МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Розробка інформаційної системи онлайн-менеджменту освітніх процесів Digital University UA: модуль "Вступна кампанія"»

Виконав студент групи ІПЗ-21-2
МАСЛАКОВ Дмитро Дмитрович

Керівник роботи:
МОРОЗОВ Андрій Васильович

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
Зав. каф. інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **МАСЛАКОВ Дмитро Дмитрович**

Керівник роботи: **МОРОЗОВ Андрій Васильович**

Тема роботи: **«Розробка інформаційної системи онлайн-менеджменту освітніх процесів Digital University UA: модуль "Вступна кампанія"»** затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: **розробка системи управління освітніми процесами в університеті, що покращить роботу його адміністрації та надасть функціонал для автоматизації та цифровізації рутинних завдань.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Морозов А.В.	27.02.2025р.	27.02.2025р.
2	Морозов А.В.	24.04.2025р.	24.04.2025р.
3	Морозов А.В.	02.05.2025р.	02.05.2025р.

РЕФЕРАТ

Кваліфікаційна робота складається з програмного комплексу для інформаційної системи онлайн-менеджменту освітніх процесів Digital University UA, зокрема модуля «Вступна кампанія» та пояснювальної записки. Пояснювальна записка до випускної роботи містить 77 сторінок, 27 ілюстрацій та 1 таблицю.

Головною метою є створення модулю інформаційної системи, який дозволить цифровізувати увесь етап проведення залікових сесій у закладах вищої освіти та впровадити зручний інтерфейс для використання користувачами різних вікових категорій та посад.

У ході роботи були визначені основні завдання для розробки модулю, проведений аналіз аналогів інформаційної системи та обрано технології, що дозволять реалізувати низку вимог до програмного забезпечення.

КЛЮЧОВІ СЛОВА: BACKEND, ІНФОРМАЦІЙНА СИСТЕМА, ВСТУП, КАМПАНІЯ, PHP, YII2, ЄДЕБО, ІМПОРТ.

ABSTRACT

The qualification work consists of a software package for the online management information system of educational processes Digital University UA, in particular the "Introduction Campaign" module and an explanatory note. The explanatory note to the graduation work contains 77 pages, 27 illustrations and 1 table.

The main goal is to create an information system module that will allow digitalizing the entire stage of conducting credit sessions in higher education institutions and implementing a convenient interface for use by users of different age categories and positions.

During the work, the main tasks for developing the module were identified, an analysis of analogues of the information system was conducted, and technologies were selected that will allow implementing a number of requirements for software.

KEYWORDS: BACKEND, INFORMATION SYSTEM, ENTRANCE, CAMPAIGN, PHP, YII2, EDEBO, IMPORT.

ЗМІСТ

РЕФЕРАТ **3**

ABSTRACT **4**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 7

ВСТУП 8

РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ 9

1.1 Постановка задачі 9

1.2 Аналіз аналогів програмного продукту 11

1.3 Вибір архітектури системи 15

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення 18

Висновки до першого розділу 19

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ОНЛАЙН-МЕНЕДЖМЕНТУ ОСВІТНІХ ПРОЦЕСІВ 21

2.1 Визначення варіантів використання та структури системи 21

2.2 Обгрунтування вибору системи управління базами даних 25

2.3 Розробка бази даних системи 26

2.4 Реалізація функціоналу системи управління навчальним процесом 28

Висновки до другого розділу 44

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ОНЛАЙН-МЕНЕДЖМЕНТУ ОСВІТНІХ ПРОЦЕСІВ 45

3.1 Порядок встановлення та налаштування параметрів додатку 45

3.2 Структура інтерфейсу додатку 47

3.3 Тестування роботи розробленого додатку 57

Висновки до третього розділу 58

ВИСНОВКИ 60

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 62

ДОДАТКИ 65

Додаток А 66

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ЗВО – заклад вищої освіти

CRUD – Create-Read-Update-Delete

СУБД – Система управління базою даних

ЄДЕБО – Єдина державна електронна база з питань освіти

ВСТУП

З розвитком цифрових технологій вища освіта стикається з новими викликами, які суттєво впливають на організацію освітнього процесу. Використання традиційних методів управління, зокрема паперової документації, стає неефективним через складність обробки великих обсягів даних, ризики, пов’язані з людськими помилками, та обмеженість у доступі до інформації поза межами закладу. Крім того, перехід до дистанційного та змішаного навчання потребує технологічних рішень, які дозволять адаптувати освітні процеси до нових умов. У цьому контексті особливо актуальним є впровадження автоматизованих систем управління навчальною діяльністю, що сприятиме її ефективності, прозорості та оптимізації.

Метою дослідження є створення цифрової системи для управління освітнім процесом, яка дозволить автоматизувати адміністративні процедури, полегшити взаємодію між студентами, викладачами та адміністрацією, а також забезпечити зручні механізми моніторингу та аналізу навчальної діяльності.

Досягнення поставленої мети передбачає виконання кількох важливих завдань, таких як аналіз сучасних методів і технологій управління освітнім процесом із виявленням їхніх переваг і недоліків, побудова моделі взаємодії користувачів системи, проєктування архітектури бази даних, та зрештою розробка веб-додатка, що стане ключовим інструментом для роботи із системою.

Об’єктом дослідження є процес розробки програмного забезпечення, яке автоматизує управління освітнім процесом, зокрема планування, організацію, контроль і моніторинг освітньої діяльності у закладах вищої освіти.

Предметом дослідження виступають технології, алгоритми та програмні засоби, що використовуються для створення цифрових систем управління освітніми процесами, а також їхня адаптація до потреб сучасних ЗВО.

РОЗДІЛ 1. АНАЛІЗ ВИМОГ КОРИСТУВАЧА ТА КОНЦЕПТУАЛЬНЕ МОДЕЛЮВАННЯ

1.1. Постановка задачі

Заклади вищої освіти сьогодні потребують сучасних цифрових рішень, які дадуть змогу більш ефективно організовувати навчальний процес. Автоматизація рутинних адміністративних завдань сприяє оптимізації роботи установ, зменшуючи навантаження на персонал. Проте значна кількість університетів усе ще використовує застарілі методи управління документацією, покладаючись на паперові носії або ведучи записи у файлах Word і таблицях Excel [1]. Такий підхід не лише займає багато часу, а й збільшує ймовірність виникнення помилок, що ускладнює пошук необхідної інформації та призводить до дублювання даних.

Децентралізоване зберігання інформації створює чимало труднощів у роботі закладу освіти. Важливі документи, зокрема списки студентів, розклади занять, накази та фінансові звіти, можуть бути розміщені в різних місцях і форматах, що ускладнює доступ до них та оновлення відомостей. Відсутність єдиної системи спричиняє плутанину, неузгодженість дій між підрозділами та значні витрати часу на обробку інформації. Крім того, архівування паперових документів вимагає великого простору, а їхнє зберігання та систематизація потребують значних ресурсів.

Щоб уникнути цих проблем, доцільно запровадити централізовану цифрову платформу для управління різними складовими освітнього процесу. Таке рішення забезпечить зручний та швидкий доступ до актуальних даних, мінімізує ризики, пов’язані з людським фактором, і дозволить автоматизувати ключові аспекти адміністративної роботи. Інтеграція всіх процесів у єдину інформаційну систему сприятиме ефективній взаємодії між структурними підрозділами закладу освіти, зробить управління більш прозорим та сучасним.

Саме тому було визнано необхідним створення нової інформаційної системи «Цифровий університет», а саме модулю «Вступна кампанія». Він буде основним

інструментом для управління контингентом здобувачів вищої освіти у закладі. Ключовою особливістю буде отримання актуальної бази даних вступників, і наразі основним джерелом інформації щодо студентів є Єдина державна електронна база з питань освіти (ЄДЕБО) [25].

Інформаційна система «Цифровий університет» надає доступ до списку всіх здобувачів, які навчалися або навчаються в університеті. Функціонал системи дозволяє сортувати, фільтрувати та здійснювати пошук інформації за різними параметрами.

Завдання системи:

- управління інформацією про здобувачів освіти:
  - відображення базових даних про кожного абітурієнта (ПІБ, дата народження, курс, спеціальність тощо);
  - можливість перегляду картки студента з детальною інформацією;
  - завантаження навчальної картки у форматі Word;
  - отримання доступу до документів вступника;
- модуль імпортування даних:
  - автоматичне імпортування даних з ЄДЕБО кожні 10-20 хвилин;
  - можливість ручного керування процесом імпорту;
  - перегляд історії імпортів за часом та кількістю імпортованих записів;
- модуль "Вступна кампанія":
  - надання доступу структурним підрозділам до інформації про вступників (пільги, потреба у гуртожитку, документи тощо);
  - генерування документів приймальної комісії;
  - перегляд статистики вступної кампанії за освітніми ступенями;
  - формування статистики за освітніми закладами абітурієнтів;
  - перегляд інформації про останні заяви на вступ;
  - обробка помилок та забезпечення стабільності системи.

1.2. Аналіз аналогів програмного продукту

У зв'язку з вищезазначеними викликами в освітньому середовищі багато компаній сфокусувалися на розробці власних цифрових рішень, спрямованих на оптимізацію управління навчальним процесом. Було виявлено кілька схожих сервісів, спеціалізованих на підтримці університетської діяльності.

Moodle [2] – це платформа з відкритим кодом, створена для організації дистанційного навчання. Вона забезпечує широкий функціонал, включаючи розробку навчальних курсів, створення тестувань, оцінювання знань студентів і підтримку інтерактивної комунікації. Головною перевагою Moodle є її гнучкість, яка дозволяє адаптувати платформу до потреб конкретного закладу освіти завдяки різноманітним модулям і плагінам.

Система підтримує інтеграцію з популярними інструментами, такими як Zoom, Google Workspace, Microsoft Teams, що значно спрощує організацію онлайн-занять. Moodle відповідає міжнародним стандартам e-learning, таким як SCORM і xAPI, що забезпечує сумісність із іншими освітніми системами та ресурсами.

Завдяки великій спільноті користувачів, Moodle має розвинену екосистему підтримки, що полегшує процес впровадження та налаштування. Крім того, система є безкоштовною, що робить її доступною навіть для невеликих навчальних закладів (рис. 1.1).

Офіційний сайт: https://moodle.org/

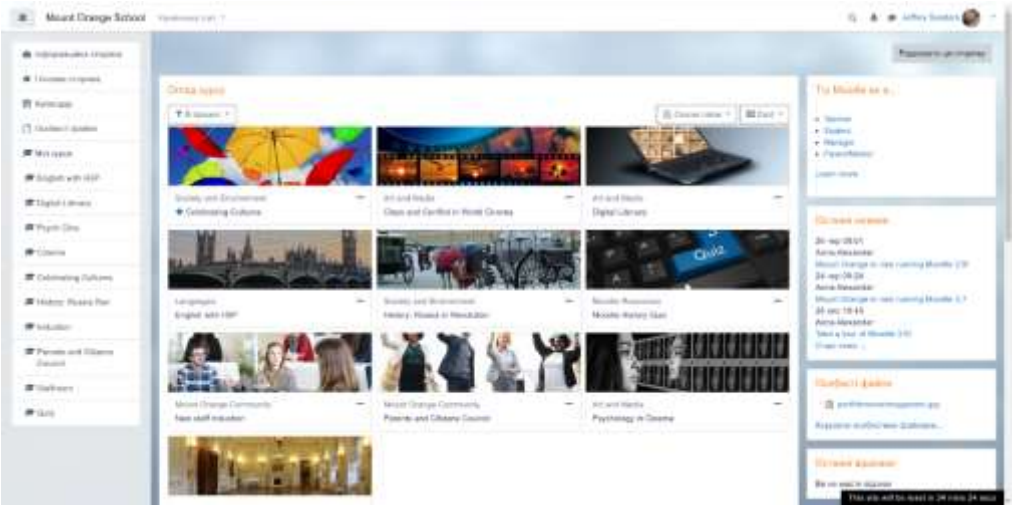

Рисунок 1.1 – Платформа Moodle

Серед ключових переваг цієї платформи можна виділити відкритий код, який дає змогу розробникам створювати власні модулі та розширення, а також наявність активної технічної спільноти, яка допомагає швидко знайти рішення більшості проблем. Moodle відзначається високою гнучкістю: систему можна налаштувати під конкретні потреби окремого навчального закладу. Вона підтримує інтеграцію з популярними сервісами для відеозв'язку та спільної роботи, зокрема Zoom, Google Workspace та Microsoft Teams. Окрім цього, платформа є мультиплатформенною, тобто доступна як на комп'ютерах, так і на мобільних пристроях, що дозволяє забезпечити безперервність навчального процесу.

Попри значні переваги, Moodle має і певні недоліки. Однією з основних складностей є високий рівень технічних знань, необхідних для налаштування й адміністрування платформи. Без додаткової оптимізації серверів складно забезпечити стабільну роботу системи при великій кількості користувачів або обсязі даних. Інтерфейс Moodle може видатися неінтуїтивним для новачків, а повноцінного функціоналу для управління кадрами або фінансами в системі не передбачено, що змушує заклади шукати додаткові рішення для комплексної автоматизації своєї діяльності. Також Moodle потребує регулярних оновлень і технічної підтримки, що може викликати труднощі в умовах обмежених ресурсів.

Таким чином, хоча Moodle є потужним інструментом для організації освітнього процесу, його впровадження може потребувати значних ресурсів та часу. Вибір цієї платформи має базуватися на реальних потребах навчального закладу та наявних технічних можливостях для її підтримки.

Canvas LMS [3] — це сучасна платформа для управління освітнім процесом, розроблена компанією Instructure. Вона використовується університетами, коледжами та іншими освітніми установами для організації онлайн-навчання. Система має хмарну архітектуру, що забезпечує високу продуктивність та доступність із будь-якого пристрою. Canvas LMS підтримує

інтеграцію з безліччю сервісів, таких як Google Workspace, Microsoft 365, Zoom, а також має відкритий API для розширення функціональності.

Canvas LMS пропонує інтуїтивний інтерфейс, який спрощує користування як для студентів, так і для викладачів. Вона підтримує створення курсів, тестів, оцінювання, форумів для обговорень і автоматичне збереження змін у режимі реального часу. Також система має мобільний додаток для швидкого доступу до матеріалів навчання (рис. 1.2).

Офіційний сайт: https://www.instructure.com/canvas

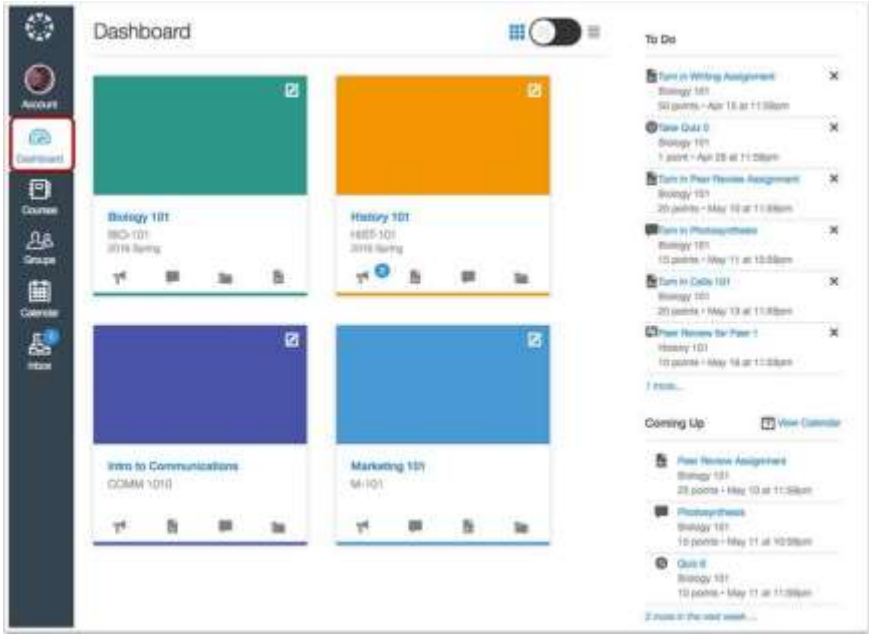

Рисунок 1.2 – Система Canvas LMS

Canvas LMS є хмарною платформою, що дозволяє навчальним закладам зменшити витрати на обслуговування інфраструктури та отримати стабільну технічну підтримку. Система пропонує широкі можливості інтеграції з іншими популярними освітніми сервісами й сторонніми інструментами, а також дозволяє гнучко налаштовувати ролі користувачів і рівні доступу відповідно до потреб конкретного закладу чи викладача.

Разом із тим, одним із основних недоліків цієї платформи є висока вартість ліцензії, що може стати суттєвою перешкодою для окремих навчальних закладів. До того ж, деякі розширені функції доступні лише у платних версіях. Частина інтеграцій потребує додаткового налаштування, що іноді вимагає залучення фахівців. Ще одним обмеженням є відсутність повноцінної підтримки

локального розгортання на серверах навчального закладу, що може бути критично важливим у випадках, коли установи мають підвищені вимоги до захисту персональних даних або роботи без підключення до інтернету.

