НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
«___» травня 2026 р.	**«___»_травня** 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення підсистеми аналізу автоматизованої системи "Деканат»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент _____ **Ганна ВАЙГАНГ**

Керівник бакалаврської кваліфікаційної роботи _____ **Дмитро ШЕВЧЕНКО**

Консультант бакалаврської кваліфікаційної роботи к.т.н., доцент _____ **Белла ГОЛУБ**

Виконав _____ **Ковалінський Олександр**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«19» грудня 2025 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Ковалівському Олександру Владиславовичу

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

Програмне забезпечення підсистеми аналізу автоматизованої системи "Деканат"

затверджена наказом від «19» грудня 2025 р. № 3204 _«С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): навчальні відомості, особисті документи студентів.

Перелік питань, що підлягають дослідженню:

Дослідження методів збору, агрегації та візуалізації статистичних даних навчального процесу

Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Дмитро ШЕВЧЕНКО**

Консультант бакалаврської кваліфікаційної роботи _____ **Белла ГОЛУБ**

Завдання взяв до виконання _____ **Олександр КОВАЛІНСЬКИЙ**

ЗМІСТ

УМОВНІ ПОЗНАЧЕННЯ

СУБД- Система управління базами даних

MVC - Model-View-Controller

HTTP - HyperText Transfer Protocol

ВСТУП

Актуальність теми. У сучасному світі інформаційні технології відіграють критичну роль у функціонуванні всіх сфер людської діяльності, зокрема й у вищій освіті. Процеси управління навчальним закладом, такі як облік студентів, контроль успішності та ведення документації, потребують значних часових та людських ресурсів. Автоматизація цих процесів за допомогою систем типу АС «Деканат» дозволяє централізувати дані, проте більшість існуючих рішень орієнтовані на збереження та базову обробку інформації, тоді як їхні аналітичні можливості залишаються обмеженими. Актуальність даної роботи зумовлена необхідністю розробки підсистеми аналізу, яка забезпечить глибоку обробку накопичених даних. Це дозволить формувати статистику в розрізі фізичних осіб, аналізувати академічне навантаження, вести детальні персональні картки студентів та здійснювати облік академічних годин. Впровадження такого інструментарію мінімізує людський фактор і суттєво підвищить якість управлінських рішень у закладах освіти

Мета розробки. Метою роботи є проектування та реалізація програмного додатку підсистеми аналізу для інформаційна система(ІС) "Електронний деканат НУБіП України". Розроблена підсистема має забезпечувати формування статистичних звітів, автоматизацію збору та візуалізацію даних про навчальний процес, а також надавати зручний доступ до персональної інформації студентів. Досягнення мети передбачає виконання таких завдань: системний аналіз предметної області, визначення функціональних вимог, проєктування архітектури на базі патерна MVC та реалізацію модулів аналізу успішності й навантаження з використанням фреймворку Laravel та СУБД MySQL.

Апробація програмного продукту. Апробація результатів розробки здійснюється шляхом тестування програмного додатку та його впровадження у реальному навчальному середовищі. Отримані результати

використовуються для оцінки ефективності функціонування підсистеми та визначення напрямків її подальшого вдосконалення. Основні положення та практичні результати роботи можуть бути представлені на науково-практичних конференціях, а також опубліковані у спеціалізованих виданнях у вигляді тез чи статей. Це підтверджує відповідність розробки сучасним вимогам до освітніх інформаційних систем та її практичну значущість для адміністративного персоналу ЗВО.

Структура пояснювальної записки. Пояснювальна записка до даної роботи має структурований характер і складається з кількох взаємопов'язаних розділів. Загальний обсяг записки становить 70 сторінок та 20 використаних джерел. У першому розділі наведено аналіз предметної області, розглянуто існуючі рішення та обґрунтовано необхідність розробки підсистеми аналізу. Другий розділ присвячений проєктуванню програмного забезпечення, де описано архітектуру системи, структуру бази даних та основні компоненти додатку. У третьому розділі представлено реалізацію програмного продукту, включаючи опис функціональних можливостей, алгоритмів роботи та інтерфейсу користувача. Четвертий розділ містить результати тестування, аналіз ефективності та оцінку якості розробленого програмного забезпечення.

У додатках наведено допоміжні матеріали, зокрема фрагменти програмного коду, схеми бази даних, діаграми та приклади інтерфейсів користувача. Така структура дозволяє повно і послідовно розкрити всі аспекти розробки програмного додатку та забезпечує зручність сприйняття матеріалу.

Таким чином, розробка програмного забезпечення підсистеми аналізу автоматизованої системи «Деканат» є важливим і актуальним завданням, спрямованим на підвищення ефективності управління навчальним процесом та вдосконалення інформаційного забезпечення закладів освіти.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметною областю даної роботи є процеси управління навчальною діяльністю закладу вищої освіти, що реалізуються в межах автоматизованої системи «Деканат». Дана система призначена для забезпечення обліку студентського контингенту, контролю навчального процесу, формування академічного навантаження викладачів, а також зберігання та обробки інформації про результати навчання.

У традиційних умовах значна частина цих процесів виконується вручну або з використанням розрізнених програмних засобів, що ускладнює доступ до інформації, підвищує ймовірність помилок та знижує ефективність управління. Впровадження автоматизованої системи «Деканат» дозволяє централізувати дані та оптимізувати роботу адміністративного персоналу, однак для повноцінного використання її можливостей необхідною є наявність підсистеми аналізу.

Підсистема аналізу є важливою складовою інформаційної системи, оскільки вона забезпечує не лише зберігання даних, а й їх обробку, узагальнення та представлення у зручному для користувача вигляді. Основним призначенням підсистеми є формування статистичної інформації, що використовується для прийняття управлінських рішень.

У межах даної предметної області виділяються основні об’єкти, до яких належать: студенти, викладачі, навчальні дисципліни, спеціальність, освітні програми, академічні групи, навчальні плани, а також академічне навантаження. Кожен із цих об’єктів має набір характеристик і взаємодіє з іншими об’єктами системи. Наприклад, студент належить до певної спеціальності, академічної групи, вивчає визначений перелік дисциплін і має індивідуальні показники успішності та кількість академічних годин. Викладач,

у свою чергу, має закріплене за ним навчальне навантаження, яке включає проведення лекцій, практичних і лабораторних занять.

Особливу роль у системі відіграє персональна картка. Це передбачає можливість отримання узагальненої та деталізованої інформації про кожного студента, включаючи їхні персональні дані, академічні показники та участь у навчальному процесі. Такий підхід дозволяє здійснювати індивідуальний контроль та оцінювання діяльності учасників освітнього процесу.

Ще одним важливим аспектом є аналіз академічного навантаження, який дає змогу оцінити розподіл навчальних годин, виявити перевантаження або недовантаження, а також оптимізувати планування навчального процесу. Крім того, значна увага приділяється веденню персональної картки студента, яка містить повну інформацію про його навчання, включаючи перелік дисциплін, оцінки, кількість відпрацьованих академічних годин та інші показники.

Облік академічних годин є важливим елементом предметної області, оскільки він дозволяє контролювати виконання навчального плану як з боку студентів, так і викладачів. На основі цих даних формується статистика, яка може використовуватись для аналізу ефективності навчального процесу та прийняття управлінських рішень.

Таким чином, предметна область характеризується наявністю великої кількості взаємопов’язаних об’єктів і процесів, що потребують систематизації, аналізу та ефективного управління. Розробка підсистеми аналізу автоматизованої системи «Деканат» дозволяє забезпечити комплексний підхід до обробки даних і створити інструменти для отримання актуальної та достовірної інформації, необхідної для підвищення якості управління навчальним процесом.

1.2 Опис предметної області

1.2.1 Use case діаграма. Аналіз функціональних вимог та бізнес-логіки системи доцільно розпочати з побудови діаграми прецедентів. Вона наочно

демонструє ролі користувачів (акторів) та їхні сценарії взаємодії з програмним забезпеченням. Відповідна UML-діаграма наведена на рис. 1.

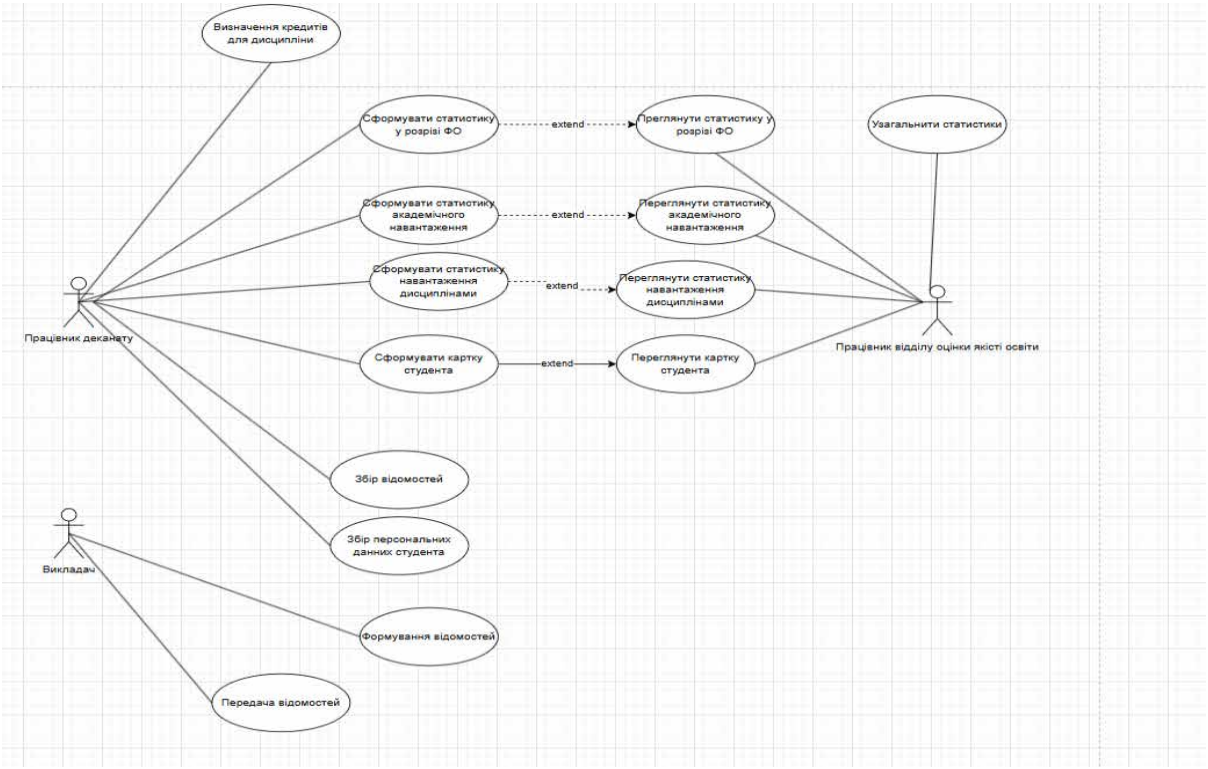


Рис. 1 Діаграма прецедентів для підсистеми аналізу автоматизованої системи "Деканат"

Основні актори системи

Працівник деканату Виконує завдання, пов’язані з формуванням та обробкою статистики, визначенням кредитів для дисциплін, збором відомостей та створенням персональної карти студента. Його діяльність спрямована на забезпечення обліку студентського контингенту та контроль навчального процесу.

Працівник відділу оцінки якості освіти Має доступ до перегляду та узагальнення статистичних даних: академічного навантаження, дисциплін, карток студентів. Його функції орієнтовані на аналіз та оцінку ефективності освітнього процесу.

Викладач Збирає персональні дані студентів, формує відомості та передає їх у систему. Це забезпечує актуальність інформації про результати навчання та дозволяє контролювати виконання навчальних планів.

Основні процеси предметної області

Облік студентів та їхніх даних:

- персональна картка студента;
- відомості про успішність та академічні години.

Формування та аналіз статистики:

- навантаження викладачів;
- розподіл дисциплін;
- узагальнення даних для управлінських рішень.

Контроль навчального процесу:

- перевірка відповідності навчальних планів;
- оцінка якості освітніх послуг;
- оптимізація академічного навантаження.

Таким чином, розроблена діаграма прецедентів дозволяє чітко розмежувати ролі користувачів та визначити функціональні межі підсистеми аналізу. Взаємодія між викладачем, працівником деканату та відділом якості освіти забезпечує наскрізний цикл обробки інформації: від первинного збору персональних даних та успішності до стратегічного аналізу й узагальнення статистики.

Визначені основні процеси предметної області підтверджують, що впровадження даної підсистеми дозволить автоматизувати облік студентського контингенту, оптимізувати розподіл академічного навантаження та створити надійне підґрунтя для прийняття обґрунтованих управлінських рішень щодо підвищення якості освітнього процесу. Спроєктована модель використання виступає базою для подальшої технічної реалізації архітектури програмного продукту.

1.2.2 UML Діаграма послідовності. Наступним етапом проєктування є розробка та аналіз діаграми послідовності, яка деталізує логіку взаємодії об'єктів системи в часі. Ця модель ілюструє хронологію обміну повідомленнями під час виконання основних процесів. Графічне представлення діаграми наведено на рис. 2.

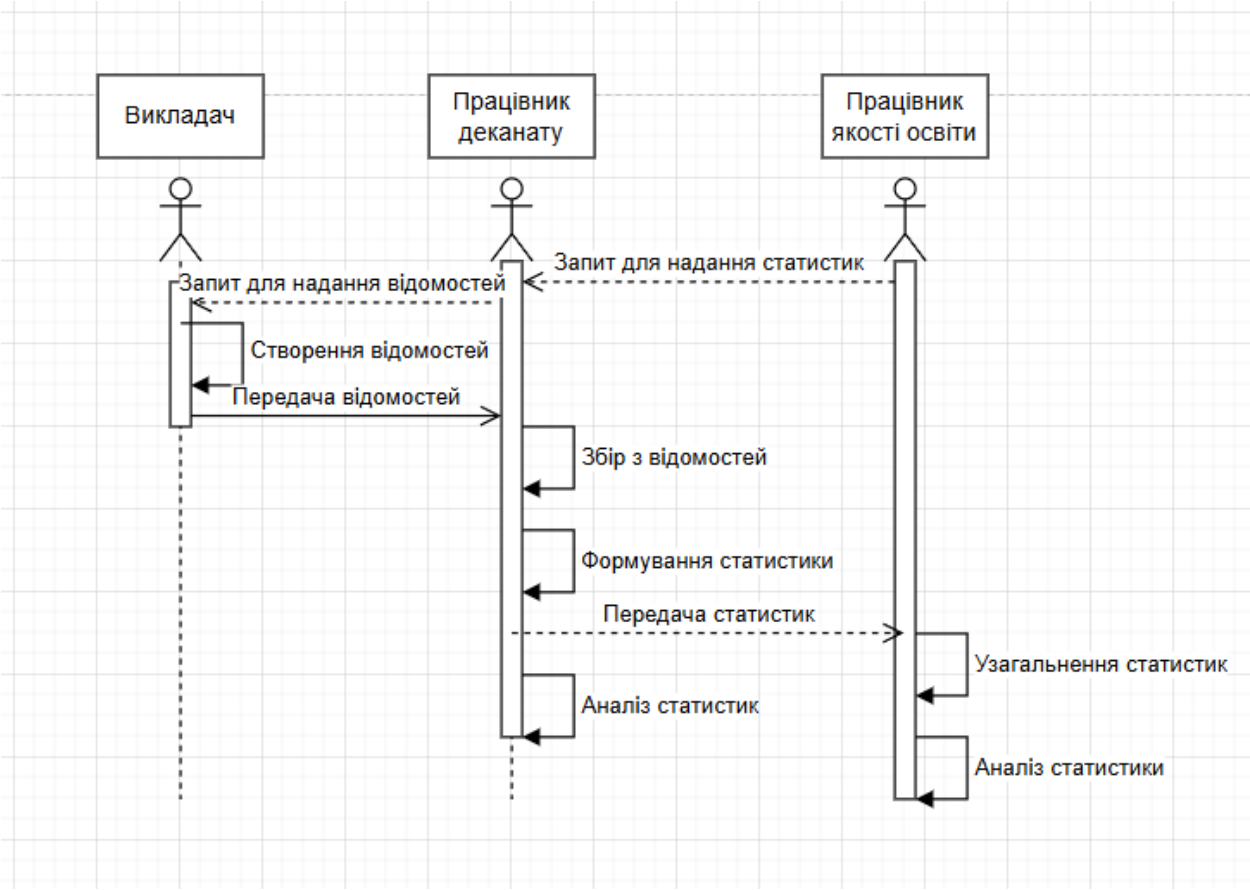


Рис. 2 Діаграма послідовності

Представлена діаграма послідовності, яка ілюструє динаміку взаємодії між основними акторами та системними компонентами підсистеми аналізу АС «Деканат». Вона відображає впорядкований у часі потік повідомлень, що забезпечує трансформацію первинних даних у аналітичні звіти.

Деталізація взаємодії учасників

Працівник відділу якості освіти (Ініціатор). Процес активується, коли працівник відділу ініціює запит на моніторинг запит для надання статистики. Ця дія спрямовується до працівника деканату як вимога надати актуальні показники за певний період або дисципліну. Лінія життя цього актора залишається активною протягом усього циклу, очікуючи на фінальний консолідований звіт.

Працівник деканату. Отримавши запит від відділу якості, працівник деканату виступає як медіатор та агрегатом інформації. Він активує ланцюжок збору даних, надсилаючи відповідні повідомлення запит відомостей до

викладачів. Після отримання інформації він виконує операції збір даних, обробка статистики та формування вибірки. Оброблений масив даних передається назад до працівника відділу якості освіти у вигляді структурованої звітності.

Викладач. На цій діаграмі викладач виконує роль джерела первинних даних. Він активується лише у відповідь на запит деканату. Викладач виконує дії створити відомість та передати сформовані дані. Після того, як інформація про успішність та навантаження передана, лінія життя викладача переходить у стан очікування.

Аналіз архітектурної логіки

Ініціація процесу з боку відділу якості освіти створює модель «тягнучої» (pull) системи, де дані збираються під конкретну аналітичну потребу. Це забезпечує наступні переваги:

- Цільовий збір даних: Система не перевантажується зайвими обчисленнями, оскільки обробка статистики відбувається лише за запитом контролюючого органу.
- Вертикальна прозорість: Діаграма наочно демонструє ієрархічну підпорядкованість: відділ якості стасує завдання деканату, а деканат — викладачам.
- Валідація на кожному рівні: Перш ніж потрапити до фінального звіту, дані проходять подвійний контроль — спочатку викладачем при введенні, потім деканатом при агрегації.
- Ефективність управління: Такий підхід дозволяє проводити оперативний аудит освітнього процесу в будь-який момент часу, що критично важливо для моніторингу стандартів освіти.

Побудована модель послідовності підтверджує, що підсистема аналізу є ефективним інструментом підтримки прийняття рішень на рівні управління якістю освіти. Чіткий розподіл обов’язків та послідовна передача повідомлень гарантують, що аналітичні висновки ґрунтуються на перевірених та актуальних даних. Дана модель слугує основою для розробки сервісних шарів

у Laravel, які будуть відповідати за обробку запитів від різних типів користувачів.

1.2.3 UML Діаграма активності. Подальший етап проєктування передбачає моделювання алгоритму виконання ключових бізнес-процесів. Для детального аналізу логіки, розгалужень та покрокової послідовності дій під час формування статистики ФО було розроблено діаграму діяльності. Відповідну UML-модель, що ілюструє потоки управління в цьому процесі, наведено на рисунку 3.

Діаграма активностей (Activity Diagram) призначена для візуалізації бізнес-процесів та алгоритмічної логіки підсистеми аналізу АС «Деканат». Вона дозволяє детально розглянути послідовність дій, умови переходів та паралельні процеси, що виникають під час моніторингу якості освіти. У даній моделі керуючий потік розпочинається з ініціативи Працівника відділу якості освіти.

Опис ключових етапів та вузлів керування:

1) Початок процесу та ініціація запиту. Потік керування активується дією «Формування запиту на статистику». Працівник відділу якості визначає параметри необхідного аналізу (період, факультет, перелік дисциплін). Це створює вхідний сигнал для наступного вузла активності, що належить до компетенції деканату.

2) Діяльність на рівні деканату. Працівник деканату отримує запит і переходить до виконання активності «Формування запиту на отримання відомостей». На цьому етапі діаграма відображає розгалуження (Decision Node), де перевіряється наявність або актуальність даних у системі. Якщо дані потребують оновлення, потік спрямовується до виконавчого рівня.

3) Виконавчий рівень (Викладач). Активність викладача полягає у «Введенні та верифікації даних». Це включає створення електронних відомостей, заповнення результатів успішності та фіксацію академічних

годин. Після завершення цієї дії потік повертається до деканату для консолідації.

