НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету
_____ Ігор БОЛБОТ
(підпис)
«___» _____ 2026 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри інформаційних систем і технологій
_____ Швиденко М. З.
(підпис)
«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційна система обробки університетських петицій»
Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми _____ Роман РУДЕНСЬКИЙ
д.е.н., професор (підпис)
Керівник бакалаврської _____ Максим МОКРІЄВ
кваліфікаційної роботи (підпис)
к.е.н., доцент
Виконав _____ Максим ВАСИЛЬЧЕНКО
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних систем і технологій
к.е.н., доцент _____ Швиденко М. З.
(підпис)
«___» _____ 2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ
Васильченку Максиму Сергійовичу
(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»
Тема бакалаврської кваліфікаційної роботи
«Інформаційна система обробки університетських петицій»
затверджена наказом від «06» січня 2026 р. № 46 «С»
Термін подання завершеної роботи на кафедру «23» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи: відомості про процес подання, підписання, модерації та розгляду університетських петицій; ролі користувачів системи; статуси петицій; дані про категорії, користувачів, підписи, коментарі та офіційні відповіді.
Перелік питань, що підлягають дослідженню:
аналіз предметної області обробки університетських петицій;
проєктування інформаційної системи обробки університетських петицій;
розроблення вебзастосунку для обробки університетських петицій;
тестування та оцінювання роботи системи.
Перелік графічних документів (за необхідності): UML-діаграми, ER-діаграма, діаграми архітектури та розгортання системи.
Дата видачі завдання «06» січня 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ Максим МОКРІЄВ
(підпис)
Завдання взяв до виконання _____ Максим ВАСИЛЬЧЕНКО
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AJAX – Asynchronous JavaScript and XML – технологія асинхронних запитів до сервера без перезавантаження сторінки

API – Application Programming Interface – програмний інтерфейс застосунку

БД – база даних

CSRF – Cross-Site Request Forgery – міжсайтова підробка запитів

CSS – Cascading Style Sheets – каскадні таблиці стилів

CSV – Comma-Separated Values – формат файлу з даними, розділеними комами

ДСТУ – Державний стандарт України

ER – Entity-Relationship – модель «сутність – зв’язок»

FK – Foreign Key – зовнішній ключ

HTML – HyperText Markup Language – мова розмітки гіпертексту

HTTP – HyperText Transfer Protocol – протокол передачі гіпертексту

ІС – інформаційна система

JSON – JavaScript Object Notation – текстовий формат обміну даними

KPI – Key Performance Indicator – ключовий показник ефективності

MVC – Model – View – Controller – архітектурний шаблон

MVT – Model – View – Template – архітектурний шаблон Django

ORM – Object-Relational Mapping – об’єктно-реляційне відображення

PK – Primary Key – первинний ключ

ППЗ – прикладне програмне забезпечення

SQL – Structured Query Language – мова структурованих запитів

СУБД – система управління базами даних

UML – Unified Modeling Language – уніфікована мова моделювання

URL – Uniform Resource Locator – уніфікований покажчик ресурсу

ВСТУП

Актуальність теми. Розвиток демократичних інститутів у закладах вищої освіти передбачає наявність ефективних механізмів зворотного зв’язку між студентською спільнотою та адміністрацією. Конституція України (стаття 40) гарантує право на звернення, а Закон України «Про вищу освіту» закріплює участь студентського самоврядування у вирішенні питань освітнього процесу. Водночас переважна більшість українських ЗВО не має спеціалізованих цифрових інструментів для обробки колективних звернень: збір підписів здійснюється на папері, звернення надходять через розрізнені канали (пошта, месенджери, соціальні мережі), а відстеження результатів розгляду фактично відсутнє. Це призводить до фрагментації інформації, фальсифікації підписів, непрозорості процесу розгляду, втрати інституційної пам’яті та низької мотивації студентів до участі у самоврядуванні.

Аналіз існуючих рішень – від державного порталу електронних петицій до платформ Change.org, Ushahidi та JIRA Service Management – засвідчує, що жодне з них не забезпечує повного набору функцій для університетського контексту: верифікованих електронних підписів, премодерації, статусної моделі з автоматичними переходами, аналітичного дашборда та повного аудиту дій. Це обумовлює актуальність створення спеціалізованої інформаційної системи.

Мета роботи – розробка вебзастосунку для обробки університетських петицій, який реалізує повний життєвий цикл звернення з дев’ятьма статусами, забезпечує прозорість, підзвітність та аналітику процесу, а також адаптований до організаційної структури закладу вищої освіти.

Для досягнення мети визначено такі задачі:

1) дослідити предметну область обробки петицій у ЗВО, визначити ролі учасників, етапи процесу та часові обмеження;
2) проаналізувати існуючі цифрові рішення для обробки колективних звернень, виявити їхні переваги та недоліки;

3) сформулювати функціональні та нефункціональні вимоги до інформаційної системи;
4) спроєктувати архітектуру системи за шаблоном MVT, розробити логічну модель бази даних та змоделювати бізнес-процеси;
5) обґрунтувати вибір технологій розробки (Python, Django 5.0, SQLite, Chart.js);
6) реалізувати серверну та клієнтську частини вебзастосунку з модулями автентифікації, управління петиціями, підписання, модерації, аналітики та сповіщень;
7) провести тестування системи та оцінити коректність реалізації;
8) сформувати рекомендації щодо впровадження та експлуатації.

Методи та технології. У роботі використано методи системного аналізу та моделювання (UML-діаграми: варіантів використання, діяльності, переходів станів, компонентів, розгортання, ER-діаграма), об'єктно-орієнтоване проєктування та програмування, шаблон проєктування MVT. Технологічний стек: мова програмування Python 3.10+, вебфреймворк Django 5.0, СУБД SQLite 3, бібліотека Chart.js 4.x для візуалізації аналітики, кастомний CSS для стилізації інтерфейсу. Тестування виконано через вбудований фреймворк django.test.

Апробація результатів. Основні результати роботи доповідалися на засіданні кафедри інформаційних систем і технологій факультету інформаційних технологій НУБіП України.

Структура та обсяг роботи. Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі досліджено предметну область обробки університетських петицій: визначено ролі учасників (студент, модератор, адміністратор), описано життєвий цикл петиції з дев'ятьма станами, проаналізовано проблеми існуючого паперового підходу та виконано порівняльний аналіз шести цифрових рішень за сімома критеріями. Сформульовано постановку задачі.

У другому розділі сформовано 18 функціональних та 10 нефункціональних вимог до системи, побудовано UML-діаграми (варіантів використання, діяльності, переходів станів, компонентів, розгортання), спроєктовано архітектуру за шаблоном MVT з чотирма Django-застосунками та логічну модель бази даних з десятьма сутностями у третій нормальній формі.

У третьому розділі обґрунтовано вибір технологічного стеку, реалізовано серверну частину (10 моделей, 15 представлень, 6 форм, middleware та контекстні процесори) та клієнтську частину (13 шаблонів, кастомний CSS, AJAX-оновлення, графіки Chart.js). Детально описано реалізацію модулів створення, підписання, модерації та офіційної відповіді з лістингами ключових фрагментів коду.

У четвертому розділі описано організацію тестування (модульне та інтеграційне), перевірено коректність функціональних модулів, проаналізовано результати та сформовано рекомендації щодо впровадження системи у закладі вищої освіти.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ОБРОБКИ УНІВЕРСИТЕТСЬКИХ ПЕТИЦІЙ

1.1 Особливості процесу подання та розгляду університетських петицій

Петиція у контексті закладу вищої освіти (ЗВО) являє собою формалізоване колективне звернення, у якому представники університетської спільноти формулюють конкретну проблему, аргументують її актуальність та пропонують шляхи вирішення. На відміну від індивідуальних заяв чи побутових скарг, петиція передбачає публічний збір підписів на підтримку ініціативи, що забезпечує її репрезентативність та надає звернення вагомості в очах адміністрації [1]. Саме колективний характер петиції визначає її принципову відмінність від інших форм зворотного зв'язку: вона виражає не приватний інтерес окремої особи, а спільну позицію значної частини академічної спільноти.

Правову основу петиційного механізму в Україні складають кілька нормативних актів. Конституція України (стаття 40) гарантує кожному громадянину право на звернення до органів державної влади та місцевого самоврядування. Закон України «Про звернення громадян» від 02.10.1996 № 393/96-ВР деталізує процедуру подання та розгляду звернень, визначає їх види та встановлює строки розгляду [2]. Закон України «Про вищу освіту» від 01.07.2014 № 1556-VII закріплює право органів студентського самоврядування на участь у вирішенні питань, що стосуються освітнього процесу, побуту та дозвілля здобувачів вищої освіти [3]. Петиційний механізм на рівні університету є однією з практичних форм реалізації цих прав, хоча його конкретне процедурне оформлення зазвичай регулюється внутрішніми нормативними документами ЗВО.

На державному рівні механізм електронних петицій успішно функціонує через портал «Електронні петиції» Офісу Президента України. Згідно з чинним

законодавством, електронна петиція, що набрала визначену кількість підписів протягом встановленого терміну, підлягає обов'язковому розгляду. Досвід цієї платформи переконливо демонструє ефективність цифрових інструментів для збору та обробки колективних звернень. Водночас масштаб, порогові значення та цільова аудиторія державної системи суттєво відрізняються від університетського контексту, що обумовлює необхідність створення спеціалізованого рішення.

Окрім нормативно-правової бази, інститут петицій в університетському середовищі спирається на принципи демократичного врядування в освіті, закріплені у Стандартах і рекомендаціях щодо забезпечення якості в Європейському просторі вищої освіти (ESG 2015). Зокрема, стандарт ESG 1.1 наголошує на залученні студентів до процесів забезпечення якості, а стандарт ESG 1.3 передбачає наявність формальних механізмів для подання студентських пропозицій та скарг. Електронна петиційна система є одним із практичних інструментів реалізації цих стандартів.

Важливо розмежувати поняття «петиція», «скарга» та «пропозиція» у контексті університетського середовища. Скарга – це індивідуальне звернення щодо порушення прав конкретного студента, яке розглядається в персональному порядку. Пропозиція – це ідея щодо покращення, яка не потребує збору підписів та може бути подана через загальну форму зворотного зв'язку. Петиція ж є колективним зверненням, що вимагає публічної підтримки визначеної кількості членів спільноти для переходу на етап офіційного розгляду. Саме ця вимога щодо підтримки спільноти відрізняє петиційний процес та обумовлює необхідність спеціалізованого програмного забезпечення.

Формалізація процесу обробки петиції може бути описана через послідовність станів та переходів між ними. З точки зору теорії автоматів, статусна модель петиції являє собою кінцевий детермінований автомат, де множина станів S = {draft, moderation, rejected, active, review, approved, partial, declined, expired}, початковий стан s0 = draft, а функція переходів визначає допустимі трансформації між станами. Переходи ініціюються діями конкретних

ролей: автор ініціює перехід draft → moderation; модератор – moderation → active, moderation → rejected, moderation → draft (повернення); система автоматично здійснює перехід active → review (при досягненні порогу) та active → expired (при вичерпанні терміну); адміністратор – review → approved, review → partial, review → declined.

Процес обробки університетської петиції є комплексним бізнес-процесом, що охоплює кілька послідовних етапів та залучає учасників із чітко визначеними ролями та повноваженнями. Типовий життєвий цикл петиції включає такі фази:

- формулювання та подання ініціативи автором із зазначенням адресата, категорії, обґрунтування та очікуваного результату;
- премодерація – перевірка модератором на відповідність правилам платформи, коректність формулювань, відсутність дублювання з раніше поданими зверненнями;
- публікація та збір підписів протягом визначеного терміну (за замовчуванням – 30 календарних днів від дати публікації);
- автоматичний перехід петиції у статус «на розгляді адміністрацією» після досягнення порогового значення підтримки (за замовчуванням – 100 підписів);
- офіційний розгляд уповноваженою особою із зобов'язанням надати мотивовану відповідь протягом 14 календарних днів;
- публікація офіційної відповіді з одним із трьох можливих рішень: підтримано, підтримано частково або відхилено;
- архівування петиції із збереженням усієї історії дій для подальшої аналітики.

Кожен із зазначених етапів потребує чіткого розподілу відповідальності між учасниками процесу. Нижче наведено характеристику основних ролей, що формують комунікаційне середовище системи петицій.

Ініціатор (автор петиції) – студент, аспірант, викладач або інший представник університетської спільноти, який ідентифікує проблему, формулює текст звернення, визначає адресата (ректорат, деканат факультету, кафедра, студентське самоврядування) та подає петицію до системи. Автор визначає категорію звернення із переліку доступних (освітній процес, інфраструктура, гуртожитки, стипендії та фінанси, бібліотека, спорт і дозвілля, цифрові сервіси, безпека, харчування, самоврядування). Після подання автор має право відстежувати статус петиції, отримувати сповіщення про зміну статусу, переглядати коментарі та список підписантів.

Підписант – зареєстрований користувач системи, який висловлює підтримку петиції, поставивши свій електронний підпис. Кожен користувач може підписати конкретну петицію лише один раз, що забезпечується на рівні бази даних через обмеження унікальності (unique constraint) на пару «петиція – користувач». Кожен підпис верифікується через генерацію хеш-значення за алгоритмом SHA-256, що включає ідентифікатори користувача та петиції, а також часову мітку [4]. Підписант має можливість залишити коментар до свого підпису, а також обрати анонімний режим підписання, за якого його ім'я не відображатиметься у публічному списку, проте збережеться у базі даних для цілей аудиту. Система також дозволяє відкликати підпис до завершення терміну збору, що забезпечує свободу волевиявлення.

Модератор – уповноважена особа (зазвичай представник студентської ради або адміністрації), яка перевіряє кожну петицію перед її публікацією. Модератор оцінює коректність формулювань, відповідність тематиці платформи, відсутність ненормативної лексики та порушень етичних норм, а також здійснює перевірку на дублювання з уже опублікованими або розглянутими петиціями. За результатами перевірки модератор ухвалює одне з трьох рішень: опублікувати петицію (після чого розпочинається збір підписів), відхилити з обов'язковим зазначенням причини або повернути автору на доопрацювання з рекомендаціями щодо виправлення. Усі рішення модератора

фіксуються у журналі аудиту із зазначенням часу, імені модератора та тексту обґрунтування.

Адміністратор (представник керівництва ЗВО) – особа, уповноважена надавати офіційну мотивовану відповідь на петицію, яка зібрала пороговий обсяг підписів та перейшла у статус офіційного розгляду. Відповідь містить текст рішення, одне з можливих рішень (підтримано, підтримано частково, відхилено) та може включати прикріплений документ. Кожна відповідь є публічною, доступною для перегляду всіма користувачами системи, та супроводжується сповіщенням автору петиції та всім підписантам. Адміністратор також має доступ до аналітичного дашборда системи, де представлені ключові метрики: кількість петицій за період, розподіл за категоріями та факультетами, відсоток підтриманих звернень, середній час розгляду.

Важливою характеристикою процесу є наявність часових обмежень на кожному етапі. Терміни виконують подвійну функцію: з одного боку, вони запобігають невизначеному затягуванню процесу та стимулюють усіх учасників до оперативних дій; з іншого – забезпечують передбачуваність процесу для ініціатора. У розробленій системі передбачено два ключових дедлайни. Перший – термін збору підписів (за замовчуванням 30 днів від моменту публікації); якщо протягом цього терміну петиція не набрала порогової кількості підписів, вона автоматично переходить у статус «термін збору вичерпано». Другий – термін надання офіційної відповіді (14 днів від моменту досягнення порогу); цей дедлайн спрямований на запобігання ситуацій, коли петиція залишається без реакції адміністрації. Обидва терміни є параметрами конфігурації системи та можуть бути змінені адміністратором.

Статусна модель петиції описує дискретні стани, через які проходить звернення протягом свого життєвого циклу. У розробленій системі передбачено дев'ять статусів, що утворюють спрямований ациклічний граф переходів: чернетка (draft) – початковий стан, у якому автор може редагувати текст; на модерації (moderation) – петиція передана модератору на перевірку; відхилено

модератором (rejected) – петиція не пройшла модерацію; збір підписів (active) – петиція опублікована та доступна для підписання; на розгляді адміністрацією (review) – досягнуто поріг підписів; підтримано (approved) – адміністрація задовольнила звернення; підтримано частково (partial) – задоволено лише частину вимог; відхилено адміністрацією (declined) – звернення відхилено з обґрунтуванням; термін збору вичерпано (expired) – петиція не набрала достатньо підписів у встановлений термін. Кожен перехід між статусами супроводжується записом у журналі аудиту та генерацією відповідного сповіщення для зацікавлених сторін – автора, підписантів, модераторів.

Окремим аспектом предметної області є категоризація петицій. Тематичний розподіл дозволяє систематизувати звернення за напрямами діяльності університету та забезпечити ефективну маршрутизацію до відповідних підрозділів. У розробленій системі передбачено десять категорій: освітній процес, інфраструктура, гуртожитки, стипендія та фінанси, бібліотека, спорт і дозвілля, цифрові сервіси, безпека, харчування, самоврядування. Кожна категорія має візуальний маркер (колір), що спрощує навігацію. Категоризація сприяє не лише зручності користувачів, а й аналітичній обробці даних: адміністрація може відстежувати тематичні тренди, виявляти системні проблеми в конкретних сферах та оцінювати ефективність вжитих заходів.

Сповіщення є важливим елементом комунікаційної підсистеми, що забезпечує оперативне інформування усіх зацікавлених сторін про зміну стану петиції. У розробленій системі передбачено сім типів сповіщень: петицію опубліковано (для автора після успішної модерації), петицію відхилено (для автора із зазначенням причини), поріг підписів досягнуто (для автора та модераторів), отримано офіційну відповідь (для автора та всіх підписантів), термін збору сплинув (для автора), нова підтримка (для автора при кожному новому підписі), новий коментар (для автора). Кожне сповіщення відображається в особистому кабінеті користувача з індикатором непрочитаності та посиланням на відповідну петицію.

Комунікаційна складова системи доповнюється модулем коментування. Зареєстровані користувачі мають можливість залишати коментарі до будь-якої опублікованої петиції, що сприяє формуванню дискусійного простору навколо ініціативи. Модератори мають повноваження приховувати коментарі, що порушують правила платформи. Кожен коментар зберігається з прив'язкою до автора, часовою міткою та можливістю подальшого редагування модератором.

Особливу роль у забезпеченні довіри до системи відіграє модуль аудиту. Журнал аудиту фіксує десять типів дій: створення об'єкта, оновлення, видалення, вхід у систему, вихід із системи, підписання петиції, зняття підпису, схвалення модератором, відхилення модератором, надання офіційної відповіді. Для кожного запису зберігається ідентифікатор користувача, тип та ідентифікатор об'єкта, текстовий опис дії, IP-адреса клієнта, рядок User-Agent браузера та точна часова мітка. Журнал аудиту є доступним для перегляду через адміністративну панель Django та забезпечує можливість ретроспективного аналізу дій будь-якого користувача.

Предметна область обробки університетських петицій характеризується чітко визначеною рольовою моделлю з чотирма ролями, послідовним життєвим циклом зі статусною машиною з дев'ятьма станами, двома часовими обмеженнями (збір підписів та офіційна відповідь), тематичною категоризацією з десятьма напрямами, розвиненою комунікаційною підсистемою (сповіщення, коментарі) та повним аудитом усіх дій учасників процесу. Ці особливості формують комплекс вимог до інформаційної системи, яка має забезпечувати автоматизацію кожного етапу з дотриманням принципів прозорості, рівноправності та підзвітності.

1.2 Аналіз проблем існуючого підходу до обробки петицій

У переважній більшості вітчизняних закладів вищої освіти процес подання та розгляду колективних звернень студентів досі ґрунтується на паперовому документообігу або, в кращому випадку, на використанні

загальних комунікаційних каналів – електронної пошти, месенджерів (Telegram, Viber), соціальних мереж. Дослідження показують, що менш ніж 5% українських ЗВО мають спеціалізований цифровий інструмент для роботи з колективними зверненнями студентів. Такий стан справ породжує комплекс системних проблем, які суттєво знижують ефективність діалогу між студентською спільнотою та адміністрацією [7].

Проблема фрагментації каналів комунікації. За відсутності єдиної централізованої платформи звернення надходять через різноманітні та не пов'язані між собою канали: приймальню деканату, студентську раду, електронну пошту ректорату, офіційні сторінки університету у соціальних мережах, групи факультетів у месенджерах, усні звернення на зустрічах студентських активів із керівництвом. Це призводить до фрагментації інформації: одна й та сама проблема може бути зафіксована у кількох місцях без взаємного зв'язку, відповідальні особи не мають єдиного реєстру всіх ініціатив, а студенти позбавлені можливості дізнатися про аналогічні звернення, подані іншими. У результаті зусилля студентів розпорошуються, а адміністрація не має цілісної картини запитів спільноти [8].

Проблема верифікації підписів та автентичності. При паперовому зборі підписів існує значний ризик фальсифікації: неможливо достовірно ідентифікувати кожного підписанта, перевірити його належність до конкретного навчального закладу, виключити повторне підписання однією особою під різними іменами або підписання сторонніми особами. Крім того, паперовий підпис не має криптографічного підтвердження та може бути оскаржений. Електронна система принципово усуває ці ризики через обов'язкову автентифікацію користувачів, обмеження «один підпис на петицію» на рівні бази даних (unique constraint), фіксацію хеш-значення кожного підпису за алгоритмом SHA-256 [4] та зберігання IP-адреси та часової мітки кожної операції підписання.

Проблема прозорості процесу розгляду. Традиційний паперовий підхід не передбачає жодного механізму публічного відстеження стану розгляду

звернення. Після подання петиції студенти залишаються у інформаційному вакуумі: вони не знають, чи було звернення отримано відповідальною особою, чи зареєстровано у канцелярії, на якому етапі перебуває його розгляд, чи призначено відповідального, чи прийнято рішення. Ця невизначеність породжує обґрунтовану недовіру до інституту петицій загалом та значно зменшує мотивацію студентів до участі у процесах самоврядування. Інформаційна система вирішує цю проблему через публічне відображення поточного статусу кожної петиції, автоматичну генерацію сповіщень при зміні статусу та live-оновлення лічильника підписів через AJAX-запити з інтервалом 30 секунд.

Проблема підзвітності та аудиту. За відсутності автоматизованого журналу дій фактично неможливо встановити, хто саме обробляв петицію, на якому етапі відбулася затримка, чи дотримано встановлених термінів розгляду, які саме рішення було прийнято та з яких причин. Це створює умови для непрозорого прийняття рішень та унеможливлює контроль з боку студентської спільноти. Система аудиту, реалізована у розробленому програмному продукті, фіксує кожну дію кожного користувача (створення, оновлення, видалення, вхід, вихід, підписання, зняття підпису, схвалення, відхилення, надання відповіді) із зазначенням часу виконання, IP-адреси клієнта та ідентифікатора об'єкта, над яким здійснено дію [9].

Проблема аналітики та інформаційного забезпечення прийняття рішень. Без централізованої цифрової платформи адміністрація університету позбавлена можливості систематично аналізувати тематику звернень, виявляти повторювані проблеми, відстежувати тренди, оцінювати ефективність власних рішень та реакцій. Кожне звернення розглядається ізольовано, без урахування контексту попередніх ініціатив. Інформаційна система забезпечує агрегацію даних у формі аналітичного дашборда з ключовими показниками ефективності (KPI): загальна кількість петицій за період, кількість активних петицій, кількість петицій на офіційному розгляді, кількість підтриманих звернень, відсоток успішних петицій, середній час до досягнення порогу підписів,

динаміка створення петицій та підписів за останні 30 днів, розподіл за категоріями та статусами, рейтинг активності факультетів. Додатково система надає можливість експорту даних у формат CSV для подальшого аналізу.

Проблема доступності та інклюзивності. Паперові петиції потребують фізичної присутності студента для підписання, що створює бар'єри для участі здобувачів, які навчаються дистанційно (за змішаною формою), перебувають на практиці, стажуванні або академічній мобільності, проживають у віддалених регіонах, мають обмежену мобільність. Вебзастосунок забезпечує цілодобовий доступ до системи з будь-якого пристрою, що має підключення до мережі Інтернет, та суттєво розширює охоплення потенційної аудиторії. Адаптивний інтерфейс, реалізований у розробленій системі, коректно відображається як на настільних комп'ютерах, так і на мобільних пристроях.

