на Drupal (CSRF + cookie), GET /check_moodle_user_to_create
4	task\sync_users	Передає масив студентів до moodle_client::sync_users()
5	moodle_client	POST /login/token.php – отримує WS-токен
6	moodle_client	core_user_get_users_by_field – перевірка існуючих за email (batch)
7	moodle_client	core_user_create_users – масове створення нових студентів
8	moodle_client	sync_user_cohort() для кожного студента
9	moodle_client	core_cohort_search_cohorts – пошук когорти
10	moodle_client	core_cohort_add/delete_cohort_members – зміна когорти

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Головною метою при виборі технологій для реалізації плагіну була максимальна сумісність зі стандартами розробки Moodle та мінімізація зовнішніх залежностей. Критеріями відбору слугували: підтримуваність у Moodle 4.0.4+, безпека, продуктивність та зручність супроводу.

Тип плагіну «local» обрано як найбільш гнучкий тип розширення Moodle, що не прив'язаний до конкретного курсу чи модуля. Local-плагіни розміщуються у директорії {moodle_root}/local/ та можуть містити будь-яку власну логіку, у тому числі scheduled tasks, сторінки адмінки, web services [7]. Це є стандартним підходом для адміністративних утиліт та інтеграційних рішень.

Moodle Scheduled Tasks – це вбудований у Moodle планувальник задач, який є аналогом системного cron [1]. Задачі реєструються у db/tasks.php та виконуються cron runner'ом Moodle. Перевагою цього підходу є централізоване управління розкладом через адмінпанель, логування виконання та автоматичне відновлення після збоїв [12].

Moodle Web Services REST API використовується для програмної взаємодії з Moodle. REST-формат обрано через його простоту та широку підтримку. Функції Web Services, що використовуються у плагіні, наведені у таблиці 2.2.

Таблиця 2.2 – Функції Moodle Web Services, що використовуються у плагіні

Функція WS	Призначення
core_user_get_users_by_field	Пошук існуючих користувачів за email (batch)
core_user_create_users	Масове створення нових користувачів
core_cohort_search_cohorts	Пошук когорти за назвою або idnumber
core_cohort_create_cohorts	Створення нової когорти
core_cohort_get_cohort_members	Отримання членів когорти (batch)
core_cohort_add_cohort_members	Додавання користувача до когорти
core_cohort_delete_cohort_members	Видалення користувача з когорти

Вбудований curl Moodle обрано замість зовнішньої бібліотеки GuzzleHttp. Клас curl у Moodle є обгорткою над стандартним curl PHP, яка автоматично враховує налаштування проксі-сервера, SSL-сертифікатів та інших параметрів, заданих адміністратором Moodle. Відмова від Guzzle усуває необхідність використовувати composer у директорії плагіну та повністю відповідає вимогам Moodle Coding Style [8].

Drupal REST API (Views REST export) обрано як джерело даних з ML-платформи. Модуль Views Drupal дозволяє створювати SQL-запити через GUI та публікувати результати у форматі JSON [2]. Авторизація відбувається через двоетапний процес: отримання CSRF-токену та сесійна авторизація з cookie.

PSR-4 автозавантаження через namespace – стандартний підхід Moodle 4.0.4+ для організації PHP-класів [18]. Namespace local_mlsync відповідає назві плагіну, що забезпечує автоматичне завантаження класів без підключення require/include [19].

Ідемпотентний алгоритм синхронізації розроблено спеціально для безпечного повторного запуску. Алгоритм перевіряє поточний стан перед кожною операцією зміни: існування користувача, поточну когорту, наявність когорти в системі. Операції виконуються лише тоді, коли поточний стан відрізняється від цільового. Це забезпечує стабільну роботу при часто повторюваних запусках cron.

Висновки до розділу 2

У другому розділі здійснено аналіз предметної області, визначено функціональні та нефункціональні вимоги до плагіну local_mlsync. На основі аналізу сформульовано архітектурну модель системи, що відповідає стандартам Moodle 4.0.4+ та принципам чистої архітектури. Розроблено схему взаємодії компонентів та блок-схему алгоритму синхронізації, що відображає три ключові сценарії обробки облікових записів студентів.

Обґрунтовано вибір технологій та підходів: тип плагіну local, Moodle Scheduled Tasks, Moodle Web Services REST API, нативний PHP curl без зовнішніх залежностей, PSR-4 автозавантаження класів, ідемпотентний алгоритм синхронізації. Визначено повний перелік функцій Moodle WS, необхідних для управління користувачами та когортами, а також описано кроки взаємодії з Drupal REST API, включаючи двоетапну авторизацію через CSRF-токен та сесійні cookies.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЧНОЇ СИНХРОНІЗАЦІЇ КОРИСТУВАЧІВ У LMS MOODLE

3.1 Практична реалізація об'єкта проєктування

Для реалізації задачі автоматичної синхронізації користувачів обрано тип плагіну `local` – найбільш гнучке розширення Moodle, що не прив'язане до жодного курсу чи контексту і дозволяє розміщувати довільну логіку, зокрема `scheduled tasks`. Плагін отримав назву `local_mlsync` відповідно до конвенції `type_name`, що використовується у Moodle для автоматичного завантаження класів.

Розробка плагіну виконувалась у наступній послідовності: спочатку реєстраційні файли, потім оголошення завдання, далі класи бізнес-логіки і насамкінець GDPR-декларація. Така послідовність обумовлена тим, що Moodle при встановленні плагіну зчитує `version.php` і `db/tasks.php` раніше за будь-який PHP-код класів. Структуру файлів плагіну наведено у таблиці 3.1.

Таблиця 3.1 – Файлова структура плагіну `local_mlsync`

Файл / директорія	Призначення
<code>version.php</code>	Метадані плагіну: версія, мінімальна версія Moodle, компонент
<code>db/tasks.php</code>	Реєстрація <code>scheduled task</code> у планувальнику Moodle
<code>db/install.php</code>	Хук встановлення (обов'язковий)
<code>lang/en/, lang/uk/</code>	Рядки локалізації (обов'язкова умова встановлення)
<code>classes/ml_client.php</code>	Клас-клієнт для ML-платформи (Drupal REST API)
<code>classes/moodle_client.php</code>	Клас-клієнт для Moodle Web Services API
<code>classes/task/sync_users.php</code>	<code>Scheduled task</code> – головна точка входу <code>cron</code>
<code>classes/privacy/provider.php</code>	GDPR-декларація (обов'язковий інтерфейс Moodle)

Файл `version.php` є першим і обов'язковим файлом будь-якого Moodle-плагіну. Він повідомляє системі ідентифікатор компоненту, версію та мінімальні вимоги. У лістингу 3.1 наведено вміст цього файлу.

Лістинг 3.1 – Файл version.php

```
<?php
defined('MOODLE_INTERNAL') || die();

$plugin->component = 'local_mlsync';
$plugin->version    = 2024010100;
$plugin->requires   = 2022041900; // Moodle 4.0
$plugin->maturity    = MATURITY_STABLE;
$plugin->release    = '1.0.0';
```

кінець лістингу 3.1

На рисунку 3.1 зображено вигляд сторінки встановлення плагіну local_mlsync в LMS Moodle версії 4.0.4+.

І перевірка компонентів

На цій сторінці відображаються плагіни, які можуть вимагати вашої уваги під час оновлення, наприклад, нові плагіни, які потрібно встановити, плагіни, які потрібно оновити, відсутні плагіни тощо. Додаткові плагіни відображаються, якщо є доступне оновлення для них. Рекомендується перевірити, чи доступні новіші версії плагінів, і оновити їх вихідний код, перш ніж продовжити оновлення Moodle.

Перевірити наявність оновлень

Додатки, які потребують уваги

Скасувати нові установки (1) Додатки, які потребують уваги 1 Всі плагіни 428

Назва модуля / Каталог	Поточна версія	Нова версія	Вимагає	Джерело / Статус
Локальні модулі				
Синхронізація користувачів ML → Moodle /local/mlsync		2024010100	• Moodle 2022041900	Розширення Буде встановлено Скасувати цю установку

Оновити

Оновити

Рисунок 3.1 – Встановлення плагіну local_mlsync

Component – унікальна назва у форматі type_name; Moodle використовує її для PSR-4 автозавантаження класів (namespace збігається з назвою компоненту). version – числовий ідентифікатор у форматі YYYYMMDDXX; при зміні версії Moodle автоматично запускає upgrade-хуки. requires – мінімальна версія Moodle (2022041900 = Moodle 4.0); плагін не встановиться на старіших версіях [6].

