МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ ДАНИМИ СТУДЕНТІВ»**

Виконав: студент 2 курсу, групи 8.1224
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
В.З. Зубко
(ініціали та прізвище)

Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" ____ " _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Зубку Владиславу Заурійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка інформаційної системи управління даними студентів

керівник роботи Матвіїшина Надія Вікторівна, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 327-с

2. Строк подання студентом роботи 25.11.2025

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка інформаційної системи управління даними студентів

5. Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 03.03.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	19.03.2025	
2.	Збір вихідних даних.	01.04.2025	
3.	Обробка методичних та теоретичних	15.06.2025	
	джерел.		
4.	Розробка першого та другого розділу.	25.06.2025	
5.	Розробка третього розділу.	21.09.2025	
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи магістра.	25.11.2025	
7.	Захист кваліфікаційної роботи.	10.12.2025	

Студент _____ В.З. Зубко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.В. Матвіїшина
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка інформаційної системи управління даними студентів»: 137 с., 28 рис., 36 джерел, 5 додатки.

автентифікація, архітектура mvc, аудит, база даних, безпека, вебзастосунок, військовий облік, Django, Excel, звітність, студент, управління даними.

Об’єкт дослідження – процес управління даними студентів для ведення військового обліку в закладі вищої освіти.

Предмет дослідження – моделі, методи та програмні засоби для розробки інформаційної системи, що автоматизує процес військового обліку студентів.

Мета роботи: розробка та реалізація інформаційної системи для автоматизації, підвищення ефективності та безпеки процесу ведення військового обліку студентів у навчальному закладі.

Методи дослідження – аналіз, синтез, проєктування, моделювання.

В результаті виконання роботи розроблено інформаційну систему у вигляді вебзастосунку для автоматизації військового обліку студентів, яка забезпечує повний цикл обробки даних: від імпорту та валідації до генерації регламентованої звітності та аудиту змін. Програмну реалізацію виконано з використанням мови програмування Python, фреймворку Django, СУБД SQLite та бібліотеки openpyxl для інтеграції з електронними таблицями. Система призначена для працівників військово-облікових підрозділів закладів вищої освіти та дозволяє суттєво зменшити трудомісткість обробки персональних даних і мінімізувати вплив людського фактора. Перспективи подальшого розвитку проєкту включають міграцію на промислову СУБД PostgreSQL, розширення аналітичного модуля.

SUMMARY

Master’s Qualifying Theses « Development of an Information System for Student Data Management »: 137 pages, 28 figures, 36 references, 5 supplements.

AUTHENTICATION, MVC ARCHITECTURE, AUDIT, DATABASE, SECURITY, WEB APPLICATION, MILITARY REGISTRY, DJANGO, EXCEL, REPORTING, STUDENT, DATA MANAGEMENT.

Object of study – the models, methods, and software tools for the development of an information system that automates the process of student military registry.

Subject of the study – the models, methods, and software tools for the development of an information system that automates the process of student military registry.

Aim of the study: the development and implementation of an information system to automate, enhance efficiency, and ensure the security of the student military registry process in an educational institution.

Methods of research – analysis, synthesis, design, and modeling.

As a result of the thesis, information system in the form of a web application for automating student military registry has been developed, ensuring a full data processing cycle: from import and validation to regulated reporting generation and audit of changes. The software implementation was carried out using the Python programming language, the Django framework, the SQLite DBMS, and the openpyxl library for spreadsheet integration. The system is intended for staff of military registry departments in higher education institutions and allows for significantly reducing the labor intensity of personal data processing and minimizing the impact of human error. Prospects for further project development include migration to the industrial PostgreSQL DBMS, expansion of the analytical module.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 44
1 Аналіз предметної області та постановка задачі 47
1.1 Огляд та аналіз предметної області військового обліку студентів 47
1.2 Огляд та аналіз існуючих методів та засобів вирішення задачі 48
1.3 Постановка задачі та формулювання вимог до системи 55
1.4 Аналіз та обґрунтування вибору методів та засобів розробки 57
2 Проєктування архітектури та структури інформаційної системи 60
2.1 Проєктування загальної архітектури системи (MVT) 60
2.2 Проєктування бази даних та моделей даних 64
2.2.1 Модель користувача та реалізація рольової моделі 65
2.2.2 Проєктування центральної моделі 67
2.2.3 Проєктування моделі аудиту 68
2.2.4 Проєктування моделі імпорту 69
2.3 Проєктування підсистеми безпеки 70
2.3.1 Проєктування механізмів аутентифікації та сесій 71
2.3.2 Проєктування захисту від брутфорсу, IP-блокування та rate-limiting 72
2.3.3 Проєктування системи аудит-логування 74
2.4 Проєктування форм та логіки валідації 75
3 Програмна реалізація та тестування модулів системи 79
3.1 Реалізація модуля управління даними студентів (CRUD) та історії редагування 79
3.2 Реалізація модуля інтеграції з Excel 82

3.2.1 Програмна реалізація експорту даних у звітні форми 83
3.2.2 Програмна реалізація імпорту та синхронізації даних 84
3.2.3 Реалізація механізму автоматичного визначення факультету 85
3.3 Реалізація підсистеми звітності та аналітики 87
3.3.1 Реалізація модуля «Список боржників» 87
3.3.2 Реалізація модуля «Призовники 25+» 88
3.4 Реалізація підсистеми управління користувачами та розмежування доступу 90
3.5 Демонстрація функціональних можливостей та інтерфейсу системи 94
Висновки 102
Перелік посилань 104
Додаток А Models.py 108
Додаток Б Views.py 117
Додаток В Excel_Utils.py 144
Додаток Г Security.py 165
Додаток Д Forms.py 168

ВСТУП

Впроваджувати інформаційні технології в освітній процес та адміністративну діяльність закладів вищої освіти є ключовою вимогою сучасності. Особливо гостро це стосується процесів, пов'язаних з управлінням великими обсягами персональних даних, де точність, безпека та оперативність відіграють вирішальну роль. Одним з таких процесів є ведення військового обліку студентів.

Традиційні підходи до ведення обліку, що часто покладаються на незв'язані електронні таблиці, наприклад, Excel, або паперовий документообіг, стикаються з низкою суттєвих проблем. До них належать висока ймовірність механічних помилок при введенні даних, відсутність централізованого контролю доступу, ризики несанкціонованої зміни або втрати чутливої інформації, а також відсутність прозорого механізму аудиту. Складність у формуванні звітності, зокрема специфічних звітів, таких як списки студентів із неповними даними, або аналітика за віковими категоріями, призводить до значних витрат часу адміністративного персоналу.

Крім того, відсутність належної системи управління користувачами та їхніми правами створює загрозу цілісності даних. Наприклад, будь-який користувач, що має доступ до файлу, може випадково або навмисно видалити критично важливі записи. Окремою проблемою є синхронізація даних, особливо коли дані надходять з різних джерел, наприклад, зі списків зарахованих бакалаврів та магістрів.

Таким чином, актуальність даної роботи полягає у нагальній потребі розробки спеціалізованої, захищеної та централізованої інформаційної системи. Така система має на меті не просто зберігання даних, а повну автоматизацію життєвого циклу управління даними військового обліку: від первинного імпорту та створення запису до формування фінальної звітності, переміщення студента в архів та відстеження всієї історії змін.

Метою кваліфікаційної роботи є розробка та реалізація інформаційної системи для автоматизації, підвищення ефективності та безпеки процесу ведення військового обліку студентів у навчальному закладі.

Для досягнення поставленої мети було визначено наступні завдання:

– провести аналіз предметної області, виявити недоліки існуючих підходів та сформулювати функціональні й нефункціональні вимоги до системи;
– спроєктувати архітектуру вебзастосунку на основі патерну MVC (Model-View-Controller) та обрати відповідний технологічний стек: Python, Django, SQLite;
– розробити структуру бази даних, що відображає всі необхідні сутності: студенти, користувачі системи, історія змін, ТЦК та СП, додаткові відмітки;
– реалізувати основний функціонал управління студентами: додавання, редагування, фільтрація, пошук та видалення студентів;
– розробити модуль інтеграції з Excel, що забезпечує імпорт даних з автоматичним оновленням, завантаження списків студентів, які навчаються, та експорт даних у встановлених форматах;
– реалізувати підсистему автоматизованої звітності: формування списків боржників з неповними даними та списків призовників, що досягають 25-річного віку;
– спроєктувати та впровадити комплексну систему безпеки, що включає рольову модель доступу (Admin, Employee), захист від брутфорс-атак, логування дій та аудит змін.

Об’єктом дослідження є процес управління даними студентів для ведення військового обліку в закладі вищої освіти.

Предметом дослідження є моделі, методи та програмні засоби для розробки інформаційної системи, що автоматизує процес військового обліку студентів.

Для досягнення поставлених завдань було обрано фреймворк Django. Цей вибір обґрунтований його перевагами: наявність вбудованої ORM для швидкої розробки моделі даних, готова система аутентифікації та адміністрування, а також потужні вбудовані механізми безпеки (захист від CSRF, XSS, Clickjacking). Використання архітектури MVC (або MVT в термінології Django) дозволяє чітко розділити логіку роботи з даними, їх обробку та представлення.

Ключовим рішенням у сфері безпеки та цілісності даних стало впровадження рольової моделі. Система чітко розмежовує права: Працівник (employee) може виконувати щоденні операції (додавання, редагування студентів), але не має доступу до видалення записів чи зміни налаштувань. Адміністратор (admin) має повний контроль над системою, включаючи управління користувачами та налаштуваннями довідників.

Для забезпечення повноцінного аудиту кожна зміна даних студента фіксується в окремій моделі StudentHistory. Ця модель зберігає JSON-поле зі змінами, посилання на користувача, який вніс зміни, та час. Це рішення гарантує повну прозорість та підзвітність дій у системі.

Інтеграція з Excel через бібліотеку openpyxl була обрана як компромісне рішення, що дозволяє зберегти звичні для користувачів інструменти та забезпечити сумісність з іншими системами, водночас переносячи основну логіку валідації та зберігання даних на бік сервера.

У результаті виконання роботи буде розроблено повнофункціональний вебзастосунок, що повністю автоматизує процес ведення військового обліку.

Розроблена система буде введена до безпосереднього впровадження в адміністративному підрозділі мобілізаційної роботи та цивільного захисту у закладі вищої освіти. Архітектура системи дозволяє її подальше масштабування та модифікацію: додавання нових звітних форм або адаптацію для інших завдань обліку персоналу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд та аналіз предметної області військового обліку студентів

Процес військового обліку студентів у закладах вищої освіти є складною та відповідальною адміністративною задачею. Він регламентується чинним законодавством та вимагає від уповноважених осіб не лише своєчасного збору та подання даних, але й забезпечення їхньої високої точності, конфіденційності та цілісності. Предметна область характеризується роботою з великими масивами персональних даних, що належать до чутливої категорії.

Основними суб’єктами цього процесу є, з одного боку, студенти, що поділяються на різні категорії: призовники, військовозобов’язані, резервісти, та, з іншого боку, адміністративний персонал ЗВО, що відповідає за ведення обліку. Зовнішнім суб’єктом виступають Територіальні центри комплектування та соціальної підтримки , куди регулярно подається звітність.

Основні завдання, що вирішуються в рамках предметної області, включають: первинний збір повного спектра даних: категорія військового обов’язку, військове звання, ПІБ, дата народження, реєстраційний номер облікового запису в Єдиному державному реєстрі Призовників, військовозобов’язаних та резервістів, військово-облікова спеціальність, РНОКПП, реквізити військово-облікового документа, реквізити паспорта громадянина України, адреса задекларованого/зареєстрованого або фактичного місця проживання, найменування територіального центру комплектування та соціальної підтримки, відомості про відстрочку, або бронювання, відомості про проходження військової служби, відомості про наявність мобілізаційного розпорядження, посада, реквізити акта про

призначення на посаду. Так само і забезпечення захищеного зберігання даних, та оперативну актуалізацію при будь-яких змінах, наприклад, зарахування, зміна прізвища, оновлення даних ТЦК та формування стандартизованих звітів.

Критичною особливістю є необхідність ведення внутрішнього контролю, зокрема моніторингу повноти та коректності заповнення даних, що в описі системи відображено як функція «Список боржників». Ця функція вимагає складної логіки, оскільки перелік обов'язкових полів відрізняється для різних категорій студентів; наприклад, для призовників не перевіряються поля, пов'язані зі званням чи відстрочкою. Також система повинна постійно відстежувати специфічні події, як-от досягнення призовниками 25-річного віку.

Таким чином, предметна область вимагає рішення, яке б значно перевищувало можливості простого сховища даних. Необхідна система, що здатна інкапсулювати складну бізнес-логіку, забезпечувати валідацію даних на етапі введення та автоматизувати рутинні операції контролю та звітності.

1.2 Огляд та аналіз існуючих методів та засобів вирішення задачі

Для вирішення завдань військового обліку на практиці ЗВО використовують різні підходи, що суттєво відрізняються за рівнем автоматизації, надійності та безпеки.

Найпростішим та найбільш застарілим є метод ведення паперового документообігу, що полягає у веденні особових справ, картотек та журналів. Цей підхід унеможливлює ефективний пошук, аналіз та автоматичне формування звітів, а також створює високі ризики фізичної втрати чи пошкодження даних.

Наступним кроком прогресу є використання децентралізованих електронних таблиць, найчастіше MS Excel. Цей метод є надзвичайно

поширеним завдяки доступності та низькому порогу входження. Однак, аналіз завдань предметної області та функціоналу розробленої системи дозволяє виявити фундаментальні недоліки цього підходу.

По-перше, електронні таблиці не забезпечують цілісність та валідацію даних. Користувач може ввести дату в текстовому форматі, РНОКПП з текстовими символами або допустити помилку у назві факультету. В імпорті даних з Excel прямо вказано на необхідність очищення РНОКПП від «.0», що є типовою проблемою автоформатування в Excel.

По-друге, відсутній будь-який дієвий механізм контролю доступу. Файл, що зберігається у спільній папці, може бути відкритий, скопійований, змінений або видалений будь-якою особою, що має доступ до цієї папки. Реалізувати гранульовані права, наприклад, «Працівник» може лише додавати та редагувати, але не видаляти в рамках Excel неможливо.

По-третє, повністю відсутній механізм аудиту. Неможливо відстежити, хто, коли і які саме зміни вніс до запису студента. Це є критичним недоліком при роботі з чутливими даними, де ціна помилки або несанкціонованої зміни є надзвичайно високою.

По-четверте, виникають серйозні проблеми синхронізації. Якщо двоє працівників одночасно відкриють файл та внесуть зміни, дані одного з них будуть втрачені. Окрім того, самі файли схильні до пошкодження, що вимагало б ручного створення резервних копій. Розроблена система вирішує цю проблему шляхом створення backup-файлу при пошкодженні та відновлення структури.

Окремі організації можуть використовувати локальні системи управління базами даних, наприклад, MS Access. Хоча вони й надають кращі засоби валідації та структурування даних, вони зазвичай прив’язані до локальної мережі, не пропонують сучасних вебінтерфейсів та не мають вбудованих механізмів захисту від сучасних кіберзагроз, таких як брутфорс-атаки чи CSRF-атаки, захист від яких є однією з ключових особливостей розробленої системи.

Огляд наукових праць показує, що існує значна кількість досліджень з автоматизації освітніх процесів та управління даними студентів, але вузькоспеціалізовані аспекти, як-от військовий облік із підвищеними вимогами до безпеки та аудиту, залишаються недостатньо висвітленими:

- у дослідженнях [8] аналізуються сучасні реалізації програмного забезпечення для управління житловими комплексами, що має спільні риси з HRM-системами. Подібні рішення можуть бути адаптовані для освітніх закладів, де важливим є ведення обліку студентів та персоналу, формування звітності та контроль доступу;
- робота [9] присвячена розробці алгоритмів та архітектури анонімних інформаційних систем комунікації, що демонструє актуальність питань захисту даних та конфіденційності в освітніх IT-системах;
- у деяких працях [10] розглядаються інтелектуальні системи управління групами мобільних роботів та моделювання паралельних обчислень стохастичними мережами Петрі. Хоча ці дослідження не стосуються безпосередньо студентських інформаційних систем, вони демонструють підходи до проектування складних систем управління, що можуть бути використані при створенні освітніх IT-рішень із високими вимогами до надійності та масштабованості;
- у статті [11] наведено огляд існуючих рішень для автоматизації роботи адміністрації ЗВО. Автори підкреслюють важливість модулів для обліку студентів, ведення навчальних планів та формування звітності. Проєкт надає змогу інтегрувати розклад занять студентів на сайт вищого навчального закладу, його оптимізовано для використання на сторінках з великим відвідуванням, також, як вказує розробник, його можна налаштувати для органічного відображення відповідно до стилю сайту навчального закладу. Показано приклад відображення

розкладу навчальних груп та викладача, які отримано з демоверсії програми (див. рис. 1.1 – 1.2)



Розклад

Виберіть навчальну групу та дату або діапазон дат для перегляду розкладу

Фільтри: Факультет лінгвістики | Денна | 1 курс

Навчальна група: ФЛ-11

Дата від: 12.12.2022 Дата по: 16.12.2022

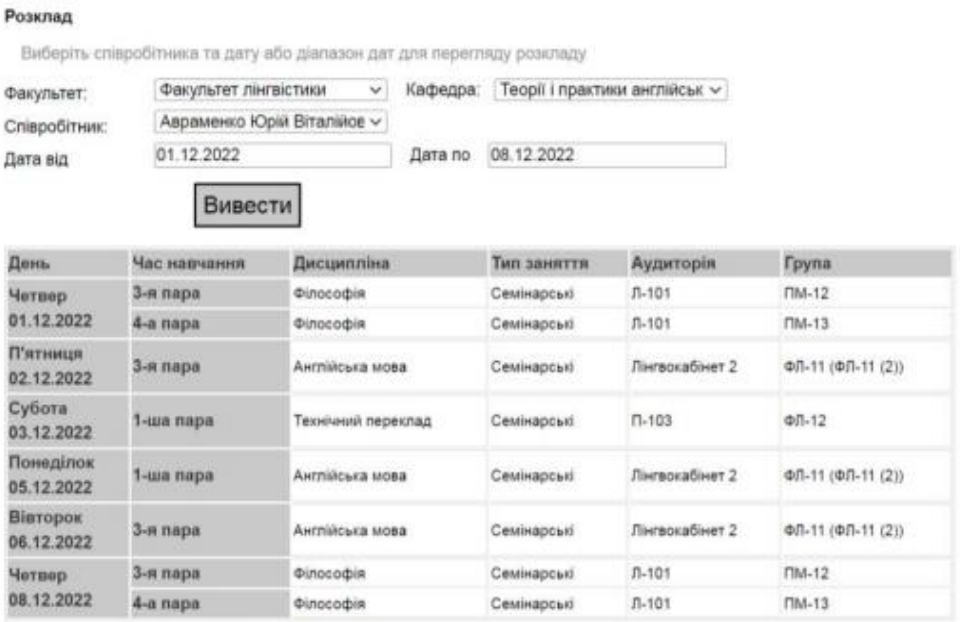
Вивести

Ви переглядаєте дані тестового розкладу! Тестовий розклад заповнено з вересня по січень поточного навчального року.

ФЛ-11

День	Час навчання	Дисципліна	Тип заняття	Аудиторія	Викладач	Підгрупа
Понеділок 12.12.2022	1-ша пара	Англійська мова	Семінарські	Лінгво кабінет 1	Цибуля О.А.	ФЛ-11 (1)
		Англійська мова	Семінарські	Лінгвокабінет 2	Авраменко Ю.В.	ФЛ-11 (2)
	2-га пара	Технічний переклад	Лекції	Л-102	Возняк С.Д.	
	3-я пара	Технічний переклад	Лекції	Л-102	Возняк С.Д.	
Вівторок 13.12.2022	1-ша пара	Філософія	Семінарські	Л-102	Бегаль Л.В.	
	2-га пара	Теорія перекладу	Лекції	П-103	Петренко В.В.	
	3-я пара	Англійська мова	Семінарські	Лінгво кабінет 1	Цибуля О.А.	ФЛ-11 (1)
		Англійська мова	Семінарські	Лінгвокабінет 2	Авраменко Ю.В.	ФЛ-11 (2)
	4-а пара	Ділова українська мова	Лекції	100	Цибуля О.А.	
Середа 14.12.2022	1-ша пара	Закономірності розвитку мовного апарату людини	Лекції	100	Сиротян Г.Г.	
	2-га пара	Вища математика	Лекції	305-б	Шевченко О.А.	
	3-я пара	Вища математика	Практичні	305-б	Шевченко О.А.	
	4-а пара	Походження мов	Лекції	Л-102	Возняк С.Д.	
П'ятниця 16.12.2022	1-ша пара	Філософія	Лекції	100	Бегаль Л.В.	
	2-га пара	ОБЖД	Лекції	100	Симоник О.П.	
	3-я пара	Англійська мова	Семінарські	Лінгво кабінет 1	Цибуля О.А.	ФЛ-11 (1)
		Англійська мова	Семінарські	Лінгвокабінет 2	Авраменко Ю.В.	ФЛ-11 (2)

Рисунок 1.1 – Демо версія програми з розкладом навчальних груп



Розклад

Виберіть співробітника та дату або діапазон дат для перегляду розкладу

Факультет: Факультет лінгвістики Кафедра: Теорії і практики англійськ

Співробітник: Авраменко Юрій Віталійов

Дата від: 01.12.2022 Дата по: 08.12.2022

Вивести

День	Час навчання	Дисципліна	Тип заняття	Аудиторія	Група
Четвер 01.12.2022	3-я пара	Філософія	Семінарські	Л-101	ПМ-12
	4-а пара	Філософія	Семінарські	Л-101	ПМ-13
П'ятниця 02.12.2022	3-я пара	Англійська мова	Семінарські	Лінгвокабінет 2	ФЛ-11 (ФЛ-11 (2))
Субота 03.12.2022	1-ша пара	Технічний переклад	Семінарські	П-103	ФЛ-12
Понеділок 05.12.2022	1-ша пара	Англійська мова	Семінарські	Лінгвокабінет 2	ФЛ-11 (ФЛ-11 (2))
Вівторок 06.12.2022	3-я пара	Англійська мова	Семінарські	Лінгвокабінет 2	ФЛ-11 (ФЛ-11 (2))
Четвер 08.12.2022	3-я пара	Філософія	Семінарські	Л-101	ПМ-12
	4-а пара	Філософія	Семінарські	Л-101	ПМ-13

Рисунок 1.2 – Демо версія програми з розкладом викладача

SCHOOLZEN – це сучасна хмарна система управління студентською інформацією, яка об’єднує всі ключові процеси навчального закладу в

єдиному цифровому середовищі. Вона дозволяє зберігати та оновлювати профілі учнів, включно з особистими даними, контактами та академічними записами, забезпечуючи прозорість і доступність інформації як для адміністрації, так і для батьків. Система автоматизує облік відвідуваності, дає змогу швидко відстежувати пропуски та формувати аналітику щодо ризикових студентів. Оцінювання та звітність також інтегровані в платформу: викладачі можуть виставляти оцінки онлайн, формувати табелі успішності та генерувати звіти, що допомагає адміністрації приймати рішення на основі даних. Окремою перевагою SCHOOLZEN є комунікаційний модуль, який забезпечує постійний зв'язок між школою, студентами та батьками через портал і мобільний застосунок. Батьки отримують повідомлення про оцінки, відвідуваність та важливі події, а студенти мають доступ до розкладу занять і навчальних матеріалів. Крім того, система підтримує онлайн-зарахування, що значно спрощує процес вступу, а також управління класами, викладачами та навчальними планами. Завдяки хмарній архітектурі SCHOOLZEN забезпечує доступ до даних у будь-який час і з будь-якого місця, зменшує паперову роботу та підвищує ефективність адміністративних процесів (див. рис.1.3)

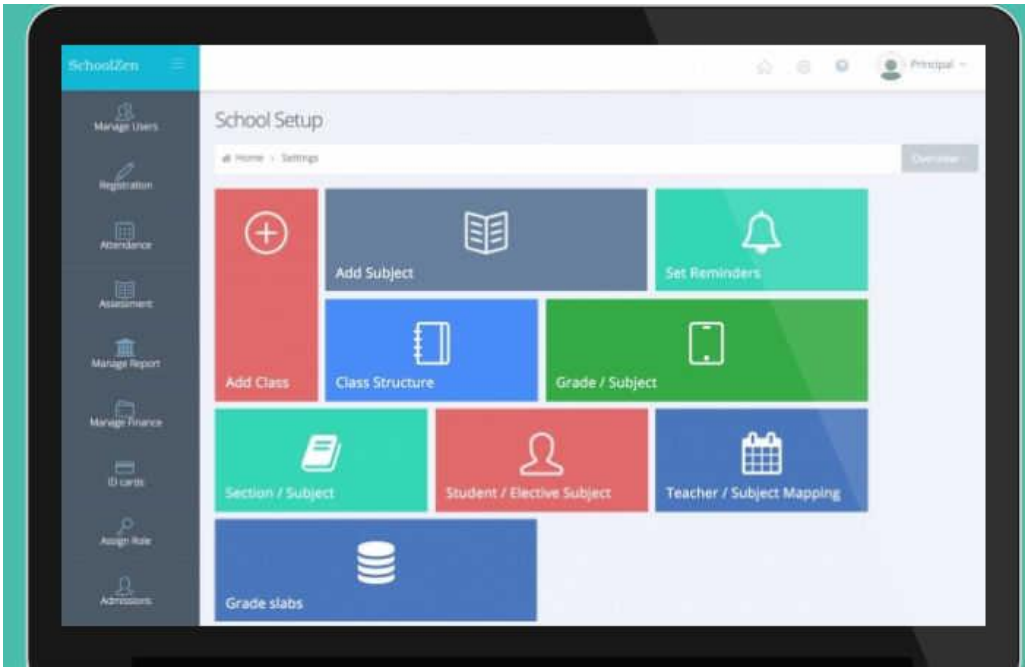


Рисунок 1.3 – Інтерфейс студентської інформаційної системи SchoolZen

Інформаційна система, представлена на сайті [12], є прикладом базового програмного продукту, створеного мовою Visual Basic для управління даними студентів. Вона виконує роль простої бази даних, у якій можна зберігати особисту інформацію студентів, редагувати її та видаляти записи. Система має інтерфейс із таблицею, де відображаються всі внесені дані, а також передбачає можливість додавання нових студентів через відповідні форми. Автор проєкту надав вихідний код, що робить його корисним для навчання програмуванню та демонстрації принципів роботи з CRUD-операціями – створення, читання, оновлення та видалення даних/ Вона не містить складних модулів, таких як управління відвідуваністю, оцінювання чи формування звітності, а також не має серйозних механізмів захисту даних – навіть логін і пароль за замовчуванням встановлені як «admin/admin». Це свідчить про те, що система не призначена для використання у реальних умовах, де потрібні високі стандарти безпеки, аудит та контроль доступу (див. рис. 1.4)

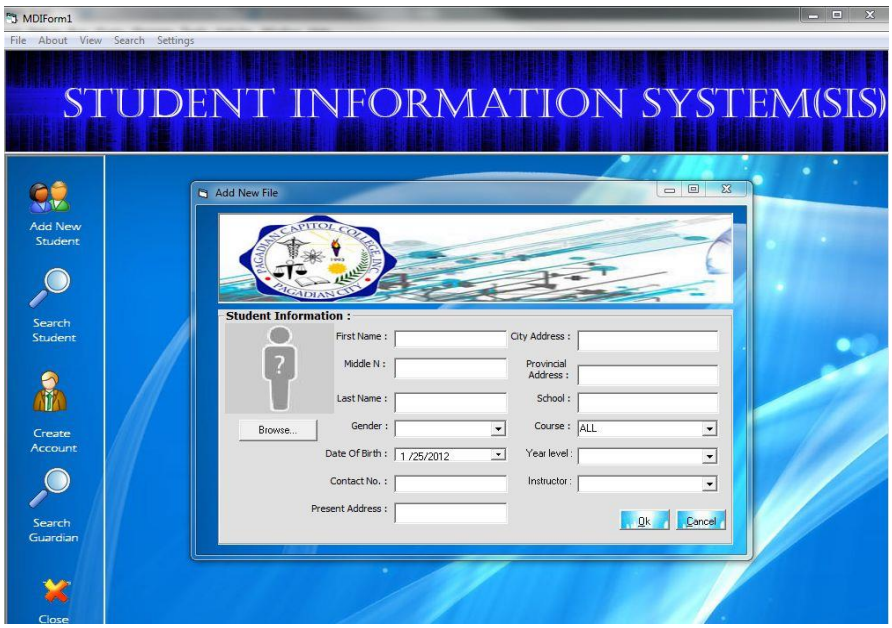


Рисунок 1.4 – Інтерфейс студентської інформаційної системи SourceCodester

Проєкт Student Management System, розміщений [13], є прикладом простої інформаційної системи для управління студентами, створеної мовою Python із використанням бібліотеки Tkinter для графічного інтерфейсу. Система працює з локальною базою даних SQLite, що дозволяє зберігати інформацію про студентів у файлі student.db. Основний скрипт student.py реалізує логіку програми та забезпечує взаємодію користувача з базою даних через віконний інтерфейс. Функціонал системи охоплює базові операції з даними: користувач може додавати нових студентів, редагувати існуючі записи, видаляти їх або здійснювати пошук за різними параметрами, такими як серійний номер, ім’я чи клас. Таким чином, система реалізує стандартні CRUD-операції (Create, Read, Update, Delete), які є основою будь-якої інформаційної системи. Разом із тим проєкт має навчальний характер і не є завершеним комерційним продуктом. Автор зазначає, що планує розширити його можливості, зокрема додати функції хмарної синхронізації та покращити інтерфейс. Наразі система не містить модулів для управління відвідуваністю, оцінками чи формування звітності, а також не має механізмів безпеки та контролю доступу (див. рис. 1.5)

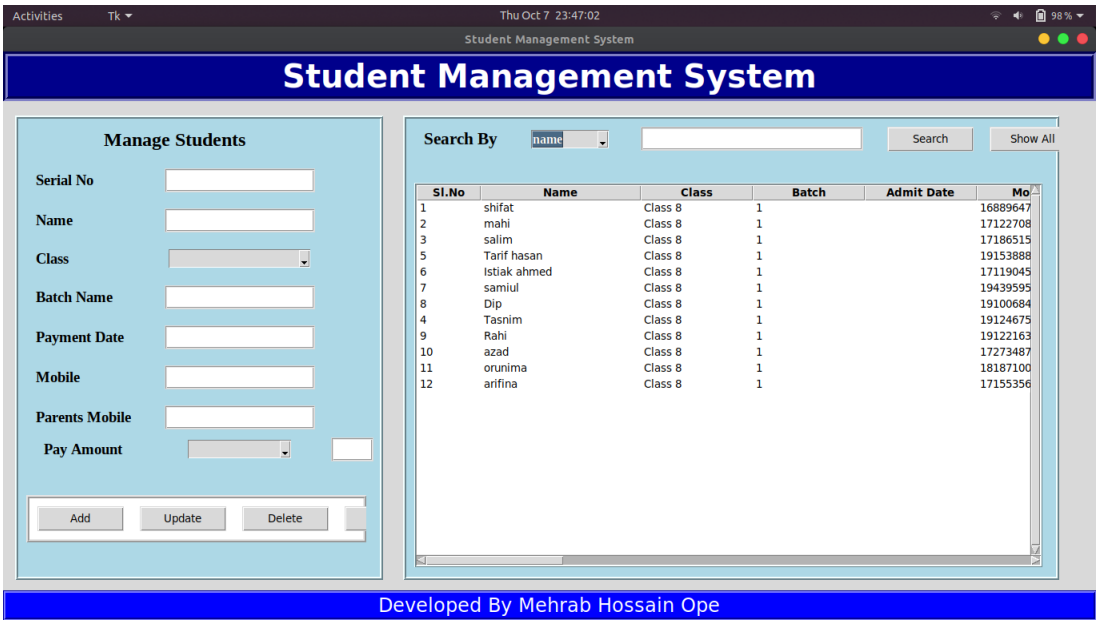


Рисунок 1.5 – Інтерфейс студентської інформаційної системи Student-management-system

Таким чином, аналіз існуючих підходів та огляд наукових праць демонструє наявність «дослідницького розриву». Попри наявність загальних досліджень з автоматизації ЗВО (сфокусованих на навчальному процесі) та захисту даних (сфокусованих на загальних принципах), вузькоспеціалізована задача військового обліку з її унікальними вимогами до звітності, специфічної бізнес-логіки та багатошарової безпеки (аудит, рольова модель, захист від атак) залишається недостатньо висвітленою.