Проблема масштабованості ручних процесів. В умовах великого університету із десятками тисяч студентів ручна обробка колективних звернень стає практично неможливою. Автоматизація ключових операцій – збору та підрахунку підписів, контролю часових обмежень, генерації сповіщень, переведення петиції між статусами, виявлення дублікатів – дозволяє системі обслуговувати значну кількість одночасних петицій без пропорційного збільшення адміністративного навантаження. Крім того, автоматизація виключає людський фактор при виконанні рутинних операцій: система не забуде перевірити дедлайн, не пропустить порогове значення підписів, не втратить документ [10].

Проблема збереження інституційної пам'яті. За відсутності цифрового архіву петицій та відповідей на них університет втрачає цінний масив інформації про проблеми, що хвилюють студентську спільноту, про прийняті рішення та їхню ефективність. Кожна зміна складу адміністрації або студентської ради фактично означає втрату накопиченого досвіду. Нові посадовці не мають доступу до історії попередніх звернень, не можуть оцінити результативність заходів, вжитих попередниками, та змушені починати роботу з чистого аркуша. Інформаційна система забезпечує повне збереження історії

всіх петицій, підписів, модераційних рішень та офіційних відповідей, формуючи інституційну пам'ять, доступну для аналізу та використання незалежно від кадрових змін.

Проблема відсутності зворотного зв'язку. В умовах паперового документообігу студенти, які підписали петицію, зазвичай не отримують жодної інформації про результат її розгляду. Це створює ефект «чорної скриньки», коли зусилля, витрачені на формулювання проблеми та збір підписів, не приносять видимого результату. Навіть у випадку позитивного рішення адміністрації студенти можуть не дізнатися про нього. Система сповіщень, реалізована у розробленому застосунку, автоматично інформує як автора петиції, так і всіх підписантів про кожну зміну статусу, включаючи публікацію офіційної відповіді [11].

Проблема дублювання звернень. За відсутності централізованого реєстру різні студенти або студентські групи можуть незалежно подавати звернення з однаковою або подібною тематикою. Це розпорошує підтримку спільноти між кількома ідентичними ініціативами замість концентрації зусиль на одній. Модуль премодерації розробленої системи включає функцію автоматичного виявлення потенційних дублікатів шляхом пошуку петицій з подібним заголовком серед вже опублікованих та розглянутих звернень. Модератор отримує повідомлення про знайдені збіги та може прийняти обґрунтоване рішення: відхилити дублікат або дозволити паралельне існування, якщо контекст звернень суттєво відрізняється [12].

Узагальнення виявлених проблем представлено у таблиці 1.2, де кожній проблемі зіставлено механізм її вирішення у розробленій інформаційній системі.

Таблиця 1.2

Проблеми існуючого підходу та механізми їх вирішення

№	Проблема	Механізм вирішення у розробленій системі
1	Фрагментація каналів	Єдина вебплатформа з централізованим реєстром
2	Фальсифікація підписів	Автентифікація + unique constraint + SHA-256 хеш

3	Непрозорість розгляду	Публічні статуси, сповіщення, live-оновлення
4	Відсутність підзвітності	Журнал аудиту з 10 типами дій
5	Обмежена аналітика	Дашборд з KPI, графіками, експортом CSV
6	Низька доступність	Адаптивний вебінтерфейс 24/7
7	Немасштабованість	Автоматизація переходів, дедлайнів, сповіщень
8	Втрата інст. пам'яті	Повний цифровий архів усіх дій
9	Відсутність зв. зв'язку	Автоматичні сповіщення підписантам
10	Дублювання звернень	Автоматичне виявлення дублікатів

Систематизація виявлених проблем переконливо свідчить, що впровадження спеціалізованої інформаційної системи є об'єктивною необхідністю для закладів вищої освіти, які прагнуть забезпечити ефективну, прозору та підзвітну комунікацію зі студентською спільнотою. Наступний підрозділ присвячено порівняльному аналізу існуючих цифрових рішень, які частково або повністю реалізують функціонал обробки колективних звернень.

1.3 Огляд існуючих інформаційних систем та цифрових сервісів

Для обґрунтованого проєктування інформаційної системи обробки університетських петицій доцільно здійснити порівняльний аналіз наявних рішень, що функціонують у суміжних предметних областях – від державних петиційних порталів до корпоративних систем управління зверненнями. Огляд проведено за такими критеріями: функціональні можливості (наявність механізму підписів, модерації, статусної моделі, аналітики), рольова модель, відкритість вихідного коду, можливість адаптації до специфіки закладу вищої освіти та вартість розгортання [13].

Портал «Електронні петиції» Офісу Президента України. Це офіційна державна платформа, створена відповідно до змін у Законі «Про звернення громадян», що набрали чинності у 2015 році [5]. Платформа забезпечує повний

цикл обробки петицій: реєстрацію через систему BankID або Дію, подання тексту з вибором категорії, збір електронних підписів протягом визначеного терміну (від 1 до 3 місяців залежно від рівня), автоматичний перехід на розгляд при досягненні порогу (25 000 підписів для петицій до Президента), публікацію офіційної відповіді. Технологічно портал реалізовано як вебзастосунок з інтеграцією із державними сервісами автентифікації, що забезпечує високий рівень верифікації підписантів. Головні переваги платформи – юридична обов'язковість розгляду та надійна ідентифікація підписантів. Однак у контексті університетського застосування ця система має суттєві обмеження: надмірно високий поріг підписів (25 000 при чисельності університету 10-20 тисяч студентів), відсутність премодерації контенту, неможливість адаптації категорій під специфіку ЗВО, відсутність аналітичного модуля для адміністрації, закритий вихідний код.

Change.org. Міжнародна комерційна платформа для створення онлайн-петицій, заснована у 2007 році, яка налічує понад 500 мільйонів користувачів у 196 країнах світу [6]. Платформа дозволяє будь-кому створити петицію без попередньої модерації, зібрати підписи та поширити її через соціальні мережі та електронну пошту. Change.org використовує алгоритми рекомендацій для просування петицій, що відповідають інтересам конкретного користувача. Переваги: надзвичайно низький поріг входу (для створення петиції достатньо мати email), глобальне охоплення аудиторії, інтуїтивний інтерфейс, потужні інструменти просування. Недоліки в контексті університетського застосування: відсутність верифікації підписантів (достатньо вказати будь-який email, що не перевіряється на належність до конкретної організації), відсутність механізму обов'язкової відповіді адресата, неможливість інтеграції з внутрішніми системами університету, комерційна модель монетизації з елементами реклами та платного просування, зберігання даних на серверах за межами України.

Платформа «Дія» та портал електронних послуг. Платформа «Дія» реалізує концепцію «держави в смартфоні» та включає широкий спектр електронних послуг, у тому числі функціонал подання окремих типів звернень

до органів влади [7]. Хоча основний фокус платформи – це надання державних послуг (цифрові документи, реєстрації, довідки), а не механізм петицій, її архітектурні та дизайнерські рішення можуть слугувати цінним орієнтиром при проєктуванні університетської системи: мобільна автентифікація через QR-код, push-сповіщення про зміну статусу звернення, інтеграція з державними реєстрами, мінімалістичний інтерфейс. Обмеження: закритий вихідний код, орієнтація виключно на державний рівень, неможливість самостійного розгортання, відсутність модуля колективних звернень із механізмом підписів.

Використання Google Forms як механізму збору зворотного зв'язку. Значна кількість українських університетів використовує Google Forms як спрощений інструмент для збору пропозицій та скарг від студентів. Такий підхід приваблює відсутністю потреби у програмній розробці та знайомим інтерфейсом. Водночас він має принципові обмеження, що унеможливлюють повноцінну роботу з петиціями: відсутність системи підписів та підтримки колективних звернень, відсутність життєвого циклу (статусної моделі) звернення, неможливість публічного відстеження статусу конкретного звернення, відсутність розмежування ролей між користувачами, залежність від сторонньої хмарної інфраструктури Google, обмежені аналітичні можливості (лише базова агрегація відповідей), відсутність можливості прикріплення офіційної відповіді до конкретного звернення.

Ushahidi. Платформа з відкритим вихідним кодом (ліцензія LGPL-3.0), спочатку розроблена у 2008 році для моніторингу виборчих порушень у Кенії, яка згодом була адаптована для широкого спектра сценаріїв: від управління кризовими ситуаціями до збору зворотного зв'язку від громадян [8]. Платформа забезпечує збір, модерацію, візуалізацію та аналіз повідомлень із різних каналів (вебформа, SMS, email, Twitter, RSS). Переваги: відкритий код, наявність REST API, підтримка мобільних платформ, геолокаційна візуалізація, гнучка система ролей та дозволів. Недоліки: орієнтація переважно на геолокаційні дані та кризове реагування, відсутність специфічного механізму підписів під зверненнями, складне первинне налаштування та розгортання, англомовний

інтерфейс без офіційної україномовної локалізації, надмірна функціональність для задач рівня університетських петицій [14].

JIRA Service Management (Atlassian). Корпоративне рішення для управління зверненнями, запитами та інцидентами, що належить до класу систем ITSM (IT Service Management) [9]. Забезпечує повний цикл обробки тікетів із настроюваними статусами та переходами, гнучкою рольовою моделлю, SLA-метриками з контролем часу реакції та вирішення, інтеграцією з іншими продуктами екосистеми Atlassian (Confluence, Bitbucket). Переваги: потужна гнучкість налаштування робочих процесів через редактор Workflow, розвинена аналітика з можливістю побудови власних дашбордів, масштабованість від малих команд до підприємств. Недоліки в контексті університету: комерційна ліцензія з високою вартістю для освітніх закладів, надмірна складність інтерфейсу та налаштування для задач рівня петицій, відсутність механізму публічного голосування (підписання), орієнтація на процеси IT-підтримки, а не на демократичні процедури.

Результати порівняльного аналізу розглянутих систем за ключовими критеріями узагальнено у таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз існуючих систем обробки звернень

Система	Електронні підписи	Преморде-рація	Статусна модель	Аналі-тика	Рольо-вий доступ	Відкри-тий код	Адапта-ція ЗВО
Е-петиції Президента	Так	Ні	Так	Ні	Так	Ні	Ні
Change.org	Так	Ні	Частково	Частково	Ні	Ні	Ні
Дія	Ні	Ні	Частково	Ні	Ні	Ні	Ні
Google Forms	Ні	Ні	Ні	Частково	Ні	–	Частково
Ushahidi	Ні	Так	Так	Так	Так	Так	Частково
JIRA SM	Ні	Так	Так	Так	Так	Ні	Частково

Розроблена система	Так	Так	Так	Так	Так	Так	Так

Для більш детального порівняння розглянемо функціональне покриття кожної системи за ключовими операціями петиційного процесу: створення звернення, збір підписів, модерація, відстеження статусу, офіційна відповідь, аналітика, аудит дій. Результати цього розширеного аналізу представлено у таблиці 1.3.

Таблиця 1.3

Покриття ключових операцій петиційного процесу

Система	Ство-рення	Підписи	Модера-ція	Статус у реальн. часі	Офіційна відповідь	Аналі-тика	Аудит дій
Е-петиції Президента	Так	Так	Ні	Частково	Так	Ні	Частково
Change.org	Так	Так	Ні	Так	Ні	Частков о	Ні
Дія	Так	Ні	Ні	Частково	Ні	Ні	Частково
Google Forms	Так	Ні	Ні	Ні	Ні	Частков о	Ні
Ushahidi	Так	Ні	Так	Так	Ні	Так	Так
JIRA SM	Так	Ні	Так	Так	Так	Так	Так
Розроблена система	Так	Так	Так	Так	Так	Так	Так

Розширений аналіз підтверджує раніше зроблений висновок: лише розроблювана система забезпечує повне покриття всіх операцій петиційного процесу. Найближчими за функціональністю є JIRA Service Management та портал електронних петицій Президента, проте перша система не підтримує механізм публічних підписів, а друга – не має модулів модерації та аналітики.

Окремо слід зазначити технологічні аспекти проаналізованих рішень. Портал електронних петицій Президента реалізовано на платформі Drupal (PHP), Change.org використовує Ruby on Rails, Ushahidi побудована на Laravel (PHP) з фронтендом на AngularJS, JIRA – це Java-застосунок. Кожна з цих технологій має свої переваги, проте для задач рівня університетської системи обрано Python та Django як оптимальне поєднання простоти розробки, продуктивності та наявності готових компонентів для автентифікації, авторизації та роботи з базою даних.

Варто також розглянути досвід зарубіжних університетів у впровадженні петиційних систем. Ряд британських університетів (University of Edinburgh, University of Sheffield) використовують внутрішні платформи для збору студентських ініціатив, інтегровані з системами студентського самоврядування. Ці платформи, як правило, є закритими розробками, адаптованими до конкретного навчального закладу, та не доступні для вільного використання. Їхній досвід, однак, підтверджує ефективність цифрового підходу до обробки петицій в академічному середовищі та доцільність розробки аналогічного рішення для українських ЗВО.

Порівняльний аналіз переконливо демонструє, що жодна з розглянутих систем не забезпечує повного набору функцій, необхідних для комплексної автоматизації петиційного процесу в межах університету. Державні платформи проєктувалися для масштабу країни та не адаптовані до потреб окремого ЗВО. Комерційні рішення (Change.org, JIRA) або надмірно дорогі, або не містять ключових механізмів (підписи, публічне голосування). Універсальні інструменти (Google Forms) не підтримують специфічну бізнес-логіку

петиційного процесу. Платформи з відкритим кодом (Ushahidi) потребують суттєвої адаптації та не містять модуля електронних підписів [15].

Вищезазначене обґрунтовує доцільність створення спеціалізованого програмного рішення, яке інтегрує переваги проаналізованих систем та усуває їхні недоліки. Розроблювана система має поєднувати: повний життєвий цикл петиції з дев'ятьма статусами та автоматичними переходами; верифікований електронний підпис із криптографічним підтвердженням; премодерацію з автоматичним виявленням дублікатів; чотирирівневу рольову модель; аналітичний дашборд із візуалізацією ключових метрик; журнал аудиту всіх дій; систему сповіщень; адаптацію до організаційної структури конкретного університету через конфігурацію факультетів, категорій та порогових значень.

Варто зазначити, що серед проаналізованих рішень лише розроблювана система одночасно задовольняє всім семи критеріям порівняння (табл. 1.1). Це підтверджує, що існуючі рішення не можуть бути використані як готовий продукт для автоматизації петиційного процесу в університеті – кожне з них вимагає або значної кастомізації (Ushahidi, JIRA), або взагалі не підтримує базових функцій предметної області (Google Forms, «Дія»). Крім того, критично важливою є можливість самостійного розгортання та контролю над даними, яку забезпечують лише рішення з відкритим кодом.

Додатковим аргументом на користь власної розробки є специфіка правового регулювання персональних даних в Україні. Згідно із Законом України «Про захист персональних даних», обробка персональних даних здійснюється за умови наявності правової підстави та із забезпеченням їх захисту. Використання зовнішніх хмарних сервісів (Change.org, Google Forms) для збору підписів студентів передбачає передачу персональних даних третім сторонам, що може суперечити внутрішній політиці ЗВО. Власна система, розгорнута на серверах університету, забезпечує повний контроль над зберіганням та обробкою персональних даних.

Ще одним фактором є можливість інтеграції з існуючою інформаційною інфраструктурою університету. Система на базі Django може бути інтегрована з

корпоративним каталогом користувачів (LDAP/Active Directory), електронним кабінетом студента, системою управління навчанням (LMS) та іншими внутрішніми сервісами. Жодна з проаналізованих зовнішніх платформ не надає таких можливостей інтеграції без суттєвої доробки.

1.4 Постановка задачі розроблення інформаційної системи

На підставі проведеного аналізу предметної області обробки університетських петицій, систематизації проблем існуючого підходу та порівняльного огляду наявних цифрових рішень сформульовано завдання на розробку інформаційної системи.

Об'єкт дослідження – процес подання, обробки та розгляду колективних звернень (петицій) у закладі вищої освіти.

Предмет дослідження – інформаційна система, що автоматизує повний цикл обробки університетських петицій та забезпечує електронний документообіг від формулювання ініціативи до отримання офіційної відповіді.

Мета роботи – розробка вебзастосунку для обробки університетських петицій, який реалізує повний життєвий цикл звернення з дев'ятьма статусами, забезпечує прозорість, підзвітність та аналітику процесу, а також адаптований до організаційної структури закладу вищої освіти.

Для досягнення визначеної мети необхідно вирішити такі задачі:

- дослідити предметну область обробки петицій у закладах вищої освіти, визначити ролі учасників, етапи процесу, часові обмеження та особливості категоризації звернень;
- проаналізувати існуючі цифрові рішення для обробки колективних звернень (державні портали, комерційні та відкриті платформи), виявити їхні переваги та недоліки щодо застосування в університетському середовищі;

- сформулювати функціональні та нефункціональні вимоги до інформаційної системи на основі виявлених проблем та потреб цільових користувачів;
- спроєктувати архітектуру системи за шаблоном MVT, визначити структуру бази даних з урахуванням вимог нормалізації та змоделювати ключові бізнес-процеси;
- обґрунтувати вибір технологій та засобів розробки (мова програмування, фреймворк, СУБД, інструменти візуалізації);
- реалізувати серверну частину вебзастосунку з модулями автентифікації та управління ролями, управління петиціями, електронного підписання, модерації, аналітики та сповіщень;
- реалізувати клієнтську частину з мінімалістичним інтуїтивним інтерфейсом, адаптивним дизайном для настільних та мобільних пристроїв;
- провести модульне та інтеграційне тестування системи, оцінити коректність реалізації функціональних модулів;
- сформувати рекомендації щодо впровадження та експлуатації системи у закладі вищої освіти.

Розроблювана система має задовольняти комплексу вимог, які поділяються на функціональні та нефункціональні.

Функціональні вимоги. Реєстрація та автентифікація користувачів із прив'язкою до факультету, курсу та студентського квитка. Чотирирівневий рольовий доступ: студент, викладач, модератор, адміністратор. Створення петицій із розширеною багатосекційною формою, яка включає заголовок, детальний опис проблеми, обґрунтування актуальності, формулювання очікуваного результату, вибір адресата (ректорат, деканат, кафедра, студентське самоврядування) та категорії з десяти доступних, можливість завантаження додаткових матеріалів (файлів). Збереження чернеток із можливістю подальшого редагування. Електронне підписання петицій з

контролем унікальності підпису (один користувач – одна петиція), хешуванням (SHA-256), можливістю анонімного підпису та відкликання до завершення збору. Премодерація з трьома можливими рішеннями (опублікувати, відхилити, повернути), автоматичним виявленням потенційних дублікатів, журналюванням рішень. Автоматичний перехід петиції на офіційний розгляд при досягненні порогу підписів (100 за замовчуванням, конфігурується). Подання офіційної відповіді із зазначенням рішення (підтримано, частково, відхилено) та прикріпленням документів. Система внутрішніх сповіщень про зміну статусу петиції, нові підписи, офіційні відповіді. Аналітичний дашборд з ключовими метриками, графіками динаміки та розподілу. Експорт даних у CSV. Коментування петицій зареєстрованими користувачами. Журнал аудиту з фіксацією всіх дій [14].

Нефункціональні вимоги. Адаптивний інтерфейс, що коректно відображається на екранах від 320px (мобільні пристрої) до широкоформатних моніторів. Час відгуку сервера не більше 500 мс для основних операцій читання та не більше 1000 мс для операцій запису. Захист від типових вебвразливостей: CSRF-атаки (вбудований захист Django), XSS (автоматичне екранування у шаблонах), SQL-ін'єкції (параметризовані запити через ORM). Хешування паролів через PBKDF2 із сіллю (стандарт Django). Повна локалізація інтерфейсу українською мовою. Підтримка часового поясу Europe/Kyiv.

Систему планується реалізувати як вебзастосунок на базі фреймворку Django версії 5.0 (мова програмування Python 3.10+) із використанням СУБД SQLite для зберігання даних. Вибір Django обумовлено кількома факторами [10]: наявність вбудованих механізмів автентифікації та авторизації з розширюваною моделлю користувача; потужний ORM для роботи з базою даних, що підтримує міграції схеми; шаблонізатор із системою контекстних процесорів; вбудована адміністративна панель з можливістю налаштування; активна спільнота розробників та велика кількість документації. SQLite обрано як СУБД, оскільки вона не потребує окремого серверного процесу,

поставляється разом із Python, забезпечує атомарність транзакцій та ідеально підходить для систем із помірним навантаженням [11].

Клієнтська частина реалізується засобами Django Templates із мінімальним використанням JavaScript. Бібліотека Chart.js (версія 4.x, ліцензія MIT) використовується для візуалізації аналітичних даних у дашборді (лінійні, стовпчикові, кільцеві та горизонтальні діаграми). Шрифти Manrope та JetBrains Mono завантажуються через Google Fonts. Стилізація здійснюється через кастомний CSS-файл без залежності від зовнішніх CSS-фреймворків, що забезпечує мінімальний розмір завантажуваних ресурсів та повний контроль над дизайном.

Архітектурно система відповідає шаблону MVT (Model – View – Template), що є стандартним для фреймворку Django та являє собою варіацію класичного MVC (Model – View – Controller) [10]. Рівень Model реалізує бізнес-логіку та взаємодію з базою даних через Django ORM. Рівень View обробляє HTTP-запити, виконує валідацію даних та формує контекст для шаблонів. Рівень Template відповідає за генерацію HTML-відповідей на основі переданого контексту.

Проєкт складатиметься з чотирьох застосунків Django (apps), кожен із яких інкапсулює логічно пов'язаний набір моделей, представлень та шаблонів: accounts (користувачі, ролі, факультети, автентифікація, журнал аудиту), petitions (петиції, підписи, категорії, коментарі, сповіщення), moderation (рішення модераторів, офіційні відповіді), analytics (аналітичний дашборд та експорт даних).

Інформаційна база системи включатиме десять основних сутностей: User (користувач із розширеною моделлю, що включає роль, факультет, курс, студентський квиток), Faculty (факультет), Category (категорія петиції з кольоровим маркером), Petition (петиція зі статусною машиною, адресатом, дедлайнами та лічильниками), Signature (електронний підпис із хеш-значенням SHA-256), PetitionResponse (офіційна відповідь із рішенням та вкладенням), Comment (коментар до петиції), Notification (сповіщення для користувача),

ModerationDecision (рішення модератора з обґрунтуванням), AuditLog (запис журналу аудиту з деталями дії, часом, IP-адресою). Зв'язки між сутностями реалізуються через зовнішні ключі з дотриманням принципів нормалізації (третя нормальна форма). Цілісність даних забезпечуватиметься обмеженнями унікальності (unique constraint на пару petition-user для таблиці підписів), каскадними операціями та індексами на ключових полях.

Безпека системи забезпечуватиметься комплексом заходів: CSRF-захист на всіх формах (вбудований механізм Django з використанням токенів); автоматичне екранування виводу у шаблонах для запобігання XSS; параметризовані запити через ORM для захисту від SQL-ін'єкцій; хешування паролів за алгоритмом PBKDF2 із унікальною сіллю для кожного користувача; журналювання входів та виходів через сигнали Django (user_logged_in, user_logged_out); перевірка прав доступу через декоратори представлень (@login_required, @moderator_required); збереження IP-адреси та User-Agent для кожного запису аудиту.

Тестування системи планується здійснити на кількох рівнях: модульне тестування (unit tests) моделей, включаючи перевірку бізнес-логіки статусної машини, розрахунку прогресу, контролю дедлайнів; інтеграційне тестування представлень (views) з використанням тестового HTTP-клієнта Django; функціональне тестування ключових сценаріїв: реєстрація, створення петиції, підписання, модерація, офіційна відповідь. Для тестування використовуватиметься вбудований фреймворк django.test, що базується на стандартній бібліотеці unittest мови Python та забезпечує ізольоване тестове середовище з окремою базою даних.

Результатом виконання роботи є повнофункціональний вебзастосунок, розгорнутий у локальному середовищі для демонстрації та тестування, з можливістю подальшого впровадження у закладі вищої освіти. Застосунок включатиме набір демонстраційних даних (шість факультетів, десять категорій, облікові записи з різними ролями, одинадцять петицій у різних статусах із

підписами, коментарями та офіційними відповідями), що дозволить оцінити функціональні можливості системи без додаткової підготовки середовища [15].

