МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: «**Проєктування та розробка інформаційної системи замовлення довідок з функцією моніторингу та аналітики»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Дзем'юк Владислав Богданович

Керівник: доктор філософії з прикладної математики,
старший викладач кафедри інформаційних технологій та аналітики даних
Красюк Богдан Віталійович

Рецензент: кандидат технічних наук, доцент,
доцент кафедри прикладної математики
Донецького національного університету
імені Василя Стуса
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від « 20 » травня 2026 р.

Острог, 2026

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: **Проєктування та розробка інформаційної системи замовлення довідок з функцією моніторингу та аналітики**

Автор: Дзем'юк Владислав Богданович

Науковий керівник: доктор філософії з прикладної математики, викладач, фахівець-практик, Красюк Богдан Віталійович

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: 67 *с., 3 рис., 5 табл., 1 додаток, 39 джерел.*

Ключові слова: *ІНФОРМАЦІЙНА СИСТЕМА, ВЕБ-ЗАСТОСУНОК, ОНЛАЙН-ЗАЯВКИ, АКАДЕМІЧНІ ДОВІДКИ, ДОКУМЕНТООБІГ, ЖИТТЄВИЙ ЦИКЛ ЗАЯВКИ, СТАТУС-ТРЕКІНГ, RBAC, REST API, MONGODB*

Короткий зміст праці: *У кваліфікаційній роботі розглянуто задачу цифровізації обслуговування академічних довідок у закладі вищої освіти та запропоновано веб-систему для подання й опрацювання онлайн-заявок. Практична мета полягає у зменшенні ручних операцій, уникненні розпорошення даних між каналами комунікації й забезпеченні прогнозованого проходження звернення від подання до завершення. Виконано аналіз предметної області, узагальнено типові організаційні ризики (дублювання дій, втрата контексту, слабкий контроль статусів, інформаційна асиметрія між заявником і деканатом), проведено огляд існуючих підходів і сформульовано інженерні вимоги до централізованого процесу. На концептуальному рівні обґрунтовано доменну модель заявки, рольову модель доступу (RBAC), життєвий цикл статусів із журналом подій (timeline) та підтримку двох режимів подання: за типом довідки й «за призначенням». Архітектурно рішення реалізовано як клієнт-серверну взаємодію через REST API: бекенд на Node.js та Express із JWT-автентифікацією, MongoDB (Mongoose) для збереження агрегованих заявок і історії змін; фронтенд на Next.js із серверними маршрутами-проксі для узгодженого транспорту авторизованих запитів. У програмному варіанті впроваджено ключові сценарії: подання заявки, зміну статусів працівником деканату з обов'язковим обґрунтуванням відмови, сповіщення користувачів та аналітичні зведення для факультету. Зроблено висновок, що запропонована архітектура підвищує керованість довідкового процесу й створює підстави для подальшого впровадження в реальному освітньому середовищі.*

_____

ANNOTATION
of the qualification work
for obtaining the educational degree of Bachelor

Topic: **Design and development of an information system for ordering certificates with monitoring and analytics functions**

Author: Vladyslav Bogdanovych Dzemiuk

Academic Advisor: Doctor of Philosophy in Applied Mathematics, Lecturer, Practitioner Specialist, Bohdan Vitaliyovych Krasiuk

Defended on «……»…… 20__.

Explanatory note to the qualification work: 67 pages, 3 figures, 5 tables, 1 appendice, 39 sources.

Keywords: *INFORMATION SYSTEM, WEB APPLICATION, ONLINE APPLICATIONS, ACADEMIC CERTIFICATES, DOCUMENT FLOW, REQUEST LIFECYCLE, STATUS TRACKING, RBAC, REST API, MONGODB*

Brief summary of the work: *The qualification work addresses digital support for academic certificate services in higher education and presents a web system for submitting and processing online applications. The practical goal is to reduce manual workload, avoid scattering information across informal channels, and ensure a predictable path from submission to completion. The study analyses the problem domain, summarises typical organisational risks (duplication, loss of context, weak status control, information asymmetry between applicants and the dean's office), reviews existing approaches, and states engineering requirements for a centralised workflow. At the conceptual level, the work defines a request domain model, role-based access control (RBAC), a status lifecycle with an event log (timeline), and two submission modes: template-based and purpose-driven. Architecturally, the solution is implemented as client–server interaction over a REST API: a Node.js/Express backend with JWT authentication and MongoDB (Mongoose) for storing requests and their history; a Next.js frontend with server-side proxy routes for consistent authorised transport. The implemented software covers core scenarios: application submission, status updates by the dean's office with mandatory rejection reasons, user notifications, and faculty analytics. The results indicate improved controllability of the certificate workflow and a feasible basis for deployment in a university setting.*

_____

ЗМІСТ

ВСТУП

Оформлення академічних довідок в університеті давно не зводиться до суто формального «папірця» в межах канцелярського руху документів. У реальній роботі це сервісний процес із кількома рівнями, де одночасно діють різні ролі: студенти й викладачі ініціюють звернення, а перевірка даних, погодження та остаточна фіксація результату залишаються зоною відповідальності деканату. Коли ж немає чітко описаного життєвого циклу документа і спільних правил контролю, усе швидко перетворюється на розрізнені ручні дії. Тоді стає складно тримати під контролем те, що для управління критично, простежуваність операцій, повторюваність рішень, дотримання строків виконання, реальне навантаження на співробітників. Тому автоматизація тут, це не просто перенесення паперової форми в електронний вигляд, а побудова керованої сервісної моделі з орієнтацією на observability-driven моніторинг.

У дослідженні таку модель реалізовано як вебплатформу, через яку академічні довідки можна замовляти дистанційно. Кожна заявка трактується як окремий доменний об’єкт, із заданими атрибутами, перевіреними параметрами та керованими переходами між станами. В основі системи лежить фіксована статусна модель:

- SUBMITTED
- IN_REVIEW
- FORMING
- READY
- REJECTED

Саме цей ланцюжок описує повний шлях заявки, від реєстрації до завершення обробки або до аргументованої відмови. Окремо виділено механізм журналювання подій у форматі timeline: будь-яка зміна статусу фіксується разом із часом, ідентифікатором оператора та коментарем, який пояснює ухвалене рішення. У підсумку з’являється наскрізна traceability для всіх дій, а також база для аналізу причин затримок, помилок чи відхилень у процесі.

Беручи до уваги специфіку університетських процедур, система підтримує два різні сценарії створення запитів, які задає параметр requestMode: TEMPLATE та

PURPOSE. У режимі TEMPLATE користувач обирає готовий тип документа з каталогу CertificateType, після чого заповнює параметризовані поля, для яких наперед визначено формат введення, обов’язковість, порядок показу та підказки. Інший варіант, «Запит за призначенням», потрібен у ситуаціях, коли користувач не може сам визначити, яка довідка йому потрібна. Тут система не заганяє запит у жорстку класифікацію, натомість переводить невизначеність у керований процес: заявник описує мету отримання документа і вказує базові дані, спеціальність, групу та дату народження. Після цього уточнення типу довідки переходить до відповідальних працівників деканату. Так навіть неформалізоване звернення вбудовується у структурований цифровий процес, придатний і для аналітики, і для поступового розширення каталогу стандартних довідок.

Об’єктом дослідження визначено процеси подання, перевірки, опрацювання та фіксації результатів виконання заявок на отримання академічних довідок у межах адміністративного середовища закладу вищої освіти.

Предмет дослідження становлять методи, алгоритмічні підходи та програмно-архітектурні рішення, які застосовано під час проєктування й розробки вебсистеми. Вона забезпечує рольову модель доступу з розмежуванням прав користувачів (Student, Teacher, Dean, Admin), керування життєвим циклом електронних заявок із детальним журналюванням подій, централізований каталог довідок із динамічною параметризацією полів, а також аналітичний модуль для агрегації показників ефективності факультету на підставі статусів заявок, ролей користувачів, способів подання та часових рядів.

Мета кваліфікаційної роботи полягає у проєктуванні та програмній реалізації інформаційної системи у вигляді вебзастосунку для замовлення академічних довідок із вбудованими засобами аналітики й моніторингу. Система спрямована на формалізацію взаємодії між заявниками та працівниками деканату, забезпечення прозорої логіки переходів між етапами обробки документів і надання адміністрації актуальних статистичних показників. Реалізований інструментарій дає змогу відстежувати загальну кількість заявок, частку успішно виконаних запитів, поточне

операційне навантаження, добову та тижневу інтенсивність звернень, а також середній час обробки однієї заявки.

Для досягнення мети були визначені й виконані такі завдання:

1. дослідити предметну область та сформувати функціональні й технічні вимоги;
2. спроєктувати доменну модель даних, що охоплює профілі користувачів, структуру факультетів, типи довідок, заявки, наскрізні лічильники реєстраційних номерів і сервіс сповіщень;
3. реалізувати механізми автентифікації та рольової авторизації;
4. розробити модулі керування заявками й моніторингу змін статусів у реальному часі;
5. забезпечити валідацію та коректну обробку неповних або помилкових вхідних даних;
6. інтегрувати аналітичні інструменти для підтримки управлінських рішень і оцінювання операційної ефективності.

Програмна частина платформи побудована на стеку JavaScript, із Next.js для клієнт-серверної взаємодії та документоорієнтованою базою даних MongoDB, інтегрованою через ODM-рівень. Це дозволяє розглядати заявку як цілісну агреговану сутність із вкладеною історією змін і подій. Такий підхід спрощує параметризацію, полегшує еволюцію доменної моделі та зменшує потребу в складних міграціях структури даних.

У сформованому технологічному комплексі система подається не як звичайна електронна форма для подання заявок, а як інструмент процесного управління роботою факультету, де статусна модель, контроль доступу й аналітичний супровід виступають базовими архітектурними компонентами.

РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Характеристика предметного середовища та специфіка адміністративного документообігу у ЗВО

Адміністративні процедури в закладах вищої освіти одночасно спираються на державні нормативні вимоги, внутрішні правила факультетів і практику щоденної роботи зі зверненнями. Академічні довідки формально належать до документів індивідуального запиту, але їх підготовка завжди тягне за собою перевірку даних у внутрішніх університетських системах. Потрібно підтвердити належність студента до факультету, його поточний статус, а також інші навчальні атрибути.

Через це видачу довідки складно трактувати як одну ізольовану дію, насправді це ланцюг пов’язаних транзакцій, де кожен крок описаний правилами і підкріплений контрольними процедурами.

Для факультетського менеджменту регулярно виникає ще одна проблема, нерівномірність і неоднорідність вхідних запитів. Частина звернень типова й добре вкладається у заздалегідь заданий сценарій, але інші суттєво визначаються конкретним контекстом. На практиці заявники часто не оперують точною адміністративною термінологією, тому описують запит через життєву ситуацію або через мету, заради якої їм потрібен документ. У паперовому або частково цифровізованому документообігу такі звернення майже неминуче супроводжуються неформальними уточненнями, це можуть бути усні консультації, дзвінки чи особисті візити до деканату. У результаті складніше керувати процесом, межі відповідальності стають розмитими, а прозорість процедур падає.

Тому сучасна інформаційна система має не обмежуватися каталогом документів, їй потрібні механізми роботи і з такими запитами, де невизначеність висока. Автоматизація тут насамперед означає структурувати звернення і далі керовано спрямовувати його по процесу, не перетворюючи заявку на неконтрольований шматок довільного тексту.

У розробленій вебплатформі це вирішено поділом подання заявок на два режими, TEMPLATE («за шаблоном») та PURPOSE («за призначенням»).

TEMPLATE орієнтований на автоматизоване складання запиту для конкретного виду довідки, поля задаються параметризованими схемами. PURPOSE застосовується тоді, коли користувач не може сам визначити потрібний формат. За такого варіанту заявник вказує мету отримання документа й базові дані про себе, а уточнення типу довідки та кінцева класифікація виконуються в межах внутрішньої перевірки працівниками деканату.

Відповідно, доменна модель системи включає не лише заявки і довідки, а й динамічні метадані шаблонів, плюс наскрізний журнал подій, який забезпечує повну простежуваність і можливість відтворити всі кроки управлінського процесу.