Це підтверджує актуальність даної роботи та обґрунтовує необхідність розробки не просто ще однієї системи обліку, а спеціалізованого, захищеного програмного комплексу, орієнтованого саме на вирішення завдань військового обліку студентів. Це створює проблему дослідження: відсутність спеціалізованого, захищеного та ефективного інструменту для ведення військового обліку студентів, яку і покликана вирішити дана кваліфікаційна робота.

1.3 Постановка задачі та формулювання вимог до системи

Виходячи з виявленої проблеми, метою даної кваліфікаційної роботи є розробка та реалізація інформаційної системи у вигляді вебзастосунку для автоматизації, підвищення ефективності та безпеки процесу ведення військового обліку студентів у ЗВО.

Призначенням розроблюваної системи є повна заміна існуючих небезпечних та неефективних методів обліку (Excel, паперові журнали) та створення централізованого, надійного та контрольованого джерела інформації про студентський контингент, що підлягає військовому обліку.

Для досягнення поставленої мети, задача дослідження полягає у розробці системи, що відповідає наступному набору функціональних та нефункціональних вимог.

Система повинна забезпечувати повний цикл управління даними студентів. Це, по-перше, базові операції CRUD (Create, Read, Update, Delete) – додавання нових студентів, редагування існуючих, фільтрація та пошук за ПІБ, а також спеціалізована операція переміщення студентів у статуси «відрахований», «виключений» або «військовослужбовець».

Другим ключовим блоком вимог є глибока інтеграція з Excel. Система не повинна повністю ізолювати користувачів від звичного інструменту, а має використовувати його як транспортний механізм. По-перше, це експорт повного реєстру даних у файл military_registry.xlsx з автоматичним форматуванням заголовків та розподілом студентів на п’ять логічних аркушів: «Додаток 5 Основа» (активні студенти), «Військовозобов’язані», «Призовники», «Відраховані» та «Виключені». По-друге, це імпорт даних з аркуша «Додаток 5 Основа», причому система повинна інтелектуально обробляти рядки: шукати існуючого студента за ПІБ та датою народження для оновлення даних, або створювати нового, якщо збігу не знайдено. По-третє, це окрема функція завантаження списків студентів (бакалаврів та магістрів) для автоматичного визначення факультету та статусу навчання на основі даних про структурний підрозділ та наказ про зарахування.

Третій блок вимог – це автоматизована звітність та аналітика. Система повинна автоматично, на основі закладеної логіки, формувати «Список боржників» – студентів з неповними даними. Ця логіка має бути диференційованою, виключаючи певні поля для призовників та ігноруючи студентів з відмітками «ТОТ» та «16 років». Також система повинна надавати активну аналітику щодо призовників, які досягли або скоро досягнуть 25-річного віку. Усі ці звіти також мають експортуватися в Excel з розбивкою по факультетах.

Нефункціональні вимоги в даному проєкті мають такий самий або навіть вищий пріоритет, ніж функціональні. Перш за все, це безпека та контроль доступу. Усі сторінки, окрім входу, мають бути захищені та вимагати аутентифікації. Система повинна реалізовувати строгу рольову

модель доступу. Має бути як мінімум дві ролі: «Адміністратор» з повним доступом, включаючи управління користувачами та налаштуваннями довідників, та «Працівник» з обмеженими правами, який може додавати та редагувати студентів, але не може їх видаляти чи отримувати доступ до звітів по боржниках та призовниках 25+.

Наступною вимогою є захист від поширених вебатак. Система повинна мати вбудований захист від CSRF, XSS та Clickjacking. Окремо має бути реалізований механізм захисту від брутфорс-атак на сторінці входу, що передбачає обмеження у 5 невдалих спроб, після чого IP-адреса блокується на 15 хвилин. Також необхідний загальний механізм обмеження швидкості запитів (rate limiting).

Ключовою нефункціональною вимогою є наявність комплексної системи аудиту. По-перше, це аудит на рівні даних: кожна зміна у моделі Student має фіксуватися в окремій моделі StudentHistory. Цей запис повинен містити посилання на студента, користувача, який вніс зміни, час та JSON-поле зі змістом змін старого та нового значення. По-друге, це системний аудит: ведення окремих файлів логів security.log для подій безпеки, таке як спроби входу та блокування та audit.log для фіксації дій користувачів, таке як створення чи видалення об’єктів.

1.4 Аналіз та обґрунтування вибору методів та засобів розробки

Виходячи з деталізованих вимог, сформульованих у попередньому пункті, необхідно обрати стек технологій та методи реалізації, що найкраще забезпечать їх виконання.

Порівняння з існуючими рішеннями (п. 1.2) чітко показує, що ні електронні таблиці, ні прості СУБД, ні дорогі ERP-системи не можуть ефективно задовольнити весь спектр вимог. Єдиним виправданим підходом є

розробка кастомного вебзастосунку. Такий підхід забезпечує ідеальний баланс:

- можливість реалізувати точну бізнес-логіку: логіку «боржників» чи парсинг наказу про зарахування;
- можливість впровадити всі необхідні нефункціональні вимоги: ролі, брутфорс, аудит, що є неможливим в Excel;
- єдина база даних виступає як єдиним джерелом достовірності, вирішуючи проблеми синхронізації;
- вебінтерфейс, доступний через браузер з будь-якого авторизованого робочого місця.

Для реалізації вебзастосунку було обрано фреймворк Django на базі мови Python 3.10+. Цей вибір є стратегічно обґрунтованим, оскільки Django надає потужний інструментарій, що безпосередньо відповідає поставленим задачам.

По-перше, Django реалізує архітектурний патерн MVC (Model-View-Controller), відомий у термінології Django як MVT (Model-View-Template). Це забезпечує чітке розділення відповідальності: models.py описують структуру даних, views.py інкапсулюють бізнес-логіку, а templates відповідають за представлення.

По-друге, вбудована система ORM (Object-Relational Mapping) дозволяє описати всі моделі даних (Student, CustomUser та ін.) у вигляді класів Python. Це не лише пришвидшує розробку, але й забезпечує цілісність даних на рівні бази та абстрагує логіку від конкретної СУБД дозволяючи легкий перехід з SQLite на PostgreSQL при розгортанні.

По-третє, Django має потужний набір batteries included, що критично важливо для реалізації нефункціональних вимог. Це, зокрема, вбудована система аутентифікації та авторизації, яка служить надійною основою для кастомної моделі CustomUser з ролями. Також Django надає захист від поширених атак (CSRF, XSS, Clickjacking), що автоматично покриває значну частину вимог безпеки.

Екосистема Python надає високоякісні бібліотеки для вирішення специфічних завдань. Для реалізації модуля інтеграції з Excel було обрано бібліотеку openpyxl, оскільки вона дозволяє працювати з форматом файлів .xlsx напряму, читати, записувати та, що важливо, застосовувати форматування, такі як: синій фон та білий текст для заголовків , як того вимагає задача експорту.

Для реалізації захисту від брутфорсу та rate limiting було обрано метод з використанням вбудованого механізму кешування Django (LocMemCache) для відстеження кількості спроб входу та запитів з конкретних IP-адрес. Це ефективний та легкий у реалізації підхід.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА СТРУКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Проєктування загальної архітектури системи (MVT)

Розробка сучасної інформаційної системи для управління даними військового обліку вимагає виваженого підходу до вибору архітектурного рішення. Обрана архітектура повинна не лише задовольняти поточні функціональні вимоги, такі як надійність зберігання даних та розмежування прав доступу, але й забезпечувати гнучкість для майбутнього масштабування, зручність підтримки коду та високий рівень безпеки.

Для реалізації системи було обрано архітектурний патерн, який у середовищі веброзробки відомий як MVC (Model-View-Controller), але в реалізації фреймворку Django, що використовується в даній роботі, він трансформується у специфічну варіацію – MVT (Model-View-Template). Цей патерн забезпечує чітке розмежування логіки роботи з даними, бізнес-логіки обробки запитів та логіки відображення інтерфейсу користувача.

Архітектура системи побудована на базі фреймворку Django 5.0, який нав’язує розробнику певну структуру проєкту. Основна ідея патерну MVT полягає у розділенні відповідальності між трьома ключовими компонентами: Моделлю, Представленням (Контролером) та Шаблоном. Такий підхід дозволяє змінювати один компонент, наприклад, дизайн інтерфейсу, без необхідності втручання в інші, таку як структуру бази даних.

Розглянемо детально кожен із компонентів архітектури розробленої системи:

Модель (Model) – Шар даних

Модель є єдиним джерелом істини про дані у системі. Вона містить суттєві поля та поведінку даних, що зберігаються. У контексті даної роботи

шар Моделі відповідає на етапі розробки за взаємодію з базою даних SQLite або іншими реляційними СУБД.

У проєкті моделі описані у файлі registry/models.py, повний код модуля представлений у додатку А. Використання Django ORM (Object-Relational Mapping) дозволяє описувати структури даних як класи Python, що значно спрощує маніпуляції з даними та підвищує безпеку, автоматично екрануючи SQL-запити. Ключовими елементами цього шару є:

– визначення типів полів (текстові, числові, дати), їх параметрів (максимальна довжина, обов'язковість) та зв'язків між таблицями (ForeignKey, ManyToMany);
– моделі містять методи, що працюють безпосередньо з даними об'єкта. Модель Student містить метод is_debtor(), який інкапсулює складну логіку перевірки повноти даних студента, та метод get_age() для розрахунку віку. Це дозволяє уникнути дублювання коду в контролерах.

Представлення (View) – Шар бізнес-логіки

У термінології Django «View» виконує роль «Контролера» у класичному MVC. Це шар, який обробляє вхідні HTTP-запити, взаємодіє з моделями для отримання або збереження даних та обирає відповідний шаблон для формування відповіді користувачу.

В архітектурі системи цей шар реалізовано у файлі registry/views.py, Повний код модуля представлений у додатку Б.

Основні функції контролерів включають:

– прийом даних від користувача з форми, додавання студента, їх перевірка та передача моделям;
– саме на рівні Views застосовуються декоратори безпеки, такі як @login_required для перевірки аутентифікації та @admin_required для перевірки ролі користувача;

– формування словника даних, які необхідно відобразити на сторінці: список студентів, відфільтрований за прізвищем, та передача цього контексту у Шаблон.

Шаблон (Template) – Шар відображення

Шаблон відповідає за те, як дані будуть представлені користувачу. У розробленій системі це HTML-файли, розташовані у директорії registry/templates/, які використовують мову шаблонів Django (DTL) для динамічної генерації контенту.

Цей шар тісно інтегрований з фронтенд-фреймворком Bootstrap 5.3.0, що забезпечує адаптивність інтерфейсу. Шаблони не містять бізнес-логіки, вони лише відображають дані, отримані від Представлення (View). Архітектура шаблонів використовує механізм наслідування: існує базовий шаблон base.html, який містить спільні елементи: навігаційне меню, підключення стилів та скриптів, а всі інші сторінки: student_form.html, index.html розширюють його.

Загальна схема функціонування розробленої архітектури базується на циклі «Запит-Відповідь» (Request-Response Cycle). Процес обробки запиту користувача в системі виглядає наступним чином:

– користувач звертається до системи за певною URL-адресою (наприклад, /student/add/). Файл конфігурації URL (registry/urls.py) аналізує цей запит і, використовуючи регулярні вирази або шляхи, визначає, яка саме функція-контролер (View) повинна його обробити;
– перед тим як запит потрапить до View, він проходить через шари Middleware – проміжного програмного забезпечення. Саме тут відбуваються перевірки безпеки: захист від CSRF-атак, перевірка сесії користувача та аутентифікація;
– контролер отримує запит. Якщо це, наприклад, додавання студента, контролер ініціалізує форму StudentForm. Якщо дані валісні, контролер звертається до Моделі Student для збереження нового

запису в базу даних. Також тут спрацьовує логіка аудиту, що створює запис у StudentHistory;

– ORM трансформує команди Python у SQL-запити, виконує їх у базі даних SQLite та повертає об’єкти Python назад у контролер;
– контролер обирає відповідний HTML-шаблон, заповнює його даними та повертає сформовану сторінку як HTTP-відповідь браузеру користувача.

Вибір фреймворку Django як основи архітектури обумовлений принципом «Batteries Included» (все необхідне в комплекті), що є критичним для створення захищеної системи обліку персональних даних.

Архітектура Django автоматично захищає від більшості поширених вебвразливостей: SQL Injection, XSS, CSRF, що дозволяє розробнику зосередитися на специфічній бізнес-логіці, а не на написанні базових механізмів захисту.

Архітектура включає автоматично згенеровану адмін-панель (django.contrib.admin), яка дозволяє керувати довідниками та користувачами без написання додаткового коду для фронтенду.

Використання ORM дозволяє при необхідності змінити СУБД з SQLite на PostgreSQL або MySQL без зміни коду проєкту, лише змінивши налаштування у файлі settings.py.

Структура проєкту, розділена на окремі додатки: registry для логіки, military_project для налаштувань, дозволяє легко додавати нові функціональні модулі в майбутньому без порушення роботи існуючої системи.

Загальну архітектуру застосунку (див. рис. 2.1)

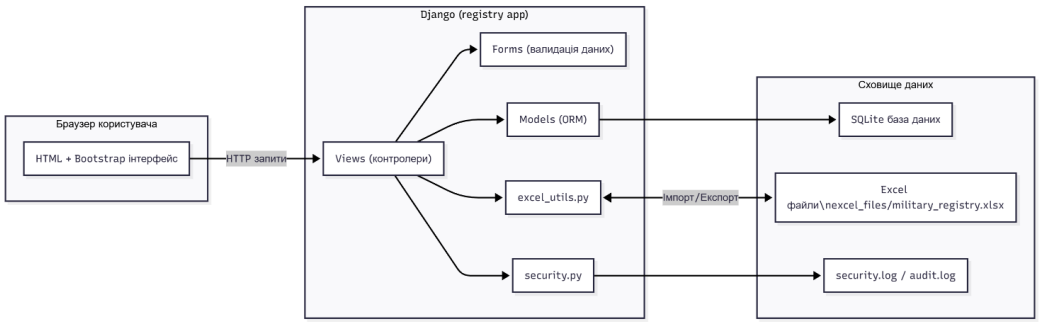


Рисунок 2.1 – Діаграма загальної архітектура застосунку

2.2 Проєктування бази даних та моделей даних

Проєктування бази даних є одним із найважливіших етапів розробки інформаційної системи, оскільки від правильності закладеної структури залежить продуктивність, масштабованість, цілісність даних та зручність подальшої підтримки програмного продукту. Враховуючи специфіку предметної області – військовий облік, де оперують великими обсягами структурованих персональних даних та суворою звітністю, – було обрано реляційну модель бази даних.

Для реалізації шару даних використано ORM (Object-Relational Mapping) фреймворку Django. Цей підхід дозволяє абстрагуватися від написання «сирих» SQL-запитів та оперувати сутностями бази даних як класами мови програмування Python. Такий підхід, відомий як «Code-First», дозволяє контролювати структуру бази даних безпосередньо через код та систему міграцій, забезпечуючи версійність змін у схемі даних.

У процесі проєктування було виділено ключові сутності предметної області, проведено їх нормалізацію для усунення надлишковості та встановлено необхідні зв’язки (один-до-багатьох, багато-до-багатьох). Розроблена схема бази даних включає шість основних таблиць, які логічно поділяються на:

– користувачі та права доступу (CustomUser);
– облік студентів (Student);

– проміжні дані (StudyingStudent);
– історія змін (StudentHistory);
– класифікатори (TerritorialCenter, CustomMark).

ER-діаграма основних моделей (див. рис. 2.2)

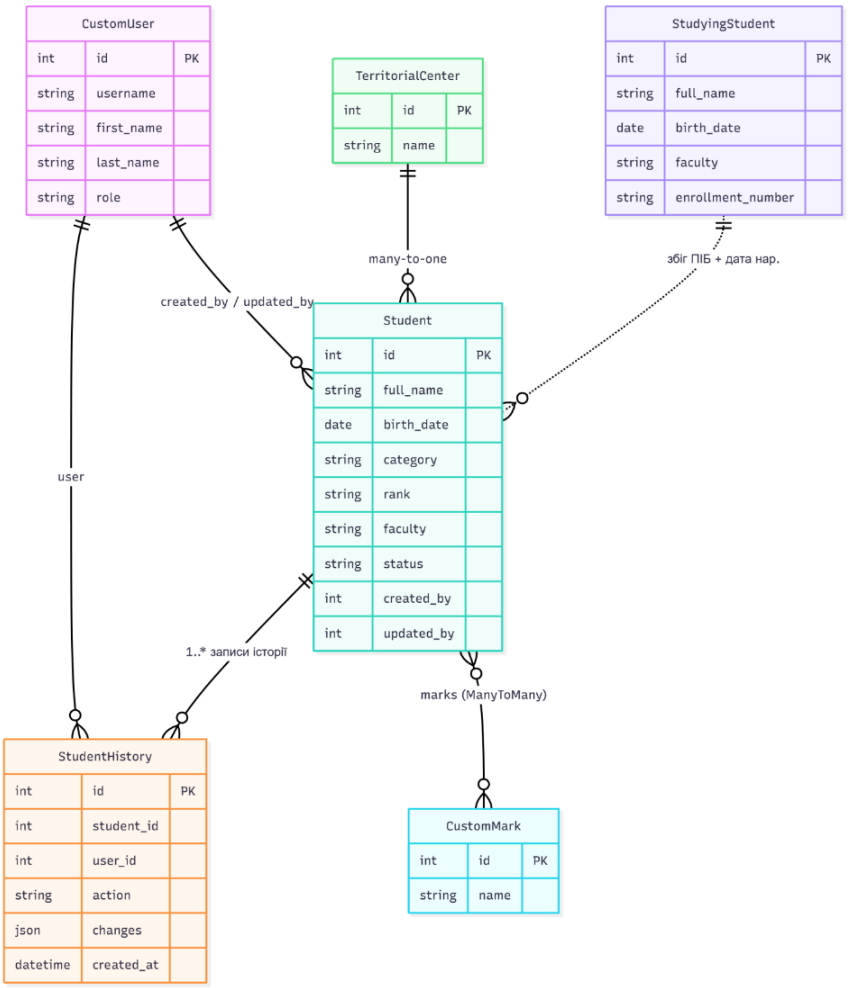


Рисунок 2.2 – ER-діаграма основних моделей

2.2.1 Модель користувача та реалізація рольової моделі

Стандартна модель користувача, яку пропонує Django (django.contrib.auth.models.User), містить базові поля для аутентифікації (логін, пароль), але є недостатньою для реалізації специфічної бізнес-логіки військового обліку, зокрема для розмежування зон відповідальності між різними типами співробітників. Тому було прийнято рішення про розробку

кастомної моделі CustomUser, яка успадковує функціонал AbstractUser, розширюючи його необхідними атрибутами.

Структура моделі CustomUser

Крім стандартних полів username (унікальний ідентифікатор), email, first_name та last_name, до моделі було додано критично важливе поле role. Поле role реалізовано як текстове поле з фіксованим набором варіантів (choices), що визначає рівень доступу користувача в системі:

– адміністратор;

– працівник.

Адміністратор – користувач з найвищим рівнем привілеїв. Проєктування передбачає, що адміністратор має доступ до налаштувань системи, управління довідниками ТЦК, перегляду логів безпеки та, що найважливіше, має право на безповоротне видалення записів про студентів. Також адміністратор має доступ до модулів аналітики боржників та призовників 25+.

Працівник – користувач з обмеженим набором прав, призначений для операційної роботи. Працівник може створювати та редагувати картки студентів, але архітектурно обмежений у правах видалення даних та доступу до конфігурації системи. Це мінімізує ризики випадкової втрати даних чи саботажу.

У модель імплементовано допоміжні методи is_admin() та is_employee(), які повертають булеві значення. Ці методи використовуються у контролерах (views) та шаблонах для умовного рендерингу елементів інтерфейсу, наприклад, приховування кнопки «Налаштування» для працівника та перевірки прав доступу через декоратори.

Таке проєктування дозволяє створити гнучку систему безпеки, де права доступу перевіряються не лише на рівні URL-адреси, а й на рівні об’єкта даних, забезпечуючи принцип найменших привілеїв.

2.2.2 Проєктування центральної моделі

Модель Student є центральним елементом системи, навколо якого будується вся бізнес-логіка. Вона спроєктована як «товста модель» (Fat Model), що означає, що вона не лише зберігає дані, але й містить методи для їх обробки та валідації.

Структура моделі включає понад 25 полів, які можна згрупувати за функціональним призначенням:.

– персональні дані;
– військово-облікові дані;
– документальний блок;
– блок відстрочок та мобілізації;
– зв’язки.

Персональні дані

Поля full_name (ПІБ) та birth_date (дата народження) є базовими ідентифікаторами особи. Вони використовуються системою для пошуку дублікатів при імпорті з Excel. Поле address зберігає місце проживання, а contact_info – телефон та email.

Військово-облікові дані

Це найбільш критичний блок. Поле category визначає статус особи (Військовозобов’язаний, Призовник, Резервіст), що впливає на правила валідації інших полів, наприклад, для призовників не вимагається звання. Поле rank реалізовано як вибір з 17 варіантів військових звань від Рекрута до Полковника. Також сюди входять унікальні ідентифікатори registry_number (ЄДРПВР).

Документальний блок

Поля military_document та passport_data призначені для зберігання реквізитів відповідних документів. Використання текстового формату дозволяє вносити серію, номер та орган видачі в одному полі, що спрощує інтерфейс для оператора.

Блок відстрочок та мобілізації

Включає дати deferment_date (відстрочка), booking_date (бронювання) та прапорці deferment_until_mobilization, mobilization_order. Ці поля є ключовими для формування звітності перед ТЦК.

Зв'язки

territorial_center: реалізовано як ForeignKey до моделі TerritorialCenter. Це нормалізує базу даних, унеможливлюючи введення різних варіацій назви одного й того ж.

marks: реалізовано як ManyToManyField до моделі CustomMark. Це дозволяє призначати студенту довільну кількість специфічних міток без зміни структури таблиці.

created_by та updated_by: поля зв'язку з CustomUser, що автоматично фіксують автора створення та останньої редакції запису.

Вбудована бізнес-логіка, що є моделлю, містить метод is_debtor(), який виконує складну перевірку заповненості полів. Алгоритм цього методу адаптивний: він перевіряє різні набори полів залежно від категорії студента, повертаючи список конкретних «боргів». Також модель містить метод get_age(), необхідний для автоматичного виявлення призовників, які досягли 25-річного віку.

2.2.3 Проєктування моделі аудиту

Для забезпечення вимог безпеки та підзвітності, які були визначені як критичні, було спроєктовано окрему модель StudentHistory. На відміну від простих текстових логів, ця модель зберігає структуровану історію змін у базі даних, що дозволяє відображати її безпосередньо в інтерфейсі системи.

Структура моделі:

student: Зовнішній ключ (ForeignKey) на модель Student. Забезпечує прив'язку історії до конкретної особи. При видаленні студента історія може

зберігатися або видалятися каскадно (в залежності від налаштувань on_delete).

user: Зовнішній ключ на CustomUser. Фіксує, хто саме виконав дію. Це критично для розслідування інцидентів некоректного введення даних.

action: Текстове поле, що позначає тип події (Створення, Оновлення, Видалення).

created_at: Часова мітка (Timestamp) моменту внесення змін.

changes: Поле типу JSONField. Це ключове архітектурне рішення. Замість створення окремих колонок для кожного поля, що змінилося, що призвело б до громіздкої та негнучкої структури, всі зміни зберігаються у форматі JSON (див. рис. 2.3).

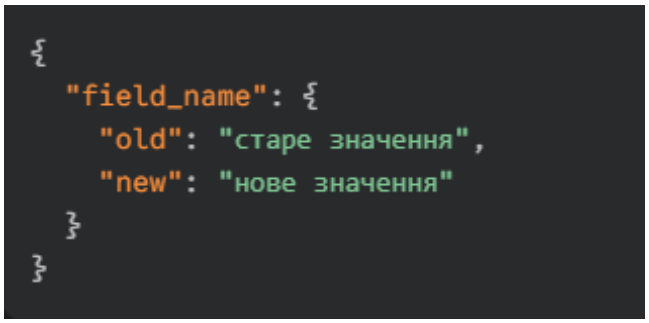


Рисунок 2.3 – Структура демонстрації в історії редагування даних

Така структура дозволяє зберігати зміни будь-яких полів, навіть якщо структура основної таблиці Student зміниться в майбутньому, забезпечуючи гнучкість системи аудиту.

Логіка заповнення цієї моделі реалізована на рівні контролера (views.py), який перед збереженням форми порівнює старий стан об’єкта з новим, формує JSON-об’єкт змін і створює запис у StudentHistory лише якщо зміни фактично відбулися.

2.2.4 Проєктування моделі імпорту

Однією з функціональних вимог є інтеграція з даними ЄДЕБО, які часто надходять у форматі неструктурованих Excel-файлів. Щоб уникнути «забруднення» основної бази даних Student потенційно некоректними або неповними даними, було спроєктовано проміжну модель (Staging Table) – StudyingStudent.

Призначення та структура

Ця модель слугує буфером для даних, завантажених із списків «Бакалаври» та «Магістри». Вона містить поля, специфічні для навчального процесу, які відсутні або мають інший формат у військовому обліку:

- enrollment_code, enrollment_number, enrollment_date: розпарсені реквізити наказу про зарахування;
- faculty: Поле, яке заповнюється автоматично на основі аналізу структурного підрозділу та номера наказу;
- degree_level: рівень освіти: бакалавр, або магістр

Логіка взаємодії

Ця модель не редагується вручну. Вона наповнюється через модуль імпорту Excel. При основному імпорті даних у модель Student, система виконує запит до StudyingStudent, шукаючи збіг за ПІБ та датою народження. Якщо запис знайдено, система автоматично підтягує з цієї проміжної моделі правильний факультет та встановлює статус «Навчається».

Таке архітектурне розділення основних даних (Student) та даних для імпорту (StudyingStudent) дозволяє підтримувати чистоту основної бази, забезпечує коректну обробку дублікатів,коли студент навчається на двох спеціальностях та автоматизує рутинний процес визначення факультету.

2.3 Проєктування підсистеми безпеки

Враховуючи специфіку предметної області – роботу з персональними даними призовників та військовозобов’язаних, підсистема безпеки є одним із

критично важливих компонентів розроблюваної інформаційної системи. Захист даних реалізовано на кількох рівнях: від мережевого (блокування IP) до прикладного (аутентифікація, перевірка прав доступу).

Проєктування системи безпеки базується на принципах «Defense in Depth» (глибокий ешелонований захист), що передбачає створення декількох незалежних шарів захисту, які перекривають один одного.

2.3.1 Проєктування механізмів аутентифікації та сесій

Першим кororком захисту є механізм аутентифікації, який гарантує, що доступ до системи отримують лише верифіковані користувачі.

Аутентифікація

Система використовує стандартний механізм аутентифікації Django, адаптований під потреби проєкту. Всі сторінки вебзастосунку, за винятком сторінки входу (/login/), захищені декоратором @login_required. Це означає, що будь-яка спроба доступу до внутрішніх ресурсів неавторизованим користувачем автоматично перенаправляє його на форму входу.

Управління сесіями

Для підтримки стану користувача після входу використовуються сесії, які зберігаються на сервері у базі даних або кеші, а клієнту передається лише ідентифікатор сесії (Session ID) у вигляді захищеного Cookie.

Проєктування параметрів сесії включає наступні налаштування безпеки:

SESSION_COOKIE_AGE

Встановлено жорсткий таймаут сесії у 30 хвилин. Якщо користувач неактивний протягом цього часу, сесія автоматично завершується, що мінімізує ризик несанкціонованого доступу, якщо працівник залишив робоче місце незаблокованим.

SESSION_SAVE_EVERY_REQUEST

Параметр встановлено в True. Це означає, що термін дії сесії оновлюється при кожному запиті. Тобто, доки користувач активно працює, його не «викине» з системи, але 30-хвилинний таймер почне відлік з моменту останньої дії.

SESSION_COOKIE_HTTPONLY

Встановлено в True. Це критично важливе налаштування, яке забороняє доступ до Cookie сесії через JavaScript. Це ефективно захищає від XSS-атак (Cross-Site Scripting), спрямованих на викрадення сесій.

2.3.2 Проєктування захисту від брутфорсу, IP-блокування та rate-limiting

Оскільки система має вебінтерфейс, вона потенційно вразлива до атак повного перебору паролів (Brute-force) та DoS-атак (Denial of Service). Для протидії цим загрозам спроєктовано спеціалізований модуль захисту, реалізований у файлі security.py, повний код модуля представлений у додатку Г.

Захист від брутфорсу (Brute-force Protection)

Алгоритм захисту працює наступним чином:

- система відстежує кількість невдалих спроб входу для кожної IP-адреси, використовуючи швидкий кеш в пам’яті (LocMemCache);
- встановлено ліміт у 5 невдалих спроб;
- при досягненні цього ліміту IP-адреса автоматично блокується на 15 хвилин. Будь-які подальші запити з цієї адреси відхиляються ще до перевірки пароля;
- після успішного входу лічильник невдалих спроб для даної IP скидається (reset_login_attempts).

Обмеження швидкості запитів (Rate Limiting)

Для захисту від автоматизованих скриптів та перевантаження сервера впроваджено механізм Rate Limiting:

– максимум 30 запитів за 60 секунд з однієї IP-адреси;
– функція check_rate_limit() перевіряє кожен вхідний запит. Якщо ліміт перевищено, система повертає помилку та тимчасово блокує доступ;

Це рішення дозволяє ефективно відсіювати ботів та сканери вразливостей, не впливаючи на роботу легітимних користувачів, які фізично не можуть генерувати запити з такою частотою.