Файл `db/tasks.php` реєструє заплановану задачу у системному планувальнику Moodle. Розклад задається у форматі `crontab`. У лістингу 3.2 наведено реєстрацію завдання з виконанням кожні 4 години.

Лістинг 3.2 – Файл `db/tasks.php` – реєстрація `cron`-завдання

```
<?php
defined('MOODLE_INTERNAL') || die();

$tasks = [
    [
        'classname' => '\local_mlsync\task\sync_users',
        'blocking'   => 0,
        'minute'     => '0',
        'hour'       => '*/4',
        'day'        => '*',
        'month'       => '*',
        'dayofweek'  => '*',
        'disabled'   => 0,
    ],
];
```

кінець лістингу 3.2

Параметр `classname` містить повністю кваліфіковане ім'я класу (FQCN); Moodle знаходить файл автоматично через PSR-4. Параметр `blocking` зі значенням 0 дозволяє паралельне виконання з іншими `cron`-завданнями. Адміністратор може змінити розклад виконання через веб-оболонку Moodle: Керування сайтом → Сервер → Заплановані завдання. Вигляд планувальника завдань в Moodle зображено на рисунку 3.2.

Варто зазначити, що механізм `scheduled tasks` у Moodle є більш надійною альтернативою традиційному системному `cron`, оскільки забезпечує централізоване керування всіма запланованими завданнями в межах однієї платформи. Кожне виконання завдання фіксується у системному журналі Moodle, що дозволяє адміністратору відстежувати час останнього запуску, тривалість виконання та наявність помилок безпосередньо через веб-інтерфейс. У разі виникнення критичної помилки під час синхронізації Moodle автоматично позначає завдання як таке, що завершилось із збоєм.

Електронний освітній портал ЛНТУ

Особистий кабінет / Керування сайтом / Сервер / Задачі / Заплановані завдання / Заплановані завдання / Зміна розкладу задачі: Синхронізація користувачів з ML-платформи

Навігація

- Особистий кабінет
 - Домашня сторінка
 - Сторінки сайту
 - Мої курси
 - Курси

Керування

- Керування сайтом
 - Повідомлення
 - Реєстрація
 - Moodle сервіси
 - Перегляд параметрів збору відгуків
 - Розширені функції
 - Користувачі
 - Курси
 - Журнал оцінок
 - Аналітика
 - Компетенції
 - Відзнаки
 - НЗР
 - Ліцензія
 - Локалізація

Зміна розкладу задачі: Синхронізація користувачів з ML-платформи

3 компонента: Синхронізація користувачів ML → Moodle

local_mlsync

Останній запуск: Thursday 19 March 2026 20:05 PM

Наступний запуск: За першої ж нагоди

Хвилина: * За умовчанням: 0

Година: */4 За умовчанням: */4

День: * За умовчанням: *

Місяць: * За умовчанням: *

День тижня: * За умовчанням: *

☐ Відключено

☐ Встановити типовий розклад

Зберегти зміни Скасувати

Рисунок 3.2 – Вигляд планувальника завдань в Moodle

3.2 Розробка компонентів програмного модуля

Програмний модуль складається з трьох ключових PHP-класів, кожен з яких відповідає за чітко визначену зону відповідальності: клас `ml_client` забезпечує взаємодію з ML-платформою, клас `moodle_client` – з Moodle Web Services API, клас `sync_users` є cron-оркестратором що координує роботу двох перших. Такий поділ відповідає принципу розділення відповідальностей (Separation of Concerns) та забезпечує можливість незалежного тестування і заміни кожного компоненту.

Клас `ml_client`, повний код якого наведено в додатку А. Клас (файл `classes/ml_client.php`, namespace `local_mlsync`) відповідає за всю комунікацію з ML-платформою на базі Drupal. Для HTTP-запитів використовуються нативні функції PHP `curl_*` замість зовнішньої бібліотеки `GuzzleHttp` – це усуває необхідність використання `composer` у директорії плагіну і гарантує сумісність з PHP 7.4+ [17]. Конструктор класу приймає три параметри (URL, логін, пароль) та зберігає їх у приватних полях без виконання мережових запитів (принцип

відкладеної ініціалізації). У лістингу 3.3 наведено оголошення полів та конструктор.

Лістинг 3.3 – Клас ml_client – поля та конструктор

```
<?php
namespace local_mlsync;
defined('MOODLE_INTERNAL') || die();

class ml_client {
    private string $base_url;
    private string $username;
    private string $password;
    private ?string $csrf_token = null;
    private string $cookie_string = '';

    public function __construct(
        string $base_url, string $username, string $password) {
        $this->base_url = rtrim($base_url, '/');
        $this->username = $username;
        $this->password = $password;
    }
}
```

кінець лістингу 3.3

Публічний метод fetch_users() є головною точкою входу: він оркеструє двоетапну авторизацію та отримання даних, перехоплює будь-який виняток і повертає порожній масив замість падіння cron. У лістингу 3.4 наведено реалізацію методу.

Лістинг 3.4 – Метод fetch_users() класу ml_client

```
public function fetch_users(): array {
    try {
```

```

        $this->authenticate();
        return $this->get_users_to_create();
    } catch (\Exception $e) {
        $this->log('ERROR: ' . $e->getMessage());
        return [];
    }
}

```

кінець лістингу 3.4

Приватний метод `authenticate()` реалізує двоетапну авторизацію у Drupal REST API. На першому кроці виконується GET до `/session/token` для отримання CSRF-токену – Drupal вимагає його для всіх POST-запитів що змінюють стан. На другому кроці виконується POST до `/user/login?_format=json` з `credentials`; у відповіді повертається оновлений CSRF-токен та `session cookie`. Метод `get_users_to_create()` виконує GET до Drupal Views endpoint `/check_moodle_user_to_create` із заголовком `X-CSRF-Token` та повертає PHP-масив студентів. Допоміжні методи класу, що використовуються у файлі `ml_client.php`, описано в таблиці 3.2.

Таблиця 3.2 – Допоміжні методи класу `ml_client`

Метод	Призначення
<code>make_curl()</code>	Ініціалізує curl handle з таймаутами 30/10 с та <code>CURLOPT_SSL_VERIFYPEER=false</code> ; повертає <code>resource/CurlHandle</code> без <code>type hint</code> (сумісність PHP 7.4+)
<code>run_curl()</code>	Виконує handle, розбиває відповідь на заголовки та тіло, перевіряє HTTP-код
<code>extract_cookies()</code>	Парсить Set-Cookie заголовки та формує рядок для <code>Cookie: header</code>
<code>log()</code>	Виводить повідомлення через <code>mtrace()</code> під час CLI/cron та <code>debugging()</code> у dev-режимі

Клас `moodle_client`, повний код якого наведено в додатку Б. Клас (файл `classes/moodle_client.php`, namespace `local_mlsync`) інкапсулює всі виклики до Moodle Web Services REST API. Архітектурно клас поділено на три групи методів: авторизація, робота з користувачами та робота з когортами.

Метод `authenticate()` отримує WS-токен через POST до `/login/token.php` з параметрами `username`, `password`, `service`. Параметр `service` має збігатися з назвою `External Service` в адмінпанелі Moodle. Токен зберігається у приватному полі та автоматично підставляється у всі наступні запити. Базовий метод `call()` є центральним у класі – він виконує POST до `/webservice/rest/server.php` та обробляє три особливі випадки відповіді: `null` (норма для деяких WS-функцій), поле `exception` (кидається `RuntimeException`) та помилку парсингу JSON. У лістингу 3.5 наведено реалізацію методу `call()`.

