МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМЕНІ С. З. ҐЖИЦЬКОГО

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему: **«Розробка інформаційної системи автоматизації вступної кампанії університету»**

Виконав: здобувач освіти групи Іт-61

Спеціальності 126 «Інформаційні системи та технології»

(шифр і назва)

Бунга Володимир Романович

(Прізвище та ініціали)

Керівник: к.е.н., доцент, Шувар Б. І.

(Прізвище та ініціали)

Рецензент: _____

(Прізвище та ініціали)

ДУБЛЯНИ-2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМЕНІ С.З.ҐЖИЦЬКОГО

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Другий (магістерський) рівень вищої освіти
Спеціальність 126 «Інформаційні системи та технології»

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____
_____
«____» _____ 202_ р.

ЗАВДАННЯ
на кваліфікаційну роботу студенту
Бунги Володимира Романовича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка інформаційної системи автоматизації вступної кампанії університету»
   Керівник роботи *к.е.н., доцент, Шувар Богдан Іванович*
   (наук.ступінь, вч. звання, прізвище, ініціали)
   затверджені наказом по університету від 28.02.2025 року №140/к-с.
2. Строк подання студентом роботи 05.12.2025р.
3. Вихідні дані до роботи: законодавча та нормативна база у сфері освіти; вимоги до захисту персональних даних; сучасні технології веб-розробки; мова програмування Python; СУБД PostgreSQL; ORM SQLAlchemy; брокер повідомлень Redis; система фонової обробки задач Celery; засоби контейнеризації Docker; клієнт-серверна архітектура; веб-інтерфейси адміністратора та користувача.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)
   Вступ; 1. Аналіз систем автоматизації вступної кампанії; 2. Обґрунтування і вибір інструментарію; 3. Проєктування і реалізація інформаційної системи; 4. Охорона праці та безпека в надзвичайних ситуаціях; 5. Визначення ефективності системи та перспективи розвитку; Висновки; Список використаних джерел;Додатки.
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових схем та моделей):*Docker-інфраструктури;Сторінка авторизації відповідальних працівників; Інтерфейс публічного застосунку web_form;Інтерфейс панелі web; Інтерфейс адміністративної панелі розробника;Діаграма логіки роботи celery; Фрагмент структури таблиці бази даних users; Інтерфейс керування користувачами;Фрагмент таблиці price.*

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	*Шувар Б. І., доцент кафедри інформаційних технологій*		
4	*Городецький І. М., доцент кафедри інженерної механіки*		
5	*Шувар Б. І., доцент кафедри інформаційних технологій*		

7. Дата видачі завдання 28.02.2025 р.

Календарний план

№ з/п	Назва етапів дипломного проєкту	Терміни виконання етапів роботи	Примітка
1.	*Написання першого розділу*	28.02.2025 – 20.03.2025	
2.	*Виконання другого розділу та аркушів ілюстраційного матеріалу до нього*	21.03.2025 – 14.06.2025	
3.	*Виконання третього, четвертого розділів та аркушів ілюстраційного матеріалу до нього*	15.06.2025 – 10.07.2025	
4.	*Написання розділу «Охорона праці»*	11.07.2025 – 31.08.2025	
5.	*Завершення оформлення розрахункового-пояснювальної записки та аркушів ілюстраційного матеріалу*	01.09.2025 – 31.10.2025	
6.	*Завершення роботи цілому*	01.11.2025 – 08.12.2025	

Студент _____, Бунга В.Р.
(підпис) (прізвище, ініціали)

Керівник роботи _____, Шувар Б.І.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

УДК 004.4:004.738.5:378

Магістерська кваліфікаційна робота присвячена розробці веб-орієнтованої інформаційної системи автоматизації вступної кампанії ЗВО. Робота містить 64 сторінку, 10 рисунків, 2 таблиці, 35 джерел та 3 додатки.

Ключові слова: автоматизація вступної кампанії, інформаційна система, Flask, Celery, PostgreSQL, Docker, захист персональних даних.

Метою роботи є створення програмного комплексу, що забезпечує цифровізацію основних процесів вступу, мінімізує кількість ручних операцій та підвищує точність обробки персональних даних абітурієнтів.

Об’єкт дослідження — процес інформаційного забезпечення діяльності приймальної комісії університету, зокрема організація збору, перевірки й обробки даних абітурієнтів, формування документів та управління внутрішніми робочими процедурами в умовах вступної кампанії.

Предмет дослідження – методи та програмні засоби побудови інтегрованої веб-системи для автоматизації вступної кампанії в університеті.

У роботі проаналізовано сучасні платформи автоматизації вступу, визначено функціональні та безпекові вимоги та обґрунтовано вибір технологічного стеку. Створено архітектуру системи на основі Flask, PostgreSQL, Redis і Celery; реалізовано модулі автентифікації, управління заявами, генерації документів і формування звітності. Запроваджено механізми рольового доступу, серверної і клієнтської валідації, а також інструменти захисту персональних даних.

Практичне значення роботи полягає у впровадженні програмної платформи, яка суттєво скорочує час опрацювання вступних заяв та істотно зменшує кількість помилок у документах порівняно з традиційними ручними процедурами. Результати апробовано в тестовому середовищі приймальної комісії, а розроблена система може бути адаптована для використання іншими закладами вищої освіти України.

ABSTRACT

UDC 004.4:004.738.5:378

The master's thesis is devoted to the development of a web-oriented information system for automating the university admission campaign. The work consists of 64 pages, 10 figures, 2 tables, 35 references, and 3 appendices. Keywords: admission campaign automation, information system, Flask, Celery, PostgreSQL, Docker, personal data protection.

The aim of the study is to create a software complex that digitalizes the core admission processes, minimizes the number of manual operations, and increases the accuracy of processing applicants' personal data.

The object of the study is the information support processes of a university admission committee, particularly the organization of collecting, verifying, and processing applicant data, generating documents, and managing internal workflows during the admission campaign.

The subject of the study is the methods and software tools for building an integrated web system for automating the university admission campaign.

The thesis analyzes modern admission automation platforms, identifies functional and security requirements, and substantiates the choice of the technological stack. The system architecture based on Flask, PostgreSQL, Redis, and Celery was designed; modules for authentication, application management, document generation, and reporting were implemented. Mechanisms of role-based access control, server-side and client-side validation, as well as personal data protection tools, were introduced.

The practical significance of the work lies in the implementation of a software platform that significantly reduces the time required to process admission applications and substantially decreases the number of document errors compared to traditional manual procedures. The results were tested in a simulated environment of a university admission committee, and the developed system can be adapted for use by other higher education institutions in Ukraine.

ЗМІСТ

ВСТУП

Вступна кампанія для українських закладів вищої освіти є складним і відповідальним процесом, у якому поєднуються велика кількість абітурієнтів, різноманітні типи документів та посилені вимоги до прозорості та точності обробки даних. У пікові періоди члени приймальних комісій змушені опрацьовувати значні обсяги інформації вручну, постійно перевіряти анкетні дані, контролювати коректність документів, готувати договори та формувати звітність відповідно до нормативних вимог. Така робота потребує часу й уважності, створює високе навантаження на персонал і часто супроводжується технічними помилками, які можуть вплинути на подальший освітній процес чи фінансове планування університету.

Інформаційні системи автоматизації вступних процесів дозволяють суттєво оптимізувати ці операції. Вони забезпечують швидке опрацювання заяв, постійний моніторинг етапів подання документів, автоматичне формування звітів та зменшення впливу людського фактора. Незважаючи на те, що на ринку вже існують платформи для роботи зі вступними кампаніями - такі як ЄДЕБО, DreamApply, UniAssist чи UCAS - університети часто стикаються з тим, що готові рішення не враховують специфіку українського освітнього середовища, особливості локальних нормативних вимог, внутрішні бізнес-процеси та потребу в адаптації документів. У таких умовах виникає необхідність створення власної гнучкої інформаційної системи, здатної підлаштовуватися під структуру та потреби конкретного закладу освіти.

Метою цієї магістерської роботи є розроблення та впровадження веб-орієнтованої інформаційної системи, яка автоматизує ключові етапи вступної кампанії: від подання анкетних даних абітурієнтів і первинної перевірки інформації до формування договорів, архівації документів та простої добудови модулів для потреб, що виникають в процесі роботи. У межах дослідження визначено технологічні вимоги, обґрунтовано вибір програмного стеку, розроблено архітектуру системи, створено модель бази даних та механізми

асинхронної обробки задач. Окрему увагу приділено питанням захисту персональних даних і відповідності нормативним вимогам, адже система працює з чутливою інформацією та повинна гарантувати її конфіденційність і цілісність.

Об’єктом дослідження є процес інформаційної підтримки вступної кампанії закладу вищої освіти. Предметом дослідження - методи та програмні засоби створення веб-орієнтованої системи для автоматизації вступних процедур. У роботі застосовано методи аналізу та синтезу програмних архітектур, структурне моделювання бізнес-процесів, підходи до проєктування баз даних, а також технології контейнеризації та методи оцінювання продуктивності.

Наукова новизна дослідження полягає у створенні комплексної архітектури, що поєднує можливості Flask, Celery та Docker для побудови сучасної, гнучкої та масштабованої системи, яка охоплює як фронт для абітурієнтів, так і інструменти автоматизації рутинних процесів для приймальної комісії. Практичне значення роботи полягає у тому, що запропонована система може бути впроваджена в університетських приймальних комісіях і забезпечити суттєве скорочення часу обробки заяв, зменшення кількості помилок, підвищення прозорості процесів та створення надійного підґрунтя для подальшої інтеграції з державними платформами, CRM-системами й аналітичними модулями прогнозування рейтингу абітурієнтів.

РОЗДІЛ 1.
АНАЛІЗ СИСТЕМ АВТОМАТИЗАЦІЇ ВСТУПНОЇ КАМПАНІЇ

1.1. Теоретичні основи інформаційних систем в освіті

Інформаційні системи в освіті - це поєднання програмних, апаратних та організаційних інструментів, які забезпечують збирання, зберігання, опрацювання й передачу даних, необхідних для ефективної роботи університету. У сучасних умовах цифрової трансформації такі системи перестають бути допоміжними і фактично перетворюються на елемент критичної інфраструктури. Вони підтримують прийняття управлінських рішень, налагоджують комунікацію з абітурієнтами, студентами й випускниками, забезпечують відповідність нормативним вимогам і інтегруються з державними реєстрами. Насправді їхнє призначення значно ширше, ніж проста автоматизація документообігу: інформаційні системи створюють фундамент для стратегічного аналізу, управління освітніми процесами та формування конкурентних переваг закладу вищої освіти.

Традиційно інформаційні системи описують через три взаємопов’язані складові: дані, процеси та технології. Дані в освітньому середовищі охоплюють не лише анкетну інформацію про абітурієнтів, студентів чи викладачів, а й результати навчання, структуру освітніх програм, розклади, рейтинги, підсумки акредитацій. Процеси відображають ключові етапи життєвого циклу студента - від вступу й зарахування до атестації та випуску - а також супровідні функції, як-от кадровий облік, фінансові операції, підготовка звітності, взаємодія зі стейкхолдерами. Технологічна складова включає платформи та сервіси, канали доступу, інструменти інтеграції та аналітичні алгоритми. Рівень взаємоузгодженості цих елементів визначає ступінь цифрової зрілості університету.

Існує кілька підходів до класифікації освітніх інформаційних систем, проте найчастіше їх поділяють за функціональним призначенням. Адміністративні системи охоплюють управління кадрами, бухгалтерією, документообігом.

Навчальні середовища (LMS) забезпечують дистанційне навчання і цифрові курси. Аналітичні модулі (BI-системи) дають змогу створювати звіти та прогнозні моделі. Окреме місце займають спеціалізовані рішення, наприклад електронний деканат, модуль формування розкладів або системи подання заяв. Важливу роль відіграють інтегровані ERP-рішення та національні платформи на кшталт Єдиної державної електронної бази з питань освіти (ЄДЕБО), яка забезпечує міжвідомчу взаємодію й централізований облік.

Життєвий цикл інформаційних систем в освіті включає аналіз вимог, проєктування, розробку, тестування, впровадження та подальший супровід. Особливістю освітнього домену є циклічність процесів - навчальний рік, вступна кампанія, періоди акредитацій - що потребує чітких графіків релізів і безперервності сервісів. Крім того, нормативна база регулярно оновлюється, з’являються нові вимоги до структури даних, форм звітності, відкритих даних і прозорості. Тому система повинна мати можливість розвиватися без збоїв у роботі та підтримувати внесення змін без ризику втрати даних або порушення доступності сервісів.

Автоматизація відіграє особливо важливу роль у трьох напрямах. Перший - підвищення операційної ефективності, що включає скорочення часу на опрацювання документів, зменшення кількості ручних операцій і технічних помилок, усунення дублювання даних. Другий - забезпечення прозорості та контролю: можливість відстежувати статуси заяв, формувати аналітичні звіти, контролювати доступ до персональних даних. Третій - покращення якості сервісу для абітурієнтів і студентів шляхом надання цілодобового доступу до послуг, автоматичних сповіщень, можливості дистанційного подання документів і зручних цифрових каналів комунікації. У випадку вступної кампанії, яка виступає «першою точкою контакту» абітурієнта з університетом, ефективність автоматизації безпосередньо впливає на репутацію закладу та виконання планових показників набору.

Окрему увагу необхідно приділяти питанням інформаційної безпеки. Системи, що працюють у сфері освіти, оперують великою кількістю чутливих

персональних даних - паспортними відомостями, контактною інформацією, ідентифікаційними кодами, медичними довідками. Законодавство України, нормативні акти МОН та рекомендації міжнародних організацій, зокрема ENQA, накладають вимоги щодо забезпечення конфіденційності, цілісності й доступності інформації. Це означає, що система повинна мати чіткі політики доступу, підтримувати шифрування, журналювання, авторизацію за ролями, а також механізми контролю ризиків. В архітектурі такі вимоги проявляються через необхідність розділення компонентів, захищені канали передачі даних та продуману модель прав доступу [16;29].

Таким чином, теоретичні засади інформаційних систем в освіті створюють методологічну основу для подальшої розробки рішень з автоматизації вступної кампанії. Розуміння ключових понять, класифікацій, життєвого циклу та нормативного поля дає змогу правильно сформулювати вимоги до майбутньої системи, визначити критерії її ефективності та забезпечити відповідність очікуванням основних користувачів - від працівників приймальної комісії до керівництва університету й державних регуляторів [12].

1.2. Аналіз існуючих рішень автоматизації вступних кампаній

Огляд сучасних платформ для подання вступних заяв дає змогу окреслити реальні потреби університетів та визначити функціональні прогалини, які доцільно закрити власною системою автоматизації. Для аналізу було обрано чотири різні за призначенням і масштабом рішення - державну платформу ЄДЕБО та міжнародні системи DreamApply, UniAssist і UCAS. Кожна з них представляє окремий підхід до організації вступних процесів і демонструє, що саме варто врахувати під час розробки локальної системи університету.

ЄДЕБО виступає центральним елементом української освітньої інфраструктури та забезпечує офіційний облік заяв, рейтингів і наказів про зарахування. Її головними перевагами є централізована валідація документів, дотримання квот, використання результатів НМТ/ЗНО та прозорість для

регулятора. Разом з тим, ЄДЕБО не охоплює внутрішні процеси університету: вона не має інструментів для створення гнучких форм, персоналізованих повідомлень чи автоматичної генерації локальних документів, які є важливими для щоденної роботи приймальної комісії. Саме тому в межах цієї магістерської роботи система розглядається не як альтернатива ЄДЕБО, а як внутрішній робочий інструмент, який допомагає підготувати дані для внесення до державної бази.

Платформа DreamApply представляє сегмент комерційних SaaS-рішень, орієнтованих переважно на міжнародних абітурієнтів. Вона пропонує зручні форми заяв, підтримує кілька мов, має вбудовані засоби комунікації, аналітику та інтеграцію з платіжними сервісами. Це справді комплексний інструмент, який можна адаптувати під різні стратегії набору [36]. Проте для українських університетів DreamApply має низку недоліків: високу вартість, залежність від зовнішньої інфраструктури та необхідність самостійно адаптувати шаблони документів до національних стандартів. Цей приклад підкреслює важливість гнучких форм, налаштовуваних бізнес-процесів і розвиненої аналітики - саме ці елементи враховано при проектуванні власної системи.