Діаграму захисту авторизації та rate limiting (див. рис. 2.4)

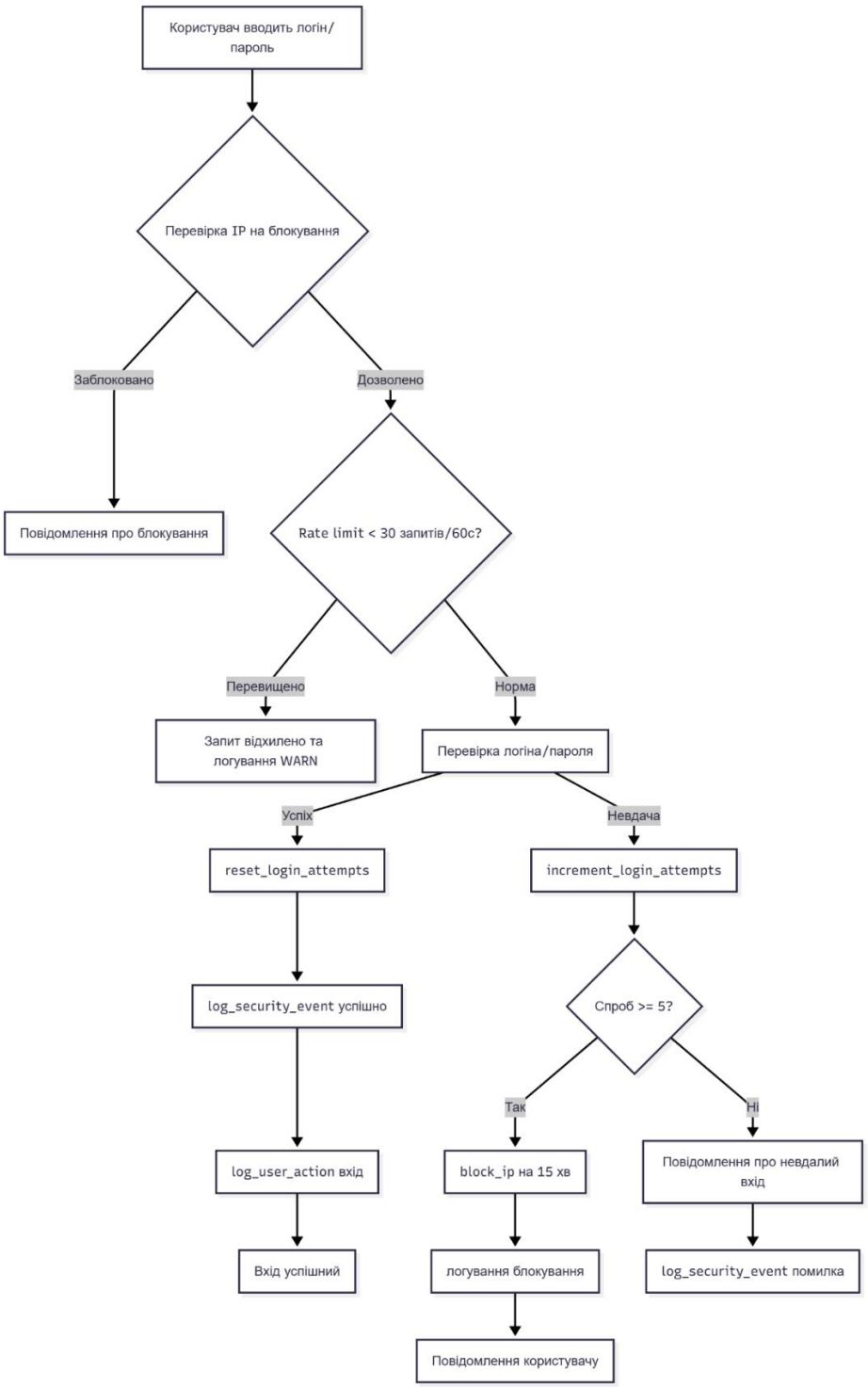


Рисунок 2.4 – Діаграма захисту авторизації та rate limiting

2.3.3 Проєктування системи аудит-логування

Окрім превентивних заходів захисту, система безпеки повинна забезпечувати можливість розслідування інцидентів та моніторингу активності. Для цього спроєктовано дворівневу систему логування.

Логи безпеки (Security Log)

В окремий файл security.log записуються події, пов’язані з технічною безпекою системи:

– успішні та невдалі спроби входу;
– факти блокування IP-адрес через брутфорс;
– перевищення лімітів запитів (Rate limit exceeded);
– підозріла активність.

Логи дій користувачів (Audit Log)

Файл audit.log фіксує бізнес-операції користувачів у системі:

– створення, редагування або видалення карток студентів;
– зміни в облікових записах користувачів;
– вхід та вихід із системи.

Кожен запис у логах має стандартизований формат: TIMESTAMP – EVENT_TYPE – User: username (Full Name), IP: ip_address, Details: details. Це дозволяє адміністратору точно встановити: хто зробив дію, коли вона відбулася, з якої IP-адреси та що саме було зроблено.

Окрім текстових логів, спроєктовано механізм глибокого аудиту на рівні бази даних через модель StudentHistory. На відміну від audit.log, який фіксує лише факт дії («Користувач X відредагував студента Y»), StudentHistory зберігає точний знімок змін у форматі JSON («Користувач X змінив поле ‘Звання’ з ‘Солдат’ на ‘Сержант’»). Це забезпечує повну прозорість та можливість відновлення історичного стану даних.

2.4 Проєктування форм та логіки валідації

Інтерфейс введення даних є критичною точкою будь-якої інформаційної системи, оскільки саме тут виникає найбільший ризик порушення цілісності даних через людський фактор («garbage in, garbage out»). Для мінімізації цих ризиків у системі спроєктовано шар форм, побудований на базі модуля django.forms.

Проєктування форм у даній системі виконує три ключові функції:

– рендеринг HTML – генерація стандартизованого, адаптивного інтерфейсу введення даних;
– санітизація (Cleaning) – попередня обробка вхідних даних, наприклад, приведення дат до єдиного формату;
– валідація (Validation) – перевірка даних на відповідність бізнес-правилам та обмеженням моделі.

Форма StudentForm є найбільш складним елементом інтерфейсу системи, оскільки вона агрегує понад 20 полів, що мають різні типи даних та залежності. Вона спроєктована як ModelForm, що дозволяє автоматично генерувати поля на основі структури моделі Student, але з значною кастомізацією віджетів та логіки.

Кастомізація віджетів та UI

Для забезпечення зручності користувача («User Experience») використано специфічні віджети:

Дати

Поля типу DateInput (наприклад, дата народження, дата бронювання) використовують нативний HTML5 тип date, що активує календар браузера.

Відмітки (Marks)

Поле marks, яке в базі даних є зв’язком Many-to-Many, у формі представлено віджетом CheckboxSelectMultiple. Це дозволяє оператору швидко обирати декілька статусів (наприклад, одночасно «ВПО» та «Проходить ВЛК») простим проставлянням галочок, замість складного вибору зі списку.

Автозаповнення ТЦК та СП

Замість стандартного випадаючого списку (Select), який містив би сотні записів і був би незручним, поле tcc_name спроєктовано як текстове поле з механізмом автозаповнення (Autocomplete). При введенні назви система через AJAX-запит шукає збіги в моделі TerritorialCenter. При збереженні

форми логіка автоматично створює новий запис ТЦК, якщо такого ще не існує, або прив’язує існуючий.

Логіка залежної валідації, тому у методі clean() форми реалізовано складну бізнес-логіку, що виходить за межі простої перевірки типів даних. Ключовим аспектом є валідація залежно від категорії військового обов’язку. Система перевіряє поле category. Якщо обрано значення «Військовозобов’язаний», то поле rank (Військове звання) стає суворо обов’язковим. Якщо ж категорія «Призовник», вимога до звання знімається. Така динамічна валідація запобігає створенню логічно некоректних записів, наприклад, військовозобов’язаного без звання.

Також форма обробляє формати дат. Оскільки користувачі можуть вводити дати у звичному форматі dd.mm.yyyy, а база даних очікує yyyy-mm-dd, форми містять логіку конвертації вхідних форматів.

Для забезпечення безпеки та рольової моделі спроєктовано дві окремі форми для роботи з обліковими записами:

CustomUserCreateForm (Створення)

Ця форма використовується адміністратором для реєстрації нових працівників. Головна увага приділена валідації паролів. Форма інтегрована з системою валідації паролів Django, яка перевіряє:

- мінімальну довжину (8 символів);
- схожість з персональними даними, пароль не може бути схожим на логін чи прізвище;
- заборона використання популярних паролів типу «123456»;
- пароль не може складатися лише з цифр

CustomUserForm (Редагування)

Ця форма має відмінну логіку. Поле пароля тут є необов’язковим. Це архітектурне рішення дозволяє адміністратору змінювати роль користувача або його email без необхідності примусової зміни пароля. Валідація перевіряє збіг паролів (password та password_confirm) лише у випадку, якщо вони були введені.

Проєктування передбачає багаторівневу систему валідації:

Рівень браузера (Client-side)

Використання атрибутів HTML5 required, maxlength, type="email". Це забезпечує миттєвий зворотний зв’язок для користувача без перезавантаження сторінки.

Рівень поля форми (Field-level)

Методи clean_<field_name> виконують специфічні перевірки для конкретного поля. Наприклад, автоматичне очищення та форматування РНОКПП.

Рівень форми (Form-level)

Метод clean() перевіряє залежності між полями (cross-field validation), наприклад, відповідність категорії та звання.

Рівень моделі (Model-level)

Фінальна перевірка на рівні бази даних, наприклад, унікальність username або registry_number.

Відображення помилок

Інтеграція з Bootstrap 5 дозволяє відображати повідомлення про помилки валідації безпосередньо під відповідними полями, підсвічуючи їх червоним кольором та додаючи пояснювальний текст. Це значно знижує когнітивне навантаження на оператора при виправленні помилок введення.

Такий підхід до проєктування форм гарантує, що в базу даних потрапляє лише коректна, повна та перевірена інформація, що є фундаментом для подальшої генерації точної звітності.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОДУЛІВ СИСТЕМИ

3.1 Реалізація модуля управління даними студентів (CRUD) та історії редагування

Модуль управління даними студентів є функціональним ядром інформаційної системи. Він реалізує класичну парадигму CRUD (Create, Read, Update, Delete), адаптовану під специфічні вимоги військового обліку, та розширену механізмом глибокого аудиту змін. Програмна реалізація цього модуля зосереджена у файлах registry/views.py (контролери), registry/forms.py (валідація) та registry/models.py (дані). Повний код модуля forms.py представлений у додатку Д.

Реалізація перегляду списку студентів (Read)

Точкою входу в систему є контролер index, який відповідає за відображення реєстру студентів. Для забезпечення швидкодії при роботі з великими масивами даних реалізовано механізм пагінації (Paginator), що розбиває вивід на сторінки по 20 записів.

Алгоритм формування списку включає динамічну фільтрацію та сортування за допомогою Django ORM:

Фільтрація

Система приймає GET-параметри для фільтрації за категорією (category) та статусом навчання (is_studying). Діаграму фільтрації студентів за категорією на головній сторінці (див. рис. 3.1).

Пошук

Реалізовано пошук за входженням підрядка у поле ПІБ (full_name__icontains).

Сортування

Застосовано складну логіку впорядкування. Якщо обрано категорію «Військовозобов'язаний», список сортується за військовим званням від вищого до нижчого згідно з визначеною ієрархією. Для категорії «Призовник» застосовується алфавітне сортування за ПІБ.

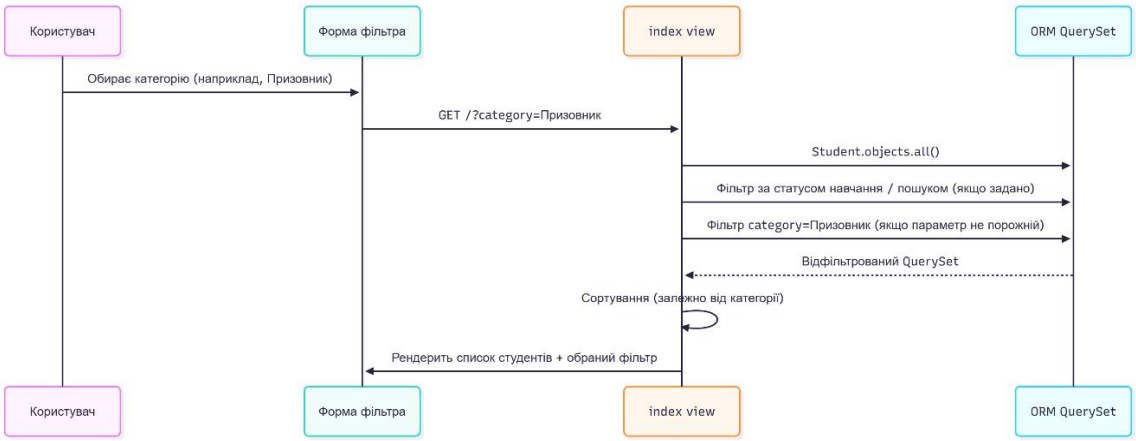


Рисунок 3.1 – Діаграма фільтрації студентів за категорією на головній сторінці

Функція додавання (Create) реалізована у контролері student_add та використовує форму StudentForm. Процес збереження даних включає кілька етапів автоматичної обробки:

Валідація форми

Виклик методу form.is_valid() запускає перевірки, описані у розділі 2, зокрема, залежність звання від категорії.

Обробка ТЦК та СП

Поле ТЦК реалізовано через текстове введення з автозаповненням. При збереженні система перевіряє наявність ТЦК та СП у моделі TerritorialCenter. Якщо такий центр існує – він прив'язується до студента, якщо ні – програмно створюється новий запис.

Прив'язка метаданих

Полю created_by автоматично присвоюється об'єкт поточного користувача (request.user), що фіксує авторство запису.

Редагування даних, реалізоване у контролері student_edit, є найбільш складним процесом з точки зору бізнес-логіки, оскільки воно поєднане з механізмом аудиту.

При відкритті сторінки редагування форма StudentForm ініціалізується даними конкретного екземпляра об’єкта (instance=student). Для зручності оператора дати автоматично конвертуються у формат, зрозумілий HTML5-віджетам (DD-MM-YYYY).

При обробці POST-запиту на збереження змін виконується наступний алгоритм:

– створюється об’єкт форми з новими даними;
– проводиться валідація;
– якщо дані валідні, запускається механізм порівняння станів для історії редагування;
– оновлюється поле updated_by посиланням на поточного користувача;
– зміни фіксуються у базі даних методом form.save().

У системі реалізовано два підходи до видалення даних, які контролюються рольовою моделлю доступу:

М’яке видалення (зміна статусу)

Для збереження історичних даних про відрахованих студентів реалізовано функцію переміщення запису в архів. Контролер дозволяє змінити статус студента на «Відрахований» (expelled), «Виключений» (excluded) або «Військовослужбовець» (serving). При виборі статусу «Відрахований» система програмно вимагає заповнення полів номера та дати наказу про відрахування.

Фізичне видалення

Повне видалення запису з бази даних доступне виключно користувачам з роллю «Адміністратор». Ця функція захищена декоратором @admin_required та вимагає додаткового підтвердження дії.

Ключовою особливістю модуля є автоматична фіксація всіх змін у моделі StudentHistory. Цей механізм реалізовано безпосередньо у контролері редагування, що дозволяє перехоплювати дані до їх збереження в БД.

Збереження та відображення: Якщо словник changes не порожній, створюється новий запис у моделі StudentHistory:

– поле changes заповнюється сформованим JSON-об’єктом;

– поле action встановлюється як «Updated»;

– фіксується час та користувач.

Така реалізація забезпечує повну прозорість дій у системі та дозволяє адміністраторам відстежувати еволюцію даних кожного студента, виявляти помилки операторів та контролювати цілісність інформації.

Діаграму процесу редагування студента (див. рис. 3.2).

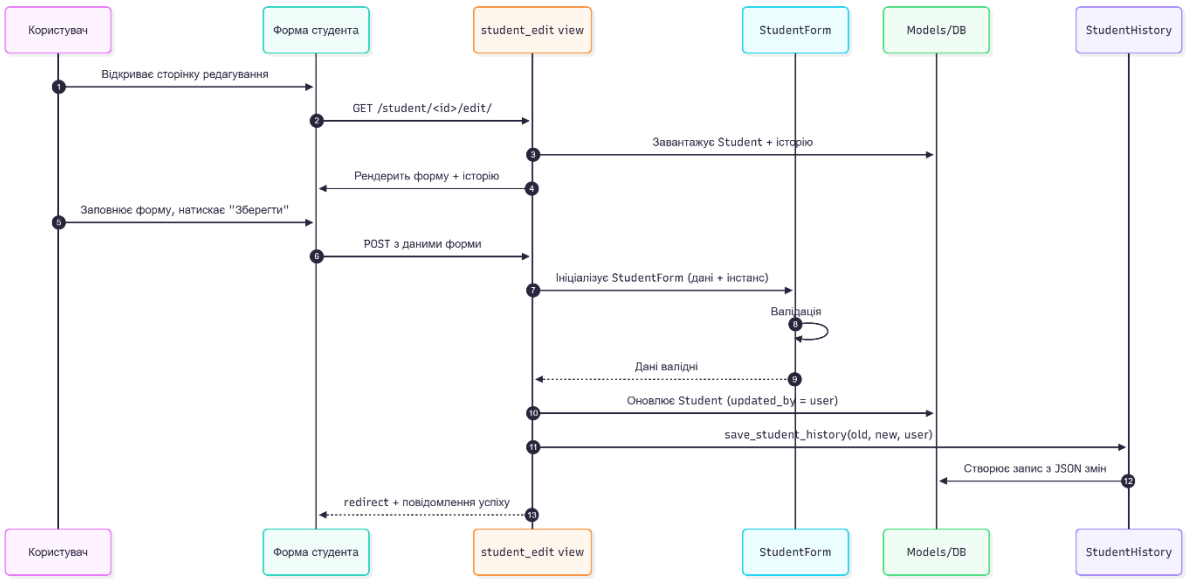


Рисунок 3.2 – Діаграма процесу редагування студента

3.2 Реалізація модуля інтеграції з Excel

Модуль інтеграції з електронними таблицями є критично важливим компонентом системи, оскільки він забезпечує сумісність з існуючими

бізнес-процесами закладу освіти та автоматизує обмін даними. Програмна реалізація цього модуля базується на використанні бібліотеки openpyxl версія 3.1.2, яка дозволяє виконувати низькорівневі операції читання та запису файлів формату .xlsx.

3.2.1 Програмна реалізація експорту даних у звітні форми

Функція експорту реалізована у файлі registry/excel_utils.py і має на меті генерацію стандартизованого файлу military_registry.xlsx, який відповідає вимогам військового обліку, повний код функції представлений у додатку В.

Алгоритм формування структури файлу

Функція export_to_excel() ініціює створення нової робочої книги (Workbook). Далі, використовуючи Django ORM, система виконує вибірку даних з бази даних, розділяючи студентів на логічні групи для формування п’яти окремих аркушів:

«Додаток 5 Основа»

Сюди потрапляють усі студенти зі статусом active.

«Військовозобов’язані»

Фільтрація за полем category (військовозобов’язані та резервісти). Особливістю є програмне сортування записів за військовим званням від вищого до нижчого.

«Призовники»

Фільтрація за категорією "Призовник" із сортуванням за алфавітом (ПІБ).

«Відраховані» та «Виключені»

Аркуші для архівних записів, що містять специфічні поля (номер та дата наказу про відрахування).

Програмний код не лише вивантажує дані, але й застосовує стилі для відповідності офіційним шаблонам звітності:

– програмно встановлюється синій колір фону (#DCE6F1), жирний шрифт та центрування тексту. Заголовки займають перші 7 рядків, а дані починаються з 8-го рядка;
– реалізовано алгоритм, який проходить по всіх колонках та адаптує їх ширину відповідно до найдовшого текстового значення в комірці, що забезпечує читабельність звіту;
– у колонці «А» автоматично генерується порядковий номер для кожного запису.

Окремо реалізовано функції export_debtors_to_excel() та export_age25_to_excel(), які генерують аналітичні звіти. Їхня особливість полягає у розбивці даних на окремі аркуші для кожного факультету, причому назва аркуша програмно обрізається до 31 символу через технічні обмеження самого Excel.

3.2.2 Програмна реалізація імпорту та синхронізації даних

Функція імпорту import_from_excel() реалізує складну логіку двосторонньої синхронізації, дозволяючи масово оновлювати базу даних через завантаження відредагованого файлу.

Алгоритм обробки даних:

а) система завантажує файл та відкриває аркуш «Додаток 5 Основа» або «Основа». Читання даних починається строго з 8-го рядка, ігноруючи «шапку» документа;
б) спеціалізована допоміжна функція зчитує дані з колонок від А до Т. На цьому етапі відбувається очищення даних (Sanitization):
   1) часто Excel автоматично форматує довгі числа як дробові (наприклад, «1234567890.0»). Програмний код перевіряє наявність суфікса «.0» та видаляє його, перетворюючи значення на рядок;

2) реалізовано конвертер, що розпізнає різні формати дат (текстові, об’єкти datetime) та приводить їх до єдиного формату бази даних (DD-MM-YYYY);
3) текстові значення відміток, розділені комами, розбиваються на список для подальшої обробки.

Логіка синхронізації (Update or Create)

Для кожного рядка система виконує пошук у базі даних за композитним ключем: ПІБ + Дата народження.

Якщо запис знайдено, тоді система оновлює всі поля існуючого об’єкта Student даними з Excel. Це дозволяє актуалізувати інформацію, наприклад, зміну адреси без дублювання записів. А якщо запис відсутній, тоді створюється новий об’єкт Student.

Обробка помилок та Backup

Для захисту від пошкодження основного файлу Excel при збоях запису, реалізовано функцію sync_db_to_excel(). Якщо під час синхронізації виникає критична помилка, система автоматично створює резервну копію (backup) пошкодженого файлу з часовою міткою у назві, після чого створює новий коректний файл із заголовками.

3.2.3 Реалізація механізму автоматичного визначення факультету

Однією з найбільш інтелектуальних функцій модуля є автоматичне визначення факультету та статусу навчання студента. Оскільки ці дані часто відсутні у військово-облікових документах, система отримує їх з альтернативного джерела – наказів про зарахування.

Eman 1 Завантаження академічних списків (upload_students_excel). Система дозволяє завантажити окремі файли списків «Бакалаври» та «Магістри», підтримуються формати .xls, .xlsx, .csv. Дані з цих файлів парсяться та зберігаються у проміжну модель StudyingStudent. Ключовим моментом є функція parse_enrollment_order(), яка за допомогою регулярних

виразів витягує з текстового рядка наказу три компоненти: код наказу, номер та дату.

Етап 2 Алгоритм визначення факультету (determine_faculty). Ця функція аналізує поле «Структурний підрозділ» з академічних списків. Реалізовано словник відповідностей, який мапить назви кафедр або інститутів до одного з 13 факультетів університету. Для вирішення проблеми дублікатів, коли студент навчається одночасно на двох спеціальностях, алгоритм враховує номер наказу про зарахування як унікальний ідентифікатор навчання.

Етап 3 Інтеграція при імпорті. Під час виконання основного імпорту, для кожного студента виконується запит до таблиці StudyingStudent.

Якщо знайдено збіг за ПІБ та датою народження:

а) поле faculty у картці студента автоматично заповнюється визначеним значенням;

б) встановлюється прапорець is_studying = True;

в) оновлюються дані про наказ про зарахування.

Така реалізація дозволяє повністю автоматизувати процес актуалізації місця навчання, усуваючи необхідність ручного введення цих даних для тисяч студентів.

Діаграму автоматичного визначення факультету при імпорті (див. рис. 3.3)

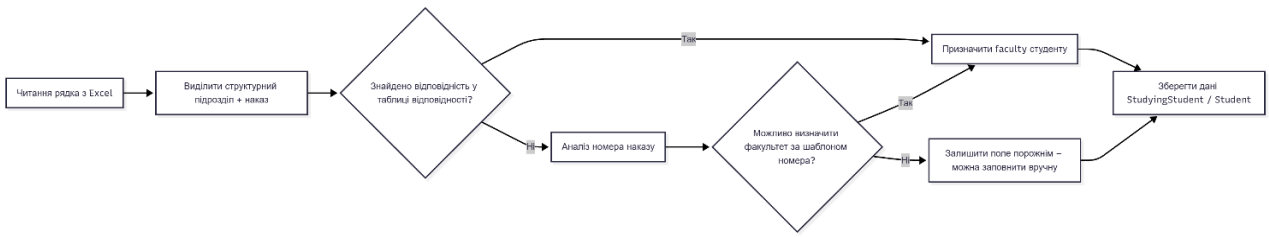


Рисунок 3.3 – Діаграма автоматичного визначення факультету при імпорті

3.3 Реалізація підсистеми звітності та аналітики

Підсистема звітності призначена для трансформації сирих даних бази даних у структуровану інформацію, необхідну для прийняття управлінських рішень та подання звітів до ТЦК та СП. На відміну від стандартних CRUD-операцій, цей модуль виконує складну алгоритмічну обробку даних, виявляючи логічні невідповідності «боржників» та відстежуючи часові тригери, таке як вік 25 років та більше.

3.3.1 Реалізація модуля «Список боржників»

Модуль «Боржники» вирішує задачу автоматичного аудиту якості даних. Його мета – виявити студентів, у картках яких відсутня обов’язкова інформація.

Алгоритм валідації повноти даних (is_debtor)

Ключова логіка реалізована не у контролері, а безпосередньо в моделі Student через метод is_debtor(). Це забезпечує інкапсуляцію логіки та дозволяє використовувати її у різних частинах системи, наприклад, підсвічувати статус боржника на сторінці редагування.

Алгоритм перевірки працює наступним чином:

а) створюється список обов’язкових полів для перевірки: ПІБ, адреса, паспортні дані тощо;

б) система перевіряє наявність у студента специфічних відміток через зв’язок ManyToMany. Якщо студент має відмітку «ТОТ» (Тимчасово окуповані території) або «16 років», метод миттєво повертає результат False (не боржник), оскільки для цих категорій відсутність документів є допустимою;

в) алгоритм розрізнює вимоги залежно від категорії військового обов’язку. Якщо категорія студента – «Призовник», із перевірки

автоматично виключається ряд полів, які є обов’язковими лише для військовозобов’язаних:

1) військове звання (Колонка C);
2) реєстраційний номер (Колонка F);
3) військово-облікова спеціальність (Колонка H);
4) дані про відстрочку (Колонка M);
5) дані про бронювання (Колонка N);
6) дані про службу (Колонка O);
7) дані про мобілізаційне розпорядження (Колонка P).

г) метод перевіряє заповненість кожного поля з фінального списку. Якщо поле порожнє, його назва додається до списку боргів. Метод повертає булеве значення статусу та список відсутніх полів.

Реалізація експорту звіту (export_debtors_to_excel)

Функція генерації Excel-звіту, реалізована в excel_utils.py, групує боржників за факультетами. Особливістю програмної реалізації є обхід технічного обмеження Excel на довжину назви аркуша (максимум 31 символ). Перед створенням аркуша назва факультету програмно обрізається: ws.title = faculty_name[:31]. Звіт містить колонки: «№», «ПІБ», «Дата народження» та «Перелік боргів» – це список відсутніх полів, об’єднаний у рядок (див. рис. 3.4).

3.3.2 Реалізація модуля «Призовники 25+»

Цей аналітичний модуль призначений для проактивного виявлення призовників, які досягли граничного віку перебування на військовому обліку призовників 25 років та підлягають переведенню у категорію військовозобов’язаних.

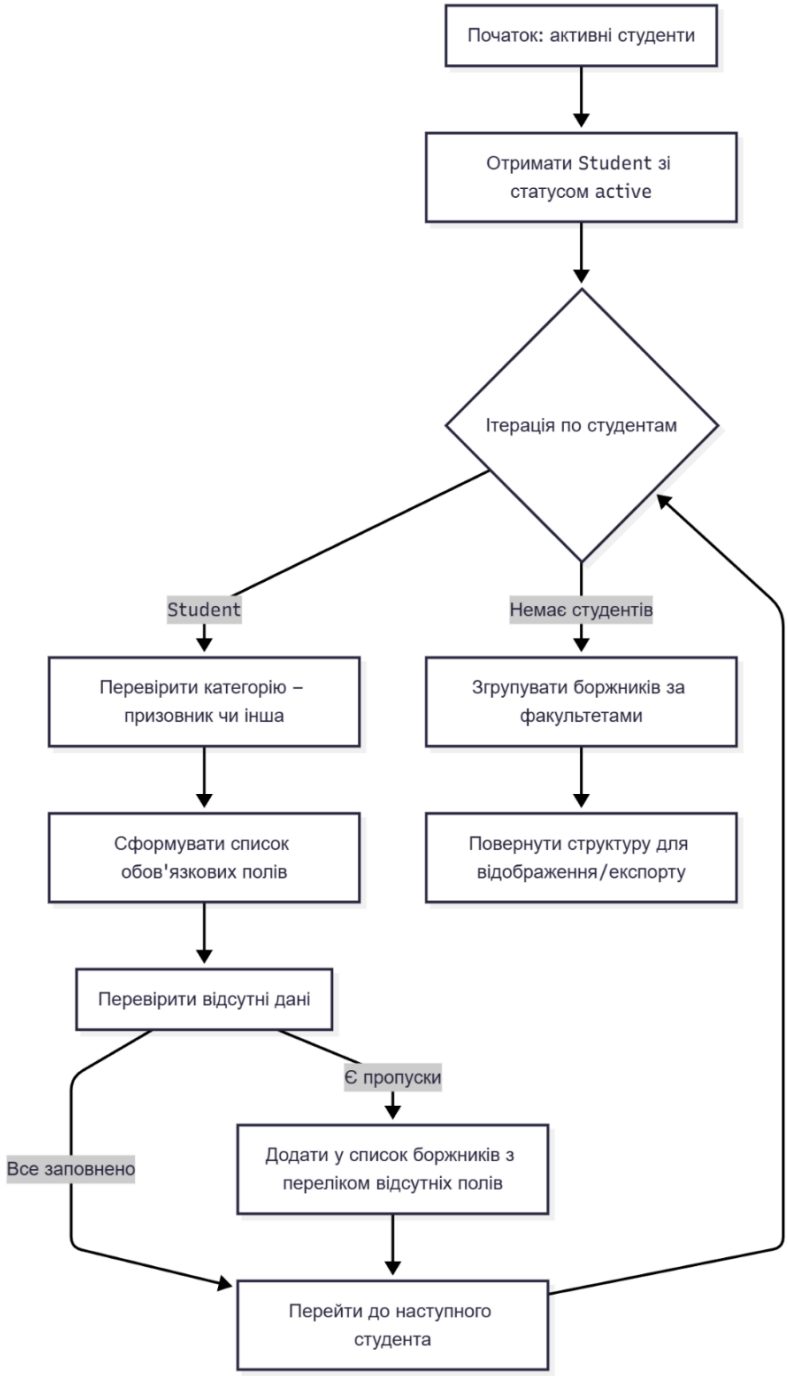


Рисунок 3.4 – Діаграма алгоритму визначення боржників

Логіка розрахунку та фільтрації (age_25_check)

Контролер age_25_check виконує вибірку студентів з бази даних, застосовуючи первинний фільтр: category=‘Призовник’. Далі для кожного студента виконується розрахунок віку за допомогою методу get_age() моделі Student, який використовує поточну дату (datetime.date.today()).

На основі розрахунку формуються два окремі списки контексту:

«Вже виповнилося 25 років»

Критична група, що потребує негайної уваги. У звіті для них відображається поточний вік.

