УДК 004.65
DOI https://doi.org/10.24144/2616-7700.2025.47(2).136-147

**М. І. Глебена[1], О. В. Корник[2], О. В. Глебена[3], В. В. Чубирка[4]**

[1] ДВНЗ «Ужгородський національний університет»,
завідувач кафедри системного аналізу та теорії оптимізації,
кандидат фізико-математичних наук, доцент
myroslava.hlebena@uzhnu.edu.ua
ORCID: https://orcid.org/0000-0003-1100-515X

[2] ДВНЗ «Ужгородський національний університет»,
аспірант кафедри теорії ймовірностей і математичного аналізу
oleksandr.kornyk@uzhnu.edu.ua
ORCID: https://orcid.org/0000-0002-0170-2309

[3] ДВНЗ «Ужгородський національний університет»,
аспірант кафедри загальної педагогіки та педагогіки вищої школи
oleksandr.hlebena1@uzhnu.edu.ua
ORCID: https://orcid.org/0009-0006-2962-920X

[4] ДВНЗ «Ужгородський національний університет»,
аспірант кафедри системного аналізу та теорії оптимізації
viktor.chubyrka@uzhnu.edu.ua
ORCID: https://orcid.org/0009-0005-7556-6118

# РОЗРОБКА СТРАТЕГІЙ ВІДМОВОСТІЙКОГО ЗАВАНТАЖЕННЯ ДАНИХ ДЛЯ ІНФОРМАЦІЙНИХ СИСТЕМ НОВОГО ПОКОЛІННЯ

У статті представлено системний підхід до реалізації механізмів відмовостійкого завантаження даних в інформаційних системах, що інтегруються із зовнішніми сервісами, зокрема з ЄДЕБО через REST API. Проаналізовано ключові виклики, серед яких мережеві збої, обмеження кількості запитів, непередбачувані помилки зовнішніх серверів та проблеми цілісності даних. Запропонована архітектура ґрунтується на використанні Laravel Framework і включає модульну організацію процесів через консольні команди, механізми повторних спроб для обробки тимчасових помилок, транзакційну модель бази даних для забезпечення узгодженості інформації та комплексне логування з подальшим моніторингом у реальному часі. Окрему увагу приділено важливості регулярного тестування та впровадженню циклу зворотного зв'язку, що сприяє постійному вдосконаленню системи. Практична реалізація довела, що запропонований підхід не лише підвищує надійність та стійкість процесів завантаження, але й формує основу для масштабованості й адаптивності інформаційної системи в умовах зростання вимог до обробки даних та динамічних змін інформаційного середовища. Таким чином, відмовостійкість розглядається не як разове рішення, а як фундаментальний принцип розроблення сучасних інформаційних систем.

**Ключові слова:** відмовостійкість, великі дані, завантаження даних, інформаційні системи, REST API, Laravel Framework, транзакційна модель, логування, моніторинг, ЄДЕБО.

**1. Вступ.** У сучасному інформаційному суспільстві дані виступають одним із найцінніших ресурсів, що визначають ефективність функціонування цифрових систем і сервісів. Зростання обсягів та різноманітності джерел даних, зокрема зовнішніх (REST API та інші), актуалізує проблему організації надійного процесу їх завантаження. Водночас цей процес супроводжується низкою технічних і методологічних викликів, серед яких — забезпечення цілісності, достовірності та безперервності обробки інформації. Особливе значення у цьому

контексті має концепція відмовостійкого завантаження даних, яка передбачає здатність системи до коректної реакції на можливі помилки та збої без втрати продуктивності й коректності результатів. У даній статті здійснюється аналіз підходів до реалізації відмовостійких механізмів завантаження даних та їх практичного застосування в умовах сучасних інформаційних систем. Актуальність досліджуваної проблематики зумовлена стрімким зростанням обсягів даних та необхідністю їх оперативної інтеграції з різноманітних джерел у сучасних інформаційних системах. Використання зовнішніх сервісів, зокрема REST API, є невід'ємною частиною багатьох цифрових застосунків, однак воно супроводжується ризиками виникнення збоїв, втрати даних чи порушення їхньої цілісності. Проблема полягає в тому, що традиційні методи завантаження даних не завжди здатні забезпечити коректну обробку помилок і відновлення після збоїв, що може призвести до втрати важливої інформації або порушення цілісності даних. Це зумовлює потребу у впровадженні механізмів відмовостійкості, які дають змогу мінімізувати наслідки технічних збоїв та гарантувати стабільність роботи системи. Відповідно, завданням даного дослідження є аналіз сучасних підходів до організації відмовостійкого завантаження даних, визначення їхніх переваг і обмежень, а також окреслення напрямів удосконалення процесу інтеграції даних у сучасних інформаційних системах.