1.1.1. Процес подання, опрацювання та видачі довідок: рольова модель доступу та життєвий цикл заявки

Функціональну архітектуру платформи вибудувано навколо чотирьох ролей, Student, Teacher, Dean і Admin. Таке розмежування потрібне не лише для організації роботи, воно має прямий інженерний зміст. Рольова модель тут працює не як формальне «обмеження прав», а як спосіб відокремити потенційно конфліктні сценарії, дотриматися принципу найменших привілеїв і підтримувати кероване перемикання станів документів.

Студенти та викладачі взаємодіють із платформою лише на рівні створення запитів. Заявка на цьому етапі фіксується як структурований цифровий об’єкт. Для TEMPLATE вона містить payload, який перевіряється відповідно до динамічної схеми полів, заданої для певного типу довідки. Для PURPOSE заявка накопичує формалізований опис мети та мінімальний набір академічних атрибутів, наприклад спеціальність, групу, дату народження й інші базові дані.

Щоб зберегти цілісність процесу, ініціатор не може самостійно змінювати статус поданої заявки. Редагування після відправлення допускається лише у визначених сценаріях, і будь-яке коригування автоматично повторно запускає процес, повертаючи звернення на початковий етап. Така схема прибирає типову для напівручного документообігу проблему «тихого» редагування даних.

Деканат у системі є операційним верифікатором і центральним вузлом процесу. Саме ця роль бере заявки в роботу, переводить їх між етапами життєвого циклу та

фіксує кінцевий результат. При цьому платформа не дозволяє відхиляти звернення без пояснення, статус REJECTED можна встановити лише разом з обов’язковим текстовим коментарем. У підсумку процес стає прозорішим, зменшується кількість повторних помилкових звернень, швидше зникає потреба в нескінченних уточненнях, а також з’являється база для ретроспективного аналізу типових причин відмов.

Administrator відповідає за метадані та конфігурацію, тобто за те, що підтримує цілісність системи й дозволяє їй рости. Це структура факультетів, призначення відповідальних осіб, системні довідники, каталог документів. З інженерної позиції поділ ролей відділяє повсякденний операційний контур від конфігураційних змін. У результаті ризик неконтрольованих помилок нижчий, а загальна стабільність і відмовостійкість вища.

Життєвий цикл заявки описано як скінченний автомат статусів:

- SUBMITTED фіксує факт створення заявки і очікування на первинну обробку
- IN_REVIEW означає, що оператор прийняв документ до перевірки й почав верифікацію
- FORMING відповідає етапу формування довідки, підготовці електронного або друкованого бланка
- READY сигналізує, що все завершено і документ можна видавати
- REJECTED позначає мотивовану відмову з обов’язковим поясненням

Щоб зберігалася повна простежуваність, кожна зміна статусу автоматично створює транзакційний запис у журналі подій (timeline). Там зберігаються час операції, ідентифікатор відповідального оператора та коментарі. Тому механізм моніторингу показує не лише поточний стан, він дозволяє відтворити весь ланцюг подій і сформувати масив даних, придатний для аналітики та оцінювання ефективності.

1.1.2. Дефекти ручного та частково автоматизованого підходу: причинно-наслідковий аналіз

Коли оформлення документів здебільшого тримається на ручних діях або рознесене між кількома неузгодженими цифровими каналами, першою проблемою

стає відсутність єдиного центра процесного керування. Дані починають рухатися через месенджери, окремі файли, усні уточнення чи локальні нотатки працівників.

У таких умовах розпадається сама доменна сутність «заявка», а разом із нею губляться зв’язки між етапом обробки, відповідальним виконавцем та історією взаємодій. Звідси випливає дублювання робіт, персональні дані вводяться повторно, копіюються між реєстрами, а перевірки виконуються щоразу заново при зміні працівника. Це не зводиться лише до людського фактора, така неефективність є прямим наслідком децентралізованої організації документообігу.

Друга системна слабкість, поступова втрата контексту під час обробки. Університетське звернення нерідко проходить через кількох співробітників. Якщо попередні дії, уточнення та проміжні рішення не закріплені у структурованій моделі, кожен наступний виконавець змушений заново відчитувати й тлумачити вимоги. Тоді з’являються розбіжності у трактуванні, росте час виконання, і падає узгодженість результату.

Окремий блок проблем пов’язаний з інформаційною асиметрією між заявником і адміністративним персоналом. Студент зазвичай не бачить реального стану виконання свого звернення, і фактично перебуває в невизначеності. Працівники деканату при цьому витрачають чимало часу на повторювані відповіді про готовність довідок. Комунікаційне навантаження росте не через складність підготовки документів, а через відсутність прозорості й передбачуваності процесу.

Непрозорі дедлайни напряму пов’язані з тим, що етапи життєвого циклу не формалізовані. Якщо не фіксуються час створення заявки, момент прийняття в роботу та завершення, складно об’єктивно оцінити середній час виконання або виявити періоди піків. Поліпшення регламентів тоді робиться інтуїтивно, без опори на статистику й емпіричні дані. Це погано узгоджується з підходом сучасних цифрових сервісів у вищій освіті, де контроль якості має спиратися на observability, прозорість і вимірювані метрики.

1.1.3. Класифікації, таблиці та аналітичні структури (фрагмент формалізації)

Таблиця 1.1

Класифікація типів академічних довідок у контексті університетських сервісів

Категорія	**Опис продукту**	**Типові параметри**	**Процесні ризики без формалізації**
Статусні	Підтвердження факту навчання/належності	факультет, група, спеціальність, період	часті уточнення, повторні перевірки облікових даних
Навчально-результативні	Відомості про дисципліни/успішність	дисципліни, семестри, кредити	помилки вибірки, неконсистентність з навчальним планом
Для соціальних/державних установ	Формулювання під зовнішній регламент	ціль, додаткові реквізити	потреба точних формулювань, підвищена ціна помилки
Для зовнішніх адресатів	Роботодавці, консульства тощо	ціль, мовні/формат ні вимоги	нестандартні поля, відсутність готового шаблона
“За призначенням” (PURPOSE)	Заявка без визначеного типу	опис мети + базові атрибути	некеровані уточнення без структурованого payload

Дана таблиця групує запити за способом їх формалізації, або через каталог шаблонів, або як звернення з неповною специфікацією («за призначенням»), де параметри визначаються під час обробки.

Таблиця 1.2

Класифікація ролей у системі та їх процесні повноваження (RBAC)

Роль	**Процесна функція**	**Дозволені дії щодо заявки**	**Обмеження, критичні для керованості**
Student	ініціація звернення	створення, перегляд власних, корекція в дозволених станах	відсутність права змінювати статуси; редагування реініціює подання
Teacher	ініціація звернення	аналогічно студенту	аналогічні інваріанти доступу
Dean	виконання/верифікація	перегляд потоку факультету, зміна статусів, рішення	відхилення вимагає обґрунтування; фіксується подія
Admin	керування метаданими	налаштування факультетів,	відокремлення метарівня від операційного потоку

Роль	Процесна функція	Дозволені дії щодо заявки	Обмеження, критичні для керованості
		каталогів, призначень	

Тут зафіксовано розподіл повноважень між Student, Teacher, Dean і Admin, а також межі допустимих дій у межах життєвого циклу заявки.

Таблиця 1.3

Функціональні можливості веб-платформи (узагальнена матриця)

Підсистема	Опис функції	Вхідні дані	Результат	Ключова доменна сутність
Каталог довідок	параметризовані шаблони з полями	опис, аудиторія, fields[]	доступні типи довідок для подання	CertificateType
Подання заявок	ініціація TEMPLATE або PURPOSE	payload / опис мети	заявка зі статусом SUBMITTED	CertificateRequest
Статус-трекінг	керування життєвим циклом	рішення, коментар, дата видачі	перехід статусу + подія в timeline	timeline[]
Сповіщення	повідомлення про значущі події	зміна статусу	нотифікація з deep-link	Notification
Аналітика факультету	агрегація показників	масив заявок факультету	метрики, тренди, топ-запити	аналітичні агрегати

Таблиця пов’язує функціонал системи з доменними сутностями та API-операціями, включно з каталогом, поданням, трекінгом, сповіщеннями та аналітикою.

Таблиця 1.4

Порівняння ручного та автоматизованого процесу в термінах процесної керованості

Ознака	Ручний/фрагментарний підхід	Централізована система
Об’єкт управління	неуніфікований “запит” у повідомленнях	доменна сутність “заявка” з атрибутами
Статус-контроль	ситуативні домовленості	формальні статуси SUBMITTED…REJECTED
Traceability	неможливість відтворити шлях рішення	подієва історія timeline з автором і часом
Маршрутизація звернень	залежить від виконавця/каналу	відображена в життєвому циклі та ролях
Оцінка строків	суб’єктивна	вимірювана через часові мітки станів
Аналітика	відсутня або неопераційна	метрики (total, pending, completionRate, середній час)

Таблиця порівнює процесні властивості, від централізації контексту до можливості відтворити історію та будувати аналітику.

Таблиця 1.5

Характеристика статусів заявки як елементів життєвого циклу

Статус	Процесне значення	Типовий тригер	Дані, що фіксуються для аналізу
SUBMITTED	формування черги	створення/повторне подання	submittedAt, подія в timeline
IN_REVIEW	прийняття в роботу	рішення деканату	reviewedAt, автор події
FORMING	виробництво документа	старт формування	formingAt, подія в timeline
READY	завершення виконання	готовність до видачі	readyAt, можливе pickupFrom
REJECTED	негативне рішення	відмова	обов’язковий decisionComment

Дана таблиця інтерпретує стани SUBMITTED / IN_REVIEW / FORMING / READY / REJECTED як етапи обробки з визначеними тригерами та фіксацією часових міток і коментарів.

1.1.4. UML-моделі: логіка взаємодії та процесна динаміка

Для архітектурної стабільності платформи принципово важливо чітко розвести відповідальність учасників і внутрішні механізми зміни станів заявок. Use Case-модель тут фіксує ізоляцію ролей, ініціатори запитів не мають доступу до операцій зі статусами, а представники деканату не створюють заявки, вони працюють лише з уже зареєстрованими зверненнями. Така декомпозиція узгоджена з RBAC і прибирає ризики змішування повноважень як в інтерфейсі, так і в API-логіці.

Use Case також розкриває відмінність двох сценаріїв подання. Для TEMPLATE система надає каталог довідок і динамічну схему полів, після чого виконується жорстка структурна перевірка payload. Для PURPOSE ключовим елементом стає текстовий опис мети, який деканат потім використовує як основу для класифікації.

Окремий прецедент стосується аналітики факультету для ролі декана, це не просто статичні звіти, а інструмент оперативного контролю, який спирається на агрегацію поточних станів заявок і часових характеристик життєвого циклу.

Activity-модель зосереджується на переходах між статусами та фіксації подієвого сліду. Після подання система ставить SUBMITTED і створює перший запис у timeline. Далі оператор деканату переводить заявку в IN_REVIEW, за потреби в FORMING, і після завершення підготовки в READY. Альтернативна гілка завершується REJECTED, при цьому коментар обов’язковий, він створює процесне обґрунтування відмови.

Кожна зміна стану автоматично передає дані в аналітичний модуль, де рахуються метрики, середній час обробки, completion rate та поточне навантаження. Останнє можна формалізувати як суму заявок у проміжних станах: {SUBMITTED} + {IN_REVIEW} + {FORMING}.

Нижче наведено специфікації PlantUML, сумісні з описаною логікою.