Лістинг 3.5 – Метод `call()` класу `moodle_client` – базовий виклик WS

```
public function call(string $function, array $params = []): array {
    if ($this->token === null) {
        throw new \RuntimeException('Спочатку викличте authenticate().');
    }
    $form_params = array_merge([
        'wstoken'          => $this->token,
        'wsfunction'       => $function,
        'moodlewsrestformat' => 'json',
    ], $params);
    $raw = $this->do_post_form(
        $this->base_url . '/webservice/rest/server.php',
        $form_params
    );
    $data = json_decode($raw, true);
    if (json_last_error() !== JSON_ERROR_NONE) {
        throw new \RuntimeException(
            "Не вдалося розібрати JSON [{function}]: " . json_last_error_msg()
        );
    }
    if ($data === null) {
        return [];
    }
    if (isset($data['exception'])) {
        throw new \RuntimeException(sprintf(
            '[%s] %s | %s',
            $data['exception'],
            $data['message'] ?? '-',
            $data['debuginfo'] ?? '-'
        ));
    }
    return $data;
}
```

Метод `sync_users()` реалізує ідемпотентну синхронізацію – повторний запуск не призводить до дублювання даних. Спочатку викликається `filter_duplicates()` (захист від некоректних вхідних даних), потім `get_existing_users()` – один batch-запит для перевірки всіх email одночасно. Далі масив ділиться на існуючих (зберігаємо їх ID) та нових (передаємо до `core_user_create_users`). Поле `cohort1` видаляється перед передачею у WS, оскільки Moodle API його не знає. Початковий пароль для нового облікового запису передається безпосередньо з ML-платформи у полі `password` вхідного масиву та встановлюється примусова зміна пароля при першому вході (`auth_forcepasswordchange=1`). У лістингу 3.6 наведено метод побудови параметрів користувача.

Лістинг 3.6 – Метод `build_users_params()` – формування параметрів для Moodle WS

```
private function build_users_params(array $users): array {
    $params = [];
    $users = array_values($users);
    foreach ($users as $i => $user) {
        foreach ($user as $key => $value) {
            if (!is_array($value)) {
                $params["users[{$i}][{$key}]" ] = $value;
            }
        }
        $preferences = $user['preferences'] ?? [];
        $has_force_change = false;
        foreach ($preferences as $pref) {
            if ($pref['type'] === 'auth_forcepasswordchange') {
                $has_force_change = true; break;
            }
        }
        if (!$has_force_change) {
            $preferences[] = ['type' => 'auth_forcepasswordchange', 'value'
=> '1'];
        }
        foreach ($preferences as $j => $pref) {
            $params["users[{$i}][preferences][{$j}][type]" ] = $pref['type'];
            $params["users[{$i}][preferences][{$j}][value]" ] = $pref['value'];
        }
    }
    return $params;
}
```

кінець лістингу 3.6

Когорта (cohort) у Moodle – це іменована група користувачів на рівні системи, аналог академічної групи. Когорти прив'язуються до курсів для автоматичного запису студентів. Методи управління когортами у класі moodle_client наведено у таблиці 3.3.

Таблиця 3.3 – Методи управління когортами класу moodle_client

Метод	Призначення
get_cohort_id(string \$id)	Пошук когорти за idnumber або name через core_cohort_search_cohorts; результат кешується у \$cohort_cache для уникнення повторних запитів
create_cohort(string \$name)	Створення нової когорти в системному контексті; ім'я задається і як name, і як idnumber
find_or_create_cohort(string \$name)	Комбінований: get_cohort_id() → якщо null → create_cohort(); завжди повертає валідний ID
get_user_cohort_ids(int \$uid)	Batch-запит: усі когорти системи + core_cohort_get_cohort_members → визначає членство конкретного студента
add_user_to_cohort()	Додавання через core_cohort_add_cohort_members з типізацією по ID
remove_user_from_cohort()	Видалення через core_cohort_delete_cohort_members
sync_user_cohort()	Визначає поточне членство, вилучає зайве, додає до цільової когорти

Метод sync_user_cohort() реалізує логіку «один студент – одна когорта»: якщо студент вже в правильній когорті – виходить без змін; якщо в іншій – видаляє зі старої та додає до нової. У лістингу 3.7 наведено реалізацію методу.

Лістинг 3.7 – Метод sync_user_cohort() – синхронізація когорти студента

```

public function sync_user_cohort(
    int    $user_id,
    string $username,
    string $cohort_identifier
): void {
    $target = $this->find_or_create_cohort($cohort_identifier);
    $current = $this->get_user_cohort_ids($user_id);

    if (in_array($target, $current, true)) {
        $this->log("{ $username } вже в когорті '{ $cohort_identifier }');
        return;
    }
}

```

```

foreach ($current as $old_id) {
    $this->remove_user_from_cohort($user_id, $old_id);
    $this->log("{ $username}: вилучено з когорти ID={ $old_id}");
}

$this->add_user_to_cohort($user_id, $target);
$this->log("{ $username}: додано до когорти '{ $cohort_identifier}'");
}

```

кінець лістингу 3.7

Клас `sync_users` (Scheduled Task), повний код якого наведено в додатку В. Клас (файл `classes/task/sync_users.php`, namespace `local_mlsync\task`) розширює `core\task\scheduled_task` та є оркестратором – не містить бізнес-логіки, лише координує роботу `ml_client` та `moodle_client`. Параметри підключення до обох систем визначені як приватні константи класу. Метод `execute()` викликається Moodle cron runner та реалізує чотири послідовні кроки: отримання даних з ML, авторизація у Moodle WS, синхронізація облікових записів, синхронізація когорти. У лістингу 3.8 наведено реалізацію методу.

Лістинг 3.8 – Метод `execute()` класу `sync_users` – головна точка входу cron

```

public function execute(): void {
    $this->log('=== Старт синхронізації ML → Moodle ===');

    $this->log('1. Отримання даних з ML-платформи...');

    $ml = new ml_client(self::ML_URL, self::ML_USER, self::ML_PASS);
    $users = $ml->fetch_users();

    if (empty($users)) {
        $this->log('Порожній список. Синхронізацію завершено.');
```

return;

```

    }

    $this->log('Отримано записів: ' . count($users));

    $this->log('2. Авторизація у Moodle WS...');

    $moodle = new moodle_client(self::MOODLE_URL);
    $moodle->authenticate(self::MOODLE_USER, self::MOODLE_PASS,
self::MOODLE_SERVICE);

```

```

$this->log('Токен отримано.');
```

```

$this->log('3. Синхронізація користувачів...');
```

```

$user_map = $moodle->sync_users($users);

// Додатковий map: email => реальний_username (для когорт)
$email_to_real_username = [];
foreach ($user_map as $real_username => $moodle_id) {
    foreach ($users as $u) {
        if (isset($u['username']) && $u['username'] === $real_username) {
            $email_to_real_username[strtolower($u['email'])] =
$real_username;
            break;
        }
    }
}

$this->log('4. Синхронізація когорт...');
```

```

foreach ($users as $user) {
    $cohort = $user['cohort1'] ?? null;

    if ($cohort === null) {
        $this->log("'{$user['username']}': cohort1 -, пропускаємо");
        continue;
    }

    $email_key      = strtolower($user['email']);
    $real_username  = $email_to_real_username[$email_key] ??
$user['username'];
    $user_id        = $user_map[$real_username] ?? null;

    if ($user_id === null) {
        $this->log("'{$user['username']}': ID -, пропускаємо");
        continue;
    }

    $moodle->sync_user_cohort((int) $user_id, $real_username,
$cohort);
}

$this->log('=== Синхронізацію завершено успішно. ===');
```

```

}
```

кінець лістингу 3.8

Важливою особливістю є захист від порожнього результату: якщо ML-платформа тимчасово недоступна та `fetch_users()` повертає порожній масив, задача завершується без помилки і не позначається Moodle як failed. Всі

повідомлення виводяться через `mtrace()` – стандарт для Moodle cron; журнал доступний в адмінпанелі.

3.3 Захист інформаційно-комп'ютерної системи

Безпека програмного модуля забезпечується на трьох рівнях.

Перший рівень – GDPR-відповідність. З версії Moodle 3.3 кожен плагін зобов'язаний декларувати персональні дані, реалізуючи інтерфейс `core_privacy\local\metadata\provider`. Оскільки `local_mlsync` не зберігає даних у власних таблицях БД (лише записує у стандартні таблиці Moodle через офіційний WS API), метод `get_metadata()` повертає порожню `collection`. У лістингу 3.9 наведено реалізацію `privacy provider`.

Лістинг 3.9 – Файл `classes/privacy/provider.php` – GDPR-декларація

```
<?php
namespace local_mlsync\privacy;
defined('MOODLE_INTERNAL') || die();
use core_privacy\local\metadata\collection;

class provider implements \core_privacy\local\metadata\provider {
    public static function get_metadata(
        collection $collection): collection {
        return $collection; // власних даних не зберігаємо
    }
}
```

кінець лістингу 3.9

Другий рівень – захист авторизаційних даних. Паролі та токени зберігаються виключно у `private`-константах класу і ніколи не потрапляють у журнали – метод `log()` виводить лише інформаційні повідомлення. Передача даних між Moodle та ML-платформою відбувається по захищеному каналу HTTPS.