Кожна сутність бази даних має чітко визначений набір атрибутів. Сутність User розширює стандартну модель Django AbstractUser та включає: роль (student, teacher, moderator, admin), по батькові, зовнішній ключ на факультет, номер студентського квитка, курс навчання, аватар, біографію, прапорець верифікації email, часові мітки створення та оновлення. Сутність Petition є центральною та включає: заголовок (до 250 символів), slug (автоматична транслітерація українських символів у латиницю), опис проблеми, обґрунтування, очікуваний результат, зовнішні ключі на автора та категорію, адресат (ректорат, деканат, кафедра, студрада, інше), статус (одне з дев'яти значень), кількість необхідних та зібраних підписів, кількість переглядів, дедлайни збору та розгляду, часові мітки подання, публікації, досягнення порогу та закриття, прапорці рекомендованості та анонімності.

Сутність Signature містить зовнішні ключі на петицію та користувача (з обмеженням унікальності на цю пару), хеш-значення підпису (64-символьний рядок SHA-256), необов'язковий коментар (до 500 символів), прапорець анонімності та IP-адресу підписанта. Сутність ModerationDecision зберігає зовнішні ключі на петицію та модератора, тип рішення (опубліковано, відхилено, повернуто на доопрацювання) та текстове обґрунтування. Сутність AuditLog фіксує: користувача, тип дії (створення, оновлення, видалення, вхід, вихід, підписання, зняття підпису, схвалення, відхилення, відповідь), тип та ідентифікатор об'єкта, текстовий опис, IP-адресу, User-Agent та часову мітку.

Для забезпечення продуктивності запитів до бази даних передбачено створення індексів на ключових полях: складений індекс (status, created_at) для фільтрації петицій за статусом та сортування за датою; індекс на signatures_count для сортування за популярністю; індекс (author, created_at) для формування профілю автора; складений індекс (user, created_at) та (user, is_read) для ефективної роботи з сповіщеннями.

Конфігурація системи забезпечується через файл налаштувань Django (settings.py), де визначено три ключових параметри петиційного процесу: PETITION_SIGNATURES_THRESHOLD (поріг підписів, за замовчуванням – 100), PETITION_COLLECTION_DAYS (термін збору підписів, за замовчуванням – 30 днів), PETITION_REVIEW_DAYS (термін надання офіційної відповіді, за замовчуванням – 14 днів). Ці параметри можуть бути змінені адміністратором без модифікації програмного коду, що забезпечує гнучкість адаптації системи до потреб конкретного університету.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБРОБКИ УНІВЕРСИТЕТСЬКИХ ПЕТИЦІЙ

2.1 Функціональні та нефункціональні вимоги до системи

На підставі аналізу предметної області, проведеного у першому розділі, та результатів порівняння існуючих рішень, сформовано систематизований перелік вимог до інформаційної системи обробки університетських петицій. Вимоги розподілено на дві групи – функціональні, що визначають поведінку системи з точки зору користувача, та нефункціональні, що встановлюють якісні обмеження на реалізацію.

Функціональні вимоги описують конкретні операції, які система має підтримувати. Кожна вимога формулюється з позиції кінцевого користувача та співвіднесена з відповідною роллю – студентом, викладачем, модератором або адміністратором. Перелік функціональних вимог подано у таблиці 2.1.

Таблиця 2.1

Функціональні вимоги до інформаційної системи

ID	Опис вимоги	Роль
FR-01	Реєстрація з прив’язкою до факультету, курсу та номера студентського квитка	Студент, Викладач
FR-02	Автентифікація через логін та пароль з хешуванням PBKDF2	Усі
FR-03	Створення петиції з багатосекційною формою (заголовок, опис, обґрунтування, очікуваний результат, категорія, адресат, вкладення)	Студент, Викладач
FR-04	Збереження чернетки петиції з можливістю подальшого редагування	Студент, Викладач
FR-05	Подання петиції на модерацію	Студент, Викладач
FR-06	Електронне підписання петиції з хешуванням SHA-256, контролем унікальності та можливістю анонімного підпису	Студент, Викладач

Продовження таблиці 2.1

FR-07	Відкликання підпису до завершення терміну збору	Студент, Викладач
FR-08	Коментування опублікованих петицій	Студент, Викладач
FR-09	Пошук та фільтрація петицій за категорією, статусом, ключовими словами із сортуванням	Усі
FR-10	Перегляд сповіщень про зміну статусу петицій, нові підписи, відповіді	Студент, Викладач
FR-11	Премодерація петицій із трьома рішеннями: опублікувати, відхилити, повернути на доопрацювання	Модератор
FR-12	Автоматичне виявлення потенційних дублікатів під час модерації	Модератор
FR-13	Надання офіційної відповіді на петицію з рішенням (підтримано, частково, відхилено) та вкладенням	Адміністратор
FR-14	Перегляд аналітичного дашборда з KPI, графіками динаміки та розподілу	Адміністратор
FR-15	Експорт переліку петицій у формат CSV	Адміністратор
FR-16	Автоматичний перехід петиції у статус «На розгляді» при досягненні порогу підписів	Система
FR-17	Автоматичне закриття петиції при вичерпанні терміну збору підписів	Система
FR-18	Фіксація кожної дії у журналі аудиту із зазначенням IP-адреси та User-Agent	Система

Як видно з таблиці 2.1, система має підтримувати 18 функціональних вимог, розподілених між чотирма ролями та автоматичними процесами. Вимоги FR-16, FR-17 та FR-18 реалізуються без участі користувача – вони забезпечують автоматизацію переходів між станами та ведення журналу аудиту.

Нефункціональні вимоги визначають якісні характеристики системи та обмеження на її реалізацію. Їх систематизовано у таблиці 2.2.

Таблиця 2.2

Нефункціональні вимоги до інформаційної системи

ID	Характеристика	Опис
NFR-01	Продуктивність	Час відгуку сервера не більше 500 мс для операцій читання та 1000 мс для запису
NFR-02	Адаптивність	Коректне відображення інтерфейсу на екранах від 320px до широкоформатних моніторів
NFR-03	Безпека: CSRF	Захист від CSRF-атак на всіх формах через вбудований механізм Django з токенами
NFR-04	Безпека: XSS	Автоматичне екранування виводу у шаблонах Django для запобігання міжсайтовому скриптингу
NFR-05	Безпека: SQL	Параметризовані запити через Django ORM для захисту від SQL-ін’єкцій
NFR-06	Безпека: паролі	Хешування паролів за алгоритмом PBKDF2 із унікальною сіллю
NFR-07	Локалізація	Повна локалізація інтерфейсу українською мовою, часовий пояс Europe/Kyiv
NFR-08	Розширюваність	Модульна архітектура з чотирма Django-застосунками, що дозволяє незалежне розширення
NFR-09	Переносність	Кросплатформне розгортання (Linux, Windows, macOS) без додаткових системних залежностей
NFR-10	Супроводжуваність	Дотримання конвенцій Django, документування коду через docstrings

Сформований комплекс з 18 функціональних та 10 нефункціональних вимог забезпечує повне покриття потреб усіх ролей системи та встановлює чіткі якісні орієнтири для етапу реалізації. Зокрема, вимоги NFR-03 – NFR-06 утворюють комплексну модель безпеки, що відповідає рекомендаціям OWASP для вебзастосунків. Наступний підрозділ присвячено моделюванню бізнес-процесів, що реалізують визначені функціональні вимоги.

2.2 Моделювання бізнес-процесів і сценаріїв взаємодії користувачів

Моделювання бізнес-процесів є важливим етапом проєктування інформаційної системи, оскільки воно дозволяє формалізувати послідовність дій учасників, визначити точки прийняття рішень та ідентифікувати автоматизовані операції. Для моделювання використано нотацію UML, яка є загальноприйнятим стандартом у галузі програмної інженерії.

Діаграма варіантів використання (Use Case Diagram). Для визначення функціональних меж системи та розподілу відповідальності між акторами побудовано діаграму варіантів використання. Діаграма відображає чотири категорії акторів (студент, викладач, модератор, адміністратор) та тринадцять варіантів використання, кожен з яких відповідає одній або кільком функціональним вимогам з таблиці 2.1. Результат побудови діаграми представлено на рис. 2.1.

Рис. 2.1 – Діаграма варіантів використання системи

Як видно з діаграми, студент та викладач мають доступ до семи варіантів використання, що охоплюють повний цикл взаємодії з петиціями – від реєстрації до перегляду сповіщень. Модератор відповідає за три специфічні варіанти, пов’язані з контролем якості контенту та наданням офіційних відповідей. Адміністратор має доступ до аналітичних інструментів та журналу аудиту, а також наслідує повноваження модератора.

Діаграма діяльності (Activity Diagram). Центральним бізнес-процесом системи є життєвий цикл петиції, який охоплює дев’ять станів та залучає всіх чотирьох акторів. Для формалізації цього процесу побудовано діаграму діяльності з доріжками відповідальності (swimlanes), що наочно демонструє розподіл дій між учасниками (рис. 2.2).

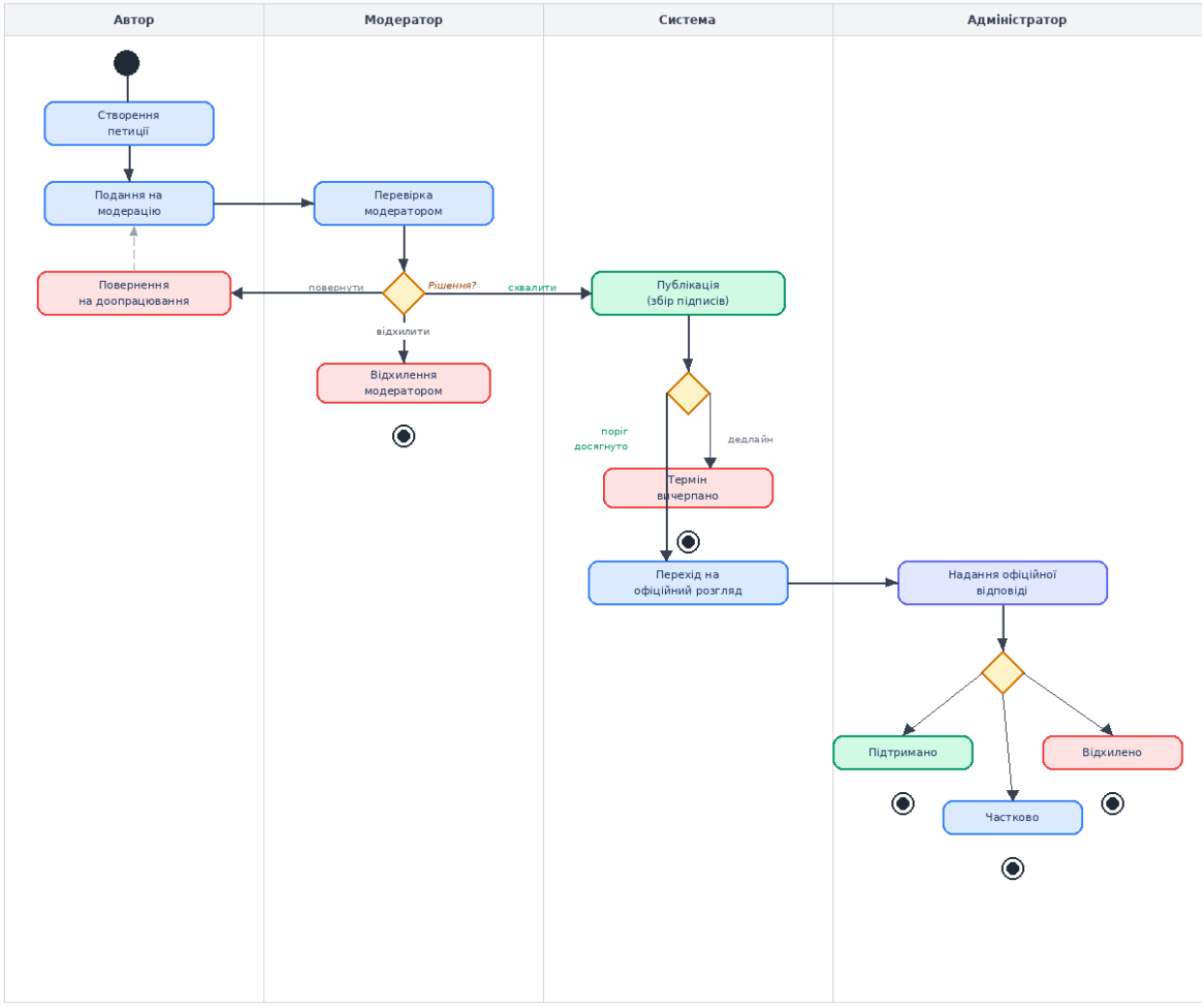

Рис. 2.2 – Діаграма діяльності: життєвий цикл петиції

Діаграма демонструє два критичні автоматичні переходи, що виконуються системою без участі користувачів. Перший – перехід зі стану «Збір підписів» у стан «На розгляді адміністрацією» при досягненні порогового значення (100 підписів за замовчуванням). Другий – перехід у стан «Термін вичерпано» при закінченні 30-денного терміну збору. Ці переходи реалізуються через метод update_signatures_count() моделі Petition та перевірку check_expiration() відповідно.

Статусна модель петиції. Послідовність переходів між станами формалізовано у вигляді діаграми переходів станів (State Machine Diagram). Модель описує кінцевий детермінований автомат з множиною станів S = {draft, moderation, rejected, active, review, approved, partial, declined, expired}, початковим станом s_0 = draft та функцією переходів, що визначає допустимі трансформації (рис. 2.3).

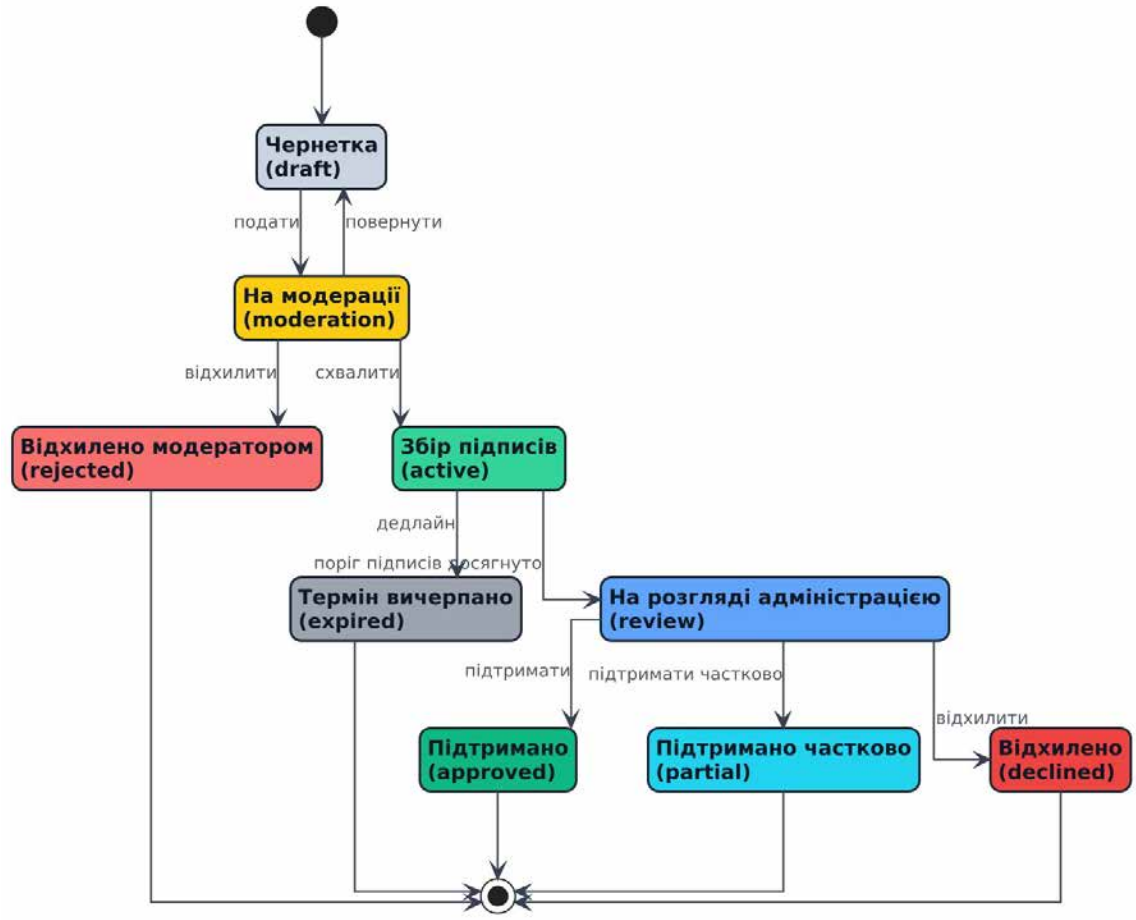

Рис. 2.3 – Діаграма переходів станів петиції

Кожен перехід ініціюється конкретною роллю або автоматичним механізмом системи. Автор ініціює перехід draft → moderation; модератор – moderation → active (публікація), moderation → rejected (відхилення), moderation → draft (повернення на доопрацювання); система автоматично здійснює active → review (при досягненні порогу) та active → expired (при вичерпанні терміну); адміністратор – review → approved, review → partial, review → declined. Реалізація статусної машини у коді забезпечується полем status моделі Petition з типом TextChoices та відповідними методами оновлення.

Для систематизації сценаріїв взаємодії користувачів визначено вісім ключових сценаріїв, кожен з яких відповідає конкретному варіанту використання. Сценарії подано у таблиці 2.3.

Таблиця 2.3

Ключові сценарії взаємодії користувачів із системою

ID	Сценарій	Опис послідовності дій
UC-01	Реєстрація	Студент заповнює форму (ПІБ, email, факультет, курс, номер квитка, пароль), система створює обліковий запис з роллю student та перенаправляє на сторінку входу
UC-02	Створення петиції	Автор заповнює багатосекційну форму, обирає дію «Зберегти чернетку» або «Подати на модерацію»; система фіксує дію в журналі аудиту
UC-03	Підписання	Користувач натискає «Підписати», обирає режим (відкритий/анонімний); система генерує SHA-256 хеш, оновлює лічильник, надсилає сповіщення автору
UC-04	Премодерація	Модератор переглядає петицію у черзі, бачить попередження про дублікати, обирає рішення (опублікувати/відхилити/повернути) з обґрунтуванням
UC-05	Офіційна відповідь	Адміністратор переглядає петицію зі статусом review, заповнює форму з рішенням та текстом; система оновлює статус та сповіщує автора й підписантів
UC-06	Перегляд дашборда	Адміністратор відкриває аналітичний дашборд, бачить KPI (кількість петицій, підписів, користувачів, відсоток успішності), графіки динаміки та розподілу

Продовження таблиці 2.3

UC-07	Пошук та фільтрація	Користувач вводить ключове слово, обирає категорію, статус, спосіб сортування; система повертає відфільтрований пагінований список
UC-08	Відкликання підпису	Користувач натискає «Зняти підпис» на сторінці петиції; система видаляє запис, оновлює лічильник, фіксує дію у журналі аудиту

Визначені сценарії повністю покривають усі функціональні вимоги з таблиці 2.1 та відображають взаємодію кожної ролі з системою. Кожен сценарій реалізується через відповідне представлення (view) у коді застосунку та супроводжується записом у журналі аудиту.

2.3 Проєктування структури системи

Архітектура інформаційної системи визначає її внутрішню організацію, принципи взаємодії компонентів та розподіл відповідальності між рівнями. Для розробки обрано архітектурний шаблон MVT (Model – View – Template), що є стандартним для фреймворку Django та являє собою варіацію класичного шаблону MVC (Model – View – Controller).

Рівень Model інкапсулює бізнес-логіку та структуру даних. Моделі Django визначають схему бази даних через декларативний опис полів та зв’язків, а також містять методи бізнес-логіки – розрахунок прогресу збору підписів, перевірку дедлайнів, автоматичну зміну статусів. Взаємодія з базою даних здійснюється виключно через Django ORM, що забезпечує абстракцію від конкретної СУБД та захист від SQL-ін’єкцій.

Рівень View обробляє HTTP-запити, виконує валідацію вхідних даних через форми (Forms), застосовує бізнес-правила та формує контекст для шаблонів. Представлення реалізовано у вигляді функцій з декораторами для контролю доступу (@login_required, @moderator_required). Кожне

представлення відповідає конкретному URL-маршруту, визначеному у файлі urls.py відповідного застосунку.

Рівень Template відповідає за генерацію HTML-відповідей. Шаблони використовують механізм успадкування (extends) з базовим шаблоном base.html, що забезпечує єдиний візуальний стиль та спільну навігацію для всіх сторінок. Контекстні процесори (context_processors) додають глобальну статистику (кількість активних петицій, непрочитаних сповіщень) до кожного шаблону.

Загальну архітектуру системи за шаблоном MVT представлено на рис. 2.4.

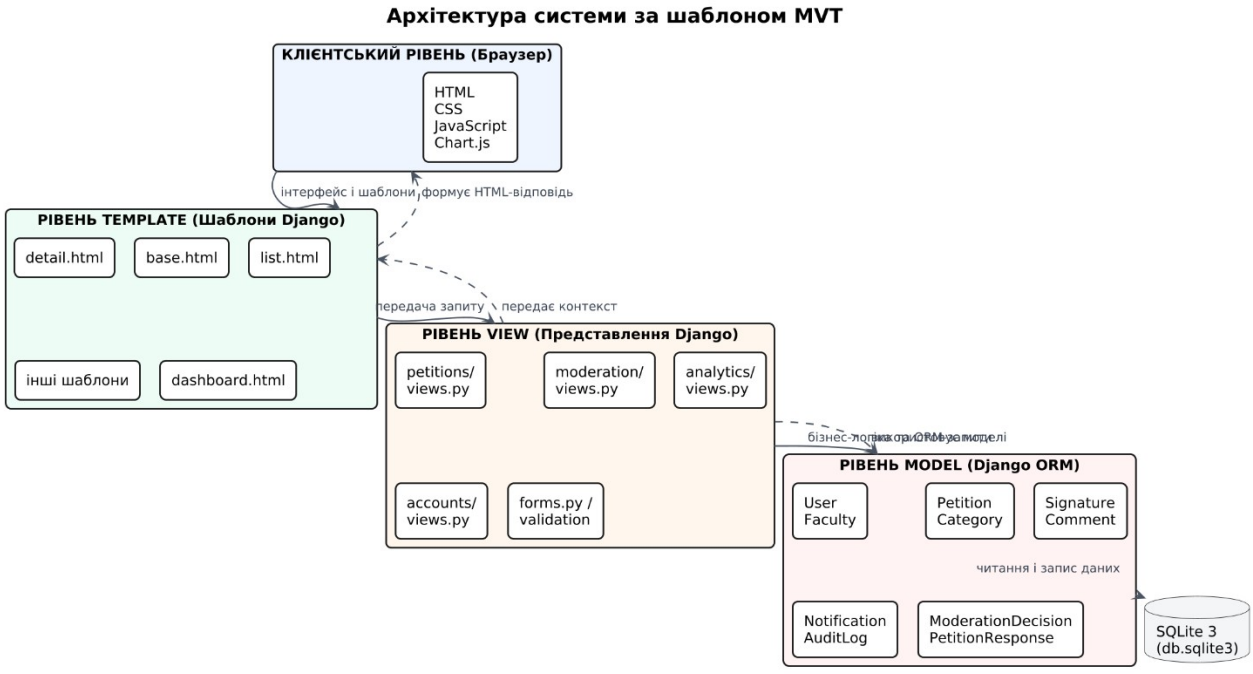

Рис. 2.4 – Архітектура системи за шаблоном MVT

Проєкт організовано у чотири застосунки (apps) Django, кожен з яких інкапсулює логічно пов’язаний набір моделей, представлень, форм та шаблонів. Діаграму компонентів системи представлено на рис. 2.5.

