НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук

/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«_02_»__червня_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення інформаційної системи обліку випускників університету»

Спеціальність 121 «Інженерія програмного забезпечення»

ОПП «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент / Вайганг Г.О./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи : / Голуб Б.Л /
підпис ПІБ

Виконав: / Драгальчук Б.Е./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис

" 16 " грудня 2024 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Драгальчуку Богдану Едуардовичу

Спеціальність 121 «Інженерія програмного забезпечення»

ОПП «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення інформаційної системи обліку випускників університету

Затверджена наказом ректора НУБіП України № 2249 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . 05 . 25
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Процес обліку випускників як об'єкт дослідження.
2. Інформаційне забезпечення.
3. Програмне забепечення.
4. Рекомендації щодо впровадження та експлуатації.
5. Висновки.

Керівник бакалаврської кваліфікаційної роботи_____ / Голуб Б.Л. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____ / Драгальчук Б.Е./
підпис ініціали та прізвище

Дата отримання завдання 2024 . 12 . 16
рік, місяць, число

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

СУБД – система управління базою даних;

ПЗ – програмне забезпечення;

НФ – нормальна форма;

SQL – Structured Query Language;

ОС – операційна система;

ЖЦ – життєвий цикл;

HTML – HyperText Markup Language;

CSS – Cascading Style Sheets.

ВСТУП

Актуалність. Сучасні вищі навчальні заклади постійно стикаються з потребою ефективного управління інформацією про своїх випускників. Така інформація є важливою не лише для аналітичної та статистичної звітності, але й для налагодження співпраці з роботодавцями, моніторингу кар'єрного шляху випускників, а також для формування репутації закладу освіти на ринку освітніх послуг. Проте на практиці дані про випускників часто зберігаються у вигляді розрізнених документів або таблиць, що ускладнює доступ, аналіз і супровід інформації. Тому з'явилася потреба у створені єдиної інформаційної системи обліку випускників університету.

Мета розробки. Основною метою розробки програмного забезпечення інформаційної системи обліку випускників університету є створення зручного інструменту для ефективного зберігання, оновлення та перегляду інформації про випускників, їх спеціальності, факультети та працевлаштування. Система покликана сприяти налагодженому інформаційному обміну між підрозділами, забезпечити узгодженість та цілісність даних, а також надати можливість користувачам здійснювати доступ до актуальної інформації відповідно до своїх повноважень. Такий підхід дозволяє підвищити ефективність роботи навчального закладу з даними випускників, що є важливим як з організаційної, так і з аналітичної точки зору.

Методи та технології. Під час розробки інформаційної системи обліку випускників університету використовувався сучасний стек технологій, орієнтований на масштабованість, стабільність та зручність у супроводі. Основною мовою програмування було обрано Python, що забезпечує високу продуктивність розробки логіки зі сторони сервера. Для побудови веб-застосунку було використано фреймворк Django, який дозволяє швидко створювати масштабовані та безпечні серверні додатки з чітким поділом на компоненти.

Для зберігання та обробки даних застосовано PostgreSQL — реляційну систему управління базами даних, що підтримує складні запити, транзакції та розмежування доступу на рівні користувачів.

Інтерфейс системи побудовано за допомогою стандартних засобів HTML/CSS та шаблонізаторів Django. Такий підхід дозволив створити простий, функціональний та інтуїтивно зрозумілий веб-інтерфейс без залучення додаткових клієнтських фреймворків.

Апробація програмного продукту.

Результати розробки продукту були представлені на VII Всеукраїнській науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем» 2025 у вигляді тез доповіді з назвою «Програмне забезпечення інформаційної системи обліку випускників університету».

Структура записки. Даний дипломний проєкт складається з наступних частин: вступ, чотири розділи та висновки. **Перший розділ** – «Процес обліку випускників як об'єкт дослідження» – містить аналітичну частину. У ньому розглянуто проблеми ведення обліку в університетах, сформульовано вимоги до системи, змодельовано предметну область, проведено аналіз аналогів, а також сформульовано постановку задачі для розробки системи.

Другий розділ – «Інформаційне забезпечення» – присвячений моделюванню даних та побудові архітектури бази. В ньому представлено логічну модель даних у вигляді ER-діаграми, описано принципи нормалізації до 3НФ, обґрунтовано вибір PostgreSQL як СУБД, наведено SQL-запити створення таблиць та реалізацію структури у середовищі Django ORM.

Третій розділ – «Програмне забезпечення» – зосереджений на моделюванні структури, проєктуванні та реалізації функціональних модулів системи. У ньому наведено UML-діаграми класів, кооперацій, пакетів та компонентів, які відображають логіку, взаємодію об'єктів та архітектуру системи.

Четвертий розділ – «Рекомендації щодо впровадження та експлуатації системи» – містить опис середовища запуску, інструкції з встановлення програмного забезпечення, склад інсталяційного пакету, вимоги до апаратного та програмного забезпечення, а також результати тестування.

Пояснювальна записка містить 69 сторінок основного тексту, 21 рисунок, 1 таблицю, список використаних джерел з 27 позицій та 7 додатків.

1 ПРОЦЕС ОБЛІКУ ВИПУСКНИКІВ ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ

1.1 Опис процесу обліку випускників

На сьогоднішній день у більшості закладів вищої освіти процес ведення обліку випускників є фрагментованим і переважно ручним. Як правило, кожен факультет або кафедра самостійно збирає інформацію про своїх випускників у вигляді електронних таблиць, текстових документів або навіть паперових анкет. Ці дані зберігаються локально — на комп'ютерах відповідальних осіб чи в окремих внутрішніх архівах підрозділів. У результаті формується значна кількість незалежних і несумісних джерел даних.

У такій організації обліку відсутній єдиний централізований механізм збору, перевірки, зберігання та оновлення інформації. Якщо один і той самий випускник був внесений до баз даних кількох підрозділів (наприклад, навчався на кількох програмах), це призводить до дублювання записів і розбіжностей у вказаних даних. Працівникам доводиться витрачати багато часу на ручну звірку, уточнення, телефонні дзвінки чи запити до інших кафедр.

Особливо складною є підготовка звітності. Для того щоб створити зведену інформацію про випускників певного року, спеціальності чи факультету, відповідальні особи змушені збирати окремі таблиці від кожного підрозділу, вручну їх об'єднувати, узгоджувати структуру полів, перевіряти актуальність даних. Такий підхід не тільки потребує значних людських ресурсів, а й спричиняє ризики помилок, втрати важливої інформації або спотворення статистики.

Ще однією проблемною ділянкою є облік працевлаштування випускників. Ці дані зазвичай збираються епізодично — або через неформальне опитування, або за ініціативою окремих кафедр. Часто це відбувається без уніфікованої структури й без подальшої систематизації. Унаслідок цього адміністрація університету не має повної картини щодо кар'єрної динаміки випускників, що

ускладнює подання звітів, участь у рейтингах та оцінювання результативності освітніх програм.

Предметна область даної роботи охоплює автоматизацію процесів обліку випускників: від первинного введення даних до їх подальшого аналізу. До типових даних, що підлягають обліку, належать повне ім’я випускника, рік завершення навчання, факультет, спеціальність, контактна інформація, а також відомості про подальше працевлаштування — назва компанії, посада, рік початку роботи.

Основними користувачами системи є представники адміністративних підрозділів — зокрема, методисти кафедр, які відповідають за ведення й оновлення інформації, та керівники (завідувачі кафедр), які мають доступ до перегляду даних, формування звітів і контролю за наповненням системи. Різниця в рівні доступу до функціоналу залежить від їхніх посадових обов’язків.

1.2 Аналіз вимог до предметної області

Для побудови ефективної інформаційної системи обліку випускників потрібно врахувати як функціональні так і нефункціональні вимоги, що випливають із потреб кінцевих користувачів та специфіки даної предметної області.

Функціональні вимоги

- **Реєстрація нових випускників.** Система повинна дозволяти створення запису про випускника з базовими даними: ПІБ, рік випуску, факультет, спеціальність, контактна інформація.
- **Редагування даних випускника.** Користувачі мають мати можливість оновлювати або доповнювати інформацію.

- **Фіксація даних про працевлаштування.** Для кожного випускника має зберігатись інформація про компанію, посаду та інші дані потрібні для певного підрозділу університету.

- **Фільтрація та пошук.** Потрібно передбачити пошук випускників за ПІБ, роком випуску, спеціальністю, факультетом, наявністю працевлаштування.

- **Розмежування доступу.** В функціональності системи ключовою є наявність ролей користувачів з широкими можливостями щодо оперування даними та ролей з обмеженими можливостями лише для перегляду інформації.

- **Формування звітності.** Система має забезпечувати виведення статистики за факультетами, спеціальностями, роками випуску та працевлаштованістю.

Нефункціональні вимоги

- **Зручність інтерфейсу.** Інтерфейс має бути інтуїтивно зрозумілим та відповідати загальним вимогам проектування UI/UX, без перевантаження елементами.

- **Масштабованість.** Система має підтримувати зберігання інформації про десятки тисяч випускників без втрати продуктивності.

- **Надійність**. Дані мають зберігатись у стабільному середовищі, із захистом від втрат.

- **Переносимість.** Рішення має бути придатним для встановлення на сервер університету або в хмарному середовищі.

- **Мінімізація адміністративних витрат.** Завдяки простій архітектурі система не потребує складного супроводу.

- **Захист від несанкціонованого доступу.** Вхід до системи повинен обмежуватись на рівні бази даних залежно від ролі користувача.

Аналіз вимог показує, що ключовими аспектами є централізований доступ до даних, чіткий розподіл прав між користувачами, простота у користуванні та можливість формування певної аналітичної звітності.

1.3 Моделювання предметної області

Моделювання предметної області є одним з ключових етапів проєктування, він дозволяє формалізувати структуру системи, взаємодію між об’єктами та поведінку системи при виконанні різних сценаріїв. Це забезпечує чітке бачення логіки функціонування інформаційної системи ще до її реалізації.

У рамках цього проєкту для опису предметної області було використано такі типи UML-діаграм:

- діаграма варіантів використання (Use Case Diagram);
- діаграма класів (Class Diagram);
- діаграма активності (Activity Diagram);
- діаграма послідовності (Sequence Diagram).

1.3.1 Діаграма варіантів використання (Use Case Diagram) — демонструє основні сценарії взаємодії користувачів системи (методиста та керівника) з функціональністю системи: створення, редагування, перегляд даних випускників, формування звітів, пошук та фільтрація. На рис. 1 зображено діагарму варіантів використання системи.

Актори системи

1. Методист – основний користувач системи з повним набором прав. Відповідає за внесення, оновлення та супровід даних про випускників.

Дії, доступні методисту:

- створення нових записів про випускників;

- редагування наявних даних;
- видалення записів у разі помилок або дублювання;
- імпорт списку випускників з зовнішніх джерел (наприклад, таблиць Excel);
- експорт бази випускників для збереження або передавання;
- перегляд та пошук інформації, спільно з керівником.

2. Керівник (завідувач кафедри)

Користувач із обмеженими правами доступу. Має можливість лише переглядати дані та формувати звітність.

Дії, доступні керівнику:

- пошук випускників за критеріями (ПІБ, спеціальність, рік випуску тощо);
- перегляд детальної інформації про конкретного випускника;
- перегляд статистики щодо працевлаштування;
- генерація аналітичних та статистичних звітів.

Прецеденти системи

Прецеденти, спільні або взаємопов’язані:

- **пошук випускників** — загальний сценарій для обох ролей, що дозволяє фільтрацію даних за заданими параметрами;
- **перегляд детальної інформації про випускника** — розширення (extend) до сценарію пошуку: стає доступним лише після виконання пошуку.

Прецеденти для методиста:

- **додавання випускника** — створення нового запису;
- **редагування даних випускника** — оновлення наявного запису;
- **видалення випускника** — повне вилучення запису з бази;

- **імпорт списку випускників** — пакетне завантаження даних;
- **експорт списку випускників** — вивантаження бази в зовнішній файл.

Прецеденти для керівника:

- **перегляд статистики працевлаштування** випускників — аналітика за напрямами, роками, спеціальностями;
- **генерація статистичних звітів** — формування документів на основі наявних даних.

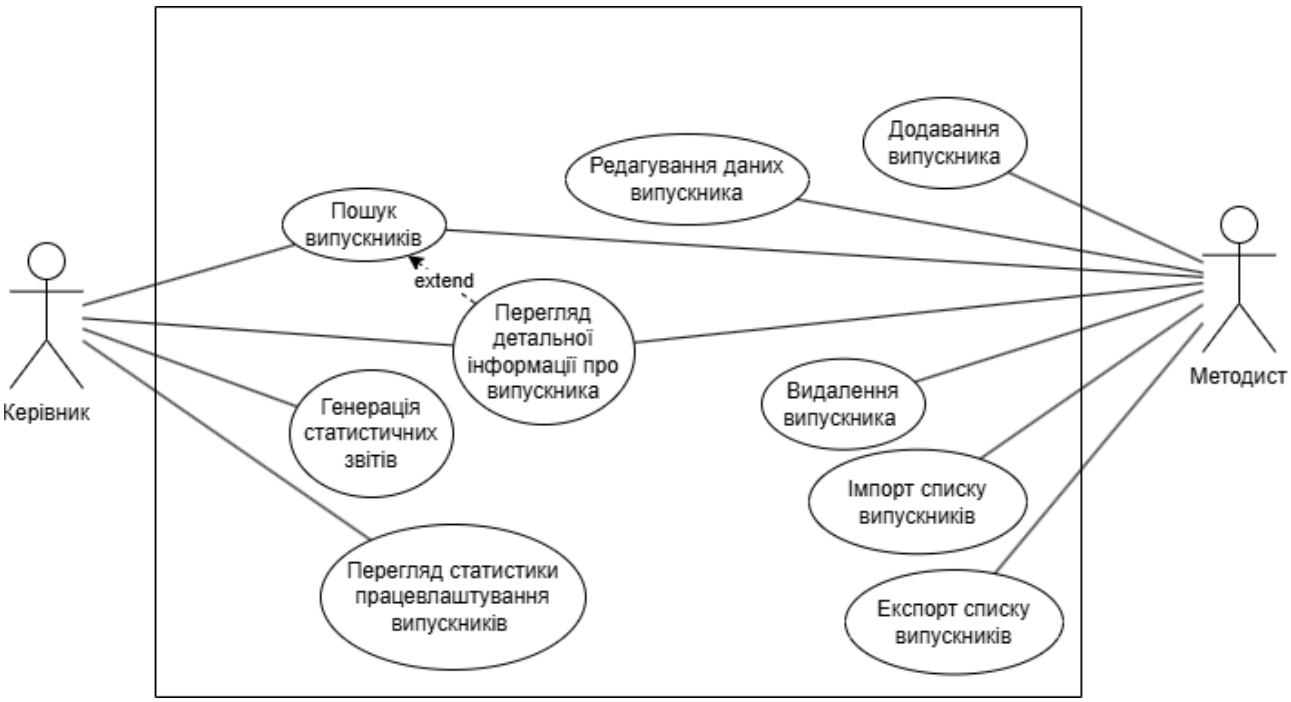


Рис. 1 Діаграма варіантів використання

1.3.2 Діаграма послідовності (Sequence Diagram) — показує взаємодію між об’єктами системи під час виконання певного сценарію, наприклад, збереження даних про працевлаштування або формування звіту. Цей тип діаграми деталізує обмін повідомленнями між компонентами в часі.

На діаграмі демонструється взаємодія між користувачами системи (акторами) — **керівником** і **методистом** — та компонентами системи під час виконання основних сценаріїв. Вона відображає послідовність викликів операцій (запитів) і відповіді на них у часі.

Об’єкти

- **Керівник** — має доступ лише до операцій перегляду та аналітики.
- **Методист** — користувач із повними правами, включно з редагуванням та імпортом/експортом даних.

Послідовності взаємодії

Дії керівника

1. **Запит пошуку** → надсилається до підсистеми фільтрації випускників.
2. **Результат запиту** ← система повертає перелік знайдених записів.
3. **Запит детальної інформації** → надсилається до модуля перегляду конкретного випускника.
4. **Результат запиту** ← повертається розгорнута інформація.
5. **Генерація статистичних звітів** → викликає відповідний модуль.
6. **Перегляд статистики працевлаштування випускників** → доступ до статистичної інформації.

