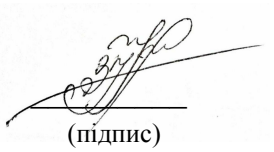
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

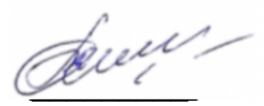
Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

СИСТЕМА УПРАВЛІННЯ НАПОВНЕННЯМ ОСВІТНІХ ПРОГРАМ

Виконала студентка 4-го курсу
Наталія ЗАХАРЧУК


(підпис)

Науковий керівник:
доцент, кандидат технічних наук
Олексій ТКАЧЕНКО


(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань

Студент


(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«05» червня 2023р.,
протокол No 18
Завідувач кафедри
Микола НІКІТЧЕНКО _____
(підпис)

Київ-2023

РЕФЕРАТ

Обсяг роботи 57 сторінок, 52 ілюстрації, 1 таблиця, 13 джерел посилань, 2 додатки.

АДМІНІСТРАТИВНА ПАНЕЛЬ, DJANGO ФРЕЙМВОРК, МОВА ПРОГРАМУВАННЯ PYTHON, ОСВІТНІ ПРОГРАМИ, ПАРСЕР, СИСТЕМА УПРАВЛІННЯ, ТЕСТУВАННЯ

Об’єктом дослідження є освітні програми для закладів вищої освіти. Предметом дослідження є система управління наповненням освітніх програм.

Метою роботи є розробка системи управління освітніми програмами зі зручним та корисним інтерфейсом для користувачів.

Методами та інструментами розроблення є мова програмування Python та фреймворк Django для реалізації системи, використання парсера для отримання та обробки інформації про освітні програми та використання адміністративної панелі Django для керування базою даних.

Результатом виконання проекту є безпечна та надійна система, яка відповідає потребам управління освітніми програмами і сприяє зручному управлінню навчальним процесом.

Новизною є те, що створена система управління наповненням освітніх програм використовує парсер, який дозволяє автоматизувати отримання та оновлення даних про освітні програми.

Інформація щодо впровадження: розроблена система може бути впроваджена в навчальні заклади для полегшення управління освітніми програмами.

Сфера застосування: розроблена система може бути використана в університетах, коледжах та інших навчальних закладах для управління освітніми програмами.

Значимість роботи: розроблена система сприяє полегшенню управління освітніми програмами, забезпечує зручний доступ до інформації та автоматизує процес оновлення даних.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

CRUD — Create, Read, Update, Delete;

CSV — Comma-Separated Values;

HTML — HyperText Markup Language;

HTTP — HyperText Transfer Protocol;

JSON — JavaScript Object Notation;

MVC — Model-View-Controller;

MVT — Model-View-Template;

ORM — Object-Relational Mapping;

SOLID — Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion;

SQL — Structured Query Language;

URL — Uniform Resource Locator.

ВСТУП

Оцінка сучасного стану об’єкта дослідження. В сучасному освітньому середовищі, де відбуваються постійні зміни та розвиток, зручне управління освітніми програмами викликає все більше зацікавлення та потребу. Забезпечення високої якості освіти, оновлення навчальних матеріалів та моніторинг освітнього процесу стають ключовими факторами для успіху навчальних закладів та викладачів.

Після проведення оцінки сучасного стану системи управління освітніми програмами, було виявлено декілька ключових аспектів, які варто врахувати.

В першу чергу, було проаналізовано існуючі системи управління, доступні на ринку. Виявлено, що деякі з них мають обмежену функціональність та є складні у використанні, що поскладнює ефективне управління освітніми програмами. Було виявлено потребу в розробці більш простої та зручної системи, яка задовольнятиме потреби викладачів та адміністраторів.

Другим аспектом, який було враховано, є проблеми та виклики, з якими зіштовхуються користувачі системи управління. Було виявлено, що існуючі рішення не забезпечують достатньої гнучкості та можливостей для адаптації до змін у вимогах освітнього процесу. Це обмежує можливості ефективного управління освітніми програмами та потребує подальших вдосконалень.