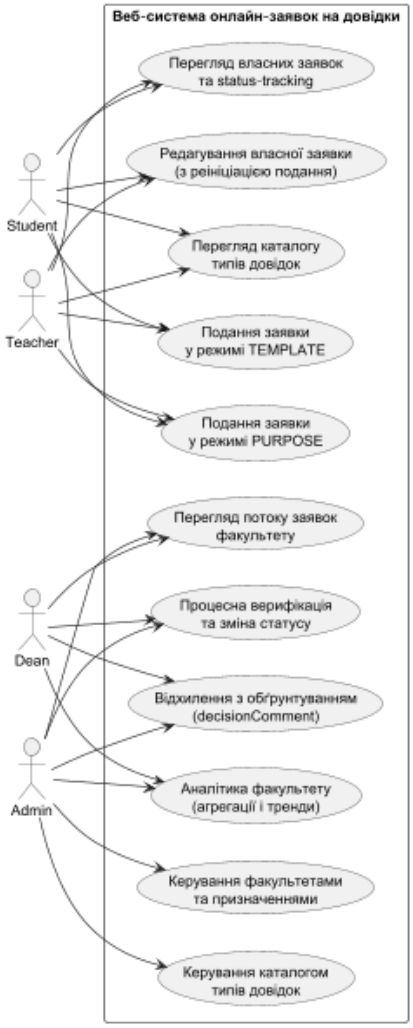

Рис. 1.1. Діаграма випадків використання веб-системи онлайн-заявок на довідки

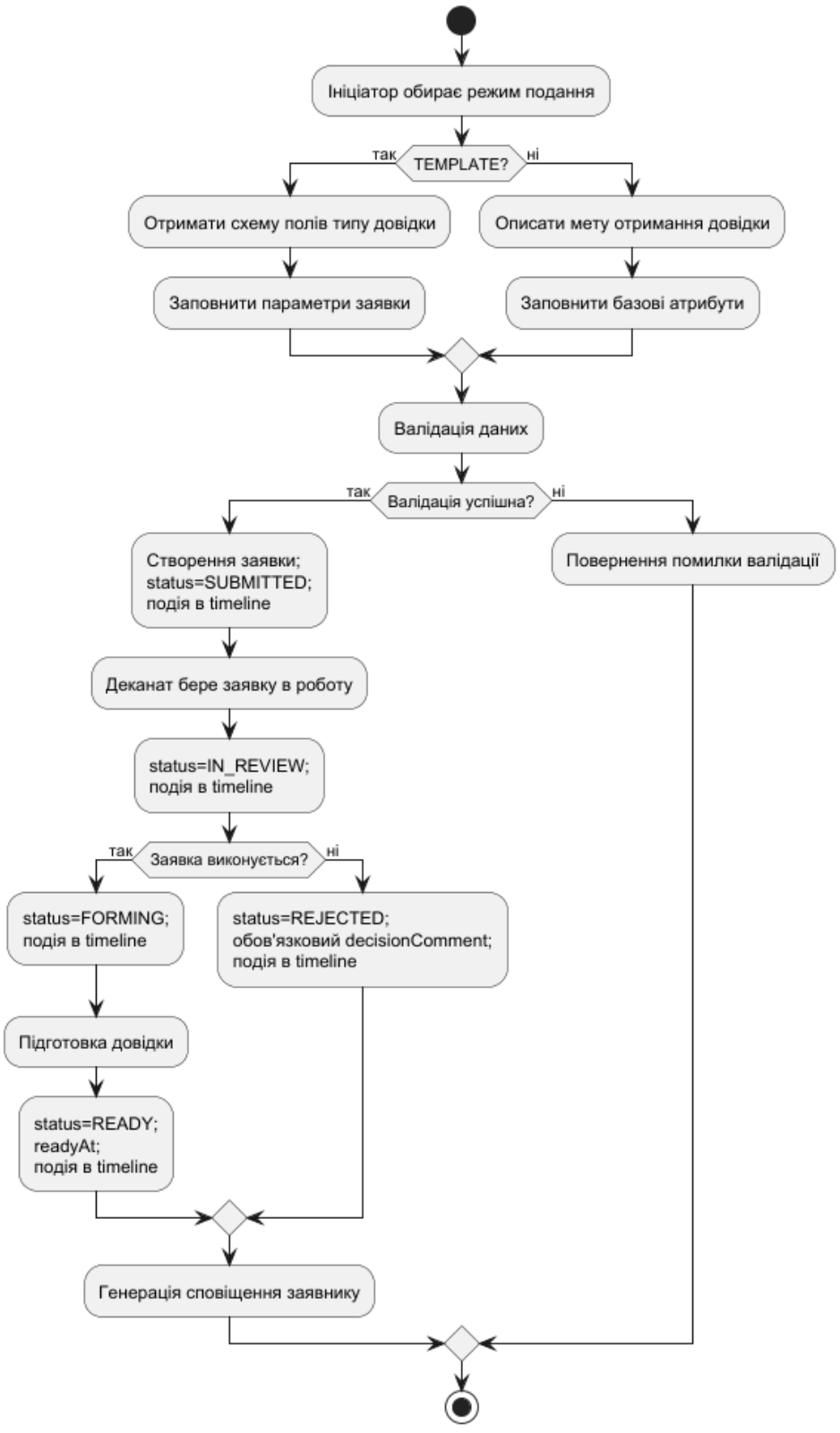

Рис. 1.2. Діаграма діяльності процесу подання заявки та її опрацювання деканатом

1.2. Огляд наявних рішень та аналогів у площині електронного документообігу й університетських сервісів

Цифрові рішення для адміністрування університетських сервісів умовно можна розвести на дві групи.

Перша, універсальні системи електронного документообігу (СЕД), що переважно підтримують внутрішні процеси реєстрації, погодження й маршрутизації службової кореспонденції. Вони зазвичай мають сильні механізми доступу, формалізовані реєстри й детальне журналювання. Але в сервісних сценаріях, зокрема для видачі академічних довідок, видно їх обмеження, статична модель «документа» витісняє динамічне розуміння «заявки» як процесної сутності. Життєвий цикл звернення тоді відображається частково, а статуси часто перетворюються на технічні позначки без подієвого змісту. Трекінг для користувача або відсутній, або існує як побічний продукт внутрішніх процедур.

Друга категорія, студентські портали й персональні кабінети, де замовлення довідок є однією з функцій поруч з іншими академічними сервісами. Їхній сильний бік, орієнтація на зручність користувача та простіша взаємодія. Проте слабкість часто в спрощеній серверній логіці. Після подання форми людина зазвичай бачить лише загальний статус «подано», а маршрутизація, перевірка й виконання відбуваються поза системою, через зовнішні канали або офлайн. Аналітика в таких продуктах обмежується загальними цифрами звернень і зазвичай не дає сегментації за типами, режимами подання чи часом. Це звужує можливості адміністрації керувати навантаженням і ресурсами на підставі даних.

Запропонована платформа відрізняється від обох підходів тим, що статус-трекінг і наскрізна простежуваність (traceability) закладені в ядро доменної моделі. Зміна стану заявки трактується не як елемент інтерфейсу, а як системна подія, що одразу впливає на аналітичні розрахунки. Базові метрики, pending, completion rate і середній час обробки, рахуються автоматично з урахуванням ролей ініціаторів та сценаріїв TEMPLATE і PURPOSE. Тому аналітичний модуль стає точним цифровим відображенням реальних адміністративних процесів факультету, а не просто статистичною надбудовою.

1.3. Постановка задачі дослідження: інженерні вимоги, архітектурні рішення та технологічне обґрунтування

Центральне завдання дослідження, спроєктувати веборієнтовану інформаційну систему для керованого опрацювання запитів на академічні довідки, коли вхідні

звернення структурно різні й неоднорідні. У системній інженерії це означає створити централізований процесний контур, де заявка є базовою доменною сутністю, а її життєвий цикл, формалізованою послідовністю станів із визначеними переходами, журналюванням подій і чітким розподілом відповідальності. Така архітектура дає деканату керованість дій і зменшує інформаційну асиметрію завдяки прозорому відстеженню статусів та автоматизованим сповіщенням.

Архітектурні вимоги задають масштабування у двох взаємозалежних площинах, за обсягом транзакцій і за розширенням доменної моделі. Для навантаження потрібні продумана структура бази даних, коректні індекси й механізми динамічного агрегування метрик для моніторингу. Для розвитку предметної області важливий гнучкий каталог типів довідок із параметризованими полями та підтримка режиму PURPOSE для запитів з високою невизначеністю. Якщо цього немає, набір жорстко заданих форм швидко втрачає практичну цінність, а користувачі знову переходять до неформальних каналів.

З огляду на вимоги до розмежування доступу і контролю бізнес-логіки введено RBAC. Змінювати стани заявок можуть лише адміністратори та працівники деканату, ініціатори обмежені створенням і переглядом власних звернень. Також задано процедурні інваріанти, зокрема заборона переводити заявку в REJECTED без текстового обґрунтування. Це підтримує дисципліну процесу і робить рішення відтворюваними.

Логування подій виконане на рівні доменної сутності, журнал стає не просто технічним записником, а структурованим потоком даних для аналітики. Вибір Next.js пояснюється доцільністю єдиного технологічного стеку для клієнтської і серверної частин, включно з рольовими інтерфейсами та API-маршрутизацією. Це зменшує ризик фрагментації шарів і спрощує захищений доступ до серверної логіки через уніфікований роутинг. MongoDB як основна база даних обрана через здатність ефективно працювати зі складними та змінними структурами доменних об’єктів, включно з історією подій і payload, який залежить від типу довідки. Поєднання документоорієнтованої моделі з ODM-рівнем дає баланс, схема може

еволюціонувати, але валідація на рівні застосунку лишається контрольованою, що критично важливо, коли номенклатура документів постійно розширюється.

Разом ці положення утворюють концептуальну й технологічну основу для подальшої реалізації системи. Предметну область формалізовано через доменну модель, обґрунтовано потребу у наскрізній простежуваності та статус-контролі, визначено ролі доступу й інваріанти життєвого циклу заявок. Обраний стек відповідає вимогам масштабування, структурованої маршрутизації, валідації, журналювання операцій та подальшого статистичного аналізу процесів.

ВИСНОВКИ ДО РОЗДІЛУ 1

Охарактеризоване предметне середовище демонструє типові деформації адміністративного документообігу у закладах вищої освіти: значна частина операцій залишається ручною або фрагментарною, інформація щодо звернень розподілена між каналами комунікації, а відсутність централізованого репозиторію заявок підсилює дублювання дій і ускладнює контроль повноти даних. У таких умовах відсутній уніфікований статус-контроль як інструмент керованості: заявник не отримує достовірної картини етапів опрацювання, а виконавець несе додаткові витрати на повторні уточнення та передачу контексту між учасниками процесу. Результатом є інформаційна асиметрія між ініціатором звернення та факультетським підрозділом, а також непрозорість фактичних строків виконання, що унеможливлює об’єктивну оцінку навантаження та якості сервісу.

Запропонована інформаційна система адресує зазначені дефекти шляхом централізації оброблення заявок у єдиному веб-контурі та формалізації процесу як життєвого циклу доменної сутності з фіксованими статусами, часовими мітками й подієвим журналом змін. Автоматизація подання заявок забезпечує структурованість вхідних даних через каталог типів довідок із параметризованими полями, а також підтримує альтернативний сценарій подання «за призначенням», який інституціоналізує неповну специфікацію запиту без виходу за межі формалізованої моделі. Механізми рольового доступу розмежовують повноваження ініціаторів і виконавців, знижуючи ризик конфліктних дій щодо стану заявки та забезпечуючи відповідність процесу принципам мінімально необхідних привілеїв.

Концептуальне моделювання відіграє методологічно ключову роль: UML-діаграми варіантів використання та активності експлікують межі системи, послідовність бізнес-операцій і точки прийняття рішень, тоді як опис доменної моделі узгоджує термінологію предметної області з програмною реалізацією. Формалізація рольової структури та життєвого циклу заявки задає основу для проєктування інтерфейсів, правил авторизації та контрольних інваріантів процесу, зокрема обов’язковості обґрунтування відмови як засобу процесної верифікації.

Отримані результати розділу узагальнюють проблематику й обмеження традиційних практик оформлення довідок і формулюють інженерні вимоги до централізованого документообігу з прозорим статус-трекінгом та можливістю аналітичного моніторингу потоку звернень. Це створює методологічну та проєктну базу для подальшої програмної реалізації системи, оскільки фіксує доменні інваріанти, рольові межі та логіку переходів станів, які мають бути безперервно відображені у програмній архітектурі та експлуатаційних сценаріях веб-платформи.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз предметної області

Потреба в цьому розділі виникає тоді, коли вимоги до сервісу академічних довідок потрібно перевести з описового рівня в інженерний, тобто у вигляд доменної моделі. Йдеться не про загальні формулювання, а про чітко визначені сутності, їхні зв’язки та обмеження, які дають змогу тримати процес під контролем і підтримувати узгоджені дані.