UniAssist - централізована служба, яка обробляє заявки іноземних студентів для університетів Німеччини. Сильними сторонами є стандартизована оцінка сертифікатів, перевірка легалізації документів і можливість подавати заяви одразу до багатьох університетів [37]. Водночас система майже повністю прив’язана до німецького законодавства, потребує значного обсягу ручної перевірки та майже не піддається адаптації. Для українського контексту UniAssist цікавий насамперед як приклад якісного модуля валідації документів та експертної перевірки, елементи якого частково можуть бути реалізовані у вигляді чек-листів та механізмів контролю статусів у межах локальної інформаційної системи.

Система UCAS, яка координує вступ до університетів Великої Британії, демонструє переваги централізованого підходу: уніфіковані дедлайни, чітка логіка подання заяв, інтеграція з тестовими організаціями та потужні аналітичні

інструменти [38]. Водночас UCAS характеризується високим рівнем стандартизації й має обмежену здатність до адаптації під індивідуальні потреби конкретного закладу поза рамками загальнонаціональних правил. Це показує, що централізованість підвищує прозорість, але істотно знижує гнучкість, що для українського університету є критичним фактором.

Таблиця 1.1 – Порівняльна таблиця існуючих систем

DreamApply	UniAssist	UCAS	Власна система
+ Зручні багатомовні форми + Аналітика, інтеграція з оплатами + SaaS-підхід	+ Стандартизо-вана перевірка документів +Централізована подача + Валідація сертифікатів	+ Централізовані дедлайни + Прозора логіка подання + Потужна аналітика	+ Гнучкі бізнес-процеси + Локальна робота з документами + Асинхронна генерація файлів
– Висока вартість – Залежність від зовнішнього сервісу – Потрібна адаптація документів	– Неадаптована до інших країн – Багато ручних перевірок – Низька гнучкість	– Висока стандартизованість – Без гнучкої адаптації – Залежність від правил	–Налаштовувані ролі та права – Внутрішня звітність – Потребує локальної підтримки

Проведений аналіз засвідчує, що готові рішення вирішують різні завдання, однак жодне з них не здатне повністю охопити специфіку внутрішніх процесів окремого закладу вищої освіти. Для українського університету важливим є поєднання відповідності нормативним вимогам із можливістю швидко адаптувати бізнес-процеси під свою структуру, документообіг та особливості вступної кампанії. Саме тому розроблювана інформаційна система зосереджена не на дублюванні функцій ЄДЕБО, а на створенні гнучкого внутрішнього середовища для роботи приймальної комісії: керування документами, асинхронна генерація файлів, система ролей і прав доступу, організація черг задач та формування локальної звітності. Вона доповнює державні інструменти й водночас забезпечує ефективне функціонування університету під час вступної кампанії.

1.3. Архітектурні підходи до веб-систем

Архітектура веб-системи визначає те, наскільки впевнено вона працюватиме під навантаженням, як швидко адаптуватиметься до змін у нормативній базі та чи зможе розширюватися разом із зростанням потреб приймальної комісії. У практиці розробки найчастіше розглядають три підходи: класичний моноліт, мікросервісну архітектуру та проміжні гібридні моделі, такі як модульний моноліт або сервісно-орієнтована архітектура (SOA). Порівняння цих підходів дозволяє обґрунтовано вибрати структурну основу для системи автоматизації вступної кампанії.

Монолітний застосунок являє собою єдиний програмний блок, у межах якого працюють усі функції - від автентифікації й обробки анкет до управління документами та аналітичних панелей. Це підхід із простою структурою, швидким розгортанням і мінімальними вимогами до інфраструктури. Він добре підходить для перших версій продукту або невеликих команд. Проте з часом моноліт стає важчим у підтримці: кожен реліз уповільнюється, модулі дедалі більше залежать одне від одного, тестування займає більше часу, а масштабування можливе переважно вертикально. Для системи вступної кампанії це означає ризики під час пікових навантажень, коли кількість поданих заяв різко зростає.

Мікросервісна архітектура натомість розділяє систему на низку незалежних сервісів: окремо працюють модулі керування користувачами, анкетами, документами, звітністю тощо. Кожен сервіс має власний життєвий цикл, може масштабуватися горизонтально та оновлюватися окремо від інших. Такий підхід підвищує відмовостійкість і дає змогу прискорити розвиток продукту, особливо у великих командах. Водночас він суттєво ускладнює DevOps-процеси: потрібні сервіс-дискверері, API-шлюзи, централізовані логування й моніторинг, а також розв'язання питань міжсервісних транзакцій.

Для університету, який не має великої технічної команди, така модель може виявитися надмірно складною на старті.

Модульний моноліт займає проміжну позицію між двома крайнощами. Застосунок залишається єдиним із точки зору розгортання, але його внутрішня структура поділена на незалежні модулі з чіткими межами відповідальності. Це дозволяє уникнути хаотичних залежностей, упорядковує розвиток коду та знижує ризики «монолітного зростання». Найресурсніші завдання - наприклад, генерація документів або масовий експорт - можуть виконуватися у фонових процесах, не перевантажуючи основний веб-інтерфейс. Модульний моноліт добре підходить для проєктів, які повинні залишатися гнучкими, але не мають ресурсів для повної мікросервісної інфраструктури.

Вибір Flask як основного фреймворку також є усвідомленим. Flask забезпечує мінімалістичну, але гнучку основу для створення REST-інтерфейсів, легко інтегрується з розширеннями (Flask-Login, Flask-Migrate, Flask-Bcrypt) і дозволяє вільно формувати архітектуру відповідно до потреб проєкту. Альтернативи також розглядалися, проте мали певні обмеження: Django може бути надто громіздким для подвійної структури веб-додатків; FastAPI потребує глибшої інтеграції з шаблонами складніших форм; Node.js створив би необхідність підтримувати два різні технологічні стеки [5].

У стратегічній перспективі модульний моноліт із Celery закладає фундамент, який дозволяє впевнено розвивати систему. За потреби окремі компоненти можна поступово відокремлювати у мікросервіси, не переписуючи весь продукт. Важливо лише дотримуватися принципів низької зв’язності, документувати внутрішні API та забезпечувати централізоване логування й моніторинг. Така архітектурна модель є оптимальною для університету: вона не вимагає надмірних інвестицій на початковому етапі, але забезпечує можливість масштабування відповідно до реальних викликів цифрової трансформації [9].

1.4. Вимоги до безпеки персональних даних у системах автоматизації вступної кампанії

Автоматизація вступної кампанії передбачає роботу з великим обсягом персональних даних абітурієнтів: паспортною інформацією, сканами документів, контактними відомостями, результатами освітніх документів, медичними довідками та даними про батьків або законних представників. Через це система повинна відповідати підвищеним вимогам до захисту інформації, які визначаються українським законодавством та міжнародними стандартами. Основними нормативними документами є Закони України «Про захист персональних даних», «Про інформацію», «Про електронні довірчі послуги», а також регуляторні акти МОН та Міністерства цифрової трансформації. У випадку роботи з іноземними абітурієнтами можуть застосовуватися положення GDPR. Порушення вимог безпеки може спричинити адміністративні санкції, зупинку роботи системи та репутаційні втрати. [1;2].

Принцип «безпека за дизайном» має бути закладений у всі етапи створення системи - від формування вимог до супроводу. Важливою складовою є політики доступу, що реалізуються через рольову модель (RBAC). Вона дозволяє чітко розмежувати права абітурієнтів, операторів, адміністраторів і технічної підтримки, забезпечуючи принцип мінімальних привілеїв. Для критичних ролей доцільним є використання двофакторної автентифікації та контроль активних сесій.

Шифрування є ключовим механізмом захисту персональних даних. Усі взаємодії між веб-інтерфейсами, адміністративною панеллю, фоновими сервісами та базою даних повинні здійснюватися через захищені канали (HTTPS/TLS). Чутливі дані - паспортні номери, адреси, персональні ідентифікатори - варто додатково шифрувати або хешувати. Паролі мають зберігатися виключно в хешованому вигляді, а секретні ключі - передаватися через механізми середовищ виконання або системи керування секретами.

Система повинна мати надійний механізм резервного копіювання. Доцільно використовувати багаторівневу схему: щоденні копії, тижневі повні резерви та довгострокові архіви. Всі копії зберігаються у зашифрованому вигляді, а процес відновлення даних має регулярно тестуватися згідно з планом аварійного відновлення.

Особливу увагу необхідно приділити роботі з документами. Оскільки система генерує договори та інші офіційні файли, тимчасові сховища повинні автоматично очищуватися після використання. Завантаження документів повинне включати перевірку безпеки файлів, обмеження розміру та формату, а посилання на документи мають бути короткостроковими та доступними лише авторизованим користувачам.

Також важливо забезпечити юридичну коректність отримання згоди на обробку персональних даних. Система має фіксувати момент надання згоди та можливість її відкликання. Для неповнолітніх потрібна окрема згода батьків або законних представників. Дотримання вимог щодо локалізації даних є обов’язковим: інформація має зберігатися в межах України або у середовищах, погоджених регулятором.

Під час тестування та розробки заборонено використовувати реальні персональні дані. Для QA-середовища створюються анонімізовані або синтетичні набори даних. Команда має дотримуватися практик безпечного кодування (OWASP Top 10), контролювати залежності та використовувати інструменти статичного та динамічного аналізу безпеки.

Сукупність цих вимог формує багаторівневу модель захисту, що охоплює технічні та організаційні аспекти. Для університету важливо не лише впровадити механізми безпеки, а й документувати політики, проводити навчання персоналу та здійснювати регулярні перевірки. Лише комплексний підхід забезпечить надійний захист персональних даних і збереже довіру абітурієнтів.

1.5. Постановка задачі

На основі проведеного аналізу архітектурних підходів, вимог до безпеки персональних даних та особливостей роботи приймальної комісії сформульовано мету та комплекс задач, які має вирішувати інформаційна система автоматизації вступної кампанії. Метою проєкту є створення гнучкого веб-орієнтованого програмного комплексу, який забезпечує цифровізацію основних процесів вступу, спрощує взаємодію між абітурієнтами та університетом, гарантує надійний захист персональних даних і надає працівникам приймальної комісії інструменти для ефективної щоденної роботи.

Для досягнення цієї мети система повинна реалізувати низку взаємопов’язаних функціональних і технічних завдань. Одним із ключових рішень є поділ системи на два веб-додатки. Перший - адміністративний інтерфейс для співробітників приймальної комісії - відповідає за автентифікацію користувачів, управління даними абітурієнтів, налаштування шаблонів документів, роботу з довідниками, перегляд статусів заяв та формування звітності. Другий застосунок призначений для абітурієнтів і забезпечує покрокове заповнення анкет, валідацію введених даних, завантаження документів та супровід користувача під час усього процесу подання заяви. Обидві частини системи працюють із єдиною моделлю даних, але суттєво відрізняються за логікою доступу й інтерфейсом.

Окремим завданням є організація фонової обробки ресурсомістких операцій - формування персоналізованих документів, пакетної генерації файлів чи експорту великих звітів. Для цього використовується окремий сервіс фонового виконання, який приймає задачі від веб-додатків, опрацьовує їх у черзі та повертає готові результати. Такий підхід зменшує навантаження на інтерфейс, запобігає затримкам для користувачів та дозволяє масштабувати систему за потреби.

Центральним сховищем даних обрано PostgreSQL, що забезпечує цілісність, транзакційність і можливість подальшого розширення. Модель даних

охоплює профілі користувачів, записи абітурієнтів, довідники, інформацію про шаблони документів і тарифи. Використання ORM дає змогу підтримувати цілісність логіки та виконувати міграції схеми без ризиків. У разі потреби окремі поля можуть зберігатися у зашифрованому вигляді.

Система повинна підтримувати генерацію персоналізованих документів - договорів, заяв, довідок - на основі шаблонів, а також формування табличних звітів у форматах Excel або CSV. Для цього передбачено керування бібліотекою шаблонів, механізм підготовки даних для підстановки, обробку сценаріїв масової генерації та контроль тимчасових сховищ.

Важливим завданням є реалізація комплексної системи автентифікації та авторизації. Адміністративний застосунок має підтримувати рольову модель доступу, хешування паролів, контроль критичних операцій і первинну ініціалізацію доступів. Публічний портал повинен залишатися легким і швидким, використовуючи кешування довідників та автоматичне оновлення кешу після змін у базі даних.

Положення щодо безпеки трансформуються у конкретні технічні вимоги: шифрування з’єднань, захист конфігураційних ключів, ведення журналів дій, очищення тимчасових файлів, використання контейнеризації та ізоляції сервісів. Окремою задачею є підготовка інструкцій та конфігурацій для розгортання системи у контейнерному середовищі Docker, що забезпечує повторюваність, портативність і можливість масштабування окремих компонентів - веб-застосунків, фонового сервісу та брокера черг.

Сформульована постановка задачі демонструє, що майбутня інформаційна система повинна не лише замінювати традиційні паперові процедури, а й створювати цілісну цифрову екосистему вступної кампанії - від реєстрації та перевірки даних до генерації документів, формування звітів і підтримки управлінської аналітики. Реалізація зазначених завдань забезпечує основу для подальшого розвитку системи, інтеграції з зовнішніми сервісами та масштабування відповідно до потреб університету.

РОЗДІЛ 2.
ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАРІЮ ТА ТЕХНОЛОГІЙ

2.1. Вибір мови програмування та веб-фреймворку

Вибір мови програмування та веб-фреймворку є одним із фундаментальних етапів проектування інформаційної системи, оскільки від нього залежить не лише швидкість початкової розробки, але й подальша підтримка, можливість масштабування та адаптації рішення до змін нормативного середовища. Для системи автоматизації вступної кампанії, де навантаження зростає нерівномірно, а вимоги та форми документів можуть змінюватися щороку, технологічний стек стає стратегічним фактором успішності проекту.

Python інтерпретована об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією. Розроблена на початку 1990-х років Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним [13].

Основною мовою програмування у даній роботі обрано Python версії 3.13. Python поєднує простоту синтаксису з високою виразністю та розвиненою екосистемою бібліотек. Це важливо для освітньої сфери, де значна частина функціональності пов’язана з обробкою документів, формуванням звітів, роботою з табличними даними та інтеграцією з іншими сервісами. Наявність стабільних бібліотек (python-docx, pandas, sqlalchemy, інструменти для асинхронних задач) дозволяє зосередитися на бізнес-логіці системи, а не на низькорівневій реалізації технічних деталей. Крім того, Python широко використовується у навчальному процесі, що спрощує залучення студентів і молодих фахівців до підтримки та розвитку системи в майбутньому.

Важливою перевагою Python є легкість супроводу. Університетські IT-підрозділи часто працюють в умовах обмежених людських ресурсів, а зміна спеціалістів є звичним явищем. Читабельність Python-коду, підтримка кількох парадигм програмування та велика кількість навчальних матеріалів значно

полегшують передавання проекту між розробниками та зменшують ризик виникнення помилок під час модернізації.

У ролі веб-фреймворку обрано Flask - мінімалістичний мікрофреймворк, який надає базовий каркас для побудови веб-застосунків та дозволяє гнучко формувати структуру проекту залежно від потреб. На відміну від «важких» фреймворків, Flask не нав’язує суворої архітектури, що є перевагою для системи, яка складається з двох окремих веб-додатків: адміністративної панелі та публічного інтерфейсу для абітурієнтів. Обидві частини можуть розвиватися незалежно, але залишаються в межах єдиної технологічної платформи. [5;8].

Flask добре інтегрується з ORM-рішеннями (SQLAlchemy), засобами міграцій (Flask-Migrate), механізмами авторизації (Flask-Login, Flask-Bcrypt), а також з фреймворками для побудови асинхронних черг, такими як Celery [11]. Це дозволяє створити модульний моноліт, де логіка поділена на окремі блоки, але система розгортається як цілісний застосунок. Найресурсніші операції - генерація документів, масове формування архівів, створення звітів - можуть виконуватися у фонових процесах без шкоди для роботи веб-інтерфейсу.

Під час вибору фреймворку було розглянуто й альтернативи. Django пропонує значно більшу кількість функцій «із коробки», включно з адміністративною панеллю та вбудованим ORM, що корисно для типових CRUD-завдань. Проте його жорстка структура та фіксована архітектура могли б ускладнити реалізацію двох незалежних веб-інтерфейсів і специфічних багатокрокових анкет. FastAPI, своєю чергою, забезпечує високу продуктивність і нативну підтримку асинхронності, проте потребує окремих рішень для побудови інтерфейсу та може ускладнити розробку в умовах обмежених ресурсів.