Отже, Canvas LMS є потужним інструментом для організації дистанційного навчання з безліччю сучасних можливостей, але її впровадження слід ретельно зважувати, враховуючи фінансові, технічні й безпекові аспекти діяльності конкретного освітнього закладу.

OpenEduCat [4] — це модульна система управління освітнім процесом із відкритим кодом, створена на базі ERP-платформи Odoo. Вона орієнтована на університети, школи та інші навчальні заклади, надаючи комплексні інструменти для адміністрування студентів, курсів, фінансових операцій та інших аспектів освітнього процесу.

Система дозволяє вести електронний журнал, формувати розклади занять, автоматизувати управління бібліотекою, фінансами та інфраструктурою університету. OpenEduCat також підтримує аналітику та звітність, що допомагає адміністрації приймати обґрунтовані рішення. Оскільки система є частиною Odoo, вона легко інтегрується з модулями для бухгалтерського обліку, CRM та інших бізнес-процесів (рис. 1.3).

Офіційний сайт: https://www.openeducat.org/

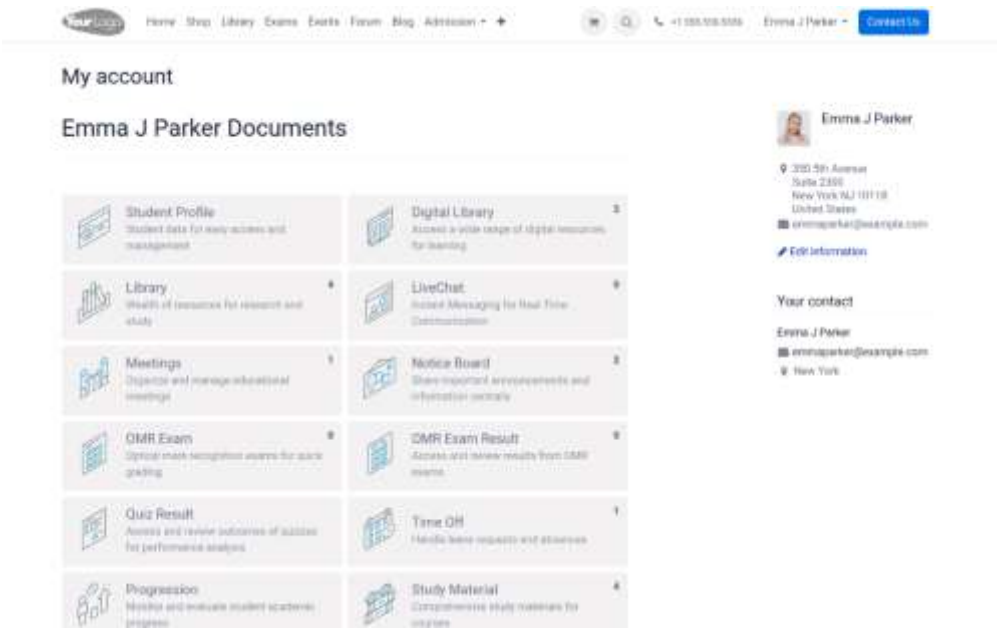

Рисунок 1.3 – Система OpenEduCat

Серед ключових переваг OpenEduCat — її відкритість і можливість адаптації під індивідуальні потреби навчального закладу, гнучке налаштування системи як у хмарному середовищі, так і на локальних серверах, а також автоматизація багатьох адміністративних процесів. Система також вирізняється потужним аналітичним інструментарієм, що дозволяє ефективно відстежувати динаміку освітнього процесу та оперативно реагувати на зміни.

Разом із тим, використання OpenEduCat може бути складним для користувачів без належної технічної підготовки. Першопочаткове налаштування системи вимагає спеціалізованих знань, а деякі функціональні модулі доступні лише у платній версії, що може обмежити її застосування у закладах з обмеженим бюджетом. Додатковою перешкодою є недостатня кількість детальної документації та обмежена підтримка з боку розробників, що може ускладнити інтеграцію з іншими цифровими системами.

Загалом OpenEduCat — це потужне рішення для цифровізації управління навчальним закладом, але його ефективне впровадження потребує технічної підготовки, чіткого планування та відповідного ресурсного забезпечення.

1.3. Вибір архітектури системи

У сучасному світі, де інформаційні технології розвиваються небаченими темпами, розробники програмного забезпечення постійно шукають оптимальні методи побудови архітектури своїх систем [13]. Зростаюча складність програмних продуктів спонукала фахівців зосередитися на питаннях структурної організації коду, що дозволяє підвищити ефективність розробки та підтримки. Обрана архітектурна модель відіграє ключову роль у забезпеченні стабільності та масштабованості програмних рішень, а також значною мірою визначає зручність подальшого розширення функціоналу та модернізації системи.

Однією з традиційних архітектур є монолітна структура [14], яка передбачає об'єднання всіх компонентів програмного продукту в єдину, нерозривну систему. Сам термін «моноліт» має історичне коріння та символізує цілісність і нерозривність структури. У програмуванні це означає, що всі частини

системи, включаючи серверну логіку, базу даних та користувацький інтерфейс, функціонують як єдине ціле. Такий підхід дозволяє швидко розробляти та розгортати програмне забезпечення, що особливо вигідно для невеликих команд, стартапів або проектів, які не передбачають частих змін і масштабування.

Перевагами монолітної архітектури є її простота у впровадженні та розгортанні, оскільки всі компоненти інтегровані в єдине середовище. Завдяки цьому процеси розробки та налагодження стають значно зрозумілішими, а сама система легше піддається моніторингу та тестуванню. Крім того, монолітний підхід сприяє підвищенню продуктивності, оскільки всі елементи програми тісно взаємопов'язані і взаємодіють без затримок, що притаманні розподіленим системам. Це також забезпечує єдиний контроль над усіма частинами програмного продукту, що спрощує управління версіями та оновленнями.

Однак із часом монолітні програми можуть стати занадто громіздкими, що значно ускладнює їх подальше оновлення та супровід. Зі зростанням кількості функцій і коду підвищується ризик конфліктів між різними модулями, а внесення змін в одну частину програми може викликати небажані наслідки в інших компонентах. Це робить розробку нових можливостей складнішою, а масштабування системи – проблематичним. Окрім цього, використання монолітного підходу може призвести до зниження швидкості розробки, оскільки навіть незначні зміни потребують ретельного тестування всього продукту.

На зміну традиційному моноліту прийшла мікросервісна архітектура, яка передбачає розбиття програми на окремі незалежні сервіси, кожен з яких відповідає за конкретну функцію. Мікросервіси взаємодіють між собою через API [20], що дозволяє значно підвищити гнучкість і масштабованість програмного продукту. Внаслідок цього розробники можуть працювати над різними частинами системи паралельно, що прискорює процес створення нових функцій і знижує ризик конфліктів у кодовій базі.

Однією з головних переваг мікросервісної архітектури є можливість незалежного масштабування кожного сервісу, що дозволяє адаптувати систему до змін у навантаженні. Це особливо актуально для високонавантажених

програм, які обробляють великі обсяги даних або мають значну кількість одночасних користувачів. Крім того, окремі сервіси можна оновлювати та замінювати без впливу на всю систему, що спрощує її підтримку та модернізацію. Додатковою перевагою є можливість використання різних технологій і мов програмування для кожного окремого мікросервісу, що дозволяє вибирати найефективніші інструменти для кожної задачі.

Проте, попри численні переваги, мікросервісна архітектура має й певні виклики. Вона потребує ретельного планування взаємодії між сервісами, налаштування мережевих з'єднань і систем моніторингу. Крім того, розподіленість компонентів призводить до збільшення затримок у комунікаціях, що може негативно впливати на продуктивність системи. Впровадження такої архітектури вимагає більше ресурсів та часу, що може бути неприйнятним для малих проектів із обмеженим бюджетом. Також необхідно враховувати складнощі, пов'язані з управлінням розподіленими транзакціями та забезпеченням безпеки між сервісами.

Вибір між монолітною та мікросервісною архітектурою залежить від специфіки проекту, його масштабу та майбутніх потреб у розвитку. Якщо необхідно швидко запустити програмний продукт і немає потреби в масштабуванні, моноліт може стати оптимальним рішенням. Водночас, якщо йдеться про складну систему, яка повинна бути гнучкою та легко розширюваною, мікросервіси можуть виявитися більш вигідним варіантом.

З огляду на всі особливості та вимоги до розроблюваної системи, було прийнято рішення використати мікросервісну архітектуру. Хоча цей підхід і вимагає більше часу на реалізацію, він забезпечує необхідну гнучкість та можливість подальшого розвитку без ризику створення надмірно складної кодової бази. Таким чином, застосування мікросервісів дозволить уникнути багатьох проблем у майбутньому та гарантуватиме стабільну й ефективну роботу програмного продукту. Додатково, використання мікросервісного підходу сприятиме підвищенню рівня автоматизації процесів, що позитивно вплине на загальну продуктивність і зручність підтримки системи.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Надалі було необхідно визначитися із інструментальними засобами для розробки майбутнього ПЗ. Одним із варіантів було використати мову PHP [7], яка успішно зарекомендувала себе у світі веб-розробки і є чудовим та зваженим рішенням – величезна кількість навчальних матеріалів, ком'юніті, а також існуючі фреймворки доводять її доцільність для нашого випадку. У якості основного фреймворку для бекенду обрано Yii2 [5], що забезпечує гнучкість у розробці та легкість інтеграції з іншими компонентами системи. У якості системи управління базами даних використовується MySQL[15], що гарантує високу продуктивність та надійність. Окремо буде розроблено мікросервіс для імпорту даних з ЄДЕБО на платформі Node.js [8], що дозволить забезпечити швидку обробку та інтеграцію даних.

Вимоги до сервера (Мікросервіс Node.js):

- Процесор:
  - не менше 4 ядер;
  - тактова частота не менше 3.00 GHz;
  - підтримка багатопоточності;
  - архітектура x86_64;
- Оперативна пам'ять (RAM):
  - не менше 32 GB;
  - бажано — підтримка ECC (для підвищення надійності в умовах інтенсивного логування та передачі даних);
- Дискова підсистема:
  - SSD накопичувач обсягом не менше 100 GB;
  - зарезервований простір для логів, тимчасових файлів і черг повідомлень (RabbitMQ / Redis);
- Операційна система:
  - LINUX (debian-based або Alpine Linux);
  - підтримка systemd або supervisord для автозапуску процесів;

    - можливість запуску у Docker-контейнерах або на віртуальній машині (KVM);
- Мережа:
    - гарантована пропускна здатність не менше 100 Mbps;
    - стабільна внутрішня комунікація з основним сервером (ping < 1 ms);
    - відкриті порти для REST API та WebSocket (80, 443, 3000+);
- Інше:
    - резервне копіювання конфігурацій та логів щонайменше раз на добу;
    - підтримка автоматичного розгортання через CI/CD (GitLab CI, GitHub Actions або подібні).

Висновки до першого розділу

У першому розділі було проведено детальний аналіз поставленого завдання, що дозволило чітко визначити мету та ключові цілі кваліфікаційної роботи. Особливу увагу приділено формуванню коректної постановки проблеми, адже саме це є відправною точкою для побудови ефективної та функціональної програмної системи. Визначення основних функціональних вимог до розроблюваного продукту стало важливим етапом у процесі планування, адже воно сприяло встановленню пріоритетів та розробці стратегічного підходу до реалізації.

Проведене дослідження аналогічних програмних рішень дало змогу глибше зрозуміти поточний стан ринку, оцінити наявні рішення та виділити їхні сильні й слабкі сторони. Це стало основою для обґрунтування шляхів удосконалення, які дозволять запропонованому продукту бути конкурентоспроможним та відповідати сучасним вимогам користувачів. У результаті аналізу було сформульовано рекомендації щодо покращення функціональних можливостей, підвищення ефективності роботи системи та забезпечення її адаптивності до майбутніх змін.

На основі отриманих висновків обрано оптимальну архітектуру, що забезпечує високу продуктивність, надійність та масштабованість системи. Обґрунтовано вимоги до апаратного забезпечення, необхідного для стабільної роботи системи. Було визначено оптимальні характеристики серверного обладнання, що забезпечать безперебійну роботу в умовах реальних навантажень та гарантують належний рівень продуктивності.

Таким чином, виконаний аналіз та прийняті рішення формують надійну концептуальну базу для наступних етапів розробки. Визначені напрямки вдосконалення, обрані інструменти та розроблена загальна архітектура системи забезпечують ефективний старт для подальшої реалізації програмного продукту, сприяючи його відповідності сучасним технологічним вимогам та очікуванням користувачів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ОНЛАЙН-МЕНЕДЖМЕНТУ ОСВІТНІХ ПРОЦЕСІВ

2.1. Визначення варіантів використання та структури системи

Для забезпечення надійного функціонування модуля «Вступна кампанія» в межах інформаційної системи цифрового університету сформульовано низку вимог до розробки [28]. Вони охоплюють як функціональні, так і нефункціональні характеристики, що спрямовані на підтримку високої ефективності, безпеки, масштабованості та сумісності з іншими підсистемами.

Бізнес вимоги

1. **Основні цілі:**

   Модуль створюється для автоматизації та цифровізації процесів вступної кампанії в закладі вищої освіти. Основною метою є централізоване управління інформацією про вступників, інтеграція з Єдиною державною електронною базою з питань освіти (ЄДЕБО), а також формування актуальної статистики, аналітики та звітності за результатами вступу.

2. **Представлення проєкту:**

   Система реалізується у вигляді веб-застосунку з монолітною архітектурою. Серверна частина побудована на основі PHP-фреймворку Yii2, що забезпечує гнучкість, надійність та можливість подальшого масштабування, і виконує рендерінг темплейтів сторінок для користувача. Вся інформація зберігається в централізованій базі даних, що дозволяє уникнути дублювання даних та забезпечити актуальність інформації в режимі реального часу. Однак для імпортування даних з ЄДЕБО використовується проміжний мікросервіс на Node.js, який має підключення до бази даних проекту, а також має власну базу для збереження даних перед конвертацією.

Вимоги адміністрації:

Однією з основних потреб є можливість перегляду заяв вступників з урахуванням фільтрації та сортування за різними критеріями, зокрема за спеціальністю, датою подачі чи поточним статусом. Окрім цього, важливим є доступ до повного комплекту документів кожного абітурієнта, а також інформації про його попередню історію навчання, що забезпечує належний рівень перевірки даних і прийняття зважених рішень щодо зарахування. Система має підтримувати можливість експорту зібраної інформації у вигляді звітів у форматах Excel, PDF або DOCX, що значно полегшує процес підготовки офіційної документації. Одним із критичних аспектів є наявність синхронізації з Єдиною державною електронною базою з питань освіти (ЄДЕБО), адже отримання актуальних даних із державної системи та забезпечення двостороннього обміну є необхідними для відповідності нормативним вимогам.

Важливою функцією також є генерація статистичних даних, таких як кількість поданих заяв на певну спеціальність, динаміка надходження заяв та аналіз успішності вступників, що дає змогу здійснювати подальше планування, а також відстеження реєстрацій на вступні іспити. Завершальним етапом є перегляд інформації про зарахованих осіб з можливістю створення зведених звітів, які узагальнюють підсумки вступної кампанії та слугують підґрунтям для подальших управлінських дій.

Вимоги власників:

Власники системи очікують наявність гнучких налаштувань ключових параметрів, можливість керування доступом до модулів відповідно до ролей користувачів, зокрема оператора, адміністратора та членів приймальної комісії, фіксацію всіх дій персоналу та вступників у системному журналі, а також регулярне створення резервних копій бази даних для забезпечення збереження інформації.

Характеристика об’єкта комп’ютеризації.

Об’єктом комп’ютеризації є процес вступу до закладу вищої освіти, що охоплює етапи подачі заяв, перевірки, конкурсного відбору та зарахування.

Основними користувачами модуля будуть співробітники приймальної комісії, адміністрація, технічні оператори, а також вступники, що взаємодіють із системою в особистому кабінеті. Модуль повинен бути інтуїтивно зрозумілим, зручним як для досвідчених користувачів, так і для новачків. Доступ до системи здійснюється через браузер, що усуває потребу у встановленні додаткового ПЗ.

Функціональні вимоги:

Система повинна забезпечувати авторизацію користувачів із подальшим призначенням відповідних ролей. Вона має підтримувати імпорт актуальних даних про подані заяви, автоматичне формування списків осіб, рекомендованих до зарахування, а також створення офіційної звітності — зведених таблиць, аналітики набору та статусів заяв. Повинен бути доступний гнучкий пошук заяв за прізвищем, номером документа, спеціальністю, факультетом та іншими критеріями. Також передбачено можливість оновлення інформації про вступників у ЄДЕБО, управління даними щодо реєстрацій на вступні випробування та отримання статистики щодо участі в ЄВІ, ЄФВВ і ЄВВ за попередні роки.