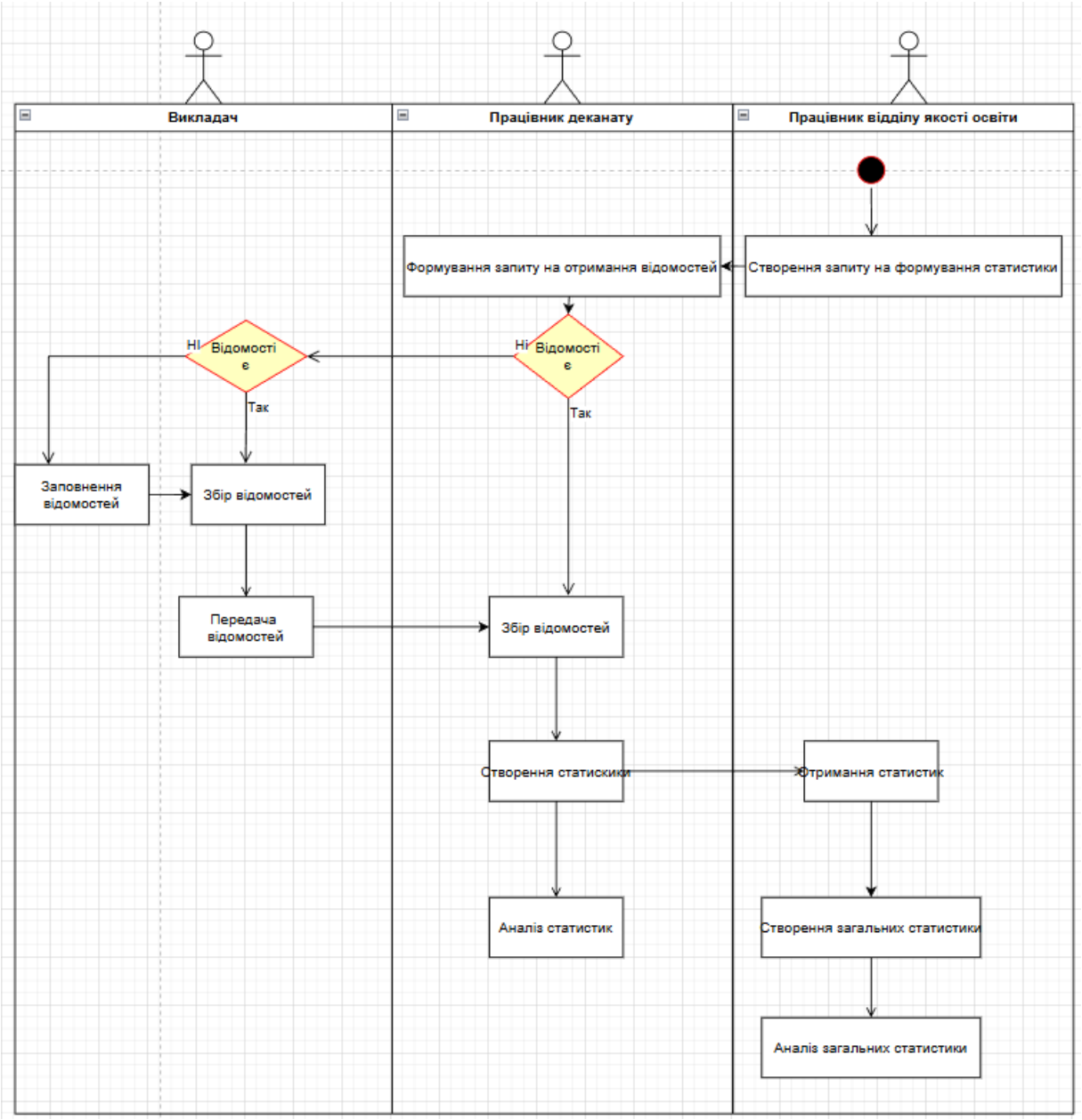


Рис.3 Діаграма активності

4) Агрегація та обробка інформації. Після збору всіх необхідних відомостей система виконує комплекс активностей: «Створення статистик». Ці дії виконуються автоматизовано за підтримки працівника деканату, що забезпечує перетворення «сирих» даних у структуровану аналітику.
5) Завершення та прийняття рішень. Фінальна активність повертається до працівника відділу якості освіти — «Узагальнення даних та оцінка

якості». На основі отриманого звіту приймається рішення про успішність проходження етапу моніторингу, що є кінцевим вузлом (Final Node) даної діаграми.

Логічні переваги моделі активностей:

- Динаміка процесів: Діаграма наочно демонструє, як запит від вищої ланки управління трансформується у конкретні дії на місцях.
- Чіткі зони відповідальності: Завдяки використанню «плавальних доріжок» (swimlanes) для кожного актора, чітко розмежовано, хто відповідає за збір, хто за обробку, а хто за фінальний аналіз.
- Обробка виключень: Модель передбачає можливість повернення на попередні етапи у разі виявлення неточностей або відсутності даних.
- Орієнтація на результат: Кожна дія в ланцюгу є необхідною умовою для досягнення мети — отримання об'єктивного аналізу якості освітнього процесу.

Побудована UML-діаграма активностей підтверджує логічну завершеність і несуперечливість бізнес-процесів підсистеми аналізу. Вона демонструє ефективну вертикальну інтеграцію між відділом якості, деканатом та викладачами. Дана модель слугує безпосереднім алгоритмічним завданням для розробки бізнес-логіки додатка на базі PHP/Laravel, визначаючи послідовність виклику методів у контролерах системи.

1.3 Аналіз вимог до підсистеми аналізу інформаційна система(ІС) "Електронний деканат НУБіП України"

Для успішної реалізації підсистеми аналізу необхідно чітко розмежувати функціональні можливості, які очікують користувачі, та технічні обмеження системи.

1.3.1 Функціональні вимоги. Функціональні вимоги визначають дії, які система повинна виконувати для кожного актора:

Для працівника деканату:

- Система повинна забезпечувати формування статистичних звітів у розрізі фізичних осіб (студентів/викладачів).
- Можливість розрахунку та відображення академічного навантаження за дисциплінами.
- Функція створення та редагування персональних карток студентів.
- Визначення та облік кредитів для кожної навчальної дисципліни.

Для працівника відділу якості освіти:

- Доступ до перегляду узагальненої статистики по всьому контингенту.
- Інструменти для аналізу ефективності освітнього процесу на основі агрегованих даних.
- Можливість експорту аналітичних звітів для прийняття управлінських рішень.

1.3.2 Нефункціональні вимоги. Ці вимоги визначають якісні характеристики системи та технологічні обмеження:

- Надійність та безпека:
  - Забезпечення цілісності даних при багатокористувацькому доступі через СУБД MySQL.
  - Забезпечення механізмів безпеки для захисту від SQL-ін’єкцій та CSRF-атак.
  - Розмежування прав доступу між різними ролями (деканат, відділ якості).
- Продуктивність:
  - Швидкість формування статистичних звітів не повинна перевищувати кількох секунд навіть при великих обсягах даних.
  - Оптимізація запитів до бази даних для мінімізації навантаження на сервер.
- Зручність використання :
  - Адаптивний веб-інтерфейс, реалізований за допомогою сучасних фронтенд-технологій.

- Інтуїтивно зрозуміла навігація між розділами статистики та персональними картками.

- Архітектурні вимоги:
  - Дотримання клієнт-серверної моделі архітектури.
  - Використання паттерну MVC (Model-View-Controller) для забезпечення масштабованості коду.

1.3.3 Специфікація обмежень. Система повинна коректно функціонувати в середовищі PHPMyAdmin (MySQL). Програмний код має бути структурованим відповідно до стандартів об’єктно-орієнтованого програмування. Система має бути сумісною з ОС Linux (враховуючи особливості розгортання).

1.4 Огляд інформаційних джерел та існуючих рішень

1.4.1 Огляд нормативно-правових та інформаційних джерел. Розробка підсистеми аналізу базується на глибокому вивченні теоретичної бази та нормативних актів:

- Законодавча база: Закон України «Про вищу освіту» та «Про захист персональних даних».
- Технічна документація: Офіційні специфікації фреймворку Laravel та СУБД MySQL, які є технологічним фундаментом розробки.

1.4.2 Огляд існуючих автоматизованих систем (аналогів). В Україні для управління освітнім процесом найчастіше використовуються такі системи:

- АСУ «ВНЗ» (Центр «Студсервіс»): комплексна система, що охоплює всі рівні управління (від абітурієнта до випускника). Має потужну систему захисту та інтеграцію з ЄДЕБО, проте її аналітичні модулі часто є закритими для кастомізації під конкретні потреби кафедри чи деканату.
- ЄДЕБО (Єдина державна електронна база з питань освіти): загальнодержавна система для збору звітності. Вона є обов'язковою, але не призначена для глибокого щоденного аналізу внутрішніх процесів факультету в реальному часі.

- Moodle : системи управління навчанням (LMS), які фокусуються на контенті та оцінках, але мають обмежений функціонал для адміністрування навантаження викладачів та ведення персональних карток у розрізі деканату

Таблиця 1

Порівняння аналогів

Функціональна можливість	АСУ «ВНЗ»	Moodle	Розроблювана підсистема
Облік студентів	+	-	+
Ведення відомостей	+	+	+
Гнучка аналітика успішності	+	+	+
Розрахунок навантаження	+	-	+
Веб-орієнтований інтерфейс	-	+	+

1.4.3 Порівняльний аналіз та обґрунтування розробки. Незважаючи на наявність потужних систем, існують суттєві обмеження:

1. Орієнтація на збереження, а не аналіз: Більшість систем орієнтовані на збереження та базову обробку даних, тоді як аналітичні можливості залишаються обмеженими.
2. Відсутність персоналізованої аналітики: Наявні рішення часто не дозволяють формувати статистику у розрізі окремих фізичних осіб для глибокого моніторингу.
3. Людський фактор: Ручне введення даних у розрізнених програмних засобах підвищує ймовірність помилок.

1.5 Постановка завдання

На основі проведеного аналізу предметної області та критичного огляду існуючих аналогів, у даному підрозділі формулюється технічне завдання на розробку. Метою роботи є проєктування та програмна реалізація підсистеми аналізу даних для автоматизованої системи управління (АС) «Деканат», яка забезпечить автоматизацію процесів збору, інтелектуальної обробки та візуалізації статистичної інформації щодо навчального процесу, академічної успішності студентів та навантаження викладацького складу.

Для досягнення поставленої мети необхідно вирішити наступний комплекс наукових та практичних завдань:

1. Системний аналіз: дослідити структуру даних та існуючі бізнес-процеси АС «Деканат», виявити «вузькі місця» у процедурах формування відомостей та аналізу звітності.
2. Вимоги до ПЗ: розробити специфікацію функціональних вимог (збір даних, генерація звітів, ведення карток) та нефункціональних вимог (безпека, продуктивність, масштабованість) для ролей «Працівник відділу якості», «Працівник деканату» та «Викладач».
3. Проєктування архітектури: розробити архітектурне рішення на базі об’єктно-орієнтованого підходу з використанням патерна MVC (Model-View-Controller) для чіткого розділення бізнес-логіки, інтерфейсу та даних.

4. Моделювання даних: спроєктувати реляційну базу даних у середовищі СУБД MySQL (phpMyAdmin), забезпечивши цілісність зв’язків між сутностями «Студент», «Викладач», «Дисципліна» та «Навантаження».
5. Програмна реалізація: використовуючи фреймворк PHP Laravel, розробити ключові модулі системи:
    - Модуль аналітики ФО: генерація показників середнього бала та ідентифікація «груп ризику».
    - Модуль персональних карток: цифровий профіль студента з повною історією успішності.
    - Модуль академічного навантаження: автоматизований розрахунок та облік годин і кредитів.

Предметом дослідження є методи, алгоритми та інструментальні засоби розробки веб-орієнтованої підсистеми аналізу на базі фреймворку Laravel.

Таблиця 2

Характеристика інформаційних потоків підсистеми

Тип даних	Склад та опис
Вхідні дані	Персональні профілі студентів/викладачів, навчальні плани (кредити, години), розподіл навантаження за кафедрами, результати поточного та підсумкового контролю.
Вихідні дані	Аналітичні звіти у розрізі фізичних осіб, динамічні графіки успішності, сформовані цифрові персональні картки, зведені таблиці виконання академічного плану.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Оскільки розроблювана підсистема аналізу є функціональним доповненням до вже впровадженої в навчальний заклад автоматизованої системи(Рисунок 4) (АС) «Деканат», у роботі не передбачається створення нової архітектури бази даних «з нуля». Натомість, використовується вже існуюча логічна модель даних АС «Деканат», що забезпечує прямий доступ до актуальної інформації про навчальний процес без необхідності надлишкового дублювання даних.

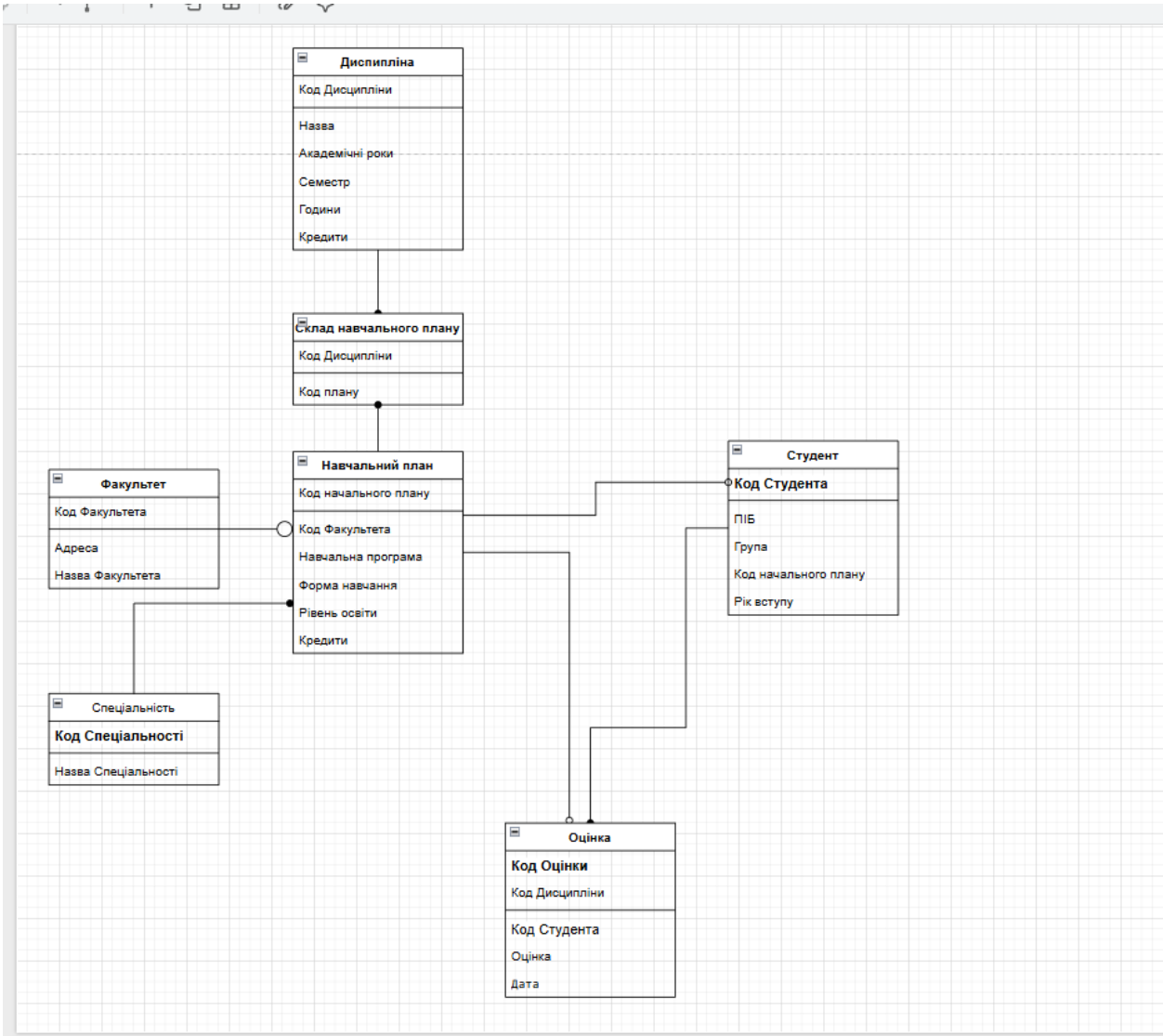


Рис.4 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних відображає структуру бази даних, яка забезпечує зберігання та обробку інформації про навчальний процес, студентський контингент та результати навчання. Модель складається з семи взаємопов'язаних сутностей, що дозволяють реалізувати функції аналізу успішності та формування персональних карток

Опис сутностей та їх атрибутів:

1) Дисципліна описує навчальні предмети.
    a. Атрибути: Код Дисципліни (PK), Назва, Академічні роки, Семестр, Години, Кредити.
2) Навчальний план : визначає освітню траєкторію для певної програми.
    a. Атрибути: Код навчального плану (PK), Код Факультету (FK), Навчальна програма, Форма навчання, Рівень освіти, Кредити.
3) Склад навчального плану: проміжна сутність, що реалізує зв'язок між дисциплінами та планами.
    a. Атрибути: Код Дисципліни (FK), Код плану (FK).
4) Студент: містить персональні та навчальні дані особи.
    a. Атрибути: Код Студента (PK), ПІБ, Група, Код навчального плану (FK), Рік вступу.
5) Оцінка : центральна сутність для аналітичної обробки результатів навчання.
    a. Атрибути: Код Оцінки (PK), Код Дисципліни (FK), Код Студента (FK), Оцінка, Дата.
6) Факультет : описує організаційну структуру ЗВО.
    a. Атрибути: Код Факультету (PK), Назва Факультету, Адреса.
7) Спеціальність : містить перелік напрямів підготовки.
    a. Атрибути: Код Спеціальності (PK), Назва Спеціальності

Зв’язки між об’єктами системи:

- Факультет — Навчальний план та Спеціальність: встановлено зв'язки типу «один-до-багатьох» (1:N). Один факультет може мати декілька спеціальностей та затверджених навчальних планів.
- Навчальний план — Дисципліна: реалізовано через сутність «Склад навчального плану». Це дозволяє гнучко формувати перелік предметів для кожної освітньої програми.
- Навчальний план — Студент: кожен студент закріплений за одним навчальним планом, що визначає його програму навчання (зв'язок 1:N).
- Студент — Оцінка — Дисципліна: сутність «Оцінка» пов’язує конкретного студента з результатом вивчення певної дисципліни. Це є ключовим вузлом для формування статистичних звітів у розрізі фізичних осіб та аналізу академічної успішності, що відповідає меті розробки підсистеми.

2.2 Вибір СУБД

Для реалізації підсистеми аналізу було обрано реляційну систему управління базами даних MySQL. Цей вибір обумовлений декількома критичними факторами, що випливають із завдань проєкту та технічного середовища розгортання:

- Інтеграція з існуючою інфраструктурою: Оскільки розроблювана підсистема є функціональним доповненням до вже впровадженої автоматизованої системи «Деканат», ключовою вимогою є повна сумісність із наявними даними. Використання MySQL дозволяє уникнути складних процесів міграції або дублювання інформації, забезпечуючи прямий доступ до таблиць існуючої бази.
- Робота з готовою архітектурою: У даній роботі не передбачається проєктування структури даних «з нуля». Натомість, підсистема аналізу використовує вже сформовану логічну модель даних АС «Деканат», що включає сутності студентів, викладачів, оцінок та навчальних планів. Це

дозволяє зосередити ресурси на розробці алгоритмів аналізу та візуалізації статистики, а не на адмініструванні сховища.

- Сумісність із технологічним стеком: MySQL є стандартом для веб-розробки на базі фреймворку Laravel, який використовується для реалізації серверної логіки. Вбудована підтримка ORM Eloquent забезпечує ефективну та безпечну взаємодію з існуючими таблицями через об’єктно-орієнтовані моделі.
- Зручність адміністрування: Для керування базою даних та виконання налагоджувальних запитів використовується середовище phpMyAdmin. Це забезпечує наочність структури та дозволяє оперативно перевіряти цілісність зв’язків між сутностями під час тестування модулів аналітики.
- Продуктивність та надійність: СУБД MySQL забезпечує високу швидкість обробки складних аналітичних запитів (наприклад, при розрахунку середніх балів GPA або агрегації академічного навантаження), що відповідає нефункціональним вимогам щодо ` швидкодії системи.

Таким чином, використання MySQL як бази даних для підсистеми аналізу є логічним та технічно обґрунтованим рішенням, яке базується на необхідності безшовної інтеграції з існуючою екосистемою АС «Деканат».

2.3 Опис структури та зв'язків існуючої БД

Оскільки підсистема аналізу інтегрується в уже розгорнуте середовище, її стабільність залежить від розуміння реляційних зв'язків існуючої СУБД MySQL. Структура побудована за нормалізованою формою, що забезпечує цілісність даних навчального процесу.

1) Рівень організаційної структури (faculty)

Таблиця faculty є базовим довідником.

Зв’язки: Виступає батьківською таблицею для сутностей person та curriculum.

Роль у системі: Дозволяє фільтрувати всі аналітичні звіти за ієрархічним принципом (університет → факультет).

2) Рівень суб’єктів навчального процесу (person)

Таблиця person містить реєстр користувачів.

Ключові поля: id (PK), faculty_id (FK), role_id (FK).

1:N до faculty: Кожна особа закріплена за певним факультетом.

1:N до education_card: Одна особа (як студент) може мати одну або декілька навчальних карток (наприклад, при паралельному навчанні або отриманні різних ступенів).