Розглядалося також використання Node.js, яке є привабливим завдяки природній асинхронності та великій екосистемі JavaScript. Однак для системи, що активно працює з документами DOCX, Excel-звітом та має потенціал для інтеграції з аналітичними інструментами Python, це створило б необхідність підтримувати два різні стеки, що підвищує вартість супроводу.

У підсумку комбінація Python 3.13 та Flask забезпечує оптимальний баланс між гнучкістю, простотою, можливістю розвитку та відповідністю вимогам проєкту. Обраний стек добре інтегрується з іншими ключовими компонентами системи - PostgreSQL, Celery, Redis і Docker - та створює цілісну технологічну платформу, придатну як для навчального використання, так і для реального розгортання у закладах вищої освіти.

2.2. Ізольована система управління базою даних

Ефективність інформаційної системи автоматизації вступної кампанії значною мірою визначається тим, наскільки надійно і гнучко організоване зберігання даних. Система має оперувати анкетними відомостями абітурієнтів, історією поданих заяв, довідниками спеціальностей і освітніх програм, тарифами на навчання, службовими записами про користувачів тощо. Усі ці об’єкти даних пов’язані між собою відношеннями «один-до-багатьох» та «багато-до-одного», вимагають забезпечення цілісності, підтримки транзакцій і можливості формувати складні вибірки. З огляду на це як основну систему управління базами даних було обрано PostgreSQL, а для взаємодії з нею на рівні коду - ORM-шар на базі SQLAlchemy у поєднанні з інструментами Flask-Migrate/Alembic.

PostgreSQL - це потужна об’єктно-реляційна СУБД з відкритим кодом, яка вже багато років використовується як у комерційних, так і в академічних системах. Її головною перевагою для освітньої системи є надійна підтримка транзакцій, реалізована згідно з ACID-властивостями, що гарантує цілісність даних навіть у разі збоїв обладнання або помилок у застосунку. Під час вступної кампанії, коли велика кількість користувачів одночасно вносить і оновлює записи, це особливо критично: операції додавання нових анкет, редагування даних, формування наказів і фінансових документів повинні виконуватися атомарно, без «висячих» або частково застосованих змін. PostgreSQL забезпечує необхідні механізми блокувань, ізоляції транзакцій і журналювання, що дозволяє проводити аудит змін і відновлювати систему до узгодженого стану.

Окремою сильною стороною PostgreSQL є розвинена підтримка складних типів даних та розширень. Для системи вступної кампанії це відкриває можливості зберігання структурованих даних у форматі JSONB (наприклад, для історії статусів заяв чи логів дій користувачів), використання обмежень цілісності для контролю коректності посилань між таблицями (таблиця з анкетами, таблиця з цінами, таблиця з користувачами). Наявність потужної мови запитів SQL, підтримка представлень (view), агрегувальних та віконних функцій дозволяють формувати гнучкі вибірки для аналітики: від простих списків зарахованих до деталізованих звітів за напрямами підготовки, формами навчання, джерелами фінансування.

Ще одним важливим чинником є стабільність і зрілість PostgreSQL, а також широка підтримка у спільноті розробників. Це означає, що університет, який впроваджує систему, не залежить від комерційної ліцензійної політики окремого вендора, може розгортати базу як на власних серверах, так і в хмарних середовищах, а також користуватися великою кількістю інструментів резервного копіювання, моніторингу та налаштування продуктивності. Для освітніх установ, які іноді мають обмежений бюджет на ПЗ, це є суттєвою перевагою.

Для взаємодії між застосунками на Flask та PostgreSQL у даному проєкті використано ORM-бібліотеку SQLAlchemy. Об’єктно-реляційне відображення (ORM) дозволяє розробникові працювати з даними у вигляді класів і об’єктів мови програмування, не будуючи вручну SQL-запити для кожної операції. Наприклад, сутності `User`, `Price`, `Student` описуються як Python-класи з полями, що відображаються на стовпці відповідних таблиць. Це спрощує логіку коду: замість роботи з рядками SQL, розробник оперує методами створення, оновлення, видалення й вибірки об’єктів. Такий підхід зменшує кількість помилок, пов’язаних із синтаксисом запитів, покращує читабельність і дозволяє легко рефакторити структуру даних.

Крім зручності, ORM-шар полегшує підтримку узгодженості між двома веб-додатками та фоновими воркерами Celery. Коли моделі даних описані в одному місці й підключаються до різних компонентів системи, ризик

розбіжностей у схемі значно нижчий. Якщо, наприклад, додається нове поле до таблиці студентів (додаткова характетика абітурієнта або службова ознака), достатньо оновити модель у коді та виконати міграцію, і всі частини системи отримають доступ до нового атрибута в єдиному форматі.

Фактично, SQLAlchemy виступає проміжною ланкою між об'єктним світом Python і реляційною природою PostgreSQL. Він надає механізми побудови складних запитів засобами мови (через вирази, фільтри, об'єднання), водночас залишаючи можливість виконувати «чистий» SQL там, де це необхідно з міркувань продуктивності. Це особливо актуально в аналітичних запитах, де потрібно, наприклад, згрупувати записи за спеціальністю та роком вступу, порахувати кількість абітурієнтів, виділити окремо бюджетну й контрактну форми.

Щоби керувати еволюцією схеми бази даних, у проєкті використано зв'язку Flask-Migrate та Alembic. Alembic - це інструмент міграцій для SQLAlchemy, що дозволяє фіксувати зміни в структурі БД (додавання таблиць, стовпців, індексів) у вигляді окремих скриптів. Flask-Migrate інтегрує Alembic із Flask-застосунком, роблячи процес міграцій більш зручним. Наявність формальної історії міграцій дозволяє команді розробників узгоджено змінювати схему на різних середовищах (development, testing, production), не ризикуючи втратити дані або опинитися в ситуації, коли код і база «не збігаються». У контексті вступної кампанії це означає, що навіть при зміні набору полів анкети (наприклад, додавання нових атрибутів чи довідкових таблиць) система може еволюціонувати плавно, без переривання роботи приймальної комісії.

Варто також відзначити, що використання ORM та міграцій суттєво спрощує задачу резервного копіювання й відновлення. Оскільки структура даних формально описана в коді й підтримується версіонуванням, IT-персонал університету отримує чітке уявлення про те, які саме сутності зберігаються в базі, як вони пов'язані між собою й що саме потрібно врахувати при плануванні схем резервування. Це, у свою чергу, гармонізується з вимогами до захисту персональних даних, описаними в попередньому розділі: стає можливим чітко

визначити, які таблиці містять критично важливу інформацію й потребують додаткового шифрування або контролю доступу.

Таким чином, поєднання PostgreSQL як надійної й потужної СУБД із ORM-шаром на базі SQLAlchemy та інструментами Flask-Migrate/Alembic створює фундамент для стійкого, масштабованого й керованого сховища даних. Воно забезпечує одночасно високу гнучкість розробки, безпеку зберігання, зручність супроводу та відповідність нормативним вимогам, що є критично важливим для інформаційної системи автоматизації вступної кампанії.

У сучасній розробці веб-систем контейнеризація стала стандартом фактичної експлуатації. Застосунки дедалі рідше встановлюють безпосередньо на сервери: натомість вони працюють у контейнерах, які ізолюють середовище виконання та забезпечують однакову поведінку системи незалежно від платформи [6;10]. У межах цієї роботи для розгортання інформаційної системи вступної кампанії використано Docker, а для координації кількох взаємопов’язаних сервісів - Docker Compose. Такий підхід дозволяє запускати веб-додатки, фонові сервіси та допоміжні компоненти як окремі модулі, але в єдиній керованій інфраструктурі.

Головною перевагою контейнеризації є ізоляція. Кожен контейнер містить лише ті залежності, які потрібні конкретному компоненту: фреймворк, бібліотеки, системні утиліти. Завдяки цьому різні частини системи не конфліктують між собою та можуть оновлюватися незалежно. Наприклад, збій у фоновому воркері не впливає на роботу веб-інтерфейсу, а оновлення Python-залежностей у одному контейнері не зачіпає інші сервіси.

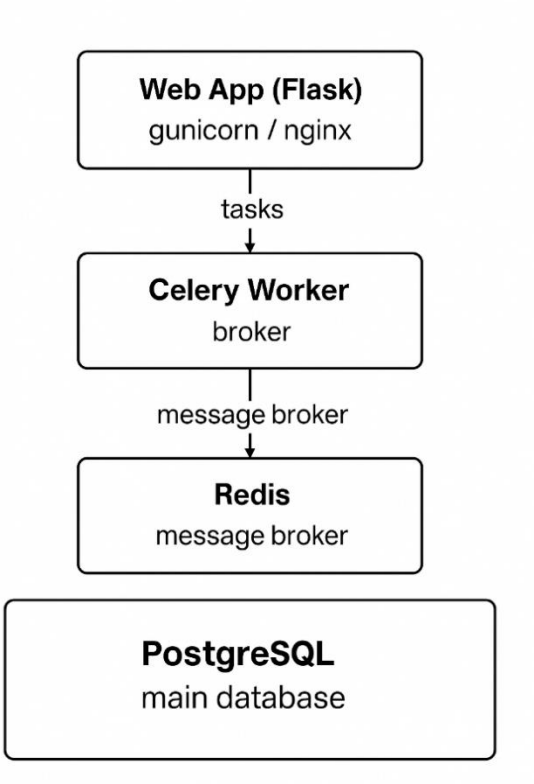

Рис. 2.1 –Docker-інфраструктура.

Dockerfile дозволяє формально описати середовище застосунку: базовий образ, порядок інсталяції залежностей, копіювання коду, налаштування змінних середовища та команду запуску. Це забезпечує повторюваність середовища у розробці, тестуванні. Для даної системи створено окремий образ веб-застосунку та образ воркера, що використовує той самий код, але іншу команду запуску. В результаті всі середовища працюють однаково, а ризики помилок через “різні конфігурації” мінімізуються.

Docker Compose об’єднує контейнери в єдиний комплекс. У конфігурації docker-compose.yml описано сервіси веб-додатка, Celery-воркера, Redis та, за потреби, бази даних або проксі-сервера. Compose автоматично створює внутрішню мережу, через яку сервіси комунікують за іменами контейнерів. Це значно спрощує налаштування з’єднань та позбавляє необхідності прописувати IP-адреси.

Важливою частиною інфраструктури є томи (volumes), які використовуються для зберігання згенерованих документів і звітів. Оскільки система активно формує файли DOCX, ZIP та таблиці, зберігання їх всередині контейнера було б ненадійним - перезапуск призводив би до втрати даних.

Натомість том generated_docs зберігається поза життєвим циклом контейнерів і може одночасно використовуватись веб-додатком та фоновими сервісами. Це забезпечує стабільність роботи та можливість резервування.

Ще одним важливим елементом є мережева ізоляція. Зовнішнім користувачам відкривається лише необхідний порт (наприклад, веб-інтерфейсу), тоді як Redis, внутрішні API чи адміністративні інструменти залишаються недоступними ззовні. За потреби конфігурація легко доповнюється реверс-проксі Nginx, який може забезпечувати HTTPS, балансування навантаження та маршрутизацію.

Docker також спрощує впровадження практик CI/CD. Наявність формалізованого середовища - Dockerfile та Compose - дозволяє автоматизувати збирання образів, тестування, статичний аналіз коду й розгортання нових версій системи. Це особливо важливо у період активної вступної кампанії, коли будь-які простої або помилки конфігурації є критичними.

Для локальної розробки Docker також є значною перевагою: розробнику достатньо запустити одну команду, щоб отримати повністю готове оточення з усіма сервісами. Це усуває потребу вручну встановлювати Redis, PostgreSQL чи створювати окремі середовища Python. Всі учасники проєкту працюють у однакових умовах, що усуває типові проблеми, пов’язані з різницею конфігурацій.

Таким чином, використання Docker і Docker Compose створює надійний фундамент для функціонування та розвитку інформаційної системи. Контейнеризація забезпечує ізоляцію сервісів, передбачувані середовища, стійке зберігання документів і можливість ефективного масштабування. Це рішення сприяє стабільності системи та дає змогу впевнено розширювати її функціонал у майбутньому.

2.3. Асинхронна обробка з використанням Celery та Redis

У процесі роботи інформаційної системи автоматизації вступної кампанії виникає цілий клас операцій, які є ресурсоємними за часом та обчисленнями, але при цьому не потребують миттєвої взаємодії з користувачем. До таких операцій належать генерація великої кількості персоналізованих документів (договорів, заяв, додаткових форм), формування ZIP-архівів із пакетом файлів, експорт даних до форматів XLSX або CSV, а також потенційно - розсилання електронних листів із повідомленнями про статус заяви. Виконання цих задач безпосередньо в обробнику HTTP-запиту призводило б до тривалих затримок у роботі веб-інтерфейсу: користувач змушений був би чекати завершення довгої операції, а сервер - утримувати відкритим з'єднання. Для системи, яка обслуговує десятки або сотні одночасних сеансів у пікові моменти вступної кампанії, така поведінка є неприйнятною. Тому в межах даного проекту застосовано підхід асинхронної обробки на основі зв'язки Celery та Redis.

Celery є фреймворком для організації фонового виконання задач на Python. Він надає механізм створення черг завдань, де кожне завдання описується як окрема функція, що може бути викликана не лише безпосередньо, а й через відкладене виконання. Коли веб-застосунок приймає від користувача запит на виконання тривалої операції, замість того, щоб обробляти її синхронно, він ставить відповідне завдання у чергу Celery та негайно повертає користувачеві відповідь із ідентифікатором задачі. Далі спеціальний процес-воркер Celery, запущений окремо від основного веб-сервера, отримує завдання з черги й виконує його у фоновому режимі. Такий підхід дозволяє розвантажити веб-шар, уникнути блокувань і рівномірно розподілити навантаження між кількома воркерами.

Redis у цій зв'язці використовується як брокер повідомлень та, за потреби, сховище результатів. Redis - це високошвидкісна система зберігання даних у пам'яті, яка підтримує різноманітні структури (списки, множини, хеші) й має низький час доступу. У контексті Celery Redis виступає посередником, через

якого веб-застосунок та воркери обмінюються інформацією про задачі: веб-частина додає повідомлення у відповідну чергу, а воркери читають їх у момент готовності до обробки. Така організація дозволяє легко масштабувати систему: за необхідності можна запустити декілька екземплярів воркерів Celery, які паралельно оброблятимуть задачі із загальної черги Redis, не вимагаючи змін у коді веб-додатка.

Важливою особливістю такого підходу є можливість відстежувати стан виконання задач. Кожній задачі Celery призначає унікальний ідентифікатор, за допомогою якого можна отримати інформацію про те, чи задача ще в черзі, виконується, завершилася успішно або завершилася з помилкою. Веб-інтерфейс може періодично опитувати бекенд щодо стану задачі й, отримавши підтвердження завершення, запропонувати користувачу завантажити сформований документ або архів. Такий механізм створює більш зручний сценарій взаємодії: замість «висящої» сторінки користувач бачить, що запит на обробку прийнято, а результат стає доступним, щойно операція завершується.

У рамках розроблюваної системи асинхронні задачі насамперед стосуються генерації документів DOCX на основі шаблонів, заповнених даними конкретних абітурієнтів, та утворення ZIP-файлів для групових операцій. Наприклад, при формуванні пакету договорів для всіх студентів певної спеціальності або для вибірки за результатами фільтрації, кожний документ створюється окремо, а потім додається до архіву. Якби така операція виконувалася в основному веб-потоці, час очікування користувача міг би обчислюватися хвилинами, а під час високих навантажень це призводило б до виснаження ресурсів сервера. Використання Celery дозволяє запускати ці процеси в окремих воркерах, які можуть бути розгорнуті на відокремлених контейнерах або серверах, забезпечуючи горизонтальне масштабування.

Ще одним напрямом застосування асинхронних задач є експорт даних у форматах XLSX та CSV. Для формування розгорнутого звіту про абітурієнтів, ціни або статистику вступу необхідно прочитати значний обсяг даних із бази, виконати агрегації, можливо, приєднати інформацію з кількох таблиць, а потім

сформувати файл у відповідному форматі. У поєднанні з бібліотеками `pandas` та `openpyxl` це може бути досить важким процесом для ЦП та пам’яті, особливо в умовах багатокористувацького режиму. Асинхронне виконання таких операцій дозволяє уникнути ситуацій, коли один користувач, який замовив великий звіт, негативно впливає на час відгуку системи для інших користувачів.