«Скоро виповниться 25 років»

Група попередження. Для цих студентів система розраховує та відображає кількість днів, що залишилися до 25-річчя.

Реалізація експорту (export_age25_to_excel)

Функція експорту створює складний багатосторінковий документ:

– для кожного факультету створюється окремий аркуш;
– на одному аркуші дані розділені на два блоки заголовками: спочатку список тих, кому вже виповнилося 25, нижче – список тих, кому скоро виповниться;
– окрім ПІБ та дати народження, звіт містить розрахункові поля: вік, або кількість днів, що дозволяє адміністратору працювати зі звітом без додаткових калькуляцій.

Така реалізація підсистеми звітності перетворює статичну базу даних на інструмент активного моніторингу, автоматизуючи найбільш трудомісткі процеси аналізу особових справ.

Діаграму перевірки призовників 25+ років (див. рис. 3.5).

3.4 Реалізація підсистеми управління користувачами та розмежування доступу

Підсистема управління користувачами є критично важливою для забезпечення конфіденційності даних військового обліку. Її програмна реалізація включає налаштування процедур аутентифікації (входу в систему), розробку інтерфейсів для адміністрування облікових записів та імплементацію програмної логіки розмежування прав доступу на рівні серверного коду (Backend) та інтерфейсу (Frontend).

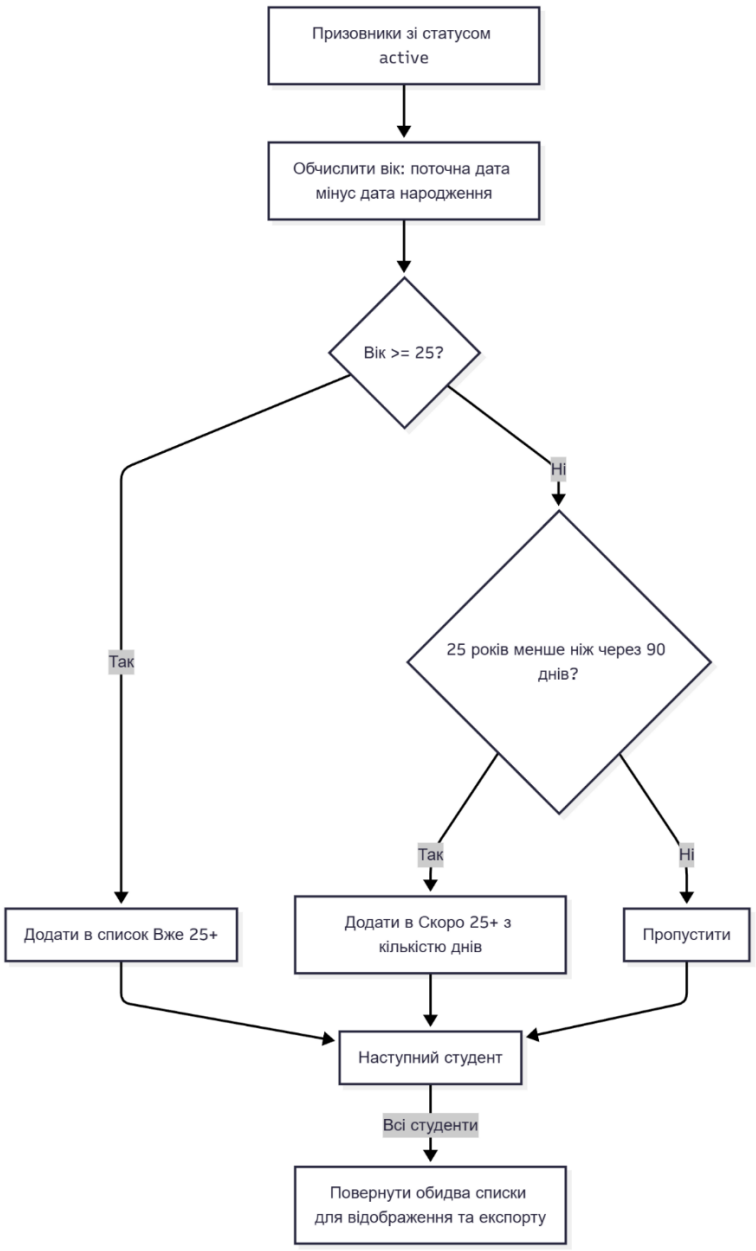


Рисунок 3.5 – Діаграма перевірка призовників 25+ років

Процес входу в систему реалізовано у контролері login_view. Хоча Django надає стандартні view для входу, у даній системі було написано кастомний контролер для інтеграції з підсистемою безпеки (захист від брутфорсу).

Алгоритм роботи контролера login_view:

– перевірка методу запиту;
– перевірка Rate Limiting;

– валідація та аутентифікація;
– логування.

Процедура виходу реалізована у функції logout_view, яка викликає стандартний метод logout() для очищення сесії користувача та перенаправляє на сторінку входу, фіксуючи подію в аудиті.

Система реалізує рольову модель (Role-Based Access Control), де права доступу визначаються полем role моделі CustomUser. Програмна реалізація розмежування доступу виконана на двох рівнях.

Обмеження на рівні контролерів (View Level)

Для захисту маршрутів, доступних тільки адміністраторам, розроблено кастомний декоратор @admin_required. Цей декоратор огортає функцію-контролер і виконує перевірку перед її запуском.

Обмеження на рівні шаблонів (Template Level)

Для забезпечення зручності інтерфейсу («User Experience») елементи, недоступні поточному користувачу, приховуються. У шаблонах, наприклад, base.html використовуються умовні конструкції Django Template Language:

– меню «Користувачі» та «Налаштування» відображається лише за умови {% if user.is_admin %};
– кнопка «Видалити студента» на сторінці редагування доступна тільки якщо користувач має відповідні права. Для цього у моделі CustomUser реалізовано властивості-хелпери is_admin та is_employee, які повертають булеві значення для спрощення.

Адміністративна частина системи надає інтерфейс для повного циклу управління користувачами (CRUD):

Перегляд списку користувачів (user_list)

Контролер отримує всіх користувачів з бази даних (CustomUser.objects.all()) та відображає їх у табличному вигляді. Таблиця містить інформацію про логін, ПІБ, роль, статус активності та дату останнього входу.

Створення користувача (user_create)

Використовує спеціалізовану форму CustomUserCreateForm. Особливістю реалізації є обробка паролів. Оскільки паролі не можна зберігати у відкритому вигляді, при збереженні форми викликається метод user.set_password(password), який хешує пароль перед записом у базу даних. Також реалізовано валідацію складності пароля (мінімум 8 символів, заборона суто числових паролів).

Редагування користувача (user_edit)

Контролер дозволяє адміністратору змінювати дані існуючого користувача, включаючи його роль та статус активності (блокування доступу). Важливим аспектом є зміна пароля. Форма CustomUserForm робить поля пароля необов'язковими. Логіка контролера перевіряє: якщо поля пароля заповнені, новий пароль хешується та зберігається; якщо порожні – пароль залишається без змін. Це дозволяє редагувати профіль користувача без необхідності скидання його пароля.

Видалення користувача (user_delete)

Реалізовано через сторінку підтвердження. Видалення користувача є критичною операцією, тому система перевіряє права доступу (@admin_required) та логує цю подію в audit.log. Важливо зазначити, що при видаленні користувача історія змін (StudentHistory), пов'язана з ним, може зберігатися (залежно від налаштувань каскадного видалення БД), щоб не втратити дані аудиту.

Така реалізація підсистеми гарантує, що лише авторизовані особи мають доступ до даних, а їхні повноваження чітко регламентовані програмним кодом, що унеможливлює обхід обмежень через інтерфейс.

3.5 Демонстрація функціональних можливостей та інтерфейсу системи

Розробка графічного інтерфейсу (GUI) системи виконувалася з дотриманням принципів ергономічності, інтуїтивної зрозумілості та адаптивності. Для реалізації клієнтської частини (Frontend) було використано CSS-фреймворк Bootstrap 5.3. Вибір цього інструментарію дозволив забезпечити коректне відображення системи на пристроях з різною роздільною здатністю екрана (від широкоформатних моніторів до смартфонів), а також стандартизувати вигляд елементів управління (кнопок, форм, таблиць, модальних вікон).

Точкою входу в систему є сторінка авторизації. Інтерфейс цієї сторінки спроєктовано мінімалістично, щоб не відволікати користувача. Центральним елементом є форма входу, яка містить поля «Логін» та «Пароль».

Важливою особливістю інтерфейсу є реалізація механізму зворотного зв’язку: у разі введення некоректних даних або блокування IP-адреси через перевищення ліміту спроб, система відображає відповідне повідомлення у червоному блоці (alert-danger) безпосередньо над формою, не перезавантажуючи сторінку повністю (див. рис. 3.6)

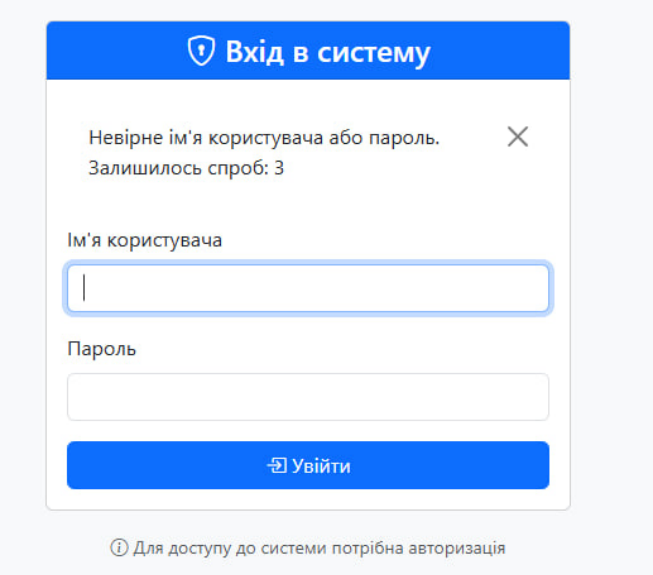


Рисунок 3.6 – Сторінка авторизації з лімітом спроб

Після успішного входу користувач отримує доступ до головного вікна системи. Навігація реалізована через бічну панель (Sidebar) або верхнє меню (Navbar), яке містить посилання на основні модулі: «Список студентів», «Боржники», «Призовники 25+» (див. рис. 3.7)

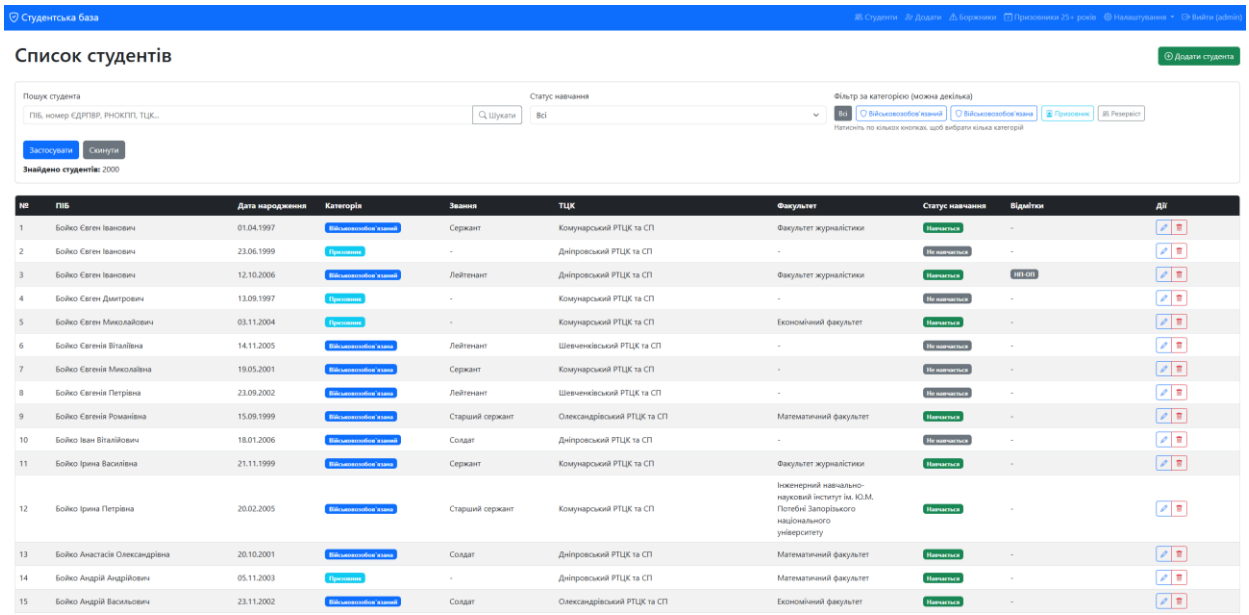


Рисунок 3.7 – Головне вікно системи

Навігаційна панель є динамічною: її вміст змінюється залежно від ролі авторизованого користувача. Завдяки використанню шаблонних тегів Django, пункти меню «Налаштування» для довідників ТЦК, відміток та «Користувачі» відображаються виключно для адміністраторів, залишаючись прихованими для звичайних працівників (див. рис. 3.8 – 3.9).

Основним робочим інструментом оператора є сторінка реєстру студентів. Дані представлені у вигляді інтерактивної таблиці з посторінковим виводом, що дозволяє зручно працювати з базою даних, яка містить тисячі записів.

У верхній частині інтерфейсу розташовано панель інструментів, що включає (див. рис. 3.10)

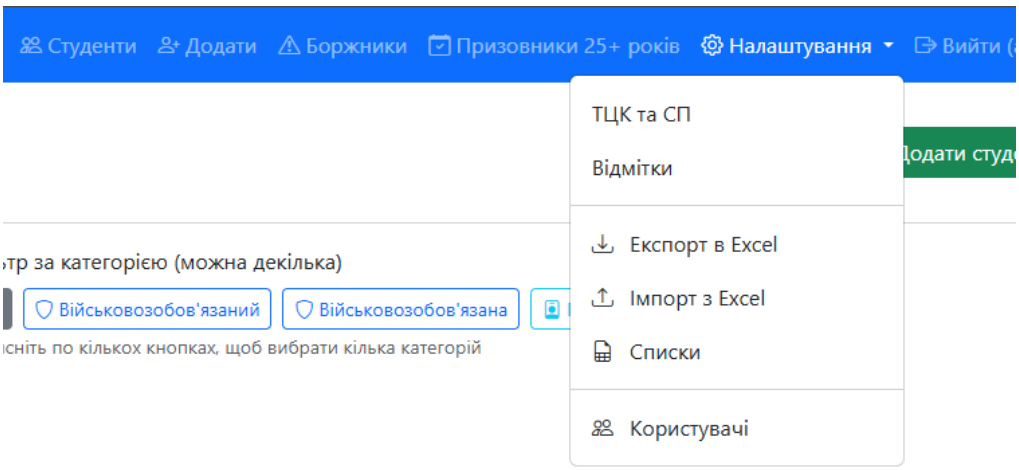


Рисунок 3.8 – Панель та меню «Налаштування» у ролі адміністратора



Рисунок 3.9 – Панель у ролі працівника

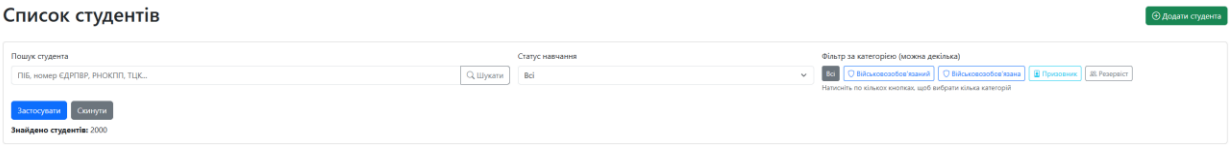


Рисунок 3.10 – Панель інструментів

Рядок пошуку

Дозволяє миттєво знаходити студента за прізвищем.

Фільтри

Випадаючі списки для фільтрації записів за категорією («Військовозобов’язаний», «Військовозобов’язана», «Резервіст», «Призовник») та статусом навчання.

Кнопки дій

Кнопки «Додати студента», яка дозволяє додати до бази нового студента.

Форма додавання та редагування студента (StudentForm) є найбільш насиченим елементом інтерфейсу. Для зручності заповнення великої

кількості полів, форму логічно структуровано за допомогою групування елементів (Персональні дані, Військовий облік, Документи, Відстрочка).

Особливу увагу приділено зручності введення специфічних даних:

Поля дат

Використовують нативний календар браузера (HTML5 date input), що унеможливлює введення дати у некоректному форматі.

Автозаповнення

Поле «ТЦК та СП» реалізовано як розумний рядок пошуку. Коли користувач починає вводити назву центру, система пропонує існуючі варіанти з бази даних, що пришвидшує введення та запобігає дублюванню назв.

Множинний вибір відміток

Блок «Відмітки» реалізовано у вигляді набору чекбоксів (Checkbox Select), що дозволяє одним кліком встановити статуси «ТОТ», «В розшуку» тощо.

Система валідації інтегрована в інтерфейс

Якщо користувач намагається зберегти форму з помилкою, наприклад, не вказав звання для військовозобов’язаного), відповідне поле підсвічується червоним кольором, а під ним з’являється текст підказки (див. рис. 3.11 – 3.12).

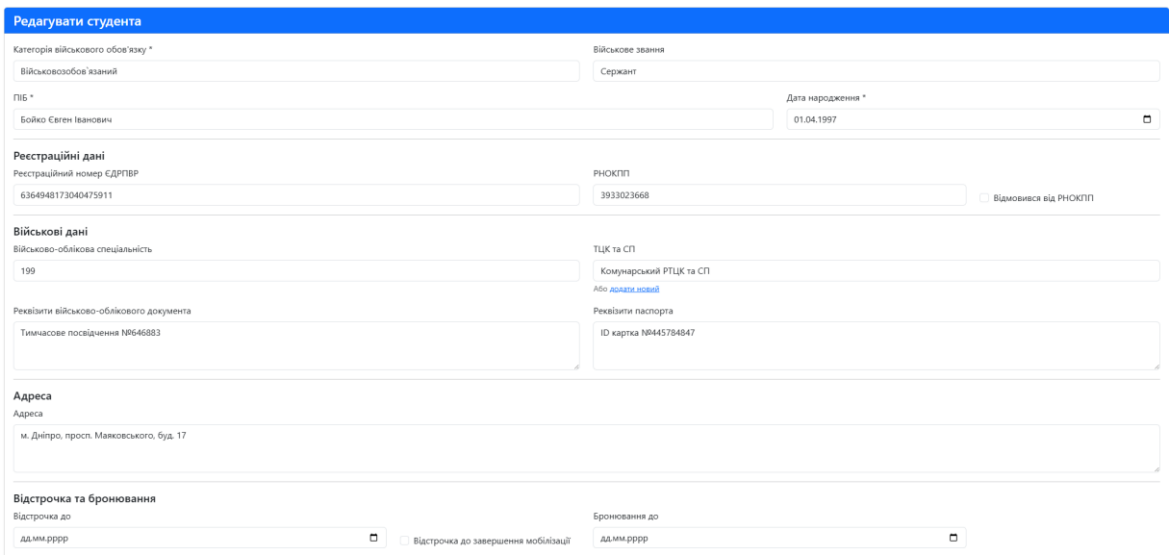


Рисунок 3.11 – Форма додавання та редагування студента (верхня частина)

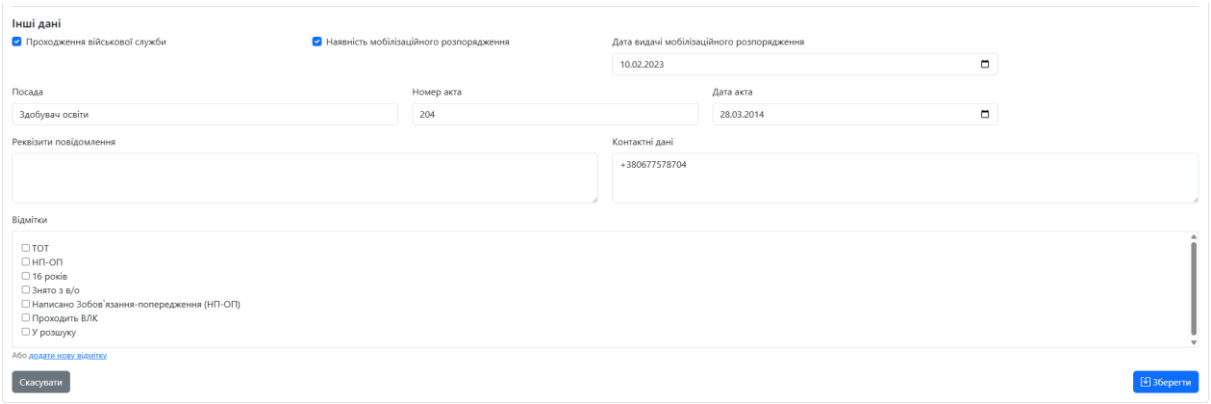


Рисунок 3.12 – Форма додавання та редагування студента (нижня частина)

Також на сторінці редагування реалізовано блок «Історія змін». Він представлений у вигляді хронологічного списку карток, де відображається дата зміни, ім’я оператора та деталізований опис (старе значення на нове). Це дозволяє візуально контролювати еволюцію даних кожного студента (див. рис. 3.13).

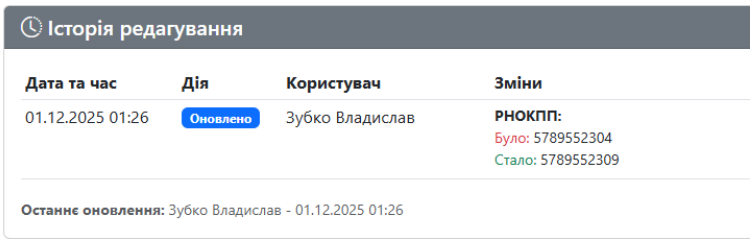


Рисунок 3.13 – Історія редагування

Основний документ для обміну даними, який формується системою у форматі .xlsx за допомогою бібліотеки openpyxl. Функція експорту генерує комплексний файл, що складається з трьох аркушів, кожен з яких відповідає за окрему категорію студентів (див. рис. 3.14 – 3.16).

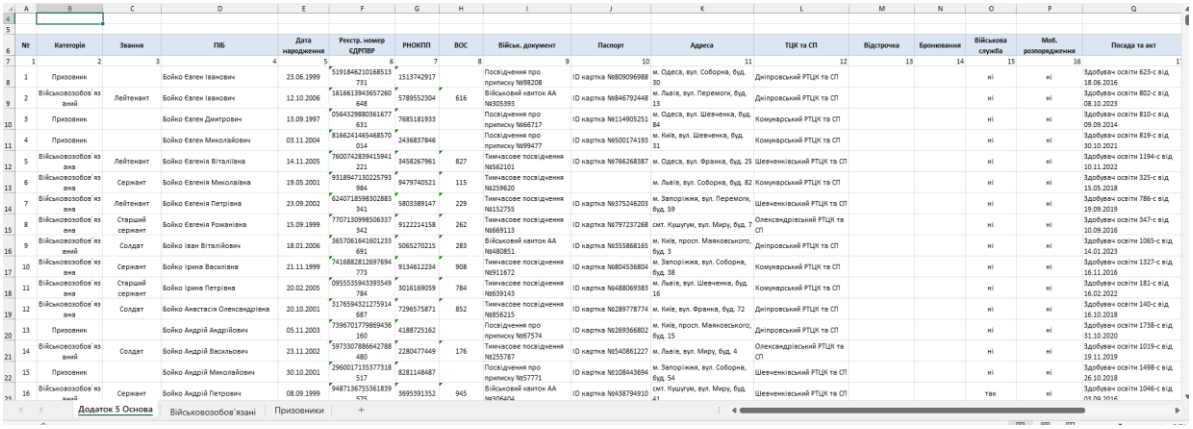


Рисунок 3.14 – Додаток 5 з усіма студентами

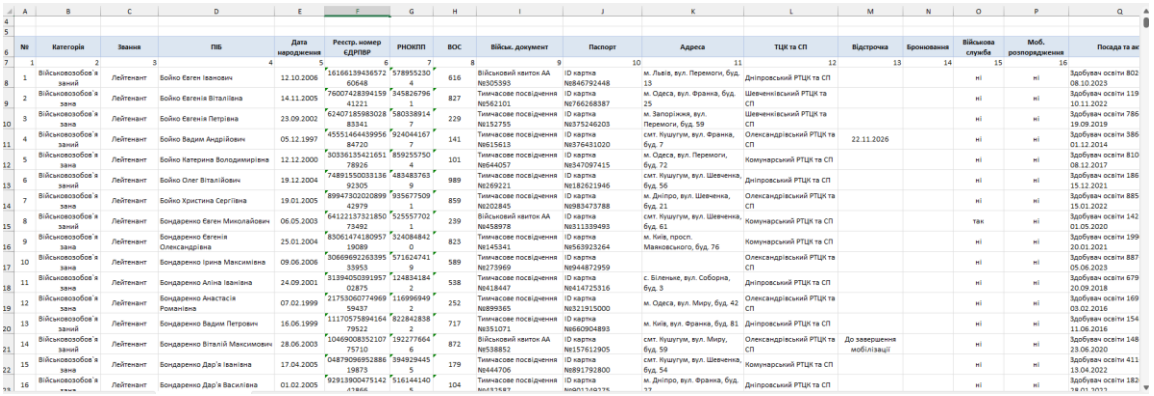


Рисунок 3.15 – Список військовозобов`язаних

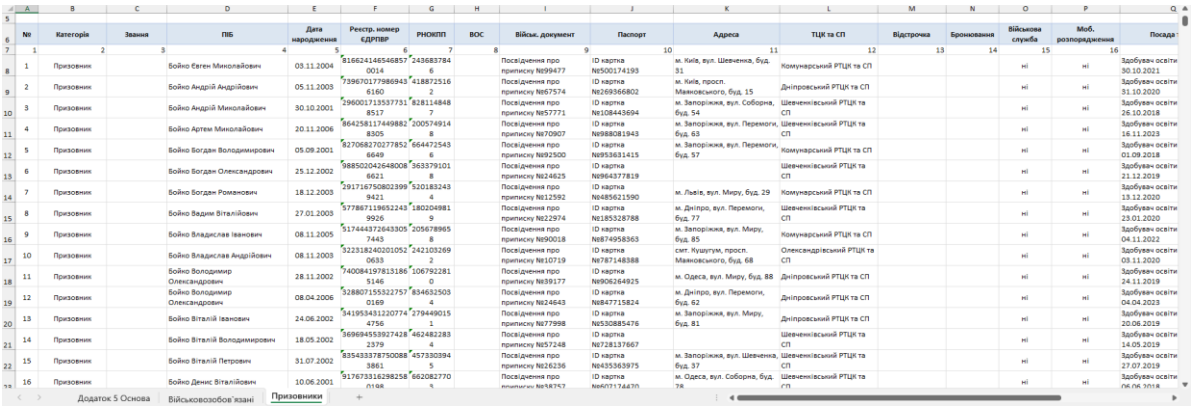


Рисунок 3.16 – Список призовників

Модуль «Список боржників» призначений для внутрішнього аудиту якості даних. Система автоматично аналізує картку кожного студента на предмет повноти заповнення обов’язкових полів (див. рис. 3.17).

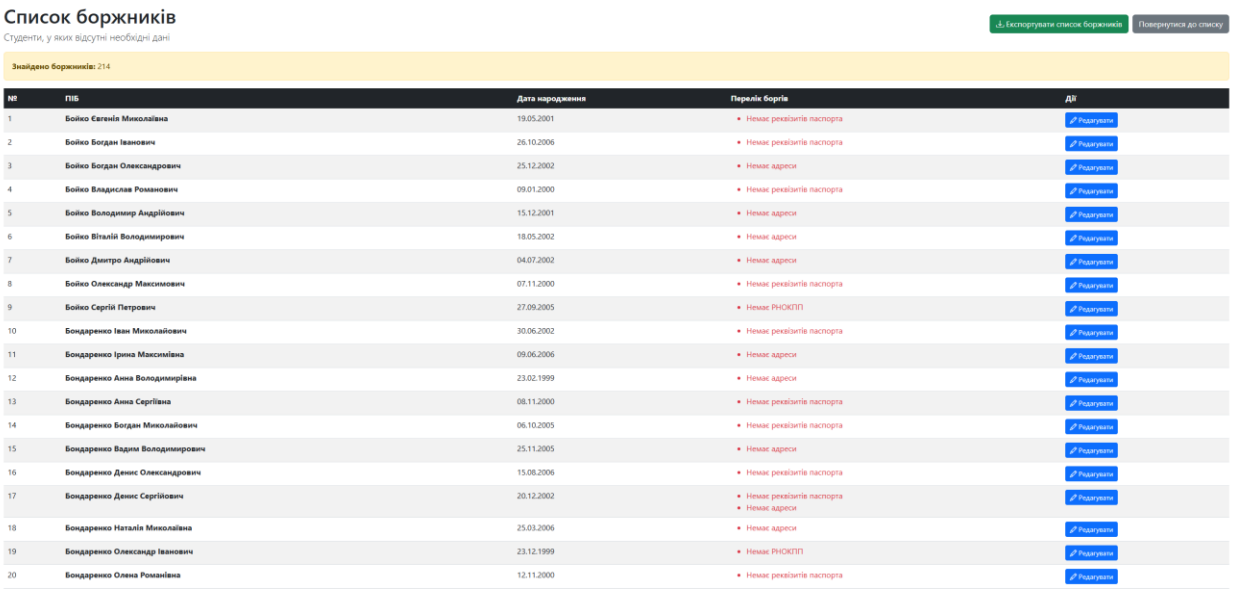


Рисунок 3.17 – Список боржників

У зв’язку зі змінами в законодавстві, критично важливим є своєчасне переведення призовників, які досягли 25-річного віку, у категорію військовозобов’язаних. Вручну відстежувати вік тисяч студентів неможливо, тому система автоматизує цей процес.

Модуль аналізує дату народження всіх студентів зі статусом «Призовник» і формує два списки (див. рис. 3.18 – 3.19).