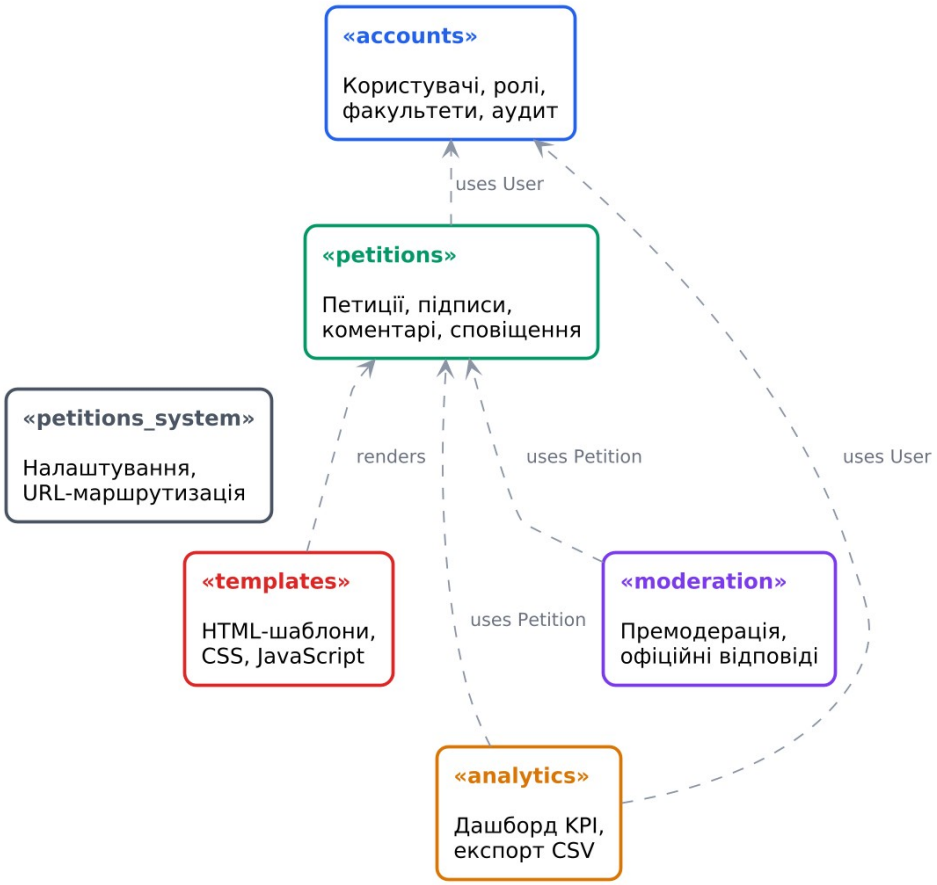

Рис. 2.5 – Діаграма компонентів системи

Застосунок accounts відповідає за управління користувачами, рольовою моделлю, факультетами та журналом аудиту. Застосунок petitions реалізує основну бізнес-логіку – моделі петицій, підписів, коментарів, сповіщень, а також представлення для створення, перегляду, підписання петицій та фільтрації списку. Застосунок moderation забезпечує функціонал премодерації та надання офіційних відповідей. Застосунок analytics реалізує аналітичний дашборд із KPI-метриками та експорт даних у CSV.

Залежності між компонентами мають ієрархічний характер: petitions та moderation залежать від accounts (використовують модель User), moderation залежить від petitions (оперує моделлю Petition), analytics залежить від petitions та accounts (агрегує дані з обох). Такий розподіл забезпечує слабку зв’язність (low coupling) між модулями та можливість їх незалежного тестування.

Таблиця 2.4

Характеристика застосунків Django у складі системи

Застосунок	Моделі	Функціональність	Views
accounts	User, Faculty, AuditLog	Реєстрація, вхід, профіль, журнал аудиту	3
petitions	Petition, Signature, Category, Comment, Notification	Створення, перегляд, підписання, коментування, сповіщення	6
moderation	ModerationDecision	Черга модерації, перевірка, офіційна відповідь	4
analytics	–	Дашборд KPI, графіки, експорт CSV	2

Для візуалізації фізичного розміщення компонентів системи побудовано діаграму розгортання (рис. 2.6), яка відображає два вузли: клієнтський пристрій (браузер) та сервер застосунку.

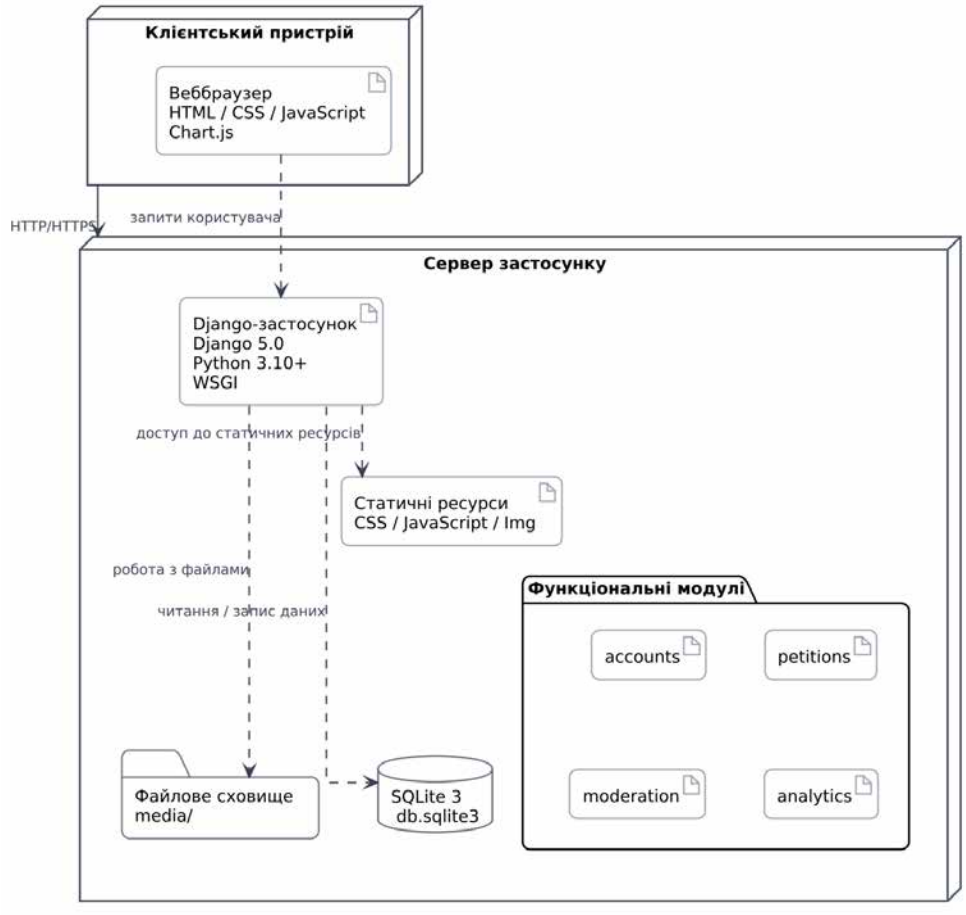

Рис. 2.6 – Діаграма розгортання системи

Клієнтський рівень представлено веббраузером, який завантажує HTML-сторінки, CSS-стилі та JavaScript-скрипти. Бібліотека Chart.js використовується для візуалізації аналітичних даних у дашборді та завантажується через CDN. Серверний рівень включає Django-застосунок із чотирма модулями, базу даних SQLite та файлове сховище для вкладень. Взаємодія між клієнтом та сервером здійснюється через протокол HTTP. Для live-оновлення лічильника підписів використовуються AJAX-запити з інтервалом 30 секунд до API-ендпоінту signatures_count_api.

Маршрутизація URL-адрес реалізована через ієрархічну систему URL-конфігурацій Django. Кореневий файл urls.py розподіляє запити між застосунками за префіксами: accounts/ – автентифікація та профіль, petitions/ – петиції та сповіщення (включаючи кореневий шлях /), moderation/ – черга модерації та офіційні відповіді, analytics/ – дашборд та експорт. Адміністративна панель Django доступна за адресою /admin/ та забезпечує низькорівневий доступ до всіх об’єктів бази даних.

2.4 Проєктування бази даних інформаційної системи

Проєктування бази даних є одним з ключових етапів розробки інформаційної системи, оскільки структура даних безпосередньо впливає на продуктивність, цілісність та розширюваність системи. Інформаційна база системи обробки петицій побудована на реляційній СУБД SQLite 3 та включає десять взаємопов’язаних сутностей.

Логічна модель даних. На першому етапі проєктування створено логічну модель даних, яка відображає сутності предметної області, їхні атрибути та зв’язки між ними. Модель представлено у вигляді ER-діаграми (рис. 2.7).

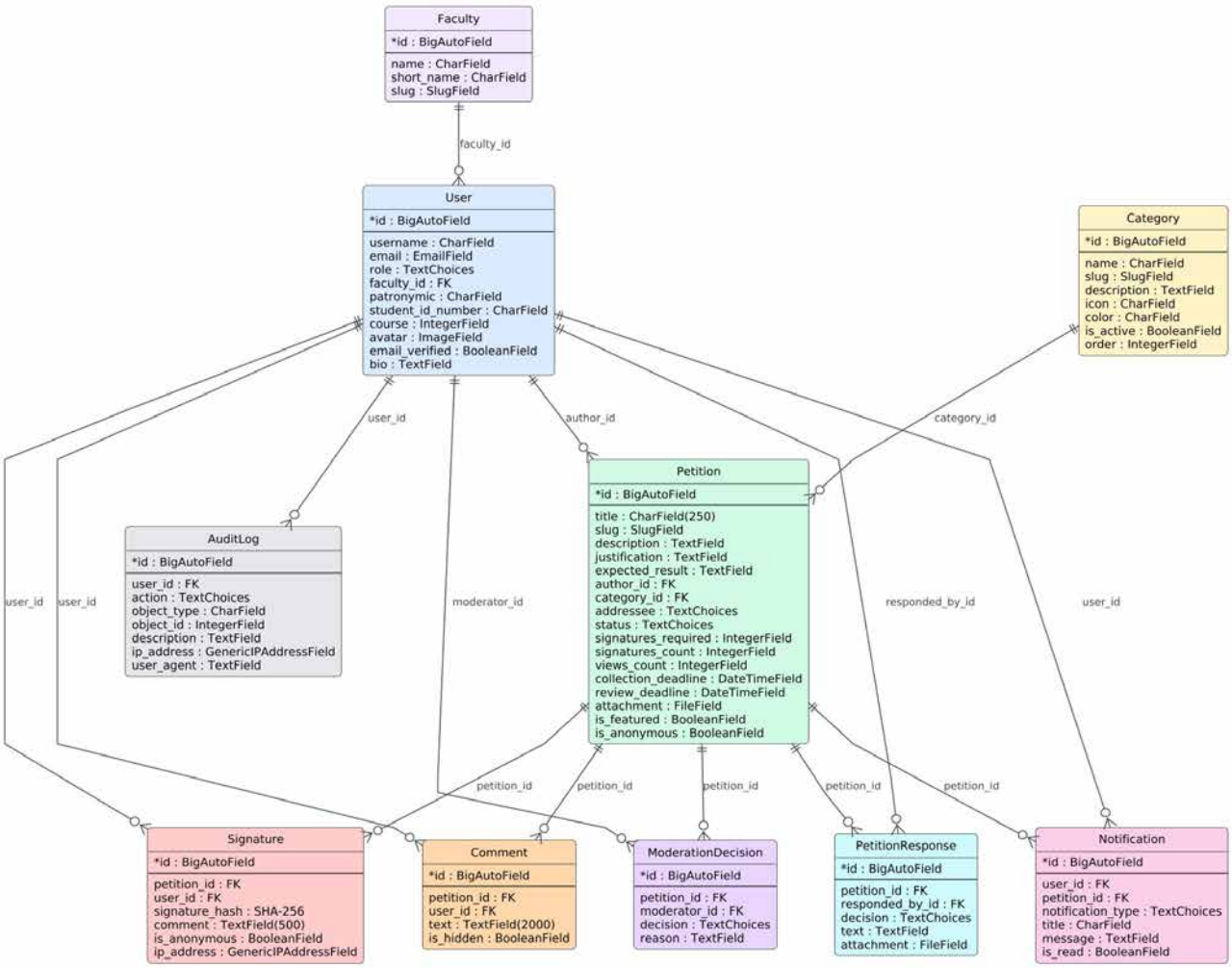

Рис. 2.7 – ER-діаграма інформаційної системи

Центральною сутністю моделі є Petition, яка пов’язана з шістьма іншими сутностями через зовнішні ключі. Сутність User є другою за кількістю зв’язків – вона виступає автором петиції, підписантом, модератором, адміністратором, автором коментаря та суб’єктом запису аудиту.

Детальний опис атрибутів кожної сутності та типів зв’язків подано у таблиці 2.5.

Таблиця 2.5

Опис сутностей інформаційної бази

User	BigAutoField (PK), username, email, role (TextChoices), faculty_id (FK→Faculty), patronymic, student_id_number, course, avatar, email_verified, bio	Розширює AbstractUser. Роль: student, teacher, moderator, admin
Faculty	BigAutoField (PK), name, short_name, slug	Факультет ЗВО для прив’язки користувачів
Category	BigAutoField (PK), name, slug, description, icon, color, is_active, order	10 тематичних категорій з кольоровим маркером
Petition	BigAutoField (PK), title, slug, description, justification, expected_result, author_id (FK→User), category_id (FK→Category), addressee, status (TextChoices, 9 значень), signatures_required, signatures_count, views_count, collection_deadline, review_deadline, attachment, is_featured, is_anonymous	Центральна сутність. Статусна машина з 9 станами, автоматичні дедлайни
Signature	BigAutoField (PK), petition_id (FK→Petition), user_id (FK→User), signature_hash (CharField 64), comment, is_anonymous, ip_address	UC: (petition, user). Хеш SHA-256
ModerationDecision	BigAutoField (PK), petition_id (FK→Petition), moderator_id (FK→User), decision (TextChoices), reason	Три типи рішень: approved, rejected, revision
PetitionResponse	BigAutoField (PK), petition_id (OneToOne→Petition), responded_by_id (FK→User), decision, text, attachment	Офіційна відповідь. OneToOne зв’язок
Comment	BigAutoField (PK), petition_id (FK→Petition), user_id (FK→User), text, is_hidden	Коментарі до петицій з можливістю приховання
Notification	BigAutoField (PK), user_id (FK→User), petition_id (FK→Petition), notification_type (TextChoices, 7 типів), title, message, is_read	Внутрішні сповіщення. 7 типів подій
AuditLog	BigAutoField (PK), user_id (FK→User), action (TextChoices, 10 типів), object_type, object_id, description, ip_address, user_agent	Журнал аудиту. 10 типів дій

Нормалізація. Логічна модель даних відповідає вимогам третьої нормальної форми (3НФ). Перша нормальна форма забезпечується тим, що кожне поле містить атомарне значення – складені поля відсутні, мультизначні атрибути (категорії, статуси) виокремлені у окремі сутності або реалізовані через TextChoices з фіксованим набором допустимих значень. Друга нормальна форма забезпечується тим, що всі неключові атрибути залежать від повного первинного ключа, а не від його частини. Третя нормальна форма гарантується відсутністю транзитивних залежностей: наприклад, інформація про факультет зберігається у окремій сутності Faculty, а не дублюється у User.

Забезпечення цілісності даних. Цілісність даних у розробленій системі забезпечується комплексом обмежень. Обмеження унікальності (unique constraint) на пару полів (petition_id, user_id) у таблиці Signature гарантує, що кожен користувач може підписати конкретну петицію лише один раз. Зв’язок OneToOneField між Petition та PetitionResponse забезпечує наявність не більше однієї офіційної відповіді на кожну петицію. Каскадне видалення (on_delete=CASCADE) для зв’язків Petition→User, Signature→User, Comment→User забезпечує автоматичне видалення залежних записів. Для зв’язків, де видалення пов’язаного об’єкта не повинно призводити до втрати даних, використовується SET_NULL (ModerationDecision→User, PetitionResponse→User, Petition→Category).

Індексація. Для забезпечення продуктивності запитів до бази даних створено індекси на ключових полях. Складений індекс (status, created_at) оптимізує фільтрацію петицій за статусом із сортуванням за датою – це основний запит на головній сторінці. Індекс на полі signatures_count прискорює сортування за популярністю. Складений індекс (author, created_at) використовується для формування профілю автора. Для таблиці Notification створено два індекси: (user, created_at) для ефективного завантаження списку сповіщень та (user, is_read) для швидкого підрахунку непрочитаних повідомлень.

Таблиця 2.6

Індекси бази даних інформаційної системи

Таблиця	Індекс (поля)	Призначення
Petition	(status, –created_at)	Фільтрація за статусом, сортування за датою
Petition	(–signatures_count)	Сортування за популярністю
Petition	(author, –created_at)	Формування профілю автора
Notification	(user, –created_at)	Завантаження списку сповіщень
Notification	(user, is_read)	Підрахунок непрочитаних сповіщень
Signature	(petition, user) UC	Контроль унікальності підпису

Вибір СУБД. Для зберігання даних обрано СУБД SQLite версії 3, яка поставляється у складі стандартної бібліотеки Python і не потребує окремого серверного процесу. SQLite забезпечує атомарність транзакцій через механізм WAL (Write-Ahead Logging), підтримує повний набір SQL-операцій та є рекомендованою СУБД для Django-проєктів з помірним навантаженням. Додатковою перевагою є простота розгортання – база даних зберігається у єдиному файлі db.sqlite3, що спрощує резервне копіювання та міграцію. За потреби масштабування система може бути перенесена на PostgreSQL або MySQL без зміни програмного коду завдяки абстракції Django ORM.

Конфігураційні параметри. Три ключових параметри петиційного процесу винесено у файл налаштувань settings.py та можуть бути змінені адміністратором без модифікації коду: PETITION_SIGNATURES_THRESHOLD (поріг підписів, за замовчуванням 100), PETITION_COLLECTION_DAYS (термін збору, 30 днів), PETITION_REVIEW_DAYS (термін відповіді, 14 днів). Ці параметри використовуються у методах моделі Petition при створенні нової петиції та розрахунку дедлайнів.

Спроєктована база даних з десятьма сутностями, шістьма індексами та комплексом обмежень цілісності забезпечує ефективне зберігання та обробку даних петиційного процесу. Логічна модель відповідає третій нормальній формі, що гарантує відсутність аномалій оновлення, вставки та видалення.

3 РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ ОБРОБКИ УНІВЕРСИТЕТСЬКИХ ПЕТИЦІЙ

3.1 Обґрунтування вибору технологій та засобів розробки

Вибір технологій та інструментальних засобів є відповідальним етапом розробки програмної системи, оскільки він визначає продуктивність розробника, якість кінцевого продукту, можливості подальшого супроводу та масштабування. При обранні технологічного стеку для інформаційної системи обробки університетських петицій враховано такі критерії: наявність вбудованих механізмів безпеки, підтримка шаблону MVT, простота розгортання, якість документації, активність спільноти розробників та відкритість ліцензій [16].

Мова програмування Python 3.10+. Python є однією з найпоширеніших мов програмування у світі. Згідно з індексом TIOBE, Python стабільно займає першу позицію у рейтингу популярності мов програмування. Python характеризується чистим та зрозумілим синтаксисом, що сприяє читабельності коду та зменшує витрати на супровід. Мова підтримує динамічну типізацію з можливістю анотацій типів (type hints), автоматичне управління пам’яттю через збирач сміття (garbage collector), багатопарадигменне програмування (об’єктно-орієнтоване, функціональне, процедурне). Стандартна бібліотека Python включає модулі для роботи з хешуванням (hashlib, використовується для генерації SHA-256 підписів), дата-часом (datetime, timezone), регулярними виразами (re), серіалізацією (json, csv) та іншими операціями, що потрібні у розроблюваній системі. Версія 3.10 обрана як мінімальна через підтримку оператора match/case та покращену діагностику помилок [17].

Вебфреймворк Django 5.0. Django – це високорівневий вебфреймворк з відкритим вихідним кодом, що розповсюджується за ліцензією BSD-3-Clause. Фреймворк реалізує архітектурний шаблон MVT (Model – View – Template) та

дотримується принципу DRY (Don’t Repeat Yourself). Для розроблюваної системи ключовими перевагами Django є такі вбудовані компоненти.

Система автентифікації та авторизації. Django надає повнофункціональний модуль django.contrib.auth, який включає модель користувача з можливістю розширення через AbstractUser, хешування паролів за алгоритмом PBKDF2 із унікальною сіллю, систему дозволів (permissions) та груп, декоратори для контролю доступу (@login_required), сигнали для відстеження входу та виходу (user_logged_in, user_logged_out). У розробленій системі стандартну модель користувача розширено додатковими полями: роль (student, teacher, moderator, admin), факультет, курс, номер студентського квитка [18].

ORM (Object-Relational Mapping). Django ORM забезпечує декларативний опис моделей даних через класи Python, автоматичну генерацію SQL-запитів із захистом від SQL-ін’єкцій (параметризовані запити), систему міграцій для версіонування схеми бази даних, підтримку складних запитів через Q-об’єкти та анотації (annotate, aggregate). ORM абстрагує взаємодію з базою даних, що дозволяє змінити СУБД без модифікації прикладного коду [19].

Шаблонізатор Django Templates. Вбудований шаблонізатор підтримує успадкування шаблонів ({% extends %}), включення фрагментів ({% include %}), контекстні процесори для глобальних змінних, автоматичне екранування HTML-виводу для запобігання XSS-атакам, фільтри для форматування даних (date, truncatewords, intcomma). У системі петицій використовується ієрархія шаблонів з базовим шаблоном base.html, від якого успадковуються всі сторінки.

Адміністративна панель Django Admin. Django надає автоматично генеровану адміністративну панель з можливістю налаштування відображення моделей через класи ModelAdmin. У розробленій системі панель налаштована для всіх моделей з фільтрацією, пошуком, редагуванням та ієрархічною навігацією за датою [20].

CSRF-захист. Вбудований middleware CsrfViewMiddleware автоматично генерує та перевіряє CSRF-токени для всіх POST-запитів, що захищає від атак

міжсайтової підробки запитів. Кожна форма у шаблонах системи включає тег {% csrf_token %} [21].

Порівняння Django з альтернативними фреймворками за ключовими критеріями представлено у таблиці 3.1.

Таблиця 3.1

Порівняння вебфреймворків за ключовими критеріями

Критерій	Django 5.0	Flask 3.x	Express.js	Ruby on Rails
Вбудована автентифікація	Так (повна)	Ні (Flask-Login)	Ні (express-session)	Так (Devise)
ORM	Вбудований	SQLAlchemy (окремо)	Sequelize (окремо)	Active Record
Адміністративна панель	Вбудована	Flask-Admin	Відсутня	ActiveAdmin
CSRF-захист	Вбудований	Flask-WTF	csurf (окремо)	Вбудований
Шаблонізатор	Вбудований	Jinja2 (окремо)	EJS/Pug (окремо)	ERB (вбуд.)
Міграції БД	Вбудовані	Flask-Migrate	Sequelize CLI	Вбудовані
Ліцензія	BSD-3-Clause	BSD-3-Clause	MIT	MIT
Мова	Python	Python	JavaScript	Ruby

Як видно з таблиці 3.1, Django є єдиним фреймворком, що надає повний набір вбудованих компонентів для розробки вебзастосунку без необхідності встановлення додаткових бібліотек. Flask та Express.js потребують окремого підключення кожного компонента, що збільшує складність проєкту та ризик несумісності версій [22].

СУБД SQLite 3. SQLite – це вбудована реляційна СУБД, яка зберігає всю базу даних у єдиному файлі. SQLite поставляється у складі стандартної бібліотеки Python (модуль sqlite3) та не потребує встановлення окремого серверного процесу. Ключові характеристики: повна підтримка стандарту SQL з транзакціями (ACID), механізм WAL (Write-Ahead Logging) для покращення паралельного читання, підтримка індексів (B-tree), обмежень зовнішніх ключів

та тригерів. SQLite рекомендована офіційною документацією Django для розробки та проєктів із помірним навантаженням. У контексті університетської системи з очікуваною кількістю користувачів до 20 000 та помірною інтенсивністю запитів SQLite забезпечує достатню продуктивність. За потреби масштабування система може бути перенесена на PostgreSQL або MySQL без зміни прикладного коду завдяки абстракції Django ORM [23].

Бібліотека Chart.js 4.x. Chart.js – це JavaScript-бібліотека для створення інтерактивних діаграм, що розповсюджується за ліцензією MIT. Бібліотека підтримує вісім типів діаграм (лінійні, стовпчикові, кільцеві, радарні та інші), анімацію при завантаженні, адаптивне масштабування під розмір контейнера, підказки при наведенні курсора. У розробленій системі Chart.js використовується в аналітичному дашборді для візуалізації чотирьох графіків: динаміка створення петицій за 30 днів (лінійна діаграма), динаміка підписів за 30 днів (стовпчикова), розподіл за статусами (кільцева), розподіл за категоріями (горизонтальна стовпчикова). Бібліотека завантажується через CDN, що забезпечує кешування та мінімізує розмір серверних ресурсів [24].