Третій рівень – захист цілісності даних. Метод `filter_duplicates()` видаляє записи з однаковим `email` або `username`, запобігаючи спробам створити

дублікати. Ідемпотентний алгоритм синхронізації перевіряє поточний стан перед кожною операцією зміни (створення користувача, додавання до когорти), що забезпечує цілісність бази даних Moodle при повторних запусках cron.

3.4 Тестування та налагодження інформаційно-комп'ютерної системи

Встановлення плагіну виконується стандартним способом: директорія `local_mlsync` копіюється у `{moodle_root}/local/`, після чого адміністратор підтверджує встановлення через Moodle → Керування сайтом → Модулі.

Вигляд встановленого плагіна `local_mlsync` в Moodle зображено на рисунку 3.3.

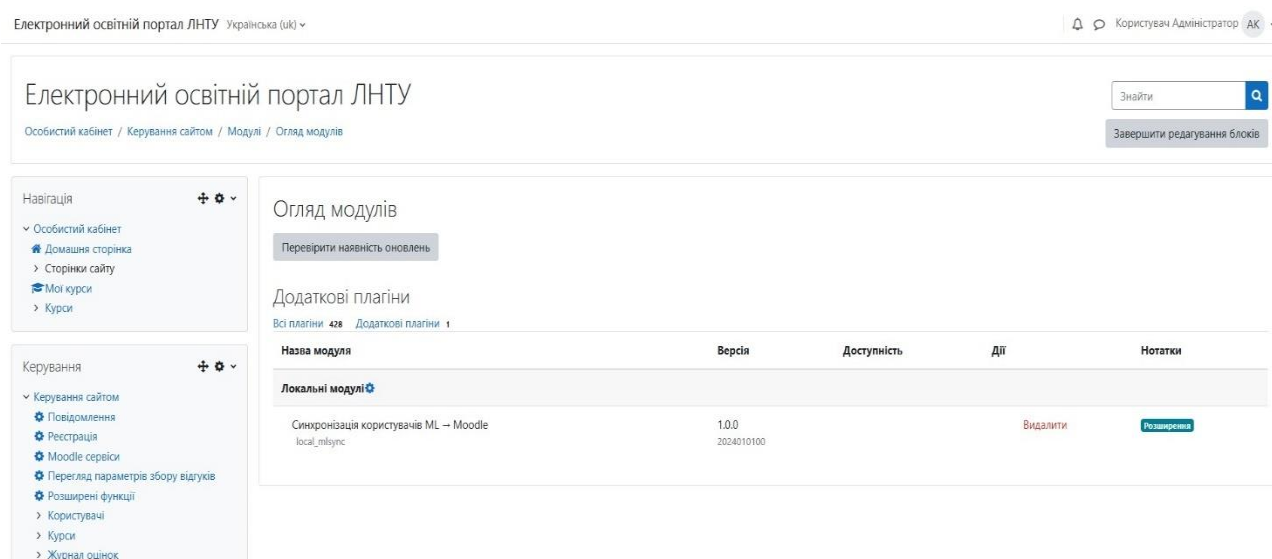


Рисунок 3.3 – Вигляд встановленого модуля через адміністративну панель

Система автоматично виконує `db/install.php` та реєструє `scheduled task`. Після встановлення завдання відображається в адмінпанелі: Керування сайтом → Сервер → Заплановані завдання → Синхронізація користувачів з ML платформи.

На рисунку 3.4 зображено логи результату виконання нашого плагіну.

Рисунок 3.5 – Повторне виконання завдання по синхронізації облікових записів

Для тестування та налагодження плагін підтримує ручний запуск через командний рядок без очікування настання часу cron. У лістингу 3.10 наведено команди для запуску та перегляду журналу в терміналі.

Лістинг 3.10 – Команди для ручного запуску та перегляду журналу

```
# Запуск задачі через CLI Moodle:
php admin/tool/task/cli/schedule_task.php \
    --execute='\local_mlsync\task\sync_users'
# Перегляд журналу виконання в реальному часі:
tail -f /path/to/moodle/dataroot/temp/tasklog/*.log
```

кінець лістингу 3.10

Під час виконання завдання виводяться детальні повідомлення через `mtrace()`. Журнал фіксує кожен етап: старт, кількість отриманих студентів з ML-платформи, результат авторизації у Moodle WS, список створених та пропущених (вже існуючих) студентів, результат синхронізації когорти для кожного студента та підсумкове повідомлення.

Функціональне тестування охоплювало три основні сценарії. Перший – студент вперше синхронізується: у Moodle створюється новий обліковий запис, присвоюється тимчасовий пароль і студент додається до вказаної когорти. Другий – ідемпотентний повторний запуск: студент вже існує в Moodle і перебуває в правильній когорті; наш плагін визначає це і не виконує жодних змін. Третій – переведення між групами: студент існує в Moodle, але його поточна когорта відрізняється від вказаної; метод `sync_user_cohort()` видаляє студента зі старої когорти і додає до нової.

Результат створеної когорти (академічної групи) та співставлення із конкретним обліковим записом студента зображено на рисунку 3.6.

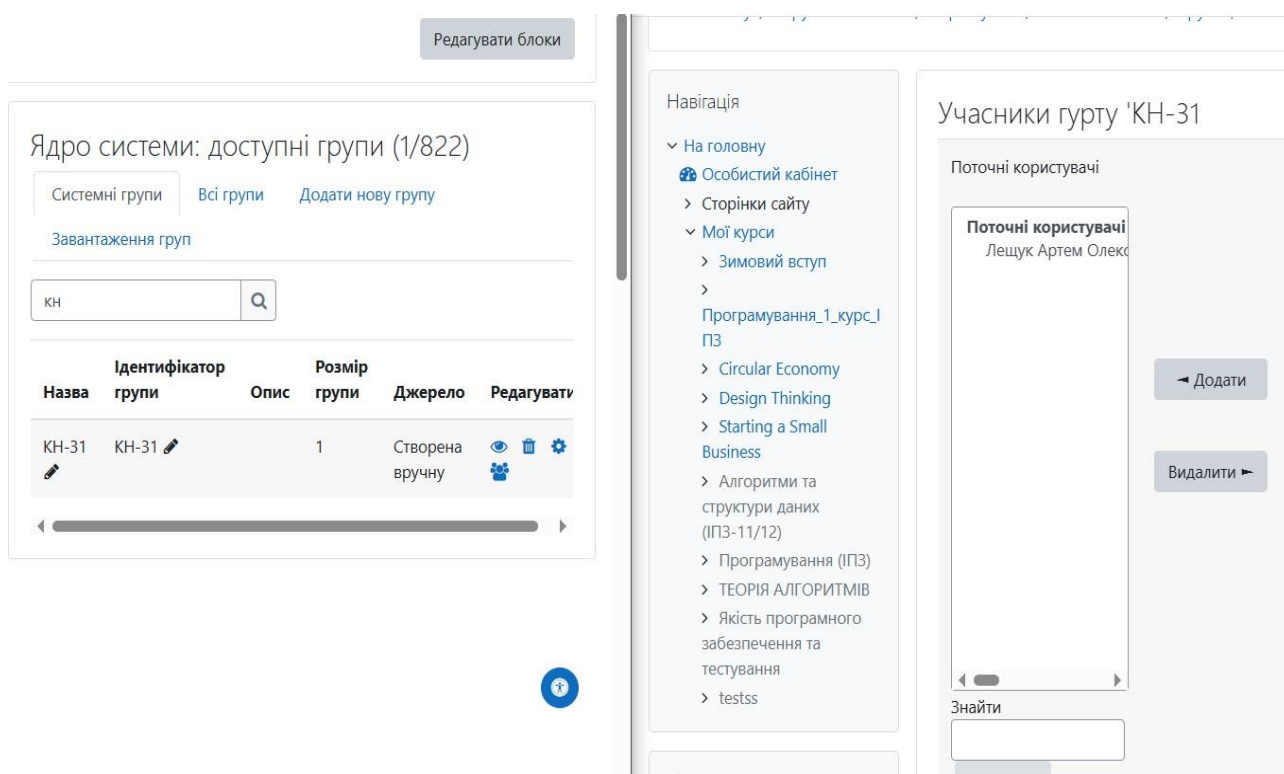


Рисунок 3.6 – Вигляд створеної когорти та учасники в ній

Тестування стійкості до помилок перевіряло такі сценарії: недоступність ML-платформи (`fetch_users()` повертає порожній масив, завдання завершується без помилки і не позначається як `failed`), помилка WS API Moodle (`RuntimeException` перехоплюється і записується у лог), дублікати у вхідних даних (`filter_duplicates()` відфільтровує з попередженням), відсутнє поле `cohort1` (запис пропускається з попередженням). Всі виявлені проблеми усунено.