Перспективним напрямом є також використання Celery для організації електронних сповіщень. Хоча в базовій конфігурації проекту розсилання електронних листів може бути неактивним або реалізованим мінімально, архітектура Celery дає можливість легко додати задачі на відправлення email-повідомлень про зміну статусу заяви, нагадування про необхідність донести документи або повідомлення про зарахування. Такі задачі не повинні блокувати роботу основного веб-інтерфейсу: достатньо поставити їх у чергу, а воркер, налаштований на взаємодію з поштовим сервером, виконає відправлення в зручний момент.

Використання зв’язки Celery + Redis добре поєднується з контейнерною архітектурою, описаною в попередньому підрозділі. В Docker Compose кожний компонент (веб-додаток, воркер, брокер Redis) працює в окремому контейнері, з’єднаному внутрішньою мережею. Це дозволяє точно контролювати ресурси, виділені під фонову обробку: за потреби в пікові періоди вступу можна збільшити кількість воркерів Celery, не змінюючи конфігурації веб-серверів, а після завершення кампанії - зменшити їх, економлячи обчислювальні ресурси.

Таким чином, впровадження асинхронної обробки за допомогою Celery та Redis є ключовим елементом архітектури інформаційної системи автоматизації вступної кампанії. Воно забезпечує відокремлення тривалих і ресурсоємних операцій від синхронної взаємодії з користувачем, підвищує масштабованість, покращує користувацький досвід і створює фундамент для подальшого розширення функціоналу - зокрема, у напрямі автоматичних сповіщень, періодичних службових завдань і інтеграції з зовнішніми сервісами без погіршення продуктивності основних модулів системи.

2.4. Безпека, керування сесіями та робота з документами і звітністю

Для інформаційної системи автоматизації вступної кампанії питання безпеки, організації сесій і роботи з документами є взаємопов’язаними, оскільки саме в цих частинах зосереджені найбільш чутливі персональні дані. Під час вступної кампанії система опрацьовує інформацію про абітурієнтів та їхніх родичів, формує договори, заяви й звіти, які можуть передаватися різним внутрішнім і зовнішнім структурам. Тому всі механізми захисту розглядаються комплексно - як єдина підсистема забезпечення довіри до роботи платформи.

Базовим елементом безпеки є автентифікація. Паролі користувачів ніколи не зберігаються у відкритому вигляді: вони хешуються за допомогою стійких алгоритмів, що мінімізує ризики у разі витоку бази даних. Система використовує рольову модель доступу, що обмежує перегляд і редагування даних лише тими співробітниками, які мають відповідні повноваження. Це дозволяє запобігати несанкціонованим діям та зменшує ризик помилок при обробці заяв абітурієнтів.

Сесійний механізм забезпечує збереження стану взаємодії з користувачем. Для абітурієнта це - проміжні відповіді в анкетах; для співробітників - дані про автентифікацію та роль користувача. Сесії підписуються секретними ключами, а їхній час життя обмежується. Усі операції з передання сесійних даних здійснюються лише через захищені канали, що знижує ризик перехоплення інформації. За потреби система може бути доповнена серверними сесіями, що покращує контроль за активністю користувачів.

Окремий блок стосується роботи з документами. Система генерує договори, заяви та інші офіційні файли на основі шаблонів, у які підставляються індивідуальні дані абітурієнта. Важливо, що такі шаблони є незмінними юридичними формами, а система заповнює лише змінні частини, не порушуючи структуру документа. Під час обробки шаблонів додатково контролюється коректність маркерів, щоб уникнути помилок у сформованих файлах.

Формування звітів здійснюється шляхом перетворення даних у табличний формат, який може завантажуватися у XLSX або CSV. Система дотримується

принципу мінімізації: кожен звіт містить лише ті поля, які необхідні його отримувачу, без поширення зайвої персональної інформації. Згенеровані файли зберігаються у тимчасових каталогах, доступ до яких обмежено, а після завершення роботи вони вилучаються згідно з політикою зберігання.

Для зберігання документів використовується виділений файловий том, який доступний лише внутрішнім сервісам системи. Цей том не публікується назовні та забезпечує збереження документів навіть при перезапуску контейнерів. Доступ до файлів надається тільки авторизованим користувачам, а посилання на завантаження формуються динамічно з урахуванням перевірки прав.

Завершальним елементом підсистеми безпеки є логування. Система фіксує всі критичні дії - генерацію документів, експорт даних, видалення або зміну записів, входи користувачів. Це дозволяє аналізувати можливі інциденти, виявляти підозрілі дії та формувати доказову базу у разі конфліктних ситуацій. Такий комплекс - криптографічний захист, контроль доступу, безпечний документообіг і аудит - гарантує надійність системи та відповідність вимогам захисту персональних даних.

РОЗДІЛ 3.
РОЗРОБКА ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Загальна архітектура системи

Загальна архітектура інформаційної системи автоматизації вступної кампанії побудована за принципом багатошарового веб-застосунку з чітким розділенням компонентів за функціональним призначенням. У її центрі - два веб-додатки на основі Flask: застосунок приймальної комісії web та публічний застосунок web_form, орієнтований на абітурієнтів. Обидва модулі взаємодіють зі спільною базою даних PostgreSQL, використовують фонові воркери Celery для ресурсомістких операцій та обмінюються завданнями через брокер повідомлень Redis. Усі сервіси розгортаються в контейнерному середовищі Docker, а їх взаємодію координує Docker Compose, що забезпечує ізоляцію, масштабованість і відтворюваність середовища.

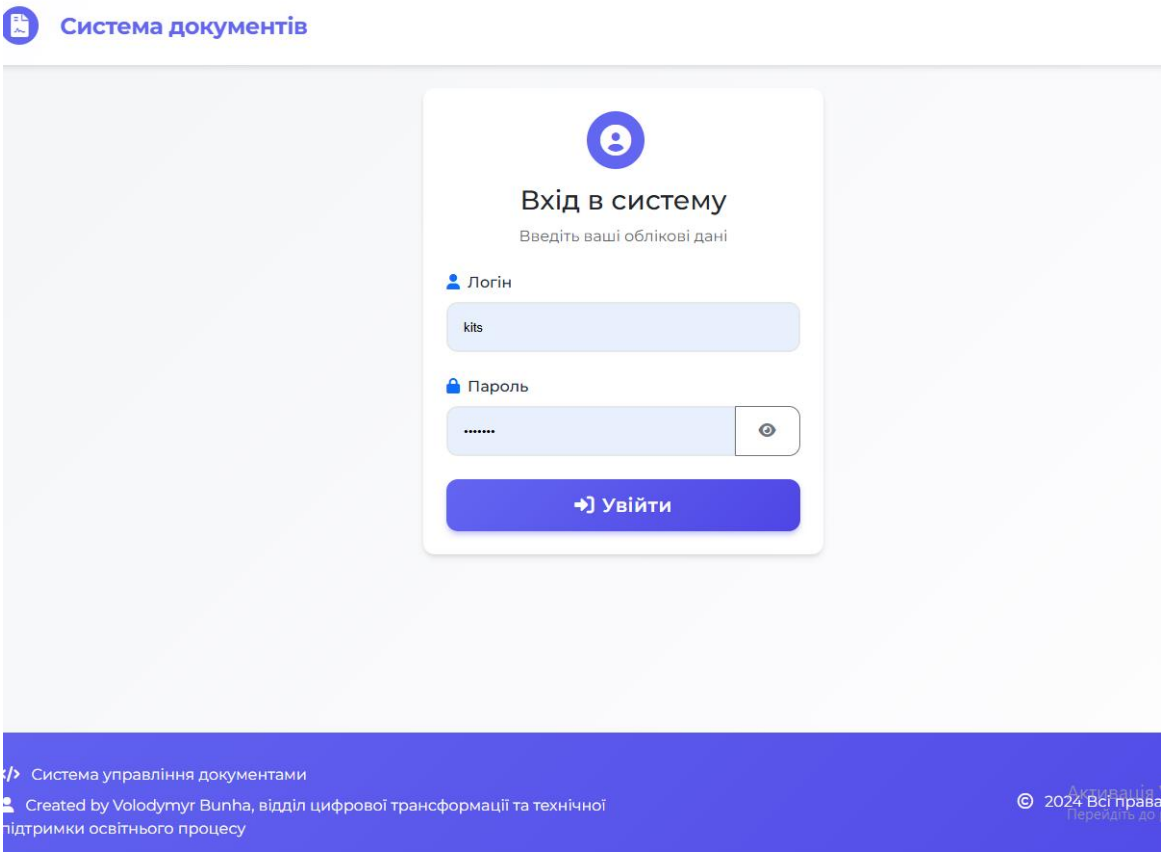

Рис. 3.1 – Сторінка авторизації відповідальних працівників

Публічний застосунок web_form забезпечує інтерфейс для абітурієнтів. Через послідовні веб-форми користувачі вводять персональні дані, інформацію про освіту, обирають спеціальності та форму навчання. Система зберігає проміжний стан анкет у сесії, виконує базову валідацію полів і підтягує довідники спеціальностей та освітніх програм з таблиці цін. Після завершення заповнення анкети дані зберігаються у таблиці students у PostgreSQL, де стають доступними для адміністративного застосунку.

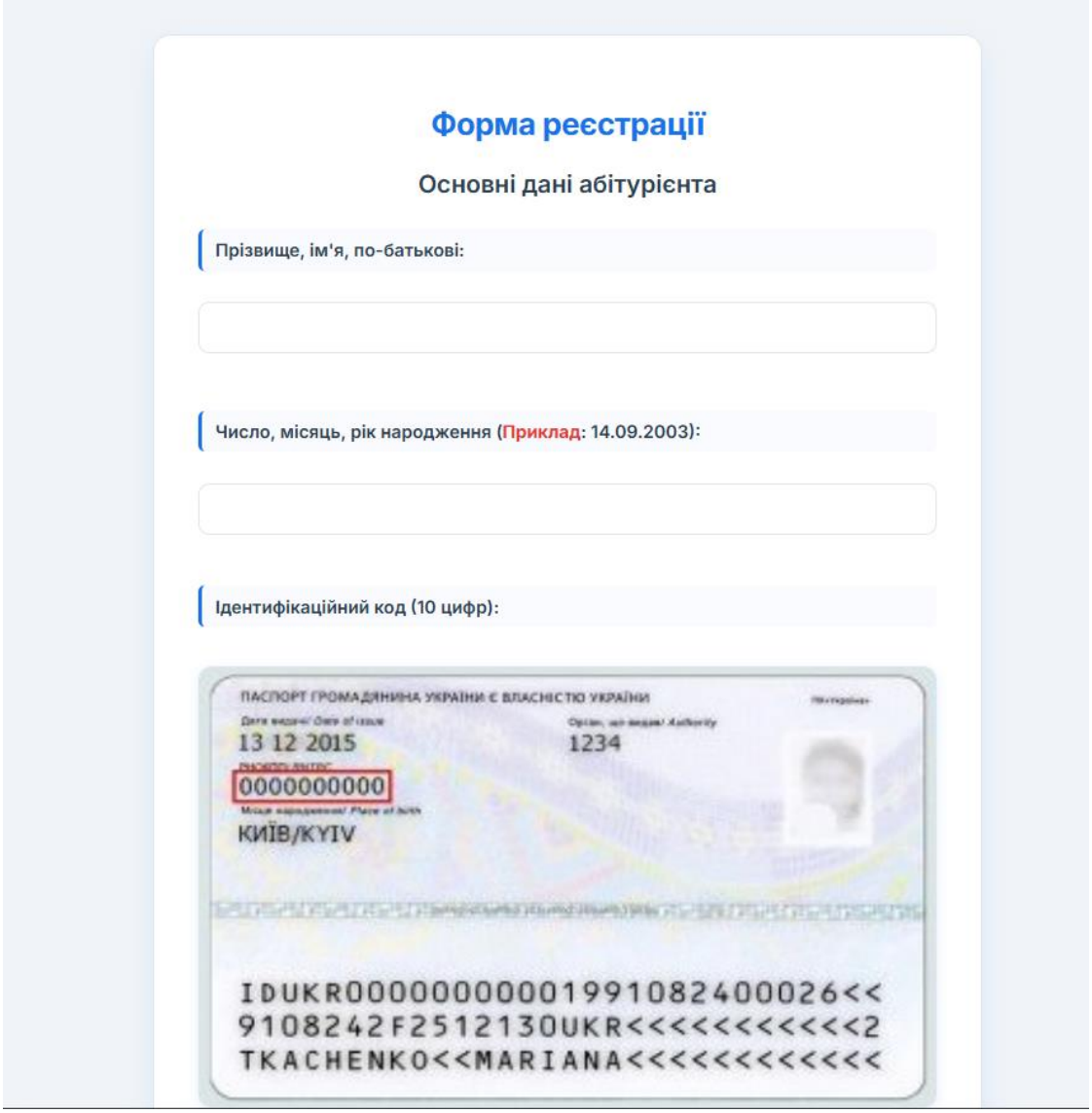

Рис. 3.2 – Інтерфейс публічного застосунку web_form

Адміністративний застосунок web призначений для співробітників приймальної комісії. Він містить механізми автентифікації, рольового доступу, модулі перегляду та редагування даних студентів, керування довідниками цін і

програм та інструменти масової генерації документів. Саме тут оператори можуть відбирати групи абітурієнтів за різними критеріями, запускати формування договорів, довідок або статистичних звітів.

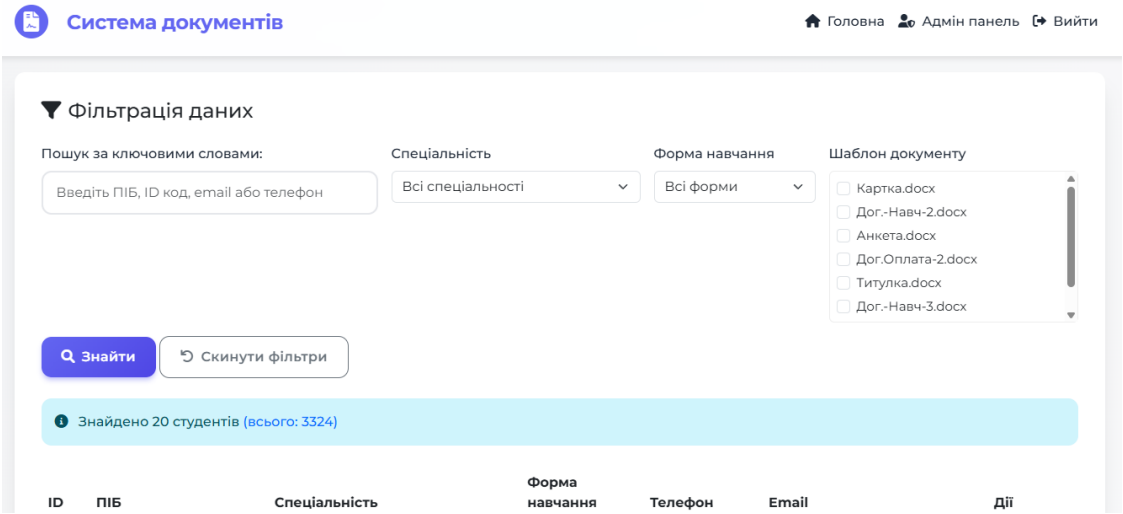

Рис. 3.3 – Інтерфейс панелі web

Адміністративна панель розробника є центральним інструментом для співробітників приймальної комісії, забезпечуючи повний цикл роботи з абітурієнтами. Інтерфейс побудований у вигляді багаторівневої навігаційної системи з розділами для перегляду, редагування та фільтрації даних студентів. Оператори можуть здійснювати пошук за ПІБ, номером заявки, освітньою програмою чи статусом вступної кампанії, а також відкривати детальну картку студента з усіма наданими документами.

Окрема частина панелі присвячена керуванню довідниками: спеціальностями, освітніми програмами, формами навчання та ціновими категоріями. Завдяки цьому структура навчальних програм може оновлюватися без зміни програмного коду.

У модулі масової генерації документів адміністратор може формувати договори, довідки, відомості та статистичні звіти для обраної групи студентів. Після запуску генерації завдання передається у Celery-чергу, що забезпечує безперервність роботи інтерфейсу навіть у разі обробки великих обсягів даних. Готові документи автоматично зберігаються у файловому томі та стають доступними для завантаження прямо в інтерфейсі.