Саме це визначає важливість і практичну значущість розглядуваної теми.

**2. Основний результат.**

**Завантаження даних: концепції та проблеми.** Завантаження даних є фундаментальним процесом в інформаційних системах, що передбачає вилучення, трансформацію та завантаження (ETL — Extraction, Transformation, Loading) інформації з різних джерел у цільову систему чи базу даних [1]. Цей процес забезпечує основу для прийняття рішень на базі даних та ефективного функціонування цифрових застосунків.

У ході завантаження виникає низка проблем, зумовлених властивостями самих даних:

– різноманітність форматів (структуровані, напівструктуровані, неструктуровані) ускладнює стандартизацію та інтеграцію;
– значний обсяг даних вимагає масштабованих рішень. Важливим фактором виступає і швидкість передавання (data velocity), адже сучасні системи орієнтуються на обробку інформації в реальному часі.

Не менш критичною є якість даних, яка визначає їхню точність, узгодженість та повноту.

Сучасні інформаційні системи використовують кілька підходів до завантаження даних:

– **пакетна обробка** передбачає завантаження даних через визначені проміжки часу. Це зручне рішення для систем, де не критична миттєва обробка, однак воно не відповідає потребам додатків реального часу [2];
– **потокова обробка** базується на безперервному отриманні даних (Apache Kafka, Apache Flink). Вона забезпечує високу оперативність, проте створює труднощі для забезпечення відмовостійкості [3];
– **ETL-інструменти** (комерційні або з відкритим кодом) автоматизують процес інтеграції та зменшують складність розробки [4];

– **інтеграція з API** дає змогу отримувати дані безпосередньо зі сторонніх джерел, але потребує особливої уваги до надійності та обробки помилок [5].

*Таблиця 1.*

Порівняння підходів до завантаження даних

| **Критерій** | **Пакетне завантаження (Batch)** | **Потокове завантаження (Streaming)** |
|---|---|---|
| **Обсяг даних** | Підходить для великих масивів, що накопичуються з часом | Оптимальне для безперервних потоків у реальному часі |
| **Частота оновлення** | Дані оновлюються періодично (щогодини, щодня, щотижня) | Дані надходять та обробляються миттєво |
| **Затримка (latency)** | Висока — від хвилин до годин | Низька — від мілісекунд до секунд |
| **Надійність** | Простішою є обробка помилок (повторне завантаження пакету) | Потребує складних механізмів обробки збоїв у режимі онлайн |
| **Архітектура** | Часто монолітна, з ETL-процесами | Ґрунтується на мікросервісах та потокових технологіях (Kafka, Flink) |
| **Ресурси** | Високе навантаження у моменти запуску завантаження | Рівномірний розподіл навантаження у часі |
| **Типові сценарії** | Аналітика, звітність, міграція даних | Онлайн-аналітика, моніторинг, фінансові транзакції |

Надійність системи визначається її здатністю забезпечувати цілісність та доступність даних навіть у разі збоїв. Серед ключових механізмів відмовостійкості можна виокремити:

– **механізм повторних спроб** (retry), що дозволяє автоматично відновлювати процес після збоїв, хоча надмірна кількість спроб може перевантажити систему [6];
– **контрольні точки**, які фіксують прогрес і дають змогу відновити процес без повторної обробки всіх даних;
– **перевірку цілісності даних** до та після завантаження;
– **резервування джерел і процесів**, що створює додаткові шляхи для збереження даних;
– **транзакційні механізми** баз даних, які гарантують узгодженість і можливість відкату;
– **системи моніторингу та оповіщення**, що виявляють аномалії в режимі реального часу.

Традиційний процес ETL передбачає три етапи [4]:

– **вилучення** даних із різних джерел (бази даних, файли, API);
– **трансформація** шляхом очищення, нормалізації та узгодження з цільовою структурою;
– **завантаження** у базу чи сховище даних.

**Концепції відмовостійкості.** Відмовостійкість розглядається як здатність інформаційної системи зберігати працездатність навіть у разі виникнення несправностей різного рівня критичності. Вона є фундаментальним принципом

побудови сучасних застосунків і сервісів, що працюють з великими обсягами даних та інтегрують численні зовнішні джерела. Основні поняття у цьому контексті включають [6]:

– **несправність** — відхилення від нормального стану апаратного чи програмного компонента;
– **відмова** — наслідок несправності, який призводить до зупинки системи, втрати даних або порушення доступності сервісу;
– **резервування** — дублювання критично важливих компонентів (серверів, баз даних, мережевих елементів), що дає змогу уникнути простою в разі відмови основних;
– **стійкість** — здатність системи відновлюватися після збоїв без втрати продуктивності та з мінімальними витратами часу.