3) Рівень планування та дисциплін (curriculum, discipline, curriculum_discipline)

Цей блок реалізує складну логіку навчальних планів:

curriculum (Навчальний план): Визначає параметри навчання для потоку (напр., «Бакалавр, Комп'ютерна інженерія, 2023 р.в.»). Має зв'язок 1:N з education_card.

discipline (Дисципліна): Автономний довідник предметів.

curriculum_discipline (Зв'язок плану з предметами):

Тип зв'язку: Реалізує зв'язок M:N (багато-до-багатьох) між планами та дисциплінами.

Ключові поля: curriculum_id (FK), discipline_id (FK).

Додаткові атрибути: Саме тут зберігаються метадані для аналізу: кількість кредитів ECTS, семестр вивчення та тип контролю (екзамен/залік). Це дозволяє підсистемі аналізу обчислювати сумарне навантаження.

4) Рівень контингенту (education_card)

Таблиця education_card є сполучною ланкою між фізичною особою та її освітньою траєкторією.

Ключові поля: id (PK), person_id (FK), curriculum_id (FK).

Роль у зв’язках: Це «точка входу» для аналізу студента. Вона пов'язує конкретного person із набором дисциплін через curriculum. Всі звіти «Статистика за студентом» базуються на ідентифікаторі навчальної картки.

5. Рівень результативності (assessment)

Таблиця assessment накопичує результати діяльності.

Ключові поля: id (PK), education_card_id (FK), discipline_id (FK).

Зв’язки:

N:1 до education_card: Багато оцінок належать одній картці студента.

N:1 до discipline: Багато оцінок з різних планів відповідають одній дисципліні.

Аналітична цінність: Це найбільш динамічна таблиця. На її основі підсистема аналізу вираховує GPA (середній бал), формує рейтинги успішності та визначає «групи ризику».

Для формування підсумкової аналітики (наприклад, середнього балу по факультету) підсистема виконує ланцюговий запит:

1. Визначається faculty.
2. Через person та education_card знаходяться всі студенти цього факультету.
3. Через assessment зчитуються бали за дисципліни, що входять у відповідні curriculum_discipline.

Така архітектура вже закладена в АС «Деканат», що дозволяє підсистемі аналізу використовувати SQL-запити з операторами JOIN без необхідності дублювання даних, працюючи з «живим» набором інформації.

2.4 Створення міграцій

2.4.1 Що таке міграції. Міграції подібні до контролю версій для вашої бази даних, що дозволяє вашій команді визначати та ділитися визначенням схеми бази даних програми. Якщо вам коли-небудь доводилося казати товаришу по команді вручну додати стовпець до їхньої локальної схеми бази

даних після отримання змін із системи контролю версій, ви зіткнулися з проблемою, яку вирішує міграція бази даних.

Laravel забезпечує агностичну підтримку для створення та маніпулювання таблицями у всіх підтримуваних системах баз даних Laravel. Як правило, міграції використовують цей фасад для створення та зміни таблиць і стовпців бази даних.

2.4.2 Створення міграцій(faculty). Команда Artisan make:migration для створення міграції бази даних. Нова міграція буде розміщена у каталозі database/migrations. Кожне ім'я файлу міграції містить позначку часу, яка дозволяє Laravel визначити порядок міграцій:

php artisan make:migration create_faculties_table

Далі було в методі up логіку створення таблиці в базі данних Рис.5

```
public function up(): void
{
    Schema::create( table: 'faculties', function (Blueprint $table) {
        $table->id();
        $table->string( column: 'FacultyName');
        $table->string( column: 'FacultyNameEn');
        $table->string( column: 'site')->nullable( value: 'true');
        $table->string( column: 'DecanLastName');
        $table->string( column: 'DecanFirstName');
        $table->string( column: 'DecanMiddleName');
        $table->enum( column: 'IsItFaculty', allowed: ['0','1'])->default( value: '0');
        $table->timestamps();
    });
}
```

Рис.5 Готова faculties міграція

Побудована міграція, представлена на рис. 5, базується на використанні вбудованого в Laravel конструктора схем (Schema Builder), який надає базовий клас Schema для маніпулювання таблицями бази даних СУБД MySQL незалежно від використання сирого SQL-коду.

Програмний код міграції містить два ключових методи, які забезпечують життєвий цикл структури таблиці: «up()» та «down()».

Функціональне призначення методів міграції:

1. **Метод «up()»** виконується під час запуску команди php artisan migrate. Усередині цього методу викликається метод Schema::create(), який приймає два аргументи: назву таблиці в множині (faculties) та анонімну функцію (замикання), що приймає об'єкт класу Blueprint. Об'єкт $table використовується для визначення стовпців, їхніх типів та атрибутів.
2. **Метод «down()»** відповідає за зворотну операцію (відкат міграції) за допомогою команди php artisan migrate:rollback. Він містить інструкцію Schema::dropIfExists('faculties'), яка безпечно видаляє таблицю з бази даних, якщо вона існує, що дозволяє підтримувати консистентність структури під час розробки.

2.4.3 Специфікація полів таблиці faculties. У межах проєктування структури даних підсистеми аналізу, для сутності «Факультет» у методі up() було визначено такий набір полів:

- $table->id(); — створює первинний ключ (Primary Key) з назвою id. У середовищі СУБД MySQL цей метод автоматично генерує поле типу BIGINT з атрибутами UNSIGNED (беззнакове) та AUTO_INCREMENT (автоматичне нарощування значення для кожного нового запису). Це гарантує унікальність ідентифікатора кожного факультету та високу швидкість індексації.
- $table->boolean('IsItFaculty') - створює поле isItFaculty для збереження статуса факультету, тип данних boolean.
- $table->string('name'); — створює поле name для збереження повної назви факультету. Фреймворк Laravel транслює цей метод у тип даних VARCHAR із довжиною за замовчуванням 255 символів. Цього обсягу достатньо для коректного збереження офіційних назв факультетів (наприклад, *«Факультет інформаційних технологій»*). Оскільки додаткових модифікаторів не вказано, поле отримує атрибут NOT NULL, що робить його обов'язковим для заповнення.

2.5 ORM Eloquent

2.5.1 Призначення та можливості Eloquent. Laravel включає Eloquent, об'єктно-реляційний маппер (ORM), який спрощує взаємодію з базою даних. Під час використання Eloquent кожна таблиця бази даних має відповідну «Модель», яка використовується для взаємодії з цією таблицею. Окрім отримання записів з таблиці бази даних, моделі Eloquent також дозволяють вставляти, оновлювати та видаляти записи з таблиці. Команда для створення моделі php artisan make:model Faculty. (Результат рис.6)

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

class Faculty extends Model
{
    use HasFactory;