Дії методиста

7. **Додавання випускника** → безпосереднє звернення до модуля збереження нових записів.
8. **Редагування даних випускника** → виклик оновлення запису.
9. **Видалення випускника** → виклик видалення з бази даних.
10. **Імпорт списку випускників** → пакетне завантаження з файлу.
11. **Фільтрація випускників / Пошук випускників / Перегляд детальної інформації** — аналогічно до сценаріїв керівника.

12. **Експорт списку випускників** → формування звіту або таблиці для вивантаження в зовнішній файл.

Діагарму послідовності продемонстровано в додатку А.

1.3.3 Діаграма активності (Activity Diagram) — ілюструє послідовність дій користувача в процесі виконання типових операцій, таких як: додавання нового випускника, редагування запису, перегляд статистики. Це дозволяє описати поведінку системи на рівні логіки дій.

Діаграма активності ілюструє логіку роботи двох користувачів системи — **методиста** та **керівника** — у процесі виконання основних операцій із базою даних випускників.

Смуга методиста

Початок процесу: стартова точка (чорне коло) розгалужується на два варіанти.

1. **Імпорт списку випускників** — методист завантажує дані з зовнішнього джерела (наприклад, Excel-файл).

   - Якщо **імпорт неуспішний**, процес припиняється.
   - Якщо **імпорт успішний**, методист переходить до редагування окремих записів.

2. **Додавання випускника вручну** — альтернатива імпорту, коли дані вносяться через форму системи.

Після імпорту або ручного додавання:

- **редагування випускника** — внесення змін до збережених даних;
- **видалення випускника** — при потребі очищення або виправлення помилкового запису;
- **експорт списку** — формування файлу зі списком випускників.

Смуга керівника

Процес починається з:

- **пошуку випускників** — введення критеріїв пошуку;
- **фільтрації випускників** — система застосовує задані фільтри.

Далі можливі варіанти:

- якщо **випускників не знайдено**, керівник може змінити параметри фільтрації;
- якщо **випускників знайдено**, система переходить до:
  - **генерації статистики** — створення аналітичних зведень;
  - **експорту звіту** — збереження результатів у вигляді файлу.

Діагарму послідовності продемонстровано в додатку Б.

1.4 Огляд інформаційних джерел та існуючих рішень

У рамках даного дипломного проекту було проаналізовано ряд програмних рішень, спрямованих на забезпечення обліку випускників та моніторингу їхнього кар'єрного шляху. Два з розглянутих рішень, Gradnet та Hivebrite, демонструють значний потенціал у контексті поставлених завдань.

Gradnet позиціонується як платформа для управління відносинами з випускниками, що пропонує комплекс інструментів для підтримки зв'язку, залучення та розвитку спільноти випускників[1].

Основні функціональні можливості Gradnet.

- **Розширений каталог випускників:** забезпечує структуровану базу даних випускників з можливістю пошуку та фільтрації за різними критеріями.
- **Персоналізований вебсайт для асоціації випускників:** можливість створення власного онлайн-представництва для організації випускників.

- **Розширена аналітика:** надання даних та звітів щодо активності випускників та ефективності комунікаційних кампаній.

- **Публікація історій успіху випускників:** інструменти для демонстрації досягнень випускників з метою натхнення та залучення інших.

- **Доступ до ексклюзивних вакансій:** можливість публікації та перегляду пропозицій роботи, доступних лише для випускників.

На рис. 2 зображено вигляд програми Gradnet.

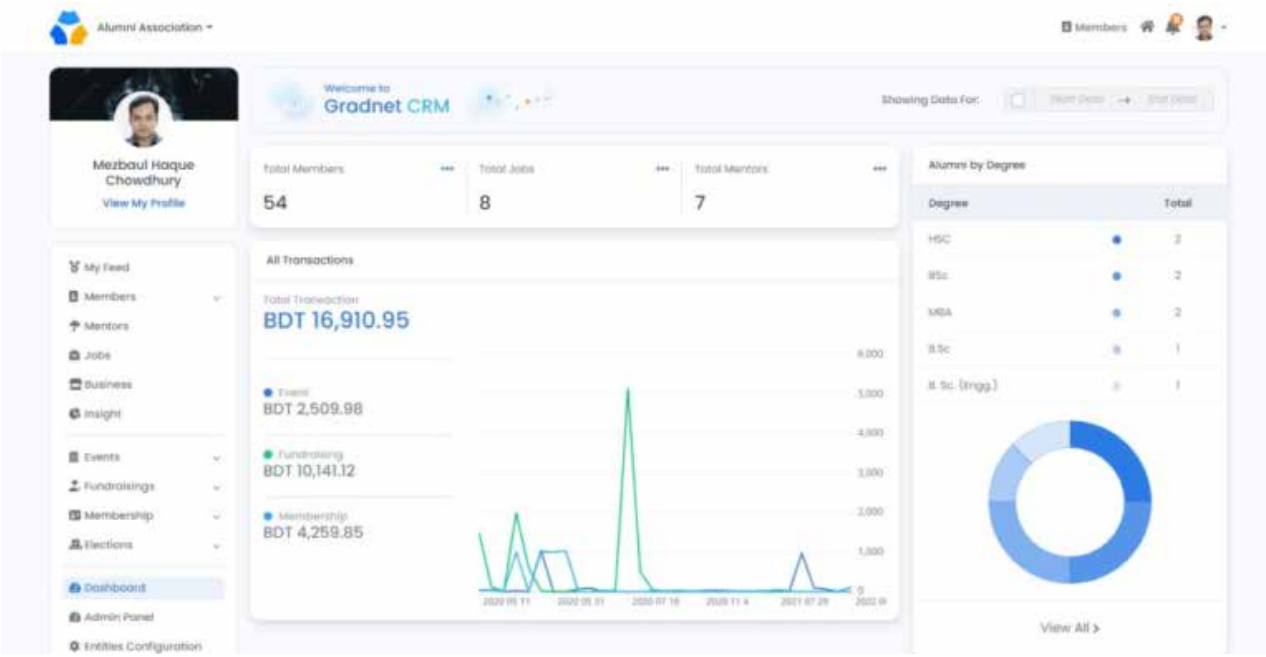


Рис. 2 Інтерфейс програма Gradnet

Аналіз функціональних можливостей Gradnet дозволяє виділити наступні ключові сильні сторони:

- **сприяння налагодженню зв'язку та професійних мереж** – платформа ефективно об'єднує випускників, створюючи можливості для спілкування та обміну досвідом;

- **ефективне залучення випускників** – різноманітні інструменти дозволяють підтримувати інтерес випускників до діяльності навчального закладу та асоціації;

- **підтримка кар’єрного розвитку** – наявність інструментів для пошуку вакансій, менторства та створення резюме сприяє професійному зростанню випускників.

На основі доступної інформації, конкретні слабкі сторони Gradnet не були чітко ідентифіковані в ході первинного дослідження. Подальший аналіз відгуків користувачів та порівняння з альтернативними рішеннями може виявити потенційні обмеження або недоліки платформи.

Hivebrite є комплексною платформою для управління спільнотами, яка може бути ефективно використана для організації та підтримки спільноти випускників[2].

Ключові функціональні можливості Hivebrite.

- **Інструменти для залучення учасників:** форуми, стрічки активності, розсилки новин та елементи гейміфікації для підвищення залученості членів спільноти.

- **Пошук учасників:** розширені можливості пошуку та фільтрації учасників за різними критеріями.

- **Управління профілями учасників:** можливості для налаштування профілів, управління підписками та рівнями доступу.

- **Управління групами:** можливість створення та управління тематичними групами за інтересами або професійними напрямками.

- **Аналітика та звітування:** надання даних про користувачів та ефективність різних інструментів платформи.

Hivebrite демонструє ряд значних переваг:

- **багатофункціональність** – платформа пропонує широкий спектр інструментів для управління спільнотою випускників, охоплюючи різні аспекти їхньої взаємодії з навчальним закладом та між собою;

- **інтеграційні можливості** – підтримка інтеграції з іншими важливими системами забезпечує ефективний обмін даними та оптимізацію робочих процесів;

- **безпека та конфіденційність даних** – забезпечення високого рівня захисту інформації користувачів є пріоритетом платформи;

- **масштабованість –** платформа здатна адаптуватися до зростання спільноти випускників та збільшення обсягу даних;

Незважаючи на значні переваги, були виявлені деякі потенційні слабкі сторони Hivebrite:

- **складність для новачків** – широкий набір функцій може ускладнити процес освоєння платформи для нових користувачів;

- **повідомлення про технічні збої** – окремі користувачі повідомляли про випадкові технічні проблеми в роботі платформи;

- **обмежена аналітика** – деякі користувачі вважають, що можливості аналізу даних, які надає платформа, є недостатньо глибокими.

Обидва розглянуті рішення, Gradnet та Hivebrite, пропонують значний набір інструментів для обліку випускників та підтримки зв'язку з ними. Gradnet фокусується на специфічних потребах асоціацій випускників, пропонуючи функції, орієнтовані на кар'єрний розвиток. Hivebrite, у свою чергу, є більш універсальною платформою для управління спільнотами, яка вирізняється широким спектром інструментів для залучення користувачів.

1.5 Постановка завдання

Метою цього дипломного проекту є розробка програмного забезпечення для інформаційної системи обліку випускників університету. З урахуванням виявлених проблем у даній галузі, проведення аналізу вимог та досліджень аналогічних систем, було сформульовано наступне технічне завдання.

В системі потрібно реалізовувати наступні функції:

- зберігання інформацію про випускників в одному місці включаючи їх ПІБ(ПІБ), рік закінчення навчання в університеті, факультет на якому вони навчалися та спеціальність яку отримали разом з контактною інформація;
- ведення списку працевлаштувань;
- фільтрація та пошук за різними параметрами;
- підготовка статистичних звітів на основі визначених критерій;
- впровадження контролю доступу, що визначаються роллю користувача: повний доступ для методиста та режим тільки перегляду для керівника;
- забезпечення безпечного зберігання і захисту даних від несанкціонованого доступу.

Одним з важливих завдань є створення простого веб-інтерфейсу для зручності користувачів у взаємодії з базою даних. Вони можуть швидко додавати, переглядати та оновлювати інформацію. Особливістю архітектурного підходу є впровадження контролю доступу на рівні бази даних шляхом реалізацій логіки ролей зімсть універсальної таблиці користувачів.

У рамках проєкту планується розроблення багаторівневої архітектури веб-додатку із наступними складовими:

- рівень відображення (інтерфейс);
- рівень логіки (обробка запитів, бізнес-логіка);
- рівень доступу до інформаційних ресурсів (робота з базою даних PostgreSQL).

Очікуваний результат - готовий до впровадження веб-додаток з можливістю розширення та інтеграції у робочий процес факультетів університету.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Логічна модель даних є концептуальним поданням інформаційної структури системи, незалежним від конкретної реалізації в системі управління базами даних (СУБД). Вона описує сутності предметної області, їхні атрибути та зв’язки між ними. Побудова логічної моделі дає змогу чітко сформувати структуру збереження даних, забезпечити їх узгодженість, а також полегшує подальше проєктування фізичної моделі та розробку запитів.

У межах даної системи було визначено такі ключові сутності:

- **факультет** — структурний підрозділ університету, який об’єднує кафедри та спеціальності;
- **кафедра** — підрозділ факультету, відповідальний за навчальні програми і спеціалізації;
- **спеціальність** — конкретна освітня програма, яку проходив випускник;
- **випускник** — студент, що завершив навчання; зберігає персональні та академічні дані (ПІБ, контактна інформація, рік випуску тощо);
- **працевлаштування** — сутність, яка описує професійну діяльність випускника після завершення навчання. Містить інформацію про назву компанії, в якій працює випускник, займану посаду та рік початку роботи;

Нормалізація даних

Нормалізація — це процес структурування реляційної бази даних з метою усунення надлишковості, уникнення аномалій оновлення та забезпечення логічної цілісності. Процес нормалізації полягає у поділі таблиць та правильному встановленні зв’язків між ними на основі функціональних залежностей.

У реляційній моделі бази даних прийнято виділяти кілька форм нормалізації (НФ). У даному випадку логічна модель відповідає **трьом першим нормальним формам**, що є найважливішими з практичної точки зору.

Перша нормальна форма (1НФ)

Відповідність моделі 1НФ означає, що:

- кожна таблиця має чітко визначений первинний ключ;
- всі атрибути містять атомарні значення (одне значення в кожній комірці);
- повторювані групи виключені.

У розроблюваній моделі:

- атрибути, наприклад full_name, email, graduation_year у таблиці випускників — є неподільними;
- кожна сутність має унікальний ідентифікатор.

Друга нормальна форма (2НФ)

Модель перебуває у 2НФ, якщо вона задовольняє вимоги 1НФ, і всі атрибути, які не належать до ключа, повністю функціонально залежать від **повного первинного ключа** (тобто не лише від частини ключа, у разі складених ключів).

У цій моделі:

- первинні ключі — прості (автоматичні ID);
- немає часткових залежностей: наприклад, у таблиці випускників всі атрибути залежать лише від id, а не від частини комбінованого ключа.

Третя нормальна форма (3НФ)

Модель відповідає 3НФ, якщо вона у 2НФ і не містить **транзитивних залежностей**, тобто жоден неключовий атрибут не залежить від іншого неключового атрибута.

У моделі:

- наприклад, інформація про факультет, кафедру, спеціальність розділена на окремі таблиці;
- у таблиці випускників зберігається лише зовнішній ключ на спеціальність, а назви факультетів і кафедр не дублюються — вони зберігаються в окремих пов’язаних таблицях;
- отже, транзитивні залежності усунено.

Переглянути побудовану логічну модель можна в додатку В.

Таким чином, логічна модель побудована у відповідності до вимог 1НФ, 2НФ та 3НФ, що дозволяє ефективно управляти даними, підтримувати їх цілісність, уникати надлишковості та забезпечити масштабованість системи в майбутньому.

2.2 Вибір та обґрунтування СУБД

Система управління базами даних (СУБД) є критично важливим компонентом будь-якої інформаційної системи, що відповідає за збереження, обробку, цілісність і доступність даних. Для розробки системи обліку випускників університету було розглянуто чотири популярних СУБД: Oracle, MySQL, Microsoft SQL Server та PostgreSQL.

Oracle Database — це потужна комерційна об’єктно-реляційна СУБД, розроблена компанією Oracle Corporation. Вона орієнтована на великі підприємства та організації з високими вимогами до продуктивності, доступності й безпеки даних[3].

Oracle підтримує зберігання не лише реляційних, але й об’єктних типів даних, має багаторівневу архітектуру з високим рівнем масштабованості та

надійності. СУБД часто використовується у критично важливих системах: банківських, фінансових, телекомунікаційних.

Переваги:

- підтримка кластерних рішень (Oracle RAC), реплікації, відмовостійкості та резервного копіювання на рівні індустріального стандарту;
- розширена підтримка PL/SQL — потужної вбудованої мови програмування;
- гнучке налаштування безпеки (рольова модель, шифрування на рівні стовпців, аудит доступу);
- підтримка розподілених транзакцій та складних аналітичних запитів.

Недоліки:

- надзвичайно висока вартість ліцензії для підприємств;
- складна архітектура й вимогливість до ресурсів сервера;
- обмежена відкритість екосистеми, залежність від інструментів Oracle.

MySQL — одна з найвідоміших СУБД з відкритим кодом, яка використовується переважно у веб-розробці. Вона належить компанії Oracle, проте зберігає безкоштовну базову версію з GPL-ліцензією[4].

MySQL має просту архітектуру, підтримку SQL, реплікації, зручні GUI-інструменти (наприклад, MySQL Workbench). Система широко застосовується у CMS (WordPress, Joomla), e-commerce (Magento), корпоративних інтернет-проєктах.

Переваги:

- простота у встановленні та обслуговуванні;
- висока продуктивність для простих запитів;

- широка підтримка хостинг-провайдерів та інтеграцій з популярними фреймворками (Laravel, Symfony);
- підтримка реплікації та кластерів (Galera Cluster).