Відмовостійкість безпосередньо впливає на якість сервісів, рівень довіри користувачів і конкурентоспроможність системи на ринку.

**Роль API у завантаженні даних.** API є ключовим інструментом для інтеграції даних із зовнішніх джерел, оскільки забезпечує стандартизований механізм взаємодії між системами. Для надійного та безпечного використання API необхідно врахувати такі аспекти [7]:

– **безпечна автентифікація** (API keys, OAuth, JWT), що захищає доступ до даних від несанкціонованого використання;
– **контроль частоти запитів** (rate limiting), який запобігає перевантаженню сервісу й блокує потенційні атаки;
– **ефективна обробка помилок** (HTTP-статуси, детальні повідомлення-відповіді), що дозволяє системі відновлюватися після збоїв без втрати даних та контролювати повторні запити.

Важливим є також використання механізмів логування та моніторингу для виявлення проблем у процесі інтеграції даних.

**Фактори вибору підходу до завантаження.** Вибір оптимальної стратегії завантаження даних залежить від сукупності факторів:

– **обсяг і швидкість даних** — великі потоки потребують потокових технологій (streaming), тоді як невеликі обсяги можуть завантажуватися пакетно;
– **затримки (latency)** — для критичних застосунків важлива мінімальна затримка, що зумовлює використання асинхронних методів;
– **характеристики джерела** — обмеження API, стабільність підключення, формат даних;
– **масштабованість** — здатність системи адаптуватися до зростання навантаження;
– **рівень відмовостійкості** — вимоги до безперервності процесів і допустимого часу відновлення.

Для високонавантажених систем у реальному часі доцільно застосовувати потокову обробку та мікросервісну архітектуру, тоді як у менш критичних сценаріях ефективними залишаються пакетні методи завантаження.

Отже, теоретична база, що охоплює концепції ETL-процесів, принципи відмовостійкості та роль API у завантаженні даних, створює підґрунтя для подальшого вивчення практичних аспектів. Її застосування дозволяє:

– усвідомити необхідність забезпечення безперервності та надійності процесів інтеграції;
– визначити ключові ризики й фактори, що впливають на стабільність роботи системи;
– сформувати критерії вибору технологічних рішень відповідно до потреб конкретного застосунку.

Таким чином, розуміння специфіки завантаження даних та механізмів відмовостійкості виступає основою для розроблення ефективних стратегій інтеграції у сучасних інформаційних системах, які функціонують у динамічному середовищі та повинні гарантувати високий рівень надійності.

**Механізми відмовостійкості.** Відмовостійкість є одним із ключових аспектів завантаження даних, адже саме вона гарантує збереження цілісності та доступності інформації навіть за умов виникнення збоїв чи помилок. У цьому контексті застосовуються різні стратегії, які дають змогу мінімізувати ризики та забезпечити надійність процесів інтеграції.

Одним із базових підходів є механізми повторних спроб [8]. Автоматичне повторення операцій дозволяє компенсувати тимчасові збої, наприклад у мережевих з'єднаннях. Для запобігання надмірному навантаженню на вихідні системи використовуються стратегії експоненціального відставання, які поступово збільшують час очікування між спробами. Водночас важливим обмеженням є встановлення максимальної кількості спроб, що унеможливлює нескінченні цикли повторень і дозволяє вчасно залучати ручне втручання. На рис. 1 показано алгоритм схеми механізмів повторних спроб.

Значну роль у відновленні процесів відіграють контрольні точки [9] та журнали повторного виконання. Збереження стану завантаження у визначені моменти часу забезпечує можливість відновлення з останнього відомого етапу, а журнали повторного виконання дозволяють відтворювати дії системи для мінімізації втрат даних.

Транзакції відіграють ключову роль у забезпеченні узгодженості та надійності даних. Одним із базових підходів у цій сфері є **ACID-транзакції**, що вважаються фундаментальним принципом у сучасних системах керування базами даних. Абревіатура **ACID** розшифровується як *Atomicity* (атомарність), *Consistency* (узгодженість), *Isolation* (ізоляція) та *Durability* (стійкість).

– **Атомарність** гарантує, що транзакція виконується цілком: або всі її операції реалізуються, або жодна з них.
– **Узгодженість** забезпечує збереження інтегральності бази даних, не допускаючи переходу системи у некоректний стан.
– **Ізоляція** запобігає взаємному впливу паралельних транзакцій, дозволяючи уникнути конфліктів і збоїв.
– **Стійкість** означає, що результати завершеної транзакції зберігаються навіть у випадку системних відмов чи збоїв.