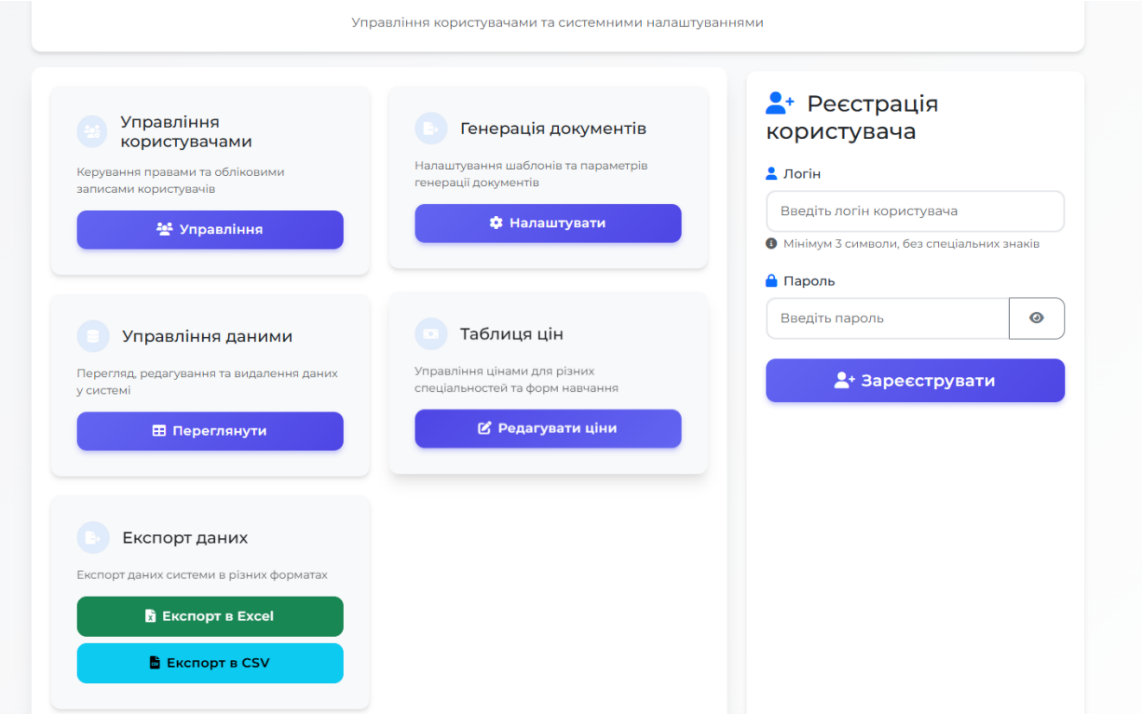

Рис. 3.4 – Інтерфейс адміністративної панелі розробника

Обидва веб-додатки працюють зі спільною базою PostgreSQL, що містить сутності students, users, prices та інші довідники. ORM-шар SQLAlchemy забезпечує єдину модель даних, спрощуючи підтримку коду та розширення схеми. Для управління міграціями використовується Alembic/Flask-Migrate.

Ресурсоємні операції - генерація DOCX-документів, створення ZIP-архівів, експорт XLSX/CSV - виконуються у фоновому режимі. Веб-додаток формує завдання і надсилає його в Redis, після чого Celery-воркери отримують це завдання, обробляють його та розміщують результати у спільному файловому томі generated_docs.

Усі компоненти працюють у контейнерах Docker, описаних у конфігурації docker-compose.yml. Compose створює внутрішню мережу, де сервіси взаємодіють за логічними іменами (db, redis, celery). Зовнішньому користувачу відкритий лише веб-порт застосунку, тоді як Redis та база даних залишаються недоступними ззовні, що підвищує безпеку системи.

Для обробки HTTP-запитів використовується виробничий сервер Gunicorn з робітниками gevent, що забезпечує стабільність роботи під навантаженням, особливо в пікові дні вступної кампанії.

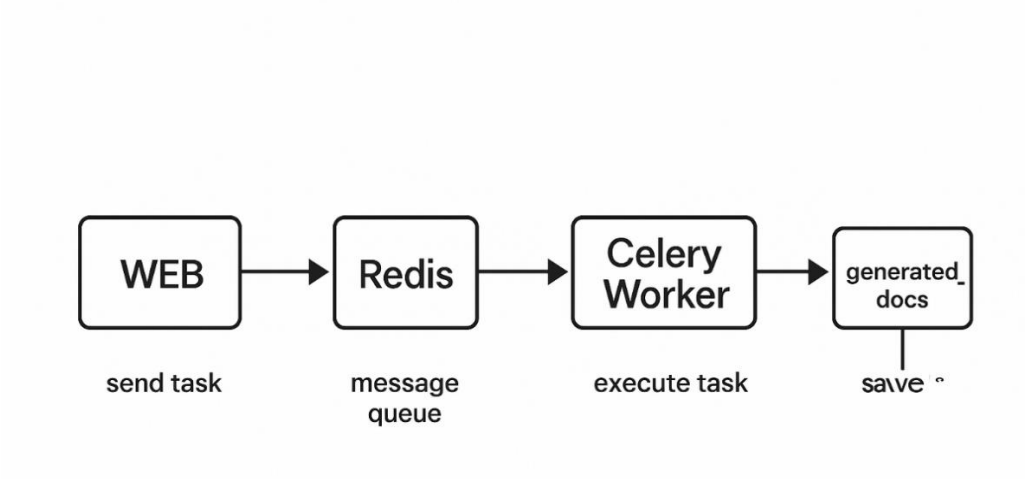

Рис. 3.5 – Діаграма логіки роботи celery

Таким чином, архітектура системи - це узгоджений комплекс: два веб-додатки, реляційна база PostgreSQL, рівень асинхронної обробки Celery, брокер Redis і файлове сховище документів. Усі компоненти розгортаються в контейнерній інфраструктурі Docker, що забезпечує масштабованість, надійність і готовність до подальшого розвитку.

3.2. Моделі та схема бази даних

Схема бази даних є фундаментом інформаційної системи автоматизації вступної кампанії, оскільки саме в ній зберігаються всі ключові відомості про абітурієнтів, користувачів та фінансові параметри. Для створення надійної та керованої структури даних було обрано реляційну СУБД PostgreSQL у поєднанні з ORM SQLAlchemy. Такий підхід дозволяє описувати модель у термінах Python-класів, підтримувати чітку типізацію та відстежувати зміни структури через механізм міграцій Alembic.

На концептуальному рівні базу даних формують три основні сутності: користувачі адміністративної панелі, абітурієнти та довідник цінових параметрів. Їхня узгоджена взаємодія забезпечує коректну роботу всіх інструментів системи - від заповнення анкет і збереження персональних даних до генерації договорів і формування статистичних звітів.

Таблиця users виконує роль сховища облікових записів співробітників приймальної комісії. У ній зберігаються логіни, хешовані паролі та ознаки адміністративних прав, що забезпечує базову модель розмежування доступу. Використання стійкого криптографічного хешування дозволяє мінімізувати ризики у випадку потенційного витоку облікових даних. Структура таблиці залишається достатньо простою, але водночас придатною для подальшого розширення у бік більш деталізованої системи ролей.

	username	password_hash	id	is_admin
4	mehanik	$2b$12$W5x4eRAtILQyst2sCuIGPupjxE5PwAjcEBiGirEcBfTyPiu4sB3eO	14	[ ]
5	agronom	$2b$12$SYBZB.JiERnyNBVo4/8wb.GBf4KF2JIZThP1o/Y.N9F6B7FbyXrq2	15	[ ]
6	zemel	$2b$12$S0G/XPtHpoWapZ/64IkIR./IheJTUNJsIEWr2YckpagJeyS1B6Ug.	16	[ ]
7	budivel	$2b$12$.0MZxBKQUHicMj8yy2c8pOowxsUf/6LrSp3z2NMw9B05tuFcX8m7W	17	[ ]
8	kindratov	$2b$12$eYC/D6rNYhEbR33dw4FOquzlax12FwsOROLzrVBUdsehpcogazFGO	18	[ ]
9	skulskaiv	$2b$12$xvLkUBwsbWB/8BCW0gqtPOuBFsRADACJQNxlVdqIP.0UzbQ7NhScu	19	[ ]
10	dubynamp	$2b$12$OASzIbpwPh/ACSBEEqTkLuEDkuuEwd/ZDKRKcoyQ5GrlAnNNcrYne	20	[ ]
11	ivaniuknt	$2b$12$5ll9s06R1sxjms4QUV25yeTl.WKPN1kKXm7va06qT4cuy02vAH9YK	21	[ ]
12	shpytiv	$2b$12$3giz.4zgfB83wygR0Q4GHeGP2fOI76l4.1faH74Rv7201gg9fVIG2	22	[ ]
13	tkachii	$2b$12$/FHVeeauEdHMSH981Eb2..CfFivCtVrbZr0rdWYVP4BlhW/azJjaW	23	[ ]
14	holubovska	$2b$12$oJPCeJZ.Yliy0Gb8yOVRaeKqPnaoE0.3mG79uyyt6z/apLFsAE1oK	24	[ ]
15	kurlaki	$2b$12$tLiZLRk.6AYgZiFjgefQr.zbUB7tBX4w24n1CCDuFPqPJCNACAjcm	25	[ ]
16	bilukoy	$2b$12$0LOZhLRLiqXI5W4zHoHNdeh.PaU6k6F1CVmoj02ZQXfYRfVu72Y8q	27	[ ]
17	freyukdv	$2b$12$I9obW4QYKsi8Qt.n6HzVJeQJWXdmCexpiWvw2PX2OkEaNEiRXh7DS	28	[ ]
18	dudchakip	$2b$12$5nobH/4zn6Wc7sstCSkUnO4GFrmqb7kpxV3At4yv2AwYpIGNqm7dO	29	[ ]
19	shalkoav	$2b$12$jx7TUwKDL4yUrdXZAjKGheBXkF7vBMZwD6Z8fL8mX.4OA.yPH554y	30	[ ]
20	pukalopy	$2b$12$GcCijt5bYORayYL3t.p7uOWprqKywfWc2fvNEoajEO5tiSlAatmSi	31	[ ]

Рис. 3.6 – Фрагмент структури таблиці бази даних users

Система підтримує операції редагування та деактивації облікових записів зображеній на рис. 3.7 . У разі звільнення співробітника або закінчення вступної кампанії його профіль може бути деактивовано або повністю видалено з панелі управління. При цьому всі записи про виконані ним дії залишаються в базі даних, що дозволяє зберегти цілісність історії змін і забезпечує можливість подальшого аудиту. Інтерфейс панелі керування користувачами містить зручні інструменти для пошуку та фільтрації за логіном, ПІБ або роллю, а також функцію відновлення пароля. Усі операції, пов’язані з управлінням користувачами, виконуються через захищений веб-інтерфейс із обмеженим доступом, що підвищує загальну безпеку системи та відповідає вимогам внутрішніх регламентів роботи приймальної комісії.

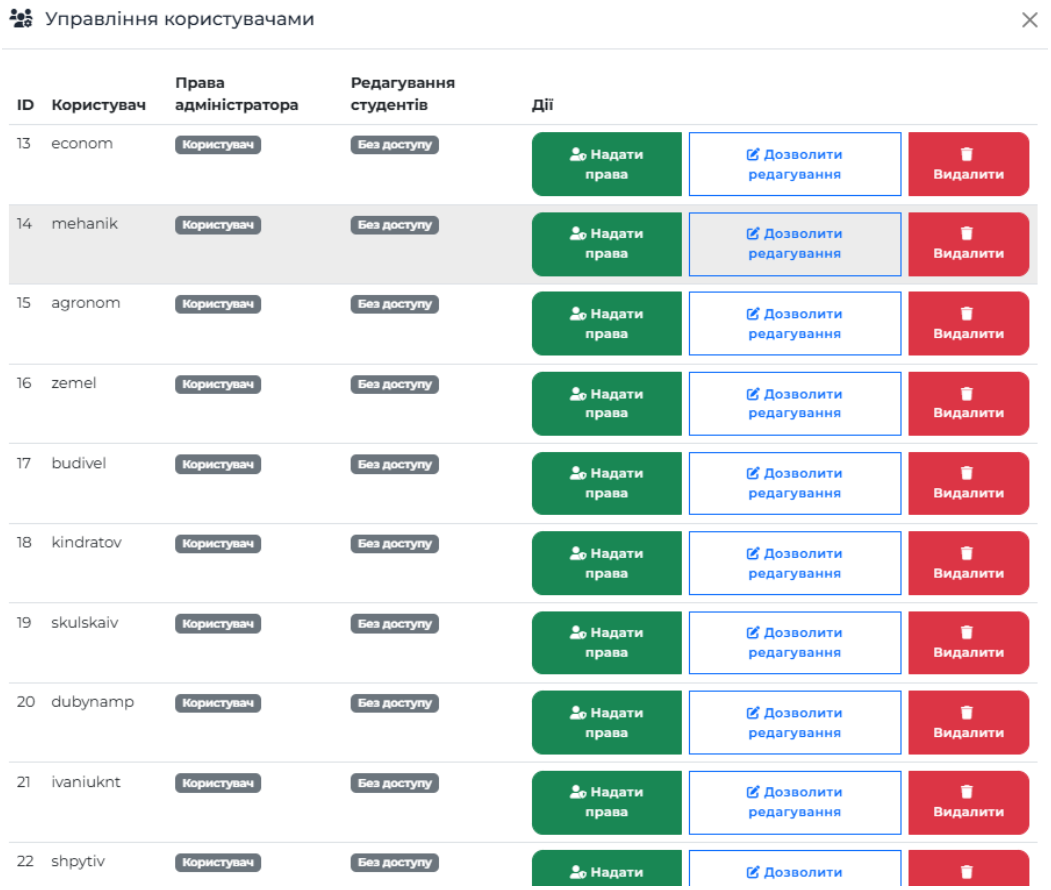

Рис. 3.7 – Інтерфейс керування користувачами

Центральне місце в базі даних посідає таблиця students, яка акумулює повний набір відомостей, потрібних для роботи приймальної комісії. Вона включає персональні дані, контактну інформацію, дані про попередню освіту, вибрану спеціальність, форму навчання, джерело фінансування, додаткові критерії та службові атрибути, необхідні для формування документів і звітності. Важливо, що структура охоплює не лише поля, передбачені типовими правилами прийому, а й елементи, які використовуються у внутрішніх бізнес-процесах ЗВО: номери справ, примітки, дані про представників тощо. Широкий набір атрибутів дозволяє генерувати договори й довідки без залучення зовнішніх джерел, що спрощує роботу операторів і зменшує ризик помилок.