Нефункціональні вимоги:

1. **Сприйняття**

   Система повинна бути достатньо простою для повноцінного освоєння оператором протягом одного навчального заняття тривалістю до 60 хвилин. Відгук інтерфейсу на основні дії не повинен перевищувати 2 секунд, тоді як для складних аналітичних запитів допустима затримка до 10 секунд. Усі форми мають містити підказки до полів, що забезпечить зручність та зменшить кількість помилок під час введення даних.

2. **Надійність**

   Система повинна забезпечувати доступність не менше ніж 96% часу протягом доби. Відновлення її роботи після збою має відбуватися за час, що не перевищує 15 хвилин. Перед кожним експортом або синхронізацією необхідно здійснювати перевірку цілісності даних для гарантування їхньої коректності та надійності.

3. **Продуктивність**

Система має підтримувати одночасну роботу понад 200 користувачів без зниження продуктивності. Вона повинна ефективно обробляти великі обсяги даних під час фільтрації, сортування та експорту. Крім того, інтерфейс повинен бути адаптивним і коректно відображатися на різних пристроях, зокрема на смартфонах і планшетах.

4. **Можливість експлуатації**

Система повинна мати модульну структуру, що дозволяє легко розширювати її функціональність для подальших вступних кампаній. Вона має підтримувати API для інтеграції з ЄДЕБО та іншими освітніми реєстрами. Також обов'язковою є наявність журналу подій, системи сповіщень і механізмів резервного копіювання для забезпечення безперебійної роботи та контролю.

Після дослідження усіх необхідних характеристик майбутнього програмного забезпечення та виокремлення ключових функцій, що потребують актори системи, було створено діаграму використання (рис. 2.1):

Рисунок 2.1 - Діаграма використання системи

Головним користувачем майбутнього додатку є адміністрація – група викладачів та працівників університету, яка відповідно до повноважень наділена різними ролями, які не є статичними. Вебпортал надає можливість адміністраторам самостійно створювати нові ролі та наділяти їх необхідними привілеями, таким чином дозволяючи за потреби масштабувати зону відповідальності кожної з них. Модуль «Вступна кампанія» охоплює функціонал, що безпосередньо стосується її проведення – а саме роботу із заявами абітурієнтів, формування звітів та можливості статистичного аналізу даних. Актор «Власник», що охоплює компанію, яка надає послуги з використання продукту, повинна своєчасно реагувати на запити від користувача, і для ефективної підтримки потребує можливості обмежувати доступ на період проведення технічних робіт, логувати дії персоналу на випадок виявлення помилок та у такому разі налаштовувати портал у реальному часі без прямого втручання у код.

2.2. Обґрунтування вибору системи управління базами даних

У процесі реалізації інформаційної системи було обґрунтовано доцільність використання системи керування базами даних MySQL [16], а для адміністрування – інструмента phpMyAdmin, який забезпечує інтуїтивно зрозуміле середовище для взаємодії з базою даних через вебінтерфейс. Зазначений вибір зумовлений низкою технологічних та практичних переваг, що вирізняють MySQL серед альтернативних рішень, зокрема в контексті веб-орієнтованих застосунків.

MySQL характеризується високою продуктивністю, надійністю та гнучкістю масштабування, що дозволяє ефективно обробляти як невеликі обсяги даних, так і великі, розподілені інформаційні потоки. Ці властивості є критично важливими при створенні систем, що працюють з навчальними даними, де передбачається постійне оновлення й зберігання значної кількості записів, пов’язаних зі студентами, результатами оцінювання, розкладом занять тощо.

Крім того, вибір MySQL як основної СУБД зумовлений її тісною інтеграцією з мовою PHP [17], яка була обрана для розробки серверної частини програмного продукту. Така технологічна сумісність значно полегшує реалізацію логіки обміну даними між прикладною частиною системи та її сховищем, мінімізуючи часові витрати на реалізацію базових CRUD-операцій (Create, Read, Update, Delete).

Застосування phpMyAdmin як інструмента адміністрування дозволило оперативно виконувати конфігурацію структури бази даних, включно з створенням таблиць, налаштуванням атрибутів, встановленням зв'язків між об'єктами та контролем над правами доступу користувачів. Завдяки підтримці гнучких механізмів безпеки, MySQL дає змогу встановлювати рівні авторизації, забезпечуючи тим самим захист системи від несанкціонованого втручання та несанкціонованого доступу до конфіденційної інформації.

У рамках етапу проєктування бази даних було побудовано діаграму класів, яка формалізує основні логічні одиниці предметної області та демонструє між ними ключові взаємозв'язки:

2.3. Розробка бази даних системи

Головною особливістю модуля є процес отримання даних про абітурієнтів. Обрання місця для подальшого навчання, розставлення пріоритетності та формування рейтингів на вступ здійснюється за допомогою Українського центру оцінювання якості освіти, що вносить зміни до Єдиної державної електронної бази з питань освіти, де відбувається їх подальше зберігання. Університети можуть надіслати запит для отримання доступу до інформації про вступників, і у разі схвалення отримають персональний логін, пароль, APP-ключ, а також ідентифікатор університету. В подальшому вищеперераховані дані будуть використовуватися під час здійснення запитів на сервер ЄДЕБО.

Однак величезна кількість інформації, яку потрібно завантажити та правильно імпортувати у БД проекту для подальшого використання, є

ресурсозатратним завданням і вимагає багато часу, тому потрібно було ретельно спланувати оптимізований спосіб їх добування.

Діаграма послідовності зображена на рисунку 2.2:

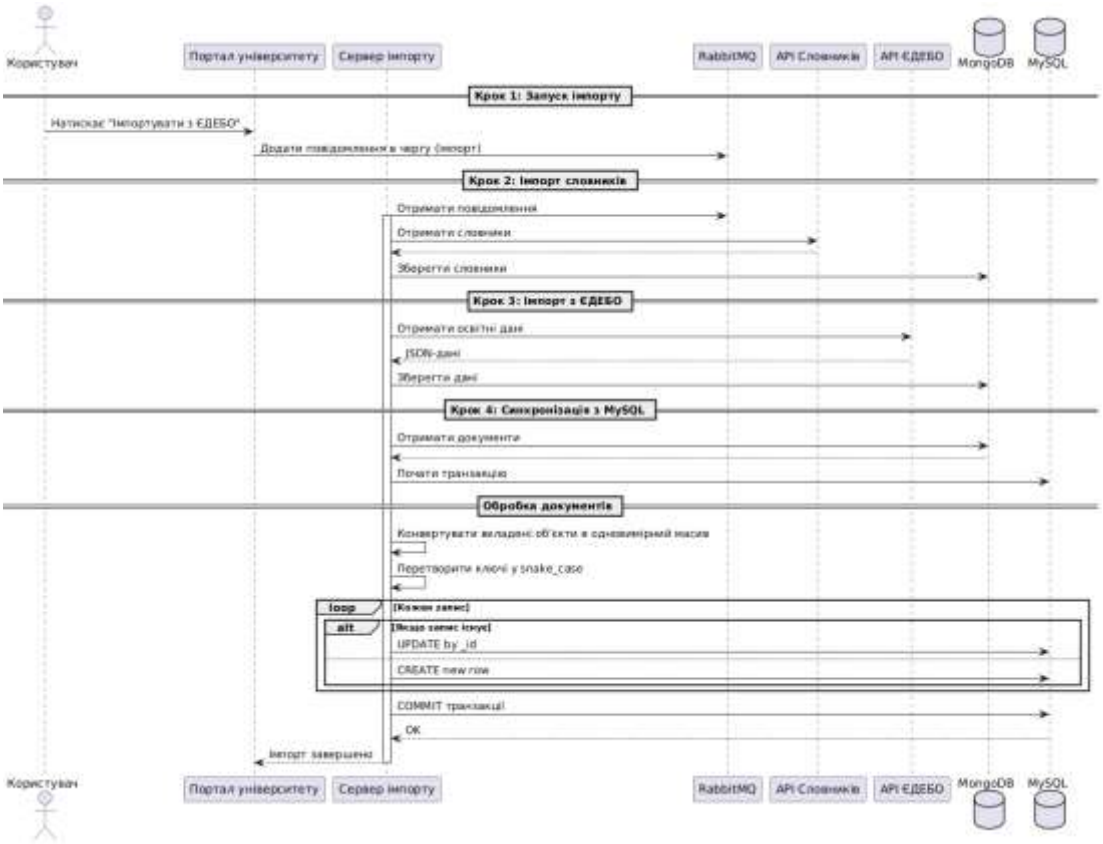

Рисунок 2.2 - Діаграма послідовності імпортування даних з ЄДЕБО

Процес імпорту даних на порталі університету починається з ініціації користувачем запиту на завантаження інформації з Єдиної державної електронної бази з питань освіти (ЄДЕБО). Запит відправляється у спеціальну чергу повідомлень RabbitMQ, що забезпечує асинхронну обробку та надійність виконання задач.

Перед безпосереднім імпортом основних даних з ЄДЕБО відбувається попереднє завантаження словників, які містять необхідні довідкові дані. Далі алгоритм [12] звертається до API ЄДЕБО, отримує необхідні освітні дані у вигляді JSON-документів [30] та зберігає їх у проміжній базі MongoDB. Після повного отримання та збереження даних запускається процес синхронізації з реляційною базою MySQL, яка використовується для подальшої безпосередньої роботи через портал університету. На цьому етапі дані проходять дві важливі

трансформації: вкладені об’єкти перетворюються у плоскі одновимірні масиви, а назви полів змінюються з формату camelCase у snake_case, що відповідає стандартам бази MySQL.

Для кожного документа виконується перевірка наявності запису в MySQL: якщо запис відсутній — він створюється, якщо існує — оновлюється. Всі операції відбуваються у межах транзакції, що гарантує цілісність даних і можливість відкотити зміни у випадку помилки.

Таким чином, описаний алгоритм забезпечує надійний, послідовний та ефективний процес перенесення даних з зовнішнього джерела через MongoDB [18] у кінцеву реляційну базу, що значно спрощує подальшу роботу з інформацією на порталі університету.

2.4. Реалізація функціоналу системи управління навчальним процесом

Розглянемо код побудованого програмного забезпечення. Почнемо огляд із запуску процедури імпорту даних:

```
public function actionImport(string $importName = '', string $collectionName = ''): int
{
   $message = json_encode([
       'importName' => $importName,
       'collectionName' => $collectionName,
   ]);

   $this->amqpManager->sendMessage($message);

   return ExitCode::OK;
}
```

Користувач при натисканні на кнопку імпортування із заданим іменем надсилає запит із інформацією про тип імпорту та тип колекції, що має бути імпортована. Ця інформація потрапляє до черги через RabbitMQ, і для зв’язку з ним використовується створений клас AMQPService:

```
/**
* @throws Exception
*/
public function getConnection(): AMQPStreamConnection
{
   $config = Yii::$app->components['rabbitmq'];

   return new AMQPStreamConnection(
```

```
        $config['RABBITMQ_IPV4'],
        $config['RABBITMQ_PORT'],
        $config['RABBITMQ_USER'],
        $config['RABBITMQ_PASSWORD'],
        $config['vhost']
    );
}
```

При ініціалізації відбувається підтягування конфігурації з’єднання, а далі при надсиланні імпорту до черги буде викликатися наступний метод:

```
public function sendMessage($message): void
{
    $connection = $this->getConnection();
    $channel = $connection->channel();
    $queueName = "edebo";
    try {
        $channel->queue_declare($queueName,   false,   false,   false,
false);

        $msg = new AMQPMessage($message);
        $channel->basic_publish($msg, "", $queueName);

        echo " [x] Sent '$message'\n";
    } catch (Exception $e) {
        echo " [x] Error sending message: {$e->getMessage()}\n";
    } finally {
        $channel->close();
        $connection->close();
    }
}
```

Алгоритм досить простий - відкривається нове з’єднання з каналом, декларується черга та додається меседж із назвою імпорту. Далі з’єднання знову закривається.

Читання черги здійснює мікросервіс на Node.js, де створений подібний клас AMQPService, однак його місія - це читати чергу та видаляти повідомлення, як завдання було взято на виконання:

```
async consumeMessage () {
 try {
   await this.ensureChannel();
   await this.channel.assertQueue("edebo", { durable: false });

   this.channel.consume("edebo", async (msg) => {
     if (msg) {
       const messageContent = JSON.parse(msg.content.toString());
       await    this.processMessage(msg,    messageContent.importName,
messageContent.collectionName, messageContent.params);
     }
```

```
    }, { noAck: false });
  } catch (error) {
    console.error("Error consuming messages:", error.message);
    throw error;
  }
}
```

Метод читає чергу, і виконує запуск ініціалізації відповідного імпорту із вказаним у черзі іменем:

```
async processMessage (msg, importName, collectionName, params) {
 try {
   this.channel.ack(msg);
   await this.close();
   await importFunction(importName, params);
   await syncFunction(importName, collectionName);

 } catch (error) {
   console.error("Response error:", error.message);
 } finally {
   await this.consumeMessage();
 }
}
```

Функція processMessage звертається до функції, що викликає за іменем необхідний імпорт, використовуючи систему колбеків, що у свою чергу завантажує дані з сервера ЄДЕБО у проміжну колекцію MongoDB. Далі імпортовані дані будуть конвертуватися в дані MySQL за допомогою методу синхронізації.

Розглянемо процес імпорту - нам доступні декілька видів імпортування, кожен з яких виконує певний набір операцій - наприклад, один з них займається отриманням інформації про університет, інший завантажує всі дисципліни тощо. Ми сфокусуємо увагу на процесі скачування даних студентів, які є головним джерелом даних вступників.

Існує два види імпортування студентів:

```
ActiveStudentsImportService:
ImportStudentsService.importActiveStudents,
AllStudentsImportService: ImportStudentsService.importAllStudents,
```

Перший виконує завантаження даних здобувачів освіти, які ще здійснюють навчання, в той час як інший скачує усіх студентів університету за всі роки. Обидва діють за однаковим принципом, але мають різний вхідний булевий параметр:

```
static async importActiveStudents() {
   sendStatusUpdate(null,   'ActiveStudentsImportService',   'Початок
імпортування студентів (Активні)...', null);
   await ImportStudentsService.importStudents(true);
}

static async importAllStudents() {
   sendStatusUpdate(null,     'AllStudentsImportService',     'Початок
імпортування студентів (Усі)...', null);
   await ImportStudentsService.importStudents(false);
}
```

Перед імпортуванням необхідно впевнитися в тому, що ми маємо переліки освітніх ступенів:

```
const collection = await
mongoose.connection.db.collection('edebo_dictionary_std_qualification
groups');
```

Через це, процес імпорт словників у методі ImportDictionaries() передує усім іншим типам імпортування:

```
console.log('Початок імпорту словників...');
const                        staticProperties                        =
Object.keys(EdeboDictionaryConstants).filter(key     =>     key     !==
'DictionaryNames');
let index = 1;
let count = staticProperties.length;
let currentDictionary = null;
let intervalId = setInterval(() => {
   if (currentDictionary) {
       let percentComplete = (index / count) * 100;
       sendStatusUpdate(Math.floor(percentComplete),
'DictionariesImport', `${currentDictionary}`, `${index} / ${count}`);
   }
}, 1000);
```

При запуску методу відбувається ініціалізація логування, яке буде записувати відсоткове значення прогресу виконання імпортування. Далі за допомогою циклу по черзі викликається завантаження кожного із словників, порядок яких був зарання заданий у масиві:

```
for (let dictionaryName of staticProperties) {
let displayName =
EdeboDictionaryConstants.DictionaryNames[dictionaryName] ||
dictionaryName;
currentDictionary = displayName;
console.log(`Імпортування словника '${displayName}' (${index} /
${count})...`);
await DictionariesImportService.ImportDictionary(dictionaryName);
let percentComplete = (index / count) * 100;
```

```
sendStatusUpdate(Math.floor(percentComplete), 'DictionariesImport',
`${displayName}`, `${index} / ${count}`, false);
index++;
}
```

Сам метод імпортування словників універсальний і приймає їх назви, додаючи префікс “edebo_dictionary_”, що є характерним для усіх словників. далі відбувається завантаження отриманих даних з API ЄДЕБО, які зберігаються у базі даних MongoDB. При виникненні помилки імпорт не завершиться, але відповідне повідомлення буде виведене в лог:

```
static async ImportDictionary(lookupKey) {
   try {
console.log(`Імпорт словника '${lookupKey}' розпочато...`);
let rows = await DictionariesImportService.dictionaryGet(lookupKey);
let collection_name = 'edebo_dictionary_' + lookupKey;
await    MongoDBService.enterDataIntoMongoDB(rows,    collection_name,
'id');
       console.log(`Імпорт    словника    '${lookupKey}'    завершено
успішно.`);
   } catch (error) {
       console.error(`Помилка під час імпорту словника '${lookupKey}':
${error.response?.data || error.message}`);
       sendError(`Помилка  під  час  імпорту  словника  '${lookupKey}':
${error.response?.data || error.message}`);
   }
}
```