    protected $table = 'faculties';
    protected $guarded = [];
    public function users(): HasMany
    {
        return $this->HasMany( related: User::class);
    }
    public function specialities(): HasMany
    {
        return $this->HasMany( related: Speciality::class);
    }
}
```

Рис.6 Демонстрація моделі

Побудована модель Faculty є яскравим прикладом реалізації архітектурного патерна Active Record, який покладено в основу Eloquent ORM.

Згідно з цим патерном, кожен екземпляр класу моделі відповідає конкретній таблиці в бази даних, а сам клас надає статичні та динамічні методи для виконання CRUD-операцій (Create, Read, Update, Delete) без необхідності написання прямих SQL-запитів. Це дозволяє абстрагувати логіку збереження даних від бізнес-логіки додатку.

2.5.2 Технічний розбір структури класу моделі Faculty: Наслідування від базового класу Model. Клас Faculty наслідує базовий клас Illuminate\Database\Eloquent\Model. Завдяки цьому механізму ООП, створювана модель автоматично отримує потужний функціонал для роботи з реляційними даними: методи для побудови запитів (where(), orderBy()), методи вибірки (find(), all()), а також можливість відстеження стану об'єкта (чи був він змінений) перед збереженням.

Використання трейту use HasFactory. Підключення трейту Illuminate\Database\Eloquent\Factories\HasFactory забезпечує інтеграцію моделі з фабриками даних (Factories). У контексті інженерії програмного забезпечення це критично важливо для автоматизації модульного тестування (Unit Testing) та початкового наповнення бази даних тестовими або дефолтними значеннями (Database Seeding). Фабрика дозволяє генерувати великі обсяги коректних екземплярів сутності «Факультет» для перевірки навантажувальної стійкості підсистеми аналізу. Явне визначення асоційованої таблиці. protected $table = 'faculties'; За замовчуванням Eloquent використовує угоду про найменування (Convention over Configuration) і шукає в базі даних таблицю, назва якої є множиною від імені моделі у нижньому регістрі. Проте, явне оголошення захищеної властивості $table підвищує гнучкість системи та унеможливлює виникнення помилок іменування, жорстко прив'язуючи модель до створеної раніше таблиці faculties у СУБД MySQL. Стратегія захисту від масового заповнення. protected $guarded = []; Масове заповнення (Mass Assignment) — це вразливість, коли користувач передає через HTTP-запит неочікувані HTTP-параметри, які автоматично змінюють поля в базі

даних, до яких у нього не повинно бути доступу. Фреймворк Laravel вимагає обов'язкового визначення безпеки полів за допомогою однієї з двох властивостей: $fillable (білий список дозволених полів) або $guarded (чорний список заблокованих полів). Оголошення порожнього масиву protected $guarded = []; означає, що всі поля таблиці є доступними для масового створення та оновлення через методи Faculty::create() або Faculty::update(). Таке рішення є доцільним для сутності «Факультет», оскільки вона має обмежену кількість атрибутів (наприклад, лише name та тимчасові мітки), які заповнюються виключно адміністратором або через автоматизовані скрипти, що спрощує розробку та оптимізує код контролерів.

2.5.3 Організація взаємодії з базою даних через шар ORM-моделі. Для забезпечення гнучкості, безпеки та абстракції від конкретної системи керування базами даних (СКБД) у проєкті реалізовано об'єктно-реляційне відображення (ORM — Object-Relational Mapping) за допомогою вбудованого інструментарію Eloquent.

Об'єктне представлення сутності: Клас Faculty є спадкоємцем базового класу Illuminate\Database\Eloquent\Model. Він автоматично пов'язується з відповідною таблицею faculties у реляційній базі даних. Це дозволяє мапувати рядки таблиці на об'єкти мови PHP, перетворюючи кожне поле запису на властивість об'єкта моделі. Абстракція SQL-запитів: Виклик статичного методу «all()» інкапсулює в собі низькорівневу логіку формування SQL-запиту. На етапі виконання цей метод генерує та надсилає до СКБД запит виду:

SELECT * FROM `faculties`;

Використання такої абстракції усуває необхідність ручного написання SQL-коду, що знижує ймовірність синтаксичних помилок та полегшує потенційну міграцію проєкту на іншу СКБД (наприклад, з MySQL на PostgreSQL) без зміни кодової бази додатка. Формування колекції об'єктів: Результат виконання методу all() повертається не у вигляді масиву «сирих» даних, а трансформується в об'єкт класу

Illuminate\Database\Eloquent\Collection. Колекція містить масив екземплярів моделі Faculty. Це надає розробнику доступ до широкого спектра вбудованих методів фільтрації, сортування та агрегації даних безпосередньо у коді додатка без додаткових запитів до БД. Використання ORM Eloquent за замовчуванням застосовує механізм підготовлених запитів (PDO parameter binding) під час взаємодії з базою даних. Це мінімізує ризики уразливостей системи перед SQL-ін'єкціями (SQL injections) та підвищує загальний рівень захищеності інформаційної системи

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Абстракції предметної області

На етапі проєктування програмного забезпечення важливою складовою є виділення ключових абстракцій предметної області, які відображають реальні об’єкти, учасників і процеси, характерні для діяльності закладу вищої освіти. Абстракції – це узагальнені моделі, що містять у собі найважливіші властивості та обов’язки відповідних сутностей, необхідні для реалізації логіки системи. Зображено діаграму активності, яка демонструє загальну структуру абстракцій предметної області та процес формування статистики фізичних осіб. На рис. 7 зображено загальну структуру абстракцій предметної області, яка є основою для подальшого проєктування системи.

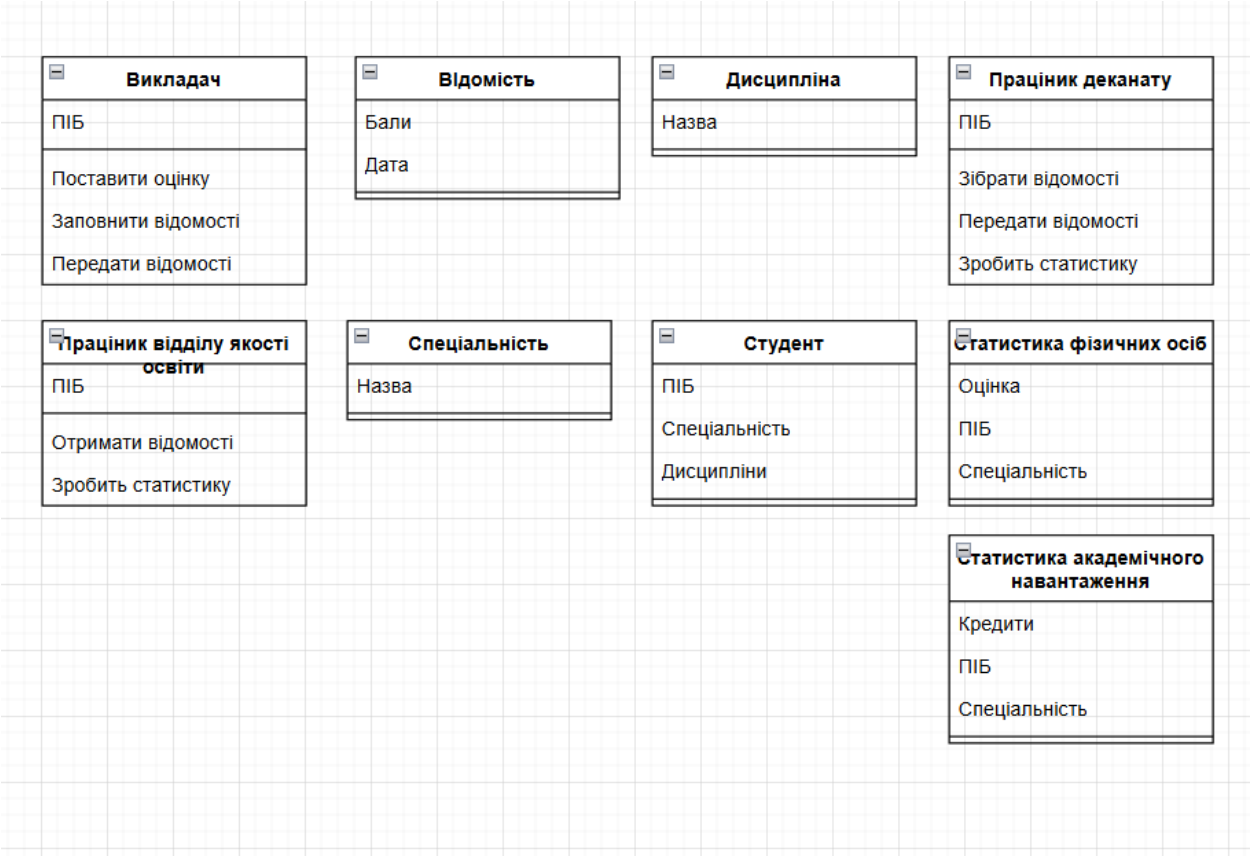


Рис. 7 Абстракції предметної області

Кожна з абстракцій охоплює типові ролі, елементи та процеси, що мають місце в роботі освітнього закладу. Зокрема, до абстракцій належать такі суб’єкти, як:

- Працівник відділу якості освіти – ініціює процес збору статистики, визначає параметри аналізу та отримує узагальнені результати для оцінки якості освітнього процесу.
- Працівник деканату – виступає посередником між викладачами та відділом якості, відповідає за збір, перевірку та консолідацію даних.
- Викладач – джерело первинної інформації, формує відомості про успішність студентів та академічні години.
- Студент – об’єкт аналізу, щодо якого ведеться персональна картка, що містить оцінки, дисципліни та інші показники навчальної діяльності.
- Відомість – документ, у якому фіксуються оцінки та академічні години.
- Статистика – узагальнені дані, що формуються на основі відомостей та використовуються для управлінських рішень.
- Академічне навантаження – показник, що відображає кількість навчальних годин викладача та студентів.

3.2 Моделювання структури та взаємодії об’єктів

Подальший етап проєктування передбачає моделювання структури та взаємодії об’єктів, що дає змогу чітко окреслити складові системи та визначити їхні зв’язки. На цьому рівні важливо не лише відобразити логіку роботи системи, а й систематизувати її компоненти, встановити, які сутності існують, як вони пов’язані між собою та яким чином взаємодіють у процесі реалізації функціоналу.

Таке моделювання дозволяє сформувати загальне уявлення про архітектуру системи та виступає базою для її програмної реалізації. Для опису використовується UML, що є універсальним стандартом у побудові об’єктно-орієнтованих моделей.

3.2.1 Діаграма класів (class diagram) — це статичне відображення структури системи, яке демонструє основні елементи моделі: класи, типи даних, їхні атрибути та взаємозв’язки. Вона є одним із ключових інструментів UML, адже саме на її основі часто здійснюється генерація програмних компонентів.

На відміну від діаграм, що описують динаміку процесів діаграма класів фокусується на структурі та є найбільш поширеним видом UML-діаграм. Її популярність пояснюється тим, що вона формує логічну моделі системи та забезпечує уявлення про об’єкти, властивості та взаємозв’язки, необхідні для реалізації програмного забезпечення. На рис. 8 діаграма класів з асоціаціями

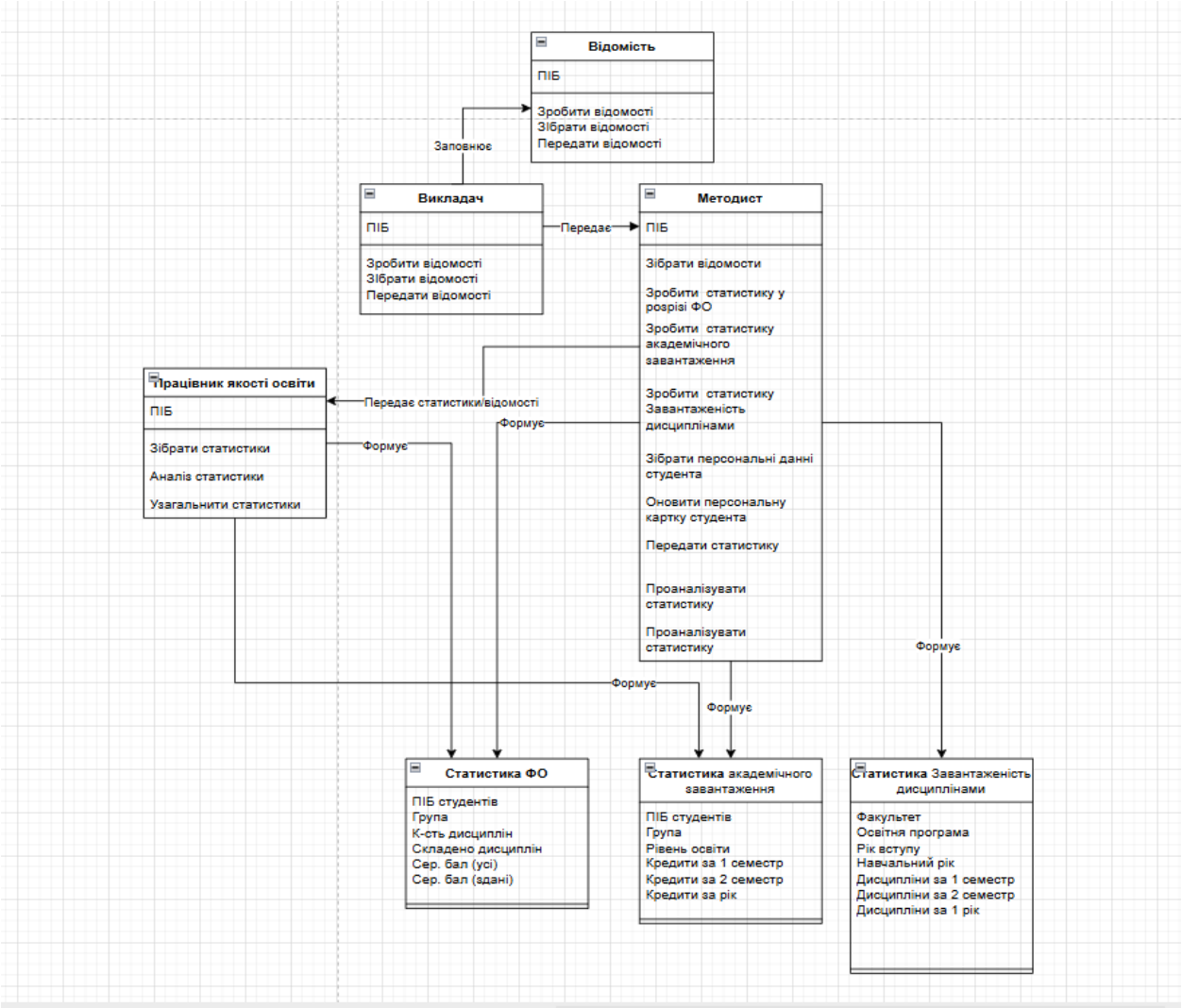


Рис. 8 Діаграма класів з асоціаціями

Діаграма класів, наведена у моделі, відображає статичну структуру підсистеми аналізу АС «Деканат» та демонструє взаємозв'язки між ключовими об'єктами освітнього процесу. Вона дозволяє визначити основні сутності, їхні атрибути та методи, а також показує асоціації, що забезпечують логіку взаємодії між ними.

Основні класи та їх атрибути:

- **Викладач**. Атрибути: ПІБ Методи: створити відомість, зібрати відомості, передати відомості.
- **Відомість.** Атрибути: ПІБ студента, оцінки, академічні години Методи: сформувати відомість, передати відомість.
- **Методист.** Атрибути: ПІБ Методи: зібрати відомості, створити статистику (ФО, академічне навантаження, дисципліни), оновити персональну карту студента, передати та проаналізувати статистику.
- **Працівник якості освіти.** Атрибути: ПІБ Методи: зібрати статистики, узагальнити статистики, провести аналіз.
- **Статистика фізичних осіб (ФО).** Атрибути: ПІБ студентів, група, кількість дисциплін, середній бал (усі та здані).
- **Статистика академічного навантаження.** Атрибути: ПІБ студентів, група, рівень освіти, кредити за семестри та рік.
- **Статистика завантаженості дисциплінами**. Атрибути: факультет, освітня програма, рік вступу, навчальний рік, дисципліни за семестри та рік.

Асоціації між класами:

- **Викладач.** Відомість – викладач формує та передає відомості, які містять дані про студентів.
- **Відомість**. Методист – методист отримує відомості від викладачів, консолідує їх та створює статистику.
- **Методист.** Статистика – методист генерує різні види статистики (ФО, академічне навантаження, дисципліни).

- **Методист**. Працівник якості освіти – передає узагальнені статистичні дані для подальшого аналізу.
- **Працівник** якості освіти Статистика – здійснює збір, узагальнення та аналіз статистики для оцінки якості освітнього процесу.
- **Студент**. Статистика – дані про студентів (оцінки, дисципліни, кредити) є основою для формування статистики.

Значення моделі:

Діаграма класів із асоціаціями забезпечує:

- чітке структурування об’єктів системи;
- визначення ролей та відповідальності кожного учасника;
- відображення логіки передачі даних від викладача до методиста та далі до відділу якості освіти;
- узгодженість між первинними даними (відомості) та результатами їх обробки (статистика).

Таким чином, ця діаграма класів є фундаментом для побудови програмної архітектури підсистеми аналізу, адже вона демонструє як статичну структуру, так і ключові взаємозв’язки між об’єктами, що забезпечують повний цикл обробки та аналізу освітніх даних.

3.2.2 Діаграма кооперацій. Діаграми кооперацій виступають важливим засобом моделювання взаємодії між об’єктами всередині системи. Вони дають змогу детально простежити, як елементи різних класів співпрацюють між собою для реалізації певних функцій. У таких діаграмах застосовуються різні типи зв’язків — асоціації, залежності, агрегації, композиції та узагальнення. Це забезпечує комплексне уявлення про логіку роботи системи та дозволяє відобразити як структуру, так і механізми взаємодії між її складовими. На рис.9 Діаграма кооперацій

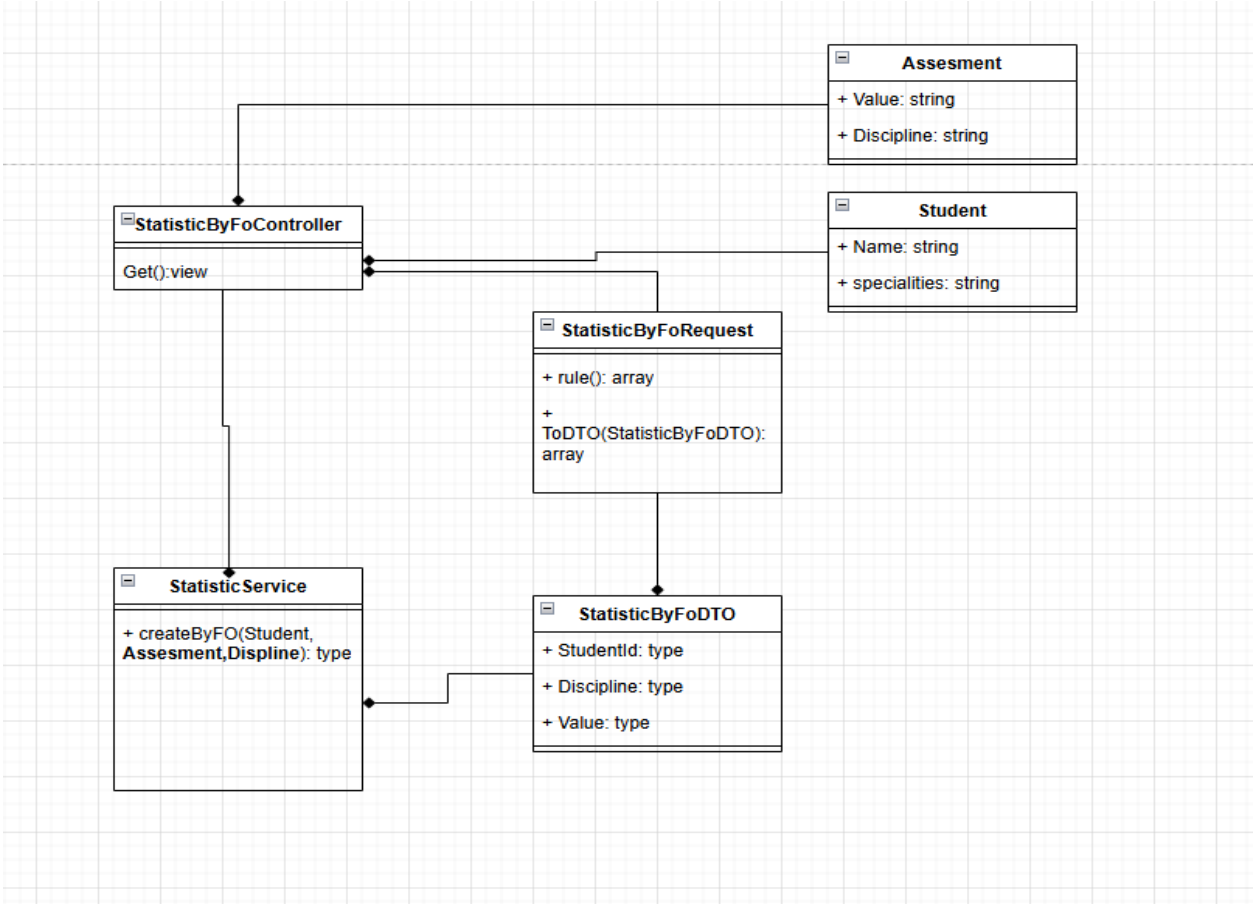


Рис.9 Діаграма кооперацій

Діаграма кооперацій, наведена на рис. 9, відображає процес створення статистики у розрізі фізичних осіб та демонструє взаємодію між ключовими об’єктами системи. Вона показує, як різні компоненти співпрацюють між собою для реалізації функціоналу збору та обробки даних.

У моделі беруть участь такі сутності:

- StatisticByFoController – відповідає за прийом запитів від користувача та ініціює процес формування статистики.
- StatisticService – основний сервіс, що реалізує бізнес-логіку створення статистики, використовуючи дані студентів, оцінок та дисциплін.
- StatisticByFoRequest – обробляє вхідні дані, виконує перевірку правил та трансформує їх у формат DTO.
- StatisticByFoDTO – виступає об’єктом передачі даних, що містить ідентифікатор студента, дисципліну та значення оцінки.

- Student – бізнес-сутність, яка зберігає інформацію про студента (ім’я, спеціальність).
- Assesment – сутність, що відображає оцінку та дисципліну, пов’язану зі студентом.

Логіка взаємодії:

1) Користувач надсилає запит через Controller.
2) Запит обробляється класом Request, де дані перевіряються та перетворюються у DTO.
3) Service отримує DTO та формує статистику, використовуючи дані студентів та їхні оцінки.
4) Результат повертається через Controller у вигляді сформованої статистики.

Значення діаграми:

- Вона демонструє чітке розмежування відповідальності між рівнями системи (Controller, Service, Request/DTO, бізнес-сутності).
- Відображає застосування патерну MVC, де контролер відповідає за взаємодію з користувачем, сервіс – за бізнес-логіку, а DTO – за передачу даних.
- Показує напрямки асоціацій та залежностей, що забезпечують узгодженість між об’єктами.

Таким чином, діаграма кооперацій ілюструє механізм формування статистики у системі та дозволяє зрозуміти, як саме відбувається взаємодія між її складовими. Вона є важливим елементом моделювання, оскільки поєднує структурний та поведінковий аспекти роботи підсистеми аналізу.

3.3 Особливості вебпрограмування та протокол HTTP

Сучасна інженерія програмного забезпечення для вебсередовища базується на специфічній архітектурній моделі, яка принципово відрізняється від проектування класичного десктопного ПЗ. Головною особливістю

вебпрограмування є розподіленість обчислень між двома ключовими вузлами: клієнтською (Client-side) та серверною (Server-side) частинами.

- Клієнтська частина (Frontend) відповідає за інтерфейс користувача та представлення даних. Вона інтерпретується безпосередньо веббраузером за допомогою технологій HTML5, CSS3 та JavaScript. Її завдання — забезпечити зручність введення відомостей викладачами та візуалізацію аналітичних звітів для відділу якості.
- Серверна частина (Backend) виконує роль ядра системи, де зосереджена вся бізнес-логіка, алгоритми агрегації статистики та взаємодія з базою даних MySQL. У даній роботі серверна логіка реалізується за допомогою мови програмування PHP та фреймворку Laravel

Архітектура та принципи роботи протоколу HTTP

Взаємодія між клієнтом та сервером у глобальній мережі регламентується протоколом HTTP - прикладним протоколом передачі даних, який працює поверх транспортного протоколу TCP/IP.

Основними архітектурними принципами HTTP, що безпосередньо впливають на розробку вебдодатків, є:

- Модель «Запит-Відповідь»: клієнт ініціює з’єднання, надсилаючи HTTP-запит, а сервер обробляє його та повертає HTTP-відповідь із кодом стану та результатом обробки.
- Відсутність збереження стану (Stateless): протокол HTTP за замовчуванням не зберігає інформацію про попередні запити користувача. Кожен запит інтерпретується сервером як незалежний. Для реалізації механізмів авторизації користувачів (розмежування ролей працівника деканату, викладача та відділу якості) у вебпрограмуванні застосовуються механізми сесій та кукі, які підтримуються на рівні фреймворку Laravel

3.4 Організаційна структура програмного забезпечення

Організаційна структура програмного забезпечення у межах теми дипломної роботи представлена за допомогою **діаграми пакетів**, яка демонструє логічний поділ системи аналізу та обліку студентських даних на окремі функціональні модулі. Такий підхід дозволяє чітко відобразити архітектуру системи, визначити її складові та показати взаємозв’язки між ними.

Діаграма пакетів (рис. 10) ілюструє, як система розділяється на окремі блоки

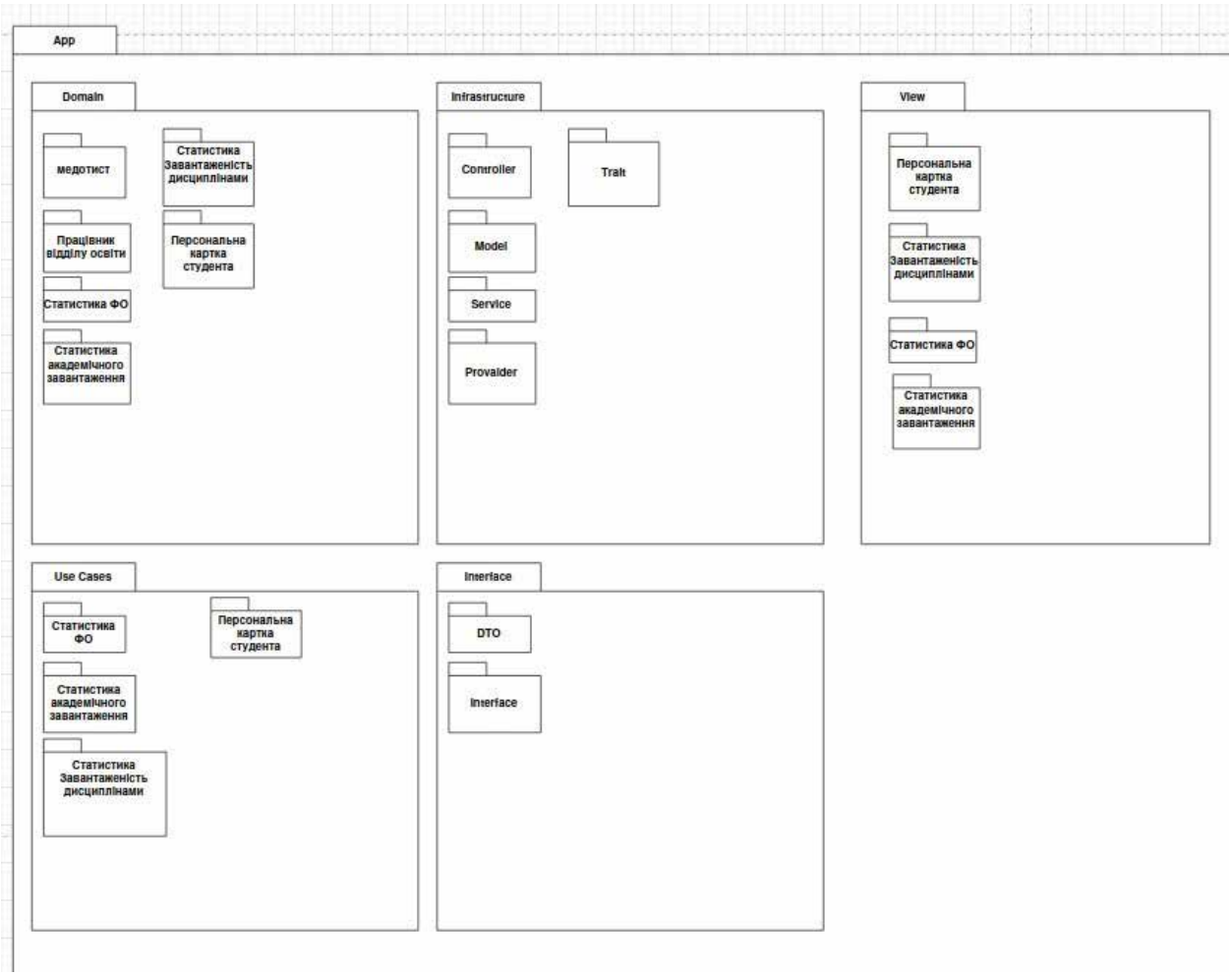


Рис.10 Діграма пакетів

На діаграмі пакетів(Рис.10) представлено логічну архітектуру програмного забезпечення, яка демонструє поділ системи на окремі функціональні модулі та взаємозв’язки між ними. Така структура дозволяє

чітко визначити ролі кожного компонента та забезпечує узгодженість між рівнями системи.

Основні пакети:

- App – головний рівень, що об’єднує всі модулі системи.
- Domain – містить ключові бізнес-сутності: користувач, персональна карта студента, статистика фізичних осіб, статистика академічного навантаження та завантаженості дисциплінами. Саме тут зосереджена предметна логіка.
- Infrastructure – технічний рівень, що включає контролери, моделі, провайдери та допоміжні елементи (traits). Він відповідає за реалізацію взаємодії між бізнес-логікою та зовнішнім середовищем.
- View – відображає інформацію для користувача. Тут реалізуються інтерфейси перегляду персональної карти студента та різних видів статистики.
- Use Cases – модуль, що описує сценарії використання системи: формування статистики фізичних осіб, академічного навантаження, завантаженості дисциплінами та робота з персональною картою студента.
- Interface – забезпечує взаємодію між компонентами через DTO та інтерфейси, що стандартизують обмін даними.

Значення моделі:

- Вона чітко розмежовує бізнес-логіку (Domain), технічну реалізацію (Infrastructure) та користувацький інтерфейс (View).
- Забезпечує прозорість архітектури та спрощує підтримку й розвиток системи.
- Дозволяє легко масштабувати програмне забезпечення завдяки модульному підходу.
- Відображає узгодженість між сценаріями використання та предметними сутностями.

Таким чином, діаграма пакетів демонструє організаційну структуру програмного забезпечення для аналізу студентських даних, показуючи, як різні рівні системи взаємодіють між собою та утворюють єдину архітектуру.

3.5 Компонентна структура системи

Діаграми компонентів у UML застосовуються для відображення архітектури програмного забезпечення на рівні його складових частин. Вони показують, як окремі модулі, бібліотеки та артефакти взаємодіють між собою, утворюючи єдину систему. Такий підхід дозволяє чітко структурувати програмний продукт, визначити його логічні блоки та встановити залежності між ними.

Компонентна діаграма є важливим інструментом проєктування, оскільки вона демонструє не лише внутрішню організацію системи, а й її інтеграцію з зовнішніми бібліотеками та сервісами. Це забезпечує прозорість архітектури, спрощує підтримку та розвиток програмного забезпечення, а також створює основу для масштабування. Діаграми компонентів зображена на рис.11

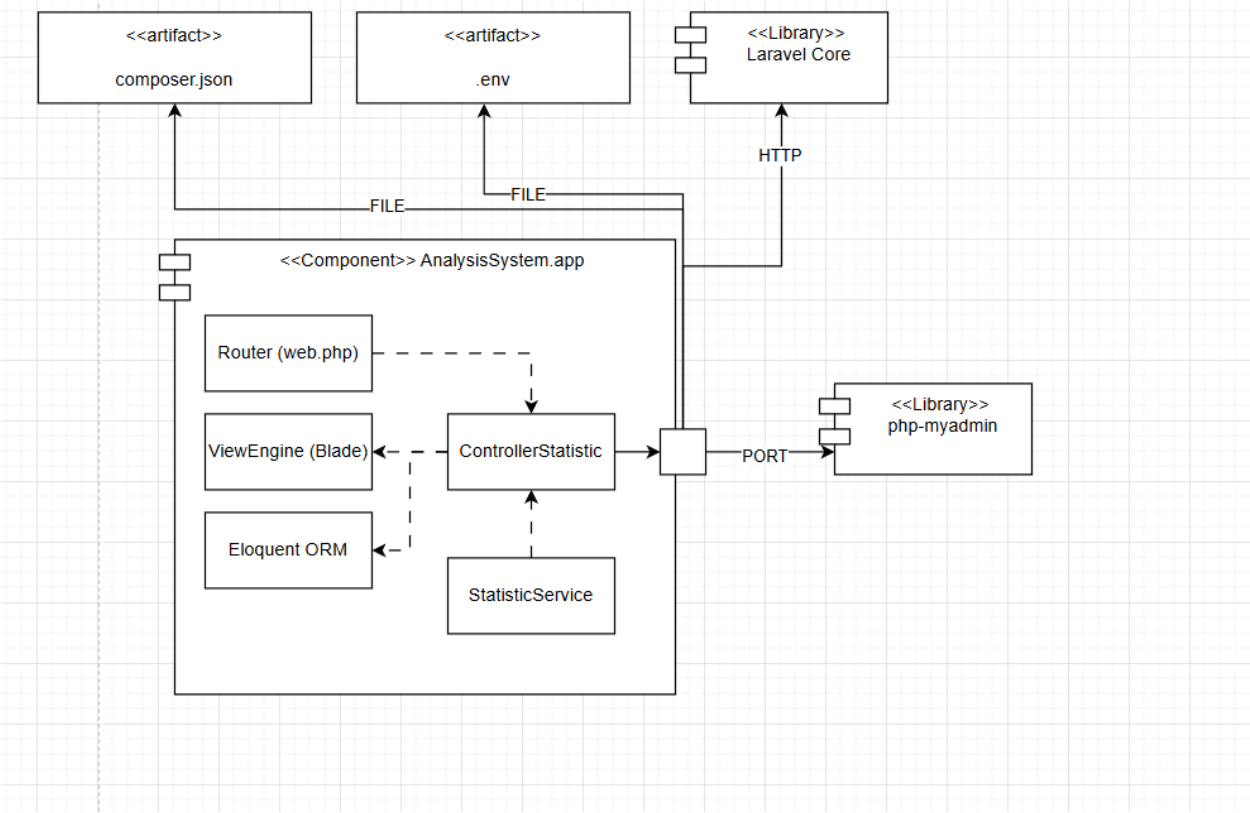


Рис.11 Діаграма компонентів

На діаграмі компонентів представлено архітектуру системи AnalysisSystem.app, реалізованої на основі фреймворку Laravel.

Основні елементи:

- Артефакти:
  - composer.json – файл конфігурації залежностей проєкту.
  - .env – файл середовищних змінних, що містить налаштування бази даних, портів та ключів доступу.
- Головний компонент – AnalysisSystem.app: Усередині нього розташовані ключові модулі:
  - Router (web.php) – відповідає за маршрутизацію запитів.
  - ControllerStatistic – контролер, що обробляє запити користувачів та викликає відповідні сервіси.
  - StatisticService – сервісний модуль, який реалізує бізнес-логіку формування статистики.
  - ViewEngine (Blade) – шаблонізатор, що забезпечує відображення даних у користувацькому інтерфейсі.
  - Eloquent ORM – інструмент для роботи з базою даних, що реалізує об’єктно-орієнтовану взаємодію з таблицями.
- Зовнішні бібліотеки:
  - Laravel Core – ядро фреймворку, що забезпечує роботу всієї системи (зв’язок через HTTP).
  - phpMyAdmin – інструмент для адміністрування бази даних, підключений через порт.

Логіка взаємодії:

1. Користувацький запит надходить до Router, який перенаправляє його до ControllerStatistic.
2. Контролер викликає StatisticService, що реалізує бізнес-логіку.
3. Сервіс звертається до Eloquent ORM для отримання даних із бази.

4. Результати передаються у ViewEngine (Blade) для відображення у вигляді веб-сторінки.
5. Конфігураційні файли (composer.json, .env) забезпечують налаштування залежностей та параметрів середовища.
6. Зовнішні бібліотеки інтегруються для підтримки роботи системи та адміністрування даних.

Таким чином, діаграма компонентів демонструє архітектуру системи на рівні її складових частин, показуючи як внутрішні модулі взаємодіють між собою та з зовнішніми бібліотеками. Вона є ключовим елементом проєктування, що забезпечує узгодженість між логікою, інтерфейсом та інфраструктурою програмного продукту.

3.6 Вибір інструментарію для створення прикладного програмного забезпечення

У процесі розробки прикладного програмного забезпечення було прийнято рішення використовувати сучасний інструментарій, який забезпечує ефективність, гнучкість та масштабованість системи. Важливим фактором стало те, що вже існував проєкт, реалізований на Laravel, що дозволило не витрачати час на вибір іншого фреймворку та забезпечило можливість повторного використання наявних напрацювань.

Використані інструменти та технології:

- Laravel Framework – основа проєкту, яка забезпечує реалізацію архітектури MVC, підтримку ORM (Eloquent), зручну маршрутизацію та інтеграцію з шаблонізатором Blade. Завдяки Laravel вдалося швидко організувати бізнес-логіку та забезпечити структурованість коду.
- Docker – застосовувався для контейнеризації системи. Це дозволило створити ізольоване середовище розробки, уникнути проблем із сумісністю та забезпечити однакові умови для запуску проєкту на різних платформах. Docker значно спростив процес розгортання та тестування.

- PhpStorm – інтегроване середовище розробки, яке стало основним інструментом для написання коду. PhpStorm забезпечив підтримку Laravel, автодоповнення, інтеграцію з системами контролю версій та інструменти для налагодження, що підвищило продуктивність роботи.
- WinSCP – використовувався для безпечного обміну файлами та управління даними на сервері. Це дало можливість швидко оновлювати проєкт, переносити конфігураційні файли та забезпечувати стабільність роботи системи.
- PowerShell – застосовувався для автоматизації рутинних завдань, налаштування середовища та управління контейнерами Docker. Використання PowerShell дозволило створювати скрипти для швидкого виконання команд, що значно спростило процес адміністрування та розгортання системи.
    - Переваги обраного інструментарію:
- Продовження існуючого проєкту на Laravel дозволило зберегти узгодженість архітектури та уникнути дублювання роботи.
- Docker забезпечив незалежність від локального середовища та спростив процес інтеграції з іншими сервісами.
- PhpStorm значно оптимізував процес розробки завдяки широкому набору інструментів для роботи з кодом.
- WinSCP гарантував безпечну передачу даних та зручне адміністрування файлів на сервері
- PowerShell дозволив автоматизувати процеси, що підвищило ефективність роботи та зменшило кількість ручних операцій

Обраний набір інструментів створив оптимальні умови для розробки прикладного програмного забезпечення. Він забезпечив баланс між продуктивністю, гнучкістю та надійністю, що дозволило ефективно реалізувати функціонал системи, підготувати її до подальшого розгортання й використання, а також спростити процес адміністрування та підтримки.

3.7 Програмна архітектура: концепція MVC у середовищі Laravel

MVC (Model-View-Controller) — це один із найпопулярніших архітектурних шаблонів проєктування програмного забезпечення. Його головна мета полягає в розділенні бізнес-логики програми, інтерфейсу користувача та даних на три окремі компоненти. Це забезпечує гнучкість, високу масштабованість коду та значно спрощує його подальший супровід.

Шаблон складається з трьох ключових елементів:

- Model (Модель): Відповідає за роботу з даними та безпосередню бізнес-логіку системи. Вона взаємодіє з базою даних, виконує операції збереження, вибірки, оновлення чи видалення інформації, а також перевіряє її цілісність.
- View (Представлення / Вигляд): Це зовнішній інтерфейс програми, тобто все те, що користувач бачить на екрані свого пристрою (HTML-сторінки, форми, адаптивний веб-інтерфейс, динамічні графіки та звіти). Представлення лише відображає дані, які передає йому контролер.
- Controller (Контролер): Виступає в ролі посередника або «мозку» системи, який пов'язує Модель та Представлення. Він перехоплює дії користувача (наприклад, клік по кнопці чи запит сторінки), звертається за потрібною інформацією до Моделі, обробляє її та передає результат у Представлення.

Laravel — це сучасний веб-фреймворк для мови програмування PHP з відкритим кодом. Він повністю базується на об'єктно-орієнтованому програмуванні (ООП) та архітектурній моделі MVC. Його головне призначення — максимально спростити, убезпечити та прискорити процес створення вебдодатків будь-якої складності

Ключові особливості та переваги Laravel:

1. Eloquent ORM: Вбудована технологія об'єктно-реляційного відображення. Вона дозволяє взаємодіяти з таблицями бази даних як зі звичайними PHP-класами (моделями). Замість написання складних SQL-запитів вручну розробник може використовувати інтуїтивно зрозумілі методи PHP.
2. Маршрутизація (Routing): Дає змогу гнучко керувати URL-адресами у додатку, чітко розподіляючи, який саме Контролер має відповідати за конкретний HTTP-запит користувача.
3. Шаблонізатор Blade: Інструмент для зручного створення Представлень (View). Він дозволяє чистенько розділяти HTML-розмітку та серверний код, підтримує конструкції циклів, умов і спадкування шаблонів.
4. Високий рівень безпеки: Laravel має надійні вбудовані механізми захисту від найпоширеніших кіберзагроз, таких як SQL-ін'єкції, CSRF-атаки (підробка міжсайтових запитів) та XSS-вразливості.
5. Міграції баз даних: Своєрідна «система контролю версій» для структури вашої БД. Вона дозволяє створювати й змінювати таблиці за допомогою PHP-коду, що полегшує розгортання проєкту на сервері чи роботу в команді

3.8 Алгоритмізація та програмування програмних модулів

Алгоритмізація є ключовим етапом у процесі створення програмного забезпечення, адже саме вона визначає логіку роботи системи та послідовність виконання її функцій. Побудова алгоритмів дозволяє формалізувати процеси, які реалізуються у програмних модулях, та забезпечує їхню узгодженість і коректність. Програмування, у свою чергу, є практичним втіленням алгоритмів у вигляді коду, що виконується комп’ютером. Таким чином, алгоритмізація та програмування утворюють нерозривний цикл: від опису логіки до її реалізації у програмному продукті.

У межах дипломного проєкту алгоритмізація застосовується для моделювання процесів збору, а програмування забезпечує їхню реалізацію у

вигляді окремих модулів системи. Це дозволяє досягти автоматизації обробки інформації. Блок-схема алгоритму представлена на рис.12.

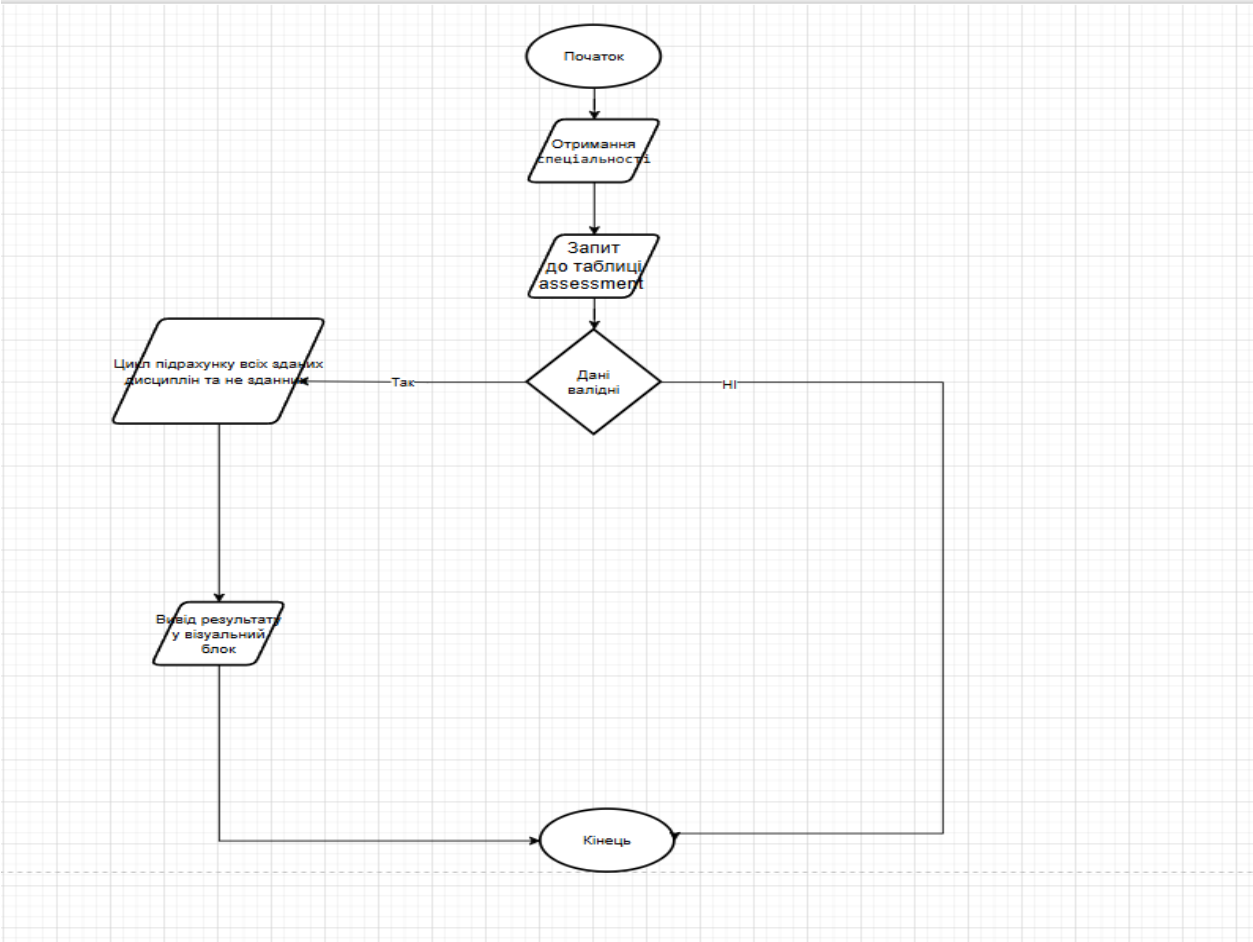


Рис.12 Блок-схема

На блок-схемі зображено алгоритм формування статистики у розрізі фізичних осіб. Він демонструє послідовність дій, які виконуються системою для отримання та перевірки даних, а також для їхнього подальшого аналізу.

3.8.1 Основні етапи алгоритму AJAX:

1. **Початок процесу** – ініціація запиту на формування статистики.
2. **Отримання спеціальності** – визначення навчальної спеціальності студента, для якого формується статистика.
3. **Запит до таблиці Assessment** – звернення до бази даних для отримання інформації про оцінки та дисципліни.
4. **Перевірка валідності даних** – Якщо дані коректні – запускається цикл підрахунку всіх зданих та незданих дисциплін.

5. **Формування результату** – підраховані дані відображаються у візуальному блоці (наприклад, у вигляді таблиці чи графіка).
6. **Завершення процесу** – користувач отримує готову статистику.

Значення діаграми:

- Вона демонструє чітку логіку роботи системи при формуванні статистики.
- Відображає можливість повернення на попередні етапи у разі виявлення помилок.
- Сприяє автоматизації аналізу успішності студентів та формуванню узагальнених результатів

Один із таких фрагментів – це використання технології **AJAX** для асинхронного отримання. Фрагмент коду на рис.13.