Доповнює модель таблиця price, що зберігає фінансові параметри освітніх програм. Вона містить інформацію про спеціальність, ступінь, форму навчання, тривалість, а також річні та семестрові вартості. При формуванні договору система не використовує жорстко заданий зовнішній ключ, а динамічно підбирає

відповідний запис на основі поєднання параметрів, указаних в анкеті абітурієнта. Такий алгоритм дає змогу коригувати ціновий довідник без необхідності змінювати існуючі записи студентів і дозволяє адаптувати систему до різних сценаріїв - наприклад, до спеціальностей, що мають кілька варіантів вартості залежно від програми або тривалості навчання.

	specialty	deg	after_y	full_part	first_course	second_course	three_course	four_course	sum	sum_in_words	acred
1	E2 Екологія	магістр	бакалаврату а	денною	18150	18150	14520		50820	П'ятдесят тисяч вісімсот двадцять	акредитов
2	E2 Екологія	магістр	бакалаврату а	заочною	11300	11300	9040		31640	Тридцять одна тисяча шістсот сорок	акредитов
3	G2 Технології захис	бакалавр	школи (трива	денною	13950	13950	13950	13950	111600	Сто одинадцять тисяч шістсот	акредитов
4	G2 Технології захис	бакалавр	школи (трива	заочною	8400	8400	8400	8400	67200	Шістдесят сім тисяч двісті	акредитов
5	G2 Технології захис	бакалавр	коледжу на ба	денною	13950	13950	13950	13950	55800	П'ятдесят п'ять тисяч вісімсот	акредитов
6	G2 Технології захис	бакалавр	коледжу на ба	заочною	8400	8400	8400	8400	33600	Тридцять три тисячі шістсот	акредитов
7	H1 Агрономія	бакалавр	коледжу на ба	денною	13950	13950	13950	13950	55800	П'ятдесят п'ять тисяч вісімсот	акредитов
8	H1 Агрономія	бакалавр	коледжу на ба	заочною	7650	7650	7650	7650	30600	Тридцять тисяч шістсот	акредитов
9	E2 Екологія	бакалавр	коледжу на ба	заочною	8400	8400	8400	8400	50400	П'ятдесят тисяч чотириста	акредитов
10	D3 Менеджмент	бакалавр	коледжу на ба	заочною	8950	8950	8950	8950	35800	Тридцять п'ять тисяч вісімсот	акредитов
11	E2 Екологія	бакалавр	школи (трива	заочною	8400	8400	8400	8400	67200	Шістдесят сім тисяч двісті	акредитов
12	D5 Маркетинг	бакалавр	школи (трива	заочною	8950	8950	8950	8950	71600	Сімдесят одна тисяча шістсот	акредитов
13	H6 Ветеринарна ме	магістр	школи (трива	денною	22600	22600	22600	22600	271200	Двісті сімдесят одна тисяча двісті	акредитов
14	H6 Ветеринарна ме	магістр	школи (трива	денною	22600	22600	22600	22600	271200	Двісті сімдесят одна тисяча двісті	не акреди
15	H6 Ветеринарна ме	магістр	школи (трива	денною	22600	22600	22600	22600	271200	Двісті сімдесят одна тисяча двісті	не акреди
16	H6 Ветеринарна ме	магістр	школи (трива	денною	22600	22600	22600	22600	271200	Двісті сімдесят одна тисяча двісті	не акреди
17	H6 Ветеринарна ме	магістр	школи (трива	денною	22600	22600	22600	22600	271200	Двісті сімдесят одна тисяча двісті	не акреди
18	H6 Ветеринарна ме	магістр	коледжу на ба	денною	22600	22600	22600	22600	226000	Двісті двадцять шість тисяч	акредитов
19	A4.11 Середня освіт	бакалавр	школи (трива	денною	13950	13950	13950	13950	111600	Сто одинадцять тисяч шістсот	не акреди
20	A4.11 Середня освіт	бакалавр	школи (трива	заочною	8950	8950	8950	8950	71600	Сімдесят одна тисяча шістсот	не акреди
21	A1 Освітні науки	магістр	бакалавр ату а	денною	18150	18150	14520		50820	П'ятдесят тисяч вісімсот двадцять	акредитов
22	A1 Освітні науки	магістр	бакалавр ату а	заочною	11300	11300	9040		31640	Тридцять одна тисяча шістсот сорок	акредитов
23	A7 Фізична культур	бакалавр	школи (трива	денною	13950	13950	13950	13950	111600	Сто одинадцять тисяч шістсот	акредитов
24	A7 Фізична культур	бакалавр	школи (трива	заочною	8950	8950	8950	8950	71600	Сімдесят одна тисяча шістсот	акредитов
25	A4.11 Середня освіт	бакалавр	коледжу на ба	заочною	8950	8950	8950	8950	53700	П'ятдесят три тисячі сімсот	не акреди

Рис. 3.8 – Фрагмент таблиці price

Схема бази даних містить низку технічних елементів, які забезпечують її стабільність: первинні ключі, індекси для часто використовуваних полів (зокрема ПІБ і спеціальності), обмеження цілісності та уніфіковані типи даних. Під час проєктування враховувалася необхідність майбутнього розширення, тож структура побудована модульно й допускає додавання нових таблиць - наприклад, для журналювання дій користувачів, зберігання шаблонів документів або реалізації більш формальної системи контролю прав.

Важливо також, що всі компоненти системи - обидва веб-додатки та Celery-воркери - працюють із єдиним набором моделей SQLAlchemy. Це дозволяє уникнути дублювання логіки та розсинхронізації між рівнями. Будь-яка зміна структури даних виконується через міграції, що гарантує однаковий стан схеми як у середовищі розробки, так і на промисловому сервері.

У результаті база даних виступає добре структурованою основою для всієї системи: вона водночас достатньо деталізована для генерування юридично

значущих документів і достатньо гнучка, щоб адаптуватися до змін нормативних вимог або внутрішніх процесів університету.

3.3. Веб-інтерфейси та шаблони

Веб-інтерфейси інформаційної системи автоматизації вступної кампанії є основним каналом взаємодії як для абітурієнтів, так і для співробітників приймальної комісії. Саме через них здійснюється введення анкетних даних, перевірка інформації, запуск операцій генерації документів і перегляд звітів. Якість опрацювання даних, кількість помилок та загальне враження користувачів від системи значною мірою залежать від того, наскільки продумано організовано веб-форми, навігацію, механізми валідації й відгуку інтерфейсу. У розроблюваній системі для побудови веб-інтерфейсів використано шаблонізатор Jinja2, що інтегрований з Flask, а також набір HTML-шаблонів для публічної частини (анкети абітурієнта) й адміністративної панелі.

Публічний інтерфейс, реалізований у застосунку `web_form`, побудований за принципом покрокового заповнення анкет. Користувач бачить послідовність сторінок (розділів анкети), на кожній із яких збирається логічно пов’язаний блок відомостей: персональні дані, інформація про освіту, вибір спеціальності та форми навчання, військовий облік, дані про батьків чи законних представників, згода на обробку персональних даних тощо. Такий підхід дозволяє не перевантажувати користувача надто великими формами й зменшує ймовірність пропуску обов’язкових полів, оскільки кожний крок супроводжується підказками та локальною валідацією. Шаблони Jinja2 дають змогу динамічно підставляти у форми списки спеціальностей, освітніх програм, форм навчання, які отримуються з бази даних через модель `Price` або кеш. Це означає, що зміна довідників у адмін-панелі автоматично відображається у веб-формах абітурієнта без потреби ручного редагування HTML-сторінок.

Адміністративний інтерфейс, реалізований у застосунку `web`, має іншу спрямованість: він орієнтований на операційні задачі співробітників

приймальної комісії. Тут шаблони Jinja2 використовуються для побудови таблиць зі списками студентів, форм фільтрації й пошуку, сторінок перегляду та редагування анкетних даних, а також сторінок управління користувачами й таблицею цін. Інтерфейс організовано таким чином, щоб забезпечити швидкий доступ до потрібної інформації: реалізовано сортування та фільтрацію за ключовими полями (ПІБ, спеціальність, форма навчання), відображення детальної інформації про окремого студента в модальному вікні або на окремій сторінці, а також інструменти масового вибору записів для подальшої генерації документів. Усі ці елементи також побудовані на основі шаблонів, де за допомогою Jinja2 відображаються динамічні дані з бази й управляються умовні елементи (наприклад, відображення тих чи інших кнопок залежно від ролі користувача).

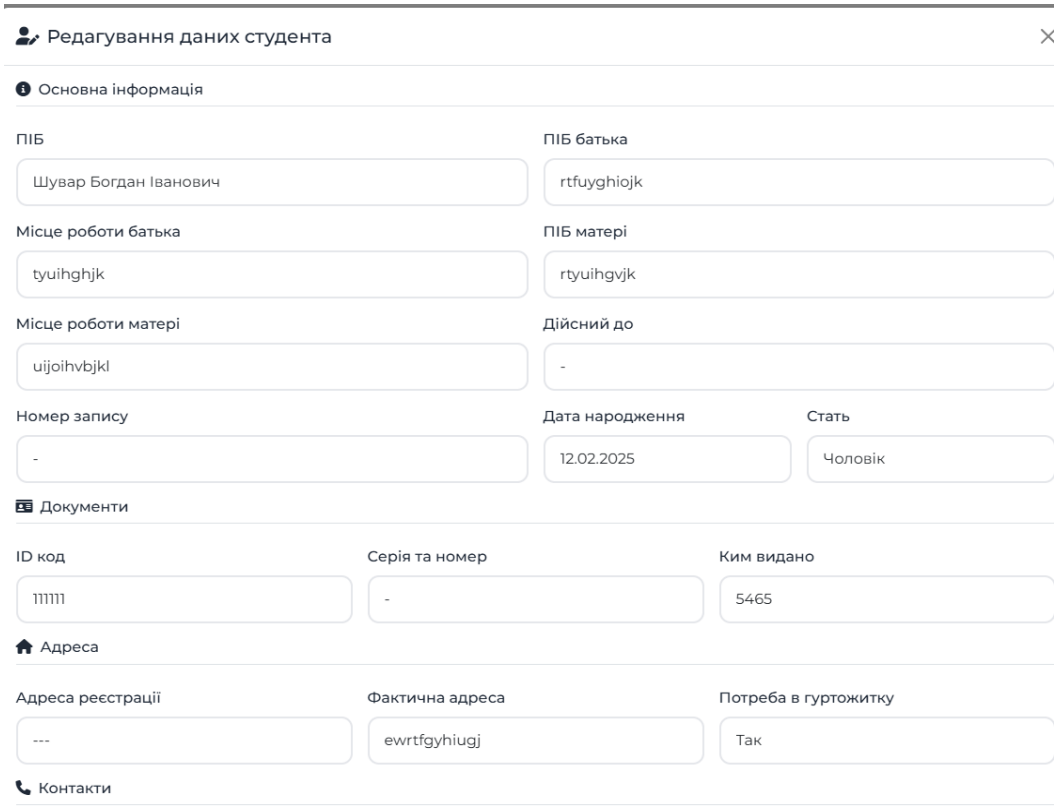

Рис. 3.9 - Сторінка перегляду та керування даними студента

Важливою складовою веб-інтерфейсів є валідація даних. Вона здійснюється на обох рівнях - клієнтському (через стандартні HTML-атрибути, регулярні вирази, JavaScript-перевірки) та серверному (в обробниках Flask). На

клієнтському рівні перевіряється коректність формату введення (наприклад, шаблон для електронної пошти, довжина ідентифікаційного коду, формат номера документа), обов’язковість заповнення певних полів, базові діапазони дат. Це дозволяє виявити очевидні помилки ще до відправлення форми на сервер, зменшує обсяг непотрібних запитів і підвищує зручність для користувача, який одразу бачить підказки та повідомлення про помилки. Однак основна, «жорстка» валідація реалізована на сервері: тут перевіряється логічна узгодженість даних (наприклад, відповідність віку можливим освітнім траєкторіям, коректність комбінацій спеціальності й ступеня, повнота даних для вибраного типу договору). У разі виявлення помилки сервер повертає ту ж сторінку форми з відповідними повідомленнями, які відображаються через механізми шаблонізатора.

Захист від типових веб-загроз, зокрема CSRF-атак (cross-site request forgery), також інтегровано в архітектуру веб-інтерфейсів. CSRF-атака полягає в тому, що зловмисник намагається змусити браузер користувача виконати небажану дію на довіреному сайті (наприклад, надіслати POST-запит з видаленням або зміною даних), використовуючи вже наявну сесію. Щоб запобігти таким сценаріям, у формах, які виконують станозмінні операції (запис анкет, оновлення даних, видалення записів, зміна прав користувачів), має використовуватися CSRF-токен - випадкове значення, що генерується на сервері й вставляється у форму як приховане поле. Під час обробки запиту сервер перевіряє наявність і валідність цього токена; у разі невідповідності запит відхиляється. У середовищі Flask така логіка може бути реалізована за допомогою спеціальних розширень (наприклад, Flask-WTF) або власних реалізацій, інтегрованих із сесійним механізмом. Хоча в початковій версії проекту захист від CSRF може бути реалізований частково, архітектура веб-інтерфейсів передбачає можливість його повноцінного впровадження на всіх критичних маршрутах.

Особливу роль відіграє уніфікація зовнішнього вигляду й поведінки інтерфейсів через базові шаблони. Як у публічній частині, так і в адмін-панелі

використовуються базові макети (layout), які містять заголовок сторінки, навігаційне меню, стандартні блоки повідомлень (успіх, помилка, попередження), футер із контактами чи технічною інформацією. Конкретні сторінки (анкети, списки, форми редагування) «наслідують» ці базові шаблони через механізм розширення Jinja2, заповнюючи визначені блоки вмістом. Це значно спрощує підтримку зовнішнього вигляду та дає можливість централізовано змінювати дизайн без необхідності редагувати десятки окремих файлів. Крім того, уніфікація інтерфейсу полегшує навчання користувачів: дотримання однакових патернів розміщення елементів (кнопки «Зберегти», «Назад», фільтри, таблиці) дозволяє швидко орієнтуватися навіть тим співробітникам, які не мають глибокого досвіду роботи з інформаційними системами.

Нарешті, веб-інтерфейси тісно пов’язані із загальною концепцією безпеки системи. Окрім CSRF-захисту й коректної обробки сесій, важливим є запобігання XSS-атакам (міжсайтовому скриптингу). Jinja2 за замовчуванням виконує екранування (escaping) даних, що виводяться у шаблони, тим самим зменшуючи ризик виконання шкідливих скриптів, впроваджених до введених користувачем полів. Для полів, де припускається HTML-форматування, потрібні додаткові механізми фільтрації та контролю. Також у шаблонах слід уникати прямого відображення службових даних (ідентифікаційних кодів, внутрішніх ідентифікаторів без потреби), щоб не розкривати зайвих технічних деталей.

Таким чином, веб-інтерфейси й шаблони системи виконують не лише роль «обличчя» інформаційної системи, а й є важливим технічним рівнем, де поєднуються юзабіліті, зв’язок із моделями даних, механізми валідації й елементи захисту. Використання Jinja2-шаблонів, покрокового заповнення форм, клієнтської та серверної валідації, інтеграції з сесійним механізмом і впровадження захисту від CSRF-атак створює передумови для того, щоб система була одночасно зручною для користувачів, стійкою до помилок і захищеною від типових веб-загроз.

3.4. Фонові задачі Celery

Фонові задачі є одним із ключових елементів архітектури системи, оскільки саме вони беруть на себе найбільш ресурсомісткі операції, які не повинні заважати роботі користувачів у веб-інтерфейсі. У контексті вступної кампанії такими операціями, передусім, є формування персоналізованих документів - договорів, заяв, довідок - а також створення архівів для пакетного опрацювання великої кількості абітурієнтів. Щоб уникнути затримок і забезпечити стабільність роботи, ця частина функціональності винесена у фоновий рівень, реалізований засобами Celery з використанням Redis як брокера.

Процес формування документа починається у веб-інтерфейсі, коли оператор обирає студента або групу студентів і запускає генерацію. Замість того щоб виконувати всі операції одразу, застосунок передає до черги Celery лише необхідні параметри - ідентифікатори записів і назви шаблонів. Далі задачу бере на себе окремий воркер, який працює незалежно від основного веб-процесу. Це дозволяє уникнути ситуацій, коли довге формування документів блокує роботу інших користувачів.

У середовищі воркера відкривається відповідний DOCX-шаблон, а на його основі формується фінальний документ. Алгоритм побудований таким чином, щоб коректно обробляти навіть складні випадки, коли маркери у шаблоні розбиті на кілька текстових сегментів. Підставляючи дані зі схеми бази, система не змінює форматування, тому зовнішній вигляд документів залишається повністю відповідним вимогам університету. Готовий файл зберігається у спільному томі, доступному як воркерам, так і веб-інтерфейсу.

Якщо ж оператор формує не один документ, а пакет документів для групи абітурієнтів, система використовує аналогічний алгоритм, але формує ZIP-архів. Кожен документ створюється у пам’яті й додається до архіву без зайвих проміжних операцій на диску. Це суттєво пришвидшує роботу в періоди пікового навантаження, коли кількість заявників значно зростає. Після завершення архів

зберігається у файловому томі, а веб-інтерфейс отримує повідомлення про готовність файлу для завантаження.

Окремий напрям роботи фонового рівня - контроль виконання завдань. Кожна задача, поставлена в чергу, отримує унікальний ідентифікатор, за яким можна відстежити її стан. Це дозволяє інтерфейсу надавати користувачу актуальну інформацію: чи задача очікує виконання, чи виконується, чи завершена, чи сталася помилка. Логування операцій - невід'ємна частина цієї підсистеми. У журналах зберігається час запуску, тип операції, ініціатор, статус і, у разі збою, зміст помилки. Такий журнал є не лише технічним інструментом, а й важливим елементом забезпечення прозорості та безпеки, оскільки дозволяє підтвердити факти формування документів і виключити несанкціоновані дії.

Під час реальної експлуатації система може стикатися з тимчасовими збоями - затримками доступу до файлової системи, нестабільністю мережі або високим навантаженням на базу даних. Для таких випадків у Celery передбачено механізм повторних спроб. Якщо задача завершилася помилкою, але її виконання потенційно можливе при повторі, воркер автоматично здійснить наступну спробу через певний інтервал. Це дозволяє системі залишатися стабільною навіть у непередбачуваних умовах і зменшує потребу у ручному втручанні.