У межах проєкту предметна область розуміється як сервіс, прив’язаний до факультетів, у якому ініціатор звернення та виконавець перебувають у різних зонах відповідальності. Завданням стає не просто передати інформацію, а зробити так, щоб вона читалася однозначно й могла бути відтворена в логіці централізованого документообігу.

Базові об’єкти предметної області утворюють замкнену схему:

- заявка
- користувач
- факультет
- тип довідки
- статус
- історія змін
- сповіщення
- адміністрування

У цій схемі заявка (CertificateRequest) не зводиться до запису в БД, вона виступає носієм процесного контексту. Усередині поєднуються атрибути різного змісту:

- організаційні (факультет як зона виконання)
- суб’єктні (ініціатор, роль, контактні дані)
- процесні (статус, часові позначки)
- змістові (payload або опис призначення)
- пояснювальні (коментар рішення, послідовність подій)

Така композиція є принциповою, бо видача довідки в ЗВО, це не одна дія, а ланцюг рішень, кожне з яких потрібно фіксувати як факт і як підставу.

Користувач (User) задає ідентичність та є джерелом повноважень у системі. Ролі Student і Teacher описують два типи ініціаторів, механіка подання для них спільна, але сценарії застосування каталогу довідок можуть відрізнятися. Роль Dean відповідає виконавчій ланці факультету, де ухвалюються процесні рішення та формується результат. Admin визначає рівень метакерування: він не замінює деканат у щоденній роботі, зате відповідає за довідники, структурні зв'язки та правила доступу. Принципово, що рольова модель тут функціональна, а не номінальна, вона задає, хто створює процесний об'єкт, хто має право змінювати його стан, а хто лише спостерігає, отримуючи підтвердження через статус-трекінг і хронологію подій.

Факультет (Faculty) у цій предметній області не є простим «довідником», він працює як інструмент масштабування та маршрутизації. Саме він визначає виконавця заявки, перелік доступних типів довідок, логіку аналітичних узагальнень і межі видимості потоку заяв. Прив'язка дисциплінує дані: заявка не існує поза контекстом, і деканат не може обробляти заявки іншої одиниці без відповідного права.

Тип довідки (CertificateType) формує каталог і перетворює неструктуроване «прохання» на формалізовану специфікацію. У проєкті каталог зроблено як набір параметризованих шаблонів: схема полів (fields[]) визначає, які атрибути обов'язкові, у якому форматі їх подавати, які підказки та обмеження діють. Тобто акцент переноситься з жорстко заданих форм на доменно керовані метадані, номенклатуру довідок і вимоги до полів можна розширювати чи уточнювати без руйнування архітектури й без дублювання логіки на клієнті.

Поряд із шаблонним поданням враховано ситуацію, коли користувач не може коректно вибрати тип довідки. Для цього введено режим «довідка за призначенням» (PURPOSE), який оформлює невизначеність як керований процесний випадок. Користувач подає опис мети та мінімально достатні атрибути, а деканат класифікує й уточнює звернення вже всередині централізованого документообігу. У доменних термінах це дає дві важливі властивості: запит не губиться в комунікаціях і зберігає

траєкторію, також накопичуються дані для розвитку каталогу, бо частота purpose-звернень і їхній зміст показують прогалини номенклатури.

Статус-трекінг і історія змін є опорою узгодженості процесу. Статуси SUBMITTED, IN_REVIEW, FORMING, READY, REJECTED, це не декоративні мітки інтерфейсу, а фази, в яких перебуває заявка. Історія змін (timeline) поєднує статус із фактом його встановлення: зберігається час, автор дії та контекст, завдяки чому рішення можна відтворити. Timeline знімає типову для адмінпроцедур проблему «пам’яті виконавця», історію тримає система, а не конкретна людина, тому зменшуються ризики втрати контексту, спірних ситуацій і повторних уточнень без документального підґрунтя.

Сповіщення (Notification) виконують не другорядну роль, вони підтримують процесну дисципліну. Завдяки їм зменшується інформаційна асиметрія: заявник бачить значущі зміни, а деканат не витрачає час на повторні відповіді щодо статусу. Сповіщення формуються як похідні від процесних подій, а не вручну, тому тексти узгоджені, є прив’язка до конкретної заявки й зручний перехід до потрібного контексту.

Вхідні дані системи охоплюють:

- дані користувача разом із факультетним контекстом
- каталог типів довідок і схеми полів
- подання заявки в режимі TEMPLATE або PURPOSE
- процесні рішення деканату, у тому числі коментар відмови та параметри видачі

Вихідні дані, це доступний каталог, створені заявки з ідентифікаторами, поточним статусом і історією змін, сповіщення, а також аналітичні структури факультету, що будуються з процесних атрибутів.

Обмеження доцільно описувати як набір структурних і процесних правил.

Структурні правила відповідають за коректність значень:

- статуси й режими належать фіксованим множинам
- поля шаблонів мають визначені типи
- частина атрибутів має ліміти довжини

- факультетні ідентифікатори є унікальними

Процесні правила задають, хто і коли впливає на стан заявки: зміна статусу віднесена до юрисдикції деканату або адміністратора; відхилення потребує мотивованого коментаря; редагування ініціатором можливе лише так, щоб не ламати узгодженість, і тягне повторний вхід у «чергу подання» з фіксацією події.

У підсумку формується доменна модель, яка підтримує маршрутизацію та централізований документообіг, не залишаючи підстав переходити в неформальні канали. Підсумок аналізу предметної області, це уточнене розуміння первинних сутностей, критичних зв’язків і необхідних інваріантів для статус-трекінгу, traceability і рольового розмежування доступу. На цій основі надалі обґрунтовуються архітектурні рішення, UML-проекції та спосіб реалізації business-workflow.

2.2. Проєктування системи

Проєктування орієнтоване на те, щоб процесні механізми не можна було обійти на рівні клієнта, а поточний стан заявки та журнал змін залишалися узгодженими незалежно від способу взаємодії із системою. За такого підходу сервер стає єдиним носієм доменних правил, а клієнт відповідає за контрольований ввід та відображення.

Архітектура побудована як клієнт-сервер із поділом відповідальності між шарами.

На рівні представлення інтерфейс відрізняється за ролями.

Студенти та викладачі переглядають каталог, створюють заявки за шаблоном або «за призначенням», бачать власні звернення, відстежують статуси й історію, працюють зі сповіщеннями.

Деканат працює з чергою заявок факультету, застосовує фільтри за станами, відкриває деталі, переводить заявки між етапами, формує обґрунтовані відмови, задає умови видачі або готовності, переглядає звітність.

Для адміністратора передбачено керування факультетами, призначення відповідальних, підтримку каталогу типів довідок, доступ до аналітики і, за потреби, до процесної конфігурації.

Доменна логіка зосереджена на прикладному рівні. Передусім забезпечуються автентифікація та RBAC-авторизація: дії зі статусами доступні лише ролям із

відповідними правами, а ініціатори обмежені власним контуром доступу. Далі йде перевірка даних: у шаблонному режимі вона базується на схемі полів вибраного типу довідки, а в режимі PURPOSE, на обов'язкових атрибутах, достатніх для подальшого тлумачення звернення. Окремо реалізовано правила переходів між статусами, зміни допускаються лише в межах заданих умов, із часовими мітками та автоматичним додаванням записів у timeline. Також цей рівень відповідає за створення сповіщень на основі подій і за аналітику, яка агрегує інформацію щодо заявок факультету для оцінки навантаження та структури звернень.

Рівень зберігання даних організовано навколо доменних агрегатів. Ключовий момент, CertificateRequest містить не тільки актуальний стан, а й повну процесну історію у вигляді timeline. Це зменшує ризик розбіжностей між статусом і послідовністю змін, бо новий стан і відповідна подія зберігаються в межах одного агрегату. Документно-орієнтована модель також дає змогу тримати payload як гнучку структуру під конкретну схему полів, не змінюючи структуру БД після правок у каталозі довідок. Окрема увага відведена бізнес-процесу як керованому workflow. Система підтримує завершені сценарії, у яких кожен етап має визначений стан. Для студента або викладача все починається з вибору типу довідки та перегляду шаблону. Після заповнення полів і успішної перевірки створюється заявка зі статусом SUBMITTED, це точка входу в процес, вона потрапляє в чергу факультету, а користувач може відстежувати зміни.

У режимі «за призначенням» шаблон не застосовується, користувач описує мету та подає базові параметри, проте заявка так само створюється як SUBMITTED, а тлумачення змісту переходить до деканату.

Для деканату робота стартує з аналізу потоку заяв. Перехід у IN_REVIEW означає формальне взяття в опрацювання: фіксується відповідальність і реєструється подія. Після перевірки даних деканат переводить заявку або у FORMING, відділяючи підготовку довідки від аналізу, або завершує її статусом REJECTED. Відмова без підстав не допускається, для REJECTED потрібен коментар із причиною. Перехід у FORMING означає, що даних достатньо і підготовка документа розпочалася. Статус READY фіксує готовність та, за потреби, умови отримання. Кожна зміна

супроводжується подією та автоматичним сповіщенням заявнику, прозорість workflow підтримується без зайвих комунікацій.

Нетипові ситуації описано як природну частину процесу, а не як «помилки». Якщо під час подання дані неповні чи некоректні, система одразу повертає повідомлення про помилки перевірки й не пропускає заявку в чергу. Якщо деканат не може однозначно зрозуміти звернення, заявка відхиляється з поясненням того, що саме потрібно уточнити. Після правок користувач подає повторно, і заявка знову стає SUBMITTED. Спроба змінити заявку після входу в активну фазу виконання блокується, оскільки це ламає процесний контекст, зміни мають проходити через цикл «відхилення → уточнення → повторне подання». Спроба встановити REJECTED без пояснення так само відхиляється. Загальний принцип один, система прибирає неформальні сценарії і залишає тільки переходи, які можна інтерпретувати як стани й події.

У рамках UML рішення показано трьома групами моделей. Use Case-модель задає межі системи та відповідність функцій ролям: ініціатори працюють з поданням і моніторингом, деканат, зі зміною станів, аналітики, із формуванням звітів та прийняттям рішень, адміністратор, зі структурою і довідниками. Class-модель фіксує центральний агрегат CertificateRequest з атрибутами status, payload, decisionComment, часовими мітками та timeline. CertificateType задає структуру введення, User і Faculty формують контекст доступу й виконання, Notification забезпечує процесну комунікацію. Activity-модель показує послідовність: подання, перевірка, постановка в чергу, початок обробки, завершення або відхилення, повторне подання після уточнення, перехід у READY, із причинно-наслідковим зв’язком між дією, статусом, подією й сповіщенням.

Запропоновані рішення формують систему, у якій централізація даних, контроль статусів і трасованість процесів є взаємопов’язаними. Заявка виступає єдиним джерелом істини, зміна стану завжди супроводжується подією, а події стають основою для сповіщень і аналітики. Через це виникає потреба в алгоритмічному забезпеченні, яке задає технічні правила для коректного workflow і узгодженості процесних даних.

2.3. Алгоритмічне забезпечення

Алгоритмічне забезпечення в цьому проєкті, це набір серверних процедур і правил, які підтримують коректну роботу доменних сутностей протягом життєвого циклу заявки. На відміну від математичних моделей, тут фіксуються технологічні й архітектурні інваріанти, що забезпечують узгодженість даних, керованість станами та відтворюваність процесів.

Алгоритм ініціації заявки починається з визначення режиму подання та відповідних перевірок. Для TEMPLATE сервер отримує ідентифікатор типу довідки, завантажує схему полів і перевіряє наявність обов'язкових атрибутів, формат значень, відповідність типів введення, а для полів вибору, належність значень до дозволених. Для PURPOSE сервер перевіряє, чи описано призначення довідки та чи подано базові атрибути, без яких деканат не зможе коректно розглянути звернення у факультетному контексті.

Після успішної перевірки сервер формує об'єкт заявки: призначає факультет та ініціатора, встановлює режим подання, задає початковий статус SUBMITTED, фіксує час подання, закріплює ідентифікаційні дані заявника в межах заявки й додає стартову подію в timeline. Від цього моменту заявка працює як процесний об'єкт централізованого документообігу і стає єдиним узгодженим джерелом інформації щодо звернення.