```
$.ajax({

    url: '{{ route('Statistic.ByFO.filter') }}',
    method: 'GET',
    data: {
        semester: semester,
        academicYear: academicYear,
        speciality_id: specialityId,
        EducationForm: educationForm,
        DebCount: debCount
    },
    success: function (response) {
        document.getElementById('students-table').innerHTML = response.html;
        hideLoader();
        let totalDetb = 0;
        let oneDetb = 0;
        let twoDetb = 0;
        let threeDetb = 0;

        response.students.forEach(student => {
            let studentDetb = student.AllDisciplines - student.DoneDisciplines;

            if (student.AllDisciplines > student.DoneDisciplines) {
                totalDetb++;
            }
            if (studentDetb == 1) {
                oneDetb++;
            }
            if (studentDetb == 2) {
                twoDetb++;
            }
            if (studentDetb >= 3) {
```

Рис. 13 Фрагмент коду AJAX

Логіка роботи коду:

1. Виклик маршруту – за допомогою $.ajax() здійснюється HTTP-запит до маршруту Statistic.ByFO.filter. У запит передаються параметри: семестр, навчальний рік, спеціальність, форма навчання та кількість академічних боргів.
2. Обробка відповіді – після успішного виконання запиту у блок students-table завантажується HTML-контент із даними студентів. Це дозволяє динамічно оновлювати інтерфейс.
3. Розрахунок статистики – для кожного студента визначається кількість академічних боргів як різниця між усіма дисциплінами (AllDisciplines) та зданими (DoneDisciplines).
    - Якщо студент має хоча б один борг – збільшується загальний лічильник боржників.
    - Окремо підраховуються студенти з одним, двома та трьома і більше боргами.
4. Оновлення графіків – перед побудовою нових діаграм попередні графіки знищуються (barChart.destroy(), donutChart.destroy()).
5. Візуалізація результатів – створюється стовпчикова діаграма (Bar Chart) за допомогою бібліотеки Chart.js, де відображаються:
    - загальна кількість боржників;
    - кількість студентів з одним боргом;
    - кількість студентів з двома боргами;
    - кількість студентів з трьома і більше боргами.

    Значення для системи:

- Код забезпечує динамічне оновлення статистики без перезавантаження сторінки, що робить систему більш інтерактивною.
- Реалізується автоматичний підрахунок академічних боргів, що дозволяє швидко отримати узагальнені дані.

- Використання графічної візуалізації робить результати зрозумілими та наочними для користувача.
- Такий підхід відповідає вимогам сучасних освітніх систем, де важлива швидкість доступу до інформації та її зручне представлення.

Таким чином, наведений фрагмент коду демонструє, як алгоритм формування статистики у розрізі фізичних осіб реалізується на практиці: від отримання даних через AJAX-запит до їхнього аналізу та графічного відображення у веб-інтерфейсі.

3.8.2 Логіка роботи контролера AJAXShowByFacultyController. Формування запиту до бази даних(Рис.14).Контролер звертається до кількох таблиць:

- personeducationcards – освітні карти студентів, що містять інформацію про статус, спеціальність та групу;
- StudentAssessment – результати складання дисциплін;
- persons – персональні дані студентів (ПІБ);
- CurriculumDiscipline – навчальні дисципліни з прив’язкою до семестру та навчального року;
- disciplines – назви дисциплін;
- studygroups – навчальні групи.

Виконується фільтрація за семестром, навчальним роком, статусом студента (активний), факультетом із сесії, а також спеціальністю (якщо вона передана у запиті). Результати сортуються за групою та ПІБ студента, що забезпечує впорядковане відображення даних.