Актуальність роботи та підстави для її виконання. Дана робота присвячена дослідженням структури наповнення навчальних програм та спрямована на автоматизацію процесу наповнення освітньої програми. Актуальність теми полягає в тому, що з розвитком технологій, навчальні програми в закладах вищої освіти потребують постійного підтримання актуальності використовуваних технологій та інформації. Це сприяє на необхідність постійного оновлення тем викладачами, що своєю чергою потребує часових затрат.

Мета й завдання роботи. Метою роботи є розробка нової системи управління освітніми програмами, яка буде відповідати сучасним вимогам та надавати широкий функціонал для зручного управління.

Для досягнення зазначеної мети, поставлено такі задачі:

- вивчення та аналіз чинних структури навчальних програм;
- схематична побудова бази даних, створення діаграм;
- створення бази даних та запитів для користування нею;
- створення адміністративної панелі для зручного використання системи;
- створення парсерів для збору інформації з вже існуючих навчальних програм;
- реалізація додаткових можливостей, таких як імпорт даних у файли (файли з розширенням .xlsx, .csv, .txt тощо).

Об'єкт, предмет і методи дослідження або розроблення. Об'єктом дослідження є освітні програми для закладів вищої освіти. Предметом дослідження є система управління освітніми програмами. Методи дослідження включають аналіз сучасних підходів до управління освітніми програмами, вивчення потреб користувачів та розробку відповідних функціональних можливостей системи. В процесі розробки будуть використовуватися методи аналізу, моделювання, проектування та програмування. Будуть застосовані сучасні технології, такі як мова програмування Python та фреймворк Django, для реалізації системи управління. Також планується використання парсерів для отримання та обробки інформації з джерел зовнішніх систем.

Можливі сфери застосування. Розроблена система управління освітніми програмами може знайти застосування в закладах освіти, де необхідно дієво організувати та управляти освітніми програмами. Вона може бути використана у вищих навчальних закладах, навчальних центрах та інших освітніх установах.

Взаємозв’язок з іншими роботами. Розроблена система управління освітніми програмами може мати взаємозв'язок з іншими роботами, спрямованими на автоматизацію та покращення освітнього процесу. Вона може бути інтегрована з іншими системами, які використовуються в освітній сфері,

такими як системи електронного навчання або платформи для навчання на відстані.

Методологічна основа. Методологічною основою для дослідження послужили освітні програми викладачів факультету комп’ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Основні поняття та терміни

Освітня програма — це система освітніх компонентів на відповідному рівні вищої освіти в межах спеціальності, що визначає вимоги до рівня освіти осіб, які навчатимуться за цією програмою, перелік навчальних дисциплін і логічну послідовність їхнього вивчення, кількість кредитів, потрібних для виконання цієї програми, а також очікувані результати навчання, які повинен опанувати здобувач відповідного ступеня вищої освіти [1].

Компонентами, які наповнюють освітньо-професійну програму, є: предмети вивчення, дисципліни, контрольні заходи — вони є обов’язковими для кожного закладу вищої освіти. Проте, структуру робочої програми учбові заклади узгоджують індивідуально. З даних міркувань, було зосереджено увагу на дослідженні програми навчальних дисциплін факультету комп’ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка.

Навчальний план складається з двох блоків: обов’язкові дисципліни та вибіркові компоненти. Розглянемо кожен з блоків окремо. [2]

Обов’язкові дисципліни — це таблиця з кодом дисципліни, назвою, та семестром, в якому вона викладається.

Обов'язкові дисципліни									
Код	Назва	1 сем	2 сем	3 сем	4 сем	5 сем	6 сем	7 сем	8 сем
ННД.19	Системне програмування						4		

Рисунок 1.1 — Зразок таблиці обов’язкових дисциплін

Вибіркові компоненти складаються з двох частин:

1) вибіркові блоки — блок дисциплін, які обираються на другому курсі. Це дисципліни, які здобувачі освіти планують поглиблено вивчати, починаючи з третього курсу, що і є розподілом всіх студентів на відповідні кафедри.