Після успішного імпорту словників ми нарешті можемо перейти до завантаження даних абітурієнтів. Так як ми потребували словник освітніх ступенів, ми дістаємо ці дані з БД і ітеруємось по їх типам.

```
let documents = await collection.find({}).toArray();
while (documents.length > 0) {
   const level = documents[0];
   console.log(`Обробка рівня ${level.name}`);
   sendStatusUpdate(0, serviceName, `Обробка рівня: ${level.name}...`,
null);
   await ImportStudentsService.importStudentsForLevel(level, isActive,
AllStudentRows, collections);
   documents = documents.slice(1);
}

if (AllStudentRows.length > 0) {
   let intervalId = setInterval(() => {
       sendStatusUpdate(50,   serviceName,   'Запис   інформації   про
студентів у MongoDB...', null);
   }, 1000);

   await MongoDBService.clearCollection(collectionName);
```

```
   await MongoDBService.insertBulkData(AllStudentRows,
collectionName);

   clearInterval(intervalId);
```

Внутрішній метод, що є універсальним для усіх ступенів, просто робить запити на сервер, що потребують ідентифікатор університету, освітній ступінь, тип студента (активний/неактивний статус навчання) та кількість записів на сторінці при пагінації:

```
const rows = await
ImportStudentsService.fetchEdeboData(`${ImportStudentsService.EDEBO_A
PI_BASE_URL}list`, {
   'universityId': config.edeboUniversityID,
   'qualificationGroupId': level.id,
   'historyFilterId': historyFilterId,
   'pageSize': ImportStudentsService.DEFAULT_PAGE_SIZE
});
```

Вставка в колекцію здійснюється через сервіс MongoDBService, що надає функціонал для роботи з нереляційною базою даних, і функцією для збереження є insertBulkData(), що використовує бібліотеку mongoose:

```
static async insertBulkData(jsonDataArray, collectionName) {
   try {
       const GenericDataModel = await
this.createOrUpdateModel(collectionName);

       const dataArray = Array.isArray(jsonDataArray) ? jsonDataArray
: [jsonDataArray];

       await GenericDataModel.insertMany(dataArray);
       console.log(`Успішно вставлено ${dataArray.length} документів
у колекцію ${collectionName}`);
   } catch (error) {
       throw new Error(`Не вдалося вставити дані в MongoDB:
${error.message}`);
   }
}
```

Якщо такий запис в поданій колекції існує - він буде оновлений, в іншому випадку його створить. Аналогічним чином, здійснюється імпорт будь-яких інших колекцій із ЄДЕБО. Швидкість виконання всього циклу імпорту залежить від кількості даних певного університету, однак в середньому цей процес займає декілька хвилин.

Після повного скачування усіх записів виникає необхідність перенести їх до основної БД порталу MySQL. Зважаючи на величезну відмінність між реляційною базою та MongoDB, було прийнято рішення розробити сервіс для правильного перенесення даних із врахуванням типізації.

Одним із завдань було розробити схему створення таблиці, якої немає в MySQL - через можливість додавання/видалення нових полів або колекцій в ЄДЕБО неможливо наперед створити чітку структуру таблиць в БД. Відповідно, потрібно створити ті самі поля із аналогічними типами даних. Класом, що виконує вищезгаданий перелік задач, є MongoSqlSyncService:

```
class MongoSqlSyncService {
     constructor (mongoConnection, mysqlSequelize) {
     this.mongo = mongoConnection;
     this.mysql = mysqlSequelize;
     }
//…
 }
```

Клас містить конструктор, що приймає підключення до обох баз даних проекту.

Перед копіюванням таблиць із однієї БД в іншу відбувається отримання повного списку усіх колекцій MongoDB, по якому ми будемо здатні ітеруватися у циклі:

```
async getMongoCollectionNames () {
 const collections = await this.mongo.db.listCollections().toArray();
 let collectionNames = [];

 for (const collection of collections) {
   let collectionName = collection.name;
   let fields = await this.getCollectionFields(collectionName);
   if    (fields    ===    null    ||    fields    ===    undefined    ||
Object.keys(fields).length === 0) {
     continue;
   }
   collectionNames.push(collectionName);
 }
 collectionNames.sort();

 return collectionNames;
}
```

Отримання здійснюється за допомогою запитів, що збирають дані про таблицю та її будову.

Зважаючи на те, що колекції MongoDB не мають типізації, а також можуть містити вкладені об’єкти, було прийнято рішення розробити алгоритм [11] розпакування змінних та отримання їх типів даних. Метод getCollectionFields() приймає collectionName і містить наступну логіку:

```
if (collectionSchema === null || collectionSchema === undefined ||
Object.keys(collectionSchema).length === 0) {
 return {};
}
```

Спочатку здійснюється перевірка на наявність будь-яких записів, за відсутності яких неможливо визначити тип даних змінної. Якщо записи є - обирається перший і здійснюється аналіз за його допомогою. Далі розміщена конструкція, що визначає прості типи даних - строки, числа та дати:

```
if (!(fieldType === "array" || fieldType === "object")) {
 if (fieldType === "string" &&
(datetimeRegex.test(collectionSchema[key]) ||
datetimeRegex1.test(collectionSchema[key]) ||
datetimeRegex2.test(collectionSchema[key]))) {
   acc[key] = "datetime";
 } else if (fieldType === "number") {
   if (!Number.isInteger(collectionSchema[key])) {
     acc[key] = "decimal";
   } else {
     acc[key] = "number";
   }
 } else {
   acc[key] = fieldType;
 }
}
```

Якщо поле містить об’єкт або масив - ми повинні дістати за допомогою циклу з нього усі вкладені поля, які будуть розміщені в один ряд в структурі реляційної бази даних. За допомогою typeof ми отримуємо тип даних, також передбачено ситуації, коли дати в ЄДЕБО записуються у вигляді текстового формату:

```
for (let i = 1; i < nestedArrayObject; i++) {
 let nextObject = collectionSchema[key][i];
 if (nextObject[nestedKey] !== null) {
   nestedFieldType = typeof nextObject[nestedKey];
   if
   (nestedFieldType === "string" &&
     (
       datetimeRegex.test(nextObject[nestedKey]) ||
       datetimeRegex1.test(nextObject[nestedKey]) ||
```

```
        datetimeRegex2.test(nextObject[nestedKey]))
    ) {
      nestedFieldType = "datetime";
    }
    break;
  }
}
```

Виконавши вищезгадані кроки, отримана інформація з методів нам знадобиться при перекачуванні даних, що починається із перевірки на наявність відповідних таблиць та їх ініціалізації за відсутності. За стандартами створення таблиць в MySQL, було розроблено сервіс для перейменування назв із camel case у snake case:

```
const camelToSnakeCase = (key) => key.replace(/([a-z])([A-Z])/g, '$1_$2').toLowerCase();
```

Для відслідковування зміни чи додавання нових полів у таблицю використовується наступна конструкція:

```
const fields = await this.getCollectionFields(collectionName);
const attributes = {};

for (const field in fields) {
 if (!field.startsWith("_")) {

   let dataType = this.getFieldDataType(fields[field]);

   if (field === "id") {
     isId = true;
     attributes.id = {
       type: DataTypes.INTEGER,
       allowNull: false,
       autoIncrement: false,
       primaryKey: true
     };
     continue;
   }
 }
}
```

Ми пропускаємо поля, що починаються з “_”, наприклад, _id, а також примусово встановлюємо поле id як первинний ключ. Відбувається отримання типу даних поля. Далі назва колонки трансформується у правильний формат запису і записується у масив attributes:

```
 const snakeCaseField = camelToSnakeCase(field);
```

```
   attributes[snakeCaseField] = {
     type: dataType,
     allowNull: true
   };
```

Надалі отриманий перебудований масив полів та їх типів зберігається у таблиці, яка створюється в реляційній БД:

```
const tableName = camelToSnakeCase(collectionName);
const model = this.mysql.define(tableName, attributes, {
 tableName: tableName,
 timestamps: false,
 primaryKey: false
});

if (!isId) {
 model.removeAttribute("id");
}
await model.sync();
```

Валідний запис даних у MySQL забезпечується за допомогою застосування транзакцій. У випадку наявності помилок, ми можемо відновити попередній стан бази даних і повідомити про знайдені неточності:

```
readTransaction = await this.mysql.transaction();
try {
 await   this.mysql.query(`LOCK   TABLES   ${tableName}   WRITE`,   {
transaction: readTransaction });
 const oldRecords = await Model.findAll({ transaction: readTransaction
});
 await readTransaction.commit();
 readTransaction = null;
} catch (readError) {
 if (readTransaction) await readTransaction.rollback();
 throw readError;
}
writeTransaction = await this.mysql.transaction();
```

Для забезпечення цілісності даних дані блокуються для редагування, а також шукаються лише ті записи. що будуть новими для вставки. Через необхідність збереження постійної кількості даних в бд без дублювання необхідно було розділити логіку створення та оновлення. Такий вигляд має ініціалізація даних - спочатку здійснюється пошук нових записів:

```
if (newRecordsIds.length !== 0) {
 for (const newRecordsId of newRecordsIds) {
   const id = new ObjectId(newRecordsId);
   const cursor = collection.find({ _id: id });
   const documents = await cursor.toArray();
```

```
    newDocs.push(...documents);
  }
}
```

Далі обрані документи буде вставлено, попередньо видобуваючи їх вкладеності у одновимірний масив:

```
for (const nestedObject of arrayOfObjects) {
 const docWithoutArrays = {};

 for (const [key, value] of Object.entries(doc)) {
   if (!Array.isArray(value)) {
     docWithoutArrays[key] = value;
   }
 }
 const mergedDoc = { ...docWithoutArrays, ...nestedObject };
 const snakeCaseDoc = mapObjectKeys(mergedDoc, (value, key) =>
camelToSnakeCase(key));
 await Model.create(snakeCaseDoc, { transaction: writeTransaction
}).catch(e=>{console.error(e);})
}
```

За допомогою транзакції дані записуються, і подібний етап повторюється для кожної таблиці.

Оновлення виконує ті самі кроки, однак має лише одну відмінність - спочатку в межах транзакції видаляються всі наявні записи, що відповідають ідентифікаторам змінених документів. Після цього кожен вкладений об’єкт об’єднується з основною частиною документа, трансформується у формат, сумісний із MySQL (зокрема, ключі перетворюються у snake_case), і додається в базу як окремий запис:

```
if (arrayOfObjects) {
 for(const updatedDocumentId of updatedDocumentIds){
   await Model.destroy({ where: {_id: updatedDocumentId.toString() },
transaction: writeTransaction });
 }
 for (const nestedObject of arrayOfObjects) {
   const mergedDoc = { ...docWithoutArrays, ...nestedObject };
   const snakeCaseMergedDoc = mapObjectKeys(mergedDoc, (value, key) =>
camelToSnakeCase(key));
   snakeCaseMergedDoc._id = snakeCaseMergedDoc._id.toString();
   await Model.create(snakeCaseMergedDoc, { transaction:
writeTransaction });
}
```

Таким чином, триває довгий процес імпортування - алгоритм читає повідомлення з RabbitMQ, виконує вказану операцію завантаження даних з

ЄДЕБО, зберігає їх у MongoDB[19] і пересилає за допомогою транзакцій у базу даних проекту MySQL, де стає можливою безпосередня робота з ними через портал університету.

Одним із найпростіших видів даних, що можуть бути імпортованими, є інформація про заклади освіти абітурієнтів - це є основою для формування подальшої статистики.

```
if (Yii::$app->request->isAjax) {
   $documentIssuedData = $this->getDocumentIssuedData();
   Yii::$app->cacheChecker->setNewCache(self::CACHE_KEY,
$documentIssuedData);
   Yii::$app->end();
}
```

Якщо запит надійшов у форматі AJAX - це є сигналом для кешування, система викликає метод getDocumentIssuedData() для отримання актуальних статистичних даних, що є запитом до БД, одразу зберігає їх у кеш за визначеним ключем, після чого примусово завершує виконання скрипта. Якщо закешованих даних немає - ми робимо це автоматично при потраплянні на заданий роут:

```
if ($documentIssuedData['data'] === null || $documentIssuedData['data']
=== false) {
   $documentIssuedData = $this->getDocumentIssuedData();
   Yii::$app->cacheChecker->setNewCache(self::CACHE_KEY,
$documentIssuedData);
   $documentIssuedData = Yii::$app->cacheChecker-
>checkCache(self::CACHE_KEY);
}
```

Далі сформовані масиви статистичних даних формуються у необхідному порядку, ітеруючись по списку і повертаючи лише необхідну частину структури. У додатку використовується серверний рендерінг, через що в кінці буде викликана функція для побудови відповіді у вигляді HTML-документу, заповнюючи зарання прописані вставки:

```
$uniqueDocumentIssued =
ArrayHelper::getColumn($documentIssuedData['data'],
'document_issued');
$documentIssuedCount = ArrayHelper::map($documentIssuedData['data'],
'document_issued', 'count');
$originalDocumentCount = array_column($documentIssuedData['data'],
'original_count', 'document_issued');

return $this->render('index', [
   'uniqueDocumentIssued' => $uniqueDocumentIssued,
```

```
   'documentIssuedCount' => $documentIssuedCount,
   'originalDocumentCount' => $originalDocumentCount,
   'isValid'=>$documentIssuedData['isValid']
]);
```

HTML-таблиця з даними [10], де міститься інформація про заклади освіти, з яких абітурієнти подали заяви на освітній ступінь “бакалавр” на базі ПЗСО із колонками, скільки було прийнято заявок та скільки осіб було прийнято, має такий вигляд і реалізована за допомогою PHP-вставок [6]:

```
<?php $count = 0;
foreach ($uniqueDocumentIssued as $document):
   $count++;
   ?>
 <tr>
   <td>
       <?= $count ?>
   </td>
   <td><?= $document ?></td>
   <td><?php if (isset($originalDocumentCount[$document])): ?>
           <?= $originalDocumentCount[$document] ?>
       <?php endif; ?></td>
 </tr>
<?php endforeach; ?>
```

За допомогою більш складних наборів даних формується і більш цінна статистика, зокрема про стан подання заяв на бакалаврат та магістратуру. Їх принцип дії подібний, тож розглянемо це на прикладі інформації про бакалаврів:

```
$educationBaseName = 'Повна загальна середня освіта';
$educationFormNamePrefix = 'bachelors';
$qualificationGroupName = 'Бакалавр';
$isAJAX = Yii::$app->request->isAjax;
[
   $educationFormName,
   $result,
   $resultProposition,
   $importTime
]    =    $this->handlePropositions('bachelors',    $educationBaseName,
$qualificationGroupName, null, $isAJAX);
```

Задаються назви освітніх ступенів та префікси, щоб можна було знайти такі дані в таблиці. Спільна логіка була винесена до методу handlePropositions. Метод отримує останній запис історії імпорту, за параметром formName визначається форма здобуття освіти. Для можливості перемикання між формами навчання задається part, що визначає денну або заочну форми навчання:

```
$history = $this->getImportHistoryLast();
```

```
$defaultFormName = 'Денна';
$formNameFromRequest = Yii::$app->request->get('formName',
$defaultFormName);
$educationFormName = ($formNameFromRequest === 'part') ? 'Заочна' :
$defaultFormName;
```

Історія імпортувань береться за допомогою спеціального SQL-запиту, побудованого за допомогою класу Query:

```
private function getImportHistoryLast(): ImportHistory|null
{
   $history = ImportHistory::find()
       ->andWhere(["statistics_import_type_id" =>
self::STATISTICS_IMPORT_TYPE_ID])
       ->andWhere([ 'not', ['import_end' => null]]) ->orderBy(['id' =>
SORT_DESC]) ->one();
   return $history;
}
```

Через велике навантаження також було застосовано кешування [22], формуються унікальні ключі кешу для отримання основних та деталізованих даних для кожної форми навчання:

```
$cacheKey = $cacheKeyBase . $educationFormName;
$cachePropositionKey = $cacheKeyBase . 'Proposition' .
$educationFormName;
```

Дані будуть отримуватися з кешу, таким чином мінімізуючи витрати продуктивності на постійні повторні запити. Однак також потрібно було продумати і механізм оновлення даних, тому для забезпечення кращої взаємодії користувача з додатком і прибирання відчуття довгого запиту оновлення статистики відбувається за домопогою AJAX-запиту[23] зі сторони Javascript[9]. Відповідно, оновлення кешу здійснюється після того, як користувач переглядає сторінку:

```
if ($isAJAX) {
   [
       $result,
       $resultProposition
   ]           =           $this->getPropositions($educationBaseName,
$qualificationGroupName, $educationFormName, $selectStatuses);

   Yii::$app->cacheChecker->setNewCache($cacheKey, $result);
   Yii::$app->cacheChecker->setNewCache($cachePropositionKey,
$resultProposition);
   Yii::$app->end();
}
```

Функція getPropositions займається отриманням структурованого масиву статистичних даних, і цей процес розбитий на декілька етапів. Спочатку здійснюється отримання списку конкурсних пропозицій відповідно до освітньої бази, форми та групи кваліфікації:

```
$propositions = Proposition::find()
   ->where([
       'qualification_group_name' => $qualificationGroupName,
       'education_base_name' => $educationBaseName,
       'education_form_name' => $educationFormName
   ])
   ->orderBy([
       'university_facultet_full_name' => SORT_ASC,
       'speciality_code' => SORT_ASC,
       'person_request_date_start' => SORT_ASC
   ])
   ->asArray()
   ->all();
```

Формується SQL-запит до таблиці Proposition, у якому відбираються конкурсні пропозиції, що відповідають зазначеним критеріям. Результати впорядковуються за назвою факультету, кодом спеціальності та датою початку подачі заяв.