```
$data = DB::table( table: 'personeducationcards')
    ->join( table: 'StudentAssessment', first: 'personeducationcards.id', operator: '=', second: 'StudentAssessment.personeducationcards_id')
    ->join( table: 'persons', first: 'personeducationcards.person_id', operator: '=', second: 'persons.id')
    ->join( table: 'CurriculumDiscipline', first: 'StudentAssessment.CurriculumDiscipline_id', operator: '=', second: 'CurriculumDiscipline.id')
    ->join( table: 'disciplines', first: 'CurriculumDiscipline.id_disciplines', operator: '=', second: 'disciplines.id')
    ->join( table: 'studygroups', first: 'personeducationcards.StudyGroup_id', operator: '=', second: 'studygroups.id')
    ->where( column: 'CurriculumDiscipline.AcademicYear', operator: '=', $request->academicYear)
    ->where( column: 'CurriculumDiscipline.Semester', operator: '=', $request->semester)
    ->where( column: 'personeducationcards.StudentStatus', operator: '=', value: '2')
    ->where( column: 'personeducationcards.faculty_id', operator: '=', session( key: 'faculty_id'))
    ->select(
        columns: 'persons.id as person_id',
        'persons.LastName',
        'persons.FirstName',
        'persons.MiddleName',
        'StudentAssessment.Value',
        'disciplines.DisciplinesName',
        'CurriculumDiscipline.AcademicYear',
        'CurriculumDiscipline.Semester',
        'CurriculumDiscipline.Old_id',
        'studygroups.StudyGroupName'
    )
    ->orderBy( column: 'studygroups.StudyGroupName')
    ->orderBy( column: 'persons.LastName')
    ->orderBy( column: 'persons.FirstName')
    ->orderBy( column: 'persons.MiddleName');
```

Рис.14 Формування запиту до бази даних

1. Групування даних по студенту
    - Для кожного студента створюється структура, яка включає:
        - ПІБ та назву навчальної групи;
        - кількість усіх дисциплін (AllDisciplines);
        - кількість зданих дисциплін (DoneDisciplines);
        - середній бал за всі дисципліни (AVGAssessments);
        - середній бал лише за здані дисципліни (AVGAssessmentsDoneDisciplines);
        - список оцінок із деталізацією по кожній дисципліні.
2. Облік повторних спроб складання дисциплін
    - Якщо студент складав дисципліну кілька разів, контролер враховує лише найвищу оцінку.
    - Це реалізується через масив maxValues, де зберігається максимальне значення для кожної дисципліни.
    - При зміні оцінки система коригує кількість зданих дисциплін:
        - **якщо нова оцінка ≥ 60, а попередня була < 60 – кількість зданих збільшується;**

- якщо нова оцінка < 60, а попередня була ≥ 60 – кількість зданих зменшується.

3. Обчислення середніх значень
    - Для кожного студента підраховується:
        - середній бал за всі дисципліни (сума оцінок / кількість дисциплін);
        - середній бал лише за здані дисципліни (сума оцінок ≥ 60 / кількість зданих).
    - Значення округлюються до двох знаків після коми.
4. Фільтрація за кількістю боргів
    - Контролер дозволяє відбирати студентів за параметром DebCount:
        - 1-3 – студенти з 1–3 боргами;
        - >3 – студенти з більш ніж трьома боргами;
        - haveDeb – студенти з будь-якою кількістю боргів;
        - All – усі студенти без додаткової фільтрації.
5. Підрахунок боржників
    - Після обробки даних визначається загальна кількість студентів, які мають хоча б один академічний борг (тобто кількість дисциплін > кількість зданих).

Значення для системи

- Контролер забезпечує **динамічне формування статистики** студентів у розрізі факультету.
- Реалізує **гнучку фільтрацію** даних за семестром, роком, спеціальністю та кількістю боргів.
- Враховує **повторні спроби складання дисциплін**, що робить аналіз більш точним і реалістичним.
- Формує **узагальнені показники успішності** (середні бали, кількість зданих дисциплін), які можуть бути використані для подальшої візуалізації у таблицях та графіках.

- Є ключовим елементом інтеграції між базою даних та інтерфейсом користувача, оскільки саме він готує дані для AJAX-запитів та відображення у веб-системі.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування є невід’ємною складовою процесу розробки програмного забезпечення, адже саме воно дозволяє переконатися у правильності реалізації функціоналу, виявити можливі помилки та оцінити відповідність системи вимогам користувачів. У ході перевірки було змодельовано різні сценарії використання, протестовано взаємодію між окремими компонентами та

перевірено роботу системи під різними ролями користувачів. Це дало змогу оцінити не лише окремі функції, а й загальну стабільність та зручність роботи.

Для забезпечення надійності та коректності функціонування було проведено комплексне тестування, яке включало кілька основних видів:

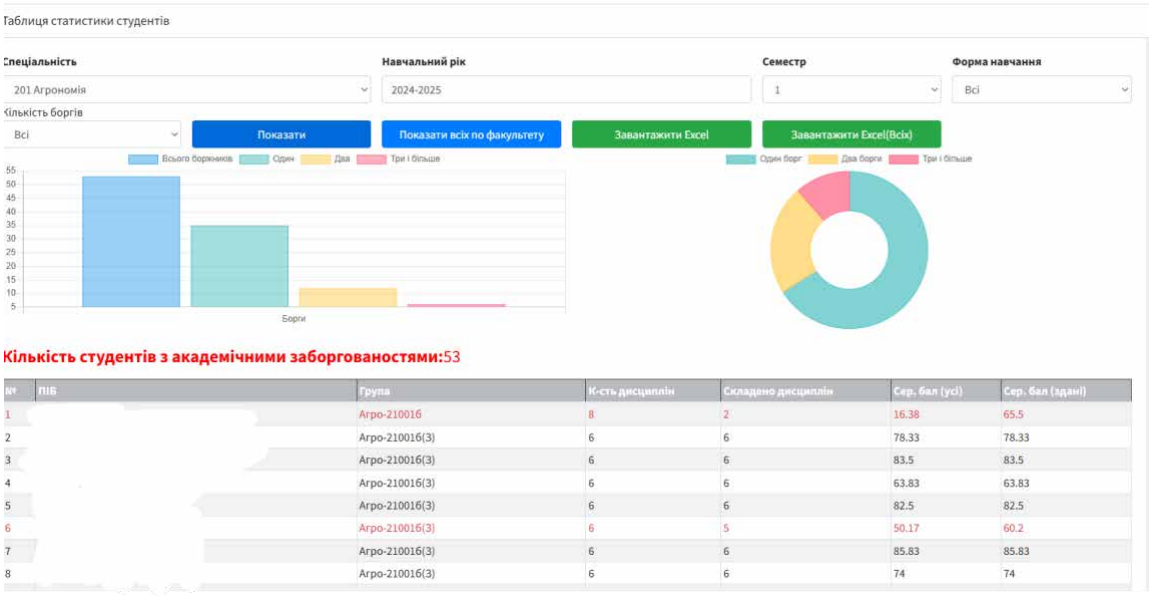
1. **Функціональне тестування** – перевірка відповідності реалізованих можливостей заявленим вимогам. Було протестовано формування статистики, роботу контролерів, обробку AJAX-запитів та відображення результатів у графічному інтерфейсі.
2. **Інтеграційне тестування** – аналіз взаємодії між модулями системи: контролерами, сервісами, базою даних та інтерфейсом користувача. Особливу увагу приділено коректності передачі даних між рівнями та узгодженості результатів.
3. **Тестування продуктивності** – оцінка швидкості виконання запитів до бази даних та відображення результатів у веб-інтерфейсі. Перевірено, що система здатна обробляти значні обсяги даних без критичних затримок.
4. **Тестування безпеки** – перевірка захищеності даних студентів, правильності роботи механізмів авторизації та доступу до інформації залежно від ролі користувача.
5. **Юзабіліті-тестування** – оцінка зручності використання системи. Було перевірено інтуїтивність інтерфейсу, зрозумілість відображення статистики та легкість навігації між основними функціями.
6. **Ручне тестування користувачами** – проведено серію перевірок із залученням реальних користувачів (адміністраторів, викладачів, студентів), які виконували типові сценарії роботи: перегляд статистики, фільтрація даних, формування звітів. Це дозволило виявити дрібні недоліки інтерфейсу та оцінити практичну зручність системи у реальних умовах.

4.1.1 Приклад тестування за методом ручного тестування. Приклад демонструє процес перевірки роботи системи без використання

автоматизованих інструментів, коли тестування здійснюється безпосередньо користувачем у реальному інтерфейсі. Такий підхід дозволяє оцінити практичну зручність використання, виявити помилки у взаємодії між компонентами та перевірити відповідність функціоналу очікуваним результатам.

Сценарій тестування(Статистика у розрізі ФО):

1. **Авторизація користувача** – тестувальник входить у систему під роллю «адміністратор» та перевіряє коректність механізму входу.
2. **Формування статистики** – у модулі «Статистика» «У розріз ФО» обирається семестр, навчальний рік та спеціальність.
3. **Перевірка відображення даних** – система повинна сформувати таблицю студентів із зазначенням кількості зданих та незданих дисциплін.
4. **Фільтрація за кількістю боргів** – користувач застосовує параметр DebCount (наприклад, «1–3 борги»), після чого перевіряє, що у списку залишилися лише студенти з відповідними показниками.
5. **Візуалізація результатів** – система будує графіки (Bar Chart, Donut Chart), які відображають кількість студентів із різним рівнем академічної заборгованості.
6. **Перевірка коректності даних** – тестувальник вручну звіряє результати у таблиці з даними у базі, щоб підтвердити правильність розрахунків.(Рис.15)



№	ПІБ	Група	К-сть дисциплін	Складено дисциплін	Сер. бал (усі)	Сер. бал (здані)
1		Агро-210016	8	2	16.38	65.5
2		Агро-210016(3)	6	6	78.33	78.33
3		Агро-210016(3)	6	6	83.5	83.5
4		Агро-210016(3)	6	6	63.83	63.83
5		Агро-210016(3)	6	6	82.5	82.5
6		Агро-210016(3)	6	5	50.17	60.2
7		Агро-210016(3)	6	6	85.83	85.83
8		Агро-210016(3)	6	6	74	74

Рис.15 Результат тестування статистики фізичних осіб

7. Тестування експорту в Excel –Система повинна сформувати файл у форматі .xlsx, який містить усі дані таблиці (Рис.16).

№	П	Група	Факультетть дисципдено дисцедній бал дній бал (к-сть боргіітня програпеціальніс	Форма на	Освітній
1		051-2021-	Економічн 1 1 82 82 0 Економіка Економіка	Заочна	Бакалавр
2		051-2021-	Економічн 1 1 80 80 0 Економіка Економіка	Заочна	Бакалавр
3		051-2021-	Економічн 1 1 80 80 0 Економіка Економіка	Заочна	Бакалавр
4		051-2022-	Економічн 8 8 71,13 71,13 0 Економіка Економіка	Заочна	Бакалавр
5		051-2022-	Економічн 8 8 75,75 75,75 0 Економіка Економіка	Заочна	Бакалавр
6		051-2022-	Економічн 8 8 88,38 88,38 0 Економіка Економіка	Заочна	Бакалавр
7		051-2022-	Економічн 8 8 79 79 0 Економіка Економіка	Заочна	Бакалавр
		51-2022-	Економічн 8 8 73,63 73,63 0 Економіка Економіка	Заочна	Бакалавр
9		51-2022-	Економічн 8 8 73 73 0 Економіка Економіка	Заочна	Бакалавр
10		51-2022-	Економічн 8 8 72,13 72,13 0 Економіка Економіка	Заочна	Бакалавр
11		1-2023-	Економічн 1 1 97 97 0 Економіка Економіка	Заочна	Бакалавр
12		51-2023-	Економічн 8 3 25,63 68,33 5 Економіка Економіка	Денна	Бакалавр
13		051-2023-	Економічн 1 1 98 98 0 Економіка Економіка	Заочна	Бакалавр
14		051-2023-	Економічн 1 1 92 92 0 Економіка Економіка	Заочна	Бакалавр
15		051-2023-	Економічн 1 1 92 92 0 Економіка Економіка	Заочна	Бакалавр
1		051-2023-	Економічн 1 1 99 99 0 Економіка Економіка	Заочна	Бакалавр
17		051-2023-	Економічн 7 7 63,86 63,86 0 Економіка Економіка	Заочна	Бакалавр
18		051-2023-	Економічн 1 1 76 76 0 Економіка Економіка	Заочна	Бакалавр
19		051-2023-	Економічн 1 1 90 90 0 Економіка Економіка	Заочна	Бакалавр
2		051-2023-	Економічн 1 1 85 85 0 Економіка Економіка	Заочна	Бакалавр
21		051-2023-	Економічн 1 1 90 90 0 Економіка Економіка	Заочна	Бакалавр
22		051-2023-	Економічн 1 1 90 90 0 Економіка Економіка	Заочна	Бакалавр
23		051-2023-	Економічн 1 1 92 92 0 Економіка Економіка	Заочна	Бакалавр
24		051-2023-	Економічн 1 1 87 87 0 Економіка Економіка	Заочна	Бакалавр
25		051-2024-	Економічн 8 7 63,38 72,43 1 Економіка Економіка	Заочна	Бакалавр
26		051-2024-	Економічн 1 1 94 94 0 Економіка Економіка	Заочна	Бакалавр

Рис. 16 Експорт даних в exel

Сценарій тестування(Статистика у анемічного навантаження):

Цей приклад демонструє перевірку роботи системи у «Статистика академічного навантаження». Мета тестування – оцінити коректність

розрахунку кредитів студентів, за 1 семестр, 2 семестри, та за навчальний рік а також перевірити зручність відображення результатів у інтерфейсі.

1. Формування статистики – у модулі обирається спеціальність (наприклад, «121 Інженерія програмного забезпечення») та навчальний рік (2023–2024). Система повинна згенерувати таблицю студентів із зазначенням кількості кредитів за семестри.
2. Перевірка відображення даних – формується таблиця з деталізацією: ПІБ студента – група – рівень освіти – кредити за 1-й та 2-й семестр – сумарні кредити – статус. Тестувальник перевіряє, що всі значення відповідають навчальному плану.
3. які відображають розподіл кредитів між студентами та групами.
4. Оцінка продуктивності – фіксується час, щоб визначити, чи відповідає він вимогам до швидкодії системи. Результат тестування рис.17

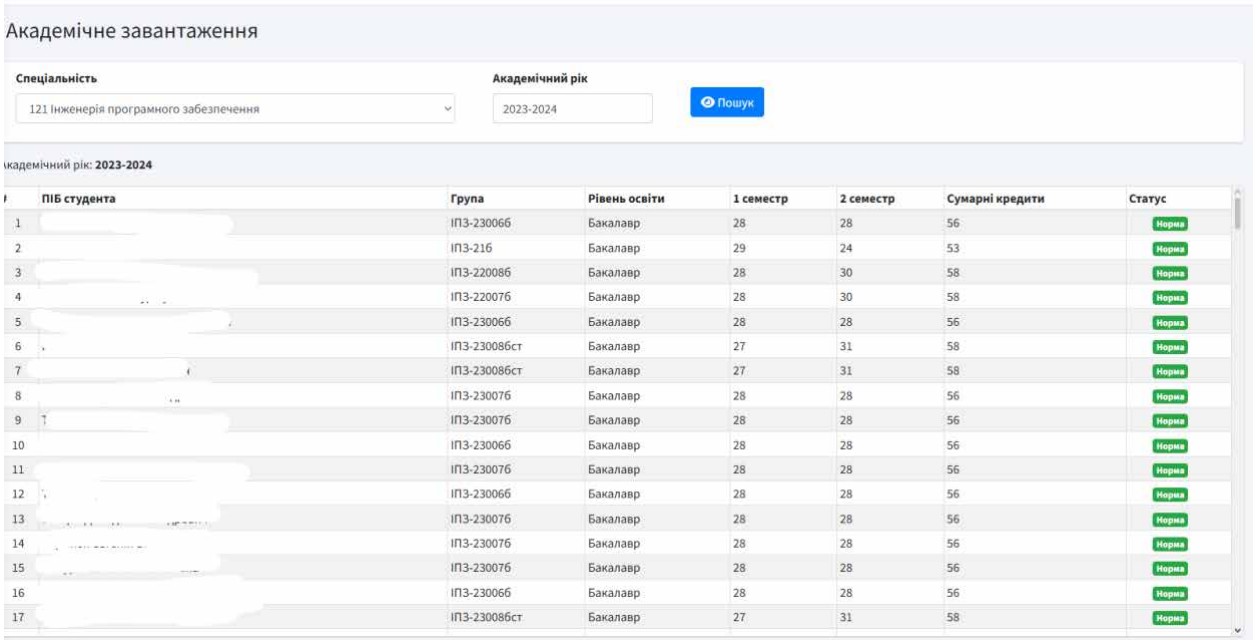


Рис.17 Результат тестування статистики акдемічного навантаження

Тестування персональної картки студента

Користувач (Адміністратор) заходить в розділ студенти->список студентів->В пошук по ПІБ вводить Ковалінський->навчальна картка

студента.(рис.18)

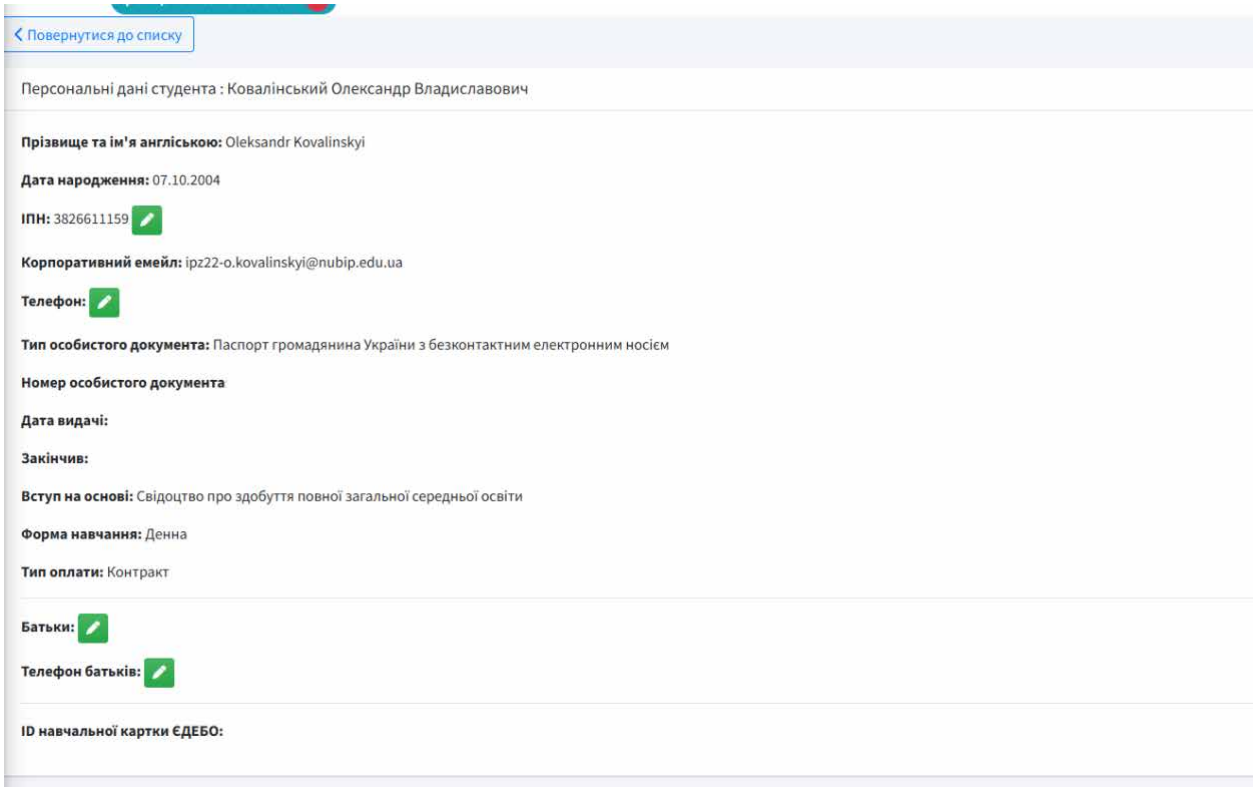


Рис.18 Результат тестування персональної картки студента

4.1.2 Функціональне тестування. Функціональне тестування у межах дипломної роботи було спрямоване на перевірку відповідності реалізованих можливостей системи заявленим вимогам. Особливу увагу приділено модулям, які забезпечують збір, обробку та синхронізацію даних студентів.

Використання моків у функціональному тестуванні. Для перевірки процесу синхронізації оцінок з системою eLearn застосовувався метод мок-тестування. Це дозволило імітувати роботу зовнішнього сервісу без реального підключення та зосередитися на логіці внутрішніх модулів. Призначення моків**:**

- Ізоляція компонентів – тестування проводиться без підключення до реальної бази даних чи зовнішніх API.
- **Прискорення перевірки** – моки повертають передбачувані результати, що дозволяє швидко перевіряти різні сценарії.
- **Виявлення помилок у логіці** – можна сфокусуватися на алгоритмах обробки даних, не відволікаючись на проблеми інтеграції.

- **Економія ресурсів** – немає потреби постійно синхронізуватися з реальними сервісами під час кожного тесту.

Логіка тестування:

1) Імітація відповіді eLearn – створюється мок-об’єкт, який повертає набір оцінок у форматі, аналогічному реальному API.
2) Передача даних у систему – тестовий модуль отримує ці оцінки та обробляє їх так, ніби вони прийшли від eLearn.
3) Перевірка алгоритму синхронізації – система повинна правильно зіставити дисципліни, оновити оцінки студентів та врахувати повторні спроби складання.
4) Аналіз результатів – перевіряється, що кількість зданих дисциплін, середній бал та кількість боргів відповідають очікуваним значенням.
5) Валідація помилкових сценаріїв – моки дозволяють змоделювати некоректні дані (наприклад, відсутність оцінки або неправильний формат), щоб перевірити, як система реагує на помилки.
    a. Різниця між Unit-тестами та Feature-тестами

- Unit-тести (модульні тести) перевіряють роботу окремих класів, методів чи функцій у повній ізоляції. Вони використовуються для перевірки дрібних одиниць логіки, наприклад: чи правильно працює метод підрахунку кількості зданих дисциплін.
- Feature-тести (функціональні тести) охоплюють роботу системи на рівні користувацьких сценаріїв. Вони перевіряють взаємодію між кількома компонентами одночасно, наприклад: чи коректно система отримує оцінки через AJAX-запит, синхронізує їх з eLearn та відображає результат у таблиці й графіках.

Приклад Feature-тесту з використання mokc рис. 19