Завдяки цим властивостям ACID-транзакції створюють основу для стабільної та послідовної роботи інформаційних систем, що особливо важливо у складних і критичних застосуваннях [10]. Механізми «відкату» дозволяють скасувати зміни, зроблені після останньої успішної транзакції чи контрольної точки.

Крім цього, у практиці широко застосовуються механізми резервування, серед яких гарячий резерв, що забезпечує безперервне перемикання на дублюючу

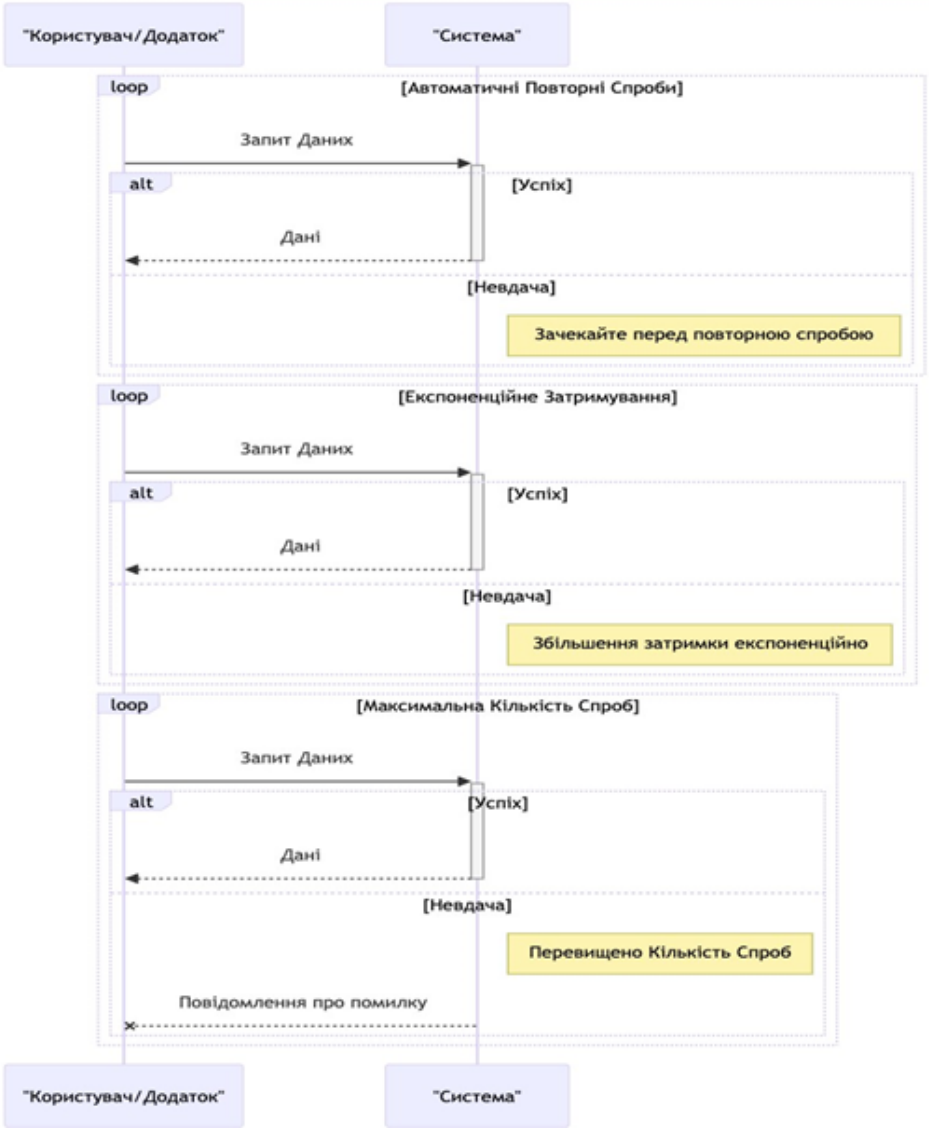


Рис. 1. Механізм повторних спроб.

систему, та реплікація даних, яка зберігає їх копії на різних серверах чи локаціях.

Важливим елементом сучасних рішень є моніторинг і система оповіщень. Вони дозволяють у реальному часі відстежувати роботу системи, виявляти аномалії та потенційні проблеми, ідентифікувати «вузькі» місця, прогнозувати ризики та своєчасно реагувати на збої. Моніторинг сприяє не лише швидкому відновленню працездатності, а й запобіганню майбутнім проблемам за рахунок аналізу трендів і формування рекомендацій щодо оптимізації.