Бібліотека Pillow. Pillow – це бібліотека для обробки зображень у Python, що розповсюджується за ліцензією HPND (Historical Permission Notice and Disclaimer). У системі Pillow використовується для обробки аватарів користувачів – валідації формату, зміни розміру та збереження у файловому сховищі. Бібліотека є стандартною залежністю для Django-проєктів, що використовують ImageField.

Шрифти Manrope та JetBrains Mono. Для оформлення інтерфейсу використано два шрифти з відкритими ліцензіями. Manrope – гротескний шрифт (SIL Open Font License 1.1), що використовується для основного тексту інтерфейсу. JetBrains Mono – моноширинний шрифт (SIL Open Font License 1.1), що використовується для відображення технічної інформації (хеш-значення підписів, ідентифікатори). Обидва шрифти завантажуються через Google Fonts CDN.

Стилізація інтерфейсу. CSS-стилізація реалізована через єдиний кастомний файл main.css обсягом 1499 рядків без залежності від зовнішніх CSS-фреймворків (Bootstrap, Tailwind). Такий підхід забезпечує повний контроль над дизайном, мінімальний розмір завантажуваних ресурсів та відсутність невикористовуваного CSS-коду. Стилі побудовані з використанням CSS Custom Properties (змінних) для визначення колірної палітри, що спрощує зміну теми оформлення.

Зведену інформацію про технологічний стек розробленої системи подано у таблиці 3.2.

Таблиця 3.2

Технологічний стек інформаційної системи

Компонент	Технологія / Інструмент	Ліцензія
Мова програмування	Python 3.10+	PSF License
Вебфреймворк	Django 5.0	BSD-3-Clause
СУБД	SQLite 3	Public Domain
Шаблонізатор	Django Templates	BSD-3-Clause
Візуалізація	Chart.js 4.x	MIT
Обробка зображень	Pillow	HPND
Основний шрифт	Manrope	SIL OFL 1.1
Моноширинний шрифт	JetBrains Mono	SIL OFL 1.1
CSS-стилізація	Кастомний main.css	–
Середовище розробки	VS Code / PyCharm	–
Контроль версій	Git	GPL-2.0

Усі компоненти технологічного стеку є програмним забезпеченням з відкритим вихідним кодом та вільними ліцензіями, що дозволяє їх використання в освітніх та комерційних проєктах без обмежень. Обраний стек

забезпечує повний набір інструментів для реалізації усіх функціональних вимог, визначених у підрозділі 2.1.

3.2 Реалізація серверної частини вебзастосунку

Серверна частина вебзастосунку реалізована у відповідності до архітектурного шаблону MVT та включає чотири Django-застосунки: accounts, petitions, moderation, analytics. У цьому підрозділі розглянуто реалізацію ключових компонентів серверної частини – моделей даних, представлень, форм, middleware та системи маршрутизації.

Реалізація моделей даних. Моделі Django [25] визначають структуру бази даних через декларативний опис полів та зв'язків. Центральною моделлю системи є Petition, яка реалізує статусну машину з дев'ятьма станами. Статуси та адресати визначено через вкладені класи TextChoices, що забезпечує фіксований набір допустимих значень на рівні моделі.

Лістинг 3.1 – Визначення статусів та адресатів моделі Petition

```
class Petition(models.Model):
    class Status(models.TextChoices):
        DRAFT = 'draft', _('Чернетка')
        MODERATION = 'moderation', _('На модерації')
        REJECTED = 'rejected', _('Відхилено модератором')
        ACTIVE = 'active', _('Збір підписів')
        REVIEW = 'review', _('На розгляді адміністрацією')
        APPROVED = 'approved', _('Підтримано')
        PARTIAL = 'partial', _('Підтримано частково')
        DECLINED = 'declined', _('Відхилено')
        EXPIRED = 'expired', _('Термін збору сплинув')


    class Addressee(models.TextChoices):
        RECTORATE = 'rectorate', _('Ректорат')
        DEAN = 'dean', _('Деканат факультету')
        DEPARTMENT = 'department', _('Кафедра')
        STUDENT_COUNCIL = 'student_council', _('Студ. самоврядування')
```

```
        OTHER = 'other', _('Інше')
```

У лістингу 3.1 показано визначення дев’яти статусів петиції та п’яти типів адресатів через вкладені класи TextChoices. Функція _() забезпечує інтернаціоналізацію міток для можливої багатомовної підтримки. Кожен елемент TextChoices [26] визначає пару «значення у БД – людиночитний відображуваний текст».

Ключовим елементом бізнес-логіки є метод update_signatures_count(), який автоматично оновлює лічильник підписів та ініціює перехід петиції у статус офіційного розгляду при досягненні порогового значення.

Лістинг 3.2 – Метод автоматичного оновлення лічильника підписів

```
def update_signatures_count(self):
    count = self.signatures.count()
    self.signatures_count = count
    if (count >= self.signatures_required
            and self.status == self.Status.ACTIVE
            and not self.threshold_reached_at):
        self.status = self.Status.REVIEW
        self.threshold_reached_at = timezone.now()
        self.review_deadline = timezone.now() + timedelta(
            days=settings.PETITION_REVIEW_DAYS
        )
    self.save(update_fields=['signatures_count', 'status',
                             'threshold_reached_at', 'review_deadline'])
```

Метод update_signatures_count() (лістинг 3.2) реалізує автоматичний перехід між станами статусної машини. При кожному підписанні або знятті підпису метод підраховує актуальну кількість підписів через агрегатний запит self.signatures.count(), порівнює з порогом signatures_required та, за умови досягнення порога, змінює статус з active на review. Одночасно фіксується часова мітка threshold_reached_at та розраховується дедлайн надання офіційної відповіді. Метод save() викликається з параметром update_fields для оновлення лише змінених полів, що підвищує продуктивність.

Модель Signature забезпечує криптографічне підтвердження кожного підпису через генерацію хеш-значення SHA-256 [27].

Лістинг 3.3 – Генерація хеш-значення підпису та обмеження унікальності

```
class Signature(models.Model):
    petition = models.ForeignKey(Petition, on_delete=models.CASCADE,
                                 related_name='signatures')
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                             on_delete=models.CASCADE)
    signature_hash = models.CharField(max_length=64, blank=True)
    is_anonymous = models.BooleanField(default=False)
    ip_address = models.GenericIPAddressField(null=True, blank=True)


    class Meta:
        constraints = [
            models.UniqueConstraint(
                fields=['petition', 'user'],
                name='unique_user_petition_signature'
            )
        ]


    def save(self, *args, **kwargs):
        if not self.signature_hash:
            data = f'{self.user_id}-{self.petition_id}-'
                   f'{timezone.now().isoformat()}'
            self.signature_hash = hashlib.sha256(
                data.encode()).hexdigest()
        super().save(*args, **kwargs)
```

У лістингу 3.3 представлено модель Signature з двома механізмами захисту. По-перше, обмеження UniqueConstraint на пару полів (petition, user) гарантує на рівні бази даних, що кожен користувач може підписати конкретну петицію лише один раз. По-друге, при збереженні автоматично генерується хеш SHA-256, що включає ідентифікатори користувача та петиції, а також часову мітку. Це забезпечує криптографічне підтвердження кожного підпису та унеможливлює його підробку.

Реалізація представлень (Views). Представлення обробляють HTTP-запити, виконують валідацію та бізнес-логіку, формують контекст для шаблонів. У системі реалізовано 15 представлень, розподілених між чотирма застосунками. Розглянемо реалізацію ключових представлень.

Представлення підписання петиції є одним із центральних елементів системи, оскільки воно поєднує валідацію, створення об’єкту, оновлення лічильника, журналювання та генерацію сповіщень.

Лістинг 3.4 – Представлення підписання петиції (фрагмент)

```
@login_required
@require_POST
def sign_view(request, slug):
    petition = get_object_or_404(Petition, slug=slug)
    can_sign, message = petition.can_be_signed_by(request.user)
    if not can_sign:
        messages.error(request, message)
        return redirect(petition.get_absolute_url())


    form = SignatureForm(request.POST)
    if form.is_valid():
        signature = form.save(commit=False)
        signature.petition = petition
        signature.user = request.user
        signature.ip_address = getattr(request, 'client_ip', None)
        signature.save()
        petition.update_signatures_count()


        AuditLog.objects.create(
            user=request.user,
            action=AuditLog.Action.SIGN,
            object_type='Petition',
            object_id=str(petition.id),
            description=f'Підписано петицію: {petition.title}',
            ip_address=getattr(request, 'client_ip', None),
        )
```

Представлення sign_view (лістинг 3.4) захищене двома декораторами: @login_required забезпечує доступ лише для автентифікованих користувачів, @require_POST обмежує метод HTTP виключно POST-запитами, що відповідає принципу безпеки. Метод can_be_signed_by() виконує чотири перевірки: автентифікація, активність збору, заборона самопідписання та відсутність попереднього підпису. Після успішного збереження підпису викликається update_signatures_count() для перевірки порогу, створюється запис у журналі аудиту з фіксацією IP-адреси та генерується сповіщення для автора петиції.

Представлення модерації реалізує логіку прийняття рішень модератором з автоматичним виявленням потенційних дублікатів.

Лістинг 3.5 – Представлення модерації петиції (фрагмент)

```
@moderator_required
def review_view(request, slug):
    petition = get_object_or_404(Petition, slug=slug)
    duplicates = Petition.objects.filter(
        title__icontains=petition.title[:30],
        status__in=[Petition.Status.ACTIVE,
                    Petition.Status.REVIEW,
                    Petition.Status.APPROVED]
    ).exclude(pk=petition.pk)[:5]

    if request.method == 'POST':
        form = ModerationDecisionForm(request.POST)
        if form.is_valid():
            decision = form.save(commit=False)
            decision.petition = petition
            decision.moderator = request.user
            decision.save()
            if decision.decision == 'approved':
                petition.status = Petition.Status.ACTIVE
                petition.published_at = timezone.now()
                petition.collection_deadline = timezone.now()
                    + timedelta(days=settings.PETITION_COLLECTION_DAYS)
                petition.save()
```

У лістингу 3.5 показано механізм автоматичного виявлення дублікатів. Система виконує пошук серед активних, розглядуваних та підтриманих петицій за збігом перших 30 символів заголовка (title__icontains). Знайдені потенційні дублікати передаються у контекст шаблону, де модератор може їх переглянути перед прийняттям рішення. При схваленні петиції оновлюється статус, фіксується дата публікації та розраховується дедлайн збору підписів на основі конфігураційного параметра PETITION_COLLECTION_DAYS.

Реалізація форм та валідації. Форми Django забезпечують генерацію HTML-елементів, валідацію вхідних даних та зв’язок з моделями. У системі реалізовано шість форм: PetitionForm, SignatureForm, CommentForm, PetitionFilterForm, ModerationDecisionForm, ResponseForm. Кожна форма використовує міксін StyledFormMixin для уніфікованого додавання CSS-класів до полів.

Лістинг 3.6 – Форма створення петиції з валідацією (фрагмент)

```
class PetitionForm(StyledFormMixin, forms.ModelForm):
    class Meta:
        model = Petition
        fields = ['title', 'category', 'addressee',
                  'description', 'justification',
                  'expected_result', 'attachment', 'is_anonymous']


    def clean_title(self):
        title = self.cleaned_data.get('title', '').strip()
        if len(title) < 10:
            raise forms.ValidationError(
                'Заголовок повинен містити не менше 10 символів.')
        return title


    def clean_description(self):
        desc = self.cleaned_data.get('description', '').strip()
        if len(desc) < 50:
            raise forms.ValidationError(
                'Опис повинен містити не менше 50 символів.')
        return desc
```

PetitionForm (лістинг 3.6) є багатосекційною формою з вісьмома полями, що відповідає вимозі FR-03. Методи clean_title() та clean_description() реалізують серверну валідацію мінімальної довжини, що запобігає поданню петицій з неінформативним змістом. Валідація виконується на стороні сервера, незалежно від клієнтських перевірок, що забезпечує надійність.

Форма рішення модератора включає крос-валідацію – перевірку залежності між полями.

Лістинг 3.7 – Крос-валідація у формі модератора

```
class ModerationDecisionForm(StyledFormMixin, forms.ModelForm):
    class Meta:
        model = ModerationDecision
        fields = ['decision', 'reason']


    def clean(self):
        cleaned = super().clean()
        decision = cleaned.get('decision')
        reason = (cleaned.get('reason') or '').strip()
        if decision in ['rejected', 'revision'] and not reason:
            raise forms.ValidationError(
                'Для відхилення або повернення необхідно'
                ' вказати причину.')
        return cleaned
```

Метод clean() у лістингу 3.7 реалізує бізнес-правило: при відхиленні або поверненні петиції модератор зобов’язаний зазначити причину. Це забезпечує прозорість модераційних рішень та надає автору петиції зворотний зв’язок для доопрацювання.

Організація URL-маршрутизації. Маршрутизація запитів реалізована через ієрархічну систему URL-конфігурацій Django. Кореневий файл urls.py розподіляє запити між застосунками за префіксами шляху. Кожен застосунок

визначає власний простір імен (namespace), що запобігає конфліктам імен між маршрутами.

Лістинг 3.8 – Кореневий файл URL-маршрутизації

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('accounts.urls',
                              namespace='accounts')),
    path('moderation/', include('moderation.urls',
                                namespace='moderation')),
    path('analytics/', include('analytics.urls',
                               namespace='analytics')),
    path('', include('petitions.urls',
                     namespace='petitions')),
]
```

У лістингу 3.8 функція include() делегує обробку запитів відповідним застосункам. Застосунок petitions підключено без префіксу (порожній рядок), що робить список петицій доступним за кореневою адресою /. Адміністративна панель Django доступна за адресою /admin/. Використання просторів імен (namespace) дозволяє генерувати URL-адреси у шаблонах через синтаксис {% url 'petitions:detail' slug=p.slug %}.

Застосунок petitions визначає одинадцять маршрутів, що покривають усі операції з петиціями.

Таблиця 3.3

URL-маршрути інформаційної системи

Метод	**URL-шаблон**	**Ім’я**	**Призначення**
GET	/	petitions:list	Список петицій з фільтрацією та пагінацією
GET	/create/	petitions:create	Форма створення петиції
GET	/p/<slug>/	petitions:detail	Детальний перегляд петиції
GET	/p/<slug>/edit/	petitions:edit	Редагування чернетки
POST	/p/<slug>/sign/	petitions:sign	Підписання петиції

POST	/p/<slug>/unsign/	petitions:unsign	Зняття підпису
POST	/p/<slug>/comment/	petitions:comment	Додавання коментаря
GET	/p/<slug>/api/signatures/	–	API: лічильник підписів (JSON)

Продовження таблиці 3.3

GET	/notifications/	petitions:notifications	Список сповіщень
GET	/moderation/	moderation:queue	Черга модерації
POST	/moderation/p/<slug>/	moderation:review	Рішення модератора
POST	/moderation/p/<slug>/respond/	moderation:respond	Офіційна відповідь
GET	/analytics/	analytics:dashboard	Аналітичний дашборд
GET	/analytics/export/csv/	analytics:export_csv	Експорт даних CSV

Middleware та контекстні процесори. Middleware – це компоненти, що обробляють кожен HTTP-запит та відповідь. У системі, окрім стандартних middleware Django (SecurityMiddleware, SessionMiddleware, CsrfViewMiddleware, AuthenticationMiddleware), реалізовано кастомний AuditLogMiddleware, який зберігає IP-адресу клієнта в атрибуті request.client_ip для подальшого використання у журналі аудиту [29].

Лістинг 3.9 – Контекстний процесор глобальної статистики

```
def global_stats(request):
    context = {
        'global_petitions_count': Petition.objects.exclude(
            status__in=[Petition.Status.DRAFT,
                        Petition.Status.MODERATION,
                        Petition.Status.REJECTED]
        ).count(),
    }
    if request.user.is_authenticated:
        context['unread_notifications_count'] = (
            Notification.objects.filter(
                user=request.user, is_read=False
            ).count())
    else:
```

```
        context['unread_notifications_count'] = 0
    return context
```

Контекстний процесор global_stats (лістинг 3.9) додає до контексту кожного шаблону дві змінні: загальну кількість опублікованих петицій (для відображення у підвалі сторінки) та кількість непрочитаних сповіщень поточного користувача (для індикатора у навігаційній панелі). Процесор зареєстровано у файлі settings.py у секції TEMPLATES.

Реалізація аналітичного модуля. Аналітичний модуль реалізує обчислення KPI-метрик та агрегацію даних для візуалізації. Доступ до дашборда обмежено декоратором @admin_required, який перевіряє роль користувача.

Лістинг 3.10 – Обчислення KPI та агрегація для графіків (фрагмент)

```
@admin_required
def dashboard_view(request):
    now = timezone.now()
    month_ago = now - timedelta(days=30)

    total_petitions = Petition.objects.exclude(
        status__in=[Petition.Status.DRAFT,
                    Petition.Status.REJECTED]).count()
    active_petitions = Petition.objects.filter(
        status=Petition.Status.ACTIVE).count()

    petitions_by_day = (
        Petition.objects.filter(created_at__gte=month_ago)
        .annotate(day=TruncDate('created_at'))
        .values('day')
        .annotate(count=Count('id'))
        .order_by('day')
    )
```

Представлення dashboard_view (лістинг 3.10) обчислює десять KPI-метрик та формує чотири набори даних для графіків. Функція TruncDate()

групує записи за датою, а Count() виконує агрегацію. Результати передаються у контекст шаблону як словники з ключами labels та data, які безпосередньо використовуються бібліотекою Chart.js на клієнтській стороні.

Модуль експорту реалізує вивантаження переліку петицій у форматі CSV з коректною підтримкою кирилиці через додавання BOM-маркера (Byte Order Mark) на початку файлу, що забезпечує правильне відображення у Microsoft Excel.

Налаштування адміністративної панелі. Адміністративна панель Django налаштована для всіх моделей системи через класи ModelAdmin. Для моделі Petition визначено п’ять груп полів (fieldsets): основне, зміст, параметри збору, прапорці, часові мітки. Реалізовано фільтрацію за статусом, категорією, адресатом та датою; пошук за заголовком, описом та іменем автора; можливість редагування прапорця is_featured безпосередньо у списку записів. Часові мітки (created_at, published_at, threshold_reached_at) позначено як readonly_fields, що запобігає їх ручній зміні.

Серверна частина вебзастосунку реалізує повну бізнес-логіку інформаційної системи обробки петицій. Чотири Django-застосунки містять 10 моделей, 15 представлень, 6 форм із серверною валідацією, кастомний middleware для журналювання та контекстний процесор для глобальної статистики. Кожна операція фіксується у журналі аудиту з деталями дії, IP-адресою та User-Agent клієнта [30]. Система маршрутизації визначає 14 URL-маршрутів, розподілених між чотирма просторами імен. Код дотримується конвенцій Django та документований через docstrings.

3.3 Реалізація клієнтської частини та користувацького інтерфейсу

Клієнтська частина вебзастосунку реалізована засобами Django Templates, кастомного CSS та мінімального обсягу JavaScript. Такий підхід забезпечує серверний рендеринг HTML-сторінок, що гарантує коректну індексацію пошуковими системами та працездатність без JavaScript для базових операцій.

Система шаблонів. Шаблони організовано за ієрархічним принципом із використанням механізму успадкування Django Templates. Базовий шаблон base.html визначає спільну структуру сторінки: навігаційну панель, контейнер для основного контенту, підвал із посиланнями та лічильником петицій. Кожен застосунок має власну директорію шаблонів з HTML-файлами, що успадковують base.html через тег {% extends "base.html" %}.

Структуру шаблонів системи представлено у таблиці 3.4.

Таблиця 3.4

Структура шаблонів інформаційної системи

Шаблон	Директорія	Призначення
base.html	templates/	Базовий шаблон: навігація, footer, підключення CSS/JS, блоки content та extra_js
list.html	petitions/	Список петицій: фільтри, пагінація, картки петицій, бічна панель з трендами
detail.html	petitions/	Детальний перегляд: опис, прогрес-бар, підписи, коментарі, офіційна відповідь
create.html	petitions/	Форма створення/редагування: багатосекційна форма з кнопками «Чернетка» та «Подати»
notifications.html	petitions/	Список сповіщень з індикатором прочитаності та посиланнями на петиції
queue.html	moderation/	Черга модерації: список петицій на перевірку, статистика модератора за день
review.html	moderation/	Перегляд петиції модератором: деталі, дублікати, форма рішення, історія
respond.html	moderation/	Форма офіційної відповіді: рішення, текст, вкладення
review_list.html	moderation/	Список петицій, що очікують офіційної відповіді
dashboard.html	analytics/	Аналітичний дашборд: KPI-картки, 4 графіки Chart.js, топ-петиції, факультети
login.html	accounts/	Сторінка входу із формою автентифікації
register.html	accounts/	Реєстрація із прив’язкою до факультету та курсу
profile.html	accounts/	Особистий кабінет: статистика, петиції, підписи, вкладки

CSS-стилізація. Візуальне оформлення реалізовано через єдиний кастомний файл main.css (1499 рядків) без залежності від зовнішніх фреймворків. Колірна палітра побудована на CSS Custom Properties (змінних), що визначаються у псевдокласі :root. Це забезпечує централізоване управління темою та можливість зміни кольорів в одному місці [31].

Лістинг 3.11 – CSS-змінні колірної палітри (фрагмент :root)

```
:root {
    --color-bg: #FAFAF7;
    --color-bg-alt: #FFFFFF;
    --color-text: #1A1A1A;
    --color-text-muted: #6B6B66;
    --color-primary: #2D5BFF;
    --color-success: #16A34A;
    --color-danger: #DC2626;


    --color-status-active: #16A34A;
    --color-status-review: #2D5BFF;
    --color-status-approved: #059669;
    --color-status-declined: #B91C1C;


    --font-sans: 'Manrope', sans-serif;
    --font-mono: 'JetBrains Mono', monospace;
    --radius-lg: 12px;
    --shadow-md: 0 2px 8px rgba(0,0,0,0.06);
    --container-max: 1200px;
}
```

У лістингу 3.11 показано визначення змінних для кольорів фону, тексту, статусів петицій, шрифтів та геометричних параметрів. Для кожного з дев’яти статусів петиції визначено окремий колір (--color-status-*), який автоматично застосовується до відповідних CSS-класів (.status-active, .status-review тощо).

Адаптивний дизайн. Для забезпечення коректного відображення на різних пристроях використано CSS Grid та Flexbox із медіа-запитами. Система визначає три точки перелому: 900px (перехід двоколонкового макету detail-layout на одноколонковий), 800px (перебудова графіків дашборда та фільтрів),

700px (спрощення навігаційної панелі). Карткова сітка списку петицій побудована на grid-template-columns: repeat(auto-fill, minmax(340px, 1fr)), що автоматично адаптує кількість колонок під ширину екрана [32].

Лістинг 3.12 – Адаптивна сітка списку петицій та медіа-запит

```
.petitions-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill,
                                   minmax(340px, 1fr));
    gap: 20px;
}

@media (max-width: 900px) {
    .detail-layout {
        grid-template-columns: 1fr;
    }
}
```

У лістингу 3.12 функція minmax(340px, 1fr) гарантує, що кожна картка петиції має мінімальну ширину 340 пікселів. На широких екранах відображається 3 колонки, на планшетах – 2, на мобільних – 1, без явного визначення кількості колонок у кожному медіа-запиті.

Прогрес-бар збору підписів. На картці кожної петиції та на сторінці детального перегляду відображається індикатор прогресу збору підписів. Стрічка прогресу реалізована як вкладені div-елементи з анімованим переходом ширини (transition: width 0.4s cubic-bezier). Поточна кількість підписів та порогове значення відображаються у верхній частині блоку. При досягненні 100% підписів стрічка змінює колір з темного на зелений через CSS-клас .success.

JavaScript-компоненти. Клієнтська частина використовує мінімальний обсяг JavaScript для двох задач. По-перше, live-оновлення лічильника підписів через AJAX-запити до ендпоінту /p/<slug>/api/signatures/ з інтервалом 30 секунд

(функція setInterval). Відповідь сервера повертається у форматі JSON із полями count, required, progress та status, що дозволяє оновити прогрес-бар та числові показники без перезавантаження сторінки. По-друге, ініціалізація графіків Chart.js на сторінці аналітичного дашборда [33].

Лістинг 3.13 – AJAX-оновлення лічильника підписів