```php
class SyncElern extends TestCase
{
    public function test_success(): void
    {
        $mockResponse = [
            'students' => [
                [
                    'id' => 1,
                    'ALLDisciplines' => 5,
                    'DoneDisciplines' => 4,
                ],
                [
                    'id' => 2,
                    'ALLDisciplines' => 6,
                    'DoneDisciplines' => 6,
                ],
            ]
        ];
        $this->mock( abstract: \App\Services\ElearnSyncService::class, function ($mock) use ($mockResponse) {
            $mock->shouldReceive('getGrades')
                ->once()
                ->andReturn($mockResponse);
        });
        $response = $this->get( uri: 'api/sync/elearn');
        $response->assertStatus( status: 200);
        $response->assertJsonFragment([
            'id' => 1,
            'DoneDisciplines' => 4,
        ]);
        $response->assertJsonFragment([
            'id' => 2,
```

SyncElern › test_success()

Рис19.Feature-тесту з використання mokc

4.2 Вимоги до апаратного та програмного забезпечення

Для коректного функціонування системи збору та аналізу статистики успішності студентів необхідно дотримуватися певних вимог до апаратного та програмного забезпечення. Це забезпечить стабільність роботи, швидке формування звітів та надійний захист даних. Правильна організація технічної бази дозволяє системі ефективно обробляти запити користувачів, підтримувати одночасну роботу великої кількості викладачів та адміністраторів, а також забезпечувати зручний інтерфейс для взаємодії з навчальною інформацією.

Відповідно до діаграми розгортання системи (рис.20)програмні компоненти розміщені на окремих апаратних вузлах, що відображає фізичну структуру реалізації.

Система реалізована за класичною схемою **клієнт–сервер.** Клієнтська частина – веб-браузер, який використовується у деканаті та на робочих місцях викладачів. Він забезпечує доступ до інтерфейсу системи через HTTPS-з’єднання.

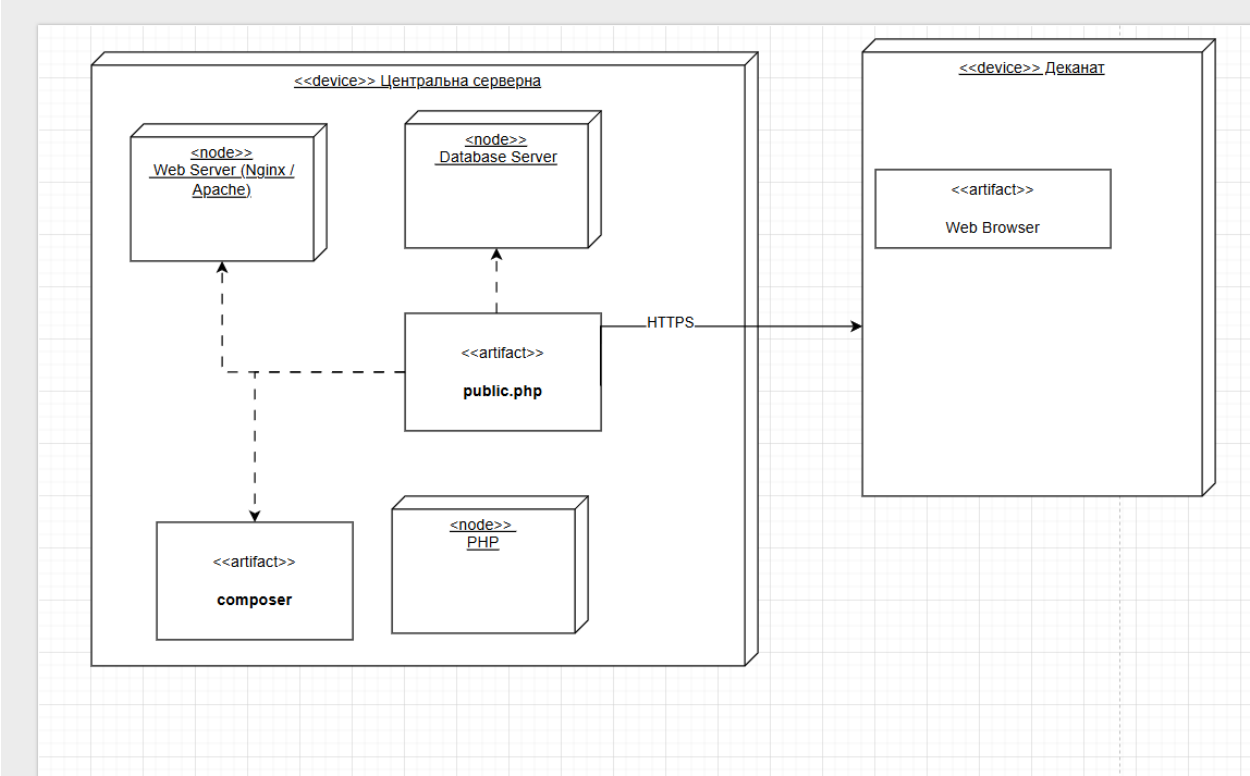


Рис. 20 Діаграма розгортання

Серверна частина – центральна серверна, що включає: Web Server (Nginx / Apache) – обробка HTTP-запитів та маршрутизація, PHP-модуль – виконання бізнес-логіки системи, Database Server (MySQL) – централізоване зберігання даних студентів, оцінок та навчальних планів, артефакти (public.php, composer), які забезпечують роботу застосунку та управління залежностями. Клієнтська частина взаємодіє із сервером через захищене HTTPS-з'єднання, що гарантує безпеку переданих даних. Клієнтська машина (робоче місце викладача/адміністратора): процесор не нижче Intel Core i3 або аналогічний, оперативна пам'ять – 4 ГБ і більше, вільне місце на диску – не менше 500 МБ, мережевий адаптер для підключення до локальної мережі або Інтернету. Серверна частина: процесор не нижче Intel Xeon або аналогічний, оперативна пам'ять – мінімум 16 ГБ, вільне місце на диску – мінімум 200 ГБ,надійне мережеве підключення з мінімальною затримкою.

Програмне забезпечення клієнтської машини вимагає операційну систему Windows 10 або новішу, сучасний веб-браузер (Google Chrome, Mozilla Firefox, Microsoft Edge),налаштований доступ до локальної мережі або Інтернету.

Програмне забезпечення серверної машини: операційну систему Linux (Ubuntu Server) або Windows Server, Web Server (Nginx / Apache),PHP (версія 8.0 і вище), СУБД MySQL / PostgreSQL (версія 12 і вище),Composer для управління залежностями, налаштовані служби резервного копіювання та безпеки.

Виконання цих вимог забезпечує: стабільну та ефективну роботу системи; можливість одночасної роботи багатьох користувачів, швидке формування статистики та звітів, надійний захист даних студентів, масштабованість та гнучкість архітектури для подальшого розвитку

4.3 Склад інсталяційного пакету системи аналізу успішності студентів

Інсталяційний пакет програмного забезпечення розроблений для швидкого та зручного розгортання як клієнтської, так і серверної частини системи. Він містить усі необхідні компоненти для повноцінної роботи та забезпечує коректну взаємодію між користувачами й центральним сервером. До складу інсталяційного пакету ваходять:

- **Інсталятор клієнтського застосунку** – забезпечує автоматичне встановлення веб-клієнта (браузерного інтерфейсу) або настільного застосунку для доступу до статистики. Інсталятор включає всі необхідні бібліотеки та середовище виконання PHP/Laravel.
- **Конфігураційний файл** – містить параметри підключення до серверної частини системи (адреса бази даних, налаштування доступу, параметри безпеки).

- **Серверні компоненти** – артефакти public.php та composer, що забезпечують роботу веб-застосунку, а також модулі PHP для виконання бізнес-логіки.
- **Документація** – включає інструкції з розгортання серверної частини (Web Server, Database Server) та налаштування клієнтської частини.

Особливості інсталяції

- Інсталятор автоматично встановлює всі необхідні бібліотеки та залежності, що забезпечує простоту та надійність розгортання клієнтської частини.
- Серверна частина (Web Server, PHP, Database Server) має бути встановлена та налаштована перед початком роботи клієнтських застосунків.
- Для адміністрування бази даних рекомендується використовувати **phpMyAdmin**.
- Конфігураційні файли дозволяють швидко змінювати параметри підключення без повторної інсталяції.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи на тему «Програмне забезпечення підсистеми аналізу автоматизованої системи Деканат» було здійснено повний цикл розробки інформаційної системи, що охоплює етапи аналізу предметної області, проєктування бази даних, створення програмних модулів та підготовки рекомендацій щодо впровадження і експлуатації. Проведений аналіз діяльності закладу вищої освіти дозволив визначити ключові напрями автоматизації, які включають збір та обробку даних про успішність студентів, формування статистики академічного навантаження, синхронізацію оцінок із зовнішніми системами (зокрема eLearn), а також створення зручних інструментів для викладачів та адміністрації. На основі виявлених потреб було сформульовано технічне завдання та розроблено концепцію програмного забезпечення. Було побудовано логічну модель даних у вигляді ER-діаграми, спроєктовано структуру бази даних у середовищі MySQL, а також реалізовано SQL-скрипти для автоматичної ініціалізації всіх необхідних об’єктів. Це дозволило створити стійку та масштабовану основу для зберігання навчальної інформації. Серверна частина системи реалізована на базі веб-технологій (Nginx/Apache, PHP, Composer), що забезпечує стабільну роботу та гнучкість у розгортанні. Клієнтська частина представлена веб-інтерфейсом, доступним через браузер, що робить систему універсальною та зручною для користувачів. Особливу увагу приділено тестуванню: проведено функціональні, інтеграційні та ручні тести, а також використано метод мок-тестування для перевірки синхронізації оцінок з eLearn. Це дозволило підтвердити коректність алгоритмів, стабільність роботи та готовність системи до практичного використання. Розроблене програмне забезпечення супроводжується інсталяційним пакетом, який містить усі необхідні компоненти для швидкого розгортання клієнтської та серверної частини. Завдяки цьому впровадження системи у навчальному закладі не потребує значних технічних зусиль і може бути здійснене

оперативно. У результаті виконаної роботи всі поставлені цілі та завдання були досягнуті. Розроблена система успішно автоматизує ключові процеси аналізу успішності студентів, поєднуючи зручність використання, стабільність роботи, ефективність обробки даних та можливість інтеграції з зовнішніми освітніми платформами. Реалізоване рішення є практичним, функціональним та готовим до застосування в реальних умовах навчального процесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sommerville I. **Software Engineering**. – 10th Edition. – Pearson Education, 2015. – 792 p.
2. Fowler M. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. – 3rd Edition. – Addison-Wesley, 2018. – 208 p.
3. Pressman R. S. **Software Engineering: A Practitioner's Approach**. – 8th Edition. – McGraw-Hill, 2014. – 976 p.
4. Gamma E., Helm R., Johnson R., Vlissides J. **Design Patterns: Elements of Reusable Object-Oriented Software**. – Addison-Wesley, 1994. – 395 p.
5. Martin R. C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. – Prentice Hall, 2017. – 432 p.
6. Martin R. C. **Clean Code: A Handbook of Agile Software Craftsmanship**. – Prentice Hall, 2008. – 464 p.
7. Evans E. **Domain-Driven Design: Tackling Complexity in the Heart of Software**. – Addison-Wesley, 2003. – 560 p.
8. Newman S. **Building Microservices: Designing Fine-Grained Systems**. – O'Reilly Media, 2021. – 432 p.
9. Richardson C. **Microservices Patterns: With Examples in Java**. – Manning Publications, 2018. – 520 p.
10. Bass L., Clements P., Kazman R. **Software Architecture in Practice**. – 4th Edition. – Addison-Wesley, 2021. – 624 p.
11. Hohpe G., Woolf B. **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**. – Addison-Wesley, 2003. – 736 p.
12. Elmasri R., Navathe S. B. **Fundamentals of Database Systems**. – 7th Edition. – Pearson, 2015. – 1272 p.
13. Silberschatz A., Korth H. F., Sudarshan S. **Database System Concepts**. – 7th Edition. – McGraw-Hill, 2020. – 1376 p.

14. Date C. J. **An Introduction to Database Systems**. – 8th Edition. – Addison-Wesley, 2003. – 1024 p.
15. Docker Documentation. – https://docs.docker.com
16. Laravel Documentation. – https://laravel.com/docs
17. PostgreSQL Documentation. – https://www.postgresql.org/docs
18. RabbitMQ Documentation. – https://www.rabbitmq.com/documentation.html (rabbitmq.com in Bing)
19. WinSCP Documentation. – https://winscp.net/eng/docs/start (winscp.net in Bing)
20. ISO/IEC 25010:2011 **Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE)**

ДОДАТКИ

ДОДАТОК А

```
@extends('layouts.main')

@section('content')
  <style>
    .card-body {
      max-height: 80vh;
      overflow-y: auto;
    }
  </style>
  <div class="card mb-4">
    <div class="card-header">
      <h3 class="card-title">Таблиця статистики студентів</h3>
    </div>
    <div class="card-body">
      <div class="row">

        {{-- Спеціальність --}}
        <div class="col-md-4">
          <label for="speciality_id" class="form-label">Спеціальність</label>
          <div id="speciality_select">
            <select class="form-control" name="speciality_id" id="speciality_id">
              <option value="">— Оберіть спеціальність —</option>
              @foreach ($specialities as $spec)
                <option value="{{ $spec->id }}">
```

```
                    {{ $spec->SpecialityNumber }} {{ $spec->SpecialityName
}}
                </option>
            @endforeach
        </select>
    </div>
</div>

{{-- Навчальний рік --}}
<div class="col-md-4">
    <label for="academicYear" class="form-label">Навчальний
рік</label>
    <input type="text" class="form-control" name="AcademicYear"
id="academicYear"
        value="2024-2025">
</div>

{{-- Семестр --}}
<div class="col-md-2">
    <label for="semester" class="form-label">Семестр</label>
    <select class="form-control" name="Semester" id="semester">
        <option value="1">1</option>
        <option value="2">2</option>
    </select>
</div>

{{-- Форма навчання --}}
<div class="col-md-2">
    <label for="EducationForm" class="form-label">Форма
навчання</label>
```

```
            <select class="form-control" name="EducationForm" id="EducationForm">
                <option value="All">Всі</option>
                <option value="Денна">Денна</option>
                <option value="Заочна">Заочна</option>
                <option value="Дистанційна">Дистанційна</option>

            </select>
        </div>


        {{--Кількість боржників--}}
        <div class="col-md-2">
            <lable for="CountDeb" class="form-lable">Кількість боргів</lable>
            <select class="form-control" name="CountDeb" id="CountDeb">
                <option value="All">Всі</option>
                <option value="1-3">1-3</option>
                <option value=">3">>3</option>
                <option value="haveDeb">Мають борги</option>
            </select>
        </div>

        {{-- Кнопка оновити --}}
        <div class="col-md-2 d-flex align-items-end">
            <button class="btn btn-primary w-100" onclick="loadSpecialities()">Показати</button>
        </div>

        {{-- Кнопка показати всіх по факультету --}}
```

```
            <div class="col-md-2 d-flex align-items-end">
              <button class="btn btn-primary w-100"
onclick="loadFaculty()">Показати всіх по факультету</button>
            </div>

            {{-- Кнопка скачати Excel --}}
            <div class="col-md-2 d-flex align-items-end mt-2">
              <button class="btn btn-success w-100"
onclick="downloadExcel()">Завантажити Excel</button>
            </div>
            {{-- Кнопка скачати Все --}}
            @if(auth()->check() && auth()->user()->role_id ===1)
              <div class="col-md-2 d-flex align-items-end mt-2">
                <button class="btn btn-success w-100"
onclick="downloadExcel()">Завантажити Excel(Всіх)</button>
              </div>
              <input type="hidden" id="isAdmin" value="1">
            @endif

          </div>
          <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
          <style>
            .chart {
              width: 100%;
              max-width: 800px;
              margin: auto;
            }
          </style>
          <div style="display: flex">
            <div class="chart" >
```

<canvas id="**barChart**" style="*min-height*:250**px**; *height*:250**px**; *max-height*:250**px**; *max-width*:100%;"></canvas>

<canvas id="**donutChart**" style="*min-height*: 250**px**; *height*: 250**px**; *max-height*: 250**px**; *max-width*: 100%; *width*: 739**px**;" width="**739**" height="**250**" class="**chartjs-render-monitor**"></canvas>

{{-- --}}
{{-- <canvas id="barChart" style="min-height:250px; height:250px; max-height:250px; max-width:100%;"></canvas>--}}
{{-- <canvas id="donutChart" style="min-height: 250px; height: 250px; max-height: 250px; max-width: 100%; width: 739px;" width="739" height="250" class="chartjs-render-monitor"></canvas>--}}

{{-- --}}
{{-- --}}

{{-- --}}

{{-- Динамічна таблиця --}}

@include('includes.loading')

```
</div>

<script>
    let barChart;
    let donutChart; // ще одна глобальна змінна

    function loadFaculty(){
        showLoader();
        const semester = document.getElementById('semester').value;
        const academicYear = document.getElementById('academicYear').value;
        const educationForm = document.getElementById('EducationForm').value;
        const debCount = document.getElementById('CountDeb').value;

        $.ajax({
            url: '{{ route('Statistic.ByFO.Filter.AllFaculty') }}',
            method: 'GET',
            data: {
                semester: semester,
                academicYear: academicYear,
                EducationForm: educationForm,
                DebCount: debCount
            },
            success: function (response) {
                document.getElementById('students-table').innerHTML = response.html;
                hideLoader();

                let totalDetb = 0;
                let oneDetb = 0;
```

```
let twoDetb = 0;
let threeDetb = 0;

response.students.forEach(student => {
  let studentDetb = student.AllDisciplines - student.DoneDisciplines;

  if (student.AllDisciplines > student.DoneDisciplines) {
    totalDetb++;
  }
  if (studentDetb == 1) {
    oneDetb++;
  }
  if (studentDetb == 2) {
    twoDetb++;
  }
  if (studentDetb >= 3) {
    threeDetb++;
  }
});

// 🧹 Чистимо попередні графіки
if (barChart) barChart.destroy();
if (donutChart) donutChart.destroy();

// === Bar Chart ===
const ctxBar = document.getElementById('barChart').getContext('2d');
barChart = new Chart(ctxBar, {
  type: 'bar',
  data: {
    labels: ['Борги'],
```

```
datasets: [
  {
    label: 'Всього боржників',
    data: [totalDetb],
    backgroundColor: 'rgba(54, 162, 235, 0.5)',
    borderColor: 'rgba(54, 162, 235, 1)',
    borderWidth: 1
  },
  {
    label: 'Один',
    data: [oneDetb],
    backgroundColor: 'rgba(75, 192, 192, 0.5)',
    borderColor: 'rgba(75, 192, 192, 1)',
    borderWidth: 1
  },
  {
    label: 'Два',
    data: [twoDetb],
    backgroundColor: 'rgba(255, 206, 86, 0.5)',
    borderColor: 'rgba(255, 206, 86, 1)',
    borderWidth: 1
  },
  {
    label: 'Три і більше',
    data: [threeDetb],
    backgroundColor: 'rgba(255, 99, 132, 0.5)',
    borderColor: 'rgba(255, 99, 132, 1)',
    borderWidth: 1
  }
]
```

```
    },
    options: {
      responsive: true,
      maintainAspectRatio: false,
      scales: {
        y: { beginAtZero: true }
      }
    }
  });

  // === Donut Chart ===
  const ctxDonut = document.getElementById('donutChart').getContext('2d');
  donutChart = new Chart(ctxDonut, {
    type: 'doughnut',
    data: {
      labels: ['Один борг', 'Два борги', 'Три і більше'],
      datasets: [{
        data: [oneDetb, twoDetb, threeDetb],
        backgroundColor: [
          'rgba(75, 192, 192, 0.7)',
          'rgba(255, 206, 86, 0.7)',
          'rgba(255, 99, 132, 0.7)'
        ],
        borderColor: [
          'rgba(75, 192, 192, 1)',
          'rgba(255, 206, 86, 1)',
          'rgba(255, 99, 132, 1)'
        ],
        borderWidth: 1
```

```
                }]
            },
            options: {
                responsive: true,
                maintainAspectRatio: false,
                plugins: {
                    legend: {
                        position: 'bottom'
                    }
                }
            }
        });
    },
    error: function () {
        hideLoader();

        document.getElementById('students-table').innerHTML =
            '<div class="alert alert-danger">Помилка при завантаженні даних</div>';
    }
});
}
function loadSpecialities() {
    showLoader();
    const semester = document.getElementById('semester').value;
    const academicYear = document.getElementById('academicYear').value;
    const specialityId = document.getElementById('speciality_id').value;
    const educationForm = document.getElementById('EducationForm').value;
    const debCount = document.getElementById('CountDeb').value;
```

```
$.ajax({

    url: '{{ route('Statistic.ByFO.filter') }}',
    method: 'GET',
    data: {
        semester: semester,
        academicYear: academicYear,
        speciality_id: specialityId,
        EducationForm: educationForm,
        DebCount: debCount
    },
    success: function (response) {
        document.getElementById('students-table').innerHTML = response.html;
        hideLoader();
        let totalDetb = 0;
        let oneDetb = 0;
        let twoDetb = 0;
        let threeDetb = 0;

        response.students.forEach(student => {
            let studentDetb = student.AllDisciplines - student.DoneDisciplines;

            if (student.AllDisciplines > student.DoneDisciplines) {
                totalDetb++;
            }
            if (studentDetb == 1) {
                oneDetb++;
            }
            if (studentDetb == 2) {
```

```
        twoDetb++;
    }
    if (studentDetb >= 3) {
        threeDetb++;
    }
});

// 🧹 Чистимо попередні графіки
if (barChart) barChart.destroy();
if (donutChart) donutChart.destroy();

// === Bar Chart ===
const ctxBar = document.getElementById('barChart').getContext('2d');
barChart = new Chart(ctxBar, {
    type: 'bar',
    data: {
        labels: ['Борги'],
        datasets: [
            {
                label: 'Всього боржників',
                data: [totalDetb],
                backgroundColor: 'rgba(54, 162, 235, 0.5)',
                borderColor: 'rgba(54, 162, 235, 1)',
                borderWidth: 1
            },
            {
                label: 'Один',
                data: [oneDetb],
                backgroundColor: 'rgba(75, 192, 192, 0.5)',
                borderColor: 'rgba(75, 192, 192, 1)',
```

```
            borderWidth: 1
        },
        {
            label: 'Два',
            data: [twoDetb],
            backgroundColor: 'rgba(255, 206, 86, 0.5)',
            borderColor: 'rgba(255, 206, 86, 1)',
            borderWidth: 1
        },
        {
            label: 'Три і більше',
            data: [threeDetb],
            backgroundColor: 'rgba(255, 99, 132, 0.5)',
            borderColor: 'rgba(255, 99, 132, 1)',
            borderWidth: 1
        }
    ]
},
options: {
    responsive: true,
    maintainAspectRatio: false,
    scales: {
        y: { beginAtZero: true }
    }
}
});

// === Donut Chart ===
const ctxDonut =
document.getElementById('donutChart').getContext('2d');
```

```
    donutChart = new Chart(ctxDonut, {
      type: 'doughnut',
      data: {
        labels: ['Один борг', 'Два борги', 'Три і більше'],
        datasets: [{
          data: [oneDetb, twoDetb, threeDetb],
          backgroundColor: [
            'rgba(75, 192, 192, 0.7)',
            'rgba(255, 206, 86, 0.7)',
            'rgba(255, 99, 132, 0.7)'
          ],
          borderColor: [
            'rgba(75, 192, 192, 1)',
            'rgba(255, 206, 86, 1)',
            'rgba(255, 99, 132, 1)'
          ],
          borderWidth: 1
        }]
      },
      options: {
        responsive: true,
        maintainAspectRatio: false,
        plugins: {
          legend: {
            position: 'bottom'
          }
        }
      }
    });
  },
```

```
            error: function () {
                hideLoader();
                document.getElementById('students-table').innerHTML =
                    '<div class="alert alert-danger">Помилка при завантаженні даних</div>';
            }
        });
    }


    function downloadExcel() {
        const semester = document.getElementById('semester').value;
        const academicYear = document.getElementById('academicYear').value;
        const specialityId = document.getElementById('speciality_id').value;
        const educationForm = document.getElementById('EducationForm').value;
        const debCount = document.getElementById('CountDeb').value;

        const isAdmin = document.getElementById('isAdmin')?.value === "1"; // ← отримаємо флаг

        if (!specialityId) {
            alert('Будь ласка, оберіть спеціальність!');
            return;
        }

        let url = '{{ route('Statistic.ByFO.downloadExcel') }}'
```

```
        + '?semester=' + encodeURIComponent(semester)
        + '&academicYear=' +
encodeURIComponent(academicYear)
        + '&speciality_id=' +
encodeURIComponent(specialityId)
        + '&EducationForm=' +
encodeURIComponent(educationForm)
        + '&debCount=' + encodeURIComponent(debCount);