Узагальнюючи, різні механізми відмовостійкості формують комплекс заходів, спрямованих на забезпечення стабільності та безперервності завантаження даних. Їх поєднання дозволяє досягти високого рівня надійності та мінімізувати ризики втрати або пошкодження інформації.

**Реалізація завантаження даних з ЄДЕБО REST API.** Розглянемо практичний підхід до реалізації процесу завантаження даних з Єдиної держав-

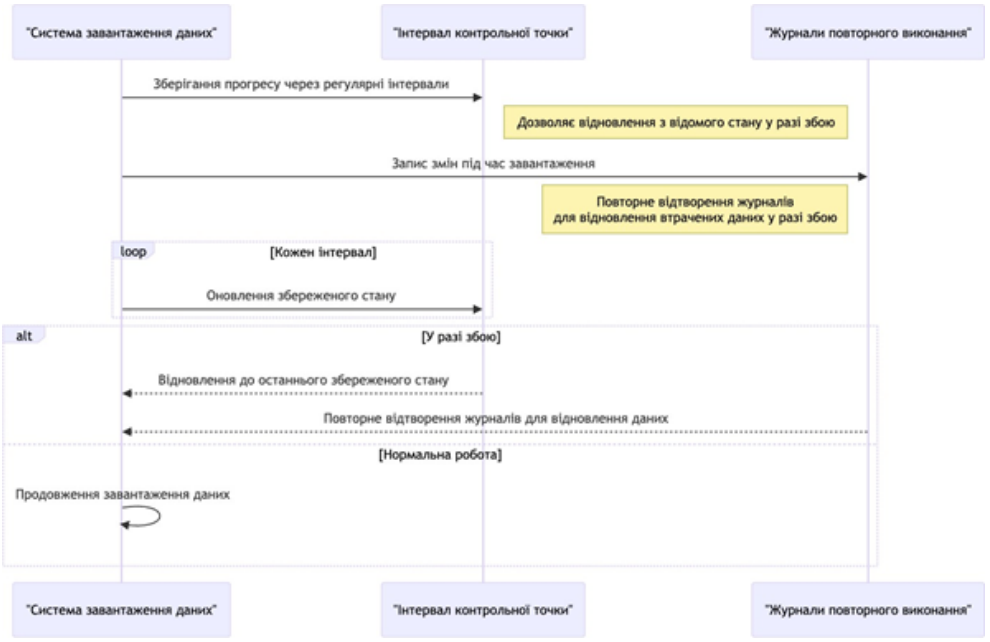


Рис. 2. Механізм журналу повторного виконання.

ної електронної бази з питань освіти (ЄДЕБО) через REST API. Основну увагу приділено забезпеченню відмовостійкості, що є ключовим чинником для збереження цілісності та доступності даних.

ЄДЕБО REST API забезпечує інтеграцію з центральною освітньою базою України, надаючи інструменти для обміну даними щодо студентів, закладів освіти, освітніх програм і пов'язаних процесів. Основні можливості API охоплюють:

- **PhysPerson** — управління персональними даними, документами та вступними заявами абітурієнтів;
- **University** — доступ до інформації про заклади освіти, їхні структурні підрозділи та кадровий склад;
- **Dictionary** — словники та довідкові дані (країни, типи документів, категорії осіб, професії тощо);
- **UniversityStudyProgram** — управління освітніми програмами закладів;
- **UniversitySpecialization** — створення та редагування спеціалізацій;
- **StudentEducations** — робота з академічною історією студентів;
- **PersonPrivilege** — управління даними про осіб із пільгами;
- **EduPrograms** — відомості про програми та спеціалізації;
- **Auth** — автентифікація та управління користувачами.

Таким чином, API виступає центральним інструментом для автоматизованого обміну освітніми даними та підтримки процесів вступу й навчання.

**Реалізація компонента завантаження даних.** Під час інтеграції із ЄДЕБО REST API виникає задача забезпечення відмовостійкого завантаження даних, що є критично важливим для підтримання стабільності та цілісності освітніх інформаційних систем. У процесі проєктування необхідно врахувати потенційні проблеми, зокрема мережеві збої, обмежену доступність серверних

ресурсів, а також неочікувані помилки, що можуть виникати під час обробки отриманих даних.

Для вирішення поставленої задачі необхідно розробити програмний компонент, що відповідає за комунікацію із REST API. Даний компонент реалізує виклики до різних методів API, охоплює можливі сценарії виникнення збоїв, а також здійснює попередню обробку та конвертацію даних у формат, придатний для подальшого використання. Такий підхід дозволяє ізолювати логіку роботи з API від інших частин системи та підвищує загальний рівень відмовостійкості.