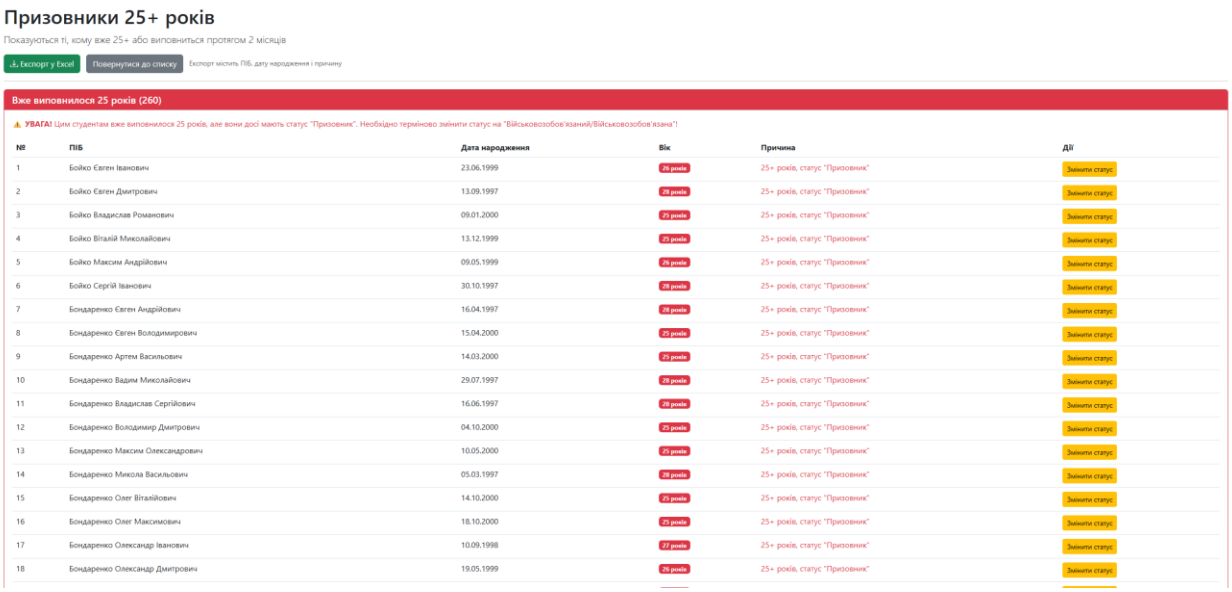


Рисунок 3.18 – Список призовників яким вже 25+ років

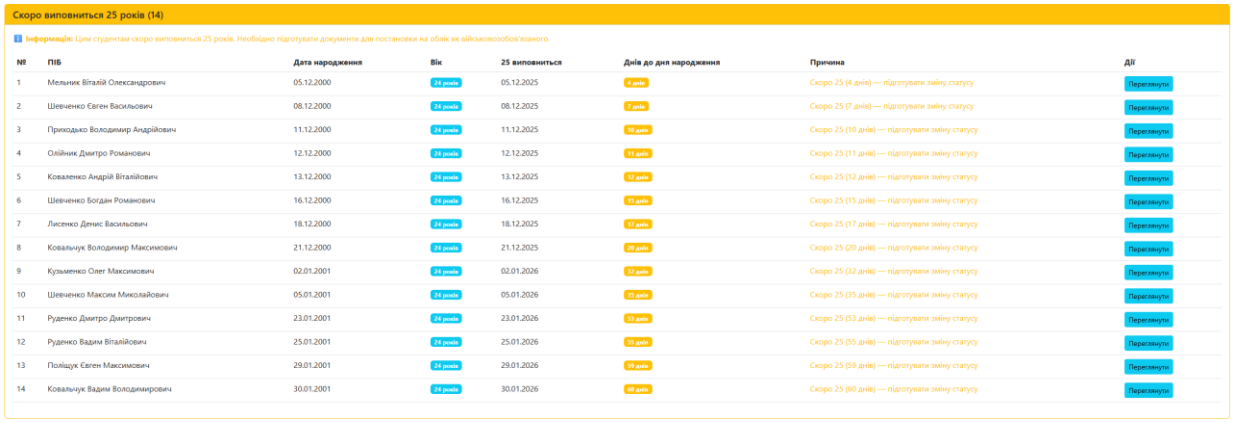


Рисунок 3.19 – Список студентів яким скоро 25 років

«Вже виповнилося 25 років»

Список осіб, які потребують негайної зміни статусу та направлення до ТЦК та СП.

«Скоро виповниться 25 років»

Список попередження, де система розраховує та показує кількість днів, що залишилися до 25-річчя.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальну науково-прикладну задачу автоматизації процесу військового обліку студентів у закладах вищої освіти шляхом розробки та впровадження спеціалізованої інформаційної системи. Проведений на початку дослідження аналіз предметної області виявив суттєві недоліки існуючих підходів, що базуються на паперовому документообігу та розрізнених електронних таблицях, зокрема відсутність цілісності даних, низький рівень захисту та неможливість оперативного формування звітності. Це обґрунтувало необхідність створення централізованого вебзастосунку, здатного вирішити ці проблеми комплексно.

В ході виконання роботи було спроєктовано архітектуру системи на основі патерну MVC з використанням високорівневого фреймворку Django, що дозволило забезпечити модульність, масштабованість та надійність програмного рішення. Розроблена реляційна модель бази даних успішно відображає всі необхідні сутності предметної області, включаючи розширені дані про студентів, довідники територіальних центрів комплектування та історію змін. Реалізація централізованого сховища даних дозволила усунути проблему дублювання інформації та забезпечити її верифікацію на етапі введення.

Ключовим результатом практичної частини роботи стала програмна реалізація модуля інтеграції з Excel, який забезпечує двосторонній обмін даними. Розроблений механізм імпорту дозволяє автоматизувати наповнення бази даних та актуалізацію інформації про навчання, включаючи інтелектуальне визначення факультету на основі наказів про зарахування. Функція експорту забезпечує формування суворо регламентованих звітних форм, таких як «Додаток 5», а також списків військовозобов’язаних та

призовників, що повністю відповідає вимогам законодавства та значно зменшує навантаження на адміністративний персонал.

Окремої уваги заслуговує реалізація підсистеми аналітики та контролю, яка автоматично виявляє студентів з неповними даними («боржників») та здійснює активний моніторинг призовників, які досягають граничного 25-річного віку. Впровадження адаптивних алгоритмів перевірки, що враховують категорію військового обов’язку та наявність специфічних відміток, дозволило мінімізувати кількість помилкових спрацьовувань та підвищити точність обліку.

Важливим досягненням роботи є забезпечення високого рівня інформаційної безпеки розробленої системи. Реалізована рольова модель доступу з чітким розмежуванням повноважень адміністраторів та працівників, механізми захисту від підбору паролів та вебатак, а також система глибокого аудиту, що фіксує історію змін кожного поля, гарантують захист персональних даних від несанкціонованого доступу та модифікації. Розроблений програмний продукт є повністю функціональним, готовим до впровадження в адміністративні процеси навчального закладу та має потенціал для подальшого масштабування, зокрема через перехід на промислові системи управління базами даних та інтеграцію з іншими університетськими сервісами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Student Report Card in Django. GeeksforGeeks. URL : https://www.geeksforgeeks.org/python/student-report-card-in-django/ (дата звернення : 15.06.2025).
2. Python Student Records Management System. GitHub. URL : https://github.com/professor0121/Python-Student-Records-Management-System/blob/main/README.md (дата звернення : 15.06.2025).
3. Master Django REST Framework: Build a Student Management System API. DEV Community. URL : https://dev.to/ebereplenty/master-django-rest-framework-build-a-student-management-system-api-with-function-based-views-1plc (дата звернення : 15.06.2025).
4. Official Documentation. Django Project. URL : https://docs.djangoproject.com/en/stable/ (дата звернення : 17.06.2025).
5. Student Management System. GitHub (m3hrab). URL : https://github.com/m3hrab/Student-management-system (дата звернення : 18.06.2025).
6. Student Management System. GitHub (m3hrab). URL : https://github.com/m3hrab/Student-management-system (дата звернення : 18.06.2025).
7. Student Management System. GitHub (m3hrab). URL : https://github.com/m3hrab/Student-management-system (дата звернення : 18.06.2025).
8. Прикидьєва А.О. Сучасні реалізації програмного забезпечення для управління житловими комплексами. Розвиток суспільства та науки в умовах цифрової трансформації: матеріали VII Міжнародної студентської наукової конференції // ГО «Молодіжна наукова ліга». Вінниця, 2024. №18.

С. 284 -286. URL : https://doi.org/10.62732/liga-inter-15.11.2024 (дата звернення: 25.06.2025).

9. Інтегроване стратегічне управління, управління портфелями, програмами, проєктами. eNTUKhPIIR. URL : https://repository.kpi.kharkov.ua/items/d929796e-7813-42f6-8402-03f084c3f031 (дата звернення : 25.06.2025).

10. Інтелектуальне управління групою підводних транспортних підводних роботів з використанням робота-координатора. #istkpi. URL : https://asac.kpi.ua/article/view/279102 (дата звернення : 25.06.2025).

11. Актуальні проблеми автоматизації та інформаційних технологій. Т. 26. Огляд існуючих автоматизованих систем управління освітніми закладами. Дніпро : Дніпровський національний університет імені Олеся Гончара. 2022. 9 с. URL : https://actualproblems.dp.ua/index.php/APAIT/article/download/213/157 (дата звернення : 25.06.2025).

12. Student Information system. Sourcecodester. URL : https://www.sourcecodester.com/visual-basic/4548/student-information-system.html (дата звернення : 25.06.2025).

13. Student-management-system. GitHub. URL : https://github.com/m3hrab/Student-management-system (дата звернення : 25.06.2025).

14. Django School Management System. GitHub. URL : https://github.com/akshaybengani/Django-School-Management-System (дата звернення : 22.07.2025).

15. Django College Management System. GitHub. URL : https://github.com/itswadesh/django-college-management-system (дата звернення : 22.07.2025).

16. Building Student Management System with Django. Medium. URL : https://medium.com/ (дата звернення : 25.07.2025).

17. Information Systems in Higher Education. ResearchGate. URL : https://www.researchgate.net/ (дата звернення : 27.07.2025).

18. Educational Information Systems : Trends and Challenges. Springer. URL : https://link.springer.com/ (дата звернення : 27.07.2025).

19. Student Data Management Systems in Universities. IEEE Xplore. URL : https://ieeexplore.ieee.org/ (дата звернення : 29.07.2025).

20. School Management System Django. GitHub. URL : https://github.com/EricksonAtHome/school-management-system-django (дата звернення : 01.08.2025).

21. Django Student Portal. GitHub. URL : https://github.com/harshil1712/django-student-portal (дата звернення : 01.08.2025).

22. University Management System Django. GitHub. URL : https://github.com/ashutosh1919/university-management-system-django (дата звернення : 03.08.2025).

23. Django REST Framework for Student APIs. Towards Data Science. URL : https://towardsdatascience.com/ (дата звернення : 05.08.2025).

24. Django Attendance Management System. GitHub. URL : https://github.com/harshil1712/django-attendance-management (дата звернення : 07.08.2025).

25. Django Exam Management System. GitHub. URL : https://github.com/ExamSystem/django-exam-management (дата звернення : 07.08.2025).

26. Django Result Management System. GitHub. URL : https://github.com/ResultSystem/django-result-management (дата звернення : 10.08.2025).

27. Django Hostel Management System. GitHub. URL : https://github.com/HostelSystem/django-hostel-management (дата звернення : 12.08.2025).

28. Django Library Management System. GitHub. URL : https://github.com/LibrarySystem/django-library-management (дата звернення : 12.08.2025).

29. ERPNext for Education (Django Integration). GitHub. URL : https://github.com/frappe/erpnext (дата звернення : 15.08.2025).

30. Django College ERP. GitHub. URL : https://github.com/CollegeERP/django-college-erp (дата звернення : 18.08.2025).

31. Django Student Result Portal. GitHub. URL : https://github.com/ResultPortal/django-student-result (дата звернення : 20.08.2025).

32. Django School ERP. GitHub. URL : https://github.com/SchoolERP/django-school-erp (дата звернення : 20.08.2025).

33. Django Student Admission System. GitHub. URL : https://github.com/AdmissionSystem/django-student-admission (дата звернення : 23.08.2025).

34. Django Student Feedback System. GitHub. URL : https://github.com/FeedbackSystem/django-student-feedback (дата звернення : 25.08.2025).

35. Django Student Grading System. GitHub. URL : https://github.com/GradingSystem/django-student-grading (дата звернення : 28.08.2025).

36. Django Student Timetable Management. GitHub. URL : https://github.com/TimetableSystem/django-student-timetable (дата звернення : 01.09.2025).

ДОДАТОК А

Models.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser


class CustomUser(AbstractUser):
    """Кастомна модель користувача з прізвищем, ім’ям та роллю"""
    ROLE_CHOICES = [
        (‘admin’, ‘Адміністратор’),
        (‘employee’, ‘Працівник’),
    ]

    last_name = models.CharField(max_length=150, verbose_name="Прізвище")
    first_name = models.CharField(max_length=150, verbose_name="Ім’я")
    role = models.CharField(max_length=20, choices=ROLE_CHOICES, default=‘employee’, verbose_name="Роль")
    is_active = models.BooleanField(default=True, verbose_name="Активний")

    class Meta:
        verbose_name = "Користувач"
        verbose_name_plural = "Користувачі"
        ordering = [‘last_name’, ‘first_name’]

    def __str__(self):
        return f"{self.last_name} {self.first_name} ({self.username})"

    def get_full_name(self):
        """Повертає повне ім’я"""
        return f"{self.last_name} {self.first_name}".strip()

    def is_admin(self):
        """Перевіряє чи користувач є адміністратором"""
        return self.role == ‘admin’ or self.is_superuser

    def is_employee(self):
        """Перевіряє чи користувач є працівником"""
        return self.role == ‘employee’


class TerritorialCenter(models.Model):
    """Територіальні центри комплектування та соціальної підтримки"""
    name = models.CharField(max_length=500, unique=True, verbose_name="Назва ТЦК та СП")
```

```
        class Meta:
            verbose_name = "ТЦК та СП"
            verbose_name_plural = "ТЦК та СП"
            ordering = [‘name’]

        def __str__(self):
            return self.name


    class Student(models.Model):
        CATEGORY_CHOICES = [
            (‘Військовозобов`язаний’, ‘Військовозобов`язаний’),
            (‘Військовозобов`язана’, ‘Військовозобов`язана’),
            (‘Призовник’, ‘Призовник’),
            (‘Резервіст’, ‘Резервіст’),
        ]

        RANK_CHOICES = [
            (‘Полковник’, ‘Полковник’),
            (‘Підполковник’, ‘Підполковник’),
            (‘Майор’, ‘Майор’),
            (‘Капітан’, ‘Капітан’),
            (‘Старший лейтенант’, ‘Старший лейтенант’),
            (‘Лейтенант’, ‘Лейтенант’),
            (‘Молодший лейтенант’, ‘Молодший лейтенант’),
            (‘Прапорщик’, ‘Прапорщик’),
            (‘Головний старшина’, ‘Головний старшина’),
            (‘Старшина’, ‘Старшина’),
            (‘Старший сержант’, ‘Старший сержант’),
            (‘Сержант’, ‘Сержант’),
            (‘Молодший сержант’, ‘Молодший сержант’),
            (‘Старший солдат’, ‘Старший солдат’),
            (‘Старший матрос’, ‘Старший матрос’),
            (‘Солдат’, ‘Солдат’),
            (‘Рядовий’, ‘Рядовий’),
            (‘Матрос (ВМС)’, ‘Матрос (ВМС)’),
            (‘Матрос’, ‘Матрос’),
            (‘Рекрут’, ‘Рекрут’),
        ]

        MARK_CHOICES = [
            (‘ТОТ’, ‘ТОТ’),
            (‘НП-ОП’, ‘НП-ОП’),
            (‘16 років’, ‘16 років’),
            (‘Знято з в/о’, ‘Знято з в/о’),
            (‘Написано Зобов`язання-попередження (НП-ОП)’, ‘Написано Зобов`язання-
попередження (НП-ОП)’),
            (‘Проходить ВЛК’, ‘Проходить ВЛК’),
            (‘У розшуку’, ‘У розшуку’),
        ]
```

```
STATUS_CHOICES = [
   (‘active’, ‘Активний’),
   (‘expelled’, ‘Відрахований’),
   (‘excluded’, ‘Виключений з обліку’),
   (‘serving’, ‘Чинний військовослужбовець’),
]

# Основні поля
status = models.CharField(max_length=20, choices=STATUS_CHOICES, default=‘active’)
category = models.CharField(max_length=50, choices=CATEGORY_CHOICES, verbose_name="Категорія військового обов’язку")
rank = models.CharField(max_length=50, choices=RANK_CHOICES, blank=True, null=True, verbose_name="Військове звання")
full_name = models.CharField(max_length=200, verbose_name="Прізвище, ім’я та по батькові")
birth_date = models.DateField(verbose_name="Дата народження")

# Реєстраційні дані
registry_number = models.CharField(max_length=100, blank=True, verbose_name="Реєстраційний номер в ЄДРПВР")
tax_number = models.CharField(max_length=10, blank=True, verbose_name="РНОКПП")
refused_tax_number = models.BooleanField(default=False, verbose_name="Відмовився від РНОКПП")

# Військові дані
military_specialty = models.CharField(max_length=100, blank=True, verbose_name="Військово-облікова спеціальність")
military_document = models.TextField(blank=True, verbose_name="Реквізити військово-облікового документа")
passport_data = models.TextField(blank=True, verbose_name="Реквізити паспорта")

# Адреса
address = models.TextField(blank=True, verbose_name="Адреса")

# ТЦК
territorial_center = models.ForeignKey(TerritorialCenter, on_delete=models.SET_NULL, null=True, blank=True, verbose_name="ТЦК та СП")

# Відстрочка
deferment_date = models.DateField(null=True, blank=True, verbose_name="Відстрочка до")
deferment_until_mobilization = models.BooleanField(default=False, verbose_name="Відстрочка до завершення мобілізації")

# Бронювання
booking_date = models.DateField(null=True, blank=True, verbose_name="Бронювання до")

# Інші дані
```

```
        military_service = models.BooleanField(default=False, verbose_name="Проходження військової служби")
        mobilization_order = models.BooleanField(default=False, verbose_name="Наявність мобілізаційного розпорядження")
        mobilization_order_date = models.DateField(null=True, blank=True, verbose_name="Дата видачі мобілізаційного розпорядження")

        # Посада
        position = models.CharField(max_length=200, default="Здобувач освіти", verbose_name="Посада")
        appointment_number = models.CharField(max_length=50, blank=True, verbose_name="Номер акта")
        appointment_date = models.DateField(null=True, blank=True, verbose_name="Дата акта")

        # Повідомлення
        notification_details = models.TextField(blank=True, verbose_name="Реквізити повідомлення")

        # Контакти
        contact_info = models.TextField(blank=True, verbose_name="Контактні дані")

        # Відмітки
        marks = models.JSONField(default=list, blank=True, verbose_name="Відмітки")

        # Дані про відрахування
        expulsion_order_number = models.CharField(max_length=50, blank=True, verbose_name="Номер наказу про відрахування")
        expulsion_order_date = models.DateField(null=True, blank=True, verbose_name="Дата наказу про відрахування")

        # Дані для виключених/служать
        exclusion_reason = models.CharField(max_length=200, blank=True, verbose_name="Причина виключення")

        # Факультет та статус навчання
        faculty = models.CharField(max_length=200, blank=True, verbose_name="Факультет")
        is_studying = models.BooleanField(default=False, verbose_name="Навчається")

        # Хто створив та редагував
        created_by = models.ForeignKey(CustomUser, on_delete=models.SET_NULL, null=True, blank=True, related_name='created_students', verbose_name="Створено")
        updated_by = models.ForeignKey(CustomUser, on_delete=models.SET_NULL, null=True, blank=True, related_name='updated_students', verbose_name="Оновлено")

        created_at = models.DateTimeField(auto_now_add=True)
        updated_at = models.DateTimeField(auto_now=True)

        class Meta:
            verbose_name = "Студент"
            verbose_name_plural = "Студенти"
```

```
        ordering = ['full_name']

    def __str__(self):
        return f"{self.full_name} ({self.birth_date})"

    def get_address(self):
        return self.address or ""

    def get_tax_number_display(self):
        """Повертає відформатований РНОКПП без .0"""
        if not self.tax_number:
            return ''
        # Видалити .0 з кінця, якщо воно є
        tax_num = self.tax_number.rstrip('.0') if self.tax_number.endswith('.0') else self.tax_number
        return tax_num

    def save(self, *args, **kwargs):
        # Очистити tax_number від .0 перед збереженням
        if self.tax_number and self.tax_number.endswith('.0'):
            self.tax_number = self.tax_number[:-2]
        super().save(*args, **kwargs)

    def get_rank_priority(self):
        """Повертає пріоритет звання для сортування"""
        rank_order = {
            'Полковник': 1,
            'Підполковник': 2,
            'Майор': 3,
            'Капітан': 4,
            'Старший лейтенант': 5,
            'Лейтенант': 6,
            'Молодший лейтенант': 7,
            'Прапорщик': 8,
            'Головний старшина': 9,
            'Старшина': 10,
            'Старший сержант': 11,
            'Сержант': 12,
            'Молодший сержант': 13,
            'Старший солдат': 14,
            'Старший матрос': 15,
            'Солдат': 16,
            'Рядовий': 17,
            'Матрос (ВМС)': 18,
            'Матрос': 19,
            'Рекрут': 20,
        }
        return rank_order.get(self.rank, 999)

    def age(self):
        from datetime import date
        today = date.today()
```

```
        return today.year – self.birth_date.year – ((today.month, today.day) < (self.birth_date.month, self.birth_date.day))

    def get_filtered_marks(self):
        """Повертає відфільтровані відмітки без контактних даних"""
        if not self.marks:
            return []

        import re
        filtered = []

        for mark in self.marks:
            if not mark:
                continue

            mark_lower = mark.lower().strip()

            # Перевірка на контактні дані
            is_contact = (
                mark_lower.startswith("тел.") or
                mark.startswith("380") or
                mark.startswith("+380") or
                (mark.startswith("(") and ")" in mark) or
                ("(" in mark and ")" in mark and "-" in mark and len(mark) > 5) or
                "неодружений" in mark_lower or
                "одружений" in mark_lower or
                "одружена" in mark_lower or
                "неодружена" in mark_lower or
                "@" in mark or
                (len(mark) > 15 and any(char.isdigit() for char in mark) and "-" in mark)
            )

            # Перевірка на формат телефону через regex
            phone_pattern = re.compile(r'[0-9]{3}[-\s]?[0-9]{2,3}[-\s]?[0-9]{2,3}[-\s]?[0-9]{2,3}')
            if phone_pattern.search(mark):
                is_contact = True

            if not is_contact:
                filtered.append(mark)

        return filtered

    def is_debtor(self):
        """Перевіряє чи є студент боржником"""
        if self.marks and ('ТОТ' in self.marks or '16 років' in self.marks):
            return False, []

        missing_fields = []
        is_conscript = self.category == 'Призовник'

        # Перевірка полів В:Р
```

```
        # B – Категорія (завжди перевіряємо)
        if not self.category:
            missing_fields.append("Немає категорії військового обов’язку")

        # C – Звання (не перевіряємо для призовників)
        if not is_conscript and not self.rank:
            missing_fields.append("Немає військового звання")

        # D – ПІБ (завжди перевіряємо)
        if not self.full_name:
            missing_fields.append("Немає ПІБ")

        # E – Дата народження (завжди перевіряємо)
        if not self.birth_date:
            missing_fields.append("Немає дати народження")

        # F – Реєстраційний номер (не перевіряємо для призовників)
        if not is_conscript and not self.registry_number:
            missing_fields.append("Немає реєстраційного номера ЄДРПВР")

        # G – РНОКПП (завжди перевіряємо)
        if not self.tax_number and not self.refused_tax_number:
            missing_fields.append("Немає РНОКПП")

        # H – Військово-облікова спеціальність (не перевіряємо для призовників)
        if not is_conscript and not self.military_specialty:
            missing_fields.append("Немає військово-облікової спеціальності")

        # I – Військовий документ (завжди перевіряємо)
        if not self.military_document:
            missing_fields.append("Немає військово-облікового документа")

        # J – Паспорт (завжди перевіряємо)
        if not self.passport_data:
            missing_fields.append("Немає реквізитів паспорта")

        # K – Адреса (завжди перевіряємо)
        if not self.get_address():
            missing_fields.append("Немає адреси")

        # L – ТЦК (завжди перевіряємо)
        if not self.territorial_center:
            missing_fields.append("Немає ТЦК та СП")

        # M – Відстрочка – не враховуємо при формуванні боргів
        # N – Бронювання – не враховуємо при формуванні боргів

        # O – Військова служба (не перевіряємо для призовників, але перевіряємо наявність даних)
        # P – Мобілізаційне розпорядження (не перевіряємо для призовників, але перевіряємо наявність даних)
```

```
            # Q – Посада (завжди перевіряємо)
            if not self.position:
                missing_fields.append("Немає посади")

            return len(missing_fields) > 0, missing_fields


        class StudentHistory(models.Model):
            """Історія змін студента"""
            ACTION_CHOICES = [
                ('created', 'Створено'),
                ('updated', 'Оновлено'),
                ('deleted', 'Видалено'),
            ]

            student = models.ForeignKey(Student, on_delete=models.CASCADE, related_name='history', verbose_name="Студент")
            action = models.CharField(max_length=20, choices=ACTION_CHOICES, verbose_name="Дія")
            user = models.ForeignKey(CustomUser, on_delete=models.SET_NULL, null=True, blank=True, verbose_name="Користувач")
            changes = models.JSONField(default=dict, blank=True, verbose_name="Зміни")
            created_at = models.DateTimeField(auto_now_add=True, verbose_name="Дата та час")

            class Meta:
                verbose_name = "Історія змін"
                verbose_name_plural = "Історія змін"
                ordering = ['-created_at']

            def __str__(self):
                return f"{self.student.full_name} – {self.get_action_display()} – {self.created_at}"


        class StudyingStudent(models.Model):
            """Студенти, які навчаються (з CSV/Excel файлів) – не показуються в основному списку"""
            full_name = models.CharField(max_length=200, verbose_name="ПІБ")
            birth_date = models.DateField(verbose_name="Дата народження")
            faculty = models.CharField(max_length=200, blank=True, default='', verbose_name="Факультет")
            structural_unit = models.CharField(max_length=500, blank=True, default='', verbose_name="Структурний підрозділ")
            enrollment_order = models.TextField(blank=True, default='', verbose_name="Наказ про зарахування")
            enrollment_code = models.CharField(max_length=50, blank=True, default='', null=True, verbose_name="Код наказу")
            enrollment_number = models.CharField(max_length=50, blank=True, default='', verbose_name="Номер наказу")
            enrollment_date = models.DateField(null=True, blank=True, verbose_name="Дата наказу")
```

```
        degree_type = models.CharField(max_length=20, choices=[('bachelor', 'Бакалавр'), ('master', 'Магістр')], default='bachelor', verbose_name="Рівень освіти")
        created_at = models.DateTimeField(auto_now_add=True)
        updated_at = models.DateTimeField(auto_now=True)

        class Meta:
            verbose_name = "Студент, який навчається"
            verbose_name_plural = "Студенти, які навчаються"
            ordering = ['full_name']
            # Використовуємо індекс замість unique_together, оскільки enrollment_number може бути порожнім
            indexes = [
                models.Index(fields=['full_name', 'birth_date', 'enrollment_number']),
            ]

        def __str__(self):
            return f"{self.full_name} ({self.birth_date})"


    class CustomMark(models.Model):
        """Додаткові відмітки"""
        name = models.CharField(max_length=200, unique=True, verbose_name="Назва відмітки")
        created_at = models.DateTimeField(auto_now_add=True)

        class Meta:
            verbose_name = "Додаткова відмітка"
            verbose_name_plural = "Додаткові відмітки"
            ordering = ['name']

        def __str__(self):
            return self.name
```

ДОДАТОК Б

Views.py

```
from django.shortcuts import render, redirect, get_object_or_404
import os
from django.contrib import messages
from django.contrib.auth.decorators import login_required
from django.contrib.auth import login, logout
from django.contrib.auth.forms import AuthenticationForm
from django.http import JsonResponse, FileResponse
from django.views.decorators.http import require_http_methods
from django.core.paginator import Paginator
from django.db import transaction
from django.db.models import Q
from datetime import datetime, date, timedelta
import calendar
from .models import Student, TerritorialCenter, CustomMark, CustomUser, StudentHistory
from .forms import StudentForm, TerritorialCenterForm, CustomMarkForm, CustomUserForm, CustomUserCreateForm
from .excel_utils import export_debtors_to_excel
from .security import (
    get_client_ip, is_ip_blocked, block_ip, get_login_attempts,
    increment_login_attempts, reset_login_attempts, check_rate_limit,
    log_security_event, log_user_action
)
from functools import wraps


def admin_required(view_func):
    """Декоратор для перевірки чи користувач є адміністратором"""
    @wraps(view_func)
    def _wrapped_view(request, *args, **kwargs):
        if not request.user.is_authenticated:
            return redirect('registry:login')
        if not request.user.is_admin():
            messages.error(request, 'У вас немає прав доступу до цієї сторінки.')
            return redirect('registry:index')
        return view_func(request, *args, **kwargs)
    return _wrapped_view


def save_student_history(student, user, action='updated', old_instance=None):
    """Зберігає історію змін студента"""
    changes = {}
```

```
        if action == 'created':
            # При створенні зберігаємо основні поля
            changes = {
                'full_name': student.full_name,
                'category': student.category,
                'birth_date': str(student.birth_date) if student.birth_date else None,
            }
        elif action == 'updated' and old_instance:
            # При оновленні зберігаємо тільки змінені поля
            for field in Student._meta.get_fields():
                if field.name in ['created_by', 'updated_by', 'created_at', 'updated_at', 'id', 'marks', 'territorial_center']:
                    continue
                if hasattr(old_instance, field.name) and hasattr(student, field.name):
                    old_value = getattr(old_instance, field.name)
                    new_value = getattr(student, field.name)
                    if old_value != new_value:
                        # Отримати читабельну назву поля
                        field_verbose = field.verbose_name if hasattr(field, 'verbose_name') else field.name
                        # Не додавати зміни, якщо обидва значення порожні або None
                        old_str = str(old_value) if old_value not in (None, '') else None
                        new_str = str(new_value) if new_value not in (None, '') else None
                        if old_str or new_str:
                            changes[field_verbose] = {
                                'old': old_str,
                                'new': new_str,
                            }

        StudentHistory.objects.create(
            student=student,
            action=action,
            user=user,
            changes=changes
        )


    def login_view(request):
        """Сторінка входу в систему з захистом від brute force"""
        if request.user.is_authenticated:
            return redirect('registry:index')

        ip_address = get_client_ip(request)

        # Перевірка чи IP заблокований
        if is_ip_blocked(ip_address):
            log_security_event('BLOCKED_LOGIN_ATTEMPT', request.user, ip_address, 'IP заблокований')
            messages.error(request, 'Ваша IP адреса тимчасово заблокована через занадто багато невдалих спроб входу. Спробуйте пізніше.')
            return render(request, 'registry/login.html', {'form': AuthenticationForm()})
```

```
        # Перевірка rate limiting
        if not check_rate_limit(ip_address):
            log_security_event('RATE_LIMIT_EXCEEDED', request.user, ip_address)
            messages.error(request, 'Занадто багато спроб входу. Спробуйте пізніше.')
            return render(request, 'registry/login.html', {'form': AuthenticationForm()})

        if request.method == 'POST':
            form = AuthenticationForm(request, data=request.POST)
            username = request.POST.get('username', '')

            if form.is_valid():
                user = form.get_user()

                # Перевірка чи користувач не заблокований
                attempts = get_login_attempts(username)
                if attempts >= 5:
                    log_security_event('BLOCKED_USER_LOGIN', user, ip_address,
f'{attempts} невдалих спроб')
                    messages.error(request, 'Ваш акаунт тимчасово заблокований через занадто
багато невдалих спроб входу. Спробуйте пізніше.')
                    return render(request, 'registry/login.html', {'form': form})

                # Успішний вхід
                login(request, user)
                reset_login_attempts(username)
                log_security_event('SUCCESSFUL_LOGIN', user, ip_address)
                log_user_action('LOGIN', user)
                messages.success(request, f'Вітаємо, {user.username}!')