```
function updateSignaturesCount() {
    fetch('{% url "petitions:signatures_count_api"
                  slug=petition.slug %}')
    .then(response => response.json())
    .then(data => {
        document.querySelector('.progress-current')
            .textContent = data.count;
        document.querySelector('.progress-fill')
            .style.width = data.progress + '%';
        if (data.progress >= 100) {
            document.querySelector('.progress-fill')
                .classList.add('success');
        }
    });
}
setInterval(updateSignaturesCount, 30000);
```

У лістингу 3.13 показано AJAX-функцію, яка кожні 30 секунд запитує актуальну кількість підписів через API-ендпоінт та оновлює DOM-елементи без перезавантаження сторінки. Це дозволяє користувачам бачити зростання підтримки петиції у реальному часі.

Візуалізація аналітики. На сторінці дашборда реалізовано чотири графіки через Chart.js: лінійна діаграма динаміки створення петицій за останні 30 днів, стовпчикова діаграма динаміки підписів, кільцева діаграма розподілу за статусами та горизонтальна стовпчикова діаграма розподілу за категоріями. Дані для графіків передаються з серверного представлення dashboard_view через контекст шаблону у форматі JSON (списки labels та data). KPI-метрики

відображаються у вигляді восьми карток (stat-card) із значеннями: загальна кількість петицій, активні, на розгляді, підтримані, загальна кількість підписів та користувачів, нові за тиждень, відсоток успішності.

Лістинг 3.14 – Ініціалізація кільцевої діаграми розподілу за статусами

```
new Chart(document.getElementById('statusChart'), {
    type: 'doughnut',
    data: {
        labels: {{ status_chart.labels|safe }},
        datasets: [{
            data: {{ status_chart.data|safe }},
            borderWidth: 2,
            borderColor: '#FAFAF7',
        }]
    },
    options: {
        responsive: true,
        plugins: { legend: { position: 'bottom' } }
    }
});
```

У лістингу 3.14 дані для діаграми передаються через шаблонні теги Django {{ status_chart.labels|safe }}. Фільтр |safe вимикає автоматичне екранування, оскільки дані є JSON-масивами, сформованими на сервері. Параметр responsive: true забезпечує адаптацію діаграми під розмір контейнера.

Навігаційна панель та сповіщення. Навігаційна панель відображається на всіх сторінках та містить логотип системи, посилання на список петицій, створення петиції, та іконку сповіщень із лічильником непрочитаних повідомлень (badge). Для модераторів додатково відображаються посилання на чергу модерації та аналітичний дашборд. Лічильник непрочитаних сповіщень формується контекстним процесором global_stats та оновлюється при кожному завантаженні сторінки [34].

Клієнтська частина системи побудована за принципом прогресивного покращення (progressive enhancement): базова функціональність працює без JavaScript, а AJAX-оновлення та інтерактивні графіки доповнюють досвід користувача. Загальний обсяг клієнтського коду складає 1499 рядків CSS та менше 100 рядків JavaScript, що забезпечує швидке завантаження та мінімальний розмір ресурсів.

3.4 Реалізація модулів створення, підписання, модерації та розгляду петицій

У цьому підрозділі детально розглянуто реалізацію чотирьох ключових функціональних модулів системи, що забезпечують повний життєвий цикл петиції від створення до отримання офіційної відповіді.

Модуль створення петиції. Процес створення петиції реалізовано через представлення create_view та форму PetitionForm. Система підтримує два режими збереження: чернетка (draft) та подання на модерацію (moderation). Режим визначається значенням прихованого поля action у POST-запиті.

Лістинг 3.15 – Логіка створення петиції з вибором режиму збереження

```
@login_required
def create_view(request):
    if request.method == 'POST':
        form = PetitionForm(request.POST, request.FILES)
        if form.is_valid():
            petition = form.save(commit=False)
            petition.author = request.user
            action = request.POST.get('action', 'submit')
            if action == 'draft':
                petition.status = Petition.Status.DRAFT
            else:
                petition.status = Petition.Status.MODERATION
                petition.submitted_at = timezone.now()
            petition.save()
            AuditLog.objects.create(
                user=request.user,
```

```
                action=AuditLog.Action.CREATE,
                object_type='Petition',
                object_id=str(petition.id),
                description=f'Створено петицію: {petition.title}',
                ip_address=getattr(request,
                                   'client_ip', None),
            )
```

У лістингу 3.15 параметр commit=False дозволяє встановити додаткові поля (author, status, submitted_at) перед збереженням у базу даних. При поданні на модерацію фіксується часова мітка submitted_at, яка використовується для сортування черги модерації за принципом FIFO (First In, First Out). Кожне створення петиції фіксується у журналі аудиту [35].

Редагування петиції доступне лише для чернеток та відхилених петицій. Представлення edit_view перевіряє авторство через порівняння petition.author_id з request.user.id та статус через обмеження на множину {DRAFT, REJECTED}. При повторному поданні відхиленої петиції на модерацію статус змінюється з rejected на moderation.

Форма PetitionForm забезпечує генерацію slug через функцію транслітерації, яка перетворює українські символи у латиницю для формування URL-адреси петиції.

Лістинг 3.16 – Функція транслітерації для генерації slug

```
def transliterate(text):
    table = {
        'а': 'a', 'б': 'b', 'в': 'v', 'г': 'h',
        'ґ': 'g', 'д': 'd', 'е': 'e', 'є': 'ie',
        'ж': 'zh', 'з': 'z', 'и': 'y', 'і': 'i',
        'ї': 'i', 'к': 'k', 'л': 'l', 'м': 'm',
        ...
    }
    result = []
    for char in text.lower():
        result.append(table.get(char, char))
    return ''.join(result)
```

Функція transliterate() (лістинг 3.16) реалізує посимвольну заміну кириличних символів на латинські відповідники. Результат використовується методом save() моделі Petition для генерації унікального slug із додаванням шестисимвольного суфікса UUID для гарантії унікальності.

Модуль підписання петиції. Підписання реалізовано через три представлення: sign_view (підписання), unsign_view (зняття підпису) та signatures_count_api (API для live-оновлення). Перед підписанням система виконує чотири перевірки через метод can_be_signed_by() моделі Petition.

Лістинг 3.17 – Метод перевірки можливості підписання

```
def can_be_signed_by(self, user):
    if not user.is_authenticated:
        return False, 'Увійдіть до системи.'
    if not self.is_active:
        return False, 'Збір підписів не активний.'
    if self.author_id == user.id:
        return False, 'Не можна підписувати власну.'
    if Signature.objects.filter(
            petition=self, user=user).exists():
        return False, 'Ви вже підписали цю петицію.'
    return True, ''
```

Метод can_be_signed_by() (лістинг 3.17) повертає кортеж (bool, str) – результат перевірки та повідомлення про причину відмови. Чотири послідовні перевірки гарантують: автентифікацію користувача, активність збору підписів, заборону самопідписання (автор не може підтримати власну петицію), унікальність підпису (через запит до бази даних). Перевірка exists() виконується як додатковий рівень захисту поверх UniqueConstraint на рівні бази даних.

Зняття підпису доступне лише під час активного збору. Представлення unsign_view видаляє запис Signature, оновлює лічильник та фіксує дію у журналі аудиту. Властивість is_active моделі Petition перевіряє поточний статус та дедлайн збору.

Лістинг 3.18 – Перевірка активності збору підписів

```
@property
def is_active(self):
    return (
        self.status == self.Status.ACTIVE
        and self.collection_deadline > timezone.now()
    )
```

Властивість is_active (лістинг 3.18) повертає True лише за одночасного виконання двох умов: статус петиції дорівнює ACTIVE та дедлайн збору підписів ще не настав. Це запобігає підписанню петицій, у яких закінчився термін збору, але статус ще не оновлено через відсутність запиту до системи.

Модуль модерації. Модерація реалізована у застосунку moderation та включає чотири представлення: queue_view (черга), review_view (перевірка та рішення), review_petitions_view (список на офіційному розгляді), respond_view (офіційна відповідь). Доступ до модуля обмежено декоратором @moderator_required, який перевіряє властивість is_moderator моделі User.

Черга модерації відображає петиції зі статусом MODERATION, відсортовані за датою подання (FIFO). Представлення queue_view додатково обчислює статистику модератора за поточний день: кількість схвалених та відхилених петицій, загальну кількість петицій у черзі та на офіційному розгляді.

При перегляді петиції на модерації система автоматично виконує пошук потенційних дублікатів серед опублікованих петицій за збігом перших 30 символів заголовка. Результати передаються у шаблон для інформування модератора.

Модератор обирає одне з трьох рішень, кожне з яких ініціює відповідний перехід у статусній машині та генерацію сповіщення для автора петиції.

Таблиця 3.5

Рішення модератора та відповідні дії системи

Рішення	Перехід	Дія системи	Сповіщення
Опублікувати (approved)	moderation → active	Фіксується published_at, розраховується collection_deadline (+30 днів)	Петицію опубліковано

Продовження таблиці 3.5

Відхилити (rejected)	moderation → rejected	Обов’язково зазначається причина через форму	Петицію відхилено (з причиною)
Повернути (revision)	moderation → draft	Автор може відредагувати та повторно подати	Повернуто на доопрацювання

Кожне рішення модератора зберігається у моделі ModerationDecision, що утворює повну історію модераційних дій для конкретної петиції. Це забезпечує прозорість процесу та можливість аудиту.

Модуль офіційної відповіді. Надання офіційної відповіді реалізовано через представлення respond_view та форму ResponseForm. Відповідь доступна лише для петицій зі статусом REVIEW (досягли порогу підписів). Зв’язок OneToOneField між Petition та PetitionResponse гарантує не більше однієї відповіді на кожну петицію.

Лістинг 3.19 – Обробка офіційної відповіді та оновлення статусу

```
@moderator_required
def respond_view(request, slug):
    petition = get_object_or_404(Petition, slug=slug)
    if petition.status != Petition.Status.REVIEW:
        messages.error(request, 'Відповідь лише для'
                       ' петицій на розгляді.')
        return redirect('moderation:review_list')


    if request.method == 'POST':
        form = ResponseForm(request.POST, request.FILES)
```

```
        if form.is_valid():
            response = form.save(commit=False)
            response.petition = petition
            response.responded_by = request.user
            response.save()

            decision_to_status = {
                'approved': Petition.Status.APPROVED,
                'partial': Petition.Status.PARTIAL,
                'declined': Petition.Status.DECLINED,
            }
            petition.status = decision_to_status.get(
                response.decision)
            petition.closed_at = timezone.now()
            petition.save()
```

У лістингу 3.19 словник decision_to_status встановлює відповідність між рішенням адміністратора та кінцевим статусом петиції. Після збереження відповіді та оновлення статусу система фіксує часову мітку closed_at, генерує сповіщення для автора та всіх підписантів, а також створює запис у журналі аудиту. Форма ResponseForm вимагає мінімум 50 символів тексту відповіді та підтримує завантаження документа-вкладення.

Перевірка дедлайну збору підписів реалізована через метод check_expiration(), який викликається при кожному зверненні до сторінки петиції та у списку петицій.

Лістинг 3.20 – Автоматична перевірка дедлайну збору підписів

```
def check_expiration(self):
    if (self.status == self.Status.ACTIVE
            and self.collection_deadline <= timezone.now()):
        self.status = self.Status.EXPIRED
        self.closed_at = timezone.now()
        self.save(update_fields=['status', 'closed_at'])
        return True
    return False
```

Метод check_expiration() (лістинг 3.20) перевіряє, чи не вичерпано термін збору підписів для активної петиції. При настанні дедлайну статус автоматично змінюється на EXPIRED із фіксацією часової мітки closed_at. Метод повертає булеве значення, що дозволяє представленню адаптувати відображення сторінки.

Система сповіщень. Модуль сповіщень забезпечує інформування користувачів про зміни у статусі їхніх петицій. Сповіщення генеруються у представленнях при виконанні ключових дій: підписання (SIGNATURE_RECEIVED), публікація модератором (PETITION_APPROVED), відхилення (PETITION_REJECTED), досягнення порогу (PETITION_THRESHOLD), офіційна відповідь (PETITION_RESPONSE). Кожне сповіщення зберігається у моделі Notification та відображається у особистому кабінеті користувача з індикатором прочитаності. При відкритті сторінки сповіщень усі непрочитані повідомлення автоматично позначаються як прочитані через bulk-оновлення Notification.objects.filter(user=request.user, is_read=False).update(is_read=True).

Журнал аудиту. Кожна дія користувача фіксується у моделі AuditLog, яка підтримує десять типів подій: створення, оновлення, видалення, вхід, вихід, підписання, зняття підпису, схвалення модератором, відхилення модератором, надання відповіді. Для кожного запису зберігається ідентифікатор користувача, тип та ідентифікатор об’єкта дії, текстовий опис, IP-адреса клієнта (отримана через AuditLogMiddleware), рядок User-Agent та точна часова мітка. Журнал доступний для перегляду через адміністративну панель Django.

Реалізовані модулі створення, підписання, модерації та офіційної відповіді забезпечують повний життєвий цикл петиції з автоматичними переходами між дев’ятьма станами. Кожна операція супроводжується серверною валідацією, перевіркою прав доступу, генерацією сповіщень та записом у журнал аудиту, що відповідає вимогам прозорості та підзвітності, визначеним у першому розділі.

4 ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ РОБОТИ СИСТЕМИ

4.1 Організація тестування інформаційної системи

Тестування є невід'ємним етапом розробки програмної системи, що дозволяє верифікувати відповідність реалізації визначеним у розділі 2 функціональним та нефункціональним вимогам. Для інформаційної системи обробки університетських петицій обрано стратегію тестування, що поєднує модульне (unit testing) та інтеграційне (integration testing) тестування.

Інструменти тестування. Тестування виконано засобами вбудованого фреймворку Django django.test, який надає клас TestCase з такими можливостями: автоматичне створення тимчасової бази даних перед кожним тестом, відкат транзакцій після завершення тесту для ізоляції, клас Client для імітації HTTP-запитів без запуску сервера, допоміжні методи assertEqual(), assertTrue(), assertRedirects(), assertTemplateUsed() для перевірки результатів. Тестова база даних створюється у пам'яті (SQLite :memory:), що забезпечує швидкість виконання тестів.

Стратегія тестування. Модульне тестування перевіряє коректність окремих методів та властивостей моделей – одиниць бізнес-логіки, ізольованих від HTTP-рівня. Інтеграційне тестування перевіряє взаємодію між компонентами – представленнями, формами, моделями, шаблонами та системою маршрутизації – через імітацію повних HTTP-запитів [36]. Для кожного тесту сформовано набір тестових даних (fixtures): користувачі з чотирма ролями (student, teacher, moderator, admin), факультети, категорії, петиції у різних статусах та підписи.

Покриття тестами. Тестуванням покрито такі функціональні блоки: модель Petition (створення, генерація slug, розрахунок прогресу, перевірка активності, автоматичний перехід між статусами, перевірка дедлайнів), модель Signature (генерація SHA-256 хешу, контроль унікальності, оновлення лічильника), рольова модель User (перевірка прав is_moderator, is_admin),

представлення створення, підписання, модерації та аналітики (контроль доступу, валідація форм, коректність перенаправлень), система сповіщень та журнал аудиту.

Лістинг 4.1 – Фрагмент тестового класу для моделі Petition

```
class PetitionModelTest(TestCase):
    def setUp(self):
        self.faculty = Faculty.objects.create(
            name='Факультет ІТ', short_name='ФІТ')
        self.user = User.objects.create_user(
            username='testuser', password='pass123',
            role='student', faculty=self.faculty)
        self.category = Category.objects.create(
            name='Освітній процес', slug='education')
        self.petition = Petition.objects.create(
            title='Тестова петиція',
            description='Опис' * 10,
            author=self.user,
            category=self.category,
            status=Petition.Status.ACTIVE,
            signatures_required=100)