Недоліки:

- менша відповідність стандарту SQL, ніж у PostgreSQL або Oracle;
- погана робота зі складними транзакціями в режимі MyISAM (InnoDB — більш стабільний варіант, але вимагає налаштування);
- обмежені аналітичні функції та слабка підтримка CTE, віконних функцій.

Microsoft SQL Server — комерційна реляційна СУБД, тісно інтегрована з екосистемою Windows. Вона надає користувачам хороші засоби адміністрування, аналітики та розробки[5].

SQL Server використовує T-SQL (розширення SQL), має вбудовані засоби побудови звітів (SSRS), обробки ETL (SSIS) та аналітики (SSAS). Вона популярна у корпоративних середовищах, де використовуються продукти Microsoft — наприклад, у зв’язці з .NET-платформою.

Переваги:

- інтуїтивно зрозумілий GUI через SQL Server Management Studio (SSMS);
- сильна підтримка бізнес-аналітики, індексів, віконних функцій, тригерів;
- можливість інтеграції з Microsoft Excel, Power BI та іншими інструментами звітності;
- підтримка інструментів для CI/CD та версіонування скриптів.

Недоліки:

- платна модель з обмеженнями у безкоштовній Express-версії;
- основна підтримка під Windows, обмежена стабільність у Linux;
- закрита екосистема з прив’язкою до інших продуктів Microsoft.

PostgreSQL — потужна об’єктно-реляційна СУБД з відкритим кодом, орієнтована на розробників, які потребують гнучкості та функціональності. Вона вважається однією з найбільш відповідних стандарту SQL систем та часто використовується у фінансових системах, GIS-рішеннях, складній бізнес-логіці[6].

PostgreSQL — це не просто класична реляційна СУБД, а набагато ширший інструмент, який підходить для розробки сучасних застосунків. Завдяки підтримці JSONB він дозволяє зберігати неструктуровані дані та при цьому забезпечує швидкий пошук та індексацію. Крім того, його можна розширювати за допомогою власних типів даних, наприклад, для роботи з геоданими через PostGIS. Завдяки функціям, тригерам та процедурам можна реалізувати складну бізнес-логіку прямо в базі. А велика кількість розширень робить PostgreSQL гнучким і надійним рішенням для найрізноманітніших завдань.

Переваги:

- повна підтримка SQL:2011, віконних функцій, CTE, тригерів, stored procedures;
- підтримка типів JSON, масивів, XML, геоданих (PostGIS);
- можливість створення користувацьких типів даних, функцій, індексів;
- відкрита ліцензія, активна спільнота, регулярні оновлення;
- надійність і масштабованість (наприклад, використовується у системах Instagram, Reddit, Skype).

Недоліки:

- вищий поріг входу через розвинену термінологію та конфігурації;
- вимагає налаштування для досягнення максимальної продуктивності;
- менше інструментів «із коробки» для візуального керування, ніж у MSSQL.

Таблиця 1

Порівняльна таблиця

Параметр	**Oracle**	**MySQL**	**MS SQL Server**	**PostgreSQL**
Тип ліцензії	Комерційна	Відкрита	Комерційна	Відкрита
Масштабованість	Висока	Середня	Висока	Висока
Інтеграція з Python	Так	Так	Обмежена	Так
Вартість	Висока	Безкоштовна	Висока	Безкоштовна
Складність у впровадженні	Висока	Низька	Середня	Середня

Обґрунтування вибору СУБД

Для реалізації системи обліку випускників було обрано **PostgreSQL**, оскільки вона забезпечує оптимальний баланс між функціональністю, надійністю та гнучкістю згідно з табл.1 . Ця СУБД підтримує складні типи даних, забезпечує високу продуктивність при роботі з великими обсягами даних і дозволяє реалізувати розширену логіку без додаткових платних рішень.

Окремо варто зазначити, що PostgreSQL добре інтегрується з фреймворком Django, на якому реалізовано прикладну частину системи. Це спрощує створення моделей, міграцій та забезпечує безпосередню підтримку роботи з базою на рівні ORM. Такий вибір також забезпечує перспективи подальшого масштабування системи, підтримки геоданих або додаткових аналітичних функцій — у разі розширення функціоналу у майбутньому.

2.3 Створення БД

На основі розробленої логічної моделі даних було реалізовано фізичну структуру бази даних системи обліку випускників університету. У якості СУБД обрано **PostgreSQL**, яка забезпечує високу надійність, підтримку транзакцій, зовнішніх ключів, каскадного видалення, індексування та гнучку типізацію.

Реалізована база даних містить окремі таблиці для кожної сутності предметної області: факультетів, кафедр, спеціальностей, випускників та їхнього працевлаштування. Зв’язки між сутностями реалізовано через зовнішні ключі із відповідними обмеженнями ON DELETE CASCADE або ON DELETE SET NULL залежно від типу взаємозалежності.

Ключові особливості структури.

Кожна таблиця має унікальний первинний ключ типу VARCHAR, що дозволяє зручно працювати з UUID-ідентифікаторами та зовнішніми системами.

Таблиця faculty містить дані про факультети, включаючи назву, скорочення та опис.

Department пов’язується з faculty зовнішнім ключем faculty_id, що дозволяє реалізувати зв’язок "один факультет — багато кафедр".

Speciality має зовнішній ключ на кафедру, забезпечуючи ієрархічну прив'язку без дублювання.

Graduate зберігає детальну інформацію про випускника, включаючи ПІБ, рік випуску, email, телефон, дату народження, а також зв’язок із конкретною спеціальністю.

Employment винесено як окрему таблицю, пов’язану з випускником через graduate_id, що дозволяє враховувати декілька записів про працевлаштування на одного випускника.

Усі зовнішні ключі супроводжуються обмеженнями цілісності (FOREIGN KEY ... REFERENCES ... ON DELETE ...), що забезпечує коректну логіку збереження та видалення пов’язаних записів.

Характеристики реалізації

Таблиця **graduate** є центральною у структурі даних. Вона пов’язана зі спеціальністю через поле speciality_id, що дозволяє легко формувати статистику за освітніми напрямами.

Таблиця **speciality** у свою чергу пов’язана з **department**, яка напряму зв’язана з **faculty** — таким чином формується повна освітня структура навчального закладу.

Таблиця **employment** дозволяє зберігати детальну інформацію про місце роботи, посаду та рік початку праці.

Реалізація структури бази даних у Django

Для взаємодії з базою даних у рамках проєкту було використано **фреймворк Django**, який надає вбудований механізм ORM (Object-Relational Mapping). Це дозволило уникнути написання SQL-коду вручну при створенні та підтримці структури таблиць. Усі сутності були реалізовані як Python-класи, що зручно інтегруються з іншими компонентами системи.

Переваги використання Django ORM:

- автоматична генерація SQL-команд за допомогою міграцій (makemigrations, migrate);
- просте створення запитів безпосередньо у Python (через filter(), get() тощо);
- явне визначення зв’язків через типи полів: ForeignKey, OneToOneField, ManyToManyField;
- збереження логіки предметної області у вигляді класів і методів, що легко тестуються та розширюються.

Структура основних таблиць у вигляді SQL-коду представлена в додатку Д.

Візуалізована фізична модель бази даних наведена у додатку Е.

Додаткові можливості Django ORM:

- автоматичне створення таблиць після виконання команд makemigrations і migrate;
- можливість додавати обмеження, індекси, унікальні поля, null/blank значення, дефолти тощо;
- розширюваність моделей через успадкування (AbstractBaseUser, models.Model);
- інтеграція з адміністративною панеллю Django Admin для управління базою без написання додаткового інтерфейсу.

У Django взаємодія з базою даних реалізується через ORM-запити, приклад такого запиту продемострovано на рис. 3.

```
# Отримати всіх випускників певної спеціальності
graduates = Vypusknyk.objects.filter(specialnist__name="Комп'ютерні науки")

# Додати нове працевлаштування
Pratsevlashtuvannya.objects.create(
    vypusknyk=graduate,
    company_name="SoftServe",
    position="Python Developer",
    start_year=2023
)
```

Рис. 3 Приклад ORM-запитів

Таким чином, реалізація бази даних через Django ORM дозволяє розробнику уникнути рутинного SQL-кодування, автоматизувати всі етапи життєвого циклу структури БД та працювати на вищому рівні абстракції, що суттєво підвищує продуктивність і знижує ризик помилок у логіці збереження даних.

2.4 Додавання користувачів до БД

Для забезпечення безпеки, контролю доступу та розмежування прав у системі обліку випускників було реалізовано створення окремих користувачів бази даних із визначеними правами доступу. Такий підхід дозволяє відмовитися від використання єдиної таблиці авторизованих користувачів у самому застосунку й делегує керування доступом безпосередньо на рівень СУБД PostgreSQL.

Основні ролі системи:

- **methodist_user** — користувач з повним доступом до операцій: створення, редагування, видалення, імпорт, експорт даних;
- **kerivnyk_user** — користувач з доступом лише до читання: перегляд інформації про випускників, формування звітів, перегляд статистики.

SQL запити для створення ролей у PostgreSQL продемонтровано на рис. 4.

```
-- Створення ролі методиста
CREATE ROLE methodist_user WITH LOGIN PASSWORD 'methodist_pass';
GRANT CONNECT ON DATABASE alumnus_db TO methodist_user;
GRANT USAGE ON SCHEMA public TO methodist_user;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO
methodist_user;

-- Створення ролі керівника
CREATE ROLE kerivnyk_user WITH LOGIN PASSWORD 'kerivnyk_pass';
GRANT CONNECT ON DATABASE alumnus_db TO kerivnyk_user;
GRANT USAGE ON SCHEMA public TO kerivnyk_user;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO kerivnyk_user;
```

Рис. 4 SQL-запити

Інтеграція з Django

У рамках Django проєкту доступ до бази даних здійснюється через файл settings.py. Для кожного користувача можна створити окремий екземпляр програми з відповідним .env-файлом, у якому будуть задані логін і пароль до відповідної ролі PostgreSQL(рис. 5).

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'alumnus_db',
        'USER': os.environ.get('DB_USER'),
        'PASSWORD': os.environ.get('DB_PASSWORD'),
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Рис. 5 Приклад коду реалізації

3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Моделювання структури та взаємодії програмної системи

На етапі проєктування інформаційної системи обліку випускників університету було побудовано ряд UML-діаграм, які дозволяють формалізувати логіку функціонування системи, її структуру, внутрішні зв’язки між модулями та способи взаємодії з користувачами. Такі діаграми є важливою складовою процесу створення програмного забезпечення, оскільки вони не тільки допомагають краще зрозуміти структуру майбутньої системи, але й слугують документаційною основою для розробників, аналітиків та тестувальників.

Моделювання програмної системи дає змогу:

- виявити архітектурні рішення на ранніх етапах розробки;
- візуалізувати компоненти та взаємозв’язки між ними;
- уникнути логічних помилок ще до написання коду;
- покращити супровід і масштабування системи у майбутньому;
- забезпечити прозорість реалізації для всіх учасників проєкту.

У межах цієї дипломної роботи було побудовано кілька ключових типів UML-діаграм, кожна з яких відіграє свою роль у процесі проєктування та розуміння функціонування системи.

3.1.1 Діаграма класів є одним із ключових інструментів об’єктно-орієнтованого проєктування, що використовується для моделювання статичної структури програмної системи. Вона дозволяє візуалізувати основні сутності (класи) предметної області, їхні атрибути, методи, а також зв’язки між об'єктами, які визначають логіку взаємодії даних у системі.

Діаграма класів включає:

- класи з атрибутами та методами;
- типи зв’язків між класами (асоціації, агрегації, композиції);

• ролі, кратності (наприклад, 1:1, 1:N), що описують кількісні обмеження.

У розробленій інформаційній системі обліку випускників університету виділено основні класи.

Факультет — представляє вищу одиницю організаційної структури. Містить назву факультету й відповідну метод-функцію для зміни назви. Один факультет об’єднує декілька кафедр (зв’язок 1 до багатьох).

Кафедра — підрозділ факультету. Містить атрибут назви кафедри, а також метод змінитиНазву(). Кафедра керує кількома спеціальностями, і сама входить до складу одного факультету.

Спеціальність — пов’язана з конкретною кафедрою. Має лише назву та метод для її зміни. Кожна спеціальність належить одній кафедрі, але на ній можуть навчатися багато випускників.

Випускник — центральний клас, що містить інформацію про ПІБ, рік випуску, факультет, спеціальність та контактні дані. Містить методи отримання, оновлення та видалення інформації. Один випускник може бути пов’язаний із кількома записами працевлаштування, а також може бути включений до кількох звітів.

Працевлаштування — зберігає інформацію про компанію, посаду, рік початку роботи та пов'язується з одним випускником. Один випускник може мати багато записів про працевлаштування (зв’язок 1 до багатьох).

Звіт — клас, що відповідає за збереження згенерованих статистичних даних. Містить назву звіту, дату створення та метод зберегтиЗвіт(). Звіт може містити інформацію про декількох випускників.

Зв’язки між класами

Факультет 1 — 1..* Кафедра: факультет об’єднує одну або більше кафедр.

Кафедра 1 — 1..* Спеціальність: кожна кафедра керує однією або кількома спеціальностями.

Спеціальність 1 — 0..* Випускник: випускники навчаються за спеціальністю.

Випускник 1 — 0..* Працевлаштування: випускник може бути працевлаштованим в одній або кількох компаніях.

Випускник 1 — 0..* Звіт: випускник може бути включений до одного або декількох звітів, а звіт містить багато випускників (зв’язок багато до багатьох реалізовано через агреговану асоціацію).

Ці зв’язки реалізуються в базі даних через зовнішні ключі та підтримуються ORM-системою Django, що автоматично формує відповідні зв’язки в реляційній базі PostgreSQL. Кратності вказують на те, скільки екземплярів одного класу можуть бути пов’язані з іншим. Діаграма класів дозволяє відобразити предметну область системи у вигляді об’єктно-орієнтованої моделі, на основі якої реалізується структура бази даних і логіка програмного коду. Вона забезпечує цілісність і узгодженість даних, дозволяє зрозуміти архітектурні зв’язки між елементами та лягла в основу подальшої реалізації функціоналу системи через моделі Django.

Діаграма класів представлена в Додатку Ж.

3.1.2 Діаграма кооперацій (комунікацій) — це один із типів поведінкових UML-діаграм, який моделює взаємодію між об’єктами (екземплярами класів) у межах певного сценарію виконання. На відміну від діаграми послідовності, де акцент зроблений на часовій шкалі викликів, діаграма кооперацій фокусується на структурі зв’язків між об'єктами та порядку повідомлень між ними.

У процесі розробки програмного забезпечення було реалізовано функціонал додавання інформації про працевлаштування випускника. Для відображення взаємодії об’єктів у межах цього процесу було побудовано діаграму кооперацій, що демонструє зв’язки між сутностями Випускник та Працевлаштування. Відповідна UML-діаграма наведена на рис. 6.

На діаграмі зображено, як об’єкт класу Випускник асоціюється з одним або кількома об’єктами класу Працевлаштування. Відповідно до логіки системи, кожен випускник може мати кілька записів про місце роботи, які містять назву компанії, посаду та рік початку роботи. Після додавання нового запису, інформація про працевлаштування автоматично зберігається й стає доступною для подальшого перегляду.

Метод додатиПрацевлаштування() ініціюється для створення нового об’єкта працевлаштування, який потім логічно прив’язується до відповідного випускника. Крім того, реалізовано метод переглядатиІнформацію(), що дозволяє отримати всі записи працевлаштування, пов’язані з конкретним випускником.

Ця діаграма дозволяє простежити послідовність викликів методів і взаємодію між об’єктами, що беруть участь у сценарії. Вона підтверджує правильність реалізації функціональності в системі та відображає поведінкову логіку у вигляді структурованої кооперації між класами.

Рис. 6 Діаграма кооперації «Додавання працевлаштування»

3.1.3 Діаграма пакетів дозволяє структуровано відобразити компоненти системи, згрупувавши їх у логічні блоки — пакети. Пакети можуть містити класи, інтерфейси, підсистеми або інші пакети. Ця діаграма:

- допомагає організувати великий код на рівні модулів;
- дозволяє бачити залежності між підсистемами;
- показує розділення на фронтенд, бекенд, логіку тощо.