Мінімальні системні вимоги для коректної роботи плагіну наведено у таблиці 3.4.

Таблиця 3.4 – Мінімальні системні вимоги для роботи `local_mlsync`

Вимога	Значення / Деталі
Версія Moodle	4.0.4+ (числовий код 2022041900)
Версія PHP	7.4 або вище (використовується нативний curl без CurlHandle)
Веб-сервіси	Увімкнено: Керування сайтом → Сервер → Увімкнути веб-сервіси
Зовнішні сервіси	Сервіс « <code>my_api</code> » з функціями управління користувачами та когортами
REST protocol	Увімкнено: Site Admin → Plugins → Web services → Manage protocols
Мережа	HTTPS-доступ з сервера Moodle до <code>ml.lntu.edu.ua</code>
Cron	Системний або Moodle cron налаштований і виконується регулярно

3.5 Модульне розширення та масштабування системи

Архітектура плагіну спроектована з урахуванням принципів розділення відповідальностей та єдиної відповідальності. Три незалежні класи – `ml_client`, `moodle_client` та `sync_users` – мають чітко визначені зони відповідальності та мінімальну зв'язаність між собою, що відповідає сучасним підходам до розробки модульних систем.

Клас `ml_client` можна замінити або доповнити без будь-яких змін в інших класах: замість Drupal REST можна реалізувати OAuth 2.0, LDAP, читання з CSV-

файлу або підключення до іншої бази даних. Достатньо реалізувати метод `fetch_users()` з тією самою сигнатурою та структурою поверненого масиву.

На рисунку 3.7 зображено архітектуру модуля `local_mlsync` та можливості його розширення.

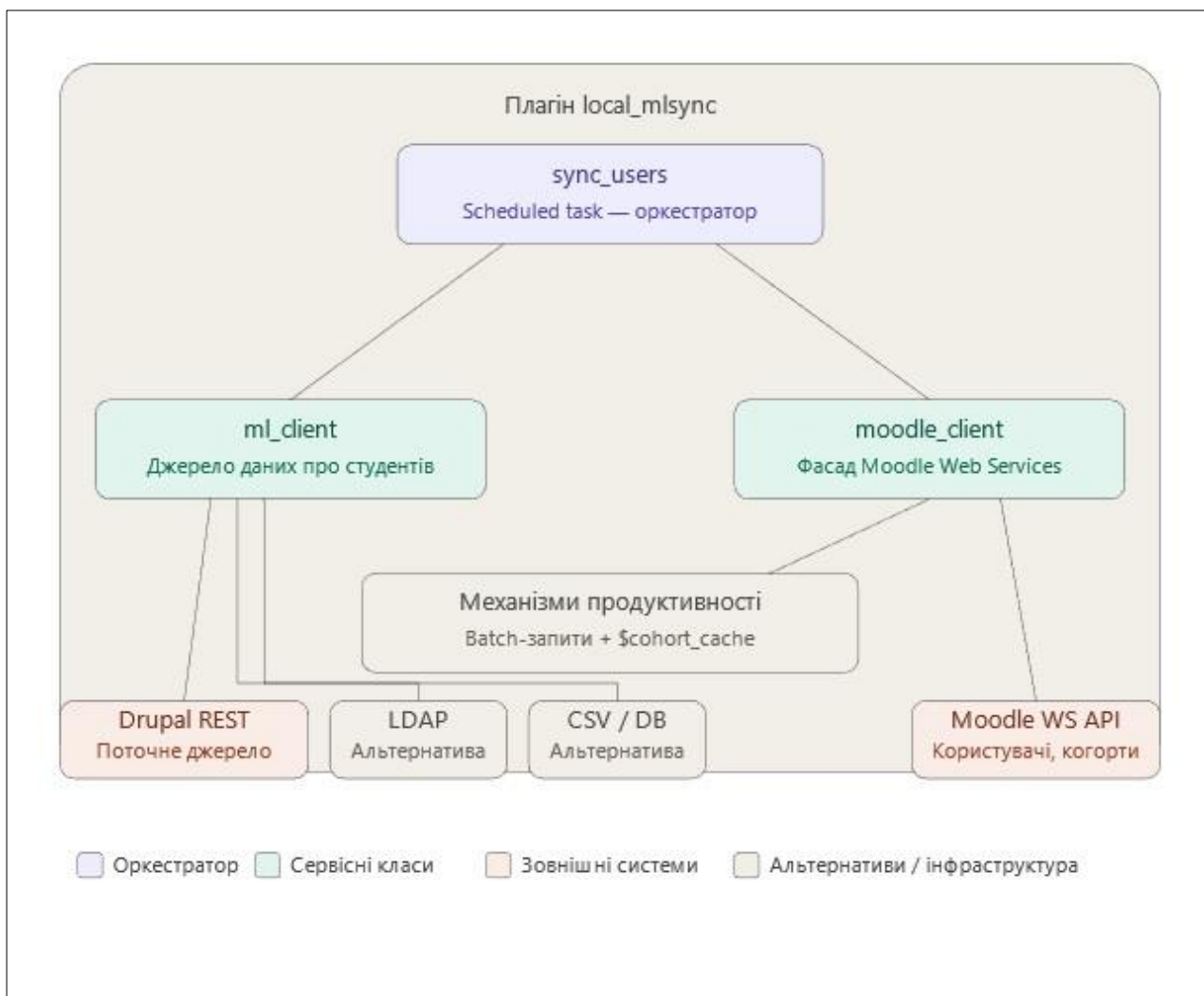


Рисунок 3.7 – Архітектура модуля `local_mlsync` та можливості його розширення

Клас `moodle_client` є універсальним фасадом для Moodle WS і може бути перевикористаний в інших плагінах. Метод `call()` дозволяє викликати будь-яку WS-функцію Moodle. Можливе розширення класу методами для управління курсами, записами студентів на курси, оцінками тощо.

Продуктивність при великій кількості студентів забезпечується двома механізмами. По-перше, batch-запити мінімізують кількість HTTP-звернень до

Moodle WS: перевірка існуючих студентів виконується одним запитом для всіх email, масове створення нових – також одним запитом. По-друге, локальний кеш когорт (\$cohort_cache) гарантує що пошук конкретної когорти виконується лише один раз за запуск незалежно від кількості студентів у ній.

Висновки до розділу 3

У третьому розділі здійснено практичну реалізацію програмного модуля local_mlsync для LMS Moodle 4.0.4+ – повноцінного плагіну типу local, що відповідає всім стандартам розробки Moodle-розширень.

Реалізовано три ключові класи: ml_client, що забезпечує двоетапну авторизацію на Drupal REST API (CSRF-токен та session cookie) та отримання даних з Views endpoint; moodle_client – фасад для Moodle Web Services з підтримкою batch-операцій та локальним кешем когорт; sync_users – scheduled task-оркестратор, що виконується за розкладом cron і координує роботу двох сервісних класів.

Реалізовано ідемпотентний алгоритм синхронізації: кожна операція перевіряє поточний стан перед внесенням змін, що забезпечує безпечний повторний запуск та захист від дублювання даних. Впроваджено автоматичне присвоєння тимчасових паролів в Moodle з обов'язковою зміною при першому вході.

Забезпечено GDPR-відповідність через privacy/provider.php, захист передачі даних через HTTPS та захист від дублювання через filter_duplicates() [14]. Проведено функціональне тестування трьох основних сценаріїв та тестування стійкості до помилок. Всі виявлені проблеми усунено. Плагін готовий до встановлення у production-середовищі Moodle 4.0.4+ університету та має значний потенціал для розширення функціоналу.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи проаналізовано та реалізовано всі етапи розробки програмного модуля автоматичної синхронізації користувачів у LMS Moodle – плагіну `local_mlsync`.