Кожен блок має свою назву та складається з дисциплін — таблиця з кодом дисципліни, назвою, та семестром, в якому вона викладається;

2) вибіркові переліки — навчальні дисципліни, які подаються у вигляді груп з двох дисциплін на вибір здобувачам освіти. Тобто, пропонується дві дисципліни, серед яких потрібно обрати лише одну. Кожен перелік має відповідний номер та дві дисципліни.

а)

Вибірковий блок "Теорія та технологія програмування"									
Код	**Назва**	**1 сем**	**2 сем**	**3 сем**	**4 сем**	**5 сем**	**6 сем**	**7 сем**	**8 сем**
ДВС.2.01	WEB-технології					3			

б)

Вибірковий перелік 1									
Код	**Назва**	**1 сем**	**2 сем**	**3 сем**	**4 сем**	**5 сем**	**6 сем**	**7 сем**	**8 сем**
ДВС.4.02.01	Комп'ютерне моделювання та обчислювальний експеримент					3			
ДВС.4.02.02	Чисельні методи в інженерних дослідженнях					3			

Рисунок 1.2 — Вибіркові компоненти: а — вибіркові блоки; б — вибіркові переліки

Управління освітніми програмами також включає стратегічне планування та управління змінами. Освітні заклади повинні бути готові до змін в навчальних потребах студентів, вимогах ринку праці, технологічних інноваціях, законодавстві та інших факторах. Освітні програми повинні бути гнучкими, щоб адаптуватися до цих змін, але також стабільними, щоб забезпечити якісне навчання.

Стратегічне планування також включає в себе постійний моніторинг та оцінювання виконання стратегії, щоб виявити та виправити будь-які відхилення від плану. Це важливо, оскільки обставини можуть змінюватися, і освітня програма повинна бути гнучкою, щоб адаптуватися до цих змін.

1.2 Сучасні підходи до створення та оновлення освітніх програм

Розглянемо сучасні підходи до створення та оновлення освітніх програм, якими користуються українські та європейські заклади надання освіти. Важливим фактором є те, що освітні програми повинні бути динамічними та

адаптивними, щоб відповідати змінам у суспільстві, технологіях та потребах ринку праці.

Сучасні підходи до створення та оновлення освітніх програм включають такі аспекти:

- застосування компетентнісного підходу: сучасні освітні програми використовують компетентнісний підхід, який спрямований на роботу з інформацією та опанування учнями компетентностей, умінь і навичок, які допомагають їм бути успішними, конкурентними та цінними на ринку праці. Він створює більш гнучку і адаптивну освітню програму, яка може краще відповідати потребам студентів та змінам на ринку праці [3];
- інтеграція технологій у навчальний процес: технології можуть використовуватися для підтримки навчання, оцінювання та співпраці, а також для створення та оновлення освітніх програм. Це може включати в себе використання навчальних платформ, цифрових інструментів для співпраці, аналітичних інструментів для відстеження та аналізу виконання студентів, та інше;
- створення гнучких навчальних планів: гнучкість є ключовою характеристикою сучасних освітніх програм. Вони повинні бути здатні адаптуватися до індивідуальних потреб студентів, змін на ринку праці, та нових наукових досліджень та відкриттів. Це може включати в себе можливість для студентів вибирати окремі курси або спеціалізації, а також можливість для викладачів оновлювати курси та матеріали для відповіді на нові розвитки в їх галузі [4];
- використання даних для прийняття рішень: сучасні освітні установи використовують дані для інформування процесу розробки та оновлення освітніх програм. Це може включати аналіз студентського виконання, зворотний зв'язок від студентів та викладачів, аналіз тенденцій на ринку праці та інше.