На рис. 7 зображена діаграма пакетів на якій продемонстрована модульна структура програмного забезпечення, яка логічно організовує елементи системи обліку випускників відповідно до архітектурних принципів Django. Центральне місце займає пакет models, де реалізовано основні класи, що відповідають за структуру та збереження даних, зокрема Faculty, Department, Speciality, Graduate і Employment. Цей пакет є базовим і використовується іншими компонентами системи, оскільки всі сутності обертаються навколо збереження інформації про випускників та пов’язані з ним об’єкти.

Пакет services реалізує бізнес-логіку застосунку, зокрема функціональність імпорту списку випускників і генерації статистичних звітів. Він безпосередньо взаємодіє з моделями, виконуючи всі операції зі зчитуванням, обробкою та створенням даних. У свою чергу, views — це контролери, які обробляють запити від користувачів, звертаються до сервісів для виконання логіки, а також передають результати у шаблони для відображення. Пакет urls пов’язує конкретні маршрути веб-запитів із відповідними представленнями, забезпечуючи взаємодію між інтерфейсом користувача та логікою системи.

У проєкті також присутній пакет admin, відповідальний за адміністрування даних через вбудовану Django-панель. Він налаштовує вигляд і поведінку моделей у середовищі адміністратора і, як і очікується, напряму залежить лише від models. Пакет tests реалізує модульні та інтеграційні тести, що дозволяє перевіряти правильність функціонування моделей і бізнес-логіки. Він містить залежності як від models, так і від services, адже перевіряє взаємодію між цими

складовими. Також на діаграмі представлений зв’язок між views і forms, який хоча й не деталізований у схемі, є типовим для Django-застосунків і відповідає за обробку користувацького введення через HTML-форми.

У результаті структура системи побудована на основі чіткого розділення відповідальності між модулями, де кожен пакет взаємодіє лише з тими компонентами, що необхідні для його функціонування. Це дозволяє підтримувати систему легкою для масштабування, тестування та супроводу.

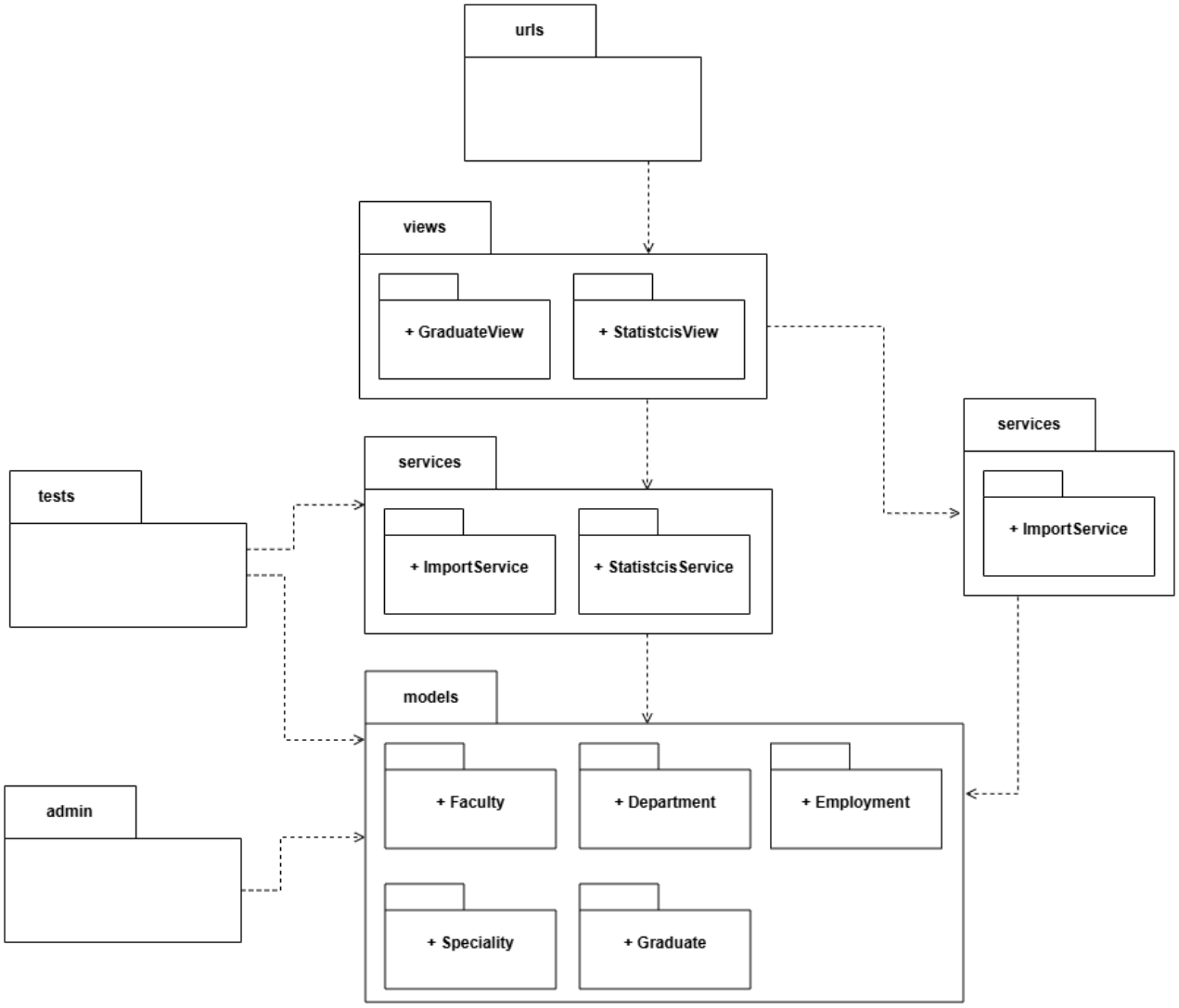


Рис. 7 Діаграма пакетів

3.1.4 Діаграма компонентів описує фізичну структуру системи, в якій кожен компонент представлений як окрема одиниця реалізації: файл, модуль, бібліотека чи зовнішній ресурс. Вона дозволяє візуалізувати, як розподілено

функціональність системи між її частинами, яким чином компоненти пов’язані між собою, які з них є залежними, та які ресурси використовуються.

У контексті розробленої інформаційної системи обліку випускників діаграма компонентів демонструє, з яких саме фізичних елементів складається застосунок. Усі модулі реалізовані на основі фреймворку Django, що зумовило поділ системи на окремі компоненти, які відповідають за різні аспекти логіки, зберігання, інтерфейсу та звітності.

На діаграмі, що представлена на рис. 8, відображено ключові компоненти.

WebApp.py — центральний виконуваний компонент, з якого починається запуск усього серверного застосунку. Саме він ініціалізує середовище Django, обробку запитів, підключення до маршрутизаторів, шаблонів, моделей і зовнішніх сервісів. У рамках системи WebApp.py виступає точкою входу до всієї логіки.

Models.py — файл, який містить класи моделей, що відповідають сутностям предметної області: випускники, факультети, кафедри, спеціальності, працевлаштування. Ці класи реалізовані з використанням Django ORM і формують структуру бази даних. Через models.py здійснюється вся взаємодія з базою PostgreSQL: створення, оновлення, вибірка й видалення записів.

Views.py — компонент, який містить контролери (представлення) системи. Він відповідає за логіку обробки вхідних HTTP-запитів, визначає, які шаблони будуть використані для формування відповідей, а також викликає необхідні функції з бізнес-логіки. У ньому реалізовано маршрути на сторінки додавання, редагування, пошуку випускників, генерації звітів тощо.

Import_service.py та **statistics_service.py** — окремі модулі, що реалізують ключову бізнес-логіку системи. Перший відповідає за обробку імпортованих даних (наприклад, з Excel), створення об’єктів випускників та їх збереження у базі. Другий реалізує функції обчислення статистики працевлаштування,

підготовку даних до візуалізації та формування статистичних звітів. Ці модулі використовуються у views.py як внутрішні сервіси.

Templates_html — набір HTML-шаблонів, які відповідають за побудову вебінтерфейсу системи. З їхньою допомогою формуються сторінки перегляду, додавання, редагування записів, відображення графіків і звітів. Вони базуються на шаблонізаторі Django і динамічно формуються залежно від запитів користувача.

README.md — текстовий файл, який містить документацію до системи. У ньому надається інформація щодо структури проєкту, інструкції з розгортання, встановлення залежностей, запуску застосунку та тестування. Цей компонент відіграє важливу роль для розробників та супровідного персоналу.

Report.docx — кінцевий результат роботи модуля статистики. Це звіт у форматі Microsoft Word, який генерується динамічно на основі даних про випускників та їхнє працевлаштування. Він може містити зведені таблиці, діаграми та описові дані.

PostgreSQL_DB — зовнішній компонент, який представляє реальну базу даних PostgreSQL. У ній фізично зберігаються всі записи, створені в системі: від особистих даних випускників до статистичних зведень. Зв’язок із цією базою забезпечується за допомогою ORM Django, а models.py відповідає за структуру таблиць.

Django — стороння бібліотека (фреймворк), що використовується як основа для реалізації всієї серверної логіки. Django надає механізми для маршрутизації, обробки форм, шаблонів, роботи з базою даних, автентифікації користувачів і багато іншого. Це ключова технологія проєкту.

Bootstrap — зовнішня CSS-бібліотека, яка використовується у шаблонах HTML для забезпечення адаптивності інтерфейсу, формування сітки сторінки, стилізації таблиць, кнопок, модальних вікон та інших UI-компонентів. Завдяки

використанню Bootstrap інтерфейс є зручним та інтуїтивно зрозумілим для користувача.

Зв’язки між компонентами відображають залежності: наприклад, WebApp.py залежить від views.py і models.py, тоді як views.py — від сервісів import_service.py і statistics_service.py, а також від шаблонів templates_html. Компонент models.py взаємодіє з базою даних PostgreSQL_DB, забезпечуючи збереження й отримання даних через ORM Django.

Діаграма компонентів чітко демонструє фізичну архітектуру проєкту, розділення обов’язків між окремими частинами коду та взаємозв’язки між ними. Це дає змогу краще розуміти структуру проєкту, спростити його супровід, масштабування та інтеграцію нових функцій.

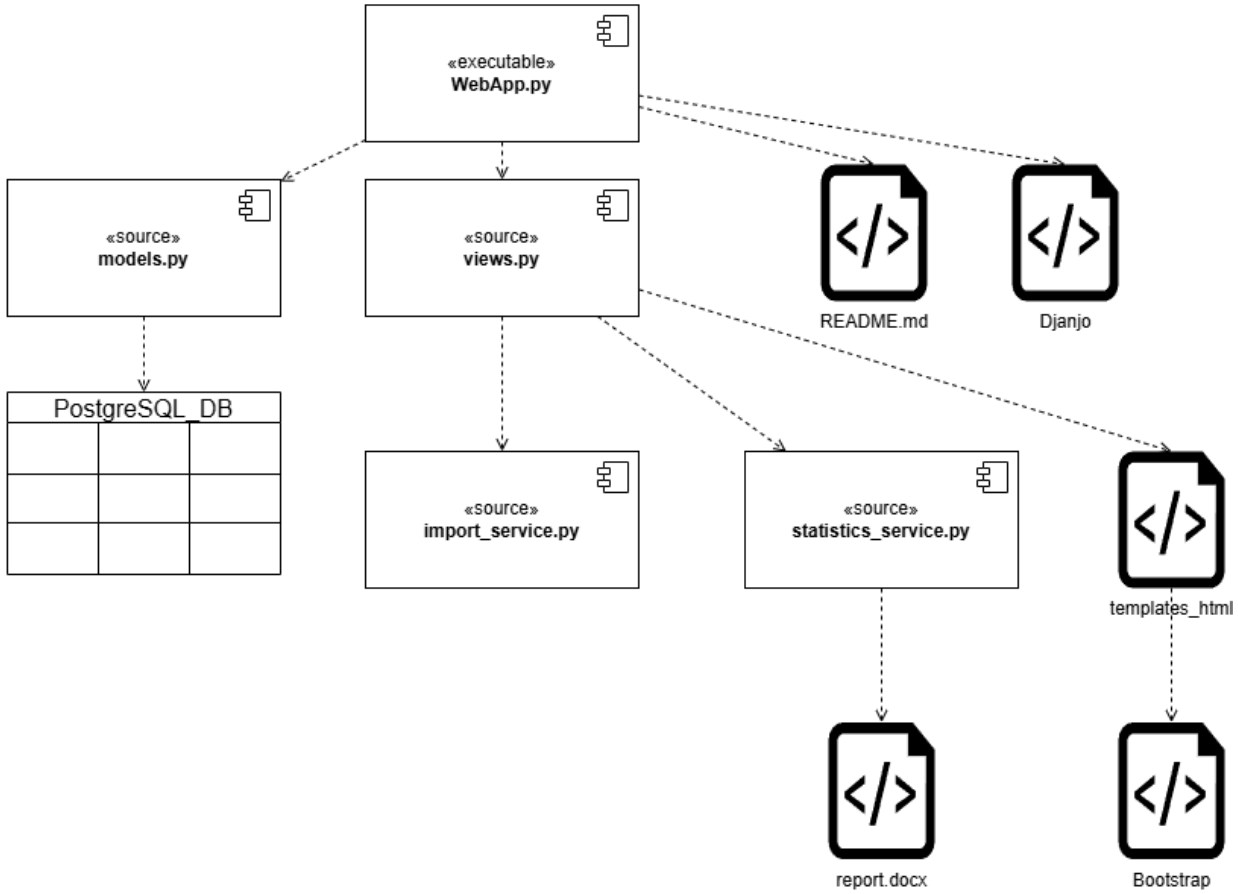


Рис. 8 Діаграма компонентів

3.2 Вибір інструментів для створення ППЗ

Розробка інформаційної системи обліку випускників університету вимагала ретельного підбору інструментів, які б забезпечили ефективність, масштабованість та зручність у використанні. Основними критеріями вибору були: відкритий вихідний код, активна спільнота розробників, підтримка сучасних стандартів безпеки та можливість швидкої розробки.

Фреймворк Django

Для серверної частини було обрано фреймворк Django, який є високорівневим веб-фреймворком на мові програмування Python. Django забезпечує швидку розробку та чистий, прагматичний дизайн. Він включає в себе потужну ORM (Object-Relational Mapping), систему шаблонів, маршрутизацію URL, вбудовану адміністративну панель та механізми безпеки. Django дотримується принципу DRY (Don't Repeat Yourself), що сприяє повторному використанню коду та зменшенню помилок. Його архітектура MTV (Model-Template-View) дозволяє чітко розділити логіку додатку, що спрощує розробку та супровід системи[7].

Система управління базами даних PostgreSQL

Для зберігання даних було обрано PostgreSQL — потужну, об'єктно-реляційну систему управління базами даних з відкритим кодом. PostgreSQL підтримує транзакції з властивостями ACID, складні типи даних, зовнішні ключі, індексацію та розширення. Вона легко інтегрується з Django через драйвер psycopg2, що дозволяє ефективно працювати з базою даних та забезпечує високу продуктивність системи[8].

Середовище розробки Visual Studio Code

Розробка програмного забезпечення здійснювалася в середовищі Visual Studio Code (VS Code) — легкому, але потужному редакторі коду з відкритим кодом. VS Code забезпечує підтримку Python через офіційне розширення, яке надає можливості автодоповнення коду (IntelliSense), відлагодження, лінтингу та

тестування. Крім того, VS Code має вбудований термінал, інтеграцію з системою контролю версій Git та підтримку віртуальних середовищ, що робить його зручним інструментом для розробки на Python[9].

Інтерфейс користувача та фронтенд

Інтерфейс користувача було реалізовано за допомогою HTML, CSS та бібліотеки Bootstrap 5, яка є популярним фреймворком для створення адаптивних та мобільних веб-сайтів. Bootstrap 5 надає набір готових компонентів, таких як кнопки, форми, навігаційні панелі та модальні вікна, що дозволяє швидко створювати привабливий та функціональний інтерфейс. Крім того, Bootstrap 5 підтримує сучасні браузери та забезпечує кросбраузерну сумісність[10].