Проаналізовано сучасний стан проблем управління обліковими записами у системах електронного навчання. Встановлено, що більшість університетів використовує кілька інформаційних систем одночасно, що породжує проблему синхронізації даних між ними. Визначено, що ручне адміністрування облікових записів призводить до технічних затримок, помилок при введенні даних та надмірного навантаження на адміністраторів, що негативно впливає на ефективність навчального процесу.

Досліджено існуючі підходи та інструменти інтеграції Moodle з зовнішніми системами, зокрема LDAP, Active Directory та Single Sign-On. Проаналізовано їхні переваги та обмеження в контексті конкретної інфраструктури університету. Визначено, що розробка спеціалізованого плагіну з використанням Moodle Web Services та `scheduled tasks` є оптимальним підходом для інтеграції з ML-платформою на базі Drupal.

Сформульовано вимоги до розроблюваного програмного модуля. Визначено функціональні вимоги: авторизація на ML-платформі, отримання та обробка даних студентів, автоматичне створення облікових записів у Moodle, управління когортами та обробка переведення між групами. Визначено нефункціональні вимоги: ідемпотентність, стійкість до помилок, GDPR-сумісність та відповідність стандартам розробки плагінів Moodle.

Обґрунтовано вибір технологій та архітектурних рішень. Обрано тип плагіну `local` як найбільш гнучкий для реалізації фонових задач, Moodle Scheduled Tasks для автоматичного запуску за розкладом, Moodle Web Services REST API для взаємодії з ядром платформи, вбудований `curl` Moodle для HTTP-запитів та PSR-4 автозавантаження для організації структури класів.

Архітектурний шаблон «scheduled task + service layer» забезпечує чіткий розподіл відповідальності між компонентами модуля.

Практично реалізовано плагін `local_mlsync` для Moodle 4.0.4+. Реалізовано ідемпотентний алгоритм синхронізації, в якому кожна операція перевіряє поточний стан перед внесенням змін, що забезпечує безпечний повторний запуск та захист від дублювання даних. Впроваджено batch-запити для мінімізації кількості HTTP-запитів до Moodle WS та локальний кеш когорт для підвищення продуктивності.

Описано структуру класів та логіку роботи модуля. Розроблено три незалежні класи: `ml_client`, `moodle_client` та `sync_users`. Клас `ml_client` реалізує двоетапну авторизацію у Drupal REST API (CSRF-токен + cookie) та отримання даних з Views endpoint. Клас `moodle_client` є фасадом для Moodle WS з підтримкою всіх необхідних операцій з користувачами та когортами. Клас `sync_users` є scheduled task-оркестратором, що координує виконання повного циклу синхронізації за розкладом.

Забезпечено захист даних та стійкість модуля до помилок. Реалізовано GDPR-декларацію (`privacy/provider.php`), захист передачі даних через HTTPS, примусову зміну пароля для нових студентів. Стійкість до помилок реалізована таким чином, що тимчасова недоступність ML-платформи або помилка обробки окремого студента не спричиняє збою cron-завдання, а всі події фіксуються у системному журналі Moodle.

Проведено тестування та налагодження програмного модуля. Перевірено основні сценарії: створення нового користувача, повторний запуск синхронізації, переведення студента між когортами. Тестування стійкості до помилок включало симуляцію недоступності зовнішніх сервісів та обробку некоректних вхідних даних. Усі виявлені у процесі налагодження проблеми усунуено, модуль успішно пройшов усі контрольні сценарії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cron job scheduling. Wikipedia. URL: <https://en.wikipedia.org/wiki/Cron> (дата звернення: 29.03.2026).
2. Drupal: Views module. Drupal.org. URL: <https://www.drupal.org/project/views> (дата звернення: 29.03.2026).
3. Drupal REST API documentation. JSON:API module. URL: <https://www.drupal.org/docs/core-modules-and-themes/core-modules/jsonapi-module> (дата звернення: 28.03.2026).
4. GDPR and Moodle. URL: <https://moodle.org/mod/forum/discuss.php?d=354332> (дата звернення: 29.03.2026).
5. LMS Integration Patterns. eLearning Industry. URL: <https://elearningindustry.com> (дата звернення: 29.03.2026).
6. Moodle 4.0 Release Notes. URL: https://docs.moodle.org/dev/Moodle_4.0_release_notes (дата звернення: 29.03.2026).
7. Moodle App Plugins Development Guide. URL: <https://moodledev.io/general/app/development/plugins-development-guide> (дата звернення: 28.03.2026).
8. Moodle Coding Style. Moodle Developer Resources. URL: <https://moodledev.io/general/development/policies/codingstyle> (дата звернення: 28.03.2026).
9. Moodle Documentation: API guides. Local plugins. URL: <https://moodledev.io/docs/4.5/apis/plugintypes/local> (дата звернення: 28.03.2026).
10. Moodle Documentation: Managing a Moodle site. Authentication. URL: <https://docs.moodle.org/404/en/Authentication> (дата звернення: 27.03.2026).
11. Moodle Documentation: Managing a Moodle site. Cohorts. URL: <https://docs.moodle.org/404/en/Cohorts> (дата звернення: 28.03.2026).
12. Moodle Documentation: Managing a Moodle site. Scheduled tasks. URL: https://docs.moodle.org/404/en/Scheduled_tasks (дата звернення: 28.03.2026).

13. Moodle Documentation: Managing a Moodle site. Web services. URL: https://docs.moodle.org/404/en/Web_services (дата звернення: 28.03.2026).

14. Moodle Documentation: Privacy API. URL: <https://moodledev.io/docs/4.5/apis/subsystems/privacy> (дата звернення: 29.03.2026).

15. Moodle statistics. Moodle HQ. URL: <https://stats.moodle.org/> (дата звернення: 27.03.2026).

16. Moodle Web Services API. Moodle HQ. URL: https://docs.moodle.org/dev/Web_service_API_functions (дата звернення: 28.03.2026).

17. PHP: cURL Functions. PHP Documentation. URL: <https://www.php.net/manual/en/book.curl.php> (дата звернення: 28.03.2026).

18. PHP 8.0 What's New. PHP.net. URL: <https://www.php.net/releases/8.0/en.php> (дата звернення: 29.03.2026).

19. PSR-4: Autoloader. PHP-FIG. URL: <https://www.php-fig.org/psr/psr-4/> (дата звернення: 28.03.2026).

20. REST API Design Best Practices. RESTful API Tutorial. URL: <https://restfulapi.net/rest-api-design-tutorial-with-example/> (дата звернення: 28.03.2026).

ДОДАТКИ

Додаток А – Повний код класу ml_client

```

<?php

namespace local_mlsync;

defined('MOODLE_INTERNAL') || die();

class ml_client {

    private string $base_url;

    private string $username;

    private string $password;

    private ?string $csrf_token = null;

    private string $cookie_string = '';

    public function __construct(string $base_url, string $username, string
$password) {
        $this->base_url = rtrim($base_url, '/');
        $this->username = $username;
        $this->password = $password;
    }

    public function fetch_users(): array {
        try {
            $this->authenticate();
            return $this->get_users_to_create();
        } catch (\Exception $e) {
            $this->log('ERROR fetch_users: ' . $e->getMessage());
            return [];
        }
    }

    private function authenticate(): void {
        // Крок 1: CSRF-токен
        [$token_body, $token_headers] = $this->do_get(
            $this->base_url . '/session/token',
            []
        );
        $this->csrf_token = trim($token_body);
        $this->cookie_string = $this->extract_cookies($token_headers);
        $this->log("CSRF токен отримано: {$this->csrf_token}");

        // Крок 2: авторизація
        $headers = [
            'Content-Type: application/json',
            'Accept: application/json',
        ];
    }
}