РОЗДІЛ 2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ

Дуже важливо вивчити вже існуючі системи, які можуть допомогти у створенні застосунку наповнення освітніх програм. По-перше, це пов’язане з тим, що не варто робити те, що вже існує. Людський ресурс та час можна витратити на іншу діяльність, якщо дана робота не принесе результату та користі. По-друге, вивчення та аналіз конкурентів та схожих систем дає змогу зрозуміти їхні переваги та недоліки, а також отримати уявлення про те, які функції можуть бути корисними.

Першим кроком у аналізі існуючих систем є вибір систем для дослідження. Візьмемо дві схожі та найпопулярніші системи для проведення порівняння та аналізу. Систем, які займаються саме наповненням освітніх програм не було знайдено, тому буде досліджено системи управління навчальними електронними курсами, оскільки вони мають схожу тематику та ту ж саму галузь. Провівши пошук мережею Інтернет та дослідивши кількість користувачів різних систем обрано системи управління електронними навчальними курсами Moodle та Blackboard. Розглянемо більш детально кожну з них, їх особливості, переваги та недоліки. І як результат, підсумуємо, які функції варто використати для реалізації програми в даній роботі.

Moodle (Modular Object-Oriented Dynamic Learning Environment) — модульне об’єктно-орієнтоване динамічне навчальне середовище, яке називають також системою управління курсами, віртуальним навчальним середовищем або просто платформою для навчання, яка надає викладачам, учням та адміністраторам великий набір інструментів для комп’ютеризованого навчання, в тому числі дистанційного [5].

Moodle є однією з найпопулярніших відкритих систем управління навчанням, яка використовується в освітніх установах по всьому світу. Це мультикультурна платформа, яка підтримує багато мов [6].

Moodle надає можливість створювати та управляти курсами, додавати ресурси та діяльності, оцінювати роботу студентів, стежити за їх прогресом,

налаштовувати відкриті багатокористувацькі режими тощо. Також в Moodle є інструменти для співпраці та комунікації, такі як форуми, чати, вікі, бази даних, опитування та інше.

Особливості:

- відкритий код: Moodle є open-source платформою, що дозволяє користувачам модифікувати та налаштовувати систему відповідно до їхніх потреб;
- багатомовність: Moodle підтримує багато мов, що робить його ідеальним для міжнародних освітніх установ.

Переваги:

- гнучкість та адаптивність: Moodle можна налаштувати під конкретні потреби освітніх установ;
- безкоштовність: Moodle є безкоштовною для використання платформою.

Недоліки:

- складність: Moodle може бути складною для користувачів без технічного досвіду;
- потреба в підтримці: через відкритий код, Moodle потребує технічної підтримки для установки, налаштування та обслуговування.

Blackboard — ще одна масштабна система управління навчанням, яка включає функції для управління освітніми програмами.

Blackboard є комерційною системою управління навчанням, яка надає обширний набір інструментів для організації навчального процесу. Вона надає можливість створювати курси, розміщувати матеріали для навчання, стежити за успіхами студентів, налаштовувати оцінювання, проводити онлайн-тести та опитування, спілкуватися зі студентами через форуми та чати, і багато іншого.

Blackboard включає також модулі для синхронного навчання, мобільного навчання, управління контентом, систему звітів та аналітики тощо. Однак, у порівнянні з Moodle, Blackboard може вимагати більше ресурсів для встановлення та утримання, а також має вартість ліцензії.

Особливості:

- навчальні інструменти: Blackboard включає різноманітні інструменти для навчання, включаючи тести, опитування, відеоконференції тощо;
- мобільне навчання: Blackboard має мобільну версію, що дозволяє студентам навчатися з будь-якого місця.

Переваги:

- інтеграція: легка інтеграція з іншими системами, включаючи системи управління навчанням та контентом.