Для підрахунку заяв за зпеціальностями формується інший запит, який обчислює загальну кількість унікальних заяв (totalCount), кількість заяв з оригіналами документів (originalsCount) та список пріоритетів заяв (priorityCount):

```
$requestQuery = Request::find()
   ->select([
       'university_specialities_name',
       'university_specialities_id',
       'totalCount' => 'COUNT(DISTINCT person_request_id)',
       'originalsCount'     =>     'COUNT(DISTINCT     CASE     WHEN
is_original_documents_added = 1 THEN person_request_id ELSE NULL END)',
'priorityCount' => 'GROUP_CONCAT(request_priority)'
])->distinct('person_request_id')-
>where(['university_specialities_id'  =>  array_column($propositions,
'university_specialities_id')])
-
>groupBy(['university_specialities_name',"university_specialities_id"
]);
```

Вся отримана інформація переформовується у вкладеності масивів, відповідно до того, як її буде простіше відобразити у формі - чероез це

відбувається розбивка за факультетами. В окремі нові поля додаються обчислені кількості поданих заяв, пріоритетів та документів:

```
foreach ($propositions as $proposition) {
   $faculty = $proposition['university_facultet_full_name'];
   if (!isset($resultProposition[$faculty])) {
       $resultProposition[$faculty] = [];}
$universitySpecialitiesName                                          =
$proposition['university_specialities_id'];
$counts = $requestCounts[$universitySpecialitiesName] ?? [
       'totalCount' => 0,
       'originalsCount' => 0,
       'priorityCounts' => []
   ];
   $proposition['requestCount'] = $counts['totalCount'];
   $proposition['originalsCount'] = $counts['originalsCount'];
   $proposition['priorityCounts'] = $counts['priorityCounts'];
```

Ці вищеперечислені допоміжні методи застосовуються у кожному із роутів, які повертають на рендерінг відформатовані дані, по яким можна генерувати сторінку:

```
return $this->render('request-qualification', [
   'educationBaseName' => $educationBaseName,
   'qualificationGroupName' => $qualificationGroupName,
   'educationFormName' => $educationFormName,
   'dataResult' => $result,
   'dataResultProposition' => $resultProposition,
   'importTime' => $importTime,
   'educationFormNamePrefix' => $educationFormNamePrefix
]);
```

Для відображення діаграм було застосовано бібліотеку Chart.js. Ось приклад, де застосовується кругова діаграма, що відображає відсоткове співвідношення студентів денної та заочної форм навчання:

```
const dataPie = {
 labels: ["Денна", "Заочна"],
 datasets: [{
   data: [<?= $data['Денна'] ?? 0 ?>, <?= $data['Заочна'] ?? 0 ?>],
   backgroundColor: ["#17a2b8", "#6c757d"]
 }]
};
```

Дані отримуються за допомогою PHP-вставок, визначаються кольори для відображення. Сама діаграма будується за допомогою класу Chart [24]:

```
new Chart(ctxPie, {
 type: "pie",
```

```
 data: dataPie,
 options: optionsPie
});
```

Аналогічним чином, формується та відображається статистика і на інших сторінках побудованої онлайн системи.

Висновки до другого розділу

У процесі проєктування та розробки системи для автоматизованого управління освітнім процесом було визначено як функціональні, так і нефункціональні вимоги, сформульовано загальні цілі розробки, а також розроблено алгоритм імпортування даних вступників із стороннього джерела за участі мікросервісів. Отримані дані дозволять візуально оцінювати різні чинники, що вплинули на хід вступної кампанії, а також ефективніше здійснювати зарахування студентів приймальною комісією і автоматизувати процеси документообігу.

У ході проведеного аналізу предметної області було створено формалізовану інформаційну модель, що охоплює всі релевантні сутності, властиві цифровому освітньому середовищу. На підставі цієї моделі побудовано діаграму класів, яка репрезентує логічні зв’язки між об’єктами системи. Архітектурні рішення розроблялись з урахуванням специфікацій технічного завдання, що забезпечило відповідність проєкту вимогам щодо масштабованості, стійкості до збоїв, безпечної обробки даних та зручності в подальшому супроводі.

Результати реалізації підтверджують відповідність програмного продукту поставленим технічним і функціональним вимогам. Зокрема, реалізовано функціонал електронного журналу, інструменти погодження та коригування оцінок, механізми імпорту історичних звітів, а також інтеграцію цифрового підпису. У підсумку досягнуто основну мету проєкту — створено ефективний засіб цифрової підтримки навчального процесу, який відповідає актуальним вимогам до сучасних освітніх IT-систем.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ОНЛАЙН-МЕНЕДЖМЕНТУ ОСВІТНІХ ПРОЦЕСІВ

3.1. Порядок встановлення та налаштування параметрів додатку

Перед запуском проекту на реальних машина було розроблено діаграму розгортання [27], що наочно демонструє зв'язки між різними компонентами програмної системи (рис. 3.1):

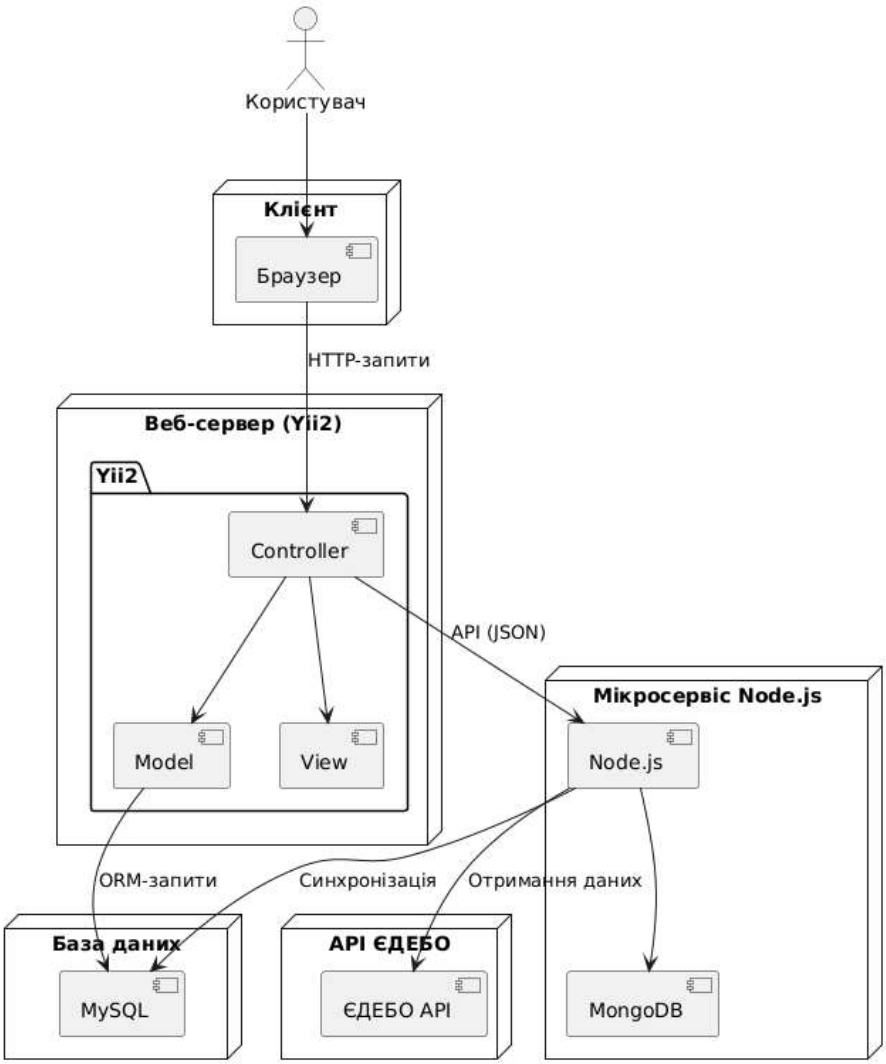

Рисунок 3.1 - Діаграма розгортання додатку

Діаграма розгортання демонструє архітектуру інформаційної системи, призначеної для управління освітніми процесами в університеті. Система реалізована за змішаним підходом: основний додаток розгорнутий як моноліт на фреймворку Yii2 з серверним рендерінгом, а додаткову функціональність

реалізовано за допомогою окремого мікросервісу на Node.js. Компоненти системи розподілені між кількома фізичними та логічними вузлами.

Користувач взаємодіє із системою через веб-браузер. Запити надсилаються до серверної частини, де розгорнуто Yii2-застосунок, що реалізує логіку за шаблоном MVC.

Основні компоненти вебсерверу:

- Controllers — обробляють HTTP-запити, виконують маршрутизацію та викликають відповідні сервіси.
- Models (Entities) — реалізують бізнес-логіку та взаємодіють з реляційною базою даних MySQL через Active Record ORM.
- Views — відповідають за формування HTML-інтерфейсу на основі даних, переданих контролерами.

Портал університету напряму підключається до MySQL - бази даних, де зберігається основна інформація про здобувачів освіти, співробітників, навчальних дисциплін, результатів сесій тощо.

Паралельно функціонує Node.js мікросервіс, який виконує допоміжні завдання:

- підключення до MongoDB для зберігання неструктурованих або проміжних даних;
- запити до зовнішнього API ЄДЕБО для отримання офіційних даних про студентів, абітурієнтів або навчальні програми;
- синхронізацію з основною MySQL-базою.

Всі сервіси взаємодіють через API-запити, переважно у форматі JSON. Це дозволяє забезпечити масштабованість, гнучкість та можливість подальшого розділення сервісів або перенесення частини навантаження.

Розгортання здійснюється на відповідному серверному обладнанні, яке забезпечує необхідну продуктивність та надійність. Система підтримується у всіх сучасних браузерах: Google Chrome (v45+), Mozilla Firefox (v38+), Microsoft Edge, Safari, що гарантує кросбраузерну сумісність [29] і зручність використання на різних пристроях без необхідності встановлення додаткового ПЗ.

Завдяки такій архітектурі система лишається гнучкою, розширюваною та легкою в обслуговуванні, що є критично важливим у контексті обробки освітніх даних та інтеграції з державними сервісами.

3.2. Структура інтерфейсу додатку

Розроблена система онлайн-менеджменту освітніх процесів характеризується наявністю інтуїтивно зрозумілого користувацького інтерфейсу та забезпечує повну адаптивність до різноманітних типів пристроїв.

Для персоналізованого доступу до ресурсу передбачено процедуру автентифікації. Зокрема, перед переходом на головну сторінку порталу користувач має пройти авторизацію шляхом введення облікових даних у відповідну форму (рис. 3.2), що гарантує захищений доступ до індивідуального функціоналу відповідно до ролі в системі.

Рисунок 3.2 - Вхід на портал

З міркувань інформаційної безпеки система не передбачає можливості самостійної реєстрації користувачів через веб-інтерфейс. Формування облікових записів покладається на відповідний структурний підрозділ університету, уповноважений здійснювати створення та розповсюдження логінів і паролів для доступу до системи. Централізований контроль за доступом до освітнього

середовища мінімізує ризики несанкціонованого проникнення особами, що не мають відношення до університету.

Після проходження процедури автентифікації користувач автоматично перенаправляється на головну сторінку платформи. Інтерфейс системи організовано таким чином, щоб забезпечити швидкий доступ до основних функціональних компонентів. З цією метою велика кількість модулів і підмодулів була логічно згрупована та представлена у вигляді зручної навігаційної панелі, розміщеної у лівій частині екрана (рис. 3.3).

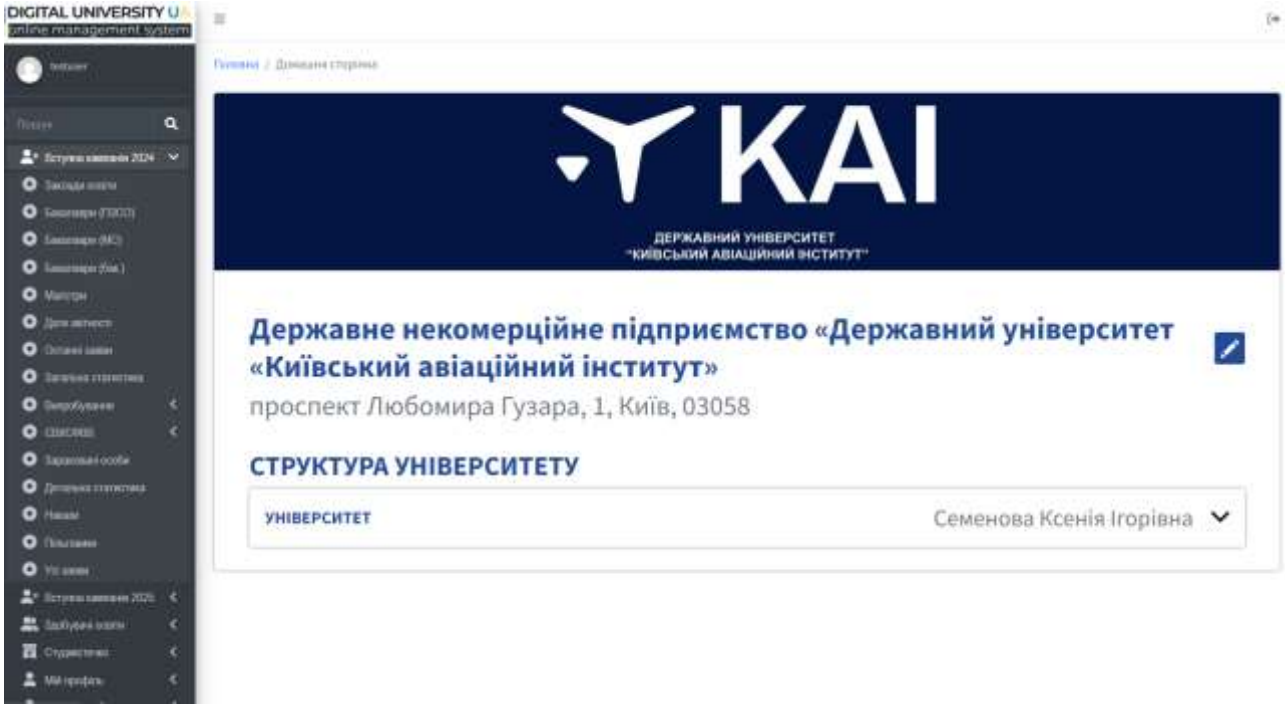

Рисунок 3.3 - Головна сторінка

Серед усіх частин спроектованого додатку ми розглядатимемо модуль «Вступна кампанія», який є ключовим для отримання усіх даних про університет, співробітників, студентів, навчальні програми тощо, з якими в подальшому працює система. Доступ до цих даних забезпечує вкладка «Зв’язок з ЄДЕБО» (рис. 3.4):

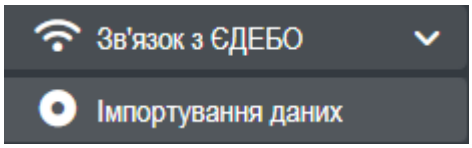

Рисунок 3.4 – Зв’язок з ЄДЕБО

Огляд системи здійснюється від імені користувача з правами адміністратора, так як інформація про вступну кампанію та можливість керування імпортуванням даних доступні лише йому. При переході на сторінку користувачу відображаться меню панелі керування (рис. 3.5):

Рисунок 3.5 – Панель керування імпортом

Відображається поточна кількість фізичних осіб, студентів, поданих заяв, а також співробітників у базі даних. Під кожним із параметрів відображається статистика, скільки даних було імпортовано за минулий день. Ліворуч розташовані кнопки для запуску процесу імпорту, а праворуч – поточна черга імпортувань. Наразі вона є порожньою, тож запустимо перший пункт – імпортування довідників, що виконують роль енумерацій в базі даних і містять здебільшого статичні дані, що відкриті до модифікування (рис.3.6):

Рис. 3.6. – Запуск імпортування

Нижче розташований список історії імпортувань (рис. 3.7):

Рисунок 3.7 – Лог імпортування

Поруч горить жовтий прапорець, це означає, що імпортування ще не завершилося. Після закінчення процесу позначка змінить колір на зелений (успішно) або на червоний (помилка) (рис. 3.8):

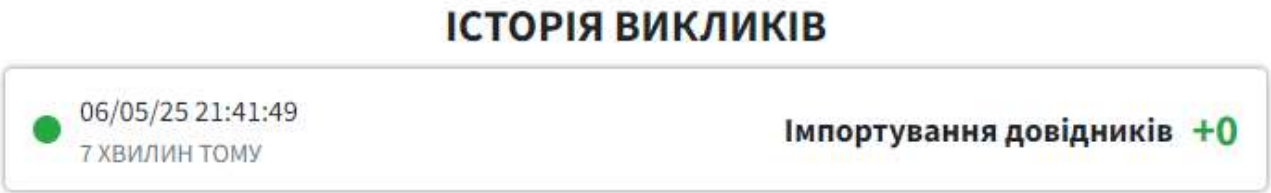

Рисунок 3.8 – Успішний імпорт

Як ми бачимо з рисунку 3.8, обсяг імпортованих даних дорівнює нулю, що означає, що кількість записів у словниках не змінилася.