```

```

        if ($this->cookie_string !== '') {
            $headers[] = 'Cookie: ' . $this->cookie_string;
        }

        [$login_raw, $login_headers] = $this->do_post_json(
            $this->base_url . '/user/login?_format=json',
            json_encode(['name' => $this->username, 'pass' => $this-
>password]),
            $headers
        );

        $login_data = json_decode($login_raw, true);
        if (!empty($login_data['csrf_token'])) {
            $this->csrf_token = $login_data['csrf_token'];
        }

        $new_cookies = $this->extract_cookies($login_headers);
        if ($new_cookies !== '') {
            $this->cookie_string = $new_cookies;
        }

        $this->log("Авторизація успішна, csrf_token: {$this-
>csrf_token}");
    }

    private function get_users_to_create(): array {
        $headers = [
            'Accept: application/json',
            'X-CSRF-Token: ' . $this->csrf_token,
        ];
        if ($this->cookie_string !== '') {
            $headers[] = 'Cookie: ' . $this->cookie_string;
        }

        [$raw] = $this->do_get(
            $this->base_url . '/check_moodle_user_to_create',
            $headers
        );

        $this->log("ML відповідь (перші 300 символів): " . substr($raw, 0,
300));

        $data = json_decode($raw, true);
        if (json_last_error() !== JSON_ERROR_NONE) {
            throw new \RuntimeException('Помилка JSON з ML: ' .
json_last_error_msg());
        }

        return is_array($data) ? $data : [];
    }
}

```

```

private function do_get(string $url, array $headers): array {
    $ch = $this->make_curl($url, $headers);
    return $this->run_curl($ch, $url);
}

private function do_post_json(string $url, string $body, array
$headers): array {
    $ch = $this->make_curl($url, $headers);
    curl_setopt($ch, CURLOPT_POST, true);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $body);
    return $this->run_curl($ch, $url);
}

private function make_curl(string $url, array $headers) {
    $ch = curl_init($url);
    curl_setopt_array($ch, [
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_HEADER         => true,
        CURLOPT_TIMEOUT         => 30,
        CURLOPT_CONNECTTIMEOUT => 10,
        CURLOPT_SSL_VERIFYPEER => false,
        CURLOPT_SSL_VERIFYHOST => 0,
        CURLOPT_FOLLOWLOCATION  => true,
        CURLOPT_HTTPHEADER     => $headers,
    ]);
    return $ch;
}

private function run_curl($ch, string $url): array {
    $response = curl_exec($ch);
    $errno    = curl_errno($ch);
    $error    = curl_error($ch);
    $header_sz = curl_getinfo($ch, CURLINFO_HEADER_SIZE);
    $http_code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
    curl_close($ch);

    if ($errno || $response === false) {
        throw new \RuntimeException("curl помилка для {$url}:
{$error}");
    }

    $headers_str = substr($response, 0, $header_sz);
    $body        = substr($response, $header_sz);

    $this->log("HTTP {$http_code} ← {$url}");

    if ($http_code >= 400) {
        throw new \RuntimeException(
            "HTTP {$http_code} для {$url}. Тіло: " . substr($body, 0,
300)
        );
    }
}

```

```

    }

    return [$body, $headers_str];
}

private function extract_cookies(string $headers_str): string {
    $cookies = [];
    foreach (explode("\r\n", $headers_str) as $line) {
        if (stripos($line, 'Set-Cookie:') === 0) {
            $cookie_part = trim(substr($line, strlen('Set-
Cookie:')));
            $semicolon = strpos($cookie_part, ';');
            $cookies[] = $semicolon !== false
                ? substr($cookie_part, 0, $semicolon)
                : $cookie_part;
        }
    }
    return implode('; ', $cookies);
}

private function log(string $message): void {
    if (defined('CLI_SCRIPT') && CLI_SCRIPT) {
        mtrace('[local_mlsync][ml_client] ' . $message);
    }
    debugging('[local_mlsync][ml_client] ' . $message,
DEBUG_DEVELOPER);
}
}

```

кінець додатку А

Додаток Б – Повний код класу moodle_client

```

<?php
namespace local_mlsync;

defined('MOODLE_INTERNAL') || die();

class moodle_client {

    private string $base_url;

    private ?string $token = null;

    private array $cohort_cache = [];

    public function __construct(string $base_url) {
        $this->base_url = rtrim($base_url, '/');
    }

    public function authenticate(
        string $username,
        string $password,
        string $service = 'moodle_mobile_app'
    ): void {
        $raw = $this->do_post_form($this->base_url . '/login/token.php',
[
            'username' => $username,
            'password' => $password,
            'service'   => $service,
        ]);

        $data = json_decode($raw, true);

        if (!empty($data['error'])) {
            throw new \RuntimeException('Помилка авторизації Moodle WS: '
. $data['error']);
        }
        if (empty($data['token'])) {
            throw new \RuntimeException(
                'Moodle токен не отримано. Відп: ' . substr($raw, 0, 300)
            );
        }

        $this->token = $data['token'];
    }

    public function call(string $function, array $params = []): array {
        if ($this->token === null) {
            throw new \RuntimeException('Спочатку викличте
authenticate().');
        }
    }
}

```

```

        $form_params = array_merge([
            'wstoken'          => $this->token,
            'wsfunction'       => $function,
            'moodlewsrestformat' => 'json',
        ], $params);

        $raw = $this->do_post_form(
            $this->base_url . '/webservice/rest/server.php',
            $form_params
        );
        $data = json_decode($raw, true);

        if (json_last_error() !== JSON_ERROR_NONE) {
            throw new \RuntimeException(
                "Не вдалося розібрати JSON [{$function}]: " .
                json_last_error_msg()
            );
        }

        if ($data === null) {
            return [];
        }

        if (isset($data['exception'])) {
            throw new \RuntimeException(sprintf(
                '[%s] %s | %s',
                $data['exception'],
                $data['message'] ?? '-',
                $data['debuginfo'] ?? '-'
            ));
        }

        return $data;
    }

    public function get_existing_users(array $users): array {
        if (empty($users)) {
            return [];
        }

        $params = ['field' => 'email'];
        foreach (array_values(array_column($users, 'email')) as $i =>
$email) {
            $params["values[{$i}]"] = $email;
        }

        $result = $this->call('core_user_get_users_by_field', $params);

        $map = [];
        foreach ($result as $u) {
            $map[strtolower($u['email'])] = [

```

```

        'id'          => (int) $u['id'],
        'username' => $u['username'],
    ];
}
return $map;
}

public function sync_users(array $users): array {
    $users      = $this->filter_duplicates($users);
    $existing    = $this->get_existing_users($users);

    $to_create   = [];
    $username_map = [];

    foreach ($users as $user) {
        $key = strtolower($user['email']);
        if (isset($existing[$key])) {
            $real = $existing[$key]['username'];
            if ($real !== $user['username']) {
                $this->log("'{$user['email']}': реальный username у
Moodle - '{$real}'");
            }
            $username_map[$real] = $existing[$key]['id'];
        } else {
            $to_create[] = $user;
        }
    }

    if (!empty($username_map)) {
        $this->log('Вже існують: ' . implode(', ',
array_keys($username_map)));
    }

    if (!empty($to_create)) {
        $this->log('Створюємо: ' . implode(', ',
array_column($to_create, 'username')));
        $api_users = array_map(
            fn($u) => array_diff_key($u, ['cohort1' => '']),
            $to_create
        );
        $created = $this->call(
            'core_user_create_users',
            $this->build_users_params($api_users)
        );
        foreach ($created as $c) {
            $username_map[$c['username']] = (int) $c['id'];
        }
    } else {
        $this->log('Нових користувачів немає.');
    }
}

```

```

        return $username_map;
    }

    private function filter_duplicates(array $users): array {
        $seen_emails = $seen_usernames = $filtered = [];
        foreach ($users as $user) {
            $email = strtolower(trim($user['email']));
            $username = trim($user['username']);
            if (isset($seen_emails[$email])) {
                $this->log("Дубликат email '{$email}' – пропускаемо");
                continue;
            }
            if (isset($seen_usernames[$username])) {
                $this->log("Дубликат username '{$username}' – пропускаемо");
                continue;
            }
            $seen_emails[$email] = $seen_usernames[$username] = true;
            $filtered[] = $user;
        }
        return $filtered;
    }

    public function get_cohort_id(string $cohort_identifier): ?int {
        if (isset($this->cohort_cache[$cohort_identifier])) {
            return $this->cohort_cache[$cohort_identifier];
        }

        $result = $this->call('core_cohort_search_cohorts', [
            'query' => $cohort_identifier,
            'context[contextid]' => 1,
            'context[contextlevel]' => 'system',
            'context[instanceid]' => 0,
            'includes' => 'all',
            'limitfrom' => 0,
            'limitnum' => 50,
        ]);

        foreach ($result['cohorts'] ?? [] as $cohort) {
            if ($cohort['idnumber'] === $cohort_identifier
                || $cohort['name'] === $cohort_identifier) {
                $id = (int) $cohort['id'];
                $this->cohort_cache[$cohort_identifier] = $id;
                return $id;
            }
        }
        return null;
    }

    public function create_cohort(string $name): int {
        $result = $this->call('core_cohort_create_cohorts', [