Інші інструменти та технології

- **python 3.10+**: основна мова програмування для розробки серверної частини;
- **pip**: менеджер пакетів для встановлення та управління залежностями;
- **venv**: модуль для створення ізольованих віртуальних середовищ Python;
- **Postman**: інструмент для тестування API та відлагодження запитів;
- **git**: система контролю версій для управління змінами в коді та спільної роботи над проектом;
- **github**: хостинг для зберігання репозиторію проекту та співпраці з іншими розробниками.

Використання сучасного стеку технологій, таких як Django, PostgreSQL, VS Code та Bootstrap 5, дозволило створити стабільну, масштабовану та зручну у використанні інформаційну систему обліку випускників. Обрані інструменти забезпечили швидку розробку, високу продуктивність та легкість у супроводі системи.

3.3 Алгоритмізація та програмування програмних модулів

Реалізація функціональності інформаційної системи обліку випускників була побудована на основі чіткої логіки взаємодії між модулями, з урахуванням принципів багаторівневої архітектури. Усі програмні компоненти розділено за рівнями: інтерфейс користувача, обробка бізнес-логіки, доступ до даних та відображення результатів.

На цьому етапі було визначено основні функціональні модулі системи, описано їхню внутрішню логіку та реалізовано алгоритми обробки даних у вигляді функцій, класів і сервісів. Кожен з модулів було спроєктовано відповідно до попередньо змодельованих структур (діаграм класів, кооперацій та компонентів), що дозволило забезпечити узгодженість між логікою роботи програми та структурою збереження даних.

Для опису реалізації основних функцій використано комбінацію Python-коду, шаблонів Django та взаємодії з ORM, а також супровідні блок-схеми, що відображають логіку виконання ключових дій у системі.

Додавання нового випускника

Функція додавання випускника — одна з основних в системі, оскільки саме з неї починається формування повноцінної бази даних. Реалізація цієї функції дозволяє методисту за допомогою вебінтерфейсу створити новий запис про особу, яка закінчила університет. Під час створення вказуються обов’язкові дані: ім’я, прізвище, по батькові, дата народження, рік випуску, електронна пошта, телефон, спеціальність, а також (за потреби) додаткова інформація.

З технічного боку, функціонал реалізовано як **представлення (View)**, що обробляє HTTP-запит типу POST. Форма, яку заповнює користувач, реалізована через GraduateForm, а всі дані перед збереженням проходять перевірку на коректність (валидацію). У разі успішного заповнення дані записуються до відповідної таблиці в базі через механізм Django ORM, що дозволяє уникнути ручного формування SQL-запитів.

Ключова логіка реалізації:

- користувач переходить на сторінку додавання випускника;
- вводить дані у форму, після чого надсилає запит. Код обробки запиту продемонстровано на рис. 9;
- форма перевіряє правильність введених значень;
- якщо дані валідні — створюється новий об'єкт graduate, який зберігається у БД;
- користувача перенаправляє на сторінку зі списком випускників або підтвердженням.

```
def add_graduate(request):
    if request.method == 'POST':
        form = GraduateForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('graduate_list')  # Перенаправлення до списку
    else:
        form = GraduateForm()
    return render(request, 'graduates/add.html', {'form': form})
```

Рис. 9 Приклад коду обробки запиту

Уся перевірка коректності даних (наприклад, правильність email, формат дати) виконується автоматично засобами Django Forms[11]. У разі помилки — користувачу повертається сторінка з формою та відповідним повідомленням.

Перевагою цього підходу є висока швидкість розробки, гнучкість у зміні форми й логіки, а також безпечна взаємодія з базою без ризику SQL-ін’єкцій[12].

Логічну послідовність виконання цієї функції подано на блок-схемі "Додавання випускника" на рис. 10.

Рис. 10 Блок-схема функції "Додавання випускника"

Імпорт списку випускників із Excel-файлу

Функціональність імпорту випускників реалізована для зручності масового введення даних без ручного заповнення кожного запису через інтерфейс. Така можливість особливо актуальна для методистів факультетів, які оперують великим обсягом інформації, збереженої у вигляді таблиць Excel. Система дозволяє швидко імпортувати всю таблицю з даними про випускників і створити відповідні об’єкти в базі даних.

Для реалізації було використано бібліотеку **openpyxl**, яка дозволяє зчитувати файли .xlsx. Імпорт реалізовано як окрема сервісна функція, яка приймає файл, зчитує дані з кожного рядка та створює об’єкт Graduate для кожного запису.

Функція читає всі рядки таблиці починаючи з другого (перший вважається заголовком). Для кожного рядка створюється новий об’єкт Graduate з автоматичним збереженням у базі через ORM. Код функцій представлений на рис. 11.

```
import openpyxl
from .models import Graduate


def import_graduates_from_excel(file):
    workbook = openpyxl.load_workbook(file)
    sheet = workbook.active
    for row in sheet.iter_rows(min_row=2, values_only=True):
        Graduate.objects.create(
            first_name=row[0],
            last_name=row[1],
            middle_name=row[2],
            birth_date=row[3],
            email=row[4],
            phone=row[5],
            graduation_year=row[6],
            speciality_id=row[7],
            additional_info=row[8],
        )
```

Рис. 11 Приклад коду функції імпорту

У процесі імпорту система очікує, що Excel-файл матиме чітко визначену структуру стовпців, кожен з яких відповідає одному з полів об'єкта Graduate. Така структура дозволяє безпомилково зчитати й інтерпретувати дані з кожного рядка таблиці. Очікується, що перший рядок файлу містить заголовки, а кожен наступний — відповідні дані одного випускника. Послідовність полів має бути наступною:

- **Ім’я** — текстове поле;
- **Прізвище** — текстове поле;

- **По батькові** — необов’язкове поле;
- **Дата народження** у форматі рік-місяць-день
- **Email** — контактна електронна адреса;
- **Телефон** — мобільний або стаціонарний номер;
- **Рік випуску** – чотиризначне число;
- **Спеціальність** — назва спеціальності;
- **Додаткова інформація** – вільне текстове поле для зауважень.

Дотримання цієї структури є критичним для успішного імпорту: будь-яке порушення порядку стовпців або типу даних може призвести до помилок зчитування або зупинки процесу.

Генерація статистичного звіту

Функція генерації статистики є ключовим аналітичним інструментом системи обліку випускників. Вона дозволяє автоматично формувати узагальнені звіти щодо кількості випускників за спеціальностями, роками, факультетами, а також обробляти інформацію про їхнє працевлаштування. Отримані дані можуть бути використані для внутрішнього аналізу, подання адміністрації або підготовки звітів для зовнішніх організацій (наприклад, МОН).

З технічної точки зору, модуль статистики реалізовано як окремий сервіс, який виконує агрегацію даних з таблиці Graduate за заданими критеріями. Використовуючи механізми Django ORM, функція виконує підрахунок кількості об’єктів за певними параметрами (наприклад, спеціальністю) і повертає словник або структуру, з якою зручно працювати для побудови графіків або формування звітних документів.

Функція generate_statistics() виконує вибірку всіх назв спеціальностей, за якими закінчили навчання випускники, і рахує кількість кожної за допомогою Counter із стандартної бібліотеки Python. Результат можна відобразити у вигляді

таблиці або кругової діаграми на сторінці статистики. Код функції generate_statistics() наведено на додатку К.

Формування інтерфейсу користувача за допомогою HTML-шаблонів

Важливою частиною розробки інформаційної системи є створення зручного та функціонального веб-інтерфейсу, через який користувач взаємодіє з основним функціоналом. Для цього в межах проєкту було використано шаблонізатор Django та бібліотеку Bootstrap 5.

Шаблонізатор Django дозволяє створювати динамічні HTML-сторінки, які формуються на основі переданих даних із контролерів (views). Шаблони можуть містити змінні, цикли, умовні конструкції, а також включення інших шаблонів (наприклад, спільного header або footer).

Код шаблонізатора продемонстрвано на рис. 12.

Інтерфейс системи було реалізовано за такими принципами:

- всі шаблони розміщені в директорії templates/;
- для виводу списків (наприклад, випускників, звітів) використовуються цикли {% for %};
- для виводу повідомлень про помилки чи успішні дії — {% if %};
- форми створені з використанням Django Forms і автоматично рендеряться в шаблонах;
- bootstrap 5 забезпечує адаптивність і стилізацію елементів.

У прикладі шаблон успадковується від base.html, де розміщено загальні стилі, меню, футер. Фрагмент коду {{ form.as_p }} рендерить усі поля форми як абзаци з відповідними <input> елементами. Тоекн csrf_token захищає форму від підробки запитів. Клас btn btn-primary з Bootstrap стилізує кнопку.

```
{% extends "base.html" %}
{% block content %}
<div class="container mt-4">
  <h2>Додавання нового випускника</h2>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit" class="btn btn-primary">Зберегти</button>
  </form>
</div>
{% endblock %}
```

Рис. 12 Приклад коду шаблонізатора

HTML-шаблони відіграють важливу роль у системі, оскільки забезпечують логічну структуру, зрозумілий інтерфейс і підвищують зручність користування. Завдяки шаблонізації також спрощується підтримка і розширення інтерфейсу, оскільки зміни в шаблоні відображаються відразу для всіх сторінок, що його використовують.

У процесі алгоритмізації було реалізовано функціональні модулі, які потрібні в системі обліку випускників. На прикладах було продемонстровано логіку додавання випускника, імпорту даних з Excel, побудови статистичних звітів та формування інтерфейсу користувача.

Для кожної функції було визначено структуру вхідних даних, порядок обробки, використані бібліотеки та принципи інтеграції з базою даних за допомогою Django ORM. Код функцій супроводжено блок-схемами, що ілюструють послідовність виконання операцій.

Особливу увагу приділено зручності для кінцевого користувача: шаблони HTML розроблені з використанням Bootstrap 5, а весь інтерфейс має зрозумілу структуру й адаптивне відображення.

Результатом алгоритмізації стало чітке визначення логіки роботи системи, її реалізація у вигляді структурованих модулів та забезпечення функціональності, що відповідає вимогам замовника.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ

4.1 Тестування системи

Тестування є невід'ємною частиною ЖЦ розробки програмного забезпечення, яке виконується на різних етапах — від верифікації окремих функціональних компонентів до перевірки повної інтеграції та оцінки поведінки системи в умовах, наближених до реального використання. Його основна мета полягає у виявленні помилок і недоліків у роботі системи до моменту її впровадження, а також у підтвердженні того, що реалізований функціонал відповідає вимогам, сформульованим на етапах проєктування та технічного завдання[25].

Тестування дозволяє виявити критичні помилки ще до передачі системи кінцевому користувачу, що суттєво знижує ризики збою в процесі експлуатування, утрати даних або порушення бізнес-логіки. Окрім цього, тестування слугує механізмом валідування: воно підтверджує, що система дійсно реалізовує необхідну поведінку згідно з очікуваннями користувачів.

У розробці інформаційної системи обліку випускників тестування відіграє особливо важливу роль, оскільки вона обробляє персональні та організаційні дані, до яких висуваються високі вимоги щодо достовірності, структурованості та збереження. Крім того, система передбачає різнорівневий доступ користувачів із відмінними повноваженнями (методисти, керівники), що потребує ретельної перевірки механізмів контролю доступу та розмежування прав[26].

Тестування системи дозволяє:

- забезпечити коректну роботу окремих функцій (додавання, редагування, пошук, експорт/імпорт даних) та взаємозв’язків між модулями;
- перевірити стабільність системи в умовах роботи з великими обсягами даних, повторного використання функцій або помилкових дій користувача;

- мінімізувати ймовірність появи критичних помилок у процесі реального використання, що можуть вплинути на цілісність або втрату даних;

- підвищити надійність, безпеку та зручність експлуатації системи для кінцевих користувачів, зокрема методистів кафедр та керівників.

Таким чином, тестування є не лише технічною перевіркою коду, а й складовою забезпечення якості проєкту, що дозволяє впевнено перейти до етапу впровадження системи в експлуатаційне середовище.

Основні методи тестування

У програмній інженерії виділяють такі основні методи тестування:

- **модульне тестування (unit testing)** — перевірка окремих функцій, класів чи методів на коректність виконання;

- **інтеграційне тестування** — перевірка взаємодії між різними модулями або компонентами системи;

- **системне тестування** — перевірка роботи всієї системи в цілому, включаючи її інтерфейс, логіку, обробку помилок;

- **тестування з боку користувача (acceptance testing)** — перевірка того, чи відповідає система очікуванням і задачам кінцевого користувача;

- **регресійне тестування** — перевірка того, чи не зламалась уже працездатна функціональність після внесення змін у код.

Тестування розробленої системи

У рамках цього проєкту було проведено переважно **модульне** та **системне тестування**. Основний акцент було зроблено на перевірці базового функціоналу.

Тестування форм: перевірено, що форми додавання, редагування та імпорту випускників приймають коректні значення, а в разі помилок — виводять повідомлення з відповідним описом.

Перевірка фільтрації та пошуку: реалізовано перевірку правильності результатів при фільтрації за роком випуску, факультетом, спеціальністю, а також при пошуку за ПІБ.

Тестування ролей: перевірено розмежування прав доступу — методист має доступ до редагування та видалення, а керівник лише до перегляду.

Експорт та імпорт: перевірено стабільність обробки файлів (Excel) при імпорті великого обсягу даних, а також правильність структури сформованих експортованих файлів.

Перевірка захищеності доступу: протестовано сценарії, коли користувач без прав доступу намагається відкрити або змінити дані[27].

У процесі тестування використовувались інструменти Django (TestCase, Client) для автоматизації модульних перевірок, а також ручне тестування в браузері (Chrome, Opera) для оцінки інтерфейсу та взаємодії з базою даних.

Результати тестування підтвердили, що реалізоване програмне забезпечення стабільно працює у межах визначених сценаріїв. Помилки, виявлені на ранніх етапах, були усунені ще до фінального розгортання.

4.2 Вимоги до апаратного та програмного забезпечення

Для ефективної роботи інформаційної системи обліку випускників університету необхідно забезпечити відповідність апаратного й програмного забезпечення вимогам, які визначаються архітектурою розробленого рішення, структурою даних та режимами доступу. У системі виділяються дві основні складові: клієнтська частина (інтерфейс користувача, що працює у браузері) та серверна частина (вебзастосунок, база даних, API), кожна з яких має власні технічні потреби й виконує окремі функціональні ролі у процесі взаємодії з користувачем і обробки інформації.

Вимоги до апаратного забезпечення

Клієнтська частина (пристрої методистів і керівників):

- процесор: не нижче ніж Dual-core, 1.6 GHz;
- оперативна пам'ять: від 2 GB RAM;
- вільне місце на диску: щонайменше 100 MB (для кешу браузера);
- мережа: стабільне інтернет-з’єднання зі швидкістю не менше 1 Mbps;
- роздільна здатність екрану: від 1024×768 пікселів.

Серверна частина (середовище виконання):

- процесор: щонайменше Quad-core, 2.0 GHz;
- оперативна пам’ять: 4–8 GB RAM (залежно від кількості одночасних користувачів);
- дисковий простір: мінімум 2 GB (включаючи систему, базу даних, логи, бібліотеки);
- мережеве підключення: 5 Mbps або вище для стабільної взаємодії з клієнтами.

Вимоги до програмного забезпечення

Клієнтська частина (вебінтерфейс у браузері):

- підтримувані операційні системи: Windows 10/11, macOS, Linux, Android, iOS;
- сумісні браузери:
  - Google Chrome (версія 89+);
  - Mozilla Firefox (87+);
  - Microsoft Edge (89+);
  - Safari (14+);

- Chrome для Android, Safari для iOS.

Програмне середовище розробки:

- операційна система: Windows, Linux, або macOS;
- мова програмування: Python 3.10+;
- фреймворк: Django (або альтернативно Flask / FastAPI);
- СУБД: PostgreSQL 14+;
- інструменти розробки:
  - IDE: Visual Studio Code, PyCharm;
  - менеджери пакетів: pip, venv;
  - супутні засоби: Git (для контролю версій), Postman (для API-тестування).

Сервер виконання:

- операційна система: Ubuntu 20.04+ або інша Linux-сумісна;
- web-сервер: Gunicorn (для Django) або Uvicorn (для FastAPI);
- база даних: PostgreSQL, розгорнута локально або в Docker-контейнері.