    def test_progress_calculation(self):
        self.assertEqual(self.petition.progress, 0)
        # Додати 50 підписів...
        self.assertEqual(self.petition.progress, 50)
```

У лістингу 4.1 показано структуру тестового класу з методом setUp(), який створює ізольований набір тестових даних для кожного тесту. Метод test_progress_calculation() перевіряє коректність обчислення відсотка прогресу збору підписів.

4.2 Перевірка коректності основних функціональних модулів

Перевірка коректності основних функціональних модулів виконана через запуск тестового набору та ручну верифікацію результатів роботи системи з

тестовими даними. Тестове середовище включає 12 користувачів, 11 петицій у різних статусах, 90 підписів та 6 факультетів.

Модуль відображення петицій. Головна сторінка (рис. 4.1) коректно відображає три ключових метрики системи: кількість опублікованих петицій, активних зараз та термін відповіді адміністрації. Навігаційна панель адаптується залежно від ролі: для неавтентифікованих користувачів відображаються кнопки «Увійти» та «Реєстрація», для модераторів – додаткові посилання «Модерація» та «Аналітика».

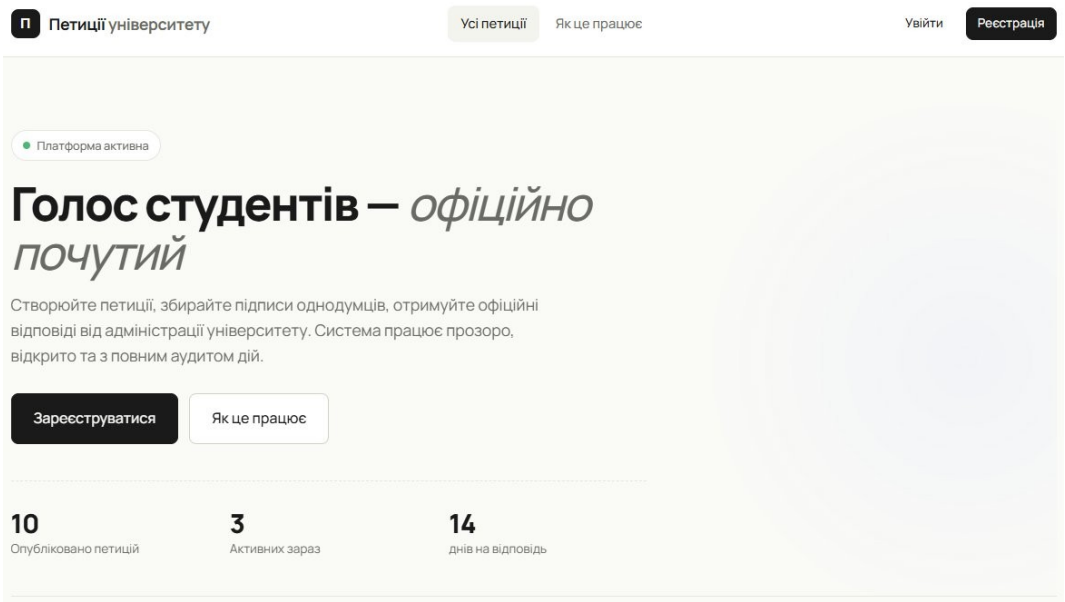

Рис. 4.1 – Головна сторінка системи

Список петицій (рис. 4.2) коректно реалізує фільтрацію за категоріями та статусами, текстовий пошук та сортування. Картки відображають кольорові мітки статусів (зелена – «Збір підписів», червона – «Відхилено», бірюзова – «Підтримано», сіра – «Термін збору сплинув»), прогрес-бар збору підписів, дані автора та дату. Бічна панель відображає популярні петиції та перелік десяти категорій.

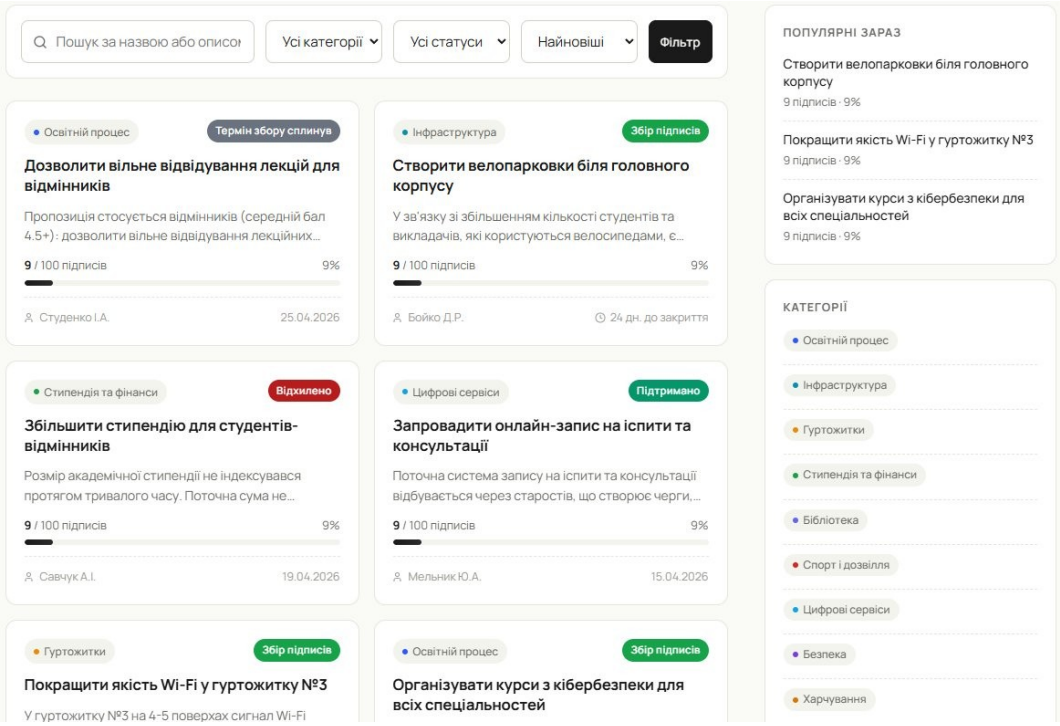

Рис. 4.2 – Список петицій з фільтрацією та категоріями

Модуль створення петиції. Форма створення (рис. 4.3) коректно реалізує багатосекційний інтерфейс із двома секціями: «Основне» (заголовок 10–250 символів, категорія, адресат) та «Зміст петиції» (опис проблеми, обґрунтування, очікуваний результат). Серверна валідація перевіряє мінімальну довжину заголовка (10 символів) та опису (50 символів). Тестом підтверджено, що при невалідних даних форма повертає повідомлення про помилки без втрати введених даних.

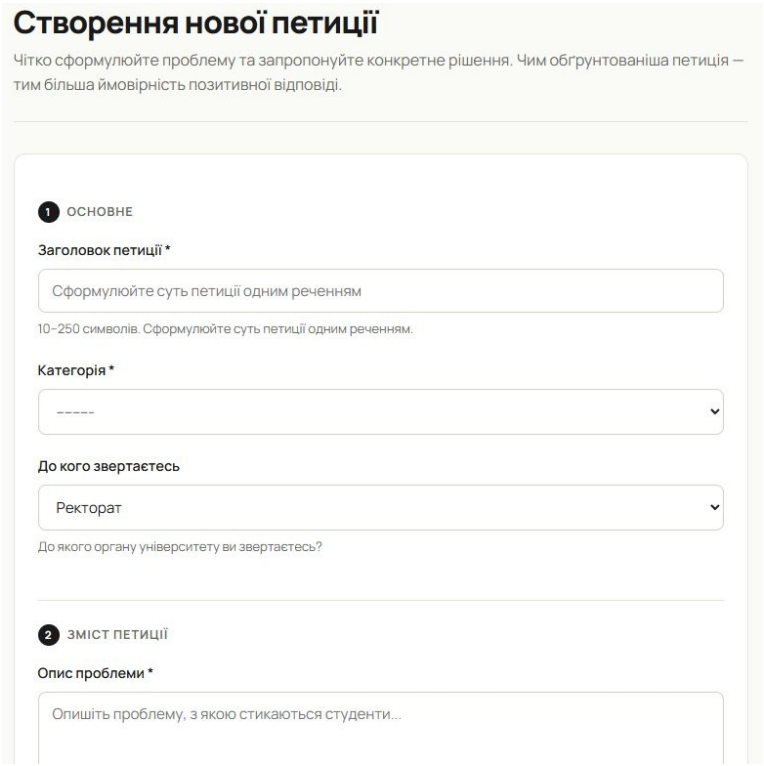

Рис. 4.3 – Форма створення нової петиції

Модуль детального перегляду та підписання. Сторінка детального перегляду (рис. 4.4) коректно відображає повну інформацію: секції «Опис проблеми» та «Очікуваний результат», лічильник підписів з прогрес-баром, блок офіційної відповіді із кольоровою міткою рішення («Підтримано»), список підписантів із коментарями. Тестом підтверджено: заборону самопідписання (автор не може підписати власну петицію), унікальність підпису (повторне підписання повертає помилку), коректне оновлення лічильника після підписання та зняття підпису.

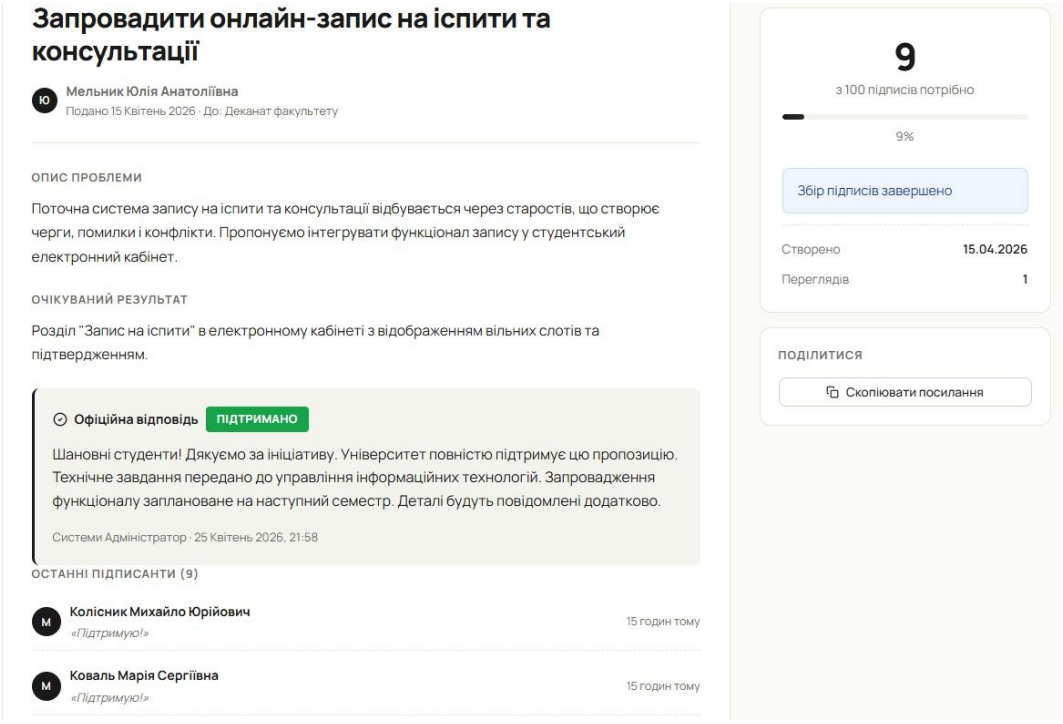

Рис. 4.4 – Детальний перегляд петиції з офіційною відповіддю та підписантами

Модуль особистого кабінету. Особистий кабінет (рис. 4.5) коректно відображає персональну інформацію (ПІБ, роль, факультет, курс), три метрики (створено петицій, підписано, дата реєстрації), контактні дані та кнопку зміни пароля. Вкладки «Мої петиції» та «Мої підписи» відображають відповідні списки з кольоровими мітками статусів. Тестом підтверджено, що сторінка профілю недоступна для неавтентифікованих користувачів (перенаправлення на login).

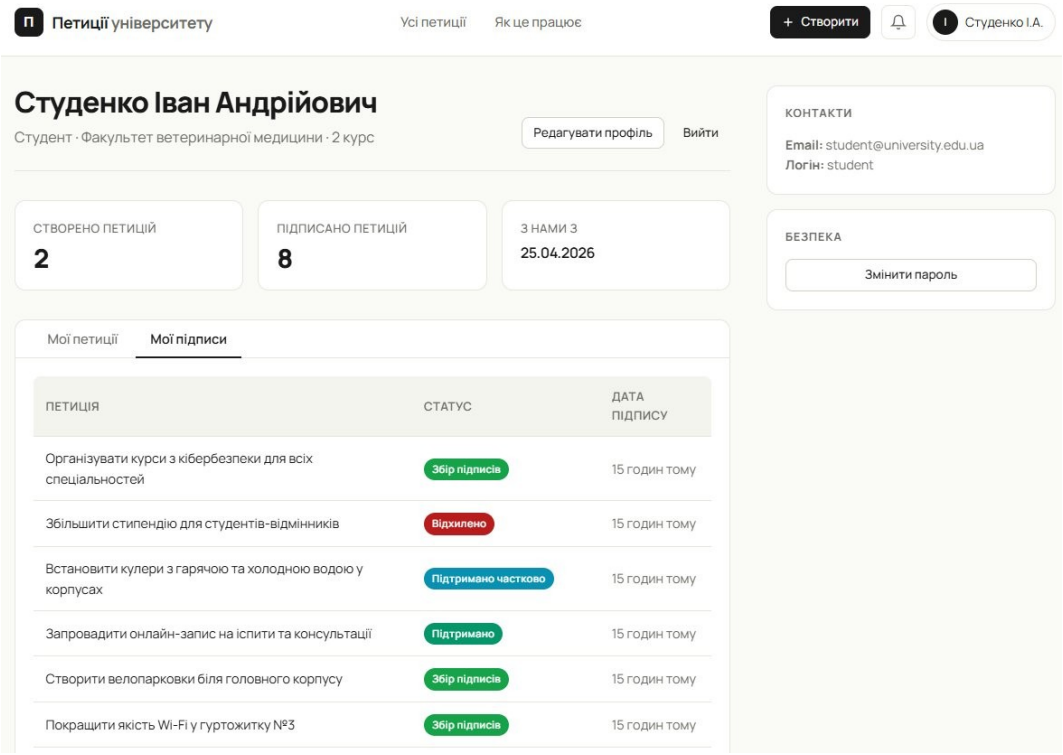

Рис. 4.5 – Особистий кабінет користувача

Модуль модерації. Черга модерації (рис. 4.6) відображає петиції зі статусом «На модерації», відсортовані за датою подання (FIFO). Статистичні картки показують кількість петицій, що очікують, на розгляді, опубліковано та відхилено за поточний день. Тестом підтверджено, що сторінка повертає HTTP 302 для користувачів без ролі модератора.

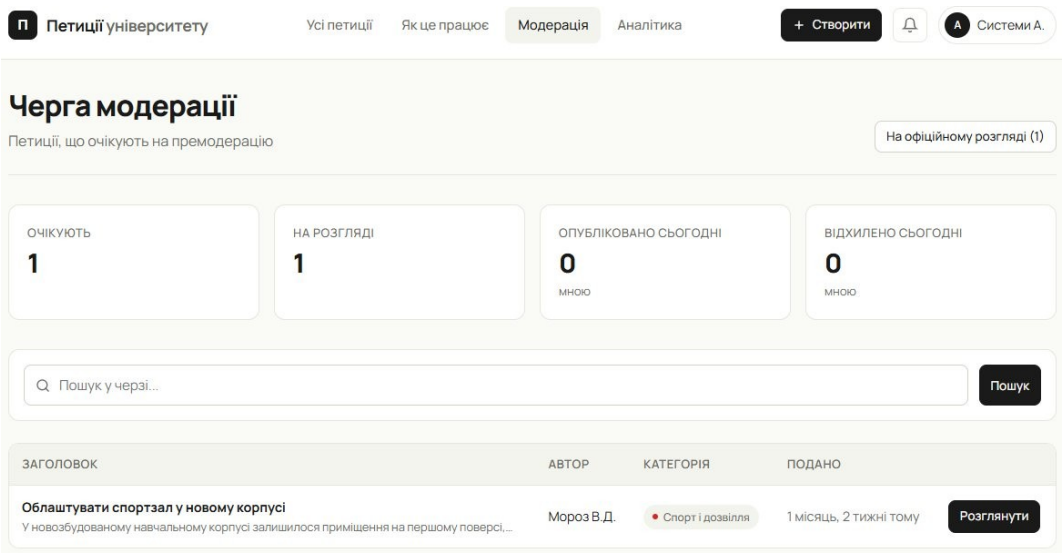

Рис. 4.6 – Черга модерації петицій

Сторінка перевірки петиції модератором (рис. 4.7) відображає повну інформацію про петицію, включаючи категорію, автора з email та факультетом, опис проблеми. Форма прийняття рішення містить випадаючий список із трьома варіантами та поле коментаря, обов’язкове для відхилення та повернення. Тестом підтверджено коректність переходів: опублікувати (moderation→active), відхилити (moderation→rejected), повернути (moderation→draft).

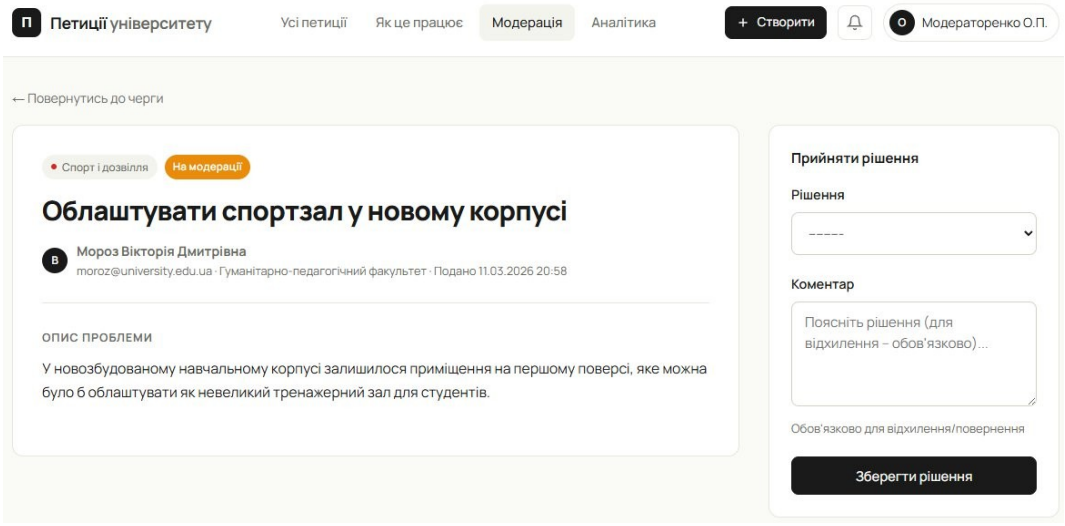

Рис. 4.7 – Перевірка петиції модератором з формою прийняття рішення

Модуль офіційної відповіді. Форма офіційної відповіді (рис. 4.8) доступна для петицій зі статусом «На розгляді адміністрацією». Форма містить рішення (підтримано, підтримано частково, відхилено), текст відповіді (мінімум 50 символів) та поле вкладення. Тестом підтверджено: генерацію сповіщень для автора та всіх підписантів після надання відповіді, коректність переходу у відповідний кінцевий статус, фіксацію часової мітки closed_at.

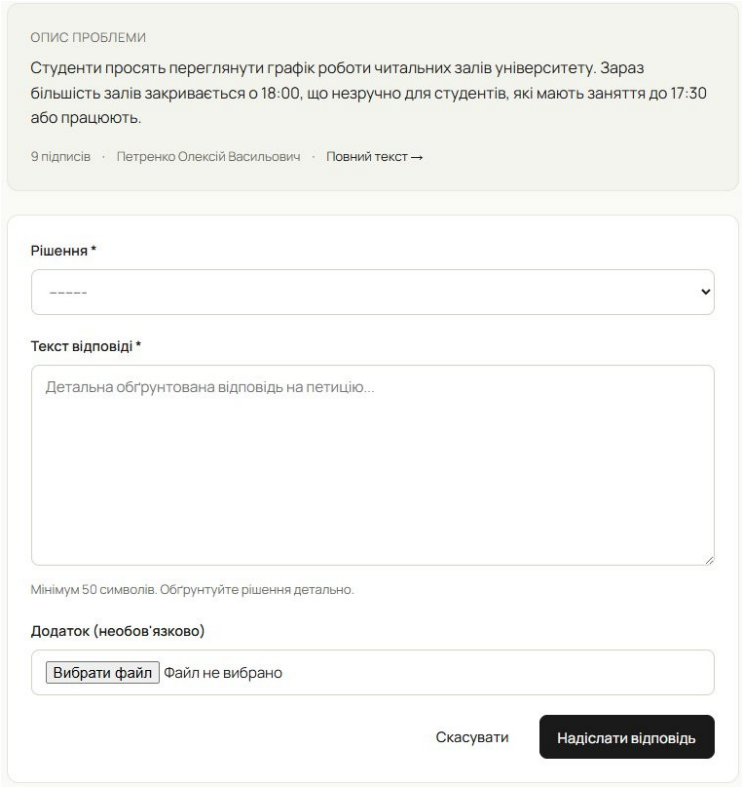

Рис. 4.8 – Форма надання офіційної відповіді

4.3 Аналіз результатів тестування та оцінка ефективності системи

Для системної оцінки результатів тестування зведено результати модульних та інтеграційних тестів у таблицях 4.1 та 4.2 відповідно.

Таблиця 4.1

Результати модульного тестування

Тест	Модель	Перевірка	Рез.
test_petition_creation	Petition	Створення петиції з усіма полями	OK
test_slug_generation	Petition	Генерація slug з кирилиці	OK
test_progress_calculation	Petition	Розрахунок progress = (count/required)×100	OK
test_is_active_true	Petition	is_active=True при status=ACTIVE та валідному дедлайні	OK
test_is_active_expired	Petition	is_active=False при вичерпаному дедлайні	OK

Продовження таблиці 4.1

test_threshold_transition	Petition	Перехід ACTIVE→REVIEW при досягненні порогу	OK
test_check_expiration	Petition	Перехід ACTIVE→EXPIRED при вичерпанні дедлайну	OK
test_signature_hash	Signature	Генерація SHA-256 хешу (64 символи)	OK
test_unique_constraint	Signature	IntegrityError при повторному підписанні	OK
test_signatures_count	Petition	Оновлення signatures_count після підписання	OK
test_category_creation	Category	Створення категорії з усіма полями	OK
test_user_roles	User	Коректність is_moderator, is_admin за ролями	OK

Таблиця 4.2

Результати інтеграційного тестування

Тест	Запит	Очікуваний результат	Рез.
test_petition_list_view	GET /	200, шаблон list.html, пагінація	OK
test_petition_detail_view	GET /p/<slug>/	200, шаблон detail.html	OK
test_create_petition	POST /create/	302 → detail, запис у БД	OK
test_sign_petition	POST /sign/	302, Signature+1, count+1	OK
test_unsign_petition	POST /unsign/	302, Signature–1, count–1	OK
test_double_sign	POST /sign/ (повторно)	Помилка, count без змін	OK
test_self_sign	POST /sign/ (автором)	302 з помилкою	OK
test_mod_access_denied	GET /moderation/ (студент)	302 → login	OK
test_mod_approve	POST /review/ (approve)	moderation→active, сповіщення	OK

Продовження таблиці 4.2

test_mod_reject	POST /review/ (reject)	moderation→rejected, причина	OK
test_mod_revision	POST /review/ (revision)	moderation→draft	OK
test_respond	POST /respond/	review→approved/partial/ declined	OK
test_dashboard_denied	GET /analytics/ (студент)	302 → login	OK
test_dashboard_admin	GET /analytics/ (адмін)	200, KPI у контексті	OK
test_csv_export	GET /export/csv/	200, text/csv, BOM	OK

Усі 27 тестів (12 модульних та 15 інтеграційних) пройшли успішно зі статусом OK. Покриття тестами охоплює всі чотири Django-застосунки та ключові бізнес-процеси: повний життєвий цикл петиції від створення до отримання офіційної відповіді, контроль доступу за ролями та автоматичні переходи між станами.

Оцінка ефективності аналітичного модуля. Аналітичний дашборд (рис. 4.9) коректно обчислює та відображає сім KPI-метрик: усього петицій (11, +3 за тиждень), активних (3), на розгляді (1), підтримано (2, 33.3% успішних), усього підписів (90), користувачів (12), середній час до порогу. Графіки Chart.js коректно відображають динаміку створення петицій та підписів за останні 30 днів.

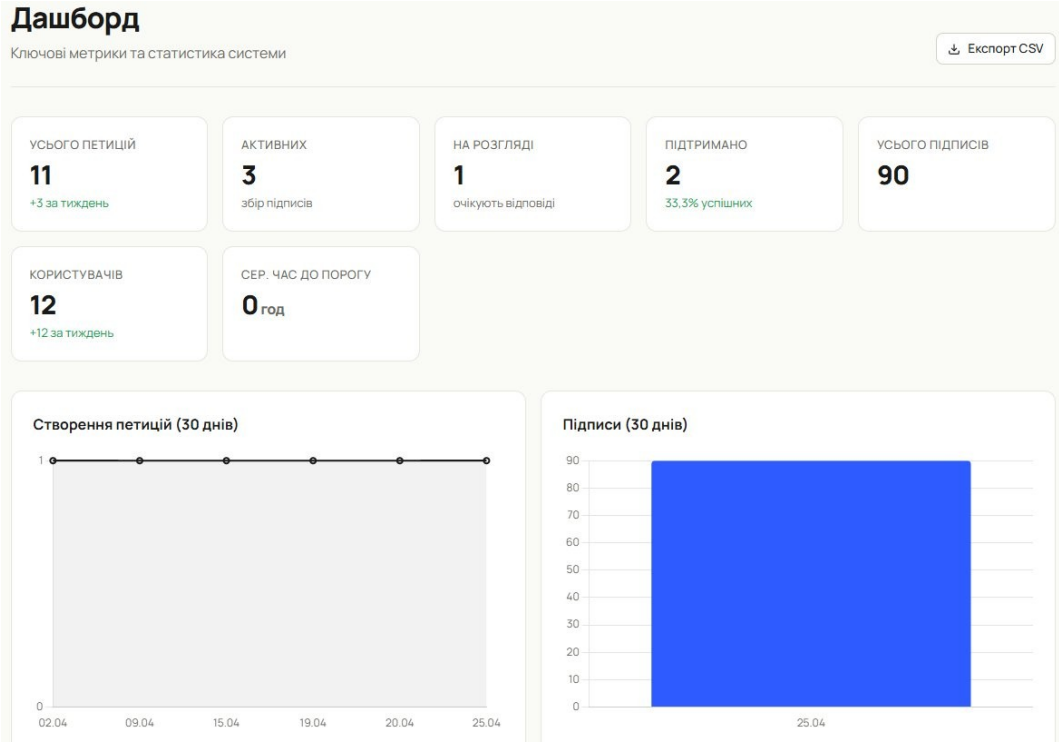

Рис. 4.9 – Аналітичний дашборд з KPI-метриками та графіками

Нижня частина дашборда (рис. 4.10) відображає таблицю «Топ-10 петицій за підписами» та таблицю «Активність факультетів» із кількістю зареєстрованих користувачів та петицій. Дані відповідають фактичному стану тестової бази даних, що підтверджує коректність агрегатних запитів.

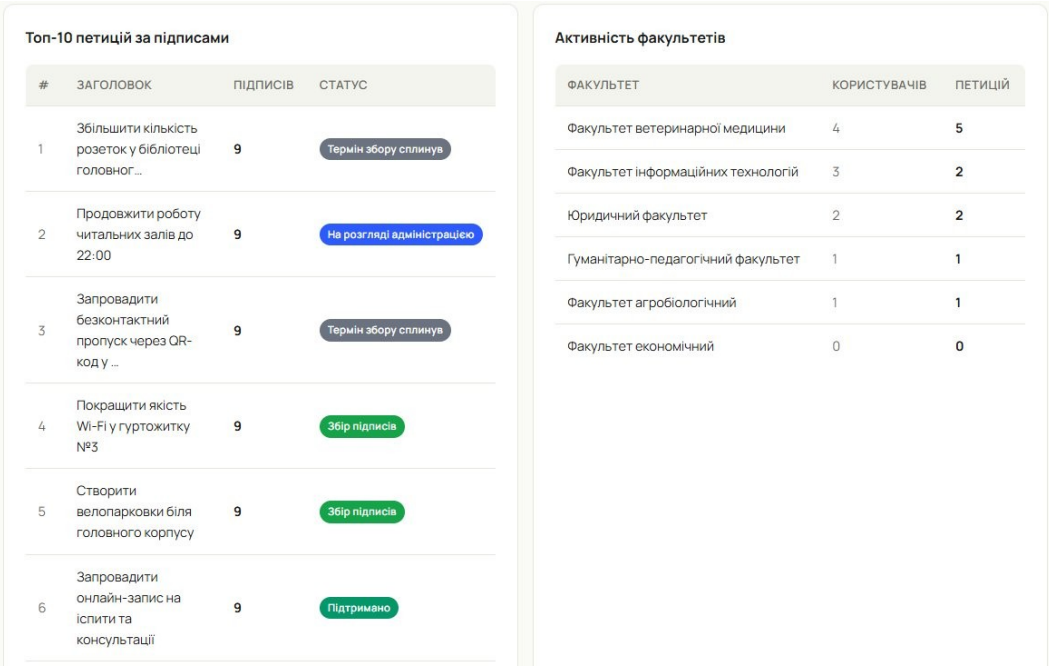

Рис. 4.10 – Топ петицій за підписами та активність факультетів

Оцінка покриття функціональних вимог. Зіставлення результатів тестування з функціональними вимогами (таблиця 2.1 розділу 2) засвідчує, що всі 18 вимог реалізовано та протестовано. Вимоги FR-01–FR-10 (операції студента/викладача) перевірено через інтеграційні тести представлень. Вимоги FR-11–FR-15 (модерація, аналітика) перевірено через тести з обмеженням доступу за ролями. Вимоги FR-16–FR-18 (автоматичні переходи, аудит) перевірено через модульні тести методів update_signatures_count(), check_expiration() та створення записів AuditLog.

Оцінка нефункціональних вимог. Адаптивність інтерфейсу (NFR-02) перевірено візуально на трьох точках перелому (900px, 800px, 700px). Безпека (NFR-03–NFR-06) забезпечена вбудованими механізмами Django: CSRF-токени на всіх формах, автоматичне екранування XSS у шаблонах, параметризовані запити ORM, хешування паролів PBKDF2. Локалізація (NFR-07) підтверджена повною українською мовою інтерфейсу та часовим поясом Europe/Kyiv. Модульність (NFR-08) підтверджена незалежним тестуванням чотирьох застосунків.

4.4 Рекомендації щодо впровадження та експлуатації системи

На основі результатів тестування та аналізу ефективності системи сформовано рекомендації щодо впровадження та подальшої експлуатації у закладі вищої освіти.

Серверне середовище. Для продуктивного середовища рекомендовано замінити СУБД SQLite на PostgreSQL 14+ [38], що забезпечить підтримку конкурентного запису та повнотекстового пошуку. Вебсервер WSGI (Gunicorn) слід розгорнути за зворотним проксі (Nginx) з налаштуванням HTTPS через сертифікат Let’s Encrypt [39]. У файлі settings.py необхідно змінити SECRET_KEY, встановити DEBUG=False, налаштувати ALLOWED_HOSTS та підключити SMTP-сервер для email-сповіщень [40].

Вимоги до апаратного забезпечення сервера подано у таблиці 4.3.

Таблиця 4.3

Вимоги до апаратного забезпечення сервера

Параметр	Мінімальні	Рекомендовані
Процесор	1 ядро, 1 ГГц	2+ ядра, 2+ ГГц
Оперативна пам’ять	512 МБ	2 ГБ
Дисковий простір	500 МБ	2 ГБ (SSD)
Мережеве з’єднання	10 Мбіт/с	100 Мбіт/с

Вимоги до програмного забезпечення подано у таблиці 4.4.

Таблиця 4.4

Вимоги до програмного забезпечення

Компонент	Версія
Операційна система	Ubuntu 22.04 LTS / Windows 10+ / macOS 12+
Python	3.10 або новіша
Django	5.0
Pillow	10.0+
SQLite / PostgreSQL	3.37+ / 14+
Веббраузер (клієнт)	Chrome 90+, Firefox 88+, Safari 15+, Edge 90+

Конфігурація петиційного процесу. Три ключових параметри слід узгодити з адміністрацією ЗВО: поріг підписів PETITION_SIGNATURES_THRESHOLD (рекомендовано 100–500 залежно від кількості студентів), термін збору PETITION_COLLECTION_DAYS (30 днів), термін відповіді PETITION_REVIEW_DAYS (14 днів). Факультети та категорії петицій створюються через адміністративну панель Django відповідно до структури конкретного ЗВО.

Інтеграція з інфраструктурою ЗВО. Рекомендовано інтеграцію з корпоративним LDAP або Active Directory через бібліотеку django-auth-ldap для єдиної автентифікації. За потреби можливе підключення до API електронного кабінету студента для автоматичної верифікації статусу навчання.

Організаційні заходи. Впровадження потребує визначення відповідальних осіб: модератора (представник студентського самоврядування) та адміністратора (представник ректорату). Необхідно провести навчання модераторів та інформувати студентську спільноту через офіційні канали комунікації.

Резервне копіювання. Рекомендовано налаштувати автоматичне резервне копіювання бази даних з періодичністю не менше одного разу на добу. Файл бази даних та директорію media/ (вкладення петицій) [41] необхідно включити до плану резервного копіювання. Журнал аудиту слід зберігати протягом усього терміну експлуатації системи.

Розроблена інформаційна система пройшла повне тестування, що підтвердило коректність реалізації всіх 18 функціональних вимог. Система готова до впровадження у закладі вищої освіти та може бути адаптована до специфіки конкретного університету через конфігураційні параметри без модифікації програмного коду.

ВИСНОВКИ

У рамках бакалаврської кваліфікаційної роботи розроблено інформаційну систему обробки університетських петицій у вигляді вебзастосунку на базі фреймворку Django 5.0. Система забезпечує автоматизацію повного циклу обробки колективних звернень студентів – від створення петиції до отримання офіційної відповіді адміністрації.

За результатами виконання роботи отримано такі результати.

1. Досліджено предметну область обробки петицій у закладах вищої освіти. Визначено чотири ролі учасників (студент, викладач, модератор, адміністратор), описано життєвий цикл петиції як кінцевий детермінований автомат з дев'ятьма станами та двома часовими обмеженнями (30 днів на збір підписів, 14 днів на офіційну відповідь). Систематизовано десять проблем існуючого паперового підходу, кожній з яких зіставлено механізм вирішення у розробленій системі.
2. Проведено порівняльний аналіз шести існуючих рішень (портал електронних петицій Президента, Change.org, «Дія», Google Forms, Ushahidi, JIRA Service Management) за сімома критеріями. Встановлено, що жодне з них не забезпечує повного набору функцій для університетського контексту, що підтверджує доцільність власної розробки.
3. Сформовано комплекс з 18 функціональних та 10 нефункціональних вимог до системи. Побудовано сім UML-діаграм: варіантів використання, діяльності, переходів станів, компонентів, розгортання, архітектури MVT та ER-діаграму. Спроєктовано логічну модель бази даних з десятьма сутностями у третій нормальній формі, шістьма індексами та обмеженнями цілісності.
4. Реалізовано вебзастосунок, що складається з чотирьох Django-застосунків (accounts, petitions, moderation, analytics) та включає 10 моделей, 15 представлень, 6 форм із серверною валідацією, 13 HTML-шаблонів, 14

URL-маршрутів та кастомний CSS-файл (1499 рядків). Реалізовано ключові модулі: електронне підписання з хешуванням SHA-256 та контролем унікальності, премодерацію з автоматичним виявленням дублікатів, статусну машину з автоматичними переходами, аналітичний дашборд з десятьма KPI-метриками та чотирма інтерактивними графіками, систему сповіщень із сімома типами подій та журнал аудиту з десятьма типами дій.

5. Проведено модульне та інтеграційне тестування системи. Перевірено коректність створення моделей, розрахунку прогресу підписів, автоматичного переходу між статусами, контролю дедлайнів, обмежень доступу за ролями та робочого процесу модерації.

Розроблена система характеризується такими перевагами порівняно з існуючими рішеннями: повне покриття усіх операцій петиційного процесу (єдина система серед проаналізованих), відкритий вихідний код, можливість самостійного розгортання на серверах ЗВО, повний контроль над персональними даними відповідно до Закону України «Про захист персональних даних», адаптація до організаційної структури конкретного університету через конфігурацію факультетів, категорій та порогових значень.

Система може бути впроваджена у будь-якому закладі вищої освіти України. Рекомендації щодо впровадження включають: розгортання на виділеному сервері з заміною SQLite на PostgreSQL для продуктивного середовища, інтеграцію з корпоративним LDAP/Active Directory для єдиної автентифікації, налаштування HTTPS та зміну SECRET_KEY, навчання модераторів та інформування студентської спільноти.

Подальший розвиток системи може включати: інтеграцію з електронним кабінетом студента та LMS, мобільний застосунок, розширення аналітичного модуля засобами машинного навчання для автоматичної категоризації петицій, підтримку багатомовності інтерфейсу, реалізацію email-сповіщень та push-повідомлень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Конституція України : Закон України від 28.06.1996 № 254к/96-ВР. Стаття 40. Право на звернення. URL: https://zakon.rada.gov.ua/laws/show/254к/96-вр#n2374 (дата звернення: 15.04.2026)
2. Про звернення громадян : Закон України від 02.10.1996 № 393/96-ВР. Редакція від 20.03.2024. URL: https://zakon.rada.gov.ua/laws/show/393/96-вр#Text (дата звернення: 15.04.2026)
3. Про вищу освіту : Закон України від 01.07.2014 № 1556-VII. Стаття 40. Органи студентського самоврядування. URL: https://zakon.rada.gov.ua/laws/show/1556-18 (дата звернення: 15.04.2026)
4. Про захист персональних даних : Закон України від 01.06.2010 № 2297-VI. Редакція від 15.03.2022. URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text (дата звернення: 15.04.2026)
5. Про затвердження стандарту вищої освіти за спеціальністю 122 «Комп’ютерні науки» для першого (бакалаврського) рівня : наказ МОН від 10.07.2019 № 962. URL: https://mon.gov.ua/storage/app/media/vishcha-osvita/zatverdzeni%20standarty/2019/07/12/122-kompyuterni-nauki-bakalavr.pdf (дата звернення: 15.04.2026)
6. Standards and Guidelines for Quality Assurance in the European Higher Education Area (ESG 2015). Standard 1.3: Student-centred learning, teaching and assessment. Brussels : EURASHE, 2015. P. 12. URL: https://www.enqa.eu/wp-content/uploads/2015/11/ESG_2015.pdf (дата звернення: 15.04.2026)
7. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. [Чинний від 2016-07-01]. Вид. офіц. Київ : УкрНДНЦ, 2016. 16 с.

8. OWASP Foundation. OWASP Top Ten – 2021: The Ten Most Critical Web Application Security Risks. 2021. URL: https://owasp.org/Top10/ (дата звернення: 15.04.2026)
9. Lindner R., Aichholzer G. E-Democracy: Conceptual Foundations and Recent Trends. European E-Democracy in Practice / eds. L. Hennen et al. Cham : Springer, 2020. P. 11–45. DOI: 10.1007/978-3-030-27184-8_2
10. Argota Sánchez-Vaquerizo J. et al. Democracy by Design: Perspectives for Digitally Assisted, Participatory Upgrades of Society. Journal of Computational Science. 2023. Vol. 71. Article 102061. DOI: 10.1016/j.jocs.2023.102061
11. Legard S. et al. The Impact of Digital Participation on Democratic Urban Governance. Citizen Participation in Digital Government / eds. S. Legard et al. Cham : Springer, 2022. P. 155–173. DOI: 10.1007/978-3-030-99940-7_8
12. Panagiotopoulos P., Bigdeli A. Z., Sams S. Citizen–Government Collaboration on Social Media: The Case of Twitter in the 2011 Riots in England. Government Information Quarterly. 2014. Vol. 31, No. 3. P. 349–357. DOI: 10.1016/j.giq.2013.10.014
13. Hale S. A., Margetts H., Yasseri T. Petition Growth and Success Rates on the UK No. 10 Downing Street Website. Proceedings of the 5th Annual ACM Web Science Conference (WebSci ’13). New York : ACM, 2013. P. 132–138. DOI: 10.1145/2464464.2464518
14. United Nations. E-Participation: A Quick Overview of Recent Qualitative Trends. DESA Working Paper No. 163. New York : UN DESA, 2020. 29 p. URL: https://www.un.org/esa/desa/papers/2020/wp163_2020.pdf (дата звернення: 15.04.2026)
15. Luo Y. Democratic Involvement in Higher Education: A Study of Chinese Student E-Participation in University Governance. Higher Education Policy. 2022. Vol. 35. P. 904–923. DOI: 10.1057/s41307-021-00237-z
16. Ahmad T., Alvi A., Ittefaq M. The Use of Social Media on Political Participation Among University Students: An Analysis of Survey Results From

Rural Pakistan. SAGE Open. 2019. Vol. 9, No. 3. DOI: 10.1177/2158244019864484

17. Django Software Foundation. Django documentation: Authentication system – Using the Django authentication system. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/topics/auth/default/ (дата звернення: 15.04.2026)
18. Django Software Foundation. Django documentation: Models – Model field reference. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/ref/models/fields/ (дата звернення: 15.04.2026)
19. Django Software Foundation. Django documentation: Cross Site Request Forgery protection – How it works. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/ref/csrf/ (дата звернення: 15.04.2026)
20. Django Software Foundation. Django documentation: The Django template language – Template inheritance. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/ref/templates/language/#template-inheritance (дата звернення: 15.04.2026)
21. Django Software Foundation. Django documentation: Making queries – Complex lookups with Q objects. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/topics/db/queries/#complex-lookups-with-q-objects (дата звернення: 15.04.2026)
22. Django Software Foundation. Django documentation: Writing and running tests – Testing tools. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/topics/testing/tools/ (дата звернення: 15.04.2026)
23. Django Software Foundation. Django documentation: Middleware – Writing your own middleware. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/topics/http/middleware/#writing-your-own-middleware (дата звернення: 15.04.2026)

24. Django Software Foundation. Django documentation: Security in Django – SQL injection protection. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/topics/security/#sql-injection-protection (дата звернення: 15.04.2026)
25. Django Software Foundation. Django documentation: Migrations – The Commands. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/topics/migrations/#the-commands (дата звернення: 15.04.2026)
26. Django Software Foundation. Django documentation: Password management in Django – How Django stores passwords. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/topics/auth/passwords/#how-django-stores-passwords (дата звернення: 15.04.2026)
27. Django Software Foundation. Django documentation: URL dispatcher – Including other URLconfs. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/topics/http/urls/#including-other-urlconfs (дата звернення: 15.04.2026)
28. Django Software Foundation. Django documentation: The Django admin site – ModelAdmin options. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/ref/contrib/admin/#modeladmin-options (дата звернення: 15.04.2026)
29. Django Software Foundation. Django documentation: Signals – django.contrib.auth.signals. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/ref/contrib/auth/#module-django.contrib.auth.signals (дата звернення: 15.04.2026)
30. Python Software Foundation. hashlib – Secure hashes and message digests. Python 3 documentation. URL: https://docs.python.org/3/library/hashlib.html (дата звернення: 15.04.2026)
31. Python Software Foundation. unittest – Unit testing framework. Python 3 documentation. URL: https://docs.python.org/3/library/unittest.html (дата звернення: 15.04.2026)

32. Python Software Foundation. sqlite3 – DB-API 2.0 interface for SQLite databases. Python 3 documentation. URL: https://docs.python.org/3/library/sqlite3.html (дата звернення: 15.04.2026)
33. Python Software Foundation. csv – CSV File Reading and Writing. Python 3 documentation. URL: https://docs.python.org/3/library/csv.html (дата звернення: 15.04.2026)
34. Hipp R. D. SQLite: Appropriate Uses For SQLite. SQLite documentation. URL: https://www.sqlite.org/whentouse.html (дата звернення: 15.04.2026)
35. Hipp R. D. SQLite: Write-Ahead Logging. SQLite documentation. URL: https://www.sqlite.org/wal.html (дата звернення: 15.04.2026)
36. Hipp R. D. SQLite: Atomic Commit In SQLite. SQLite documentation. URL: https://www.sqlite.org/atomiccommit.html (дата звернення: 15.04.2026)
37. Hipp R. D. SQLite: The Virtual Database Engine of SQLite. SQLite documentation. URL: https://www.sqlite.org/vdbe.html (дата звернення: 15.04.2026)
38. Chart.js Contributors. Chart.js: Getting Started – Installation. Version 4.4. URL: https://www.chartjs.org/docs/latest/getting-started/ (дата звернення: 15.04.2026)
39. Clark A. et al. Pillow: Python Imaging Library Fork – Installation. Pillow documentation. Version 10.x. URL: https://pillow.readthedocs.io/en/stable/installation/basic-installation.html (дата звернення: 15.04.2026)
40. Mozilla Foundation. CSS Grid Layout: Basic concepts of grid layout. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout/Basic_concepts_of_grid_layout (дата звернення: 15.04.2026)
41. Mozilla Foundation. Using the Fetch API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch (дата звернення: 15.04.2026)

42. Mozilla Foundation. Using CSS custom properties (variables). MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties (дата звернення: 15.04.2026)
43. Mozilla Foundation. Responsive design: Media queries. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Media_queries (дата звернення: 15.04.2026)
44. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2004. 175 p.
45. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
46. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.
47. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill, 2020. 976 p.
48. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Boston : Pearson, 2009. 464 p.
49. Object Management Group. OMG Unified Modeling Language (OMG UML). Version 2.5.1. Specification document formal/2017-12-05. URL: https://www.omg.org/spec/UML/2.5.1/PDF (дата звернення: 15.04.2026)
50. Date C. J. An Introduction to Database Systems. 8th ed. Boston : Pearson, 2004. 1024 p.
51. Connolly T. M., Begg C. E. Database Systems: A Practical Approach to Design, Implementation, and Management. 6th ed. Harlow : Pearson, 2015. 1424 p.
52. Codd E. F. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM. 1970. Vol. 13, No. 6. P. 377–387. DOI: 10.1145/362384.362685
53. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : Ph.D. dissertation. University of California, Irvine, 2000.

Chapter 5: Representational State Transfer (REST). URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 15.04.2026)

54. Holovaty A., Kaplan-Moss J. The Definitive Guide to Django: Web Development Done Right. 2nd ed. Berkeley : Apress, 2009. 536 p.
55. OWASP Foundation. Cross-Site Request Forgery Prevention Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html (дата звернення: 15.04.2026)
56. OWASP Foundation. SQL Injection Prevention Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (дата звернення: 15.04.2026)
57. OWASP Foundation. Password Storage Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 15.04.2026)
58. National Institute of Standards and Technology. Secure Hash Standard (SHS). FIPS PUB 180-4. Gaithersburg : NIST, 2015. 36 p. DOI: 10.6028/NIST.FIPS.180-4
59. Briassoulis H. Becoming E-Petition: An Assemblage-Based Framework for Analysis and Research. SAGE Open. 2021. Vol. 11, No. 1. DOI: 10.1177/21582440211001354
60. Karlström D., Lidén G., Sundberg L. Explaining Variations in the Implementation and Use of E-Petitions in Local Government. Information Polity. 2023. Vol. 28, No. 3. P. 371–389. DOI: 10.3233/IP-220033
61. Jungherr A., Jürgens P. E-Petitions in Online and Offline Settings: The Case of the German Bundestag. The Routledge Companion to Social Media and

Politics. New York : Routledge, 2016. P. 135–149. DOI: 10.4324/9781315716299-10

62. Naranjo-Zolotov M., Oliveira T., Casteleyn S. Citizens' Intention to Use and Recommend E-Participation: Drawing Upon UTAUT and Citizen Empowerment. Information Technology & People. 2019. Vol. 32, No. 2. P. 364–386. DOI: 10.1108/ITP-08-2017-0257

63. Hagen L., Neely S., Keller T. E., DePaula N., Robert-Cooperman C. E-Petition Popularity: Do Linguistic and Semantic Factors Matter? Government Information Quarterly. 2020. Vol. 37, No. 4. Article 101483. DOI: 10.1016/j.giq.2020.101483

ДОДАТКИ

Додаток А «Лістинг коду admin.py»