      // Якщо адмін → додаємо прапорець у запит
      if (isAdmin) {
        url += '&isAdmin=1';
      }

      window.location.href = url;

    }
  </script>
@endsection
```

ДОДАТОК Б

```
<?php

namespace App\Http\Controllers\Statistics\ByFO;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class AJAXShowByFacultyController extends Controller
{
   public function __invoke(Request $request)
   {
      $students = [];

      $data = DB::table('personeducationcards')
         ->join('StudentAssessment', 'personeducationcards.id', '=', 'StudentAssessment.personeducationcards_id')
         ->join('persons', 'personeducationcards.person_id', '=', 'persons.id')
         ->join('CurriculumDiscipline',
```

```
'StudentAssessment.CurriculumDiscipline_id', '=', 'CurriculumDiscipline.id')
        ->join('disciplines', 'CurriculumDiscipline.id_disciplines', '=',
'disciplines.id')
        -
>join('studygroups','personeducationcards.StudyGroup_id','=','studygroups.id')
        ->where('CurriculumDiscipline.AcademicYear', '=', $request-
>academicYear)
        ->where('CurriculumDiscipline.Semester', '=', $request->semester)
        ->where('personeducationcards.StudentStatus', '=', '2')
        ->where('personeducationcards.faculty_id','=', session('faculty_id'))
        ->select(
          'persons.id as person_id',
          'persons.LastName',
          'persons.FirstName',
          'persons.MiddleName',
          'StudentAssessment.Value',
          'disciplines.DisciplinesName',
          'CurriculumDiscipline.AcademicYear',
          'CurriculumDiscipline.Semester',
          'CurriculumDiscipline.Old_id',
          'studygroups.StudyGroupName'
        )
        ->orderBy('studygroups.StudyGroupName')
        ->orderBy('persons.LastName')
        ->orderBy('persons.FirstName')
        ->orderBy('persons.MiddleName');


    if($request->speciality_id != null){
```

```
        $data->where('personeducationcards.speciality_id', '=', $request->speciality_id);
    }

    $data = $data->get();

    if (!$data->isEmpty()) {
        $grouped = [];

        foreach ($data as $item) {
            $key = $item->person_id;
            $discId = $item->Old_id;
            $value = $item->Value;

            if (!isset($grouped[$key])) {
                $grouped[$key] = [
                    'LastName' => $item->LastName,
                    'MiddleName'=>$item->MiddleName,
                    'FirstName' => $item->FirstName,
                    'StudyGroupName'=> $item->StudyGroupName,
                    'AVGAssessments'=> null,
                    'DoneDisciplines'=> 0,
                    'AllDisciplines'=> 0,
                    'assessments' => [],
                    'maxValues' => [],
                    'AVGAssessmentsDoneDisciplines'=>0

                ];
            }
```

```
if (!isset($grouped[$key]['maxValues'][$discId])) {
    $grouped[$key]['maxValues'][$discId] = $value;
    $grouped[$key]['AllDisciplines']++;

    $grouped[$key]['assessments'][] = [
        'DisciplinesID' => $discId,
        'DisciplinesName' => $item->DisciplinesName,
        'Value' => $value,
        'AcademicYear' => $item->AcademicYear,
        'Semester' => $item->Semester,
    ];

    if ($value >= 60) {
        $grouped[$key]['DoneDisciplines']++;
    }
} else {
    if ($value > $grouped[$key]['maxValues'][$discId]) {
        $prevValue = $grouped[$key]['maxValues'][$discId];
        $grouped[$key]['maxValues'][$discId] = $value;

        foreach ($grouped[$key]['assessments'] as &$assessment) {
            if ($assessment['DisciplinesID'] == $discId) {
                $assessment['Value'] = $value;
                break;
            }
        }
        unset($assessment);

        if ($prevValue < 60 && $value >= 60) {
            $grouped[$key]['DoneDisciplines']++;
```

```
            }
            if ($prevValue >= 60 && $value < 60) {
                $grouped[$key]['DoneDisciplines']--;
            }
        }
    }
}

foreach ($grouped as &$student) {
    $sumAll = 0;
    $sumDone = 0;
    $doneCount = 0;

    foreach ($student['maxValues'] as $discId => $value) {
        $sumAll += $value;

        if ($value >= 60) {
            $sumDone += $value;
            $doneCount++;
        }
    }

    $student['AVGAssessments'] = $student['AllDisciplines'] > 0
        ? round($sumAll / $student['AllDisciplines'], 2)
        : null;

    $student['AVGAssessmentsDoneDisciplines'] = $doneCount > 0
        ? round($sumDone / $doneCount, 2)
        : null;
```

```
            unset($student['maxValues']);
        }

        unset($student);

        $students = array_values($grouped);

// Фільтрація за кількістю боргів
        if ($request->DebCount !== 'All') {
            $filtered = [];

            foreach ($students as $student) {
                $debts = $student['AllDisciplines'] - $student['DoneDisciplines'];

                if ($request->DebCount === '1-3' && $debts >= 1 && $debts <= 3) {
                    $filtered[] = $student;
                } elseif ($request->DebCount === '>3' && $debts > 3) {
                    $filtered[] = $student;
                } elseif ($request->DebCount === 'haveDeb' && $debts > 0) {
                    $filtered[] = $student;
                }
            }

            $students = $filtered;
        }

    }
    $debtorCount = 0;

    foreach ($students as $s) {
```

```
        if ($s['DoneDisciplines'] < $s['AllDisciplines']) {
            $debtorCount++;
        }
    }

    $html = '';

    if (count($students) > 0) {
        $html .= '<h4 style="color: red"><strong>Кількість студентів з академічними заборгованостями:</strong>'. $debtorCount .'</h4>' ;
        $html .= '<div class="table-responsive mt-3">';
        $html .= '<table class="table table-bordered table-striped table-hover">';
        $html .= '<thead class="table-dark">';
        $html .= '<tr>';
        $html .= '<th>№</th>';
        $html .= '<th>ПІБ</th>';
        $html .= '<th>Група</th>';
        $html .= '<th>К-сть дисциплін</th>';
        $html .= '<th>Складено дисциплін</th>';
        $html .= '<th>Сер. бал (усі)</th>';
        $html .= '<th>Сер. бал (здані)</th>';
        $html .= '</tr>';
        $html .= '</thead><tbody>';

        foreach ($students as $index => $student) {
            $rowClass = ($student['AllDisciplines'] > $student['DoneDisciplines']) ? ' class="text-danger fw-bold"' : '';

            $html .= '<tr' . $rowClass . '>';
            $html .= '<td>' . ($index + 1) . '</td>';
```

```
            $html .= '<td>' . $student['LastName'] . ' ' . $student['FirstName'] . ' ' .
$student['MiddleName'] . '</td>';
            $html .= '<td>' . $student['StudyGroupName'] . '</td>';
            $html .= '<td>' . $student['AllDisciplines'] . '</td>';
            $html .= '<td>' . $student['DoneDisciplines'] . '</td>';
            $html .= '<td>' . ($student['AVGAssessments'] ?? '-') . '</td>';
            $html .= '<td>' . ($student['AVGAssessmentsDoneDisciplines'] ?? '-') .
'</td>';
            $html .= '</tr>';
        }

        $html .= '</tbody></table></div>';
    } else {
        $html .= '<div class="alert alert-info mt-4">Немає даних для
відображення</div>';
    }



    return response()->json([
//                                'success' => true,
//                            'semester' => $request->semester,
//                          'academicYear'=>$request->academicYear,
//                            'speciality_id'=>$request->speciality_id,

        'request' => $request->all(),
        'students' => $students,
        'html'=>$html
    ]);
```

```
    }
}
```

ДОДАТОК В

```
@extends('layouts.main')

@section('content')

  <a href="{{ route('student.index') }}" class="btn btn-outline-primary mb-3">
    <i class="fas fa-chevron-left me-2"></i> Повернутися до списку
  </a>
  {{-- Персональні дані студента --}}
  <div class="card mb-4">
    <div class="card-header">
      <h3 class="card-title">Персональні дані студента : {{ $transformed['LastName'] }} {{ $transformed['FirstName'] }} {{$transformed['MiddleName']}}</h3>
    </div>
    <div class="card-body">

{{-- <p><strong>Прізвище та ім'я:</strong> {{ $transformed['LastName'] }} {{ $transformed['FirstName'] }} {{$transformed['MiddleName']}}</p>--}}
      @if(!empty($transformed['FirstNameEn']) && !empty($transformed['LastNameEn']))

        <p><strong>Прізвище та ім'я англійською:</strong> {{ $transformed['FirstNameEn'] }} {{ $transformed['LastNameEn'] }} {{$transformed['MiddleNameEn']}}</p>

      @endif
      <p><strong>Дата народження:</strong> {{ $transformed['Birthday'] }}</p>
      <p>
        <strong>ІПН:</strong> {{ $transformed['IPN'] }}
        <button type="button" class="btn btn-sm bg-gradient-success" data-bs-toggle="modal" data-bs-target="#changeIpnModal">
          <i class="fas fa-pen"></i>
        </button>
```

```
        </p>


        <p><strong>Корпоративний емейл:</strong> {{ $transformed['Email'] }}</p>
        <p>
          <strong>Телефон:</strong> {{ $transformed['PhoneNumber'] }}
          <button type="button" class="btn btn-sm bg-gradient-success" data-bs-toggle="modal" data-bs-
target="#changePhoneModal">
            <i class="fas fa-pen"></i>
          </button>
        </p>
        @if(!empty($transformed['PrevilegesName']))
        <p><strong>Тип привелегії:</strong> {{$transformed['PrevilegesName']}}</p>
        @endif
        <p><strong>Тип особистого документа:</strong> {{$transformed['PersonDocumentType']}}</p>
        <p><strong>Номер особистого документа:</strong> {{$transformed['PersonDocumentNumber']}}</p>
        @if(!empty($transformed['PersonDocumentSeries']))
      <p><strong>Серійний номер</strong> {{$transformed['PersonDocumentSeries']}}
      @endif
      <p><strong>Дата видачі:</strong> {{$transformed['IssuedDate']}}</p>


      <p><strong>Закінчив:</strong> {{$transformed['IssuedBy']}}</p>
      <p><strong>Вступ на основі:</strong> {{$transformed['EducationBase']}}</p>
      <p><strong>Форма навчання:</strong> {{$transformed['EducationForm']}}</p>
      <p><strong>Тип оплати:</strong> {{$transformed['StudentStatus']}}</p>

        <hr>
        <p>
          <strong>Батьки:</strong> {{ $transformed['ParentsName'] }}
          <button type="button" class="btn btn-sm bg-gradient-success" data-bs-toggle="modal" data-bs-
target="#changeParentsNameModal">
            <i class="fas fa-pen"></i>
          </button>
        </p>
        <p>
          <strong>Телефон батьків:</strong> {{ $transformed['ParentsPhoneNumber'] }}
          <button type="button" class="btn btn-sm bg-gradient-success" data-bs-toggle="modal" data-bs-
target="#changeParentsPhoneModal">
            <i class="fas fa-pen"></i>
          </button>
        </p>

        <hr>
        <p><strong>ID навчальної картки ЄДЕБО:</strong> {{ $transformed['id'] }}</p>
        @if($transformed['debt'] > 0)
```

```
        <hr>
        <h4 style="color: red"><strong>Кількість академічних заборгованостей:</strong> {{ $transformed['debt'] }}</h4>
      @endif
    </div>
  </div>

  {{-- Таблиця успішності --}}
  @if(!empty(collect($transformed['assessments'])->flatten(1)))

    @foreach ($transformed['assessments'] as $year => $semesters)
      @php $isFirst = $loop->first; @endphp
      <div class="card card-primary card-outline mb-4 {{ $isFirst ? '' : 'collapsed-card' }}">
        <div class="card-header">
          <h3 class="card-title">Навчальний рік: {{ $year }}</h3>
          <div class="card-tools">
            <button type="button" class="btn btn-tool" data-card-widget="collapse" title="Згорнути">
              <i class="fas {{ $isFirst ? 'fa-minus' : 'fa-plus' }}"></i>
            </button>
          </div>
        </div>
        <div class="card-body" style="{{ $isFirst ? '' : 'display: none;' }}">
          @foreach ($semesters as $semester => $subjects)
            <h5 class="mt-3">Семестр: {{ $semester }}</h5>
            <table class="table table-bordered table-striped w-100" style="table-layout: fixed;">
              <thead>
              <tr>
                <th class="w-70">Дисципліна</th>
                <th>Оцінка</th>
              </tr>
              </thead>
              <tbody>
              @foreach ($subjects as $subject)
                @if($subject['Value']<60)
                  <tr>
                    <td class="font-weight-bold text-danger">
                      {{ $subject['DisciplinesName'] }}
                      @if (!empty($subject['typeValue']))

                        ({{ $subject['typeValue'] }})
                      @endif
                    </td>
                    <td class="text-danger">{{ $subject['Value'] }}</td>
                  </tr>
                @endif
                @if($subject['Value']>=60)
```

```
                    <tr>
                      <td class="text-truncate">
                        {{ $subject['DisciplinesName'] }}
                        @if (!empty($subject['typeValue']))

                          ({{ $subject['typeValue'] }})
                        @endif
                      </td>
                      <td>{{ $subject['Value'] }}</td>
                    </tr>
                    @endif
                  @endforeach
                  <tr>
                    <td class="font-weight-bold">Середній бал за семестр:</td>
                    <td class="font-weight-bold">
                      {{ $transformed['avgAssessmentsSemester'][$year][$semester] ?? '—' }}
                    </td>
                  </tr>
                  @if(($transformed['debtPerSemester'][$year][$semester] ?? 0) > 0)
                    <tr>
                      <td class="font-weight-bold text-danger">К-сть заборгованостей у семестрі:</td>
                      <td class="font-weight-bold text-danger">
                        {{ $transformed['debtPerSemester'][$year][$semester] }}
                      </td>
                    </tr>
                  @endif
                  </tbody>
                </table>
              @endforeach
            </div>
          </div>
        @endforeach
      @else
        <div class="alert alert-info">
          Даних про успішність не знайдено.
        </div>
      @endif
      <div class="modal fade" id="changeIpnModal" tabindex="-1" aria-labelledby="changeIpnModalLabel" aria-
hidden="true">
        <div class="modal-dialog modal-dialog-centered">
          <div class="modal-content">
            <form method="POST" action="{{ route('student.card.updateIPN', ['student' => $transformed['person_id']])
}}">
              @csrf
              @method('PUT')
              <div class="modal-header">
```

```
            <h5 class="modal-title" id="changeIpnModalLabel">Змінити ІПН</h5>
          </div>
          <div class="modal-body">
            <div class="mb-3">
              <label for="newIpn" class="form-label">Новий ІПН</label>
              <input type="text" class="form-control" id="newIpn" name="IPN"
                value="{{ $transformed['IPN'] }}" required pattern="[0-9]{10}">
              <small class="text-muted">10 цифр (наприклад: 1234567890)</small>
            </div>
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Скасувати</button>
            <button type="submit" class="btn btn-success">Зберегти</button>
          </div>
        </form>
      </div>
    </div>
  </div>


  <div class="modal fade" id="changePhoneModal" tabindex="-1" aria-labelledby="changePhoneModalLabel" aria-
hidden="true">
    <div class="modal-dialog modal-dialog-centered">
      <div class="modal-content">
        <form id="changePhoneForm" name="tel" method="POST" action="{{ route('student.card.updatePhone',
['id' => $transformed['person_id']]) }}">
          @csrf
          @method('PUT')
          <div class="modal-header">
            <h5 class="modal-title" id="changePhoneModalLabel">Змінити номер телефону</h5>
          </div>
          <div class="modal-body">
            <div class="form-group">
              <label>Телефон</label>
              <div class="input-group">
                <div class="input-group-prepend">
                  <span class="input-group-text"><i class="fas fa-phone"></i></span>
                </div>
                <input type="text" class="form-control" data-inputmask="&amp;quot;mask&amp;quot;: &amp;quot;(999) 999 99
99&amp;quot;" data-mask="" inputmode="text" id="newPhoneNumber" name="PhoneNumber" value="{{
$transformed['PhoneNumber'] }}" required>
              </div>
              <!-- /.input group -->
            </div>
{{-- <div class="mb-3">--}}
{{-- <label for="newPhoneNumber" class="form-label">Новий номер телефону</label>--}}
```

```
{{--                <input data-inputmask="&amp;quot;mask&amp;quot;: &amp;quot;(999) 999 99 99&amp;quot;" data-mask="" type="text"
class="form-control"                                id="newPhoneNumber"                                name="PhoneNumber"--}}
{{--                                value="{{ $transformed['PhoneNumber'] }}" required pattern="\+?[0-9\s\-]{10,15}"">--}}
{{--                                                <small class="text-muted">Наприклад: +380XXXXXXXXX</small>--}}
{{--                                                                                                                </div>--}}
            </div>
            <div                                                                                                class="modal-footer">
              <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Скасувати</button>
              <button                type="submit"                class="btn                btn-success">Зберегти</button>
            </div>
          </form>
        </div>
      </div>
    </div>
{{--                <div class="modal fade" id="changeParentsPhoneModal" tabindex="-1" aria-
labelledby="changeParentsPhoneModalLabel"                                                                aria-hidden="true">--}}
{{--                                                <div class="modal-dialog modal-dialog-centered">--}}
{{--                                                                                <div class="modal-content">--}}
{{--                <form method="POST" action="{{ route('student.card.EditParentsNumberController', ['student' =>
$transformed['person_id']])                                                                                }}">--}}
{{--                                                                                                                @csrf--}}
{{--                                                                                                                @method('PUT')--}}
{{--                                                                                <div class="modal-header">--}}
{{--            <h5 class="modal-title" id="changeParentsPhoneModalLabel">Змінити номер телефону батьків</h5>-
-}}
{{--                                                                                                                </div>--}}
{{--                                                                                <div class="modal-body">--}}
{{--                                                                                    <div class="mb-3">--}}
{{--                                <label for="ParentsPhoneNumber" class="form-label">Телефон батьків</label>--}}
{{--                                                                                    <div class="input-group">--}}
{{--                                                                                <div class="input-group-prepend">--}}
{{--                                                <span class="input-group-text"><i class="fas fa-phone"></i></span>--}}
{{--                                                                                                                </div>--}}
{{--                                                                                    <input type="text" class="form-control"--}}
{{--                                                                data-inputmask="'mask': '(999) 999 99 99'" data-mask--}}
{{--                                                                id="ParentsPhoneNumber" name="ParentsPhoneNumber"--}}
{{--                                                            value="{{ $transformed['ParentsPhoneNumber'] }}" required>--}}
{{--                                                                                                                </div>--}}
{{--                                                <small class="text-muted">Наприклад: (050) 123 45 67</small>--}}
{{--                                                                                                                </div>--}}
{{--                                                                                                                </div>--}}
{{--                                                                                <div class="modal-footer">--}}
{{--            <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Скасувати</button>--}}
{{--                                                <button type="submit" class="btn btn-success">Зберегти</button>--}}
{{--                                                                                                                </div>--}}
{{--                                                                                                                </form>--}}
```

```
{{--                                                                                                    </div>--}}
{{--                                                                                                    </div>--}}
{{--                                                                                                    </div>--}}
{{--                    <div class="modal fade" id="changeParentsNameModal" tabindex="-1" aria-
labelledby="changeParentsNameModalLabel"                                                    aria-hidden="true">--}}
{{--                                                    <div class="modal-dialog modal-dialog-centered">--}}
{{--                                                                    <div class="modal-content">--}}
{{--                            <form method="POST" action="{{ route('student.card.updateParents', ['student' =>
$transformed['person_id']])                                                                                    }}">--}}
{{--                                                                                                    @csrf--}}
{{--                                                                                                    @method('PUT')--}}
{{--                                                                    <div class="modal-header">--}}
{{--                    <h5 class="modal-title" id="changeParentsNameModalLabel">Змінити ПІБ батьків</h5>--}}
{{--                                                                                                    </div>--}}
{{--                                                                    <div class="modal-body">--}}
{{--                                                                            <div class="mb-3">--}}
{{--                                            <label for="ParentsName" class="form-label">ПІБ батьків</label>--}}
{{--                    <input type="text" class="form-control" id="ParentsFullName" name="ParentsFullName"--}}
{{--                                                    value="{{ $transformed['ParentsName'] }}" required>--}}
{{--                                                                                                    </div>--}}
{{--                                                                                                    </div>--}}
{{--                                                                    <div class="modal-footer">--}}
{{--                <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Скасувати</button>--}}
{{--                                            <button type="submit" class="btn btn-success">Зберегти</button>--}}
{{--                                                                                                    </div>--}}
{{--                                                                                                    </form>--}}
{{--                                                                                                    </div>--}}
{{--                                                                                                    </div>--}}
{{--                                                                                                    </div>--}}


@endsection
```

ДОДАТОК Д

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Ковалінський Олександр, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення підсистеми аналізу автоматизованої системи "Деканат"» є результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

3. Щодо використання інструментів ШІ зазначаю, що:

інструменти ШІ ChatGPT, були використані для: перекладу іншомовних джерел,стилістичного редагування та перевірки граматики. Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«21»травня 2026 р. _____ **Олександр КОВАЛІНСЬКИЙ** (підпис)