З боку користувача взаємодія з фоновою підсистемою виглядає просто: після запуску процесу він отримує повідомлення про те, що завдання прийнято до виконання, і невдовзі - посилання для завантаження файлів або повідомлення про помилку. Такий підхід забезпечує плавний і безпечний досвід роботи навіть тоді, коли система опрацьовує великі обсяги даних.

У цілому Celery слугує надійним інструментом, що дозволяє винести ресурсомісткі операції поза межі основного застосунку й забезпечити незалежне, контрольоване та масштабоване виконання задач. Це робить систему стійкою до навантажень, підвищує її продуктивність і створює простір для подальшого розвитку - зокрема для впровадження періодичних робіт та інтеграцій із зовнішніми сервісами, які також можуть виконуватися у фоновому режимі.

3.5. Тестування та налагодження програмного комплексу

Тестування інформаційної системи проводилося переважно вручну, у форматі практичних перевірок, максимально наближених до того, як система працюватиме під час реальної вступної кампанії. Основний акцент робився не на формальному проходженні тест-кейсів, а на перевірці того, наскільки стабільно й передбачувано взаємодіють її компоненти, чи зручні веб-інтерфейси для різних категорій користувачів і чи коректно система опрацьовує типові та нестандартні ситуації. Для розгортання використовувалося контейнерне середовище Docker, у якому окремо запускалися сервіси web і web_form, база даних PostgreSQL, брокер Redis, воркери Celery та файловий том. Це дозволило відтворити структуру майбутнього продуктивного середовища та оцінити не лише роботу окремих модулів, а й якість їхньої взаємодії.

Функціональне тестування включало серію сценаріїв, що відображають типову поведінку абітурієнтів. Перевірявся весь шлях проходження анкети: від заповнення персональних даних і вибору спеціальності до фінального підтвердження. Особливу увагу приділяли валідації — як система реагує на пропущені обов’язкові поля, неправильно введені дати чи email, а також на суперечливі комбінації параметрів. Очікуваним результатом було коректне повідомлення про помилку та збереження вже введених користувачем даних, щоб уникнути повторного заповнення.

В адміністративній частині тестувалися сценарії з урахуванням різних ролей користувачів — адміністратора та оператора. Перевірялися фільтри пошуку студентів, редагування анкет, робота з довідником цін, створення й видалення користувачів. Окремий блок тестування був присвячений генерації документів: перевірялася як одинична генерація договорів за шаблонами DOCX, так і пакетна — для групи абітурієнтів із формуванням ZIP-архіву. Під час перевірок особливу увагу приділяли правильності підставлення персональних даних, інформації про спеціальність і форми навчання, сум оплати, строків дії договору та значень із довідника price.

У процесі налагодження активно використовувалися журнали серверних застосунків і воркерів Celery. Саме логування дозволило вчасно виявити SQL-помилки, проблеми з підключенням до Redis, окремі винятки при роботі з файлами або ситуації, коли користувач запускав генерацію без вказаних шаблонів. На основі цих повідомлень уточнювалися умови вибірки даних з БД, додавалися перевірки на відсутні значення, коригувалися повідомлення для користувачів. Також відпрацьовувалися нестандартні ситуації — повторні запуски однакових задач, звернення до неіснуючих записів тощо.

Попри доволі скромні ресурси, система стабільно витримувала понад 100 одночасних користувачів, які заповнювали анкету, та 30–50 користувачів, що паралельно запускали генерацію документів. Параметри черги Celery були налаштовані так, щоб одночасно виконувалося до 8 задач, а всі інші запити ставали в чергу. Завдяки цьому системі вдалося уникнути перевантаження процесора, запобігти зупинкам і забезпечити стабільний час відгуку навіть за пікового навантаження.

РОЗДІЛ 4.
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Аналіз травмонебезпечних ситуацій під час виконання робіт

Розробка та вживання ефективних заходів запобігання аварійним і травмонебезпечних ситуаціям можливі лише при завчасному виявлення тих небезпек, з яких починаються процеси їх формування. Оскільки небезпечні умови не завжди завчасно можна виявити, а для вивчення небезпечних дій іноді потрібно багато часу, щоб зібрати статичний матеріал, то і методи виявлення цих небезпек повинні бути відповідно диференційовані .

Відповідно до аналізу небезпечних умов, які існують у виробничому процесі виокремлено такі наступні за характером дії на працівника їх групи:

- характеризують стан або рівень небезпеки обладнання, які використовуються.
- сприяють виникненню технологічних помилок обслуговуючого персоналу впродовж виробничого процесу;
- створювати умови та варіанти проникнення працівника в небезпечну зону;
- приводять до виникнення небезпечних дій (внаслідок низького рівня професійної підготовки працівників та організації навчання з охорони праці).

4.2. Структурно-функціональний аналіз дотримання охорони праці при виконанні роботи з комп’ютером

При виконанні роботи з комп'ютером важливо забезпечити належне дотримання норм охорони праці. Для цього необхідно провести структурно-функціональний аналіз, що дозволить виявити елементи та процеси, пов'язані з

безпекою праці під час роботи з комп'ютером. Нижче наведено кроки, які допоможуть у проведенні аналізу:

1. Визначення робочого місця: Аналізується структура та організація робочого місця, включаючи комп'ютерну систему, робочий стіл, стілець, освітлення та інші елементи. Важливо забезпечити правильну постановку обладнання, оптимальні умови освітлення, регулювання висоти столу та стільця для запобігання неправильній позі та напруженню м'язів.

2. Оцінка ризиків: Визначення потенційних ризиків, пов'язаних з роботою з комп'ютером, таких як перенапруження очей, неправильна постава, пов'язані з довготривалим сидінням та відсутністю перерв. Оцінка факторів, що можуть впливати на здоров'я працівників, таких як шум, радіація, електромагнітні поля тощо.

3. Розробка процедур безпеки: Розроблення та впровадження процедур, спрямованих на запобігання можливим травмам та проблемам зі здоров'ям, пов'язаним з роботою з комп'ютером. Це можуть бути рекомендації щодо правильної постави, використання захисних окулярів, проведення регулярних перерв для відпочинку та розтяжки, а також застосування ергономічних принципів.

4. Навчання та тренування персоналу: Проведення навчання та тренування працівників з питань безпеки праці при роботі з комп'ютером. Це включає ознайомлення з правилами безпеки, освіту щодо користування комп'ютером та його периферійними пристроями, а також навчання профілактичних вправ та розтяжок для зменшення напруги в м'язах.

5. Регулярні перевірки та аудит. Проведення регулярних перевірок робочих місць та комп'ютерних систем для виявлення можливих недоліків та проблем безпеки праці. Аудит безпеки праці допоможе забезпечити виконання встановлених стандартів та процедур безпеки.

Структурно-функціональний аналіз дотримання охорони праці при роботі з комп'ютером допомагає ідентифікувати потенційні ризики та визначити необхідні заходи для забезпечення безпеки працівників. Виконання цього аналізу

дозволяє зменшити випадки травм та проблем зі здоров'ям, пов'язаними з роботою з комп'ютером, та забезпечити належні умови праці.

4.3. Обґрунтування організаційно-технічних рекомендацій з охорони праці

Організаційно-технічні рекомендації з охорони праці є важливим етапом у забезпеченні безпеки працівників під час виконання роботи. Для покращення умов праці з комп’ютером та запобігання травмам і проблемам зі здоров’ям нижче наведено обґрунтування основних рекомендацій.

Регулярні перевірки технічного стану обладнання. Необхідно запровадити систематичні перевірки та обслуговування комп'ютерної техніки з метою своєчасного виявлення можливих проблем. Це включає контроль роботи жорсткого диска, системи охолодження, клавіатури, миші та інших периферійних пристроїв.

Забезпечення правильної постановки обладнання. Комп’ютер, монітор, клавіатура та миша повинні бути розташовані відповідно до ергономічних принципів. Важливо враховувати оптимальну висоту столу й стільця, належне освітлення та правильну поставу працівника, що зменшує негативний вплив на опорно-руховий апарат і зір.

Встановлення регулярних перерв. Працівникам, які тривалий час працюють за комп’ютером, необхідно рекомендувати перерви кожні 1–2 години. Короткий відпочинок, розтяжки та вправи для очей і шиї допомагають знизити перенапруження м’язів та ризики професійних захворювань.

Навчання та підвищення свідомості працівників. Слід проводити навчання і інформування щодо правил безпеки при роботі з комп’ютером: правильної постави, необхідності перерв, виконання розтяжок, а також програм зі зниження стресу та напруги.

Моніторинг та аналіз показників безпеки праці. Важливо впровадити систему моніторингу таких показників, як кількість травматичних випадків, час

відпочинку, рівень задоволеності працівників. Це дозволить своєчасно виявляти проблемні місця й ухвалювати заходи для покращення умов праці.

Обґрунтування організаційно-технічних рекомендацій з охорони праці є необхідним для створення комфортного та безпечного середовища під час роботи з комп'ютером. Реалізація цих заходів сприяє підвищенню продуктивності, збереженню здоров'я та загальному благополуччю працівників.

4.4. Безпека в надзвичайних ситуаціях

Забезпечення захисту населення та території у разі загрози або надзвичайних ситуацій є одним з найважливіших завдань держави. Захист населення реалізується через систему загальнодержавних заходів, що проводяться центральними і місцевими органами виконавчої влади, органами цивільного захисту та підприємствами. Ці заходи включають організаційні, інженерно-технічні, санітарно-гігієнічні та інші дії, спрямовані на запобігання і ліквідацію наслідків надзвичайних ситуацій.

Загрози життєво важливих інтересів поділяються на зовнішні та внутрішні, що можуть виникати під час техногенних і природних катастроф, а також у період воєнних конфліктів. Принципи захисту базуються на положеннях Женевської конвенції щодо захисту жертв війни, прогнозованому характері можливих воєнних дій та реальних можливостях держави щодо створення матеріальної бази для забезпечення оборони та безпеки.

Для зменшення втрат і шкоди в разі надзвичайних ситуацій проводиться спеціальний комплекс заходів. До них належить завчасне створення та підтримання в постійній готовності систем оповіщення населення, що дозволяє оперативно реагувати на небезпеки та вживати заходів для збереження життя і здоров'я людей.

РОЗДІЛ 5
ДОЦІЛЬНІСТЬ ПРОЄКТУ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

5.1. Практична, загальна економічна та соціальна доцільність

Розроблена інформаційна система автоматизації вступної кампанії є сучасним інструментом, спрямованим на підвищення ефективності роботи закладу вищої освіти та покращення взаємодії між абітурієнтами й приймальною комісією. Традиційний механізм організації вступу базується на значному обсязі ручної праці, що уповільнює робочі процеси, зумовлює високу ймовірність помилок та створює додаткове навантаження на персонал. Автоматизація усуває ці недоліки, переводячи ключові етапи обробки даних у цифрове середовище, де операції виконуються швидше, стабільніше і з мінімальною залежністю від людського фактора.

Практична доцільність системи проявляється у суттєвому скороченні часу, необхідного для виконання рутинних операцій. Зокрема, формування договору, яке раніше займало до десяти хвилин, у автоматизованому режимі виконується менше ніж за секунду. Пакет документів, що опрацьовувався протягом декількох годин, тепер генерується за одну хвилину. Це забезпечує значне підвищення пропускної здатності системи під час пікових навантажень, що є особливо важливим у період масових подач заяв.

Економічна доцільність полягає у скороченні витрат на оплату праці персоналу приймальної комісії та оптимізації внутрішніх процесів. У ручному режимі витрати ґрунтуються на оплаті часу працівників, у той час як автоматизована система потребує мінімального енергоспоживання. Собівартість формування ста документів становить менше однієї копійки, що у сотні разів нижче вартості ручного опрацювання. Для середнього ЗВО це означає економію десятків тисяч гривень за одну вступну кампанію, а у випадку масштабування — швидку окупність інвестицій у впровадження системи.

Соціальна доцільність полягає у створенні комфортного, зручного і прозорого механізму подання документів. Абітурієнти отримують можливість швидко проходити всі етапи вступу без тривалого очікування та зайвих бюрократичних процедур. Працівники університету, своєю чергою, звільняються від надмірної кількості технічних завдань, що дозволяє їм приділяти більше уваги консультативній та організаційній роботі. Таким чином, система сприяє покращенню якості сервісу, підвищує довіру до роботи приймальної комісії та підтримує загальнодержавну тенденцію цифровізації освітніх процесів.

Узагальнюючи, можна стверджувати, що проєкт є практично значущим, економічно вигідним і соціально корисним. Він відповідає сучасним вимогам цифрової трансформації та створює фундамент для подальшого вдосконалення процедур вступу.

5.2. Результати навантажувальних випробувань

Для цього було проведено серію навантажувальних сценаріїв з вимірюванням характеристик контейнерів Docker, які реалізують сервіси системи. Результати узагальнено у таблиці 5.1.

Таблиця 5.1 – Використання ресурсів контейнерів у штатному та піковому режимах

Контейнер	CPU (середнє)	CPU (пік)	RAM (середнє)	RAM (пік)	Характер навантаження
web (адмін-панель)	~0.23%	1.2–1.8%	488 MB	620–700 MB	ORM-запити, великі таблиці, фільтри
web_form (анкети)	~0.01%	0.15-0.25%	71 MB	95–120 MB	Обробка форм, POST-запити
web-celery (воркери)	~0.11%	2.5–3.8%	376 MB	550–680 MB	Генерація DOCX, ZIP, робота з томом
redis	~0.18%	0.35–0.50%	9 MB	12–18 MB	Черга задач, мінімальне навантаження

Аналіз навантаження дозволив встановити, що система функціонує стабільно навіть за умов одночасного доступу великої кількості користувачів.

Найбільше навантаження припадає на Celery-воркери, які виконують обчислювальні операції та формують документи. Основний веб-інтерфейс та форми абітурієнтів залишаються малоресурсними, що забезпечує високу швидкість відгуку навіть у пікові моменти.

Серед часових метрик ключовими стали:

- генерація одного договору: ~0.8 с (фонове виконання; UI не блокується);
- генерація пакета зі 100 договорів: 50–60 с;
- завантаження таблиці студентів (500 записів): 1.2–1.5 с;
- збереження кроку анкети: <0.5 с.

Ці показники демонструють реальне скорочення часу обробки заяв у порівнянні з ручним режимом, а також високу реактивність інтерфейсу за великих навантажень.

Результати навантажувальних випробувань підтвердили, що система демонструє стабільну, керовану та передбачувану роботу навіть за умов значної кількості одночасних користувачів. Під час тестування моделювалися характерні ситуації вступної кампанії: одночасне масове заповнення анкет, перегляд і фільтрація великих масивів даних у адміністративній панелі, запуск пакетної генерації документів, робота із довідниками та формування звітів.

У нормальному режимі всі контейнери працюють з мінімальним використанням обчислювальних ресурсів: завантаження CPU зазвичай не перевищує 1%, а споживання оперативної пам’яті залишається у межах 70–500 МБ залежно від сервісу. Такі показники свідчать про те, що система має великий запас продуктивності та може масштабуватися без модернізації обладнання.

Під час моделювання пікового навантаження — коли 30–50 користувачів одночасно запускають формування документів — очікувано зростає навантаження на Celery-воркери. Саме вони виконують основний обсяг обчислень, пов’язаних із обробкою шаблонів DOCX та формуванням ZIP-архівів. У пікові моменти спостерігалося збільшення використання RAM до 550–680 МБ та короткочасні стрибки CPU до 3–4%. При цьому веб-інтерфейси залишалися стабільними й не демонстрували помітного збільшення часу відгуку.

Розподіл навантаження між сервісами виявився рівномірним і технічно виправданим. Веб-додатки відповідають за інтерфейсну логіку та більшість часу залишаються малонавантаженими; сервер бази даних працює у межах мінімальних ресурсів; Redis виконує роль швидкого брокера задач без істотного впливу на продуктивність. Водночас Celery забезпечує обробку ресурсомістких операцій, не блокуючи веб-інтерфейс і не створюючи затримок у роботі користувачів.

Результати випробувань свідчать, що обрана архітектура — винесення складних операцій у фоновий режим при збереженні легкого й відгукового веб-інтерфейсу — повністю себе виправдала. Система демонструє високу стійкість до пікових навантажень і здатна підтримувати одночасну роботу десятків адміністраторів та сотень абітурієнтів без деградації продуктивності.