                next_url = request.GET.get('next', None)
                if next_url:
                    return redirect(next_url)
                return redirect('registry:index')
            else:
                # Невдала спроба входу
                if username:
                    is_blocked = increment_login_attempts(username)
                    attempts = get_login_attempts(username)

                    if is_blocked:
                        block_ip(ip_address)
                        log_security_event('BRUTE_FORCE_DETECTED', request.user,
ip_address, f'Користувач {username}, {attempts} спроб')
                        messages.error(request, 'Занадто багато невдалих спроб входу. Ваша IP
адреса тимчасово заблокована.')
                    else:
                        remaining = 5 – attempts
                        log_security_event('FAILED_LOGIN', request.user, ip_address,
f'Користувач {username}, спроба {attempts}/5')
                        if remaining > 0:
```

```
                    messages.error(request, f'Невірне ім\'я користувача або пароль. Залишилось спроб: {remaining}')
                else:
                    messages.error(request, 'Невірне ім\'я користувача або пароль. Акаунт заблокований.')
        else:
            form = AuthenticationForm()

        return render(request, 'registry/login.html', {'form': form})


    def logout_view(request):
        """Вихід з системи"""
        user = request.user
        ip_address = get_client_ip(request)
        was_authenticated = user.is_authenticated
        logout(request)
        if was_authenticated:
            log_user_action('LOGOUT', user, details=f'Користувач {user.username} вийшов з системи')
            log_security_event('LOGOUT', user, ip_address)
        messages.info(request, 'Ви вийшли з системи.')
        return redirect('registry:login')


    @login_required(login_url='registry:login')
    def index(request):
        """Головна сторінка – список студентів"""
        category_filter = request.GET.getlist('category')
        search_query = request.GET.get('search', '')
        studying_filter = request.GET.get('studying', '')

        students = Student.objects.filter(status='active')

        # Видалено можливість масового очищення відміток через параметри запиту

        # Пошук
        if search_query:
            students = students.filter(
                Q(full_name__icontains=search_query) |
                Q(registry_number__icontains=search_query) |
                Q(tax_number__icontains=search_query) |
                Q(territorial_center__name__icontains=search_query) |
                Q(military_specialty__icontains=search_query)
            )

        # Фільтр за категоріями (можна декілька)
        if category_filter:
            students = students.filter(category__in=category_filter)

        # Фільтр за статусом навчання
        if studying_filter == 'yes':
```

```
        students = students.filter(is_studying=True)
    elif studying_filter == 'no':
        students = students.filter(is_studying=False)

    # Сортування
    obliged_set = {'Військовозобов`язаний','Військовозобов`язана','Резервіст'}
    if category_filter and set(category_filter).issubset(obliged_set):
        students = sorted(list(students), key=lambda s: s.get_rank_priority())
    elif category_filter == ['Призовник']:
        students = students.order_by('full_name')
    else:
        students = students.order_by('full_name')

    # Пагінація для великих списків
    page_str = request.GET.get('page', '1')
    try:
        page = int(page_str)
    except Exception:
        page = 1
    per_page_str = request.GET.get('per_page', '50')
    try:
        per_page = max(1, int(per_page_str))
    except Exception:
        per_page = 50
    paginator = Paginator(list(students), per_page)
    # Безпечне отримання сторінки (не кидає помилок)
    page_obj = paginator.get_page(page)
    students = page_obj.object_list

    # Підрахунок
    students_count = paginator.count

    context = {
        'students': students,
        'category_filter': category_filter,
        'search_query': search_query,
        'studying_filter': studying_filter,
        'students_count': students_count,
        'page_obj': page_obj,
        'per_page': per_page,
    }

    return render(request, 'registry/index.html', context)


@login_required(login_url='registry:login')
def student_add(request):
    """Додавання нового студента"""
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            student = form.save(commit=False)
```

```
            student.created_by = request.user
            student.updated_by = request.user
            student.save()
            form.save_m2m()

            # Зберегти історію
            save_student_history(student, request.user, action='created')

            # Логування дії
            log_user_action('CREATE_STUDENT', request.user, 'Student', student.id,
f'Створено студента: {student.full_name}')

            messages.success(request, f'Студента {student.full_name} успішно додано!')
            return redirect('registry:index')
        else:
            form = StudentForm()

        return render(request, 'registry/student_form.html', {'form': form, 'action':
'Додати'})


    @login_required(login_url='registry:login')
    def student_edit(request, pk):
        """Редагування студента"""
        student = get_object_or_404(Student, pk=pk)

        if request.method == 'POST':
            # Зберегти старий стан для історії
            old_instance = Student.objects.get(pk=student.pk)

            form = StudentForm(request.POST, instance=student)
            if form.is_valid():
                student = form.save(commit=False)
                student.updated_by = request.user
                student.save()
                form.save_m2m()

                # Зберегти історію
                save_student_history(student, request.user, action='updated',
old_instance=old_instance)

                # Логування дії
                log_user_action('UPDATE_STUDENT', request.user, 'Student', student.id,
f'Оновлено студента: {student.full_name}')

                messages.success(request, f'Дані студента {student.full_name} оновлено!')
                return redirect('registry:index')
        else:
            form = StudentForm(instance=student)

        # Отримати історію
        history = StudentHistory.objects.filter(student=student).order_by('-created_at')[:50]
```

```
    return render(request, 'registry/student_form.html', {
        'form': form,
        'student': student,
        'action': 'Редагувати',
        'history': history
    })


@login_required(login_url='registry:login')
@admin_required
def student_delete(request, pk):
    """Видалення/переміщення студента (тільки для адмінів)"""
    student = get_object_or_404(Student, pk=pk)

    if request.method == 'POST':
        action = request.POST.get('action')

        if action == 'expel':
            # Відрахування
            order_number = request.POST.get('order_number')
            order_date_str = request.POST.get('order_date')

            if order_number and order_date_str:
                try:
                    order_date = datetime.strptime(order_date_str, "%Y-%m-%d").date()
                    student.status = 'expelled'
                    student.expulsion_order_number = order_number
                    student.expulsion_order_date = order_date
                    student.save()

                    messages.success(request, f'Студента {student.full_name} відраховано!')
                    return redirect('registry:index')
                except Exception as e:
                    messages.error(request, f'Помилка: {str(e)}')
            else:
                messages.error(request, 'Необхідно вказати номер та дату наказу!')

        elif action == 'exclude':
            # Виключення з обліку
            reason = request.POST.get('reason', '')
            student.status = 'excluded'
            student.exclusion_reason = reason
            student.save()

            messages.success(request, f'Студента {student.full_name} виключено з обліку!')
            return redirect('registry:index')

        elif action == 'serving':
            # Чинний військовослужбовець
            reason = request.POST.get('reason', 'Чинний військовослужбовець')
```

```
            student.status = 'serving'
            student.exclusion_reason = reason
            student.save()

            messages.success(request, f'Студента {student.full_name} позначено як військовослужбовця!')
            return redirect('registry:index')

        return render(request, 'registry/student_delete.html', {'student': student})


    @login_required(login_url='registry:login')
    @admin_required
    def debtors_list(request):
        """Список боржників – студентів з неповними даними"""
        active_students = Student.objects.filter(status='active')
        debtors = []

        for student in active_students:
            is_debtor, missing_fields = student.is_debtor()
            if is_debtor:
                debtors.append({
                    'student': student,
                    'missing_fields': missing_fields
                })

        context = {
            'debtors': debtors,
        }

        return render(request, 'registry/debtors.html', context)


    @login_required(login_url='registry:login')
    @admin_required
    def export_debtors_excel(request):
        """Завантажити окремий Excel файл зі списком боржників."""
        try:
            file_path = export_debtors_to_excel()
            response = FileResponse(
                open(file_path, 'rb'),
                content_type='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'
            )
            response['Content-Disposition'] = f'attachment; filename="{os.path.basename(file_path)}"'
            return response
        except Exception as e:
            messages.error(request, f'Помилка експорту боржників: {str(e)}')
            return redirect('registry:debtors_list')

    @login_required(login_url='registry:login')
```

```
@admin_required
def age_25_check(request):
    """Перевірка призовників, яким виповнилося/виповниться 25 років"""
    today = date.today()
    turned_25 = []
    turning_25_soon = []

    # Призовники, яким вже виповнилося 25
    conscripts = Student.objects.filter(
        status='active',
        category='Призовник'
    )

    def safe_25th_birthday(bd: date) -> date:
        target_year = bd.year + 25
        last_day = calendar.monthrange(target_year, bd.month)[1]
        safe_day = min(bd.day, last_day)
        return date(target_year, bd.month, safe_day)

    for student in conscripts:
        if student.birth_date:
            age = student.age()
            if age >= 25:
                turned_25.append(student)
            else:
                # Розрахунок днів до 25-річчя; беремо поріг 60 днів
                next_birthday = safe_25th_birthday(student.birth_date)
                days_until = (next_birthday – today).days
                if 0 <= days_until <= 60:
                    turning_25_soon.append({
                        'student': student,
                        'date': next_birthday,
                        'days_until': days_until
                    })

    # Сортувати скоро виповняться за кількістю днів
    turning_25_soon.sort(key=lambda x: x['days_until'])

    context = {
        'turned_25': turned_25,
        'turning_25_soon': turning_25_soon,
    }

    return render(request, 'registry/age_25_check.html', context)


@login_required(login_url='registry:login')
@admin_required
def export_age25_excel(request):
    """Експорт списку призовників 25+ та тих, кому <=60 днів до 25, у Excel з окремими аркушами для кожного факультету."""
    try:
```

```
from openpyxl import Workbook
from openpyxl.utils import get_column_letter

today = date.today()
def safe_25th_birthday(bd: date) -> date:
    target_year = bd.year + 25
    last_day = calendar.monthrange(target_year, bd.month)[1]
    safe_day = min(bd.day, last_day)
    return date(target_year, bd.month, safe_day)
conscripts = Student.objects.filter(status='active', category='Призовник')

turned = []
soon = []
for s in conscripts:
    if not s.birth_date:
        continue
    age = s.age()
    if age >= 25:
        turned.append({'student': s, 'reason': '25+ років, статус "Призовник"'})
    else:
        nb = safe_25th_birthday(s.birth_date)
        d = (nb – today).days
        if 0 <= d <= 60:
            soon.append({'student': s, 'reason': f'Скоро 25 ({d} днів) – змінити
статус'})

all_items = turned + soon

# Групувати по факультетах
items_by_faculty = {}
items_no_faculty = []

for item in all_items:
    faculty = item['student'].faculty
    if faculty:
        if faculty not in items_by_faculty:
            items_by_faculty[faculty] = []
        items_by_faculty[faculty].append(item)
    else:
        items_no_faculty.append(item)

wb = Workbook()
# Видалити дефолтний аркуш
if 'Sheet' in wb.sheetnames:
    wb.remove(wb['Sheet'])

# Створити аркуш для кожного факультету
for faculty, faculty_items in items_by_faculty.items():
    # Обмежити назву аркуша до 31 символу (обмеження Excel)
    sheet_name = faculty[:31] if len(faculty) > 31 else faculty
    sh = wb.create_sheet(sheet_name)
```

```
                headers = ['№', 'ПІБ', 'Дата народження', 'Причина']
                for i, h in enumerate(headers, start=1):
                    sh.cell(row=1, column=i, value=h)

                row = 2
                idx = 1
                for item in faculty_items:
                    s = item['student']
                    sh.cell(row=row, column=1, value=idx)
                    sh.cell(row=row, column=2, value=s.full_name)
                    sh.cell(row=row, column=3, value=s.birth_date.strftime('%d.%m.%Y') if
s.birth_date else '')
                    sh.cell(row=row, column=4, value=item['reason'])
                    row += 1
                    idx += 1

                # Автоширина
                for column in sh.columns:
                    max_len = 0
                    col_letter = get_column_letter(column[0].column)
                    for c in column:
                        if c.value:
                            max_len = max(max_len, len(str(c.value)))
                    sh.column_dimensions[col_letter].width = min(max_len + 2, 70)

            # Створити аркуш для призовників без факультету (якщо є)
            if items_no_faculty:
                sh = wb.create_sheet('Без факультету')

                headers = ['№', 'ПІБ', 'Дата народження', 'Причина']
                for i, h in enumerate(headers, start=1):
                    sh.cell(row=1, column=i, value=h)

                row = 2
                idx = 1
                for item in items_no_faculty:
                    s = item['student']
                    sh.cell(row=row, column=1, value=idx)
                    sh.cell(row=row, column=2, value=s.full_name)
                    sh.cell(row=row, column=3, value=s.birth_date.strftime('%d.%m.%Y') if
s.birth_date else '')
                    sh.cell(row=row, column=4, value=item['reason'])
                    row += 1
                    idx += 1

                # Автоширина
                for column in sh.columns:
                    max_len = 0
                    col_letter = get_column_letter(column[0].column)
                    for c in column:
                        if c.value:
                            max_len = max(max_len, len(str(c.value)))
```

```
                sh.column_dimensions[col_letter].width = min(max_len + 2, 70)

            from django.conf import settings
            import os
            os.makedirs(getattr(settings, 'EXCEL_FILES_DIR',
os.path.join(settings.BASE_DIR, 'excel_files')), exist_ok=True)
            file_path = os.path.join(settings.EXCEL_FILES_DIR,
f'conscripts_25_{datetime.now().strftime("%Y%m%d_%H%M%S")}.xlsx')
            wb.save(file_path)

            response = FileResponse(open(file_path, 'rb'),
content_type='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet')
            response['Content-Disposition'] = f'attachment;
filename="{os.path.basename(file_path)}"'
            return response
        except Exception as e:
            messages.error(request, f'Помилка експорту 25+: {str(e)}')
            return redirect('registry:age_25_check')


    @login_required(login_url='registry:login')
    @admin_required
    def territorial_centers(request):
        """Управління ТЦК та СП"""
        if request.method == 'POST':
            form = TerritorialCenterForm(request.POST)
            if form.is_valid():
                form.save()
                messages.success(request, 'ТЦК додано!')
                return redirect('registry:territorial_centers')

        centers = TerritorialCenter.objects.all().order_by('name')
        form = TerritorialCenterForm()

        context = {
            'centers': centers,
            'form': form,
        }

        return render(request, 'registry/territorial_centers.html', context)


    @login_required(login_url='registry:login')
    @admin_required
    def territorial_center_delete(request, pk):
        """Видалення ТЦК"""
        center = get_object_or_404(TerritorialCenter, pk=pk)
        if request.method == 'POST':
            center.delete()
            messages.success(request, 'ТЦК видалено!')
            return redirect('registry:territorial_centers')
        return render(request, 'registry/territorial_center_delete.html', {'center': center})
```

```
@login_required(login_url='registry:login')
@admin_required
def custom_marks(request):
    """Управління відмітками"""
    if request.method == 'POST':
        form = CustomMarkForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, 'Відмітку додано!')
            return redirect('registry:custom_marks')

    marks = CustomMark.objects.all().order_by('name')
    form = CustomMarkForm()

    context = {
        'marks': marks,
        'form': form,
    }

    return render(request, 'registry/custom_marks.html', context)


@login_required(login_url='registry:login')
@admin_required
def custom_mark_delete(request, pk):
    """Видалення відмітки"""
    mark = get_object_or_404(CustomMark, pk=pk)
    if request.method == 'POST':
        mark.delete()
        messages.success(request, 'Відмітку видалено!')
        return redirect('registry:custom_marks')
    return render(request, 'registry/custom_mark_delete.html', {'mark': mark})


@login_required(login_url='registry:login')
@admin_required
def export_to_excel(request):
    """Експорт даних в Excel файл"""
    try:
        from .excel_utils import sync_db_to_excel
        file_path = sync_db_to_excel()

        log_user_action('EXPORT_EXCEL', request.user, details='Експорт даних в Excel
файл')

        response = FileResponse(
            open(file_path, 'rb'),
            content_type='application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet'
        )
```

```
            response['Content-Disposition'] = f'attachment; filename="military_registry_{datetime.now().strftime("%Y%m%d_%H%M%S")}.xlsx"'

            messages.success(request, 'Файл Excel успішно згенеровано!')
            return response
        except Exception as e:
            messages.error(request, f'Помилка експорту: {str(e)}')
            return redirect('registry:index')


    @login_required(login_url='registry:login')
    @admin_required
    def import_from_excel(request):
        """Імпорт даних з Excel файлу"""
        if request.method == 'POST' and request.FILES.get('excel_file'):
            excel_file = request.FILES['excel_file']
            clear_database = request.POST.get('clear_database') == '1'

            try:
                # Очистити базу перед імпортом, якщо потрібно
                if clear_database:
                    with transaction.atomic():
                        deleted_count = Student.objects.filter(status='active').count()
                        Student.objects.filter(status='active').delete()
                        messages.info(request, f'Видалено {deleted_count} активних студентів перед імпортом')

                from django.core.files.storage import default_storage
                from django.core.files.base import ContentFile

                # Зберегти файл тимчасово
                temp_path = default_storage.save('temp_import.xlsx', ContentFile(excel_file.read()))
                full_path = default_storage.path(temp_path)

                # Завантажити workbook
                from openpyxl import load_workbook
                wb = load_workbook(full_path, data_only=True)

                # Читати з аркуша "Додаток 5 Основа" або "Основа"
                sheet_name = None
                for name in ['Додаток 5 Основа', 'Основа', 'Sheet1', 'Sheet']:
                    if name in wb.sheetnames:
                        sheet_name = name
                        break

                if not sheet_name:
                    messages.error(request, 'Не знайдено аркуш з даними!')
                    return redirect('registry:index')

                sheet = wb[sheet_name]
```

```
                # Читати дані з рядка 8
                imported_count = 0
                updated_count = 0

                # Визначити фактичний останній рядок з даними (незалежно від "хвоста" max_row)
                def row_has_data(r: int) -> bool:
                    for c in range(2, 21):  # B..T
                        val = sheet.cell(row=r, column=c).value
                        if val is not None and str(val).strip() != "":
                            return True
                    return False
                last_row = sheet.max_row
                while last_row >= 8 and not row_has_data(last_row):
                    last_row -= 1

                with transaction.atomic():
                    for row_num in range(8, last_row + 1):
                        # Перевірити чи є ПІБ (обов'язкове поле)
                        full_name = sheet.cell(row=row_num, column=4).value  # D колонка
                        if full_name is None or str(full_name).strip() == "":
                            continue

                        try:
                            from .excel_utils import read_student_from_row
                            from .models import StudyingStudent
                            data = read_student_from_row(sheet, row_num)

                            # Перевірити чи існує студент з таким ПІБ та датою народження
                            if data.get('birth_date'):
                                # Шукаємо в StudyingStudent для підтягування факультету та статусу навчання
                                # Спочатку шукаємо по ПІБ та даті народження
                                studying_students = StudyingStudent.objects.filter(
                                    full_name=data['full_name'],
                                    birth_date=data['birth_date']
                                )

                                # Якщо знайдено кілька студентів, намагаємося знайти по номеру наказу
                                # (якщо номер наказу зберігається в notification_details або appointment_number)
                                studying_student = None
                                appointment_number = data.get('appointment_number', '')
                                notification = data.get('notification_details', '')

                                if studying_students.count() > 1:
                                    # Спробуємо знайти по номеру наказу з appointment_number або notification_details
                                    # enrollment_number може бути в форматі "123-с" або "123"
                                    # appointment_number може бути "123" (без "-с")
                                    if appointment_number:
```

```
                        # Нормалізуємо appointment_number (прибираємо "-с" якщо є)
                        normalized_appointment = appointment_number.replace('-с', '').replace('-', '').strip()
                        for ss in studying_students:
                            if ss.enrollment_number:
                                # Нормалізуємо enrollment_number
                                normalized_enrollment = ss.enrollment_number.replace('-с', '').replace('-', '').strip()
                                if normalized_enrollment == normalized_appointment:
                                    studying_student = ss
                                    break
                                # Також перевіряємо чи enrollment_number міститься в notification
                                if ss.enrollment_number in notification:
                                    studying_student = ss
                                    break
                        # Якщо не знайдено, перевіряємо notification
                        if not studying_student and notification:
                            for ss in studying_students:
                                if ss.enrollment_number and ss.enrollment_number in notification:
                                    studying_student = ss
                                    break
                        # Якщо не знайдено, беремо першого
                        if not studying_student:
                            studying_student = studying_students.first()
                    elif studying_students.exists():
                        studying_student = studying_students.first()

                    # Якщо знайдено студента, який навчається, підтягуємо дані
                    if studying_student:
                        data['faculty'] = studying_student.faculty
                        data['is_studying'] = True
                    else:
                        data['is_studying'] = False

                    # Шукаємо існуючого студента по ПІБ, даті народження та номеру наказу
                    # Це дозволяє створювати окремі записи для студентів, які навчаються на кількох спеціальностях
                    existing_student = None
                    if appointment_number:
                        # Якщо є номер наказу, шукаємо по ньому
                        existing_student = Student.objects.filter(
                            full_name=data['full_name'],
                            birth_date=data['birth_date'],
                            appointment_number=appointment_number,
                            status='active'
                        ).first()
                    else:
                        # Якщо немає номера наказу, шукаємо по ПІБ та даті
                        existing_student = Student.objects.filter(
```

```
                full_name=data['full_name'],
                birth_date=data['birth_date'],
                appointment_number='',
                status='active'
            ).first()

            if existing_student:
                # Оновити існуючого
                for key, value in data.items():
                    if key == 'territorial_center_name':
                        if value:
                            tcc, created = TerritorialCenter.objects.get_or_create(name=value)
                            existing_student.territorial_center = tcc
                    elif hasattr(existing_student, key) and key != 'marks':
                        setattr(existing_student, key, value)
                    elif key == 'marks':
                        existing_student.marks = value
                existing_student.save()
                updated_count += 1
            else:
                # Створити нового (може бути кілька записів для одного студента з різними номерами наказів)
                student = Student.objects.create(
                    category=data.get('category', 'Призовник'),
                    rank=data.get('rank'),
                    full_name=data['full_name'],
                    birth_date=data.get('birth_date'),
                    registry_number=data.get('registry_number', ''),
                    tax_number=data.get('tax_number', ''),
                    refused_tax_number=data.get('refused_tax_number', False),
                    military_specialty=data.get('military_specialty', ''),
                    military_document=data.get('military_document', ''),
                    passport_data=data.get('passport_data', ''),
                    address=data.get('address', ''),
                    faculty=data.get('faculty', ''),
                    is_studying=data.get('is_studying', False),
                    territorial_center=TerritorialCenter.objects.get_or_create(
                        name=data.get('territorial_center_name', '')
                    )[0] if data.get('territorial_center_name') else None,
                    deferment_date=data.get('deferment_date'),
                    deferment_until_mobilization=data.get('deferment_until_mobilization', False),
                    booking_date=data.get('booking_date'),
                    military_service=data.get('military_service', False),
                    mobilization_order=data.get('mobilization_order', False),
                    mobilization_order_date=data.get('mobilization_order_date'),
                    position=data.get('position', 'Здобувач освіти'),
                    appointment_number=data.get('appointment_number', ''),
                    appointment_date=data.get('appointment_date'),
                    notification_details=data.get('notification_details', ''),
                    contact_info=data.get('contact_info', ''),
```

```
                        marks=data.get('marks', []),
                        status='active'
                    )
                    imported_count += 1

                except Exception as e:
                    messages.warning(request, f'Помилка обробки рядка {row_num}: {str(e)}')
                    continue

            # Видалити тимчасовий файл
            default_storage.delete(temp_path)

            log_user_action('IMPORT_EXCEL', request.user, details=f'Імпорт з Excel: додано {imported_count}, оновлено {updated_count}')
            messages.success(
                request,
                f'Імпорт завершено! Додано: {imported_count}, Оновлено: {updated_count}'
            )

            # Синхронізувати після імпорту (поза транзакцією для швидкодії)
            try:
                from .excel_utils import sync_db_to_excel
                sync_db_to_excel()
            except Exception:
                pass

        except Exception as e:
            messages.error(request, f'Помилка імпорту: {str(e)}')

    return render(request, 'registry/import_excel.html')


@login_required(login_url='registry:login')
def save_to_excel_ajax(request):
    """AJAX endpoint для збереження даних в Excel при редагуванні"""
    if request.method == 'POST':
        try:
            from .excel_utils import sync_db_to_excel
            sync_db_to_excel()
            return JsonResponse({'status': 'success', 'message': 'Дані збережено!'})
        except Exception as e:
            return JsonResponse({'status': 'error', 'message': str(e)})

    return JsonResponse({'status': 'error', 'message': 'Невірний метод'})


@require_http_methods(["GET"])
@login_required(login_url='registry:login')
def search_tcc(request):
    """Пошук ТЦК для autocomplete"""
```

```
        query = request.GET.get('q', '')
        if query:
            centers = TerritorialCenter.objects.filter(name__icontains=query)[:10]
            results = [{'id': c.id, 'name': c.name} for c in centers]
            return JsonResponse({'results': results})
        return JsonResponse({'results': []})


    @login_required(login_url='registry:login')
    @admin_required
    def upload_excel_file(request):
        """Завантаження існуючого Excel файлу як основного"""
        if request.method == 'POST' and request.FILES.get('excel_file'):
            excel_file = request.FILES['excel_file']

            try:
                import shutil
                from django.conf import settings
                from pathlib import Path

                excel_dir = Path(settings.EXCEL_FILES_DIR)
                excel_dir.mkdir(exist_ok=True)

                main_file = excel_dir / 'military_registry.xlsx'
                backup_file = excel_dir / f'military_registry_backup_{datetime.now().strftime("%Y%m%d_%H%M%S")}.xlsx'

                # Створити резервну копію поточного файлу, якщо він існує
                if main_file.exists():
                    shutil.copy2(main_file, backup_file)
                    messages.info(request, f'Поточний файл збережено як резервну копію: {backup_file.name}')

                # Зберегти завантажений файл як основний
                with open(main_file, 'wb') as f:
                    for chunk in excel_file.chunks():
                        f.write(chunk)

                messages.success(request, 'Excel файл успішно завантажено як основний!')

                # Опціонально: імпортувати дані з нового файлу
                import_option = request.POST.get('import_after_upload', False)
                if import_option:
                    from openpyxl import load_workbook as _load_wb
                    wb = _load_wb(main_file)

                    # Знайти аркуш
                    sheet_name = None
                    for name in ['Додаток 5 Основа', 'Основа', 'Sheet1', 'Sheet']:
                        if name in wb.sheetnames:
                            sheet_name = name
                            break
```

```
                if sheet_name:
                    sheet = wb[sheet_name]
                    imported_count = 0
                    updated_count = 0
                    with transaction.atomic():
                        # Визначити фактичний останній рядок з даними і пройтися лише до нього
                        def _row_has_data(r: int) -> bool:
                            for c in range(2, 21):
                                val = sheet.cell(row=r, column=c).value
                                if val is not None and str(val).strip() != "":
                                    return True
                            return False
                        _last = sheet.max_row
                        while _last >= 8 and not _row_has_data(_last):
                            _last -= 1

                        for row_num in range(8, _last + 1):
                            full_name = sheet.cell(row=row_num, column=4).value
                            if full_name is None or str(full_name).strip() == "":
                                continue
                            try:
                                from .excel_utils import read_student_from_row
                                from .models import StudyingStudent
                                data = read_student_from_row(sheet, row_num)
                                if data.get('birth_date'):
                                    # Шукаємо в StudyingStudent для підтягування факультету та статусу навчання
                                    studying_students = StudyingStudent.objects.filter(
                                        full_name=data['full_name'],
                                        birth_date=data['birth_date']
                                    )

                                    # Якщо знайдено кілька студентів, намагаємося знайти по номеру наказу
                                    studying_student = None
                                    appointment_number = data.get('appointment_number', '')
                                    notification = data.get('notification_details', '')

                                    if studying_students.count() > 1:
                                        # Спробуємо знайти по номеру наказу з appointment_number або notification_details
                                        if appointment_number:
                                            # Нормалізуємо appointment_number
                                            normalized_appointment = appointment_number.replace('-c', '').replace('-', '').strip()
                                            for ss in studying_students:
                                                if ss.enrollment_number:
                                                    normalized_enrollment = ss.enrollment_number.replace('-c', '').replace('-', '').strip()
                                                    if normalized_enrollment == normalized_appointment:
```

```
                        studying_student = ss
                        break
                    if ss.enrollment_number in notification:
                        studying_student = ss
                        break
            if not studying_student and notification:
                for ss in studying_students:
                    if ss.enrollment_number and ss.enrollment_number in notification:
                        studying_student = ss
                        break
            if not studying_student:
                studying_student = studying_students.first()
        elif studying_students.exists():
            studying_student = studying_students.first()

        # Якщо знайдено студента, який навчається, підтягуємо дані
        if studying_student:
            data['faculty'] = studying_student.faculty
            data['is_studying'] = True
        else:
            data['is_studying'] = False

        # Шукаємо існуючого студента по ПІБ, даті народження та номеру наказу
        existing_student = None
        if appointment_number:
            existing_student = Student.objects.filter(
                full_name=data['full_name'],
                birth_date=data['birth_date'],
                appointment_number=appointment_number,
                status='active'
            ).first()
        else:
            existing_student = Student.objects.filter(
                full_name=data['full_name'],
                birth_date=data['birth_date'],
                appointment_number='',
                status='active'
            ).first()

        if existing_student:
            for key, value in data.items():
                if key == 'territorial_center_name':
                    if value:
                        tcc, created = TerritorialCenter.objects.get_or_create(name=value)
                        existing_student.territorial_center = tcc
                elif hasattr(existing_student, key) and key != 'marks':
                    setattr(existing_student, key, value)
                elif key == 'marks':
                    existing_student.marks = value
```

```
                                existing_student.save()
                                updated_count += 1
                            else:
                                Student.objects.create(
                                    category=data.get('category', 'Призовник'),
                                    rank=data.get('rank'),
                                    full_name=data['full_name'],
                                    birth_date=data.get('birth_date'),
                                    registry_number=data.get('registry_number', ''),
                                    tax_number=data.get('tax_number', ''),
                                    refused_tax_number=data.get('refused_tax_number', False),
                                    military_specialty=data.get('military_specialty', ''),
                                    military_document=data.get('military_document', ''),
                                    passport_data=data.get('passport_data', ''),
                                    address=data.get('address', ''),
                                    faculty=data.get('faculty', ''),
                                    is_studying=data.get('is_studying', False),
                                    territorial_center=TerritorialCenter.objects.get_or_create(
                                        name=data.get('territorial_center_name', '')
                                    )[0] if data.get('territorial_center_name') else None,
                                    deferment_date=data.get('deferment_date'),
deferment_until_mobilization=data.get('deferment_until_mobilization', False),
                                    booking_date=data.get('booking_date'),
                                    military_service=data.get('military_service', False),
                                    mobilization_order=data.get('mobilization_order', False),
                                    mobilization_order_date=data.get('mobilization_order_date'),
                                    position=data.get('position', 'Здобувач освіти'),
                                    appointment_number=data.get('appointment_number', ''),
                                    appointment_date=data.get('appointment_date'),
                                    notification_details=data.get('notification_details', ''),
                                    contact_info=data.get('contact_info', ''),
                                    marks=data.get('marks', []),
                                    status='active'
                                )
                                imported_count += 1
                        except Exception:
                            continue
                    messages.info(request, f'Додатково імпортовано: додано {imported_count}, оновлено {updated_count} студентів')

            except Exception as e:
                messages.error(request, f'Помилка завантаження файлу: {str(e)}')

        return redirect('registry:index')