Архітектура розгортання

На рис. 13 зображено діаграму розгортання, що відображає фізичну архітектуру системи. Вона складається з двох основних вузлів:

1. **клієнт (браузер)** — пристрій кінцевого користувача, через який здійснюється доступ до інтерфейсу системи. Інтерфейс реалізований за допомогою HTML, CSS та JavaScript.

2. **сервер застосунку** — сервер, на якому розгорнуто вебзастосунок, реалізований з використанням Django, Flask або FastAPI. Тут обробляються запити, реалізована логіка контролерів та API.

База даних PostgreSQL розгорнута на тому ж сервері, що і застосунок, що спрощує обмін даними між сервісами на початковому етапі. У подальшому, за необхідності, можливе винесення бази на окремий сервер або у хмарне середовище.

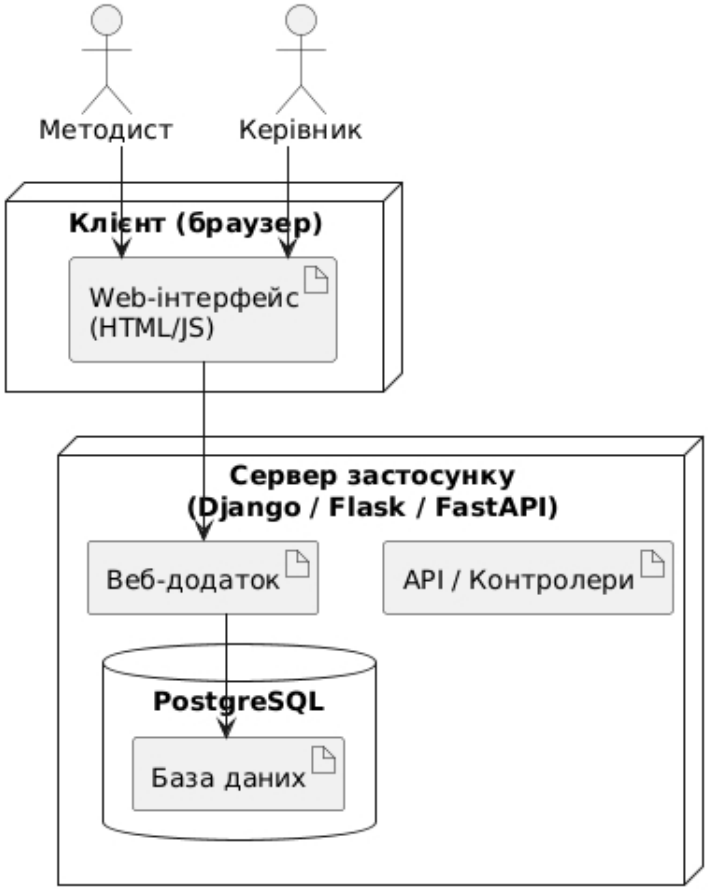


Рис. 13 Діаграма розгортання розробленої системи

Сформована архітектура забезпечує гнучке масштабування, централізоване управління та можливість подальшого розширення функціональності системи. Вимоги до обладнання залишаються помірними, що дозволяє впровадити систему навіть у середовищі з обмеженими ресурсами.

4.3 Склад інсталяційного пакету

Для того щоб програмне забезпечення змогло працювати коректно на сервері або в середовищі розробки, потрібно пройти ряд обов’язкових етапів з інсталяції всіх необхідних компонентів. Основними вимогами до запуску

системи є наявність відповідного програмного середовища, бібліотек, фреймворку та допоміжних інструментів.

Основні кроки підготовки середовища:

1. **Вибір операційної системи**
   - Ubuntu;
   - Debian або інший Linux-дистрибутив.
2. **Установка базових компонентів**
   - Python ≥ 3.10;
   - PostgreSQL ≥ 14;
   - Django ≥ 4.2;
   - Gunicorn — для продакшн-запуску;
   - Nginx — для маршрутизації запитів;
   - Docker — за потреби контейнеризації.
3. **Установка віртуального середовища та бібліотек**

Після створення віртуального середовища (наприклад, через venv) потрібно інсталювати всі залежності з файлу requirements.txt. Він містить бібліотеки, що використовуються у проєкті:

- Django==4.2;
- psycopg2-binary==2.9.6;
- djangorestframework==3.14.0;
- python-dotenv==1.0.0;
- gunicorn==21.2.0;
- whitenoise==6.5.0;

- redis==5.0.1;

Інсталяція. Всі залежності можна встановити за допомогою однієї команди – *pip install -r requirements.txt*

Після встановлення. Після того як усі компоненти будуть успішно встановлені, можна переходити до налаштування змінних середовища, міграцій бази даних, створення суперкористувача та запуску сервера.

Таким чином, інсталяційний пакет містить:

- вихідний код системи;
- requirements.txt;
- документацію для запуску;
- файл README.md.

Після налаштування інфраструктури система може бути розгорнута на сервері або в хмарному середовищі. Це забезпечує її універсальність, гнучкість у встановленні та мінімальну залежність від конкретної ОС або середовища.

4.4 Опис роботи програмного забезпечення

У процесі тестування й перевірки системи обліку випускників було виконано перевірку всіх ключових функцій у реальному середовищі, що підтвердило її працездатність і відповідність поставленим вимогам. Інтерфейс системи доступний через веббраузер і розрахований на роботу з двома основними ролями: **методист** і **керівник**.

Авторизація користувача

На першому етапі користувач вводить логін і пароль, після чого система виконує автентифікацію. На рис. 14 продемонстрована форма автентифікації у застосунок.

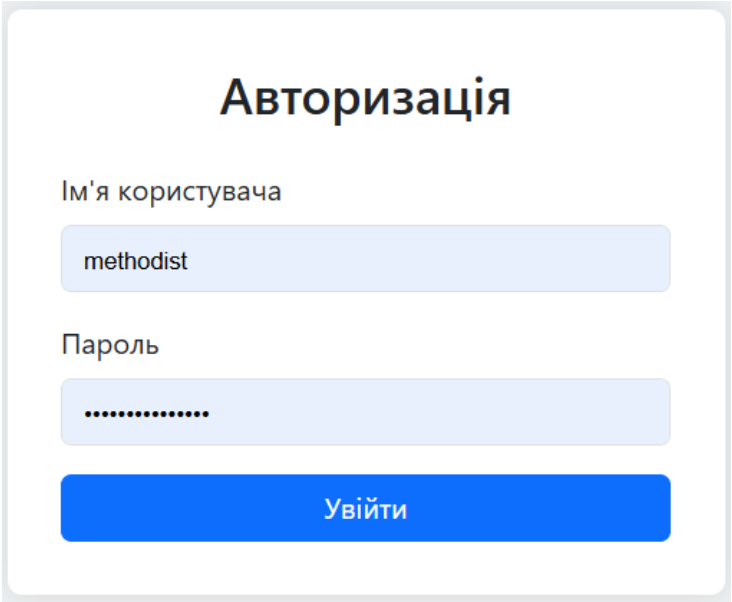


Рис. 14 Форма автентифікації

Перегляд загального списку випускників

Після входу методист потрапляє на головну сторінку з таблицею усіх зареєстрованих випускників, яка продемонстрована на рис. 15. Виводиться ПІБ, контактні дані, спеціальність, рік випуску, а також доступні дії: редагування, видалення, перегляд працевлаштування.

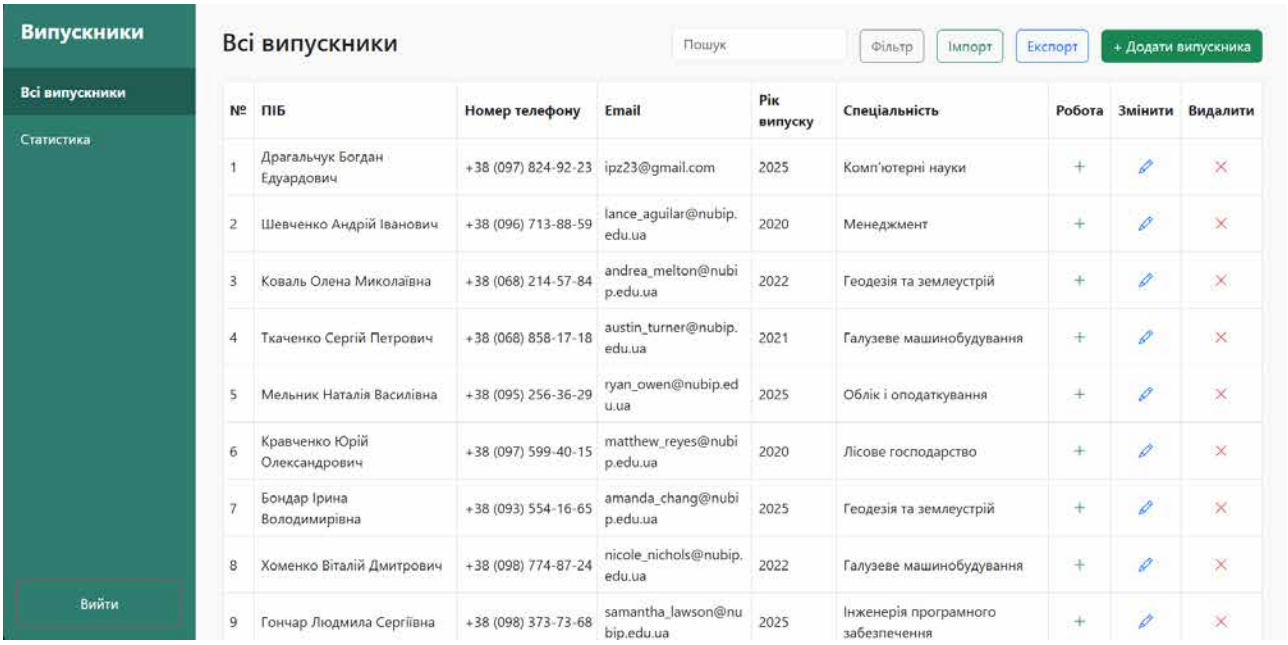


Рис. 15 Головна сторінка з таблицею випускників з панеллю дій

Фільтрація даних за параметрами

Система дозволяє фільтрувати список за роком випуску, факультетом, кафедрою та спеціальністю, що спрощує аналіз інформації, модальне вікно фільтрації продемонстровано на рис. 16.

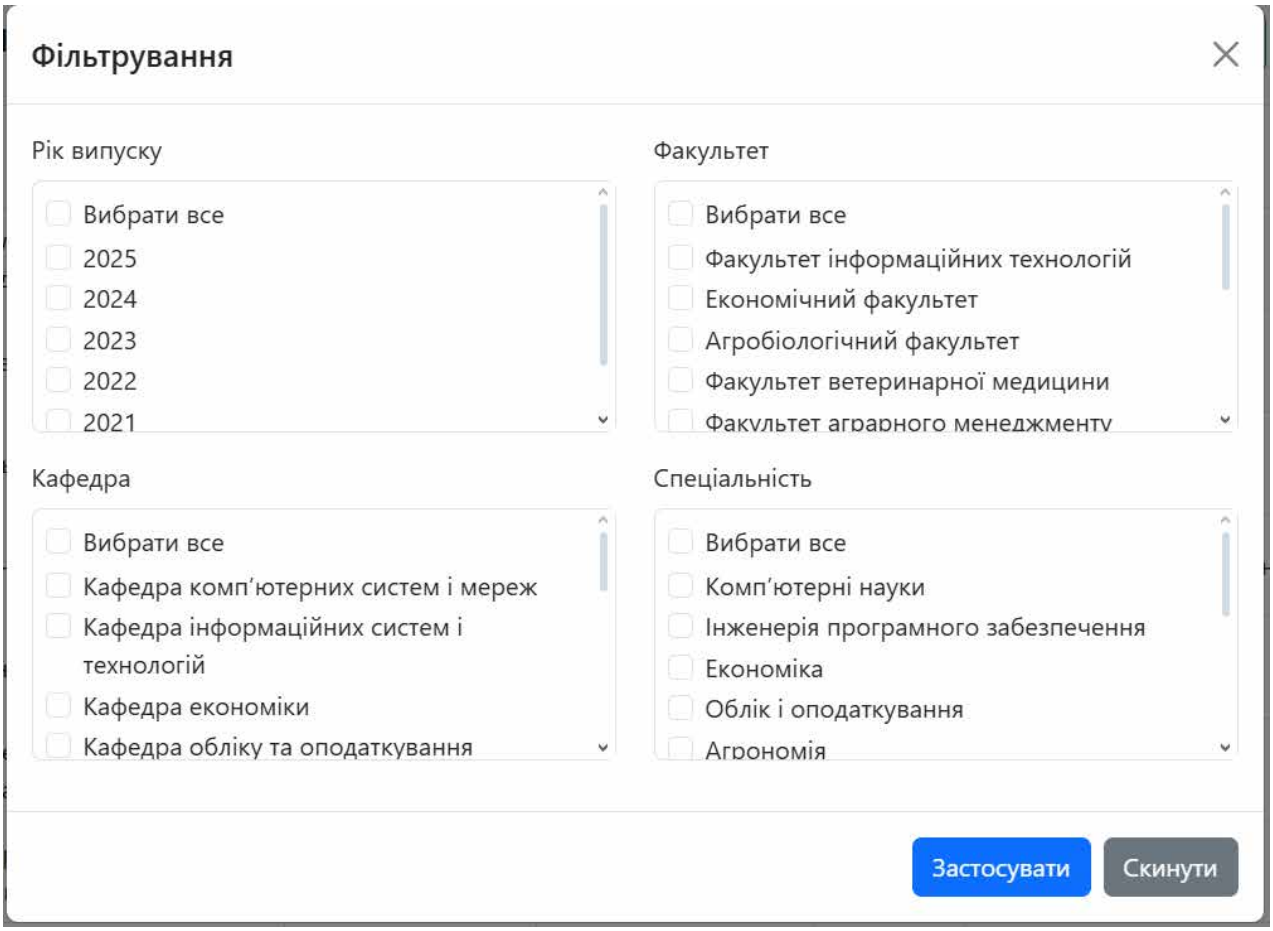


Рис. 16 Модальне вікно фільтрації випускників

Імпорт випускників з Excel-файлу

Методист має можливість завантажити файл формату .xlsx зі списком випускників. Після підтвердження дані автоматично додаються до бази. Модальне вікно вибору файлу продемонстровано на рис. 17.

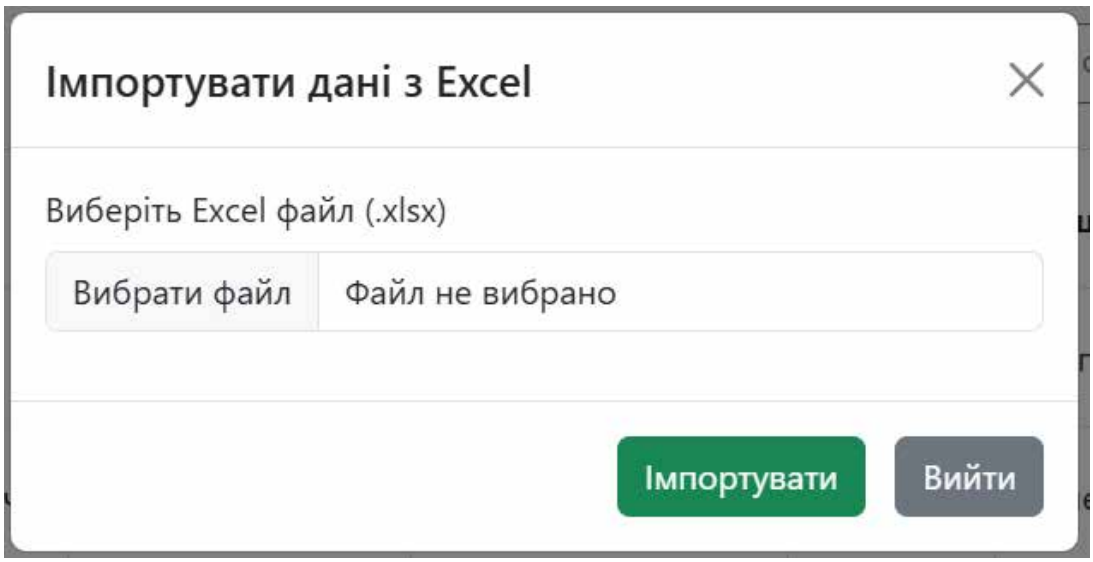


Рис. 17 Модальне вікно імпорт списку випускників

Додавання випускника вручну

При натсканні на кнопку додавання випускника передбачена форма для ручного введення даних випускника з усіма необхідними полями: ім’я, прізвище, дата народження, email, телефон, спеціальність тощо. Форма продемонстрована на рис. 18.