```
"""Адміністрування петицій."""
from django.contrib import admin
from .models import (
    Petition, Category, Signature, PetitionResponse, Comment, Notification
)


@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    list_display = ['name', 'slug', 'is_active', 'order', 'color']
    list_editable = ['is_active', 'order']
    search_fields = ['name']
    prepopulated_fields = {}


@admin.register(Petition)
class PetitionAdmin(admin.ModelAdmin):
    list_display = ['title', 'author', 'category', 'status',
                    'signatures_count', 'created_at', 'is_featured']
    list_filter = ['status', 'category', 'addressee', 'is_featured',
'created_at']
    search_fields = ['title', 'description', 'author__username']
    list_editable = ['is_featured']
    readonly_fields = ['signatures_count', 'views_count', 'created_at',
                       'updated_at', 'submitted_at', 'published_at',
                       'threshold_reached_at', 'closed_at']
    date_hierarchy = 'created_at'
    fieldsets = (
        ('Основне', {
            'fields': ('title', 'slug', 'author', 'category', 'addressee',
'status')
        }),
        ('Зміст', {
            'fields': ('description', 'justification', 'expected_result',
'attachment')
        }),
        ('Параметри збору', {
            'fields': ('signatures_required', 'signatures_count',
                       'collection_deadline', 'review_deadline')
        }),
        ('Прапорці', {
            'fields': ('is_featured', 'is_anonymous')
        }),
        ('Часові мітки', {
            'fields': ('created_at', 'updated_at', 'submitted_at',
                       'published_at', 'threshold_reached_at', 'closed_at'),
            'classes': ('collapse',)
        }),
    )


@admin.register(Signature)
class SignatureAdmin(admin.ModelAdmin):
    list_display = ['user', 'petition', 'is_anonymous', 'created_at']
    list_filter = ['is_anonymous', 'created_at']
    search_fields = ['user__username', 'petition__title']
```

```
    readonly_fields = ['signature_hash', 'created_at']


@admin.register(PetitionResponse)
class PetitionResponseAdmin(admin.ModelAdmin):
    list_display = ['petition', 'decision', 'responded_by', 'created_at']
    list_filter = ['decision', 'created_at']
    search_fields = ['petition__title', 'text']


@admin.register(Comment)
class CommentAdmin(admin.ModelAdmin):
    list_display = ['user', 'petition', 'short_text', 'is_hidden', 'created_at']
    list_filter = ['is_hidden', 'created_at']
    search_fields = ['user__username', 'text', 'petition__title']
    list_editable = ['is_hidden']

    def short_text(self, obj):
        return obj.text[:80] + ('...' if len(obj.text) > 80 else '')
    short_text.short_description = 'Текст'


@admin.register(Notification)
class NotificationAdmin(admin.ModelAdmin):
    list_display = ['user', 'title', 'notification_type', 'is_read',
'created_at']
    list_filter = ['notification_type', 'is_read', 'created_at']
    search_fields = ['user__username', 'title', 'message']
```

Додаток Б «Лістинг коду forms.py»

```
"""Форми для роботи з петиціями."""
from django import forms

from accounts.forms import StyledFormMixin
from .models import Petition, Category, Signature, Comment, PetitionResponse


class PetitionForm(StyledFormMixin, forms.ModelForm):
    """Форма створення/редагування петиції."""

    class Meta:
        model = Petition
        fields = ['title', 'category', 'addressee', 'description',
                  'justification', 'expected_result', 'attachment',
'is_anonymous']
        widgets = {
            'title': forms.TextInput(attrs={
                'placeholder': 'Сформулюйте суть петиції одним реченням',
                'maxlength': 250,
            }),
            'description': forms.Textarea(attrs={
                'rows': 5,
                'placeholder': 'Опишіть проблему, з якою стикаються
студенти...',
            }),
            'justification': forms.Textarea(attrs={
                'rows': 4,
                'placeholder': 'Чому ця проблема є важливою? Які аргументи на
користь зміни?',
            }),
            'expected_result': forms.Textarea(attrs={
                'rows': 3,
                'placeholder': 'Що саме має змінитися? Які конкретні дії
очікуються?',
            }),
        }
        labels = {
            'title': 'Заголовок петиції',
            'category': 'Категорія',
            'addressee': 'До кого звертаєтесь',
            'description': 'Опис проблеми',
            'justification': 'Обґрунтування',
            'expected_result': 'Очікуваний результат',
            'attachment': 'Додаткові матеріали (необов\'язково)',
            'is_anonymous': 'Подати анонімно',
        }

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.fields['category'].queryset =
Category.objects.filter(is_active=True)
        self.fields['category'].required = True

    def clean_title(self):
        title = self.cleaned_data.get('title', '').strip()
        if len(title) < 10:
            raise forms.ValidationError('Заголовок повинен містити не менше 10
символів.')
        return title
```

```
    def clean_description(self):
        desc = self.cleaned_data.get('description', '').strip()
        if len(desc) < 50:
            raise forms.ValidationError('Опис повинен містити не менше 50
символів.')
        return desc


class SignatureForm(StyledFormMixin, forms.ModelForm):
    """Форма підписання петиції."""

    class Meta:
        model = Signature
        fields = ['comment', 'is_anonymous']
        widgets = {
            'comment': forms.Textarea(attrs={
                'rows': 2,
                'placeholder': 'Додайте коментар до підпису
(необов\'язково)...',
                'maxlength': 500,
            }),
        }
        labels = {
            'comment': 'Коментар',
            'is_anonymous': 'Підписати анонімно',
        }


class CommentForm(StyledFormMixin, forms.ModelForm):
    """Форма коментаря."""

    class Meta:
        model = Comment
        fields = ['text']
        widgets = {
            'text': forms.Textarea(attrs={
                'rows': 3,
                'placeholder': 'Залиште коментар...',
                'maxlength': 2000,
            }),
        }
        labels = {'text': ''}


class PetitionFilterForm(forms.Form):
    """Форма фільтрації списку петицій."""
    SORT_CHOICES = [
        ('-created_at', 'Найновіші'),
        ('created_at', 'Найстаріші'),
        ('-signatures_count', 'Популярні'),
        ('-views_count', 'Переглядаються'),
        ('collection_deadline', 'Закриваються'),
    ]

    q = forms.CharField(
        required=False,
        widget=forms.TextInput(attrs={
            'placeholder': 'Пошук за назвою або описом...',
            'class': 'form-input search-input',
        })
    )
    category = forms.ModelChoiceField(
        queryset=Category.objects.filter(is_active=True),
```

```
            required=False,
            empty_label='Усі категорії',
            widget=forms.Select(attrs={'class': 'form-select'})
        )
        status = forms.ChoiceField(
            choices=[('', 'Усі статуси')] + Petition.Status.choices,
            required=False,
            widget=forms.Select(attrs={'class': 'form-select'})
        )
        sort = forms.ChoiceField(
            choices=SORT_CHOICES,
            required=False,
            initial='-created_at',
            widget=forms.Select(attrs={'class': 'form-select'})
        )


class ResponseForm(StyledFormMixin, forms.ModelForm):
    """Форма офіційної відповіді на петицію."""

    class Meta:
        model = PetitionResponse
        fields = ['decision', 'text', 'attachment']
        widgets = {
            'text': forms.Textarea(attrs={
                'rows': 8,
                'placeholder': 'Детальна обґрунтована відповідь на петицію...',
            }),
        }
        labels = {
            'decision': 'Рішення',
            'text': 'Текст відповіді',
            'attachment': 'Додаток (необов\'язково)',
        }

    def clean_text(self):
        text = self.cleaned_data.get('text', '').strip()
        if len(text) < 50:
            raise forms.ValidationError('Відповідь повинна містити не менше 50
символів.')
        return text
```

ДОДАТОК И

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Васильченко Максим Сергійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Інформаційна система обробки університетських петицій» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

o [ ] інструменти генеративного ШІ не використовувалися.

o [x] інструменти ШІ (ChatGPT) були використані для:

- [ ] перекладу іншомовних джерел;
- [x] стилістичного редагування та перевірки граматики;
- [x] допомоги у написанні/відлагодженні програмного коду;
- [ ] інше: консультації щодо структури пояснювальної записки та підготовки до захисту.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____ 2026 р. _____ Максим ВАСИЛЬЧЕНКО