```

```

        'cohorts[0][categorytype][type]' => 'system',
        'cohorts[0][categorytype][value]' => '',
        'cohorts[0][name]' => $name,
        'cohorts[0][idnumber]' => $name,
        'cohorts[0][description]' => '',
        'cohorts[0][descriptionformat]' => 1,
        'cohorts[0][visible]' => 1,
    ));

    $id = (int) $result[0]['id'];
    $this->cohort_cache[$name] = $id;
    return $id;
}

public function find_or_create_cohort(string $name): int {
    $id = $this->get_cohort_id($name);
    if ($id === null) {
        $this->log("Когорту '{$name}' не найдено – створюємо...");
        $id = $this->create_cohort($name);
        $this->log("Когорту '{$name}' створено (ID={$id})");
    }
    return $id;
}

public function get_user_cohort_ids(int $user_id): array {
    $all = $this->call('core_cohort_search_cohorts', [
        'query' => '',
        'context[contextid]' => 1,
        'context[contextlevel]' => 'system',
        'context[instanceid]' => 0,
        'includes' => 'all',
        'limitfrom' => 0,
        'limitnum' => 1000,
    ]);

    $cohorts = $all['cohorts'] ?? [];
    if (empty($cohorts)) {
        return [];
    }

    $params = [];
    foreach (array_values($cohorts) as $i => $cohort) {
        $params["cohortids[{$i}]"] = (int) $cohort['id'];
    }

    $members_data = $this->call('core_cohort_get_cohort_members',
    $params);
    $member_of = [];

    foreach ($members_data as $group) {
        $cohort_id = (int) $group['cohortid'];

```

```

        foreach ($group['userids'] ?? [] as $member) {
            $uid = is_array($member) ? (int) $member['id'] : (int)
$member;
            if ($uid === $user_id) {
                $member_of[] = $cohort_id;
                break;
            }
        }
    }
    return $member_of;
}

public function add_user_to_cohort(int $user_id, int $cohort_id):
void {
    $this->call('core_cohort_add_cohort_members', [
        'members[0][cohorttype][type]' => 'id',
        'members[0][cohorttype][value]' => $cohort_id,
        'members[0][usertype][type]'    => 'id',
        'members[0][usertype][value]'   => $user_id,
    ]);
}

public function remove_user_from_cohort(int $user_id, int
$cohort_id): void {
    $this->call('core_cohort_delete_cohort_members', [
        'members[0][cohortid]' => $cohort_id,
        'members[0][userid]'   => $user_id,
    ]);
}

public function sync_user_cohort(
    int    $user_id,
    string $username,
    string $cohort_identifier
): void {
    $target = $this->find_or_create_cohort($cohort_identifier);
    $current = $this->get_user_cohort_ids($user_id);

    if (in_array($target, $current, true)) {
        $this->log("{ $username } вже в когорті
'{$cohort_identifier}'");
        return;
    }

    foreach ($current as $old_id) {
        $this->remove_user_from_cohort($user_id, $old_id);
        $this->log("{ $username } : вилучено з когорти ID={$old_id}");
    }

    $this->add_user_to_cohort($user_id, $target);
}

```

```

        $this->log("{ $username}: додано до когорти
'{$cohort_idenfifier}');
    }

    private function build_users_params(array $users): array {
        $params = [];
        $users = array_values($users);

        foreach ($users as $i => $user) {
            foreach ($user as $key => $value) {
                if (!is_array($value)) {
                    $params["users[{$i}][{$key}]" ] = $value;
                }
            }
            foreach ($user['customfields'] ?? [] as $j => $cf) {
                $params["users[{$i}][customfields][{$j}][type]" ] =
$cf['type'];
                $params["users[{$i}][customfields][{$j}][value]" ] =
$cf['value'];
            }

            $preferences = $user['preferences'] ?? [];
            $has_force_change = false;
            foreach ($preferences as $pref) {
                if ($pref['type'] === 'auth_forcepasswordchange') {
                    $has_force_change = true;
                    break;
                }
            }
            if (!$has_force_change) {
                $preferences[] = ['type' => 'auth_forcepasswordchange',
'value' => '1'];
            }
            foreach ($preferences as $j => $pref) {
                $params["users[{$i}][preferences][{$j}][type]" ] =
$pref['type'];
                $params["users[{$i}][preferences][{$j}][value]" ] =
$pref['value'];
            }
        }
        return $params;
    }

    private function do_post_form(string $url, array $params): string {
        $ch = curl_init($url);
        curl_setopt_array($ch, [
            CURLOPT_RETURNTRANSFER => true,
            CURLOPT_POST           => true,
            CURLOPT_POSTFIELDS     => $params,    // масив → curl кодує
cam, без http_build_query
            CURLOPT_TIMEOUT        => 30,

```

```

        CURLOPT_CONNECTTIMEOUT => 10,
        CURLOPT_SSL_VERIFYPEER => false,
        CURLOPT_SSL_VERIFYHOST => 0,
        CURLOPT_FOLLOWLOCATION => true,
        // Content-Type НЕ виставляємо – curl встановить його
автоматично
    ));

    $raw    = curl_exec($ch);
    $errno  = curl_errno($ch);
    $error  = curl_error($ch);
    curl_close($ch);

    if ($errno || $raw === false) {
        throw new \RuntimeException("curl POST помилка для {$url}:
{$error}");
    }
    return $raw;
}

private function log(string $message): void {
    if (defined('CLI_SCRIPT') && CLI_SCRIPT) {
        mtrace('[local_mlsync][moodle_client] ' . $message);
    }
    debugging('[local_mlsync][moodle_client] ' . $message,
DEBUG_DEVELOPER);
}
}

```

кінець додатку Б

Додаток В – Повний код класу sync_users

```
<?php

namespace local_mlsync\task;

defined('MOODLE_INTERNAL') || die();

use core\task\scheduled_task;
use local_mlsync\ml_client;
use local_mlsync\moodle_client;

class sync_users extends scheduled_task {

    private const MOODLE_URL = 'http://local.moodle.test:801';
    private const MOODLE_USER = 'my_user';
    private const MOODLE_PASS = '1my_user1Q';
    private const MOODLE_SERVICE = 'my_api';
    private const ML_URL = 'https://ml.lntu.edu.ua';
    private const ML_USER = 'admin';
    private const ML_PASS = 'admin';

    public function get_name(): string {
        return get_string('task_sync_users', 'local_mlsync');
    }

    public function execute(): void {
        $this->log('=== Старт синхронізації ML → Moodle ===');

        $this->log('1. Отримання даних з ML-платформи...');

        $ml = new ml_client(self::ML_URL, self::ML_USER, self::ML_PASS);
        $users = $ml->fetch_users();

        if (empty($users)) {
            $this->log('Порожній список. Синхронізацію завершено.');
```

```

        $user_map = $moodle->sync_users($users); // реальний_username =>
moodle_id

        $email_to_real_username = [];
        foreach ($user_map as $real_username => $moodle_id) {
            foreach ($users as $u) {
                if (isset($u['username']) && $u['username'] ===
$real_username) {
                    $email_to_real_username[strtolower($u['email'])] =
$real_username;
                    break;
                }
            }
        }

        $this->log('4. Синхронізація когорт...');

        foreach ($users as $user) {
            $cohort = $user['cohort1'] ?? null;

            if ($cohort === null) {
                $this->log("'{$user['username']}' : поле cohort1 не
вказано, пропускаємо");
                continue;
            }

            $email_key = strtolower($user['email']);
            $real_username = $email_to_real_username[$email_key] ??
$user['username'];
            $user_id = $user_map[$real_username] ?? null;

            if ($user_id === null) {
                $this->log("'{$user['username']}' : ID не знайдено,
пропускаємо");
                continue;
            }

            $moodle->sync_user_cohort((int) $user_id, $real_username,
$cohort);
        }

        $this->log('=== Синхронізацію завершено успішно. ===');
    }

    private function log(string $message): void {
        mtrace('[local_mlsync] ' . $message);
    }
}

```

кінець додатку В