    @login_required(login_url='registry:login')
    @admin_required
    def upload_students_excel(request):
        """Завантаження Excel файлів зі списками студентів (Бакалаври та Магістри)"""
        if request.method == 'POST':
```

```
            bachelor_file = request.FILES.get('bachelor_file')
            master_file = request.FILES.get('master_file')

            if not bachelor_file and not master_file:
                messages.error(request, 'Необхідно завантажити хоча б один файл!')
                return redirect('registry:upload_students_excel')

            try:
                import os
                from django.core.files.storage import default_storage
                from django.core.files.base import ContentFile
                from .excel_utils import load_students_from_excel
                from .models import StudyingStudent

                bachelor_count = 0
                master_count = 0

                # Обробка файлу бакалаврів
                if bachelor_file:
                    # Визначаємо розширення файлу
                    file_ext = os.path.splitext(bachelor_file.name)[1].lower()
                    temp_filename = f'temp_bachelor{file_ext}'
                    temp_path = default_storage.save(temp_filename,
ContentFile(bachelor_file.read()))
                    full_path = default_storage.path(temp_path)

                    try:
                        students_data = load_students_from_excel(full_path,
degree_type='bachelor')

                        for student_data in students_data:
                            full_name = student_data.get('full_name')
                            birth_date = student_data.get('birth_date')

                            if not full_name or not birth_date:
                                continue

                            # Створюємо або оновлюємо запис в StudyingStudent
                            enrollment_number = student_data.get('enrollment_number') or ''
                            enrollment_code = student_data.get('enrollment_code') or ''

                            # Використовуємо update_or_create з правильними параметрами
                            defaults = {
                                'faculty': student_data.get('faculty') or '',
                                'structural_unit': student_data.get('structural_unit') or '',
                                'enrollment_order': student_data.get('enrollment_order') or '',
                                'enrollment_code': enrollment_code if enrollment_code else None,
                                'enrollment_date': student_data.get('enrollment_date'),
                                'degree_type': 'bachelor'
                            }

                            # Якщо enrollment_number порожній, шукаємо тільки по ПІБ та даті
```

```
                    if enrollment_number:
                        studying_student, created = StudyingStudent.objects.update_or_create(
                            full_name=full_name,
                            birth_date=birth_date,
                            enrollment_number=enrollment_number,
                            defaults=defaults
                        )
                    else:
                        # Якщо enrollment_number порожній, шукаємо по ПІБ та даті, а якщо не знайдено – створюємо
                        studying_student = StudyingStudent.objects.filter(
                            full_name=full_name,
                            birth_date=birth_date,
                            enrollment_number=''
                        ).first()
                        if studying_student:
                            for key, value in defaults.items():
                                setattr(studying_student, key, value)
                            studying_student.save()
                            created = False
                        else:
                            studying_student = StudyingStudent.objects.create(
                                full_name=full_name,
                                birth_date=birth_date,
                                enrollment_number='',
                                **defaults
                            )
                            created = True

                    bachelor_count += 1

                messages.success(request, f"Бакалаври: завантажено {bachelor_count} студентів")
            finally:
                default_storage.delete(temp_path)

        # Обробка файлу магістрів
        if master_file:
            # Визначаємо розширення файлу
            file_ext = os.path.splitext(master_file.name)[1].lower()
            temp_filename = f"temp_master{file_ext}"
            temp_path = default_storage.save(temp_filename, ContentFile(master_file.read()))
            full_path = default_storage.path(temp_path)

            try:
                students_data = load_students_from_excel(full_path, degree_type='master')

                for student_data in students_data:
                    full_name = student_data.get('full_name')
                    birth_date = student_data.get('birth_date')
```

```
                    if not full_name or not birth_date:
                        continue

                    # Створюємо або оновлюємо запис в StudyingStudent
                    enrollment_number = student_data.get('enrollment_number') or ''
                    enrollment_code = student_data.get('enrollment_code') or ''

                    # Використовуємо update_or_create з правильними параметрами
                    defaults = {
                        'faculty': student_data.get('faculty') or '',
                        'structural_unit': student_data.get('structural_unit') or '',
                        'enrollment_order': student_data.get('enrollment_order') or '',
                        'enrollment_code': enrollment_code if enrollment_code else None,
                        'enrollment_date': student_data.get('enrollment_date'),
                        'degree_type': 'master'
                    }

                    # Якщо enrollment_number порожній, шукаємо тільки по ПІБ та даті
                    if enrollment_number:
                        studying_student, created = StudyingStudent.objects.update_or_create(
                            full_name=full_name,
                            birth_date=birth_date,
                            enrollment_number=enrollment_number,
                            defaults=defaults
                        )
                    else:
                        # Якщо enrollment_number порожній, шукаємо по ПІБ та даті, а якщо не знайдено – створюємо
                        studying_student = StudyingStudent.objects.filter(
                            full_name=full_name,
                            birth_date=birth_date,
                            enrollment_number=''
                        ).first()
                        if studying_student:
                            for key, value in defaults.items():
                                setattr(studying_student, key, value)
                            studying_student.save()
                            created = False
                        else:
                            studying_student = StudyingStudent.objects.create(
                                full_name=full_name,
                                birth_date=birth_date,
                                enrollment_number='',
                                **defaults
                            )
                            created = True

                    master_count += 1

                messages.success(request, f'Магістри: завантажено {master_count} студентів')
            finally:
```

```
                default_storage.delete(temp_path)

            messages.success(request, f'Завантаження завершено успішно! Всього завантажено: {bachelor_count + master_count} студентів')
            return redirect('registry:index')

        except Exception as e:
            messages.error(request, f'Помилка завантаження файлів: {str(e)}')
            import traceback
            traceback.print_exc()

    return render(request, 'registry/upload_students_excel.html')


# ==================== Управління користувачами (тільки для адмінів) ====================

@login_required(login_url='registry:login')
@admin_required
def user_list(request):
    """Список користувачів (тільки для адмінів)"""
    users = CustomUser.objects.all().order_by('last_name', 'first_name')

    context = {
        'users': users,
    }
    return render(request, 'registry/user_list.html', context)


@login_required(login_url='registry:login')
@admin_required
def user_create(request):
    """Створення нового користувача (тільки для адмінів)"""
    if request.method == 'POST':
        form = CustomUserCreateForm(request.POST)
        if form.is_valid():
            user = form.save()
            log_user_action('CREATE_USER', request.user, 'CustomUser', user.id, f'Створено користувача: {user.username} ({user.get_full_name()})')
            messages.success(request, f'Користувача {user.get_full_name()} успішно створено!')
            return redirect('registry:user_list')
    else:
        form = CustomUserCreateForm()

    return render(request, 'registry/user_form.html', {'form': form, 'action': 'Створити'})


@login_required(login_url='registry:login')
@admin_required
def user_edit(request, pk):
    """Редагування користувача (тільки для адмінів)"""
```

```
    user = get_object_or_404(CustomUser, pk=pk)

    if request.method == 'POST':
        form = CustomUserForm(request.POST, instance=user)
        if form.is_valid():
            user = form.save()
            log_user_action('UPDATE_USER', request.user, 'CustomUser', user.id, f'Оновлено користувача: {user.username} ({user.get_full_name()})')
            messages.success(request, f'Дані користувача {user.get_full_name()} оновлено!')
            return redirect('registry:user_list')
    else:
        form = CustomUserForm(instance=user)

    return render(request, 'registry/user_form.html', {'form': form, 'user': user, 'action': 'Редагувати'})


@login_required(login_url='registry:login')
@admin_required
def user_delete(request, pk):
    """Видалення користувача (тільки для адмінів)"""
    user = get_object_or_404(CustomUser, pk=pk)

    # Не можна видалити самого себе
    if user == request.user:
        messages.error(request, 'Ви не можете видалити самого себе!')
        return redirect('registry:user_list')

    if request.method == 'POST':
        user_name = user.get_full_name()
        username = user.username
        user_id = user.id
        user.delete()
        log_user_action('DELETE_USER', request.user, 'CustomUser', user_id, f'Видалено користувача: {username} ({user_name})')
        messages.success(request, f'Користувача {user_name} видалено!')
        return redirect('registry:user_list')

    return render(request, 'registry/user_delete.html', {'user': user})
```

ДОДАТОК В

Excel_Utils.py

```
from datetime import datetime
import os
from django.conf import settings
from openpyxl import Workbook, load_workbook
from openpyxl.utils import get_column_letter
from openpyxl.styles import Alignment, Font, PatternFill, Border, Side


def _format_header_row(sheet, row_index: int, num_columns: int) -> None:
    """Apply header formatting: bold, centered, dark background, wrapped text."""
    header_fill = PatternFill(start_color="FFDCE6F1", end_color="FFDCE6F1",
fill_type="solid")
    header_font = Font(bold=True)
    thin = Side(border_style="thin", color="FFB4C6E7")
    border = Border(top=thin, left=thin, right=thin, bottom=thin)

    for col in range(1, num_columns + 1):
        cell = sheet.cell(row=row_index, column=col)
        cell.alignment = Alignment(horizontal='center', vertical='center', wrap_text=True)
        cell.font = header_font
        cell.fill = header_fill
        cell.border = border


def _format_data_cell(cell, number_format: str | None, horizontal: str = 'center', wrap:
bool = True) -> None:
    """Apply data cell formatting with optional number format, alignment and
wrapping."""
    cell.alignment = Alignment(horizontal=horizontal, vertical='center', wrap_text=wrap)
    if number_format:
        cell.number_format = number_format


def export_debtors_to_excel():
    """
    Експортує список боржників в окремий Excel файл з окремими аркушами для
кожного факультету
    """
    from .models import Student

    wb = Workbook()
    # Видалити дефолтний аркуш
    if 'Sheet' in wb.sheetnames:
```

```
    wb.remove(wb['Sheet'])

# Отримати всіх боржників
active_students = Student.objects.filter(status='active')
debtors = []

for student in active_students:
    is_debtor, missing_fields = student.is_debtor()
    if is_debtor:
        debtors.append({
            'student': student,
            'missing_fields': missing_fields
        })

# Групувати боржників по факультетах
debtors_by_faculty = {}
debtors_no_faculty = []

for item in debtors:
    faculty = item['student'].faculty
    if faculty:
        if faculty not in debtors_by_faculty:
            debtors_by_faculty[faculty] = []
        debtors_by_faculty[faculty].append(item)
    else:
        debtors_no_faculty.append(item)

# Створити аркуш для кожного факультету
for faculty, faculty_debtors in debtors_by_faculty.items():
    # Обмежити назву аркуша до 31 символу (обмеження Excel)
    sheet_name = faculty[:31] if len(faculty) > 31 else faculty
    sheet = wb.create_sheet(sheet_name)

    # Заголовки
    headers = ["№", "ПІБ", "Дата народження", "Перелік боргів"]
    for col, header in enumerate(headers, start=1):
        sheet.cell(row=1, column=col, value=header)
    _format_header_row(sheet, 1, len(headers))

    # Записати дані
    row_num = 2
    for idx, item in enumerate(faculty_debtors, start=1):
        # №
        cell = sheet.cell(row=row_num, column=1)
        cell.value = idx
        _format_data_cell(cell, None, 'center', True)

        # ПІБ (вирівнювання зліва)
        cell = sheet.cell(row=row_num, column=2)
        cell.value = item['student'].full_name
        _format_data_cell(cell, None, 'left', True)
```

```
                # Дата народження
                cell = sheet.cell(row=row_num, column=3)
                cell.value = item['student'].birth_date.strftime("%d.%m.%Y") if item['student'].birth_date else ""
                _format_data_cell(cell, None, 'center', True)

                # Перелік боргів
                debts_str = ", ".join(item['missing_fields'])
                cell = sheet.cell(row=row_num, column=4)
                cell.value = debts_str
                _format_data_cell(cell, None, 'left', True)

                row_num += 1

            # Автопідбір ширини колонок
            for column in sheet.columns:
                max_length = 0
                column_letter = get_column_letter(column[0].column)
                for cell in column:
                    try:
                        if cell.value:
                            max_length = max(max_length, len(str(cell.value)))
                    except:
                        pass
                adjusted_width = min(max_length + 2, 80)
                sheet.column_dimensions[column_letter].width = adjusted_width

        # Створити аркуш для боржників без факультету (якщо є)
        if debtors_no_faculty:
            sheet = wb.create_sheet("Без факультету")

            # Заголовки
            headers = ["№", "ПІБ", "Дата народження", "Перелік боргів"]
            for col, header in enumerate(headers, start=1):
                sheet.cell(row=1, column=col, value=header)
            _format_header_row(sheet, 1, len(headers))

            # Записати дані
            row_num = 2
            for idx, item in enumerate(debtors_no_faculty, start=1):
                # №
                cell = sheet.cell(row=row_num, column=1)
                cell.value = idx
                _format_data_cell(cell, None, 'center', True)

                # ПІБ (вирівнювання зліва)
                cell = sheet.cell(row=row_num, column=2)
                cell.value = item['student'].full_name
                _format_data_cell(cell, None, 'left', True)

                # Дата народження
                cell = sheet.cell(row=row_num, column=3)
```

```
                cell.value = item['student'].birth_date.strftime("%d.%m.%Y") if item['student'].birth_date else ""
                _format_data_cell(cell, None, 'center', True)

                # Перелік боргів
                debts_str = ", ".join(item['missing_fields'])
                cell = sheet.cell(row=row_num, column=4)
                cell.value = debts_str
                _format_data_cell(cell, None, 'left', True)

                row_num += 1

            # Автопідбір ширини колонок
            for column in sheet.columns:
                max_length = 0
                column_letter = get_column_letter(column[0].column)
                for cell in column:
                    try:
                        if cell.value:
                            max_length = max(max_length, len(str(cell.value)))
                    except:
                        pass
                adjusted_width = min(max_length + 2, 80)
                sheet.column_dimensions[column_letter].width = adjusted_width

        # Зберегти файл
        excel_dir = getattr(settings, 'EXCEL_FILES_DIR', os.path.join(settings.BASE_DIR, 'excel_files'))
        os.makedirs(excel_dir, exist_ok=True)
        file_path = os.path.join(excel_dir, f'debtors_{datetime.now().strftime("%Y%m%d_%H%M%S")}.xlsx')
        wb.save(file_path)

        return file_path


    def sync_db_to_excel():
        """
        Мінімальний експорт всієї бази у файл military_registry.xlsx
        Створює аркуш "Додаток 5 Основа" і записує основні поля з форматуванням.
        """
        from .models import Student

        # Відкриваємо існуючий основний файл; якщо його немає – піднімаємо помилку
        excel_dir = getattr(settings, 'EXCEL_FILES_DIR', os.path.join(settings.BASE_DIR, 'excel_files'))
        os.makedirs(excel_dir, exist_ok=True)
        file_path = os.path.join(excel_dir, 'military_registry.xlsx')

        if not os.path.exists(file_path):
            raise FileNotFoundError('Не знайдено основний Excel файл military_registry.xlsx. Спочатку завантажте його.')
```

```
            # Спробувати відкрити файл з обробкою помилок
            try:
                wb = load_workbook(file_path)
            except Exception as e:
                # Якщо файл пошкоджений, створити резервну копію та новий файл
                import shutil
                backup_path = os.path.join(excel_dir, f'military_registry_corrupted_{datetime.now().strftime("%Y%m%d_%H%M%S")}.xlsx')
                try:
                    shutil.copy2(file_path, backup_path)
                except:
                    pass  # Якщо не вдалося створити резервну копію, продовжуємо

                # Створити новий файл
                wb = Workbook()
                if 'Sheet' in wb.sheetnames:
                    wb.remove(wb['Sheet'])

                # Створити основний аркуш
                base_name = 'Додаток 5 Основа'
                sheet = wb.create_sheet(base_name)

                # Додати заголовки
                headers = [
                    "№", "Категорія", "Звання", "ПІБ", "Дата народження",
                    "ЄДРПВР", "РНОКПП", "ВОС", "Військ. документ", "Паспорт",
                    "Адреса", "ТЦК", "Відстрочка", "Бронювання", "Військ. служба",
                    "Моб. розпорядження", "Посада та акт", "Повідомлення", "Відмітки", "Контакти"
                ]
                for col, header in enumerate(headers, start=1):
                    sheet.cell(row=1, column=col, value=header)

                _format_header_row(sheet, 1, len(headers))

                # Зберегти новий файл
                wb.save(file_path)

                # Підняти помилку з інформацією про створення нового файлу
                raise Exception(f'Excel файл був пошкоджений. Створено новий файл. Пошкоджений файл збережено як: {os.path.basename(backup_path)}. Будь ласка, завантажте оригінальний файл знову.')

            # Не чіпаємо інші аркуші, оновлюємо цільові
            base_name = 'Додаток 5 Основа'
            obliged_name = 'Військовозобов`язані'
            conscripts_name = 'Призовники'

            sheet = wb[base_name] if base_name in wb.sheetnames else wb.create_sheet(base_name)
```

```
# Заголовки (A:T)
headers = [
    "№",  # A
    "Категорія",  # B
    "Звання",  # C
    "ПІБ",  # D
    "Дата народження",  # E
    "Реєстр. номер ЄДРПВР",  # F
    "РНОКПП",  # G
    "ВОС",  # H
    "Військ. документ",  # I
    "Паспорт",  # J
    "Адреса",  # K
    "ТЦК та СП",  # L
    "Відстрочка",  # M (залишаємо як є, але не впливає на боржників)
    "Бронювання",  # N (залишаємо як є, але не впливає на боржників)
    "Військова служба",  # O
    "Моб. розпорядження",  # P
    "Посада та акт",  # Q
    "Повідомлення",  # R
    "Контакти",  # T (примітка: у специфікації T – але зручно тут поряд)
    "Відмітки",  # S
]

# Очищаємо попередні дані з 8 рядка вниз (залишаємо заголовки як є або перезаписуємо)
# Перезаписуємо заголовки у 6 рядку
for col, header in enumerate(headers, start=1):
    sheet.cell(row=6, column=col, value=header)
_format_header_row(sheet, 6, len(headers))

# Записуємо числа 1..20 у 7-й рядок
for i in range(1, 21):
    sheet.cell(row=7, column=i, value=i)

# Повністю очищаємо старі рядки з 8-го, щоб не лишалися «хвости»
if sheet.max_row > 7:
    try:
        sheet.delete_rows(8, sheet.max_row – 7)
    except Exception:
        max_cols = max(len(headers), sheet.max_column or 1)
        if sheet.max_row >= 8:
            for row in range(8, sheet.max_row + 1):
                for col in range(1, max_cols + 1):
                    sheet.cell(row=row, column=col, value=None)

# Дані (усі активні для базового аркуша)
# iterator() зменшує споживання пам’яті на великих вибірках
students = Student.objects.filter(status=‘active’).iterator(chunk_size=500)
# Уникнути дублів за (ПІБ, дата народження)
seen_main = set()
unique_students_main = []
```

```
            for s in students:
                key = (getattr(s, 'full_name', None), getattr(s, 'birth_date', None))
                if key in seen_main:
                    continue
                seen_main.add(key)
                unique_students_main.append(s)

            row_num = 8
            for idx, s in enumerate(unique_students_main, start=1):
                # A №
                cell = sheet.cell(row=row_num, column=1, value=idx)
                _format_data_cell(cell, None, 'center', True)

                # B Категорія
                cell = sheet.cell(row=row_num, column=2, value=s.category or "")
                _format_data_cell(cell, None, 'center', True)

                # C Звання
                cell = sheet.cell(row=row_num, column=3, value=s.rank or "")
                _format_data_cell(cell, None, 'center', True)

                # D ПІБ (зліва)
                cell = sheet.cell(row=row_num, column=4, value=s.full_name or "")
                _format_data_cell(cell, None, 'left', True)

                # E Дата народження
                cell = sheet.cell(
                    row=row_num, column=5,
                    value=s.birth_date.strftime("%d.%m.%Y") if s.birth_date else ""
                )
                _format_data_cell(cell, None, 'center', True)

                # F Реєстр. номер
                cell = sheet.cell(row=row_num, column=6, value=s.registry_number or "")
                _format_data_cell(cell, None, 'center', True)

                # G РНОКПП
                cell = sheet.cell(row=row_num, column=7, value=s.tax_number or ("Відмова" if
getattr(s, 'refused_tax_number', False) else ""))
                _format_data_cell(cell, None, 'center', True)

                # H ВОС
                cell = sheet.cell(row=row_num, column=8, value=s.military_specialty or "")
                _format_data_cell(cell, None, 'center', True)

                # I Військ. документ
                cell = sheet.cell(row=row_num, column=9, value=s.military_document or "")
                _format_data_cell(cell, None, 'left', True)

                # J Паспорт
                cell = sheet.cell(row=row_num, column=10, value=s.passport_data or "")
                _format_data_cell(cell, None, 'left', True)
```

```
            # K Адреса
            address_text = s.get_address() if hasattr(s, 'get_address') else ""
            cell = sheet.cell(row=row_num, column=11, value=address_text or "")
            _format_data_cell(cell, None, 'left', True)

            # L ТЦК
            tcc_name = s.territorial_center.name if getattr(s, 'territorial_center', None) else ""
            cell = sheet.cell(row=row_num, column=12, value=tcc_name)
            _format_data_cell(cell, None, 'left', True)

            # M Відстрочка
            defer_str = (
               "До завершення мобілізації" if getattr(s, 'deferment_until_mobilization', False)
               else (s.deferment_date.strftime("%d.%m.%Y") if getattr(s, 'deferment_date',
None) else "")
            )
            cell = sheet.cell(row=row_num, column=13, value=defer_str)
            _format_data_cell(cell, None, 'center', True)

            # N Бронювання
            cell = sheet.cell(row=row_num, column=14,
value=(s.booking_date.strftime("%d.%m.%Y") if getattr(s, 'booking_date', None) else ""))
            _format_data_cell(cell, None, 'center', True)

            # O Військова служба
            cell = sheet.cell(row=row_num, column=15, value=("так" if getattr(s,
'military_service', False) else "ні"))
            _format_data_cell(cell, None, 'center', True)

            # P Моб. розпорядження
            mob_text = "так" if getattr(s, 'mobilization_order', False) else "ні"
            if getattr(s, 'mobilization_order_date', None):
               mob_text = f"{mob_text} ({s.mobilization_order_date.strftime('%d.%m.%Y')})"
            cell = sheet.cell(row=row_num, column=16, value=mob_text)
            _format_data_cell(cell, None, 'center', True)

            # Q Посада та акт
            pos_str = s.position or "Здобувач освіти"
            if getattr(s, 'appointment_number', None) and getattr(s, 'appointment_date', None):
               pos_str = f"{pos_str} {s.appointment_number}-с від
{s.appointment_date.strftime('%d.%m.%Y')}"
            elif getattr(s, 'appointment_number', None):
               pos_str = f"{pos_str} {s.appointment_number}-с"
            cell = sheet.cell(row=row_num, column=17, value=pos_str)
            _format_data_cell(cell, None, 'left', True)

            # R Повідомлення
            cell = sheet.cell(row=row_num, column=18, value=getattr(s, 'notification_details',
""))
            _format_data_cell(cell, None, 'left', True)
```

```
            # T Контакти (ставимо у 19 колонку)
            cell = sheet.cell(row=row_num, column=19, value=getattr(s, 'contact_info', ""))
            _format_data_cell(cell, None, 'left', True)

            # S Відмітки (20 колонка – але порядок S/T у різних шаблонах, тут показуємо після контактів)
            marks = s.get_filtered_marks() if hasattr(s, 'get_filtered_marks') else (s.marks or [])
            marks_text = ", ".join(marks) if marks else ""
            cell = sheet.cell(row=row_num, column=20, value=marks_text)
            _format_data_cell(cell, None, 'left', True)

            row_num += 1

        # Фіксовані ширини замість повільного автопідбору
        fixed_widths = [6, 18, 16, 32, 14, 18, 12, 12, 24, 22, 28, 26, 18, 14, 14, 18, 28, 24, 24, 22]
        for idx, w in enumerate(fixed_widths, start=1):
            sheet.column_dimensions[get_column_letter(idx)].width = w

        # Зберігаємо у той самий файл (можемо створити бекап)
        backup_path = os.path.join(excel_dir, f"military_registry_backup_{datetime.now().strftime('%Y%m%d_%H%M%S')}.xlsx")
        try:
            # Створюємо резервну копію існуючого файлу
            with open(file_path, 'rb') as src, open(backup_path, 'wb') as dst:
                dst.write(src.read())
        except Exception:
            pass

        # Допоміжний запис у категорійні аркуші
        def write_category_sheet(target_sheet_name: str, students_list):
            sh = wb[target_sheet_name] if target_sheet_name in wb.sheetnames else wb.create_sheet(target_sheet_name)
            # Заголовки у 6 рядку
            headers_local = headers
            for col, header in enumerate(headers_local, start=1):
                sh.cell(row=6, column=col, value=header)
            _format_header_row(sh, 6, len(headers_local))

            # Нумерація 1..20 у 7 рядку
            for i in range(1, 21):
                sh.cell(row=7, column=i, value=i)
            # Повне видалення старих рядків з 8-го, щоб уникати дублювань
            if sh.max_row > 7:
                try:
                    sh.delete_rows(8, sh.max_row - 7)
                except Exception:
                    max_cols_local = max(len(headers_local), sh.max_column or 1)
                    if sh.max_row >= 8:
                        for row in range(8, sh.max_row + 1):
                            for col in range(1, max_cols_local + 1):
                                sh.cell(row=row, column=col, value=None)
```

```
                # Запис
                row_i = 8
                # Уникнути випадкових дублікатів: унікальність за (ПІБ, дата народження)
                seen = set()
                unique_students = []
                for s in students_list:
                    key = (getattr(s, 'full_name', None), getattr(s, 'birth_date', None))
                    if key in seen:
                        continue
                    seen.add(key)
                    unique_students.append(s)

                for idx, s in enumerate(unique_students, start=1):
                    # A №
                    c = sh.cell(row=row_i, column=1, value=idx)
                    _format_data_cell(c, None, 'center', True)
                    # B Категорія
                    c = sh.cell(row=row_i, column=2, value=s.category or "")
                    _format_data_cell(c, None, 'center', True)
                    # C Звання
                    c = sh.cell(row=row_i, column=3, value=s.rank or "")
                    _format_data_cell(c, None, 'center', True)
                    # D ПІБ
                    c = sh.cell(row=row_i, column=4, value=s.full_name or "")
                    _format_data_cell(c, None, 'left', True)
                    # E Дата народження
                    c = sh.cell(row=row_i, column=5, value=s.birth_date.strftime('%d.%m.%Y') if
s.birth_date else "")
                    _format_data_cell(c, None, 'center', True)
                    # F ЄДРПВР
                    c = sh.cell(row=row_i, column=6, value=s.registry_number or "")
                    _format_data_cell(c, None, 'center', True)
                    # G РНОКПП
                    c = sh.cell(row=row_i, column=7, value=s.tax_number or ("Відмова" if
getattr(s,'refused_tax_number', False) else ""))
                    _format_data_cell(c, None, 'center', True)
                    # H ВОС
                    c = sh.cell(row=row_i, column=8, value=s.military_specialty or "")
                    _format_data_cell(c, None, 'center', True)
                    # I Військ. документ
                    c = sh.cell(row=row_i, column=9, value=s.military_document or "")
                    _format_data_cell(c, None, 'left', True)
                    # J Паспорт
                    c = sh.cell(row=row_i, column=10, value=s.passport_data or "")
                    _format_data_cell(c, None, 'left', True)
                    # K Адреса
                    addr = s.get_address() if hasattr(s,'get_address') else ""
                    c = sh.cell(row=row_i, column=11, value=addr or "")
                    _format_data_cell(c, None, 'left', True)
                    # L ТЦК
                    tcc_name = s.territorial_center.name if getattr(s,'territorial_center', None) else ""
```

```
                c = sh.cell(row=row_i, column=12, value=tcc_name)
                _format_data_cell(c, None, 'left', True)
                # M Відстрочка
                defer_str = (
                   "До завершення мобілізації" if getattr(s, 'deferment_until_mobilization', False)
                   else (s.deferment_date.strftime('%d.%m.%Y') if getattr(s,'deferment_date', None) else "")
                )
                c = sh.cell(row=row_i, column=13, value=defer_str)
                _format_data_cell(c, None, 'center', True)
                # N Бронювання
                c = sh.cell(row=row_i, column=14, value=(s.booking_date.strftime('%d.%m.%Y') if getattr(s,'booking_date', None) else ""))
                _format_data_cell(c, None, 'center', True)
                # O Військова служба
                c = sh.cell(row=row_i, column=15, value=("так" if getattr(s,'military_service', False) else "ні"))
                _format_data_cell(c, None, 'center', True)
                # P Моб. розпорядження
                mob_text = "так" if getattr(s,'mobilization_order', False) else "ні"
                if getattr(s,'mobilization_order_date', None):
                   mob_text = f"{mob_text} ({s.mobilization_order_date.strftime('%d.%m.%Y')})"
                c = sh.cell(row=row_i, column=16, value=mob_text)
                _format_data_cell(c, None, 'center', True)
                # Q Посада та акт
                pos_str = s.position or "Здобувач освіти"
                if getattr(s,'appointment_number', None) and getattr(s,'appointment_date', None):
                   pos_str = f"{pos_str} {s.appointment_number}-с від {s.appointment_date.strftime('%d.%m.%Y')}"
                elif getattr(s,'appointment_number', None):
                   pos_str = f"{pos_str} {s.appointment_number}-с"
                c = sh.cell(row=row_i, column=17, value=pos_str)
                _format_data_cell(c, None, 'left', True)
                # R Повідомлення
                c = sh.cell(row=row_i, column=18, value=getattr(s,'notification_details', ""))
                _format_data_cell(c, None, 'left', True)
                # T Контакти
                c = sh.cell(row=row_i, column=19, value=getattr(s,'contact_info', ""))
                _format_data_cell(c, None, 'left', True)
                # S Відмітки
                marks = s.get_filtered_marks() if hasattr(s,'get_filtered_marks') else (s.marks or [])
                marks_text = ", ".join(marks) if marks else ""
                c = sh.cell(row=row_i, column=20, value=marks_text)
                _format_data_cell(c, None, 'left', True)

                row_i += 1

            # Фіксовані ширини для швидкодії
```

```
            fixed_widths_local = [6, 18, 16, 32, 14, 18, 12, 12, 24, 22, 28, 26, 18, 14, 14, 18, 28,
24, 24, 22]
            for idx2, w2 in enumerate(fixed_widths_local, start=1):
                sh.column_dimensions[get_column_letter(idx2)].width = w2

        # Побудова вибірок і сортування
        def is_excluded_from_sheets(s):
            # Не розносити по аркушах: якщо у відмітках ТОТ/16 років, або Призовник
25+
            try:
                marks = set((s.marks or []))
            except Exception:
                marks = set()
            if 'ТОТ' in marks or '16 років' in marks:
                return True
            try:
                age = s.age()
            except Exception:
                age = 0
            if s.category == 'Призовник' and age >= 25:
                return True
            return False

        obliged_qs_all = Student.objects.filter(status='active',
category__in=['Військовозобов`язаний','Військовозобов`язана','Резервіст']).iterator(chunk_
size=500)
        obliged_qs = [s for s in obliged_qs_all if not is_excluded_from_sheets(s)]
        # сортування за пріоритетом звань (вищі зверху)
        obliged_list = sorted(list(obliged_qs), key=lambda s: s.get_rank_priority())

        conscripts_qs_all = Student.objects.filter(status='active',
category='Призовник').iterator(chunk_size=500)
        conscripts_qs = [s for s in conscripts_qs_all if not is_excluded_from_sheets(s)]
        conscripts_qs = sorted(conscripts_qs, key=lambda s: s.full_name)

        write_category_sheet(obliged_name, obliged_list)
        write_category_sheet(conscripts_name, list(conscripts_qs))

        wb.save(file_path)
        return file_path


    def _parse_date(value):
        if not value:
            return None
        if isinstance(value, datetime):
            return value.date()
        s = str(value).strip()
        for fmt in ("%Y-%m-%d", "%d.%m.%Y", "%d/%m/%Y"):
            try:
                return datetime.strptime(s, fmt).date()
            except Exception:
```