Алгоритм обробки деканатом задає статус-орієнтований workflow з контрольованими переходами. Сервер перевіряє роль виконавця та належність заявки до факультетного контуру і лише після цього дозволяє зміну стану. IN_REVIEW є механізмом явного взяття в роботу, фіксується час фази, створюється подія та надсилається сповіщення. FORMING відображає початок практичного формування довідки й відділяє його від аналізу, це важливо і для внутрішньої організації черги, і для прозорості для заявника. READY означає завершення: фіксується час готовності та, за потреби, дата або умови видачі, далі створюється подія і сповіщення. REJECTED вимагає коментаря, сервер блокує відмову без пояснення, бо вона робить стан нечитабельним і провокує повторні звернення без розуміння причини. Тут

коментар є процесним елементом, своєрідним «інтерфейсом уточнення», який повертає роботу до керованого циклу корекції.

Уточнення і повторне подання реалізуються через контрольоване редагування ініціатором. Сервер перевіряє, що користувач є ініціатором саме цієї заявки, і що поточний стан допускає зміни. Якщо умови виконано, оновлюються дані звернення, очищуються атрибути рішення деканату, а статус повертається до SUBMITTED. У timeline додається подія повторного подання. Це не «вільне редагування» будь-коли, зміна специфікації допускається лише як формалізований крок, що повертає заявку в чергу й залишає traceability попереднього рішення та причин корекції.

Збереження історії змін спирається на правило: кожна значуща модифікація має супроводжуватися процесною подією. Створення, редагування і кожен перехід статусу записуються в timeline з часом, автором і контекстом. Завдяки цьому поточний статус завжди читається у зв’язці з історією, а система має дані для аудиту й управлінського аналізу, можна відтворити, хто і коли ухвалював рішення та з яким коментарем. Алгоритм сповіщень продовжує статус-трекінг як механізм комунікації.

Після кожної важливої події сервер створює Notification, визначає адресата й формує посилання на заявку в інтерфейсі, релевантному ролі. Так заявник не дізнається про готовність або відмову випадково, комунікація стає передбачуваною частиною workflow і не залежить від особистої дисципліни виконавця.

Аналітичні зведення факультету будуються з інтерпретованих процесних атрибутів: статусу, режиму подання, ролі ініціатора, часових міток фаз, назви звернення. Сервер робить зведення за статусами, виділяє активне навантаження як заявки у фазах очікування та виконання. Також обчислюються розподіли за ролями ініціаторів, що дає змогу бачити структуру попиту, і розподіли за режимами TEMPLATE/PURPOSE, які показують частку нестандартних звернень та напрями розвитку каталогу. Окремо формуються часові тренди подання та завершення, визначаються найчастіші типи звернень, відображається перелік останніх активностей. Оперативність оцінюється зіставленням моменту подання з моментом завершення для заявок зі статусом READY або REJECTED.

Аналітика тут не є окремою «паралельною» частиною, вона виростає з доменних даних, накопичених під час status tracking, і відображає фактичний рух заявок у workflow.

2.4. Обмеження та проблеми традиційної організації процесу, що зумовлюють вимоги до системи

Потреба у централізованій веб-системі найкраще помітна тоді, коли розбирати слабкі місця ручного або напів автоматизованого опрацювання довідок. Такі підходи майже завжди дають повтори, і в самих заявках, і в діях.

Коли немає одного каналу подання й спільного об’єкта типу «заявка», заявники підстраховуються, надсилають те саме ще раз, додають уточнення окремими листами, пишуть «нагадування» в інших каналах. Для деканату це не означає більше корисної роботи, натомість зростає інформаційний шум, складніше виставляти пріоритети, легше помилитися під час зіставлення, яке звернення справді актуальне.

Друга проблема, інформаційна асиметрія. Без прозорого відстеження стану заявник не відрізняє ситуацію «заявку не бачили» від «заявка вже в роботі», а деканат змушений витрачати час на відповіді про статус. Наслідки ширші, ніж просто більше повідомлень, з’являється конфлікт очікувань: заявник сприймає мовчання як затримку або ігнорування, тоді як виконавець вважає звернення прийнятим і таким, що опрацьовується.

У ручних процесах, де немає одного джерела даних, легко губиться контекст. Вхідні відомості розкладені між повідомленнями, файлами й усними уточненнями, і будь-яка заміна виконавця або перерозподіл задач у деканаті змушують заново «склеювати» зміст та відновлювати передісторію. Це прямо впливає на строки і підвищує ризик, що частина вимог буде втрачена або передана зі спотворенням.

Через людський фактор нестійкість проявляється не лише як поодинокі помилки, вона перетворюється на системний ризик, бо немає інваріантів процесу, журналювання та контролю станів. Коли фази на кшталт «прийнято в роботу», «формується», «готово» не оформлені як формальні технологічні стани, відсутній об’єктивний спосіб визначити, де виникає вузьке місце. Так само складно відрізнити загальне накопичення черги від затримки на конкретному кроці. У підсумку

управлінські рішення стають реактивними, їх ухвалюють на основі окремих випадків і фрагментарних вражень, без опори на зведені показники.

Саме ці обмеження традиційних підходів і формують вимоги до централізації, відстеження статусів, подієвої історії та аналітики.

2.5. Розширений огляд аналогів і порівняльні висновки

Практика застосування цифрових інструментів у довідкових та адміністративних процедурах ЗВО містить кілька класів рішень. Частково вони збігаються за функціями, але суттєво різняться за процесною зрілістю.

Перший клас, універсальні системи електронного документообігу, які орієнтовані на реєстрацію документів, маршрутизацію погоджень, розмежування доступу та архів. Зазвичай у них добре опрацьовані права й протоколювання. Однак у довідкових послугах їхня пристосованість часто обмежена: логіка таких систем будується навколо «документа як одиниці обліку», тоді як для довідкової послуги базовою сутністю виступає «заявка як процесний агрегат». Через це статус для кінцевого користувача або не передбачений, або має нечітку семантику, а налаштування каталогів і параметрів потребує значних адміністративних дій. У результаті система стає інерційною до змін на рівні факультету.

Другий клас, університетські портали й «особисті кабінети», де довідки є одним із сервісів у ширшому наборі функцій. Їхній плюс, низький поріг входу та природна інтеграція з навчальними модулями. Проте процесний контур у багатьох випадках зводиться до фіксації факту «подано», без детальних статусів супроводу і без історії змін, яку можна стабільно відтворити. Аналітика часто обмежується підрахунком звернень, без розкладання за фазами робочого процесу, без розбору нестандартних запитів і без оцінювання оперативності. Деканат отримує канал надходження, але не отримує інструмент процесного керування.

Третій клас, індивідуальні «легкі» реалізації: форми та заяви на базі конструкторів, поштові скриньки з ручним опрацюванням. Вони дають вхідний канал, але не забезпечують централізованого документообігу в процесному сенсі. Статуси не є керованими об’єктами, історія або відсутня, або фрагментована, права

доступу та відповідальність залишаються неформалізованими. Саме тут найчастіше з’являється дублювання звернень, втрата контексту й непрозорі строки.

Порівняння показує, що запропонована система відрізняється поєднанням каталогу типів довідок із процесною дисципліною відстеження статусів і подієвою історією. Підтримуються як стандартизовані шаблонні звернення, так і запити «за призначенням», без винесення останніх у зовнішні канали. Аналітика факультету закладена в доменну логіку: агрегування статусів, режимів подання, ролей ініціаторів і часових характеристик показує реальний рух заяв у робочому процесі та створює основу для управлінських рішень.

Саме цей зв’язок, централізація даних, керовані стани, історія змін і аналітичні узагальнення, у наведених класах рішень або слабко виражений, або відсутній, і в цьому полягає прикладна цінність запропонованого підходу.

2.6. Аналітичні узагальнення: причинно-наслідкові зв’язки між проблемами предметної області та рішенням

Складнощі ручного налагодження процедур опрацювання запитів зводяться до однієї базової причини: немає чітко визначеного об’єкта процесу для керованої роботи й одночасно відсутні незмінні умови, які гарантували б правильну послідовність кроків.

Коли немає єдиної «точки входу», виникає дублювання, бо ніщо не гарантує, що запит буде ідентифіковано як одне джерело інформації.

Якщо звернення не інтегроване в доменну модель і не має фіксованої хронології, воно втрачає цілісний зміст, а дані розходяться між каналами зв’язку та людьми.

Непрозорість строків зумовлена не тільки порушенням регламенту, а браком маркерів чітких станів та фіксації часу для кожного етапу.

Інформаційна нерівність проявляється тоді, коли заявник не бачить інтерпретованого статусу і повної історії руху звернення. Запропонована система прибирає ці вади через системний підхід.

Централізація означає, що запит стає уніфікованим носієм інформації, поточного стану та історії обробки. Відстеження станів розбиває процес на зрозумілі етапи, кожен етап має відповідальну роль і дозволені наступні дії.

Подієвий журнал гарантує, що кожна важлива зміна зберігається як факт із контекстом.

Аналітика робить процес спостережуваним за обсягами роботи, типами звернень і результатами, що дозволяє переходити від управління «за відчуттями» до управління на основі даних.

Режим «за призначенням» знімає проблему неповного переліку в класифікаторах: система не блокує запит і водночас не виводить його в позасистемні канали, зберігаючи контроль і збираючи матеріал для розвитку каталогу.

Отже, оновлена версія розділу утворює узгоджену концепцію:

1. аналіз предметної області виділяє елементи процесу та обмеження
2. проєктування переносить їх в архітектуру з RBAC-доступом, статус-трекінгом і traceability
3. алгоритмічний опис деталізує логіку роботи системи в типових і нетипових ситуаціях, а також показує, як з процесних даних утворюються сповіщення і зведена аналітика.

Така структура робить текст придатним для прямого включення до кваліфікаційної роботи без суттєвого перероблення.

ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому розділі окреслено головні засади проблеми, яка виникає через переважно ручне або частково автоматизоване опрацювання документів під час оформлення академічних довідок. Показано, що за відсутності єдиного сховища даних та цілісного ланцюга процесів з’являються повторні звернення і втрачається зміст запитів на етапах обробки. Доведено, що нечіткий контроль статусів у звичних практиках не дає прозоро відстежувати рух справи й підвищує ризик непередбачуваних затримок. Встановлено, що нерівність поінформованості між ініціатором запиту та виконавцями збільшує витрати на комунікацію й погіршує керованість процесу.

З урахуванням цих висновків запропоновано підхід до усунення недоліків через створення централізованого інформаційного інструментарію, зосередженого на формалізованому опрацюванні звернень. Підкреслено, що автоматизація має охоплювати не лише подання, а й регламентоване проходження заданих етапів із фіксацією рішень. Показано, що розроблений інструментарій зводить дані в одну точку й формує єдине джерело достовірної інформації про зміст запиту, його стан та історію змін. Зазначено, що стандартизація життєвого циклу звернення й упорядковане управління статусами зменшують потребу в неформальних уточненнях і знижують ризик суперечливих тлумачень.

Обґрунтовано, що рольова модель доступу гарантує функціональну керованість, розділяючи ініціацію запиту від ухвалення рішень щодо стану та від виконання. Зафіксовано, що структурований журнал змін підвищує відтворюваність процесу й підтримує контроль ухвалених рішень. Узгоджено набір ключових елементів предметної області та зв’язків між ними як інформаційну основу системи. Показано, що визначення вхідних і вихідних даних та умов щодо них є необхідним для коректної обробки звернень.

Аргументовано значущість концептуального моделювання як способу формалізації меж системи та її робочої будови. Наголошено, що UML-схеми закріплюють логіку взаємодії сторін із системою, структуру основних сутностей та динаміку процесу обслуговування запиту. Встановлено, що інформаційна модель забезпечує узгодженість проєктних рішень щодо збереження відомостей, доступу та моніторингу станів. Визначення ролей і процесних обмежень закладає основу для однозначної реалізації бізнес-правил, а опис життєвого циклу звернення є підґрунтям для контрольованого робочого потоку та зменшення невизначеності виконання.

У підсумку сформовано підхід, який поєднує аналіз предметної області, проєктування та технічні вимоги до впровадження, створюючи підстави для переходу до програмної реалізації компонентів системи й інтеграції прикладної логіки опрацювання звернень.