Розробку компонента рекомендовано виконувати мовою програмування PHP 8.1, що забезпечить високу продуктивність та сумісність із сучасними серверними середовищами. Для забезпечення інтеграції та подальшого використання розроблений компонент публікується через менеджер пакетів Composer, що є стандартним інструментом керування залежностями в екосистемі PHP та гарантує зручність його встановлення й оновлення. На рис. 3 зображено схему архітектури компонента завантаження даних із ЄДЕБО REST API у вигляді діаграми.

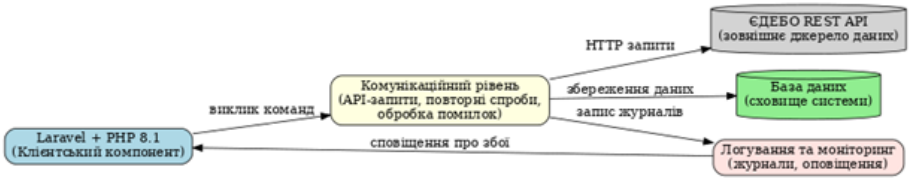


Рис. 3. Архітектура компонента завантаження даних.

Таким чином, створений компонент виконує роль проміжного рівня комунікації із зовнішнім сервісом ЄДЕБО, забезпечуючи захищений та стабільний механізм завантаження даних. Це формує основу для подальшої інтеграції та обробки освітніх відомостей у рамках інформаційних систем, орієнтованих на високий рівень надійності та безперервності роботи.

**Відмовостійка архітектура завантаження даних із ЄДЕБО: виклики та рішення.** У процесі інтеграції з ЄДЕБО REST API постає низка викликів, які безпосередньо впливають на надійність і стабільність завантаження даних. Одним із ключових чинників є мережеві збої, що можуть призвести як до відмови у виконанні запитів, так і до значного зниження швидкості обробки відповідей. Додатковою проблемою виступають непередбачувані помилки на стороні серверів ЄДЕБО, включно з їхньою тимчасовою недоступністю чи відмовою в опрацюванні запитів. Важливим обмеженням є також політика контролю кількості запитів, яка у разі перевищення встановлених лімітів здатна заблокувати доступ до API на певний проміжок часу. Окрему категорію ризиків становлять помилки валідації даних, які можуть виникати як на етапі їхнього надсилання до API, так і під час отримання відповіді, що призводить до некоректного функціонування процесу завантаження.

У цьому контексті особливої ваги набуває поняття відмовостійкості, яке розглядається як здатність системи зберігати працездатність навіть у випадку збоїв, помилок чи інших непередбачуваних ситуацій. Для процесів завантаження даних це означає необхідність впровадження стратегій і механізмів, що забезпечують коректне відновлення після відмов, мінімізацію втрат інформації та

гарантію завершення процесу відповідно до визначених вимог. Саме реалізація відмовостійкості виступає фундаментальною передумовою побудови ефективних інформаційних систем, здатних функціонувати у динамічному середовищі з високими вимогами до безперервності й надійності обробки даних.

Для забезпечення відмовостійкості процесу завантаження даних з ЄДЕБО потрібно врахувати низку критично важливих аспектів архітектурного проєктування. Одним із ключових рішень є використання консольних команд у межах Laravel Framework (зокрема, edbo:load-dictionaries, edbo:load-students), що дозволить реалізувати модульний підхід до організації процесів. Подібна інкапсуляція окремих завдань у вигляді команд відповідає сучасним принципам побудови надійних інформаційних систем, оскільки забезпечує чітке розмежування функціональності, спрощує налагодження та знижує ймовірність поширення помилок між різними компонентами системи.

Важливим елементом архітектури є впровадження механізму автоматизованого планування завантажень, що передбачає виконання процесів у заздалегідь визначений час доби. Такий підхід відповідає концепціям регулярності та передбачуваності обробки даних у розподілених системах (fault-tolerant scheduling) і забезпечує своєчасне оновлення інформації навіть у випадку виникнення тимчасових збоїв.

Окрему увагу потрібно приділяти системі логування, яка функціонує як інструмент забезпечення прозорості процесів та діагностики відмов. У ході виконання кожного завдання формується детальний журнал, що фіксує всі виняткові ситуації, а також метаінформацію про перебіг завантаження. Практика впровадження логування відповідає загальновизнаним підходам до забезпечення надійності інформаційних систем, оскільки створює основу для подальшого аналізу збоїв, формування відновлювальних стратегій та підвищення рівня експлуатаційної готовності системи.

Таким чином, сукупність описаних рішень дозволить реалізувати архітектуру, що відповідає сучасним вимогам до відмовостійких програмних систем і сприяє підвищенню ефективності управління даними у контексті освітніх інформаційних платформ.