```
            continue
        return None


    def read_student_from_row(sheet, row_num: int, start_col: int = 1):
        """
        Зчитує дані студента з рядка Excel (структура "Додаток 5 Основа").
        Очікувані колонки з offset start_col (A=1):
          B Категорія, C Звання, D ПІБ, E Дата нар., F ЄДРПВР, G РНОКПП,
          H ВОС, I Військ. документ, J Паспорт, K Адреса, L ТЦК,
          M Відстрочка, N Бронювання, O Військ. служба, P Моб. розпорядження,
          Q Посада та акт, R Повідомлення, S Відмітки, T Контакти
        """
        import re

        def cell(col_idx):
            return sheet.cell(row=row_num, column=start_col + col_idx – 1).value

        def get_text(v):
            return ‘‘ if v is None else str(v).strip()

        data = {}
        data[‘category’] = get_text(cell(2))
        data[‘rank’] = get_text(cell(3))
        data[‘full_name’] = get_text(cell(4))
        data[‘birth_date’] = _parse_date(cell(5))
        data[‘registry_number’] = get_text(cell(6))

        # G – РНОКПП або відмова
        g_val = get_text(cell(7))
        refused = False
        if g_val:
            if ‘відмова’ in g_val.lower() or ‘відмов’ in g_val.lower():
                refused = True
                g_val = ‘‘
            else:
                # Видалити .0 з кінця, якщо це число (наприклад, 1234567890.0 ->
1234567890)
                if g_val.endswith(‘.0’):
                    g_val = g_val[:-2]
        data[‘tax_number’] = g_val
        data[‘refused_tax_number’] = refused

        data[‘military_specialty’] = get_text(cell(8))
        data[‘military_document’] = get_text(cell(9))
        data[‘passport_data’] = get_text(cell(10))

        # Адреса як суцільний текст
        address_text = get_text(cell(11))
        data[‘address’] = address_text

        data[‘territorial_center_name’] = get_text(cell(12))
```

```
            # Відстрочка
            m_val = get_text(cell(13))
            data['deferment_until_mobilization'] = False
            data['deferment_date'] = None
            if m_val:
                if 'до заверш' in m_val.lower() or 'мобілізац' in m_val.lower():
                    data['deferment_until_mobilization'] = True
                else:
                    d = _parse_date(m_val)
                    if not d:
                        # спробуємо витягти дату з тексту
                        m = re.search(r'(\d{1,2}[\./]\d{1,2}[\./]\d{4}|\d{4}-\d{2}-\d{2})', m_val)
                        if m:
                            d = _parse_date(m.group(1))
                    data['deferment_date'] = d

            # Бронювання
            n_val = get_text(cell(14))
            data['booking_date'] = _parse_date(n_val) or (
_parse_date(re.search(r'(\d{1,2}[\./]\d{1,2}[\./]\d{4}|\d{4}-\d{2}-\d{2})', n_val).group(1)) if
re.search(r'(\d{1,2}[\./]\d{1,2}[\./]\d{4}|\d{4}-\d{2}-\d{2})', n_val) else None )

            # Військова служба O
            o_val = get_text(cell(15)).lower()
            data['military_service'] = o_val.startswith('т') or o_val == 'yes' or o_val == '1'

            # Моб. розпорядження P (так/ні + опц. дата)
            p_val = get_text(cell(16))
            pv = p_val.lower()
            has_mob = pv.startswith('т') or pv == 'yes' or pv == '1'
            mob_date = _parse_date(p_val) or (
_parse_date(re.search(r'(\d{1,2}[\./]\d{1,2}[\./]\d{4}|\d{4}-\d{2}-\d{2})', p_val).group(1)) if
re.search(r'(\d{1,2}[\./]\d{1,2}[\./]\d{4}|\d{4}-\d{2}-\d{2})', p_val) else None )
            data['mobilization_order'] = has_mob
            data['mobilization_order_date'] = mob_date

            # Q Посада та акт (розширений парсинг)
            q_val = get_text(cell(17))
            data['position'] = q_val if q_val else 'Здобувач освіти'
            data['appointment_number'] = ''
            data['appointment_date'] = None

            if q_val and q_val != 'Здобувач освіти':
                # спробуємо знайти дату
                date_match = re.search(r'(\d{1,2}[\./]\d{1,2}[\./]\d{4}|\d{4}-\d{2}-\d{2})', q_val)
                found_date = _parse_date(date_match.group(1)) if date_match else None

                # спробуємо знайти номер (перед "-с" або після №)
                num_match = re.search(r'№\s*(\d+(?:-\d+)?)', q_val)
                if not num_match:
                    num_match = re.search(r'(\d+(?:-\d+)?)\s*-?\s*с\b', q_val, re.IGNORECASE)
```

```
            if not num_match:
                # просто остання група цифр
                num_match = re.search(r'(\d{2,})', q_val)

            if found_date:
                data['appointment_date'] = found_date
            if num_match:
                data['appointment_number'] = num_match.group(1).replace('-', '')

            # позиція – це початок рядка до номера/"від"
            pos_part = q_val
            if date_match:
                pos_part = pos_part[:date_match.start()].strip()
            if num_match:
                pos_part = pos_part[:num_match.start()].strip()
            if pos_part:
                data['position'] = pos_part

        # R Повідомлення
        data['notification_details'] = get_text(cell(18))

        # Якщо appointment_number не знайдено в полі Q, спробуємо витягти з notification_details
        if not data.get('appointment_number') and data.get('notification_details'):
            notification = data['notification_details']
            # Шукаємо номер наказу в форматі "Номер наказу: ****-с" або просто номер
            num_match = re.search(r'Номер наказу:\s*(\d+(?:-\d+)?)', notification, re.IGNORECASE)
            if not num_match:
                num_match = re.search(r'(\d+(?:-\d+)?)\s*-?\s*с\b', notification, re.IGNORECASE)
            if num_match:
                data['appointment_number'] = num_match.group(1).replace('-', '')

        # S Відмітки
        s_val = cell(19)
        if isinstance(s_val, list):
            marks = [str(x).strip() for x in s_val if x]
        else:
            marks = [m.strip() for m in str(s_val or '').split(',') if m.strip()]
        data['marks'] = marks

        # T Контакти
        data['contact_info'] = get_text(cell(20))

        return data


    def parse_enrollment_order(order_text):
        """
        Парсить наказ про зарахування з формату:
```

```
"Код наказу про зарахування: ******; Номер наказу: ****-с; Дата наказу: ddmm.yyyy"

Повертає словник з полями:
– code: код наказу
– number: номер наказу
– date: дата наказу (date object)
"""
import re
from datetime import datetime

if not order_text:
    return {'code': None, 'number': None, 'date': None}

order_text = str(order_text).strip()
result = {'code': None, 'number': None, 'date': None}

# Парсинг коду наказу
code_match = re.search(r'Код наказу про зарахування:\s*(\d+)', order_text, re.IGNORECASE)
if code_match:
    result['code'] = code_match.group(1)

# Парсинг номера наказу
number_match = re.search(r'Номер наказу:\s*([\d-]+(?:-с)?)', order_text, re.IGNORECASE)
if number_match:
    result['number'] = number_match.group(1)

# Парсинг дати наказу (формат ddmm.yyyy)
date_match = re.search(r'Дата наказу:\s*(\d{2})\.(\d{2})\.(\d{4})', order_text, re.IGNORECASE)
if date_match:
    try:
        day = int(date_match.group(1))
        month = int(date_match.group(2))
        year = int(date_match.group(3))
        result['date'] = datetime(year, month, day).date()
    except (ValueError, IndexError):
        pass

return result


def determine_faculty(structural_unit, full_name=None, birth_date=None, enrollment_order=None):
"""
Визначає факультет на основі структурного підрозділу та інших даних.

Args:
    structural_unit: Структурний підрозділ з колонки V2
    full_name: ПІБ студента
```

```
        birth_date: Дата народження
        enrollment_order: Текст наказу про зарахування (для парсингу номера наказу)

    Returns:
        Назва факультету або None
    """
    if not structural_unit:
        return None

    structural_unit = str(structural_unit).strip()

    # Список факультетів
    faculties = [
        'Біологічний факультет',
        'Економічний факультет',
        'Інженерний навчально-науковий інститут ім. Ю.М. Потебні Запорізького національного університету',
        'Математичний факультет',
        'Факультет журналістики',
        'Факультет іноземної філології',
        'Факультет історії та міжнародних відносин',
        'Факультет менеджменту',
        'Факультет соціальної педагогіки та психології',
        'Факультет соціології та управління',
        'Факультет фізичного виховання, здоров\'я та туризму',
        'Філологічний факультет',
        'Юридичний факультет',
    ]

    # Перевірка точного співпадіння
    for faculty in faculties:
        if faculty.lower() in structural_unit.lower():
            return faculty

    # Якщо не знайдено точного співпадіння, спробуємо знайти за ключовими словами
    structural_lower = structural_unit.lower()

    if 'біолог' in structural_lower:
        return 'Біологічний факультет'
    elif 'економ' in structural_lower:
        return 'Економічний факультет'
    elif 'інженер' in structural_lower or 'потебн' in structural_lower:
        return 'Інженерний навчально-науковий інститут ім. Ю.М. Потебні Запорізького національного університету'
    elif 'математ' in structural_lower:
        return 'Математичний факультет'
    elif 'журнал' in structural_lower:
        return 'Факультет журналістики'
    elif 'іноземн' in structural_lower or 'філолог' in structural_lower:
        if 'іноземн' in structural_lower:
            return 'Факультет іноземної філології'
```

```
            else:
                return 'Філологічний факультет'
        elif 'історі' in structural_lower or 'міжнародн' in structural_lower:
            return 'Факультет історії та міжнародних відносин'
        elif 'менеджмент' in structural_lower:
            return 'Факультет менеджменту'
        elif 'соціальн' in structural_lower and 'педагог' in structural_lower:
            return 'Факультет соціальної педагогіки та психології'
        elif 'соціолог' in structural_lower or ('соціальн' in structural_lower and 'управлін' in structural_lower):
            return 'Факультет соціології та управління'
        elif 'фізичн' in structural_lower or 'здоров' in structural_lower or 'туризм' in structural_lower:
            return 'Факультет фізичного виховання, здоров\'я та туризму'
        elif 'юрид' in structural_lower:
            return 'Юридичний факультет'

        return None


    def load_students_from_csv(file_path, degree_type='bachelor'):
        """
        Завантажує список студентів з CSV файлу.

        Args:
            file_path: Шлях до CSV файлу
            degree_type: 'bachelor' або 'master'

        Returns:
            Список словників з даними студентів
        """
        import csv

        students = []

        # Визначаємо кодування файлу
        encodings = ['utf-8', 'utf-8-sig', 'windows-1251', 'cp1251']

        for encoding in encodings:
            try:
                with open(file_path, 'r', encoding=encoding, newline='') as csvfile:
                    # Автоматично визначаємо роздільник
                    sample = csvfile.read(1024)
                    csvfile.seek(0)
                    sniffer = csv.Sniffer()
                    delimiter = sniffer.sniff(sample).delimiter

                    reader = csv.reader(csvfile, delimiter=delimiter)

                    # Пропускаємо перший рядок (заголовки)
                    next(reader, None)
```

```
                # Читаємо дані з рядка 2 (індекс 1 після пропуску заголовка)
                # F2 – ПІБ (індекс 5), G2 – Дата народження (індекс 6),
                # V2 – Структурний підрозділ (індекс 21), AY2 – Наказ про зарахування
(індекс 50)
                for row_num, row in enumerate(reader, start=2):
                    if len(row) < 51:  # Перевіряємо чи є достатньо колонок
                        continue

                    # Перевіряємо чи є ПІБ у колонці F (індекс 5)
                    full_name = row[5].strip() if len(row) > 5 and row[5] else ""
                    if not full_name:
                        continue

                    # Читаємо дані
                    birth_date_val = row[6] if len(row) > 6 else None  # G (індекс 6)
                    structural_unit = row[21] if len(row) > 21 else None  # V (індекс 21)
                    enrollment_order = row[50] if len(row) > 50 else None  # AY (індекс 50)

                    # Парсимо дату народження
                    birth_date = _parse_date(birth_date_val)

                    # Парсимо наказ про зарахування
                    enrollment_data = parse_enrollment_order(enrollment_order) if
enrollment_order else {}

                    # Визначаємо факультет
                    faculty = determine_faculty(structural_unit, full_name, birth_date,
enrollment_order)

                    student_data = {
                        'full_name': full_name,
                        'birth_date': birth_date,
                        'structural_unit': str(structural_unit).strip() if structural_unit else '',
                        'enrollment_order': str(enrollment_order).strip() if enrollment_order else
'',
                        'faculty': faculty,
                        'enrollment_code': enrollment_data.get('code'),
                        'enrollment_number': enrollment_data.get('number'),
                        'enrollment_date': enrollment_data.get('date'),
                        'degree_type': degree_type
                    }

                    students.append(student_data)

                    # Обмеження для безпеки
                    if row_num > 10000:
                        break

                break  # Якщо успішно прочитали, виходимо з циклу по кодуванням
            except (UnicodeDecodeError, csv.Error):
                continue
            except Exception as e:
```

```
            if encoding == encodings[-1]:  # Останнє кодування
                raise Exception(f"Помилка читання CSV файлу: {str(e)}")
            continue

    return students


def load_students_from_excel(file_path, degree_type='bachelor'):
    """
    Завантажує список студентів з Excel або CSV файлу.

    Args:
        file_path: Шлях до Excel або CSV файлу
        degree_type: 'bachelor' або 'master'

    Returns:
        Список словників з даними студентів:
        {
            'full_name': str,
            'birth_date': date,
            'structural_unit': str,
            'enrollment_order': str,
            'faculty': str,
            'enrollment_code': str,
            'enrollment_number': str,
            'enrollment_date': date
        }
    """
    import os

    # Визначаємо формат файлу по розширенню
    file_ext = os.path.splitext(file_path)[1].lower()

    if file_ext == '.csv':
        return load_students_from_csv(file_path, degree_type)

    # Для Excel файлів
    from openpyxl import load_workbook

    wb = load_workbook(file_path, data_only=True)
    sheet = wb.active  # Використовуємо активний аркуш

    students = []

    # Знаходимо перший рядок з даними (починаючи з рядка 2)
    # F2 – ПІБ, G2 – Дата народження, V2 – Структурний підрозділ, AY2 – Наказ про зарахування
    row = 2

    while True:
        # Перевіряємо чи є ПІБ у колонці F
        full_name = sheet.cell(row=row, column=6).value  # F = 6
```

```
            if not full_name or str(full_name).strip() == "":
                # Якщо немає ПІБ, перевіряємо ще кілька рядків нижче
                if row > 1000:  # Обмеження для безпеки
                    break
                row += 1
                continue

            # Читаємо дані
            full_name = str(full_name).strip()
            birth_date_val = sheet.cell(row=row, column=7).value  # G = 7
            structural_unit = sheet.cell(row=row, column=22).value  # V = 22
            enrollment_order = sheet.cell(row=row, column=51).value  # AY = 51

            # Парсимо дату народження
            birth_date = _parse_date(birth_date_val)

            # Парсимо наказ про зарахування
            enrollment_data = parse_enrollment_order(enrollment_order) if enrollment_order
else {}

            # Визначаємо факультет
            faculty = determine_faculty(structural_unit, full_name, birth_date,
enrollment_order)

            student_data = {
                'full_name': full_name,
                'birth_date': birth_date,
                'structural_unit': str(structural_unit).strip() if structural_unit else '',
                'enrollment_order': str(enrollment_order).strip() if enrollment_order else '',
                'faculty': faculty,
                'enrollment_code': enrollment_data.get('code'),
                'enrollment_number': enrollment_data.get('number'),
                'enrollment_date': enrollment_data.get('date'),
                'degree_type': degree_type
            }

            students.append(student_data)
            row += 1

            # Обмеження для безпеки
            if row > 10000:
                break

        return students
```

ДОДАТОК Г

Security.py

```
from django.core.cache import cache
from django.utils import timezone
from datetime import timedelta
import logging
logger = logging.getLogger('django.security')

# Налаштування rate limiting
MAX_LOGIN_ATTEMPTS = 5  # Максимальна кількість спроб входу
LOCKOUT_TIME = 15 * 60  # Час блокування в секундах (15 хвилин)
RATE_LIMIT_WINDOW = 60  # Вікно для rate limiting (60 секунд)
RATE_LIMIT_MAX_REQUESTS = 30  # Максимальна кількість запитів за вікно


def get_client_ip(request):
    """Отримати IP адресу клієнта"""
    x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
    if x_forwarded_for:
        ip = x_forwarded_for.split(',')[0]
    else:
        ip = request.META.get('REMOTE_ADDR')
    return ip


def is_ip_blocked(ip_address):
    """Перевірити чи IP адреса заблокована"""
    cache_key = f'blocked_ip_{ip_address}'
    return cache.get(cache_key) is not None


def block_ip(ip_address, duration=LOCKOUT_TIME):
    """Заблокувати IP адресу на певний час"""
    cache_key = f'blocked_ip_{ip_address}'
    cache.set(cache_key, True, duration)
    logger.warning(f'IP адреса {ip_address} заблокована на {duration} секунд')


def get_login_attempts(username):
    """Отримати кількість невдалих спроб входу для користувача"""
    cache_key = f'login_attempts_{username}'
    return cache.get(cache_key, 0)


def increment_login_attempts(username):
    """Збільшити лічильник невдалих спроб входу"""
    cache_key = f'login_attempts_{username}'
    attempts = get_login_attempts(username) + 1
```

```
        cache.set(cache_key, attempts, LOCKOUT_TIME)

        if attempts >= MAX_LOGIN_ATTEMPTS:
            logger.warning(f'Користувач {username} заблокований після {attempts} невдалих спроб входу')
            return True  # Користувач заблокований
        return False


    def reset_login_attempts(username):
        """Скинути лічильник невдалих спроб входу"""
        cache_key = f'login_attempts_{username}'
        cache.delete(cache_key)


    def check_rate_limit(ip_address):
        """Перевірити rate limiting для IP адреси"""
        cache_key = f'rate_limit_{ip_address}'
        requests = cache.get(cache_key, 0)

        if requests >= RATE_LIMIT_MAX_REQUESTS:
            logger.warning(f'Rate limit перевищено для IP {ip_address}')
            return False

        cache.set(cache_key, requests + 1, RATE_LIMIT_WINDOW)
        return True


    def log_security_event(event_type, user, ip_address, details=''):
        """Логувати подію безпеки"""
        user_info = f'{user.username} ({user.get_full_name()})' if user.is_authenticated else 'Anonymous'
        message = f'{event_type} – User: {user_info}, IP: {ip_address}'
        if details:
            message += f', Details: {details}'
        logger.info(message)


    def log_user_action(action, user, object_type=None, object_id=None, details=''):
        """Логувати дію користувача для audit log"""
        if not user.is_authenticated:
            return

        log_data = {
            'action': action,
            'user': f'{user.username} ({user.get_full_name()})',
            'user_id': user.id,
            'timestamp': timezone.now().isoformat(),
        }

        if object_type:
            log_data['object_type'] = object_type
```

```
        if object_id:
            log_data['object_id'] = object_id
        if details:
            log_data['details'] = details

logger.info(f'AUDIT: {log_data}')
```

ДОДАТОК Д

Forms.py

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from .models import Student, TerritorialCenter, CustomMark, CustomUser


class StudentForm(forms.ModelForm):
    tcc_name = forms.CharField(required=False, label='ТЦК та СП',
widget=forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Почніть вводити назву ТЦК...'}))
    class Meta:
        model = Student
        fields = [
            'category', 'rank', 'full_name', 'birth_date',
            'registry_number', 'tax_number', 'refused_tax_number',
            'military_specialty', 'military_document', 'passport_data',
            'address',
            # 'territorial_center' – керуємо через tcc_name
            'deferment_date', 'deferment_until_mobilization',
            'booking_date',
            'military_service', 'mobilization_order', 'mobilization_order_date',
            'position', 'appointment_number', 'appointment_date',
            'notification_details', 'contact_info', 'marks'
        ]
        widgets = {
            'birth_date': forms.DateInput(attrs={
                'type': 'date',
                'class': 'form-control'
            }, format='%Y-%m-%d'),
            'deferment_date': forms.DateInput(attrs={
                'type': 'date',
                'class': 'form-control'
            }, format='%Y-%m-%d'),
            'booking_date': forms.DateInput(attrs={
                'type': 'date',
                'class': 'form-control'
            }, format='%Y-%m-%d'),
            'mobilization_order_date': forms.DateInput(attrs={
                'type': 'date',
                'class': 'form-control'
            }, format='%Y-%m-%d'),
            'appointment_date': forms.DateInput(attrs={
                'type': 'date',
                'class': 'form-control'
```

```
            }, format='%Y-%m-%d'),
            'category': forms.Select(attrs={'class': 'form-control'}),
            'rank': forms.Select(attrs={'class': 'form-control'}),
            'full_name': forms.TextInput(attrs={'class': 'form-control'}),
            'registry_number': forms.TextInput(attrs={'class': 'form-control'}),
            'tax_number': forms.TextInput(attrs={'class': 'form-control', 'maxlength':
'10'}),
            'refused_tax_number': forms.CheckboxInput(attrs={'class': 'form-check-
input'}),
            'military_specialty': forms.TextInput(attrs={'class': 'form-control'}),
            'military_document': forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),
            'passport_data': forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),
            'address': forms.Textarea(attrs={'class': 'form-control', 'rows': 3, 'placeholder':
'Область, місто, вулиця, будинок, квартира'}),
            # прихований, керується через tcc_name при збереженні
            # 'territorial_center': forms.HiddenInput(),
            'deferment_until_mobilization': forms.CheckboxInput(attrs={'class': 'form-
check-input'}),
            'military_service': forms.CheckboxInput(attrs={'class': 'form-check-input'}),
            'mobilization_order': forms.CheckboxInput(attrs={'class': 'form-check-input'}),
            'position': forms.TextInput(attrs={
                'class': 'form-control',
                'placeholder': 'Здобувач освіти'
            }),
            'appointment_number': forms.TextInput(attrs={'class': 'form-control'}),
            'notification_details': forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),
            'contact_info': forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),
            'marks': forms.CheckboxSelectMultiple(),
        }
        labels = {
            'category': "Категорія військового обов'язку *",
            'rank': "Військове звання",
            'full_name': "ПІБ *",
            'birth_date': "Дата народження *",
            'registry_number': "Реєстраційний номер ЄДРПВР",
            'tax_number': "РНОКПП",
            'refused_tax_number': "Відмовився від РНОКПП",
            'military_specialty': "Військово-облікова спеціальність",
            'military_document': "Реквізити військово-облікового документа",
            'passport_data': "Реквізити паспорта",
            'address': "Адреса",
            # 'territorial_center': "ТЦК та СП",
            'deferment_date': "Відстрочка до",
            'deferment_until_mobilization': "Відстрочка до завершення мобілізації",
            'booking_date': "Бронювання до",
            'military_service': "Проходження військової служби",
            'mobilization_order': "Наявність мобілізаційного розпорядження",
            'mobilization_order_date': "Дата видачі мобілізаційного розпорядження",
            'position': "Посада",
            'appointment_number': "Номер акта",
            'appointment_date': "Дата акта",
            'notification_details': "Реквізити повідомлення",
            'contact_info': "Контактні дані",
```

```
            'marks': "Відмітки",
        }

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # Ініціалізація tcc_name з пов'язаної моделі
        if self.instance and getattr(self.instance, 'territorial_center', None):
            self.fields['tcc_name'].initial = self.instance.territorial_center.name

        # Встановити значення за замовчуванням для посади, якщо поле порожнє
        if not self.instance.pk or not self.instance.position:
            self.fields['position'].initial = "Здобувач освіти"
        elif self.instance.position == "Здобувач":
            # Виправлення старого значення
            self.instance.position = "Здобувач освіти"
            self.fields['position'].initial = "Здобувач освіти"

        # Динамічні відмітки
        all_marks = list(Student.MARK_CHOICES)
        custom_marks = CustomMark.objects.all()
        for mark in custom_marks:
            all_marks.append((mark.name, mark.name))

        self.fields['marks'] = forms.MultipleChoiceField(
            choices=all_marks,
            widget=forms.CheckboxSelectMultiple(),
            required=False,
            label="Відмітки"
        )

        # Переконатися що формат дат правильний для HTML5 date input
        from datetime import date as date_type

        date_fields = ['birth_date', 'deferment_date', 'booking_date',
                       'mobilization_order_date', 'appointment_date']
        for field_name in date_fields:
            if field_name in self.fields:
                field = self.fields[field_name]
                # Встановлюємо формат якщо ще не встановлено
                if not hasattr(field.widget, 'format') or not field.widget.format:
                    field.widget.format = '%Y-%m-%d'
                # Встановлюємо input_formats для правильного парсингу
                field.input_formats = ['%Y-%m-%d', '%d.%m.%Y', '%d/%m/%Y']

                # Якщо є instance з датою, встановлюємо значення для відображення
                if self.instance and hasattr(self.instance, field_name):
                    date_value = getattr(self.instance, field_name, None)
                    if date_value and isinstance(date_value, date_type):
                        # Встановлюємо initial для правильного відображення в HTML5 date
input
                        field.initial = date_value.strftime('%Y-%m-%d')
                        # Також встановлюємо значення в widget
                        if hasattr(field.widget, 'attrs'):
```

```
                field.widget.attrs['value'] = date_value.strftime('%Y-%m-%d')

        def clean(self):
            cleaned_data = super().clean()
            category = cleaned_data.get('category')
            rank = cleaned_data.get('rank')

            # Для військовозобов'язаних обов'язкове звання
            if category in ['Військовозобов`язаний', 'Військовозобов`язана', 'Резервіст']:
                if not rank:
                    self.add_error('rank', 'Для військовозобов\'язаних обов\'язкове військове звання')

            return cleaned_data

        def save(self, commit=True):
            instance = super().save(commit=False)
            name = (self.cleaned_data.get('tcc_name') or '').strip()
            if name:
                tcc, _ = TerritorialCenter.objects.get_or_create(name=name)
                instance.territorial_center = tcc
            else:
                instance.territorial_center = None
            if commit:
                instance.save()
                self.save_m2m()
            return instance


    class TerritorialCenterForm(forms.ModelForm):
        class Meta:
            model = TerritorialCenter
            fields = ['name']
            widgets = {
                'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Назва ТЦК та СП'}),
            }
            labels = {
                'name': 'Назва ТЦК та СП',
            }


    class CustomMarkForm(forms.ModelForm):
        class Meta:
            model = CustomMark
            fields = ['name']
            widgets = {
                'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Назва відмітки'}),
            }
            labels = {
                'name': 'Назва відмітки',
            }
```

```
class CustomUserForm(forms.ModelForm):
    password = forms.CharField(
        required=False,
        widget=forms.PasswordInput(attrs={'class': 'form-control'}),
        label='Пароль',
        help_text='Залиште порожнім, якщо не хочете змінювати пароль'
    )
    password_confirm = forms.CharField(
        required=False,
        widget=forms.PasswordInput(attrs={'class': 'form-control'}),
        label='Підтвердження пароля'
    )

    class Meta:
        model = CustomUser
        fields = ['username', 'first_name', 'last_name', 'email', 'role', 'is_active']
        widgets = {
            'username': forms.TextInput(attrs={'class': 'form-control'}),
            'first_name': forms.TextInput(attrs={'class': 'form-control'}),
            'last_name': forms.TextInput(attrs={'class': 'form-control'}),
            'email': forms.EmailInput(attrs={'class': 'form-control'}),
            'role': forms.Select(attrs={'class': 'form-control'}),
            'is_active': forms.CheckboxInput(attrs={'class': 'form-check-input'}),
        }
        labels = {
            'username': 'Ім\'я користувача',
            'first_name': 'Ім\'я',
            'last_name': 'Прізвище',
            'email': 'Email',
            'role': 'Роль',
            'is_active': 'Активний',
        }

    def clean(self):
        cleaned_data = super().clean()
        password = cleaned_data.get('password')
        password_confirm = cleaned_data.get('password_confirm')
        if password and password != password_confirm:
            self.add_error('password_confirm', 'Паролі не співпадають')

        return cleaned_data

    def save(self, commit=True):
        user = super().save(commit=False)
        password = self.cleaned_data.get('password')

        if password:
            user.set_password(password)
        elif not user.pk:
            # При створенні нового користувача пароль обов'язковий
```

```
                raise forms.ValidationError({‘password’: ‘Пароль обов\’язковий при створенні нового користувача’})

            if commit:
                user.save()

            return user


        class CustomUserCreateForm(UserCreationForm):
            first_name = forms.CharField(max_length=150, required=True, label="Ім’я", widget=forms.TextInput(attrs={‘class’: ‘form-control’}))
            last_name = forms.CharField(max_length=150, required=True, label=‘Прізвище’, widget=forms.TextInput(attrs={‘class’: ‘form-control’}))
            email = forms.EmailField(required=True, label=‘Email’, widget=forms.EmailInput(attrs={‘class’: ‘form-control’}))
            role = forms.ChoiceField(choices=CustomUser.ROLE_CHOICES, required=True, label=‘Роль’, widget=forms.Select(attrs={‘class’: ‘form-control’}))

            class Meta:
                model = CustomUser
                fields = [‘username’, ‘first_name’, ‘last_name’, ‘email’, ‘role’, ‘password1’, ‘password2’]
                widgets = {
                    ‘username’: forms.TextInput(attrs={‘class’: ‘form-control’}),
                    ‘password1’: forms.PasswordInput(attrs={‘class’: ‘form-control’, ‘autocomplete’: ‘new-password’, ‘placeholder’: ‘Введіть пароль’}),
                    ‘password2’: forms.PasswordInput(attrs={‘class’: ‘form-control’, ‘autocomplete’: ‘new-password’, ‘placeholder’: ‘Підтвердіть пароль’}),
                }
                labels = {
                    ‘username’: ‘Ім\’я користувача’,
                    ‘password1’: ‘Пароль’,
                    ‘password2’: ‘Підтвердження пароля’,
                }
                help_texts = {
                    ‘username’: None,
                    ‘password1’: None,  # Прибираємо стандартний допоміжний текст
                    ‘password2’: None,  # Прибираємо стандартний допоміжний текст
                }

            def __init__(self, *args, **kwargs):
                super().__init__(*args, **kwargs)
                # Прибрати допоміжний текст з поля username
                if ‘username’ in self.fields:
                    self.fields[‘username’].help_text = None

                # Прибрати стандартний допоміжний текст з полів паролю
                if ‘password1’ in self.fields:
                    self.fields[‘password1’].help_text = None
                if ‘password2’ in self.fields:
                self.fields[‘password2’].help_text = No
```