РОЗДІЛ 3. ТЕХНІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ ВЕБ-СИСТЕМИ ОНЛАЙН-ЗАЯВОК НА АКАДЕМІЧНІ ДОВІДКИ

У попередньому розділі було доведено до завершеного вигляду модель предметної області, визначено схему авторизаційних повноважень для встановлених ролей, описано послідовності опрацювання звернень протягом їхнього життєвого циклу, а також подано UML, діаграми, що розкривають конфігурацію рішення, його сутності та характерні шаблони взаємодії.

Від цих засад тепер робиться практичний крок, у третьому розділі пояснюється вибір технологічного стеку, фіксуються вимоги до середовища роботи застосунку, описуються архітектурні рішення реалізації бізнес, логіки через RESTful API та визначені інтерфейси взаємодії. Окремо проєктується структура сховища даних на MongoDB і подаються послідовні інструкції з розгортання системи та перевірки її працездатності. Виклад вибудувано у прямій логіці: «вибір інструментів → вимоги до середовища → архітектурна реалізація → тестування та фіналізація». Кожна наступна частина показує, як сформульовані вище концепції стають працюючим програмним рішенням, яке дає прозорий стан системи, підтримує простежуваність операцій (traceability) і забезпечує рольову модель контролю доступу (RBAC).

3.1. Засоби розробки

З архітектурного погляду це клієнт,серверна веб,платформа. Усі незмінні елементи процесу, рольове обмеження доступу (RBAC), перевірка даних, контроль станів, генерація подій і сповіщень, зосереджено на серверній стороні. Клієнтський застосунок обмежується інтерфейсом введення і відображенням актуального стану. Такий розподіл обов’язків випливає з потреби жорстко керувати життєвим циклом заявок: клієнт не повинен бути джерелом істини щодо зміни статусу, інакше з’являється технічна можливість обходити встановлені процедурні норми. Отже, сервер виступає єдиним центром, де відбувається перевірка прав, контроль дозволених переходів і ведення повного аудиту.

На сервері обрано Node.js разом із Express. Це дозволяє зібрати HTTP, інтерфейс як модульне API, де обробка запитів вибудовується ланцюжком

middleware, компонентів. Така схема зручна, коли потрібно крок за кроком накладати і вимоги безпеки, і доменні правила на кожен вхідний запит. Загальні параметри HTTP, комунікації, логування, захисні заголовки, CORS, розбір JSON, єдиний формат помилок, винесено у стартовий блок ініціалізації. Це забезпечує централізований контроль вхідного трафіку й дає одне місце для фіксації критичних регламентів.

Code preview JavaScript

```
import express from 'express';
import helmet from 'helmet';
import cors from 'cors';
import cookieParser from "cookie-parser";
import { env } from './config/env.js';
import { httpLogger } from './logger/http.js';
import { router } from './routes/index.js';
import { notFound } from './middlewares/notFound.js';
import { errorHandler } from './middlewares/errorHandler.js';
export function createApp() {
    const app = express();
    app.set('trust proxy', true);
    app.use(cookieParser())
    app.use(httpLogger);
    app.use(helmet());
    app.use(cors({ origin: env.CORS_ORIGINS.length ? env.CORS_ORIGINS : true,
credentials: true }));
    app.use(express.json({ limit: '1mb' }));
    app.use(express.urlencoded({ extended: false }));
    // ...
    app.use('/api', router);
    app.use(notFound);
    app.use(errorHandler);
    return app;
}

```

Лістинг 3.1. Ініціалізація застосунку Express і базовий конвеєр проміжного ПЗ для маршруту /api (Node.js, app.js)

Інженерний сенс такого рішення в тому, що кожне звернення до доменного API проходить один і той самий вхідний конвеєр. У ньому по черзі спрацьовують безпекові механізми, парсинг, маршрутизація та уніфікована помилкова відповідь. Для системи, де відмова в доступі або помилка процесної перевірки мають повертатися стабільно й передбачувано, це принципово, клієнтський інтерфейс може коректно реагувати лише на стандартизований формат.

На клієнті використано Next.js, що доречно для застосунку з кількома рольовими контурами (ініціатор, деканат, адміністратор) і потребою в узгодженому транспортному шарі. Ключовим рішенням стало використання серверних маршрутів Next.js як проксі,рівня: вони приймають запит від UI та передають авторизоване звернення до бекенду. Так транспортна логіка не розповзається по інтерфейсу, а правила передавання заголовків і контент,типів стають однаковими. Приклад проксування PATCH, операції оновлення статусу виглядає так:

Code preview JavaScript

```
import { type NextRequest } from "next/server";
import { proxyAuthorizedRequest } from "@/lib/server/backend-proxy";
export async function PATCH(request: NextRequest, { params }: { params: { id: string } })
{
  const body = await request.text();
  return proxyAuthorizedRequest(`/api/admission/requests/${params.id}/status`, {
    method: "PATCH",
    body,
    contentType: request.headers.get("content-type") ?? "application/json",
  });
}
```

Лістинг 3.2. Проксі-роут PATCH для оновлення статусу заявки деканатом через proxyAuthorizedRequest (TypeScript, app/api/admission/requests/[id]/status/route.ts)

Фактично UI працює з внутрішнім API Next.js, а все, що стосується доступу до бекенду, включно з пересиланням токена, тримається в proxyAuthorizedRequest. Це компроміс, з’являється додатковий рівень, зате інтеграція стає керованішою і меншає шансів, що різні частини застосунку почнуть авторизуватися по різному.

Сховище даних побудовано на MongoDB із Mongoose. Вибір документної бази логічно випливає з предметної області. По перше, заявка містить вкладену хронологію подій (timeline), яку зручно тримати поруч із поточним станом. По друге, корисне навантаження (payload) частково змінне, бо залежить від набору полів конкретного типу довідки. Тому заявка природно зберігається як агрегат в одному документі, а не розсипається по великій кількості таблиць із жорсткими схемами.

Апаратні вимоги залишаються типовими для веб сервісів подібного класу: сервер або віртуальна інфраструктура для Node.js і MongoDB, клієнтські пристрої з сучасними браузерами. Мережева основа, університетська LAN або Інтернет,

забезпечує транспорт для клієнт, серверної взаємодії, де критичні параметри це доступність сервісу, коректна маршрутизація і захист каналу під час роботи з персональними даними.

Інженерне резюме вибору стеку зводиться до того, що він зменшує розрив між моделлю бізнес, процесу та її реалізацією. Express дає прозорий middleware, конвеєр для RBAC і процесних інваріантів, Next.js закриває рольові інтерфейси й контрольований транспорт, MongoDB/Mongoose органічно підтримують «заявку як агрегат» з історією та варіативним payload. З іншого боку, дисципліна підтримання цілісності на рівні сервісів і схем стає обов’язковою, документна модель за замовчуванням не гарантує коректності предметної області.

3.2. Вимоги до технічного та програмного забезпечення

Умови роботи програми доцільно фіксувати через принципи побудови відкритих систем, розглядаючи рішення як прикладний сервіс. Він працює поверх стандартних мережевих протоколів і виконує бізнес, функції через HTTP, інтерфейси. Якщо накласти це на модель OSI, вимоги до інфраструктури визначаються рівнями, що безпосередньо стосуються веб додатків: наявність фізичних вузлів (сервер і пристрої користувачів), забезпечення зв’язку на канальному та мережевому рівнях (мережа ЗВО або Інтернет), TCP як надійний транспорт, і HTTP(S) як спосіб взаємодії клієнта з API. На рівні представлення даних важливо правильно перетворювати об’єкти предметної моделі, зазвичай у JSON, і забезпечити підтримку національних символів через UTF-8, адже система працює з українськими назвами та описами.

ПЗ, потрібне для роботи, поділяється на три блоки:

- системне: ОС сервера, мережевий стек, TLS (для продакшн, середовища), механізми логування та синхронізації часу, на клієнті ОС і веб-браузер
- інструментальне: Node.js як runtime, npm, IDE, Git, засоби збирання та розгортання, також інструменти документації і UML
- прикладне: клієнт на Next.js, серверна логіка на Express API, база MongoDB, плюс модулі для сервісних функцій (сповіщення, аналітика, обробка помилок).

Мережеву взаємодію побудовано у класичній клієнт, серверній моделі: браузер формує HTTP, запити, сервер виконує бізнес, операції й звертається до MongoDB. У цій архітектурі мережа не є другорядною деталлю, від неї напряму залежать стійкість і безпека, оскільки контроль доступу тримається не лише на внутрішній логіці, а й на механізмах передавання даних, токенах, заголовках, CORS, межах довіри між компонентами.

Інженерний аргумент на користь OSI, підходу простий: він дозволяє відділити логіку програми від інфраструктурних залежностей. Мінус у тому, що OSI не описує бізнес, обмежень, вона лише дає рамку для правильного опису мережевих зв'язків, тоді як цілісність у життєвому циклі забезпечується на рівні L7 та структури даних.

3.3. Опис програмної реалізації

3.3.1. Потік одного запиту як єдина траєкторія даних (клієнт → API → БД)

Архітектурні рішення найнаочніше видно на прикладі проходження типового запиту з домену «життєвий цикл заявки». Показовий сценарій, оновлення статусу працівником деканату, бо тут одночасно з'являються RBAC, процесна валідація, зміна стану агрегату в БД, запис події і формування сповіщення. Представник деканату з інтерфейсу запускає зміну статусу, далі запит іде у проксі, маршрут Next.js, де формується авторизоване звернення до серверної частини. На бекенді PATCH, операція проходить послідовність middleware, починаючи з автентифікації. Автентифікація тут це не лише перевірка заголовка. Вона формує контекст виконання: користувач верифікується, дістається з бази і прив'язується до req.user. Завдяки цьому всі подальші кроки, перевірка ролей, фільтрація за факультетом, контроль допустимих дій, опираються на одне джерело достовірного контексту.

Code preview JavaScript

```
import jwt from "jsonwebtoken";
import { env } from "../config/env.js";
import { ApiError } from "../utils/ApiError.js";
import { User } from "../modules/users/user.model.js";
export function authenticate() {
  return async (req, res, next) => {
    const header = req.headers.authorization || "";
    const [type, token] = header.split(" ");
    if (type !== "Bearer" || !token) {
      return next(new ApiError(401, "Відсутній токен авторизації"));
    }
    try {
      const payload = jwt.verify(token, env.JWT_SECRET);
      const user = await User.findById(payload.sub).lean();
      if (!user) return next(new ApiError(401, "Недійсний токен"));
      if (user.isActive === false) return next(new ApiError(403, "Обліковий запис
вимкнено"));
      req.user = user;
      req.auth = payload;
      next();
    } catch (e) {
      return next(new ApiError(401, "Недійсний або протермінований токен"));
    }
  };
}

```

Лістинг 3.3. Middleware автентифікації за JWT Bearer і завантаження користувача в req.user (JavaScript, auth.js)

Далі на admission, маршруті накладається requireRoles ("Dean", "Admin"). Важливо, що RBAC стоїть на межі, ще до запуску бізнес, операції, користувач без прав не доходить до контролера чи сервісу, інваріант доступу зберігається до виконання доменної логіки.