Механізми відмовостійкості у процесі завантаження даних з ЄДЕБО орієнтовані насамперед на обробку збоїв, що виникають унаслідок зовнішніх викликів API, оскільки саме ця складова визначає стабільність і цілісність усього процесу обміну інформацією. Виклики до API є критично важливими для підтримання актуальності даних, тому їх надійна обробка становить фундаментальний елемент архітектури системи.

Одним із ключових рішень у цьому контексті є впровадження надійного механізму перехоплення та обробки виняткових ситуацій безпосередньо у командах завантаження даних. Якщо під час виклику API виникає виняток (наприклад, ApiException), він не призводить до аварійного завершення виконання, а обробляється у контрольованій формі. Інформація про збій реєструється у відповідному журналі, а конкретний елемент даних, який не вдалося завантажити, позначається як невдалий. Такий підхід відповідає принципам fault isolation, коли помилка локалізується в межах одного завдання і не впливає на цілісність загального процесу.

Водночас реалізовано механізм повторних спроб (retry policy), що дає змогу

мінімізувати вплив тимчасових збоїв у роботі API. Якщо певний запит завершується невдачею, система виконує повторний виклик визначену кількість разів, перш ніж остаточно зафіксувати його як постійну помилку. Подібна стратегія відповідає концепції підвищення надійності через повторюваність транзакцій, яка широко використовується у високонавантажених інформаційних системах.

Окреме місце у забезпеченні відмовостійкості займає система логування. Для кожного виклику API формується детальний журнал, що включає дані HTTP-запиту та отриманої відповіді, а також метаінформацію про перебіг виконання. Ці журнали становлять важливу основу для подальшого аналізу та діагностики, оскільки дають змогу відстежити першопричини відмов і розробити стратегії для їх запобігання у майбутньому.

На додаток до збоїв, пов'язаних із викликами API, архітектура системи повинна передбачати механізми обробки відмов, що виникають на рівні бази даних. Одним із ключових інструментів забезпечення цілісності даних у цьому контексті є використання транзакцій. Усі команди завантаження даних виконуються в межах транзакційної моделі, що відповідає принципам ACID-парадигми (Atomicity, Consistency, Isolation, Durability) та забезпечує узгодженість даних навіть у випадку виникнення помилок.

Якщо під час виконання завантаження виникає помилка, транзакція автоматично відкочується, унеможливлюючи часткове або некоректне збереження даних. Такий підхід гарантує збереження інваріантів бази даних і дозволяє уникнути порушення її структурної цілісності. У результаті відповідне завдання позначається як неуспішне, а процес завантаження даних повторюється, забезпечуючи повторну спробу їх коректного збереження. Це рішення відповідає сучасним практикам fault-tolerant data loading, коли система відновлюється від збоїв без втрати консистентності та без додаткового втручання з боку адміністратора.

Використання транзакцій у процесі завантаження даних не лише мінімізує ризики пошкодження інформації, а й забезпечує прозорість відновлення після збоїв. Завдяки цьому, навіть у разі повторних помилок на рівні API чи мережевих взаємодій, база даних зберігає узгоджений стан, що є критично важливим для подальшого функціонування системи.

Ефективна система реєстрації журналів та моніторинг є невід'ємними складовими забезпечення відмовостійкості процесу завантаження даних. Реєстрація охоплює не лише фіксацію помилок, але й документування повної історії виконання завдань. Система повинна передбачати збереження детальної інформації про успішні сесії завантаження, що дозволить здійснювати ретроспективний аналіз і відстежувати динаміку роботи компонентів. Такі журнали виступають важливим джерелом даних для оцінки продуктивності та надійності механізмів інтеграції, а також допомагають виявляти приховані закономірності у процесах завантаження.

Поряд із веденням журналу, важливим елементом є моніторинг у реальному часі, спрямований на своєчасне виявлення збоїв і аномалій. У випадку виявлення критичних помилок чи затримок повинна спрацювати система оперативних сповіщень, що інформує відповідальних осіб або операційну команду. Завдяки цьому забезпечується мінімізація часу реагування та знижується ризик тривалого простою або накопичення помилок. Такий підхід підсилює надійність

системи загалом і формує основу для її стабільного функціонування у виробничих умовах.

Відмовостійкість не може розглядатися як одноразове впровадження чи статичне рішення; натомість вона є безперервним процесом удосконалення, який вимагає постійного моніторингу та адаптації. Одним із ключових аспектів цього процесу є регулярне тестування, що передбачає моделювання різноманітних сценаріїв відмов для перевірки коректності роботи захисних механізмів. Такий підхід дозволяє своєчасно виявляти потенційні вразливості та гарантувати, що система здатна зберігати працездатність навіть за умов несподіваних збоїв.