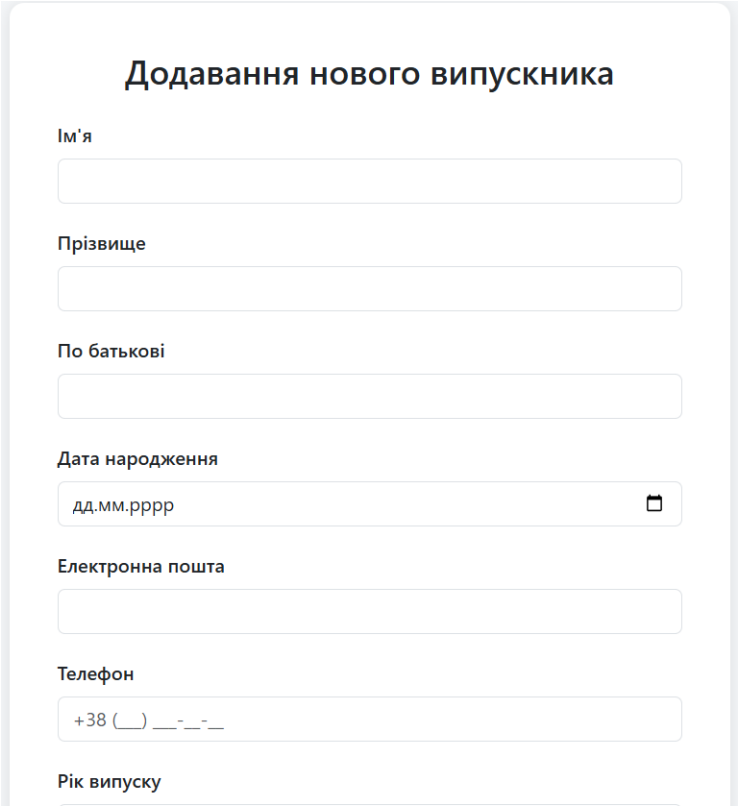


Рис. 18 Форма додавання нового випускника

Перегляд статистики

Окремий розділ призначений для аналітики. Виводиться кількість усіх випускників, кількість працевлаштованих, відсоток працевлаштування та інші метрики. Основна інформація розділу продемонстрована на рис. 19.

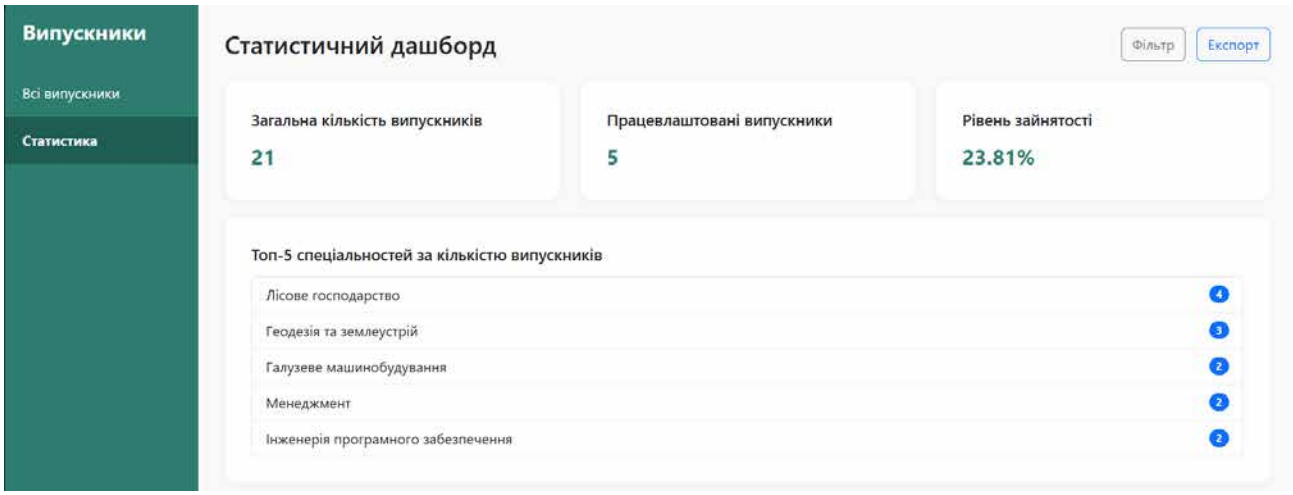


Рис. 19 Панель статистики

Графічне представлення статистик

Система генерує кругову діаграму, гістограму за спеціальностями та стовпчикову діаграму кількості випускників за роками. Кругову діаграму та гістограму продемонстровано на рис. 20. Стовпчикову діаграму кількості випускників за роками показано на рис. 21.

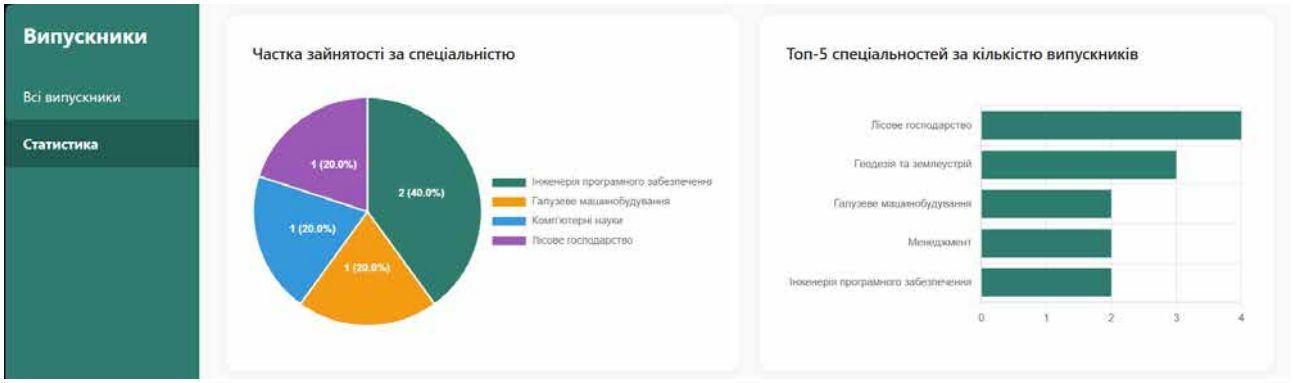


Рис. 20 Статистичні діаграми

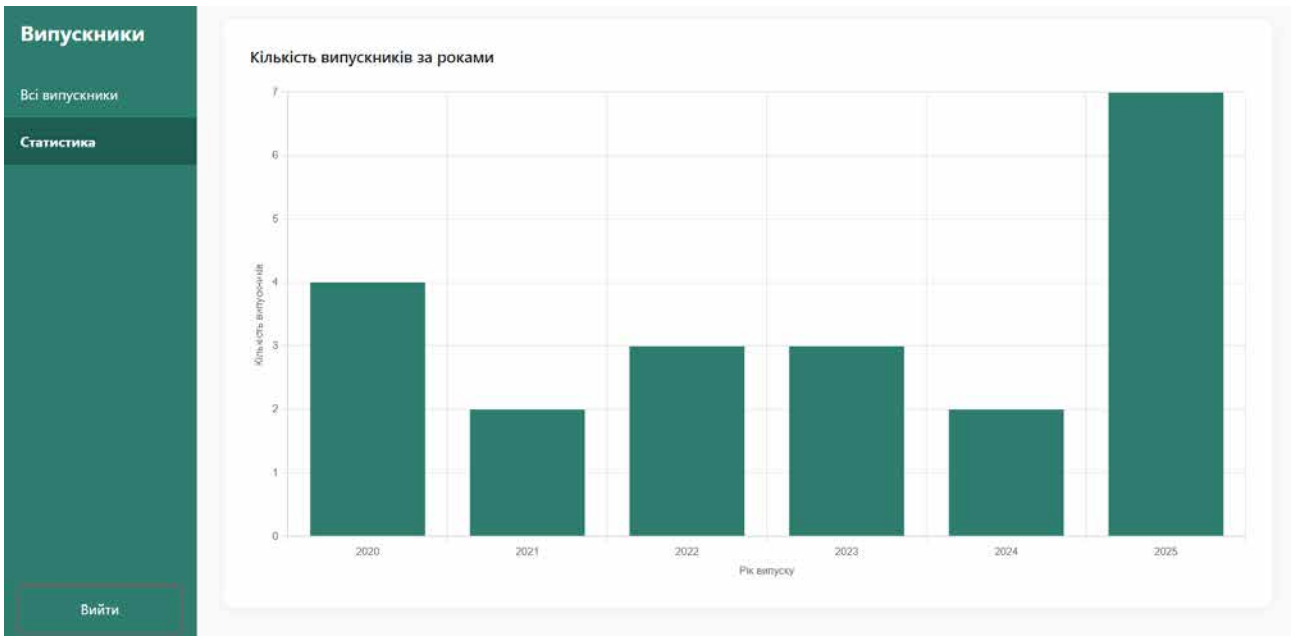


Рис. 21 Кількість випускників за роками

Загальна взаємодія з системою є інтуїтивною та швидкою, а кожен розділ має адаптивний інтерфейс і логічну структуру. Демонстрація підтвердила повну функціональну готовність системи до реального впровадження.

ВИСНОВКИ

У ході виконання дипломної роботи було всебічно досліджено проблему організації обліку випускників у закладах вищої освіти. Проаналізовано типові труднощі, з якими стикаються факультети та кафедри: розрізненість інформації, відсутність централізованої бази, дублювання даних, складність формування звітності та низька аналітична доступність. На основі проведеного аналізу було розроблено архітектуру та реалізовано інформаційну систему, яка дозволяє централізовано зберігати, обробляти та аналізувати дані про випускників.

Розроблена система базується на сучасних технологіях: мові програмування Python, фреймворку Django та реляційній СУБД PostgreSQL. Було побудовано логічну модель даних, реалізовано фізичну структуру бази з урахуванням вимог до нормалізації, зв’язків між сутностями та підтримки цілісності даних. У процесі роботи створено модульну систему з окремими компонентами для імпорту, статистики, перегляду та адміністрування даних.

У програмному продукті реалізовано основні функції: додавання, редагування та видалення інформації про випускників; імпорт даних з Excel-файлів; побудова звітів; перегляд статистики у вигляді графіків; формування аналітичних панелей. Крім того, передбачено рольову модель доступу: методист має повний функціонал для роботи з даними, тоді як керівник обмежений у правах і може лише переглядати статистику й результати.

Використано засоби UML для моделювання структури системи: побудовано діаграми класів, кооперацій, компонентів, пакетів, що дозволило формалізувати логіку предметної області. Описано процес створення та взаємодії між модулями системи у вигляді блок-схем і структурних діаграм.

Було виконано тестування основного функціоналу, визначено вимоги до апаратного та програмного забезпечення, сформовано інсталяційний пакет і підготовлено рекомендації щодо розгортання системи. У рамках демонстрації

було успішно перевірено повний цикл взаємодії — від авторизації до створення та перегляду статистичних звітів.

Представлене рішення має практичну цінність та може бути впроваджене у навчальному закладі для автоматизації обліку випускників, полегшення звітності та покращення аналітики.

Таким чином, у дипломному проєкті реалізовано повноцінну інформаційну систему, яка відповідає сучасним вимогам до програмного забезпечення та має всі передумови для впровадження й подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Система управління відносинами з випускниками: [Електронний ресурс] – Режим доступу: https://gradnet.io (дата звернення 12.01.2025)

2. Система управління спільнотами: [Електронний ресурс] – Режим доступу: https://hivebrite.io (дата звернення 12.01.2025)

3. Oracle Database: [Електронний ресурс] – Режим доступу: https://www.oracle.com/database/ (дата звернення 17.01.2025)

4. MySQL Database: [Електронний ресурс] – Режим доступу: https://www.mysql.com (дата звернення 17.01.2025)

5. Microsoft SQL Server Database: [Електронний ресурс] – Режим доступу: https://www.microsoft.com/uk-ua/sql-server/sql-server-2022 (дата звернення 17.01.2025)

6. PostgreSQL Database: [Електронний ресурс] – Режим доступу: https://www.postgresql.org (дата звернення 17.01.2025)

7. Django: The web framework for perfectionists with deadlines: [Електронний ресурс] – Режим доступу: https://www.djangoproject.com/ (дата звернення 23.01.2025)

8. Django - Connect to PostgreSQL Database. W3Schools: [Електронний ресурс] – Режим доступу: https://www.w3schools.com/django/django_db_connect.php (дата звернення 23.01.2025)

9. Python in Visual Studio Code: [Електронний ресурс] – Режим доступу: https://code.visualstudio.com/docs/languages/python (дата звернення 23.01.2025)

10. Introduction Bootstrap v5.0: [Електронний ресурс] – Режим доступу: https://getbootstrap.com/docs/5.0/getting-started/introduction/ (дата звернення 30.01.2025)

11. Django documentation: [Електронний ресурс] – Режим доступу: https://docs.djangoproject.com/en/5.2/ (дата звернення: 30.01.2025)

12. PostgreSQL 16.9 Documentation: [Електронний ресурс] – Режим доступу: https://www.postgresql.org/files/documentation/pdf/16/postgresql-16-A4.pdf (дата звернення: 04.02.2025)

13. Python 3.13.3 documentation: [Електронний ресурс] – Режим доступу: https://docs.python.org/ (дата звернення: 12.02.2025)

14. Bootstrap 5 Tutorial - W3Schools: [Електронний ресурс] – Режим доступу: https://www.w3schools.com/bootstrap5/ (дата звернення: 04.02.2025)

15. Django REST framework: [Електронний ресурс] – Режим доступу: https://www.django-rest-framework.org/ (дата звернення: 12.02.2025)

16. PostgREST Documentation: [Електронний ресурс] – Режим доступу: https://postgrest.org/ (дата звернення: 12.02.2025)

17 Python Official Documentation: [Електронний ресурс] – Режим доступу: https://www.python.org/doc/ (дата звернення: 12.02.2025)

18. Getting Started with Python in VS Code: [Електронний ресурс] – Режим доступу: https://code.visualstudio.com/docs/python/python-tutorial (дата звернення: 23.01.2025)

19. Django Web Framework (Python) - Learn web development | MDN: [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django (дата звернення: 23.02.2025)

20. Bootstrap The most popular HTML, CSS, and JS library in the world: [Електронний ресурс] – Режим доступу: https://getbootstrap.com/ (дата звернення: 04.02.2025)

21. Python - Visual Studio Marketplace: [Електронний ресурс] – Режим доступу: https://marketplace.visualstudio.com/items?itemName=ms-python.python (дата звернення: 11.04.2025)

22. Python Reference - W3Schools: [Електронний ресурс] – Режим доступу: https://www.w3schools.com/python/python_reference.asp (дата звернення: 11.04.2025)

23. Django 5.2.1 documentation: [Електронний ресурс] – Режим доступу: https://django.readthedocs.io/en/stable/contents.html (дата звернення: 10.05.2025)

24.PostgreSQL Tutorial - Neon: [Електронний ресурс] – Режим доступу: https://neon.tech/postgresql/tutorial (дата звернення: 10.05.2025)

25. Основні методи тестування програмного забезпечення: [Електронний ресурс] – Режим доступу: https://conf.ztu.edu.ua/wp-content/uploads/2016/06/248.pdf (дата звернення: 18.05.2025)

26. Тестування програмного забезпечення (навчальний посібник): [Електронний ресурс] – Режим доступу: https://eprints.cdu.edu.ua/1482/1/testyvan.pdf (дата звернення: 18.05.2025)