Це простежується у визначенні відповідного маршруту:

Code preview JavaScript

```
export const admissionRouter = Router();
admissionRouter.use(authenticate());
// ...
admissionRouter.patch(
  "/requests/:id/status",
  requireRoles("Dean", "Admin"),
  validate(updateRequestStatusSchema),
  ctrl.updateRequestStatus,
);
```

Лістинг 3.4. Маршрут PATCH зміни статусу заявки з обмеженням ролей Dean/Admin і валідацією updateRequestStatusSchema (JavaScript, admission.routes.js)

Окремий необхідний крок тут, перевірка DTO через validate (updateRequestStatusSchema). Вона потрібна, щоб помилковий або неповний запит не дійшов до модифікації постійного стану. Коли всі middleware, перевірки пройдено, запускається сервісна функція, що змінює статус як одну бізнес, операцію. Вона знаходить заявку у межах відповідного факультету, застосовує доменні правила, наприклад обов’язковий коментар при відхиленні, проставляє часові позначки, додає подію в timeline, зберігає зміни та ініціює сповіщення заявника.

updateRequestStatus

```
export async function updateRequestStatus(user, id, dto, queryFacultyId) {
  const facultyId = getScopedFacultyId(user, queryFacultyId);
  const request = await CertificateRequest.findOne({ _id: id, faculty: facultyId });
  if (!request) throw httpError(404, "Заявку не знайдено");
  const now = new Date();
  request.status = dto.status;
  request.lastUpdatedBy = user._id;
  if (dto.status === CERT_STATUSES.IN_REVIEW && !request.reviewedAt) request.reviewedAt = now;
  if (dto.status === CERT_STATUSES.FORMING && !request.formingAt) request.formingAt = now;
  if (dto.status === CERT_STATUSES.READY) request.readyAt = now;
  if (dto.pickupFrom) request.pickupFrom = normalizeDateInput(dto.pickupFrom, "Дата видачі");
  if (dto.decisionComment !== undefined) {
    request.decisionComment = String(dto.decisionComment).trim() || undefined;
  }
  if (dto.status === CERT_STATUSES.REJECTED) {
    if (!dto.decisionComment || String(dto.decisionComment).trim().length < 3) {
      throw httpError(400, "Для відхилення заявки потрібен коментар");
    }
  }
  request.timeline = request.timeline || [];
  request.timeline.push({
    status: dto.status,
    by: user._id,
    comment: statusTimelineComment(dto.status, dto),
  });
  await request.save();
  const data = await CertificateRequest.findById(request._id)
    .populate("type", "title audience")
    .populate("requester", "email fullName role")
    .lean();
  await notifyRequesterAboutStatus({
    request: {
      ...request.toJSON(),
      requesterRole: data?.requesterRole || request.requesterRole,
    },
    status: request.status,
    decisionComment: request.decisionComment,
    pickupFrom: request.pickupFrom,
  });
  return data;
}
```

Лістинг 3.5. Сервісна функція updateRequestStatus: часові мітки етапів, запис у timeline, валідація коментаря при REJECTED, збереження та сповіщення заявника (JavaScript, admission.service.js)

У зіставленні з моделлю розділу 2 цей фрагмент відповідає Activity, діаграмі: зміна стану робиться як єдина операція, що одразу породжує процесну подію. Водночас видно і компроміс: переходи між станами не замкнено в жорстку матрицю дозволених переходів. Натомість коректність підтримується контекстом ролі та серверними інваріантами, обов'язкові поля, часові мітки, аудит. Це спрощує взаємодію і зменшує надлишкові обмеження, але частково переносить дисципліну процесу на UI та регламенти експлуатації, при тому критичні умови цілісності все одно гарантуються сервером.

3.3.2. REST API як набір бізнес-операцій

API тут мислиться не як набір відокремлених endpoint-ів, а як набір бізнес, операцій над доменними агрегатами. Кожна операція прив'язана до етапу життєвого циклу заявки і до ролей, зафіксованих у UML Use Case. Ініціаторські дії (студент або

викладач) створюють CertificateRequest і запускають процес зі статусом SUBMITTED. Деканатні операції (декан або адміністратор) відповідають за ключові рішення процесу, зміну статусу, реєстрацію подій, формування сповіщень. Адміністративні дії керують метаданими, факультетами та каталогом типів довідок, від яких залежить коректність валідації та доступність сценаріїв.

Критичний принцип тут такий: жодна бізнес, операція не має ламати цілісність агрегату «заявка». Створення чи зміна статусу завжди приводять до узгодженого стану, коректний status, повна історія, релевантні timestamps. Через це аналітику можна будувати без окремих обхідних джерел, історія змін уже інтегрована в модель.

Узагальнення підходу таке: декомпозиція API повторює Use Case, модель, а не існує окремо від неї. При цьому деякі аспекти узгодженості, наприклад допустимість переходів, частково задаються не лише кодом, а й правилами користування системою. Тому модуль тестування має включати сценарії процесних помилок і перевірки інваріантів.

3.3.3. MongoDB: гнучкість і цілісність; роль історії статусів у аналітиці

CertificateRequest реалізовано як документ, що поєднує «поточний стан» та «історію», тобто підхід «агрегат + події». Модель одночасно тримає контекст (факультет, ініціатор, режим подання), зміст (payload), керований стан (status і timestamps) та журнал (timeline):

Code preview JavaScript

```javascript
const certificateRequestSchema = new Schema(
  {
    faculty: { type: Schema.Types.ObjectId, ref: "Faculty", required: true, index: true },

    type: { type: Schema.Types.ObjectId, ref: "CertificateType", index: true },
    requestMode: {
      type: String,
      enum: REQUEST_MODE_ENUM,
      default: CERT_REQUEST_MODES.TEMPLATE,
      index: true,
    },
    requestTitle: { type: String, required: true, trim: true, maxlength: 240 },
    requestPurpose: { type: String, trim: true, maxlength: 2000 },
    requester: { type: Schema.Types.ObjectId, ref: "User", required: true, index: true },
    requesterEmail: { type: String, lowercase: true, trim: true, index: true },
    requesterFullName: { type: String, trim: true, index: true },
    requesterRole: { type: String, index: true },
    payload: { type: Schema.Types.Mixed, default: {} },
    status: { type: String, enum: STATUS_ENUM, default: CERT_STATUSES.SUBMITTED, index:
true },
    requestNo: { type: String, index: true },
    regNumber: { type: String, index: true },
    submittedAt: { type: Date, default: () => new Date(), index: true },
    reviewedAt: { type: Date },
    formingAt: { type: Date },
    readyAt: { type: Date },
    pickupFrom: { type: Date },
    decisionComment: { type: String, trim: true },
    timeline: {
      type: [
        {
          status: { type: String, enum: STATUS_ENUM, required: true },
          at: { type: Date, default: () => new Date() },
          by: { type: Schema.Types.ObjectId, ref: "User" },
          comment: { type: String, trim: true },
        },
      ],
      default: [],
    },
    lastUpdatedBy: { type: Schema.Types.ObjectId, ref: "User" },
  },
  { timestamps: true, versionKey: false }
);

```

Лістинг 3.6. Mongoose-схема CertificateRequest: зв’язки, режим подання, статуси, payload типу Mixed, часові мітки та подієвий журнал timeline (JavaScript, certificateRequest.model.js)

З інженерного боку тут витримано баланс. Гнучкість забезпечує payload: Mixed, він дозволяє зберігати різні набори атрибутів залежно від CertificateType.fields[] або вибраного PURPOSE. Цілісність тримається не «жорсткістю» payload, схеми, а сукупністю механізмів: (1) enum, переліки для status та requestMode, (2) перевірка структури запиту на вході через middleware і доменні перевірки, (3) процесні інваріанти в сервісному шарі. Виходить, що гарантія коректності переноситься зі схеми БД на логіку обробки та бізнес, правила.

Історія статусів працює на аналітичний контур двома способами. Вона дає простежуваність життєвого циклу й дозволяє оцінювати якість опрацювання. Плюс часові мітки ключових етапів (submittedAt, readyAt, updatedAt) дають змогу будувати агреговані метрики оперативності по завершених процесах. Тобто потреба в окремих «таблицях звітності» відпадає, потрібні дані вже лежать у доменній моделі.

Узагальнення тут таке: заявка розглядається як один агрегат, це спрощує оновлення стану і реєстрацію подій. Але відповідальність сервісного шару за перевірку payload та за коректність послідовності статусів зростає, успішний запис у БД сам по собі не означає відповідність предметній області.

3.3.4. RBAC «в глибину»: доступ як частина життєвого циклу

RBAC реалізовано на двох рівнях. Спочатку автентифікація формує req.user, потім авторизація на маршруті блокує або пропускає до бізнес, операції. Така двоступенева схема гарантує, що сервіс отримує вже перевірений контекст і може зосередитися на процесних інваріантах.

Мінімалістичний requireRoles робить основне, перетворює «роль як дані» на «роль як умову виконання»:

Code preview JavaScript

```javascript
export function requireRoles(...roles) {
  return (req, _res, next) => {
    const role = req.user?.role;
    if (!role || !roles.includes(role)) {
      const err = new Error("Недостатньо прав");
      err.status = 403;
      return next(err);
    }
    next();
  };
}

```

Лістинг 3.7. Middleware requireRoles: перевірка req.user.role проти дозволених ролей і повернення помилки 403 (JavaScript, authorize.js)

Важливо, що RBAC охоплює всі фази життєвого циклу. Ініціатор може створювати і переглядати, але не змінює статус. Деканат виконує переходи, але лише в межах власного факультету. Адміністратор керує метаданими, що визначають структуру запиту та правила валідації. Це зменшує ризик процедурних збоїв, коли стан заявки змінюється суб'єктом без належних повноважень.

Обґрунтування підходу таке: контрольні точки на маршрутах роблять рольову модель прозорою і підзвітною, простіше бачити дозволені дії та їх виконавців. Водночас виникає потреба узгоджувати ролі з факультетським обмеженням уже на рівні сервісів, оскільки роль без організаційного контексту недостатня. Це закривається через механізм скоупінгу в доменних сервісах.

3.3.5. Сповіщення як наслідок подій, а не окремий «острівець»

Сповіщення тут не живуть окремо від процесу. Коли змінюється статус, викликається notifyRequesterAboutStatus. Вона збирає повідомлення з урахуванням значення нового статусу та формує deep link, який залежить від ролі отримувача.

У підсумку виходить цілісна зв'язка: подія зміни стану, повідомлення, перехід у релевантний контекст.

Code preview — JavaScript

```
function requestLinkForRecipient(role, requestId) {
  if (!requestId) return undefined;
  return role === "Dean" || role === "Admin"
    ? `/dean/orders?request=${encodeURIComponent(requestId)}`
    : `/certificates/ordered?request=${encodeURIComponent(requestId)}`;
}
function statusNotificationCopy(
  requestTitle,
  status,
  decisionComment,
  pickupFrom,
) {
  if (status === CERT_STATUSES.IN_REVIEW) {
    return {
      kind: "request.in_review",
      title: "Заявку взято в роботу",
      message: `Деканат почав опрацьовувати вашу заявку "${requestTitle}".`,
    };
  }
  // ...
  if (status === CERT_STATUSES.REJECTED) {
    return {
      kind: "request.rejected",
      title: "Заявку відхилено",
      message: decisionComment?.trim()
        ? `Заявку "${requestTitle}" відхилено. Коментар деканату:
${decisionComment.trim()}`
        : `Заявку "${requestTitle}" відхилено деканатом.`,
    };
  }
  // ...
}
```

Лістинг 3.8. Формування deep-link для сповіщень залежно від ролі та шаблонів заголовка/тексту повідомлень за статусом заявки (JavaScript, notification.service.js)

Цей фрагмент демонструє підхід, де зміст повідомлень є прямим наслідком стану системи. Тексти формуються детерміновано, це зменшує різночитання і задає однакове трактування статусів для всіх ролей. Централізувати тексти і посилання в окремому модулі зручно для підтримки процесної комунікації, мінус очевидний, потрібно слідкувати, щоб він не розійшовся зі статусною логікою. Проте це помітно безпечніше за дублювання однакових повідомлень по різних частинах інтерфейсу.

3.4. Керівництво користувача (сценарії, інтерфейс, тестування та виключні ситуації)

3.4.1. Призначення системи та базові сценарії

Система задумана як єдина точка подання і подальшого опрацювання заявок на академічні довідки. Вона містить керування переліком доступних видів довідок, підтримує варіант «за потребою», дає відстеження статусу з повною історією, має модуль сповіщень і аналітичні інструменти для деканатів. Якщо описувати через UML Use Case, базові взаємодії такі: створення запиту користувачем, супровід обробки деканатом, керування довідниками й факультетськими даними, контроль метрик ефективності.

3.4.2. Сценарій користувача-ініціатора (студент/викладач)

Після входу користувач переглядає список шаблонів, вибирає потрібний і заповнює форму згідно з визначеною структурою полів. За потреби можна активувати режим «за цільовим призначенням», додавши опис запиту й ключові дані. Після надсилання форми користувач або отримує повідомлення про помилки валідації, тоді заявка не створюється, або підтвердження успішного створення зі статусом SUBMITTED. Далі заявка відображається в особистому списку разом із зафіксованою хронологією. Подальша обробка йде через зміни статусів і сповіщення з прямими посиланнями на деталі заявки.