Не менш важливим є використання циклу зворотного зв'язку, що ґрунтується на результатах моніторингу, аналізі журналів і звітів про інциденти. Зібрана інформація виступає основою для прийняття рішень щодо оптимізації процесів та вдосконалення існуючих рішень. Завдяки ітеративному підходу команда розробників має змогу постійно підвищувати рівень відмовостійкості, адаптуючи систему до нових викликів та змін у середовищі її функціонування. Таким чином, відмовостійкість постає як динамічна характеристика інформаційної системи, яка підтримується завдяки циклічному процесу перевірки, аналізу й удосконалення.

**3. Висновки та перспективи подальших досліджень.** Узагальнюючи результати, можна стверджувати, що впровадження механізмів відмовостійкості у процес завантаження даних до інформаційної системи є ключовим чинником забезпечення її надійності та безперервності роботи. Практична реалізація на базі Laravel Framework засвідчує, що ефективне використання консольних команд, механізмів повторних спроб та транзакційної моделі дозволяє не лише гарантувати цілісність даних, але й мінімізувати вплив мережевих збоїв і помилок API.

Детальне логування та моніторинг створюють умови для прозорості процесів і формують основу для зворотного зв'язку, що забезпечує ітеративне вдосконалення архітектури системи. Таким чином, відмовостійкість постає не як разова функціональна можливість, а як постійний процес оптимізації, інтегрований у життєвий цикл системи.

У підсумку, застосовані підходи демонструють, що використання сучасних шаблонів проєктування дає змогу сформувати відмовостійке рішення, здатне до масштабування й адаптації, що є вирішальним для довгострокової стабільності інформаційних систем в умовах динамічного середовища.

### Список використаної літератури

1. TechTarget. (n.d.). 5 Vs of Big Data. Retrieved from `https://www.techtarget.com/searchdatamanagement/definition/5-Vs-of-big-data`
2. Amazon Web Services. (n.d.). What is Batch Processing? Retrieved from `https://aws.amazon.com/what-is/batch-processing`
3. Wikipedia. (n.d.). Stream processing. Retrieved from `https://en.wikipedia.org/wiki/Stream_processing`
4. Skyvia. (n.d.). What Is Batch ETL Processing? The Only Guide You Need. Retrieved from `https://blog.skyvia.com/batch-etl-processing`
5. Skyvia. (n.d.). What is API Integration. Retrieved from `https://blog.skyvia.com/batch-etl-processing`
6. Abrkljac. (n.d.). Building a Resilient Data Infrastructure: Best Practices for Fault-Tolerant Systems. *Medium.* Retrieved from `https://abrkljac.medium.com/building-a-resilient-data-infrastructure-best-`

practices-for-fault-tolerant-systems-562587e136a
7. Acode. (n.d.). What is an API and how does it work? Retrieved from https://acode.com.ua/what-is-api [in Ukrainian].
8. Code Curated. (n.d.). Designing a Retry Mechanism For Reliable Systems. Retrieved from https://codecurated.com/blog/designing-a-retry-mechanism-for-reliable-systems
9. Amazon Web Services. (n.d.). Build an incremental data load solution using AWS DMS checkpoints and database logs. Retrieved from https://aws.amazon.com/blogs/database/build-an-incremental-data-load-solution-using-aws-dms-checkpoints-and-database-logs
10. Wikipedia. (n.d.). ACID. Retrieved from https://en.wikipedia.org/wiki/ACID

**Hlebena M. I., Kornyk O. V., Hlebena O. V., Chubyrka V. V.** Development of strategies for failure-resilient data loading for new-generation information systems.

The article presents a systematic approach to implementing fault-tolerant data loading mechanisms in information systems that integrate with external services, in particular with EDEBO via REST API. Key challenges are analysed, including network failures, request limits, unpredictable external server errors, and data integrity issues. The proposed architecture is based on the use of the Laravel Framework and includes modular organisation of processes through console commands, retry mechanisms for handling temporary errors, a transactional database model to ensure information consistency, and comprehensive logging with subsequent real-time monitoring. Particular attention is paid to the importance of regular testing and the implementation of a feedback cycle, which contributes to the continuous improvement of the system. Practical implementation has proven that the proposed approach not only increases the reliability and stability of loading processes, but also forms the basis for the scalability and adaptability of the information system in the context of growing data processing requirements and dynamic changes in the information environment. Thus, fault tolerance is considered not as a one-time solution, but as a fundamental principle of modern information system development.

**Keywords:** fault tolerance, Big Data, data loading, information systems, REST API, Laravel Framework, transactional model, logging, monitoring, EDEBO.

Одержано 15.09.2025