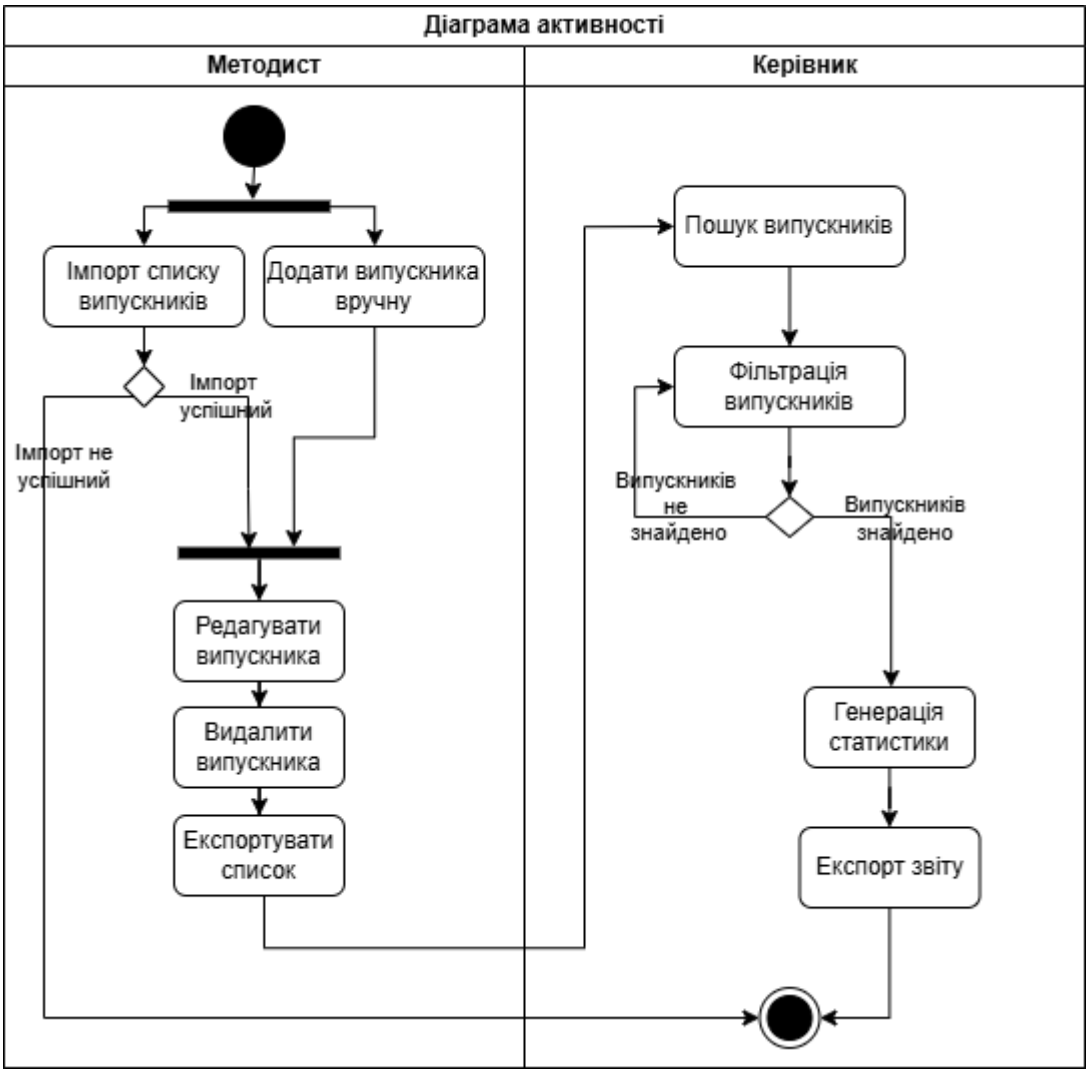
27. Типи тестування ПЗ (100 прикладів): [Електронний ресурс] – Режим доступу: https://www.guru99.com/uk/types-of-software-testing.html (дата звернення: 18.05.2025)

ДОДАТОК А

Діаграма послідовності

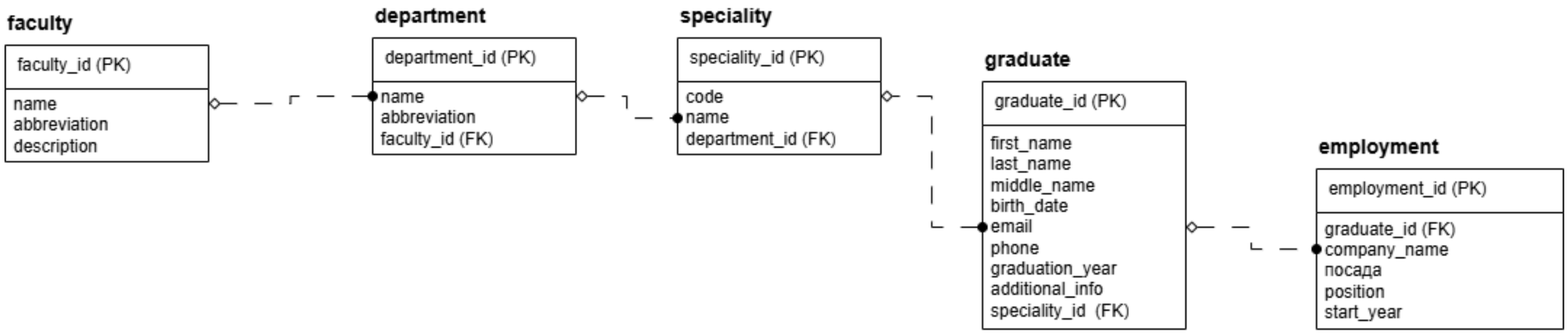
ДОДАТОК Б

Діаграма активності



ДОДАТОК В

Логічна модель



ДОДАТОК Д

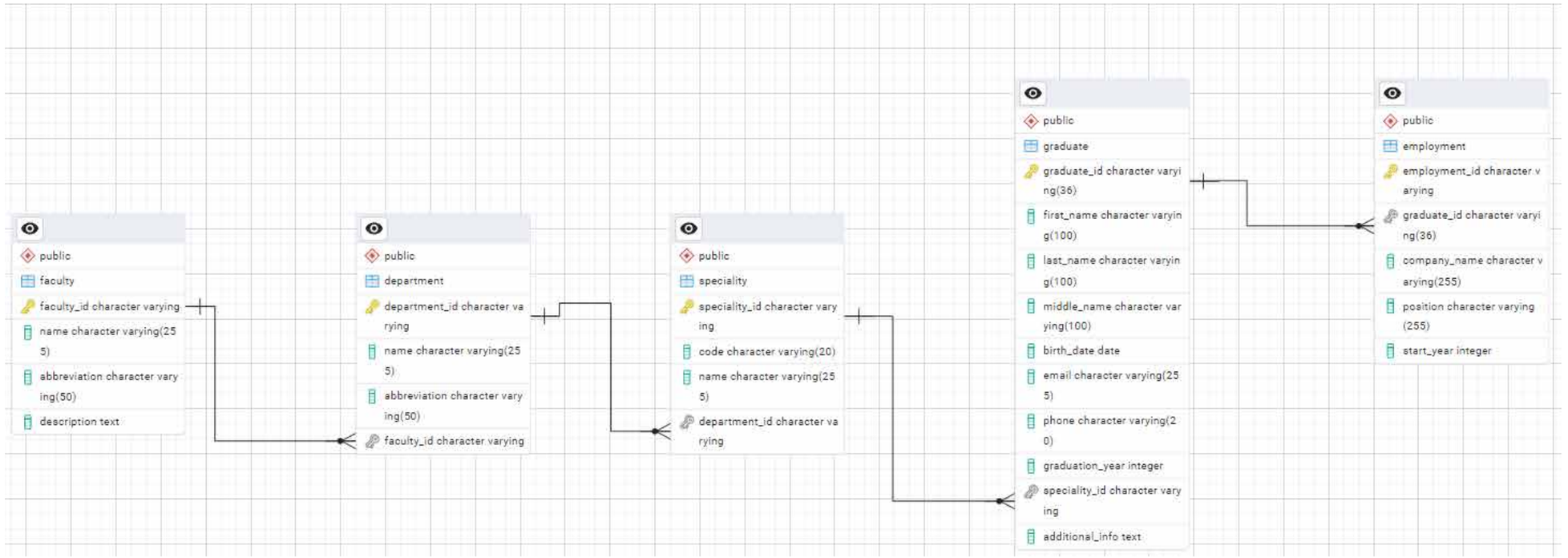
SQL структура таблиць

```
CREATE TABLE faculty (
    faculty_id VARCHAR PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    abbreviation VARCHAR(50),
    description TEXT);
CREATE TABLE department (
    department_id VARCHAR PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    abbreviation VARCHAR(50),
    faculty_id VARCHAR NOT NULL,
    CONSTRAINT fk_department_faculty FOREIGN KEY (faculty_id)
        REFERENCES faculty(faculty_id)
        ON DELETE CASCADE);
CREATE TABLE speciality (
    speciality_id VARCHAR PRIMARY KEY,
    code VARCHAR(20),
    name VARCHAR(255) NOT NULL,
    department_id VARCHAR NOT NULL,
    CONSTRAINT fk_speciality_department FOREIGN KEY (department_id)
        REFERENCES department(department_id)
```

```
    ON DELETE CASCADE);
CREATE TABLE graduate (
    graduate_id VARCHAR(36) PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    middle_name VARCHAR(100),
    birth_date DATE,
    email VARCHAR(255),
    phone VARCHAR(100),
    graduation_year INTEGER NOT NULL,
    speciality_id VARCHAR NOT NULL,
    additional_info TEXT,
    CONSTRAINT fk_graduate_speciality FOREIGN KEY (speciality_id)
        REFERENCES speciality(speciality_id)
        ON DELETE SET NULL);
CREATE TABLE employment (
    employment_id VARCHAR PRIMARY KEY,
    graduate_id VARCHAR(36) NOT NULL,
    company_name VARCHAR(255),
    position VARCHAR(255),
    start_year INTEGER,
    CONSTRAINT fk_employment_graduate FOREIGN KEY (graduate_id)
        REFERENCES graduate(graduate_id) ON DELETE CASCADE);
```

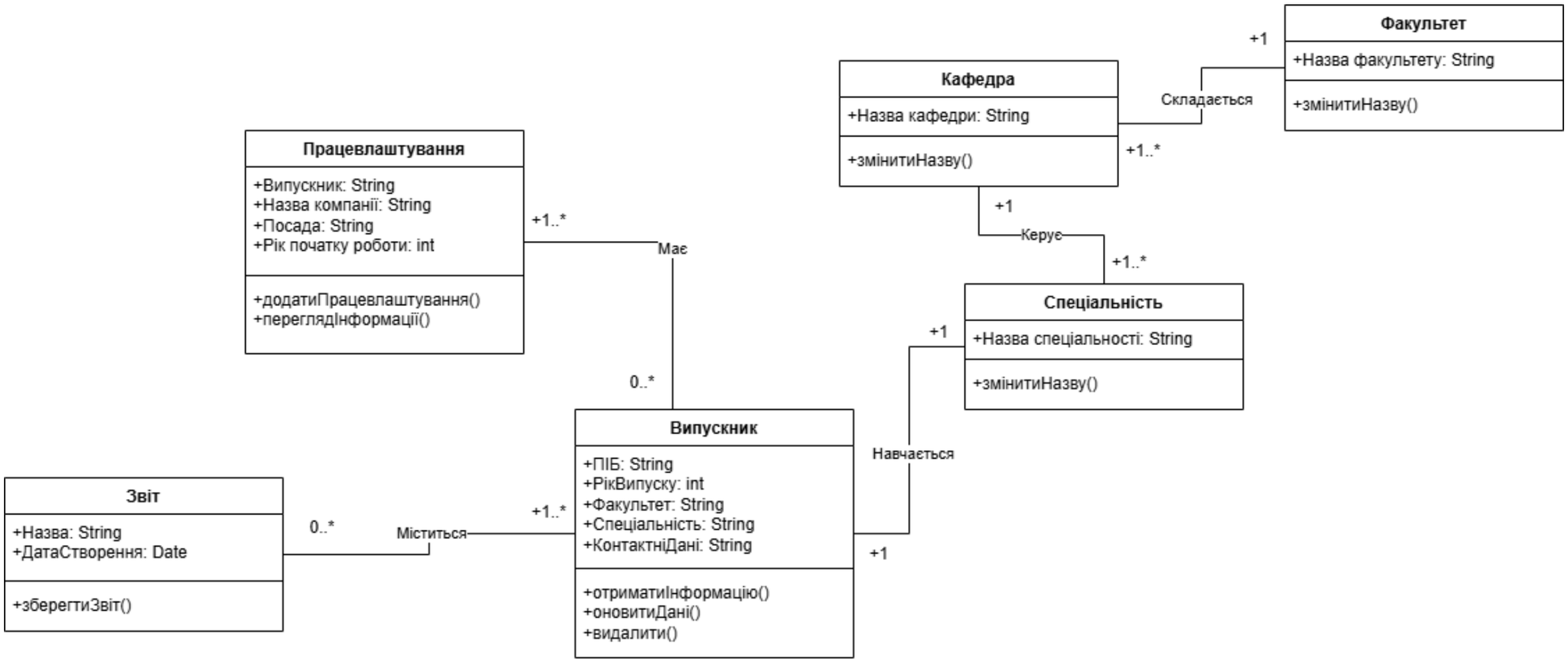
ДОДАТОКЕ

Фізична модель



ДОДАТОК Ж

Діаграма класів



ДОДАТОК К

Код функцій генерації статистики

```
def statistics(request):
    graduates = Graduate.objects.all()

    selected_years = request.GET.getlist('graduation_year')
    selected_faculties = request.GET.getlist('faculty')
    selected_departments = request.GET.getlist('department')
    selected_specialities = request.GET.getlist('speciality')

    if selected_years:
        graduates = graduates.filter(graduation_year__in=selected_years)
    if selected_faculties:
        graduates =
graduates.filter(speciality__department__faculty__faculty_id__in=selected_faculties)
    if selected_departments:
        graduates =
graduates.filter(speciality__department__department_id__in=selected_departments)
    if selected_specialities:
        graduates = graduates.filter(speciality__speciality_id__in=selected_specialities)

    total_graduates = graduates.count()
    total_employed = graduates.filter(employment__isnull=False).distinct().count()
```

```
    employment_rate = round((total_employed / total_graduates) * 100, 2) if
total_graduates else 0


    top_specialities = (
        Speciality.objects
        .filter(graduate__in=graduates)
        .annotate(num_graduates=Count('graduate'))
        .order_by('-num_graduates')[:5]
    )


    context = {
        'graduates': graduates,
        'total_graduates': total_graduates,
        'total_employed': total_employed,
        'employment_rate': employment_rate,
        'top_specialities': top_specialities,
        'faculties': Faculty.objects.all(),
        'departments': Department.objects.all(),
        'specialities': Speciality.objects.all(),
        'years': Graduate.objects.values_list('graduation_year',
flat=True).distinct().order_by('-graduation_year'),
        'selected_years': request.GET.getlist('graduation_year'),
        'selected_faculties': request.GET.getlist('faculty'),
```

```
        'selected_departments': request.GET.getlist('department'),
        'selected_specialities': request.GET.getlist('speciality'),
    }

    employments_by_speciality = (
        Speciality.objects
        .filter(graduate__in=graduates, graduate__employment__isnull=False)
        .annotate(count=Count('graduate', distinct=True))
        .order_by('-count')
    )

    chart_data = {
        'labels': [s.name for s in employments_by_speciality],
        'counts': [s.count for s in employments_by_speciality]
    }

    context.update({
        'chart_data': chart_data
    })
    top_specialities_labels = [name for name, count in
top_specialities.values_list('name', 'num_graduates')]
    top_specialities_counts = [count for name, count in
top_specialities.values_list('name', 'num_graduates')]
```

```
context['top_specialities_labels'] = top_specialities_labels
context['top_specialities_counts'] = top_specialities_counts

graduates_by_year = (
    graduates.values('graduation_year')
    .annotate(count=Count('graduate_id'))
    .order_by('graduation_year')
)

chart_by_year = {
    'labels': [str(row['graduation_year']) for row in graduates_by_year],
    'counts': [row['count'] for row in graduates_by_year]
}

context.update({
    'chart_by_year': chart_by_year
})

return render(request, 'statistics.html', context)
```