Після виконання першого іспортування університету доступний перегляд статистичних даних про вступ абітурієнтів у цьому році.

Користувач має можливість переглядати статистику вступної кампанії за обраний рік. Візуальне оформлення із використанням кругових діаграм, гістограм та табличних даних сприяє кращому візуальному сприйняттю (рис. 3.9):

Рисунок 3.9 – Стан подання заяв на освітній ступінь «Бакалавр»

Аналогічно, можна переглянути і статистику вступної кампанії на магістратуру (рис.3.10):

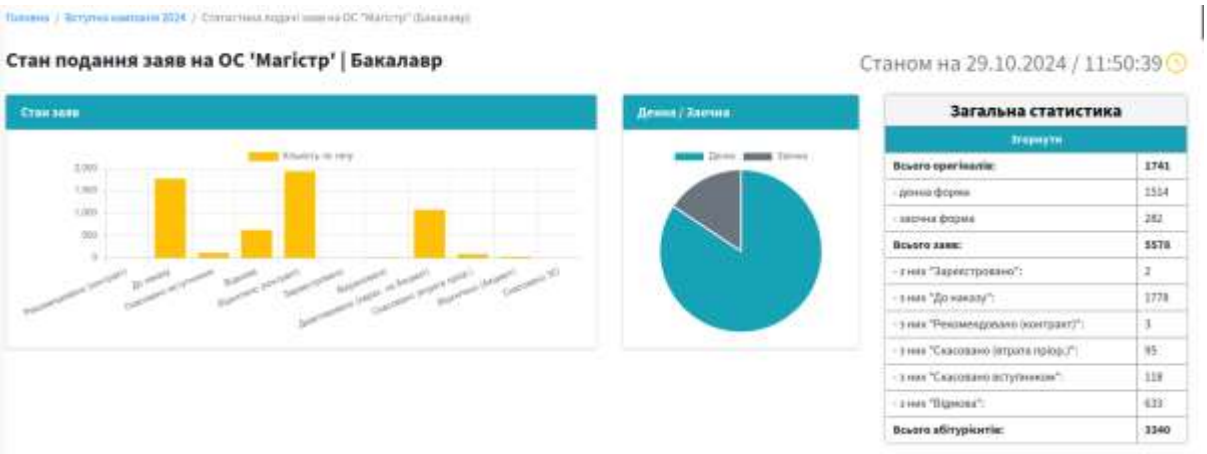

Рисунок 3.10 – Стан подання заяв на освітній ступінь «Магістр»

Нижче на сторінці статистики подачі заяв розташована таблиця, де виведена детальна інформація за факультетами, спеціальностями та пріоритетностями (рис. 3.11):

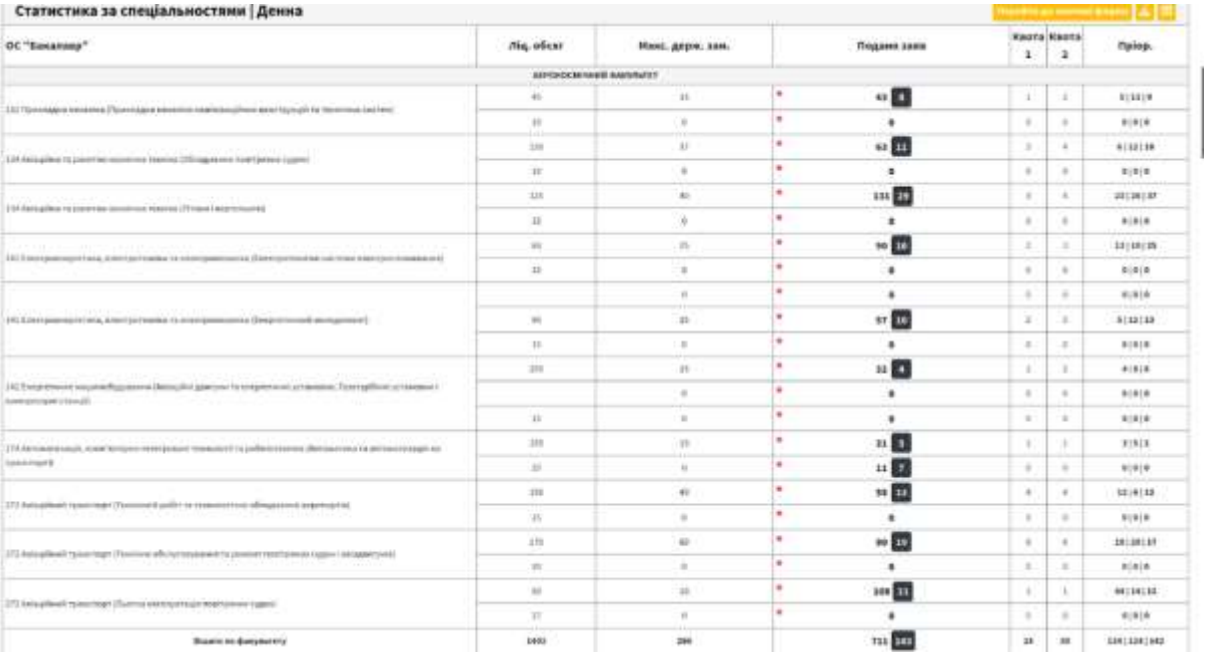

Рисунок 3.11 – Таблиця статистики за спеціальностями

Щоб дізнатися більше інформації, є можливість натиснути на відповідну комірку із заявами (рис. 3.12):

Рисунок 3.12 – Інформація про конкурсну пропозицію

Користувача перенесе на сторінку, що містить детальну інформацію про конкурсну пропозицію - спеціальність, тип пропозиції, курс, термін навчання, обсяг бюджетних місць тощо. Нижче також знаходиться таблиця, в якій представлено абітурієнтів, що подали заявки (рис. 3.13):

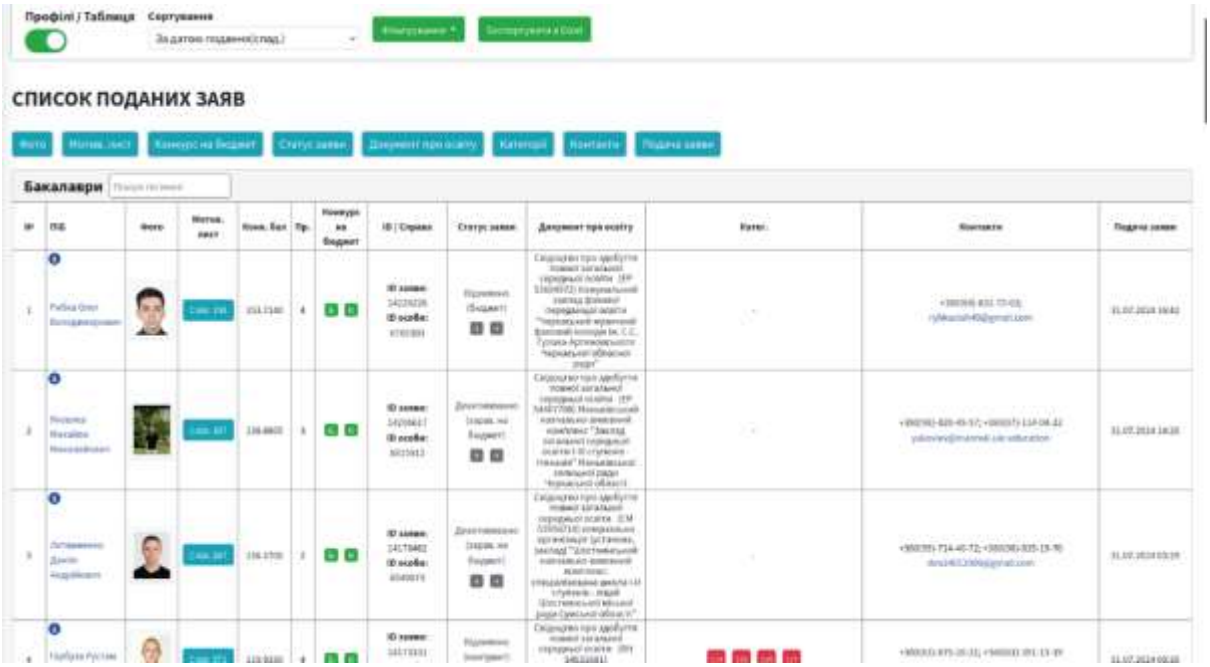

Рисунок 3.13 – Список поданих заяв на спеціальність

При натисканні відбудеться відкриття сторінки з особистими даними вступника (рис. 3.14):

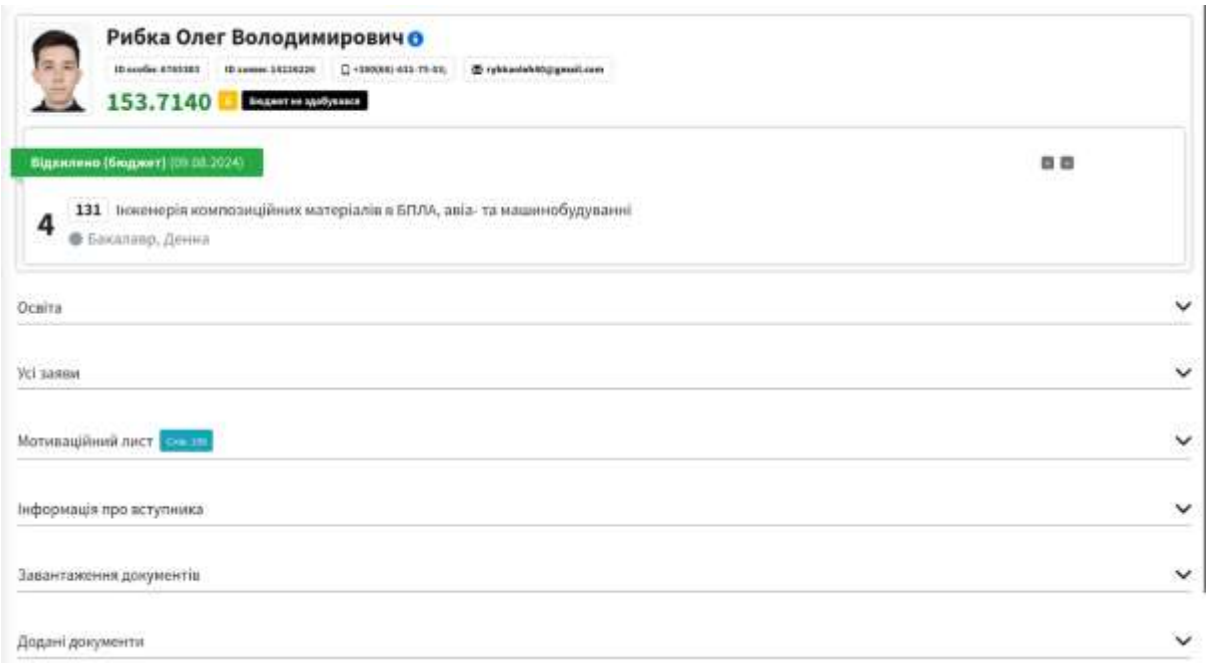

Рисунок 3.14 – Дані абітурієнта

Крім перегляду інформації про попередню освіту, мотиваційний лист, можливе завантаження та додавання документів (рис. 3.15):

Рисунок 3.15 – Список поданих заяв на спеціальність

Якщо необхідно працювати із будь-якими заявами, незалежно від факультету - сторінка “Останні заяви” надає таку можливість (рис. 3.16):

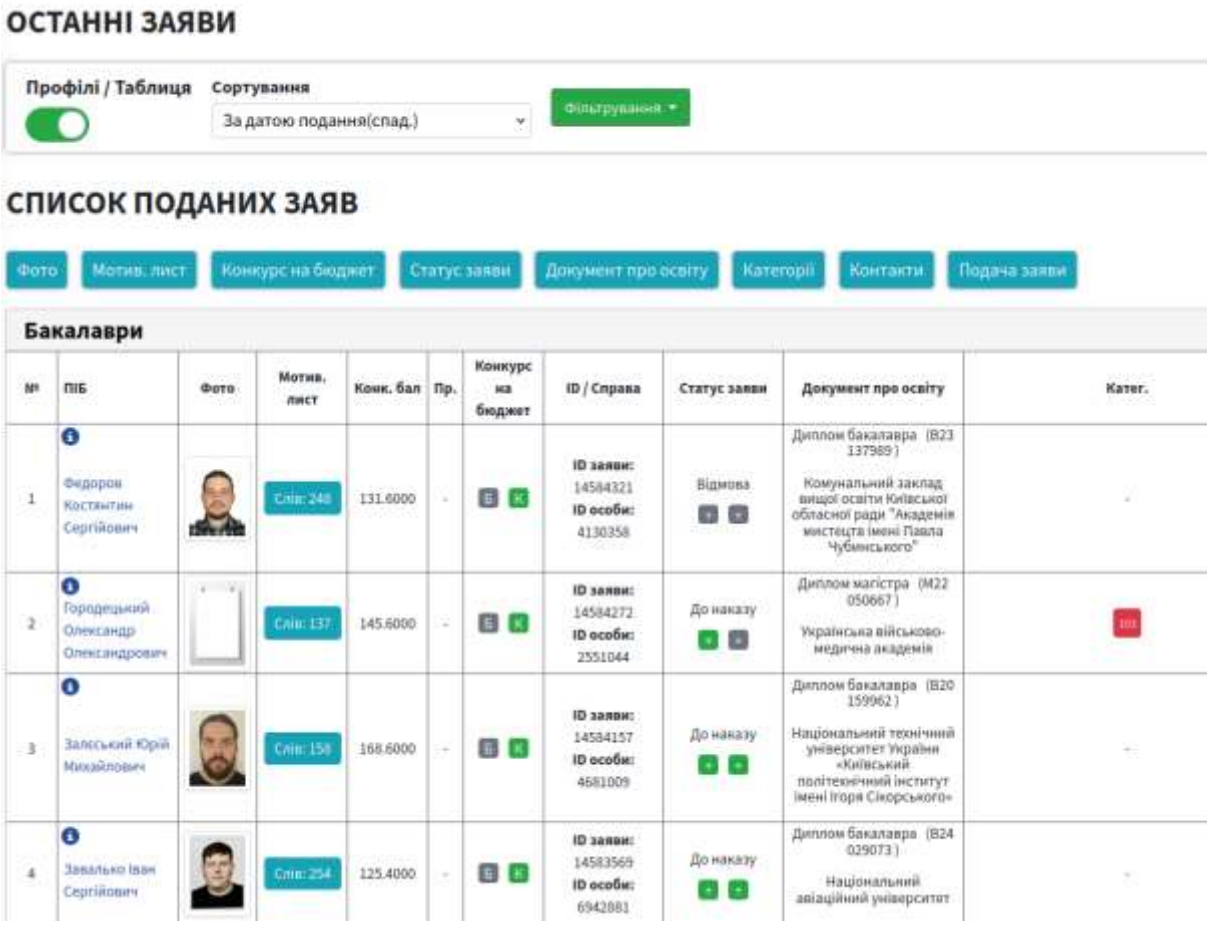

Рисунок 3.16 – Останні заяви

З метою зручності та оптимізації навантаження на сервер інформацію про загальну статистику було винесено на окрему сторінку (рис. 3.17):

Загальна статистика

	Бакалавр (ПЗСО)	Бакалавр (МС)	Бакалавр (бак.)	Молодший бакалавр	Магістр	РАЗОМ
Всього абітурієнтів:	1679	159	2	0	1725	3565
Всього заяв:	12456	802	2	0	5578	18839
Всього оригіналів:	1727	161	2	0	1796	3686
- Денне:	1579	143	1	0	1514	3239
- Заочне:	148	18	1	0	282	447
- з них "Відмова":	596	37	0	0	608	1242
- з них "Відраховано":	46	2	0	0	17	65
- з них "Відкликано (бюджет)":	107	5	0	0	36	148
- з них "Відкликано (контракт)":	7616	425	0	0	1902	9943
- з них "Деактивовано (зарах. на бюджет)":	1754	120	0	0	1078	2952
- з них "До наказу":	1639	156	2	0	1723	3520
- з них "Допущено":	3	0	0	0	0	3
- з них "Зареєстровано":	0	0	0	0	2	2
- з них "Рекомендовано (контракт)":	1	0	0	0	2	3
- з них "Скасовано (втрата пріор.)":	224	30	0	0	90	344
- з них "Скасовано вступником":	470	28	0	0	115	613
- з них "Скасовано ЗО":	0	0	0	0	5	5
- з них "Заява надійшла з сайту":	0	0	0	0	0	0
- з них "Затримано":	0	0	0	0	0	0
- з них "Рекомендовано (бюджет)":	0	0	0	0	0	0