3.4.3. Сценарій адміністратора

Адміністратор веде довідники факультетів і класифікатор типів довідок. З технічного боку це керування метаданими, від яких залежить набір шаблонів, перелік обов’язкових полів і правила валідації на сервері. Функція адміністратора критична для стабільності, помилка у структурі полів легко призводить до проблем із

формуванням заявок, а некоректна прив’язка до організаційної структури ламає проходження заявок у процесі.

3.4.4. Тестування: методи і сценарії відповідності вимогам

Перевірки доцільно трактувати як підтвердження інваріантів, сформульованих у розділі 2:

- функціональні сценарії: подання TEMPLATE, подання PURPOSE, зміна станів деканатом, відхилення з поясненням, повторне подання після виправлень
- RBAC, перевірки: спроба змінити статус без ролі Декан або Адміністратор має завершуватися відмовою
- контроль процесних інваріантів: REJECTED не встановлюється без decisionComment (див. updateRequestStatus), а зміни даних під час активного опрацювання мають блокуватися на рівні сервісів
- traceability: кожен перехід фіксується в timeline, а сповіщення виникають лише після відповідної події.

3.4.5. Обробка виключних ситуацій

Виключні ситуації групуються так: проблеми з підтвердженням особи, відсутній або протермінований токен, порушення доступу, брак повноважень, збої перевірки структури даних, процесні помилки, наприклад відхилення без пояснення або неможливість операції через поточний стан, і недоступність інфраструктури, сервер чи база не відповідають.

У кожному випадку потрібна клієнт орієнтована реакція: система має давати чітке пояснення причини відмови, а інтерфейс повинен показувати цю інформацію прямо, без маскування суті.

Конструктивне обґрунтування такого підходу в тому, що методологія тестування, орієнтована на інваріанти, прямо випливає з UML-моделювання. Акцент зсувається з перевірки наявності елементів UI на підтвердження контрактних умов виконання процесів. Оскільки значна частина надійності визначається серверними інваріантами та їх тестовим покриттям, основний фокус регресійного тестування має бути на бекенд-контурі.

ВИСНОВКИ ДО РОЗДІЛУ 3

У цьому розділі зведено основні проблеми предметної області, які виникають через домінування паперового або неструктурованого документообігу під час оформлення академічних довідок. З’ясовано, що за відсутності спільного сховища даних порушується цілісність звернення, з’являється зайве дублювання заявок і повторення операцій, а передача завдань між відповідальними особами стає важчою для контролю. Так само встановлено, що слабкий нагляд за станом виконання у традиційних підходах не дає змоги прозоро відстежувати прогрес, через що виникають затримки, які складно передбачити. Окремо показано, що нерівний доступ до інформації між заявником і виконавцями збільшує комунікаційне навантаження та провокує повторні уточнення там, де формально в цьому немає потреби.

Звідси випливає висновок: для усунення цих недоліків недостатньо точкової цифровізації, потрібна повна автоматизація процесу як єдиної керованої траєкторії. Також наголошено, що запропонована програмна система прибирає виявлені проблеми, закріплюючи заявку як єдиний об’єкт предметної області, із централізованим зберіганням, механізмом відстеження статусу та повною історією змін. Показано, що рольова модель доступу (RBAC) дає змогу дотримуватися процедурних обмежень, розводячи за ролями ініціювання звернення, ухвалення рішень щодо статусів і адміністрування довідників. Встановлено, що фіксація змін статусу та подій процесу прибирає неформалізований підхід до виконання операцій і задає спільне трактування стану заявки для всіх учасників. Зазначено й те, що сповіщення та аналітичні дані будуються на тих самих процесних даних, отже потреба підтримувати паралельні джерела інформації зникає.

Окремий акцент зроблено на концептуальному моделюванні як на методологічному засобі узгодити вимоги з подальшою реалізацією системи. Продемонстровано, що UML-діаграми та інформаційна модель формалізують межі системи, будову її елементів і логіку процесів. Встановлено, що визначена структура ролей і опис життєвого циклу заявки задають інваріантну основу для реалізації API, моделі даних і механізмів контролю доступу. Зроблено висновок, що такий підхід

знижує ризик розбіжностей між проєктною моделлю та програмною реалізацією і підтримує повторюваність архітектурних рішень.

У підсумку розділ створює узгоджену базу для практичного впровадження системи у форматі компонентної реалізації. Отримані результати визначають методологічний фундамент для наступного етапу, який охоплює розроблення компонентів системи, інтеграцію бізнес-логіки з контролем доступу та перевірку коректності роботи в умовах експлуатації.

ВИСНОВКИ

У цьому розділі зведено основні проблеми предметної області, які виникають через домінування паперового або неструктурованого документообігу під час оформлення академічних довідок. З’ясовано, що за відсутності спільного сховища даних порушується цілісність звернення, з’являється зайве дублювання заявок і повторення операцій, а передача завдань між відповідальними особами стає важчою для контролю. Так само встановлено, що слабкий нагляд за станом виконання у традиційних підходах не дає змоги прозоро відстежувати прогрес, через що виникають затримки, які складно передбачити. Окремо показано, що нерівний доступ до інформації між заявником і виконавцями збільшує комунікаційне навантаження та провокує повторні уточнення там, де формально в цьому немає потреби. Звідси випливає висновок: для усунення цих недоліків недостатньо точкової цифровізації, потрібна повна автоматизація процесу як єдиної керованої траєкторії.

Також наголошено, що запропонована програмна система прибирає виявлені проблеми, закріплюючи заявку як єдиний об’єкт предметної області, із централізованим зберіганням, механізмом відстеження статусу та повною історією змін. Показано, що рольова модель доступу (RBAC) дає змогу дотримуватися процедурних обмежень, розводячи за ролями ініціювання звернення, ухвалення рішень щодо статусів і адміністрування довідників. Встановлено, що фіксація змін статусу та подій процесу прибирає неформалізований підхід до виконання операцій і задає спільне трактування стану заявки для всіх учасників. Зазначено й те, що сповіщення та аналітичні дані будуються на тих самих процесних даних, отже потреба підтримувати паралельні джерела інформації зникає.

Окремий акцент зроблено на концептуальному моделюванні як на методологічному засобі узгодити вимоги з подальшою реалізацією системи. Продемонстровано, що UML-діаграми та інформаційна модель формалізують межі системи, будову її елементів і логіку процесів. Встановлено, що визначена структура ролей і опис життєвого циклу заявки задають інваріантну основу для реалізації API, моделі даних і механізмів контролю доступу. Зроблено висновок, що такий підхід знижує ризик розбіжностей між проєктною моделлю та програмною реалізацією і

підтримує повторюваність архітектурних рішень. У підсумку розділ створює узгоджену базу для практичного впровадження системи у форматі компонентної реалізації. Отримані результати визначають методологічний фундамент для наступного етапу, який охоплює розроблення компонентів системи, інтеграцію бізнес-логіки з контролем доступу та перевірку коректності роботи в умовах експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні вимоги та правила складання. Чинний від 2017-01-01. Київ : ДП «УкрНДНЦ», 2016. 46 с.

2. ISO/IEC 7498-1:1994. Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model [Електронний ресурс]. URL: https://www.iso.org/standard/20269.html (дата звернення: 23.09.2025).

3. ITU-T Recommendation X.200 : Open Systems Interconnection — Basic Reference Model [Електронний ресурс]. URL: https://www.itu.int/rec/T-REC-X.200/ (дата звернення: 26.09.2025).

4. Hypertext Transfer Protocol (HTTP/1.1). RFC 9110 [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc9110 (дата звернення: 01.10.2025).

5. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc3986 (дата звернення: 06.10.2025).

6. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc8446 (дата звернення: 11.10.2025).

7. JSON Web Token (JWT). RFC 7519 [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc7519 (дата звернення: 17.10.2025).

8. JSON Web Signature (JWS). RFC 7515 [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc7515 (дата звернення: 22.10.2025).

9. OWASP Cheat Sheet Series : REST Security [Електронний ресурс]. URL: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html (дата звернення: 27.10.2025).

10. OWASP Cheat Sheet Series : JSON Web Token for Java [Електронний ресурс]. URL: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html (дата звернення: 02.11.2025).

11. MDN Web Docs : HTTP [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення: 08.11.2025).

12. MDN Web Docs : HTTP cookies [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies (дата звернення: 14.11.2025).

13. MDN Web Docs : Cross-Origin Resource Sharing (CORS) [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (дата звернення: 20.11.2025).

14. Fetch Living Standard [Електронний ресурс]. URL: https://fetch.spec.whatwg.org/ (дата звернення: 26.11.2025).

15. Fetch Living Standard : Cross-Origin Resource Sharing (CORS) [Електронний ресурс]. URL: https://fetch.spec.whatwg.org/#http-cors-protocol (дата звернення: 02.12.2025).

16. Node.js Documentation [Електронний ресурс]. URL: https://nodejs.org/docs/ (дата звернення: 09.12.2025).

17. Node.js API Reference : crypto module [Електронний ресурс]. URL: https://nodejs.org/api/crypto.html (дата звернення: 15.12.2025).

18. npm : jsonwebtoken package [Електронний ресурс]. URL: https://www.npmjs.com/package/jsonwebtoken (дата звернення: 21.12.2025).

19. npm : bcryptjs package [Електронний ресурс]. URL: https://www.npmjs.com/package/bcryptjs (дата звернення: 27.12.2025).

20. npm : cookie-parser package [Електронний ресурс]. URL: https://www.npmjs.com/package/cookie-parser (дата звернення: 03.01.2026).

21. npm : cors package [Електронний ресурс]. URL: https://www.npmjs.com/package/cors (дата звернення: 10.01.2026).

22. Express – Node.js web application framework. Documentation [Електронний ресурс]. URL: https://expressjs.com/ (дата звернення: 17.01.2026).

23. Express : Routing [Електронний ресурс]. URL: https://expressjs.com/en/guide/routing.html (дата звернення: 24.01.2026).

24. Express : Writing middleware [Електронний ресурс]. URL: https://expressjs.com/en/guide/using-middleware.html (дата звернення: 31.01.2026).

25. Express : Error handling [Електронний ресурс]. URL: https://expressjs.com/en/guide/error-handling.html (дата звернення: 07.02.2026).

26. Helmet [Електронний ресурс]. URL: https://helmetjs.github.io/ (дата звернення: 14.02.2026).

27. Next.js Documentation [Електронний ресурс]. URL: https://nextjs.org/docs (дата звернення: 21.02.2026).

28. Next.js : Route Handlers (App Router) [Електронний ресурс]. URL: https://nextjs.org/docs/app/building-your-application/routing/route-handlers (дата звернення: 28.02.2026).

29. Next.js : Middleware [Електронний ресурс]. URL: https://nextjs.org/docs/app/building-your-application/routing/middleware (дата звернення: 07.03.2026).

30. Next.js : Environment Variables [Електронний ресурс]. URL: https://nextjs.org/docs/app/building-your-application/configuring/environment-variables (дата звернення: 14.03.2026).

31. React Documentation [Електронний ресурс]. URL: https://react.dev/ (дата звернення: 21.03.2026).

32. MongoDB Documentation [Електронний ресурс]. URL: https://www.mongodb.com/docs/ (дата звернення: 28.03.2026).

33. MongoDB Manual : Data Modeling Introduction [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/data-modeling/ (дата звернення: 04.04.2026).

34. MongoDB Manual : Indexes [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/indexes/ (дата звернення: 11.04.2026).

35. MongoDB Manual : Aggregation Operations [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/aggregation/ (дата звернення: 18.04.2026).

36. Mongoose Documentation [Електронний ресурс]. URL: https://mongoosejs.com/docs/ (дата звернення: 25.04.2026).

37. Mongoose : Schemas [Електронний ресурс]. URL: https://mongoosejs.com/docs/guide.html (дата звернення: 02.05.2026).

38. Mongoose : Population [Електронний ресурс]. URL: https://mongoosejs.com/docs/populate.html (дата звернення: 06.05.2026).

39. Mongoose : Validation [Електронний ресурс]. URL: https://mongoosejs.com/docs/validation.html (дата звернення: 09.05.2026).

ДОДАТОК А

Посилання на GitHub репозиторій:

https://github.com/Vlad-Dzemiuk/uni_system

або див. Рис. 4.

Рис. 4. QR код посилання на GitHub репозиторій