5.3. Вплив та можливі шляхи розвитку

Впроваджена система суттєво впливає на організацію роботи приймальної комісії, оптимізуючи як технічні, так і адміністративні процеси. Уже на початкових етапах використання система демонструє здатність впорядковувати документообіг, прискорювати обробку інформації, зменшувати навантаження на персонал та підвищувати якість взаємодії з абітурієнтами. Такий підхід змінює внутрішню культуру університету, сприяючи розвитку цифрової грамотності та формуванню ефективної моделі управління.

Подальші можливості розвитку системи є широкими і відкривають шлях до створення комплексного цифрового рішення для організації вступу. Одним із ключових напрямів є інтеграція з Єдиною державною електронною базою з питань освіти, що дозволить автоматизувати обмін даними та забезпечити повну відповідність нормативним вимогам. Розвиток аналітичного модуля надасть змогу генерувати прогностичні моделі конкурсного навантаження, формувати статистичні звіти та підтримувати прийняття управлінських рішень на основі даних.

Перспективним кроком є повна цифровізація документообігу із застосуванням кваліфікованих електронних підписів. Це дозволить мінімізувати паперові процеси та забезпечити юридичну значущість електронних документів. Додаткові переваги може надати створення мобільного застосунку, який дозволить абітурієнтам оперативно отримувати інформацію про статус заяв, переглядати створені документи та отримувати рекомендації щодо подальших дій.

З технічної точки зору подальший розвиток системи може бути спрямований на використання хмарних платформ та контейнерної оркестрації. Це забезпечить автоматичне масштабування, високу відмовостійкість і можливість обробляти тисячі паралельних запитів навіть у період максимальних навантажень. Таким чином, система має потенціал стати універсальним рішенням для широкого кола закладів освіти.

У цілому вплив проекту значно виходить за межі суто технічних покращень. Система формує нову модель організації вступної кампанії, у якій автоматизація стає ключовим елементом ефективності та прозорості.

ВИСНОВКИ

У магістерській роботі проведено дослідження та практичну реалізацію інформаційної системи автоматизації вступної кампанії закладу вищої освіти. У результаті виконано комплексний аналіз сучасних підходів до цифровізації цього процесу, визначено вимоги до захисту персональних даних, обґрунтовано вибір технологій і створено програмний комплекс, який автоматизує ключові операції приймальної комісії - від збору анкетних даних до формування договорів і звітів.

Аналіз предметної області дав змогу окреслити проблеми ручних процедур, серед яких висока трудомісткість, значна кількість повторних дій, ризики помилок у документах і відсутність централізованого джерела даних. Порівняння з національними та міжнародними платформами продемонструвало необхідність розроблення гнучкого рішення, здатного поєднувати вимоги українського законодавства, безпекові стандарти та зручність використання. На основі цього було сформовано вимоги до системи та параметри її подальшого впровадження.

Обґрунтування технологічного стеку дозволило сформувати збалансоване рішення: Flask забезпечує гнучкість веброзробки, PostgreSQL - стабільність роботи з даними, Celery і Redis - асинхронну обробку ресурсоємних операцій, а Docker - ізоляцію, відтворюваність та зручність розгортання. Така комбінація технологій дала змогу створити архітектуру, яка поєднує модульність, надійність і потенціал до подальшого масштабування.

Результатом роботи стало проєктування та реалізація системи, що складається з двох взаємодіючих вебдодатків: порталу абітурієнта та адміністративної панелі для співробітників. Система підтримує покрокове заповнення анкет, централізоване управління довідниками, валідацію даних, генерацію документів DOCX на основі шаблонів, формування ZIP-архівів, експорт звітів у форматах XLSX та CSV, а також механізми автентифікації, рольового доступу та логування дій. Реалізовані функціональні можливості

підтвердили придатність архітектурного рішення та ефективність використаних технологій.

Проведене тестування показало, що система забезпечує стабільний час відгуку веб-інтерфейсів, коректно обробляє пакетні операції, зберігає працездатність при паралельних зверненнях та дозволяє безпечно формувати велику кількість документів. Практичні випробування засвідчили зменшення навантаження на приймальну комісію, скорочення кількості помилок у договорах і підвищення швидкості обробки анкет. Економічний аналіз підтвердив доцільність упровадження системи: автоматизація дає можливість скоротити витрати часу співробітників, уникнути дублювання роботи та зменшити ризики, пов’язані з некоректним оформленням документів.

Наукова новизна роботи полягає у створенні комплексної архітектури на базі Flask, Celery, Redis і Docker, орієнтованої на автоматизацію вступної кампанії з урахуванням вимог кібербезпеки, особливостей обробки персональних даних та специфіки освітньої галузі. Практичне значення полягає у можливості впровадження системи в реальні умови роботи приймальних комісій, що сприятиме підвищенню прозорості процедур, покращенню сервісу для абітурієнтів і створить основу для подальшої цифрової трансформації закладів вищої освіти.

Перспективи розвитку системи передбачають інтеграцію з ЄДЕБО та іншими державними сервісами, розширення можливостей аналітики, впровадження інтелектуальних методів прогнозування і модулів автоматичного інформування, а також створення мобільних клієнтів для абітурієнтів. Ці напрямки дадуть змогу підвищити рівень цифрової зрілості університету та забезпечити комплексну підтримку всього циклу взаємодії вступника з освітньою установою.

Розроблена інформаційна система відповідає сучасним вимогам до програмного забезпечення у сфері освіти та демонструє реальні переваги впровадження автоматизації в роботу приймальної комісії. Вона створює

технологічну основу для подальшої модернізації вступних процедур та розвитку електронних сервісів у закладах вищої освіти України.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. ДСТУ 8302:2015 Інформація та документація. Бібліографічний опис. Загальні положення та правила складання. К.: ДП «УкрНДНЦ», 2016. 60 с.
2. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. Sebastopol: O'Reilly Media, 2018. 448 p.
3. Brown A. Docker for DevOps. San Francisco: Leanpub, 2020. 380 p.
4. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 10.02.2025).
5. Flask Documentation. URL: https://flask.palletsprojects.com/ (дата звернення: 10.02.2025).
6. Celery Documentation. URL: https://docs.celeryq.dev/ (дата звернення: 10.02.2025).
7. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 10.02.2025).
8. SQLAlchemy Documentation. URL: https://docs.sqlalchemy.org/ (дата звернення: 10.02.2025).
9. Коваль О. В., Семенов І. М. Інформаційні системи в освіті: навч. посіб. К.: Освіта, 2018. 256 с.
10. Петренко С. М. Автоматизація управлінських процесів у закладах вищої освіти // Вісник НУ «Львівська політехніка». 2019. № 5. С. 45–52.
11. Мельник В. П. Цифрова трансформація освіти: виклики та можливості // Інформаційні технології в освіті. 2020. № 2. С. 12–25.
12. ЄДЕБО: Єдина державна електронна база з питань освіти. URL: https://info.edbo.gov.ua/ (дата звернення: 10.02.2025).
13. Wikipedia python URL: https://uk.wikipedia.org/wiki/Python(дата звернення: 09.02.2025).
14. Python Documentation. URL: https://docs.python.org/3/ (дата звернення: 10.02.2025).
15. Redis Documentation. URL: https://redis.io/docs/ (дата звернення: 10.02.2025).

16. python-docx Documentation. URL: https://python-docx.readthedocs.io/ (дата звернення: 10.02.2025).
17. pandas Documentation. URL: https://pandas.pydata.org/docs/ (дата звернення: 10.02.2025).
18. Білецький В. І. Охорона праці та безпека життєдіяльності: навч. посіб. К.: Знання, 2019. 320 с.
19. Гринь В. М., Зайцев О. В., Коваленко І. С. Безпека інформаційних систем: підруч. К.: Освіта, 2020. 412 с.
20. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. New York: Wiley, 2015. 784 p.
21. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2012. 560 p.
22. Richardson L., Ruby S. RESTful Web Services. Sebastopol: O’Reilly Media, 2013. 440 p.
23. Martin R. Clean Code: A Handbook of Agile Software Craftsmanship. Boston: Prentice Hall, 2008. 464 p.
24. Beck K. Extreme Programming Explained. Boston: Addison-Wesley, 2019. 224 p.
25. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 2020. 395 p.
26. ISO/IEC 27001:2013. Information technology – Security techniques – Information security management systems – Requirements. Geneva: ISO, 2013. 40 p.
27. Семенов О. І., Бондаренко І. С. Оцінка ефективності інформаційних систем: метод. рекомендації. К.: КПІ ім. Ігоря Сікорського, 2019. 64 с.
28. Сидоренко О. М., Ковальчук М. В. Тестування веб-додатків: практичний підхід. К.: ВГ ВНV, 2021. 256 с.
29. Петренко С. В. (ред.) Розробка інформаційних систем для освіти: досвід та перспективи. К.: Освіта, 2022. 320 с.

30. Шевченко В. М., Коваль Т. О. Економічна ефективність впровадження інформаційних систем у закладах освіти // Економіка та управління навчальним закладом. 2019. № 4. С. 56–63.
31. Григоренко С. І., Ткаченко О. В. Проектування баз даних для освітніх систем // Вісник НТУ «ХПІ». 2020. № 1. С. 34–42.
32. Лисенко О. Ю. Інформаційні системи та технології в освіті: навч. посіб. К.: КНУ ім. Т. Шевченка, 2021. 280 с.
33. DreamApply - Student Recruitment Platform [Електронний ресурс].. -Режим доступу: https://www.dreamapply.com/ (дата звернення: 12.02.2025).
34. UniAssist - Application Service for International Students [Електронний ресурс]..- Режим доступу: https://www.uni-assist.de/ (дата звернення: 12.02.2025).
35. UCAS - Universities and Colleges Admissions Service [Електронний ресурс].. - Режим доступу: https://www.ucas.com/ (дата звернення: 12.02.2025).

ДОДАТКИ

Додаток А

Таблиця А.1 – Структура бази даних

Таблиця	Ключові поля	Призначення
users	id (PK), username, password_hash, is_admin, can_edit_students, created_at	Облікові записи співробітників системи. is_admin визначає адміністративні права, can_edit_students — право редагування анкет абітурієнтів. Паролі зберігаються у вигляді хешів (Flask-Bcrypt).
students	id (PK), full_name, birth_date, id_code, doc_series_number, issued_by, registration_address, actual_address, phone_number, email, specialty, study_form, degree, after_education, budget_or_paid, need_dormitory, gender, military_structure, fitness_for_service, military_rank, vos, military_branch, accounting_category, special_accounting, marital_status, self_contract, representative_full_name, representative_passport, representative_address, representative_id_code, representative_phone, representative_issued, os_programs, specialization, father_name, father_work, mother_name, mother_work, nationality, education, document_ed, work_b_study, valid_until, record_number, year_study, created_at, updated_at	Центральна таблиця анкетних даних абітурієнтів. Містить персональні, навчальні, військові та документальні відомості. Використовується для формування договорів і звітності.
price	id (PK), specialty, specialization, degree, after_what, full_part, first_course … twelve_course, sum, sum_in_words, acred, do_acred, credits, os_program, year_study, faculty, created_at	Довідник вартості освітніх програм і параметрів навчання. Використовується для автоматичного формування контрактів.

Додаток Б

Таблиця Б.1 – Повний список маркерів

Маркер	Опис / поле джерела
{FullName}	Повне ПІБ абітурієнта (students.full_name)
{surname}, {name}, {middle_name}	Прізвище, ім’я, по батькові
{BirthDate}	Дата народження
{IDCode}	РНОКПП
{DocSeriesNumber}	Серія та номер документа
{IssuedByDate}	Ким і коли видано
{RegistrationAddress}	Адреса реєстрації
{ActualAddress}	Фактична адреса
{PhoneNumber}	Номер телефону
{Email}	Електронна пошта
{Specialty}	Спеціальність
{StudyForm}	Форма навчання
{Degree}	Освітній ступінь
{AfterEducation}	Попередній освітній рівень
{BudgetOrPaid}	Форма фінансування
{NeedDormitory}	Потреба в гуртожитку
{Gender}	Стать
{MilitaryStructure}	Військовий облік
{MilitaryRank}	Військове звання
{VOS}	Військово-облікова спеціальність
{MilitaryBranch}	Рід військ
{AccountingCategory}	Категорія обліку
{SpecialAccounting}	Спеціальний облік
{MaritalStatus}	Сімейний стан
{SelfContract}	Сторона договору
{RepresentativeFullName}	ПІБ представника
{RepresentativePassport}	Паспорт представника
{RepresentativeAddress}	Адреса представника

Продовження табл. Б.1

Маркер	**Опис / поле джерела**
{RepresentativeIDCode}	РНОКПП представника
{RepresentativePhone}	Телефон представника
{RepresentativeIssued}	Ким видано документ представника
{OsvPrograms}	Освітня програма
{Specialization}	Спеціалізація
{year_study}	Роки навчання
{father_name}	ПІБ батька
{mother_name}	ПІБ матері
{father_work}	Місце роботи батька
{mother_work}	Місце роботи матері
{Nationality}	Національність
{Education}	Освіта
{DocumentED}	Документ про освіту
{WorkBStudy}	Робота під час навчання
{valid_until}	Термін дії документа
{record_number}	Реєстраційний номер
{FirstCourse} – {TwelveCourse}	Вартість навчання за курсами
{Sum}	Загальна сума
{SumInWords}	Сума прописом
{Accreditation}	Акредитація
{AccreditationUntil}	Термін акредитації
{Credits}	Кількість кредитів
{OsProgram}	Освітня програма
{StudyDuration}	Тривалість навчання
{Faculty}	Факультет
{ContractPerson}	Сторона договору
{ContractPassport}	Паспорт сторони договору
{ContractAddress}	Адреса сторони договору
{ContractIDCode}	РНОКПП сторони договору
{ContractPhone}	Телефон сторони договору

Додаток В

Фрагменти реалізації асинхронної генерації документів

Нижче наведено повні фрагменти програмного коду, що реалізують механізм асинхронної генерації документів у форматі DOCX та ZIP з використанням Celery і Redis.

В.1 Ініціювання асинхронної задачі з вебінтерфейсу

```
@app.route('/generate_doc/<int:id>', methods=['GET', 'POST'])
@login_required
def generate_doc(id):
template_names = request.form.getlist('template') or
(request.args.get('templates') or '').split(',')
if not template_names or template_names == ['']:
return "Не вибрано шаблон", 400
task = generate_docx_task.delay(id, template_names)
return jsonify({"task_id": task.id}), 202
```

В.2 Celery-задача генерації DOCX або ZIP-архіву

```
@celery.task(bind=True)
def generate_docx_task(self, student_id, template_names):
student = Data.query.get(student_id)
data = prepare_template_data(student.to_dict())
task_id = self.request.id

if len(template_names) > 1:
zip_buffer = io.BytesIO()
with zipfile.ZipFile(zip_buffer, 'w', zipfile.ZIP_DEFLATED) as zipf:
for tmpl in template_names:
path = os.path.join(TEMPLATE_DIR, tmpl)
if not os.path.exists(path):
continue
doc = fill_certificate(path, data, f"{data['full_name']}_{tmpl}_{task_id}.docx")
zipf.writestr(f"{data['full_name']}_{tmpl}_{task_id}.docx", doc.getvalue())
out_path = f"/web/tmp/{data['full_name']}_документи_{task_id}.zip"
with open(out_path, "wb") as f:
f.write(zip_buffer.getvalue())
return os.path.basename(out_path)
else:
tmpl = template_names[0]
path = os.path.join(TEMPLATE_DIR, tmpl)
doc = fill_certificate(path, data, f"{data['full_name']}_{tmpl}_{task_id}.docx")
```

```
out_path = f"/web/tmp/{data['full_name']}_{tmpl}_{task_id}.docx"
with open(out_path, "wb") as f:
f.write(doc.getvalue())
return os.path.basename(out_path)
```

В.3 Перевірка статусу виконання асинхронної задачі

```
@app.route('/task_status/<task_id>')
def task_status(task_id):
task = celery.AsyncResult(task_id)
if task.state == 'SUCCESS':
return jsonify({"status": "ready", "file": task.result})
return jsonify({"status": task.state})
```

В.4 Завантаження згенерованого документа

```
@app.route('/download_doc/<path:filename>')
@login_required
def download_doc(filename):
abs_path = safe_join('/web/tmp', filename)
return send_file(abs_path, as_attachment=True)
```