Рисунок 3.17 – Загальна статистика

Також веб-система дозволяє перегляд списків абітурієнтів, що подавали заяви на вступні випробування у даному навчальному закладі, з можливістю сортування, фільтрації та пагінації (рис. 3.18):

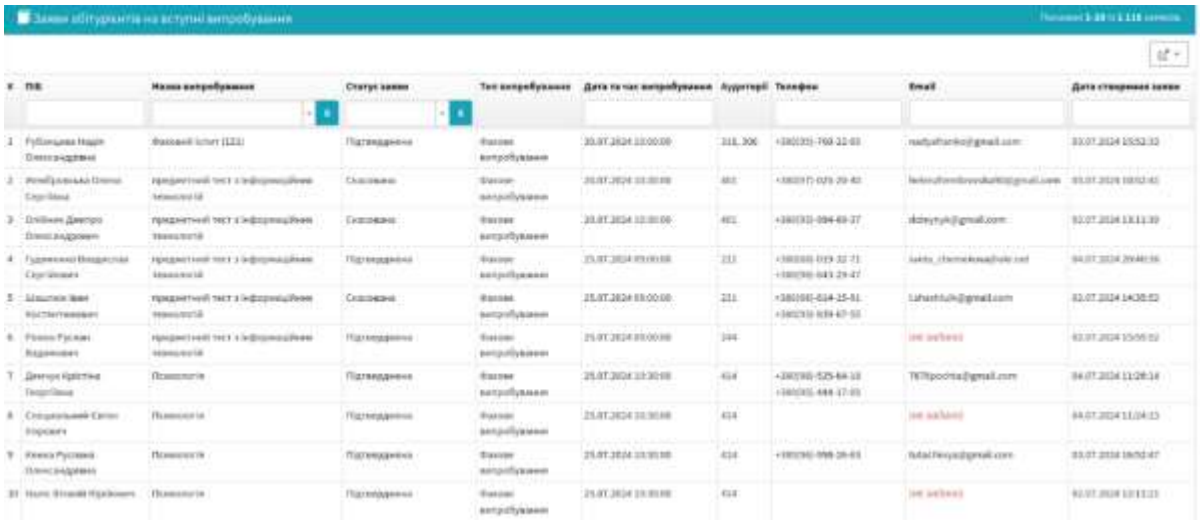

Рисунок 3.18 – Заяви на вступні випробування

Окремо розміщена сторінка із інформацією про зареєстрованих осіб на ЄВІ та ЄФВВ (рис. 3.19):

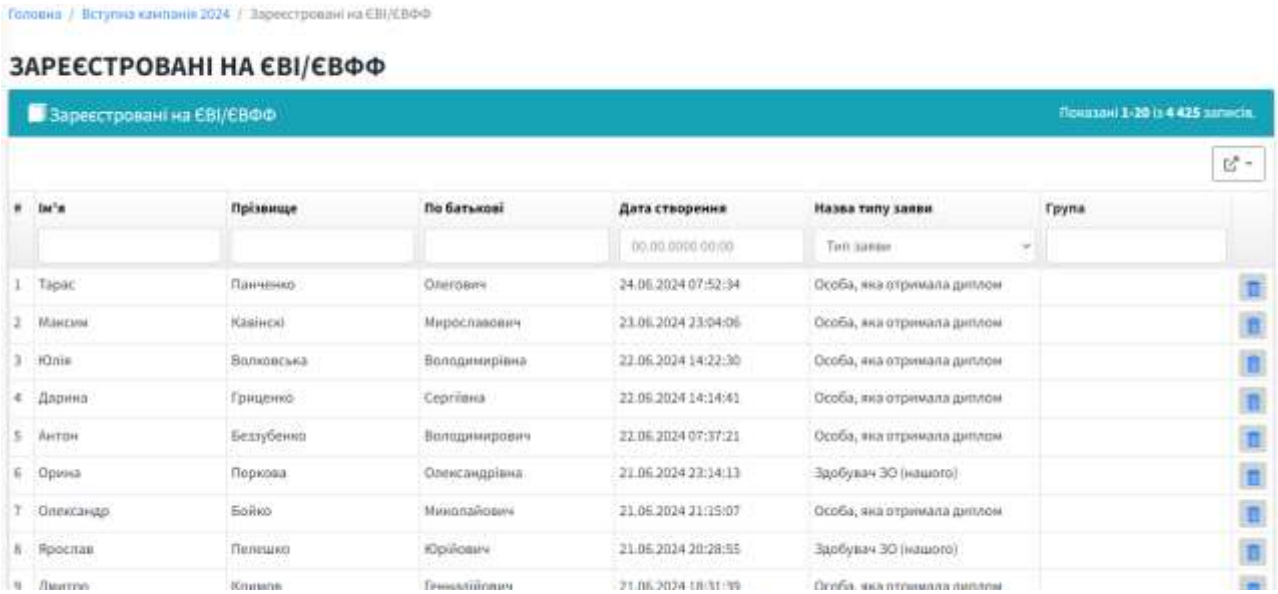

Рисунок 3.19 – Заяви ЄВІ та ЄФВВ

За потреби адміністрація університету також має доступ до списків зарахованих осіб за період певної вступної кампанії з можливістю перемикання між вкладками (рис. 3.20-3.21):

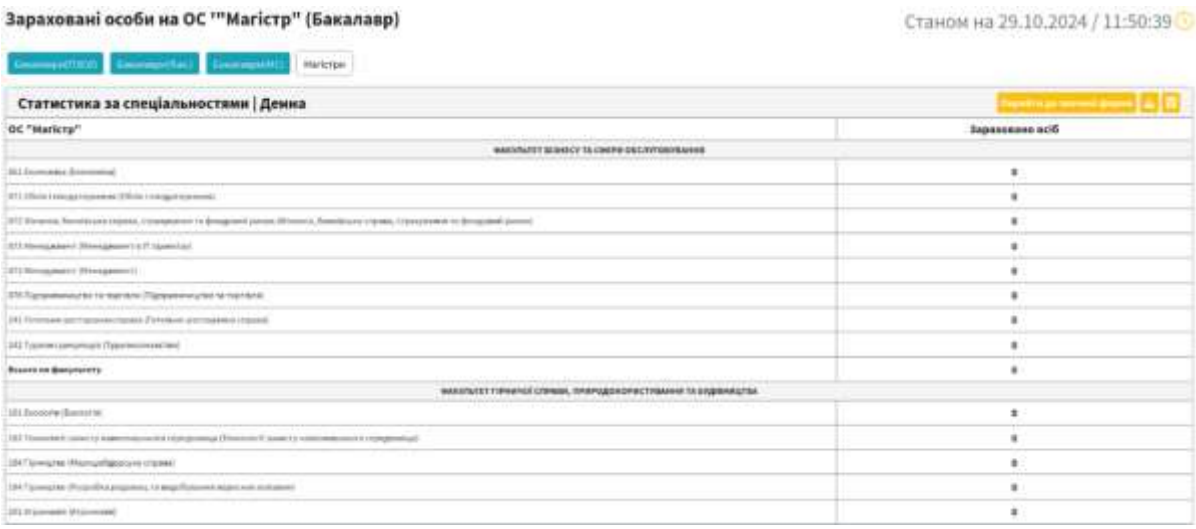

Рисунок 3.20 - 3.21 – Зараховані особи

При натисканні на кнопку завантаження відбувається генерація Excel файлу із даними вступників, який автоматично завантажується на пристрій користувача (рис. 3..22):

Рисунок 3.22 – Файл із даними вступників

3.3. Тестування роботи розробленого додатку

У межах розробки системи для автоматизації процесів вступної кампанії до закладу вищої освіти було проведено ручне функціональне тестування [21], що передбачає перевірку основного функціоналу без використання автоматизованих скриптів. Такий підхід дозволив максимально наблизити перевірку до реальних сценаріїв використання, виявити неочевидні помилки, а також оцінити зручність інтерфейсу та логіку навігації для різних категорій користувачів.

Основною метою тестування було переконатися в стабільності роботи системи, коректності виконання кожного модуля та відповідності реалізованого функціоналу до встановлених бізнес-вимог і очікувань адміністрації та власників системи.

1. Сценарій: Перегляд заяв вступників

Було перевірено працездатність головного інтерфейсу для співробітників приймальної комісії. У процесі тестування виявлено, що таблиця заяв коректно відображає список абітурієнтів, підтримує сортування за статусом, спеціальністю та датою подачі. Інтерфейс не перевантажений, навігація інтуїтивно зрозуміла. Тестування показало, що дані завантажуються динамічно, навіть при великому обсязі інформації, система не демонструє помітних затримок.

Результат: Успішно. Відображення, фільтрація і сортування працюють належним чином.

2. Сценарій: Експорт даних у Excel

Тестування експорту даних з таблиці заяв підтвердило, що формат згенерованого файлу відкривається у різних редакторах, зокрема Microsoft Excel та Google Sheets [26], файли формуються швидко, збереження здійснюється без втрати даних. Особливу увагу приділено кодуванню українських символів — помилок в експортованих файлах не виявлено.

Результат: Успішно. Генерація звітів працює згідно з технічним завданням.

4. Сценарій: Синхронізація з ЄДЕБО

Було проведено імітацію повного імпорту нових записів через окремий мікросервіс на Node.js. Система коректно обробляє запити, виконує синхронізацію даних у фоновому режимі, дублювання записів не виявлено. Логіка перевірки унікальності працює згідно з очікуваннями. Також протестовано вибіркове імпортування, лог імпорту показує відповідну інформацію про кількість імпортованих даних та час виконання операції, помилок не виявлено.

Результат: Успішно. Дані синхронізуються коректно, помилок у записах не виявлено.

5. Сценарій: Перегляд реєстрацій на вступні іспити

Інтерфейс дозволяє швидко знайти зареєстрованих осіб на відповідний іспит, здійснено перевірку на можливість сортування за датами та категоріями. Доступні поля: ПІБ, спеціальність, дата реєстрації, аудиторія проведення.

Результат: Успішно. Всі реєстраційні дані відображаються та фільтруються згідно з вимогами.

Висновки до третього розділу

У третьому розділі було проведено розгортання вебзастосунку на серверному середовищі, що відповідає технічним параметрам, встановленим на початковому етапі розробки. Після розгортання відбулося комплексне тестування реалізованого функціоналу, що дало змогу перевірити стабільність та коректність роботи ключових компонентів системи.

У межах функціоналу для адміністративного персоналу реалізовано інструменти перегляду заяв вступників із можливістю фільтрації та сортування за різними критеріями — спеціальностями, датами подачі, статусами тощо. Додатково забезпечено доступ до повного переліку документів абітурієнтів, а також історії їх попереднього навчання.

Суттєвим етапом стала реалізація експорту даних у формат Excel, що забезпечує зручне формування звітів. Передбачено механізм синхронізації з ЄДЕБО для отримання актуальної інформації. Для зручного аналізу

реалізовано блок статистики, який дозволяє оцінювати динаміку подачі заяв, кількість заяв за спеціальностями та показники успішності вступників.

Окремо опрацьовано функціонал, що стосується вступних іспитів: реалізовано відображення зареєстрованих осіб, збереження інформації про результати, а також перегляд списку зарахованих вступників із можливістю побудови статистичних звітів.

Результати тестування підтвердили повну працездатність розробленого вебсайту, відповідність заявленим функціональним вимогам та його готовність до подальшої експлуатації. Таким чином, поставлені задачі були успішно реалізовані, що свідчить про досягнення головної мети розробки.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було реалізовано повний життєвий цикл розробки функціонального модуля, призначеного для автоматизації процесів, пов’язаних із організацією та супроводом вступної кампанії в межах системи цифрового менеджменту освітніх процесів. Основна мета полягала в оптимізації процедур планування, збору та обробки даних під час заліково-екзаменаційного періоду, а також забезпеченні прозорого контролю за освітньою діяльністю.

На етапі підготовки виконано порівняльний аналіз сучасних вебтехнологій, що використовуються для створення масштабованих інформаційних систем. В результаті техніко-економічного обґрунтування було обрано мову програмування PHP у поєднанні з фреймворком Yii2, що забезпечило високу продуктивність, модульність коду та захищену роботу з базами даних. Вибір монолітної архітектури дозволив досягти балансу між швидкістю реалізації та простотою супроводу системи.

У рамках проєктного етапу сформульовано як функціональні, так і нефункціональні вимоги до майбутнього програмного продукту. Було структуровано основні логічні процеси, розроблено UML-діаграми, які візуалізують сценарії використання модуля та взаємозв’язки його ключових компонентів. Окрему увагу приділено інтеграції з зовнішніми інформаційними системами, а також реалізації інструментів імпорту/експорту даних, перегляду заяв абітурієнтів і доступу до аналітичної інформації.

Фінальний етап включав розгортання модуля на серверному середовищі та проведення серії функціональних тестів. Було перевірено коректність механізмів автентифікації, імпорту даних у фоновому режимі, обробки заяв, завантаження та перегляду супровідної документації, а також генерації звітності та виведення статистичних показників за вибраний період.

Таким чином, реалізований модуль «Вступна кампанія» продемонстрував здатність підвищувати ефективність і прозорість адміністративних процесів у сфері освіти, а також готовність до впровадження в комплексну систему керування навчальним закладом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Цифровізація у вищій освіті: інституційні підходи до викладання та навчання / Семеняко Ю.Б., Брюховецька І.В., Бохонько О.Є. – Класичний приватний університет, 2023. – 186 с.
2. Платформа Moodle [Електронний ресурс] – Режим доступу до ресурсу: https://moodle.org/
3. Canvas [Електронний ресурс] – Режим доступу до ресурсу: https://www.instructure.com/canvas
4. OpenEduCat [Електронний ресурс] – Режим доступу до ресурсу: https://www.openeducat.org/
5. The Definitive Guide to Yii 2.0 [Електронний ресурс] – Режим доступу до ресурсу: https://www.yiiframework.com/doc/guide/2.0/en
6. Data Structures and Algorithm Analysis in PHP / Michael A. Redmond – Morgan Kaufmann, 2014. – 400 с.
7. PHP Cookbook: Modern Code Solutions for Professional Developers / Eric Mann - O'Reilly Media, 2023 – 420 с.
8. Building Microservices with Node.js / Daniel Kapexhiu – Packt Publishing, 2024. – 324 с.
9. Hands-On Microservices with JavaScript: Build scalable web applications with JavaScript, Node.js, and Docker / Tural Suleymani – Packt Publishing, 2024. – 406 с.
10. Learning Web Design: A Beginner’s Guide to HTML, CSS, JavaScript and Web Graphics / Jennifer Niederst Robbins – O’Reilly, 2021. – 640 с.
11. Introduction to Algorithms / Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein – MIT Press, 2022. – 1312 с.
12. Algorithms in a Nutshell / George T. Heineman, Gary Pollice, Stanley Selkow – O’Reilly, 2017. – 800 с.

13. Microservices: How To Make Your Application Scale / Dragoni et al. – Cornel University, 2017. – 346 с.
14. Microservices: Flexible Software Architectures / Eberhard Wolff – Addison-Wesley, 2016. – 400 с.
15. MySQL Crash Course / Ben Forta – Addison-Wesley, 2024. – 400 с.
16. PHP, MySQL and Apache / Julie C. Meloni – Peachpit Press, 2022. – 512 с.
17. High Performance MySQL: Optimization, Backups, Replication & Load Balancing / Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, Jeremy Zawodny – O'Reilly, 2023. – 1024 с.
18. MongoDB in Action, Third Edition / Arek Borucki – Manning, 2023. – 450 с.
19. Data Modeling for MongoDB / Steve Hoberman – O'Reilly, 2021. – 320 с.
20. API Design Patterns / JJ Geewax – Manning, 2022. – 350 с.
21. The Art of Unit Testing / Roy Osherove – Manning Publications, 2013. – 320 с.
22. Pro PHP Application Performance: Tuning PHP, MySQL and Apache / Armando Padilla – Apress, 2011. – 240 с.
23. Ajax: The Complete Reference / Thomas Powell – McGraw-Hill (Complete Reference), 2007. – 560 с.
24. Creating Software with Modern Diagramming Techniques / Matthew Skelton, Alexander Paisley – Pragmatic Bookshelf, 2021. – 280 с.
25. Інструкції по роботі з ЄДЕБО [Електронний ресурс] – Режим доступу до ресурсу: https://www.inforesurs.gov.ua/edebo/instructions/
26. PHP Excel Programming with OpenTBS / Daniel Opitz – Leanpub, 2020. – 154 с.
27. UML Distilled: A Brief Guide to the Standard Object Modeling Language / Martin Fowler – Addison-Wesley, 2018. – 208 с.
28. Software Requirements / Karl Wiegers, Joy Beatty – Microsoft Press, 2014. – 648 с.

29. How Browsers Work: Behind the scenes of modern web browsers [Електронний ресурс] – Режим доступу до ресурсу: https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/
30. Introducing JSON [Електронний ресурс] – Режим доступу до ресурсу: https://www.json.org/json-en.html

ДОДАТКИ

Додаток А

```
public function actionImport(string $importName = '', string $collectionName = ''): int
{
    $message = json_encode([
        'importName' => $importName,
        'collectionName' => $collectionName,
    ]);

    $this->amqpManager->sendMessage($message);

    return ExitCode::OK;
}

/**
* @throws Exception
*/
public function getConnection(): AMQPStreamConnection
{
    $config = Yii::$app->components['rabbitmq'];

    return new AMQPStreamConnection(
        $config['RABBITMQ_IPV4'],
        $config['RABBITMQ_PORT'],
        $config['RABBITMQ_USER'],
        $config['RABBITMQ_PASSWORD'],
        $config['vhost']
    );
}

public function sendMessage($message): void
{
    $connection = $this->getConnection();
    $channel = $connection->channel();
    $queueName = "edebo";
    try {
        $channel->queue_declare($queueName, false, false, false, false);

        $msg = new AMQPMessage($message);
        $channel->basic_publish($msg, "", $queueName);

        echo " [x] Sent '$message'\n";
    } catch (Exception $e) {
        echo " [x] Error sending message: {$e->getMessage()}\n";
    } finally {
        $channel->close();
        $connection->close();
    }
}

async consumeMessage () {
 try {
    await this.ensureChannel();
```

```
    await this.channel.assertQueue("edebo", { durable: false });

    this.channel.consume("edebo", async (msg) => {
      if (msg) {
        const messageContent = JSON.parse(msg.content.toString());
        await    this.processMessage(msg,    messageContent.importName,
messageContent.collectionName, messageContent.params);
      }
    }, { noAck: false });
  } catch (error) {
    console.error("Error consuming messages:", error.message);
    throw error;
  }
}

async processMessage (msg, importName, collectionName, params) {
  try {
    this.channel.ack(msg);
    await this.close();
    await importFunction(importName, params);
    await syncFunction(importName, collectionName);

  } catch (error) {
    console.error("Response error:", error.message);
  } finally {
    await this.consumeMessage();
  }
}

ActiveStudentsImportService:
ImportStudentsService.importActiveStudents,
AllStudentsImportService: ImportStudentsService.importAllStudents,

static async importActiveStudents() {
    sendStatusUpdate(null,    'ActiveStudentsImportService',    'Початок
імпортування студентів (Активні)...', null);
    await ImportStudentsService.importStudents(true);
}

static async importAllStudents() {
    sendStatusUpdate(null,     'AllStudentsImportService',     'Початок
імпортування студентів (Усі)...', null);
    await ImportStudentsService.importStudents(false);
}

const                collection                =                 await
mongoose.connection.db.collection('edebo_dictionary_std_qualification
groups');

console.log('Початок імпорту словників...');
const                        staticProperties                        =
Object.keys(EdeboDictionaryConstants).filter(key     =>     key     !==
'DictionaryNames');
let index = 1;
let count = staticProperties.length;
```

```
let currentDictionary = null;
let intervalId = setInterval(() => {
   if (currentDictionary) {
       let percentComplete = (index / count) * 100;
       sendStatusUpdate(Math.floor(percentComplete),
'DictionariesImport', `${currentDictionary}`, `${index} / ${count}`);
   }
}, 1000);

for (let dictionaryName of staticProperties) {
let                          displayName                          =
EdeboDictionaryConstants.DictionaryNames[dictionaryName]            ||
dictionaryName;
currentDictionary = displayName;
console.log(`Імпортування   словника   '${displayName}'   (${index}   /
${count})...`);
await DictionariesImportService.ImportDictionary(dictionaryName);
let percentComplete = (index / count) * 100;
sendStatusUpdate(Math.floor(percentComplete),    'DictionariesImport',
`${displayName}`, `${index} / ${count}`, false);
index++;
}
static async ImportDictionary(lookupKey) {
   try {
console.log(`Імпорт словника '${lookupKey}' розпочато...`);
let rows = await DictionariesImportService.dictionaryGet(lookupKey);
let collection_name = 'edebo_dictionary_' + lookupKey;
await    MongoDBService.enterDataIntoMongoDB(rows,    collection_name,
'id');
       console.log(`Імпорт    словника    '${lookupKey}'    завершено
успішно.`);
   } catch (error) {
       console.error(`Помилка під час імпорту словника '${lookupKey}':
${error.response?.data || error.message}`);
       sendError(`Помилка  під  час  імпорту  словника  '${lookupKey}':
${error.response?.data || error.message}`);
   }
}

let documents = await collection.find({}).toArray();
while (documents.length > 0) {
   const level = documents[0];
   console.log(`Обробка рівня ${level.name}`);
   sendStatusUpdate(0, serviceName, `Обробка рівня: ${level.name}...`,
null);
   await ImportStudentsService.importStudentsForLevel(level, isActive,
AllStudentRows, collections);
   documents = documents.slice(1);
}

if (AllStudentRows.length > 0) {
   let intervalId = setInterval(() => {
       sendStatusUpdate(50,   serviceName,   'Запис   інформації   про
студентів у MongoDB...', null);
   }, 1000);
```

```
   await MongoDBService.clearCollection(collectionName);
   await                 MongoDBService.insertBulkData(AllStudentRows,
collectionName);

   clearInterval(intervalId);

const                  rows                  =                  await
ImportStudentsService.fetchEdeboData(`${ImportStudentsService.EDEBO_A
PI_BASE_URL}list`, {
   'universityId': config.edeboUniversityID,
   'qualificationGroupId': level.id,
   'historyFilterId': historyFilterId,
   'pageSize': ImportStudentsService.DEFAULT_PAGE_SIZE
});

static async insertBulkData(jsonDataArray, collectionName) {
   try {
       const            GenericDataModel            =            await
this.createOrUpdateModel(collectionName);

       const dataArray = Array.isArray(jsonDataArray) ? jsonDataArray
: [jsonDataArray];

       await GenericDataModel.insertMany(dataArray);
       console.log(`Успішно вставлено ${dataArray.length} документів у
колекцію ${collectionName}`);
   } catch (error) {
       throw  new  Error(`Не  вдалося  вставити  дані  в  MongoDB:
${error.message}`);
   }
}

class MongoSqlSyncService {
     constructor (mongoConnection, mysqlSequelize) {
     this.mongo = mongoConnection;
     this.mysql = mysqlSequelize;
     }
//…
 }

async getMongoCollectionNames () {
 const collections = await this.mongo.db.listCollections().toArray();
 let collectionNames = [];

 for (const collection of collections) {
   let collectionName = collection.name;
   let fields = await this.getCollectionFields(collectionName);
   if   (fields   ===   null   ||   fields   ===   undefined   ||
Object.keys(fields).length === 0) {
     continue;
   }
   collectionNames.push(collectionName);
 }
 collectionNames.sort();
```

```
 return collectionNames;
}

if (collectionSchema === null || collectionSchema === undefined ||
Object.keys(collectionSchema).length === 0) {
 return {};
}

if (!(fieldType === "array" || fieldType === "object")) {
 if (fieldType === "string" &&
(datetimeRegex.test(collectionSchema[key]) ||
datetimeRegex1.test(collectionSchema[key]) ||
datetimeRegex2.test(collectionSchema[key]))) {
   acc[key] = "datetime";
 } else if (fieldType === "number") {
   if (!Number.isInteger(collectionSchema[key])) {
     acc[key] = "decimal";
   } else {
     acc[key] = "number";
   }
 } else {
   acc[key] = fieldType;
 }

}

for (let i = 1; i < nestedArrayObject; i++) {
 let nextObject = collectionSchema[key][i];
 if (nextObject[nestedKey] !== null) {
   nestedFieldType = typeof nextObject[nestedKey];
   if
   (nestedFieldType === "string" &&
     (
       datetimeRegex.test(nextObject[nestedKey]) ||
       datetimeRegex1.test(nextObject[nestedKey]) ||
       datetimeRegex2.test(nextObject[nestedKey]))
   ) {
     nestedFieldType = "datetime";
   }
   break;
 }
}

const camelToSnakeCase = (key) => key.replace(/([a-z])([A-Z])/g,
'$1_$2').toLowerCase();
const fields = await this.getCollectionFields(collectionName);
const attributes = {};

for (const field in fields) {
 if (!field.startsWith("_")) {

   let dataType = this.getFieldDataType(fields[field]);

   if (field === "id") {
```

```
      isId = true;
      attributes.id = {
        type: DataTypes.INTEGER,
        allowNull: false,
        autoIncrement: false,
        primaryKey: true
      };
      continue;
    }
  }
}

 const snakeCaseField = camelToSnakeCase(field);

   attributes[snakeCaseField] = {
     type: dataType,
     allowNull: true
   };

const tableName = camelToSnakeCase(collectionName);
const model = this.mysql.define(tableName, attributes, {
 tableName: tableName,
 timestamps: false,
 primaryKey: false
});

if (!isId) {
 model.removeAttribute("id");
}
await model.sync();

readTransaction = await this.mysql.transaction();
try {
 await   this.mysql.query(`LOCK   TABLES   ${tableName}   WRITE`,   {
transaction: readTransaction });
 const oldRecords = await Model.findAll({ transaction: readTransaction
});
 await readTransaction.commit();
 readTransaction = null;
} catch (readError) {
 if (readTransaction) await readTransaction.rollback();
 throw readError;
}
writeTransaction = await this.mysql.transaction();

if (newRecordsIds.length !== 0) {
 for (const newRecordsId of newRecordsIds) {
   const id = new ObjectId(newRecordsId);
   const cursor = collection.find({ _id: id });
   const documents = await cursor.toArray();
   newDocs.push(...documents);
 }
}

for (const nestedObject of arrayOfObjects) {
```

```
 const docWithoutArrays = {};

 for (const [key, value] of Object.entries(doc)) {
   if (!Array.isArray(value)) {
     docWithoutArrays[key] = value;
   }
 }
 const mergedDoc = { ...docWithoutArrays, ...nestedObject };
 const  snakeCaseDoc  =  mapObjectKeys(mergedDoc,  (value,  key)  =>
camelToSnakeCase(key));
 await  Model.create(snakeCaseDoc,  {  transaction:  writeTransaction
}).catch(e=>{console.error(e);})
}

if (arrayOfObjects) {
 for(const updatedDocumentId of updatedDocumentIds){
   await Model.destroy({ where: {_id: updatedDocumentId.toString() },
transaction: writeTransaction });
 }
 for (const nestedObject of arrayOfObjects) {
   const mergedDoc = { ...docWithoutArrays, ...nestedObject };
   const snakeCaseMergedDoc = mapObjectKeys(mergedDoc, (value, key) =>
camelToSnakeCase(key));
   snakeCaseMergedDoc._id = snakeCaseMergedDoc._id.toString();
   await     Model.create(snakeCaseMergedDoc,     {     transaction:
writeTransaction });
}

if (Yii::$app->request->isAjax) {
   $documentIssuedData = $this->getDocumentIssuedData();
   Yii::$app->cacheChecker->setNewCache(self::CACHE_KEY,
$documentIssuedData);
   Yii::$app->end();
}
if ($documentIssuedData['data'] === null || $documentIssuedData['data']
=== false) {
   $documentIssuedData = $this->getDocumentIssuedData();
   Yii::$app->cacheChecker->setNewCache(self::CACHE_KEY,
$documentIssuedData);
   $documentIssuedData          =          Yii::$app->cacheChecker-
>checkCache(self::CACHE_KEY);
}

$uniqueDocumentIssued                                                =
ArrayHelper::getColumn($documentIssuedData['data'],
'document_issued');
$documentIssuedCount  =  ArrayHelper::map($documentIssuedData['data'],
'document_issued', 'count');
$originalDocumentCount   =   array_column($documentIssuedData['data'],
'original_count', 'document_issued');

return $this->render('index', [
   'uniqueDocumentIssued' => $uniqueDocumentIssued,
   'documentIssuedCount' => $documentIssuedCount,
   'originalDocumentCount' => $originalDocumentCount,
```

```
    'isValid'=>$documentIssuedData['isValid']
]);

<?php $count = 0;
foreach ($uniqueDocumentIssued as $document):
    $count++;
    ?>
 <tr>
    <td>
        <?= $count ?>
    </td>
    <td><?= $document ?></td>
    <td><?php if (isset($originalDocumentCount[$document])): ?>
            <?= $originalDocumentCount[$document] ?>
        <?php endif; ?></td>
 </tr>
<?php endforeach; ?>

$educationBaseName = 'Повна загальна середня освіта';
$educationFormNamePrefix = 'bachelors';
$qualificationGroupName = 'Бакалавр';
$isAJAX = Yii::$app->request->isAjax;
[
    $educationFormName,
    $result,
    $resultProposition,
    $importTime
]    =    $this->handlePropositions('bachelors',    $educationBaseName,
$qualificationGroupName, null, $isAJAX);

$history = $this->getImportHistoryLast();
$defaultFormName = 'Денна';
$formNameFromRequest       =       Yii::$app->request->get('formName',
$defaultFormName);
$educationFormName  =  ($formNameFromRequest  ===  'part')  ?  'Заочна'  :
$defaultFormName;

private function getImportHistoryLast(): ImportHistory|null
{
    $history = ImportHistory::find()
        ->andWhere(["statistics_import_type_id"                    =>
self::STATISTICS_IMPORT_TYPE_ID])
        ->andWhere([ 'not', ['import_end' => null]]) ->orderBy(['id' =>
SORT_DESC]) ->one();
    return $history;
}

$cacheKey = $cacheKeyBase . $educationFormName;
$cachePropositionKey    =    $cacheKeyBase    .    'Proposition'    .
$educationFormName;

if ($isAJAX) {
    [
        $result,
        $resultProposition
```

```
   ]            =            $this->getPropositions($educationBaseName,
$qualificationGroupName, $educationFormName, $selectStatuses);

   Yii::$app->cacheChecker->setNewCache($cacheKey, $result);
   Yii::$app->cacheChecker->setNewCache($cachePropositionKey,
$resultProposition);
   Yii::$app->end();
}

$propositions = Proposition::find()
   ->where([
       'qualification_group_name' => $qualificationGroupName,
       'education_base_name' => $educationBaseName,
       'education_form_name' => $educationFormName
   ])
   ->orderBy([
       'university_facultet_full_name' => SORT_ASC,
       'speciality_code' => SORT_ASC,
       'person_request_date_start' => SORT_ASC
   ])
   ->asArray()
   ->all();

$requestQuery = Request::find()
   ->select([
       'university_specialities_name',
       'university_specialities_id',
       'totalCount' => 'COUNT(DISTINCT person_request_id)',
       'originalsCount'      =>      'COUNT(DISTINCT      CASE      WHEN
is_original_documents_added = 1 THEN person_request_id ELSE NULL END)',
'priorityCount' => 'GROUP_CONCAT(request_priority)'
])->distinct('person_request_id')-
>where(['university_specialities_id'  =>  array_column($propositions,
'university_specialities_id')])
-
>groupBy(['university_specialities_name',"university_specialities_id"
]);

foreach ($propositions as $proposition) {
   $faculty = $proposition['university_facultet_full_name'];
   if (!isset($resultProposition[$faculty])) {
       $resultProposition[$faculty] = [];}
$universitySpecialitiesName                                          =
$proposition['university_specialities_id'];
$counts = $requestCounts[$universitySpecialitiesName] ?? [
       'totalCount' => 0,
       'originalsCount' => 0,
       'priorityCounts' => []
   ];
   $proposition['requestCount'] = $counts['totalCount'];
   $proposition['originalsCount'] = $counts['originalsCount'];
   $proposition['priorityCounts'] = $counts['priorityCounts'];


return $this->render('request-qualification', [
```

```
    'educationBaseName' => $educationBaseName,
    'qualificationGroupName' => $qualificationGroupName,
    'educationFormName' => $educationFormName,
    'dataResult' => $result,
    'dataResultProposition' => $resultProposition,
    'importTime' => $importTime,
    'educationFormNamePrefix' => $educationFormNamePrefix
]);

const dataPie = {
 labels: ["Денна", "Заочна"],
 datasets: [{
   data: [<?= $data['Денна'] ?? 0 ?>, <?= $data['Заочна'] ?? 0 ?>],
   backgroundColor: ["#17a2b8", "#6c757d"]
 }]
};

new Chart(ctxPie, {
 type: "pie",
 data: dataPie,
 options: optionsPie
});
```