Недоліки:

- вартість: в порівнянні з безкоштовною системою Moodle, Blackboard є платною, що може бути проблемою для деяких освітніх установ;
- необхідність технічної підтримки: як і Moodle, Blackboard також потребує технічної підтримки для установки, налаштування та обслуговування.

В обох системах є можливість управляти освітніми програмами, включаючи створення і оновлення навчального матеріалу, визначення послідовності вивчення матеріалу, керування оцінюванням та відстеження прогресу студентів.

Але, детально вивчивши вищевказані системи управління навчанням, можна зробити висновок, що вони — непрямі конкуренти, оскільки саме створення системи наповнення освітніх програм не є їх завданням, це системи управління електронними навчальними курсами. Варто зазначити, що реалізація управління контентом може стати прикладом для даної роботи. Обидві системи, Moodle та Blackboard, пропонують розширені можливості для управління навчальним контентом. Інтерпретуючи це під специфіку даної роботи, управління контентом має включати створення, завантаження, редагування та видалення навчальних планів.

РОЗДІЛ 3 ПРОЕКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ УПРАВЛІННЯ ОСВІТНІМИ ПРОГРАМАМИ

3.1 Вимоги та функціональність системи

3.3.1 Вимоги до системи в цілому

В системі управління освітніми програмами, основна увага буде приділена розробці простого та корисного інтерфейсу для користувача. Система буде включати основну функціональну систему — адміністративну.

Адміністративна систем**а** призначена для збереження, обробки та оновлення інформації, представленої в системі. Вона буде включати дані про освітні програми, дисципліни та іншу пов'язану інформацію.

Система має бути досить гнучкою, щоб враховувати потреби різних користувачів, в той же час забезпечуючи високий рівень безпеки та надійності.

3.1.2 Вимоги до функцій, які виконуються системою

Таблиця 1 — Перелік функцій, задач що підлягають автоматизації для системи адміністратора

Функція	**Задача**
Керувати базою даних системи управління наповненням освітніх програм	Наповнення, редагування та видалення інформації про наповнення систем
	Експортувати та імпортувати програми до бази даних

	Запуск парсингу системи факультету
	Заповнення бази даних інформацією, отриманою з парсингу
Робота з інформацією в адміністративній панелі	Керування наповненням (створення, редагування та видалення) бази даних через адміністративну панель
	Обробка інформації: фільтрування, сортування та пошук інформації по певним критеріям
	Перегляд та експорт навчальної системи у файл
	Створення нових користувачів системи з адміністраторськими правами та доступами

3.2 Архітектура та технології реалізації системи

Система управління освітніми програмами буде реалізована на основі трирівневої архітектури, що включає рівні даних, бізнес-логіки та представлення.

1) Рівень даних. На цьому рівні буде розташована база даних, що зберігає всю інформацію, необхідну для роботи системи. Для реалізації цього рівня планується використовувати PostgreSQL, відкритий реляційний СУБД, що надає повний набір функцій, необхідних для зберігання та обробки даних [7]. Для безпосередньої роботи з СУБД використовується сервіс ElephantSQL, оскільки

він легкий у використанні та його безкоштовного плану буде достатньо для заповнення бази даних усіма необхідними даними [8].

2) Рівень бізнес-логіки. На цьому рівні розташовані серверні компоненти, що реалізують основну бізнес-логіку системи. Для реалізації цього рівня планується використовувати мову програмування Python та фреймворк Django, що надає широкий набір інструментів для розробки веб-застосунків [9], [10].

3) Рівень представлення. На цьому рівні розташовані клієнтські компоненти системи, які забезпечують взаємодію користувача з системою. Для цього використовується адміністративна панель Django, налаштована за допомогою бібліотеки jazzmin, яка дозволяє створювати кастомізовані, інтуїтивно зрозумілі та ефективні інтерфейси [11].

В якості додаткових технологій для розробки системи планується використовувати систему контролю версій Git для відслідковування змін у коді.

Зобразимо логіки виконання програми за допомогою діаграми сценаріїв:

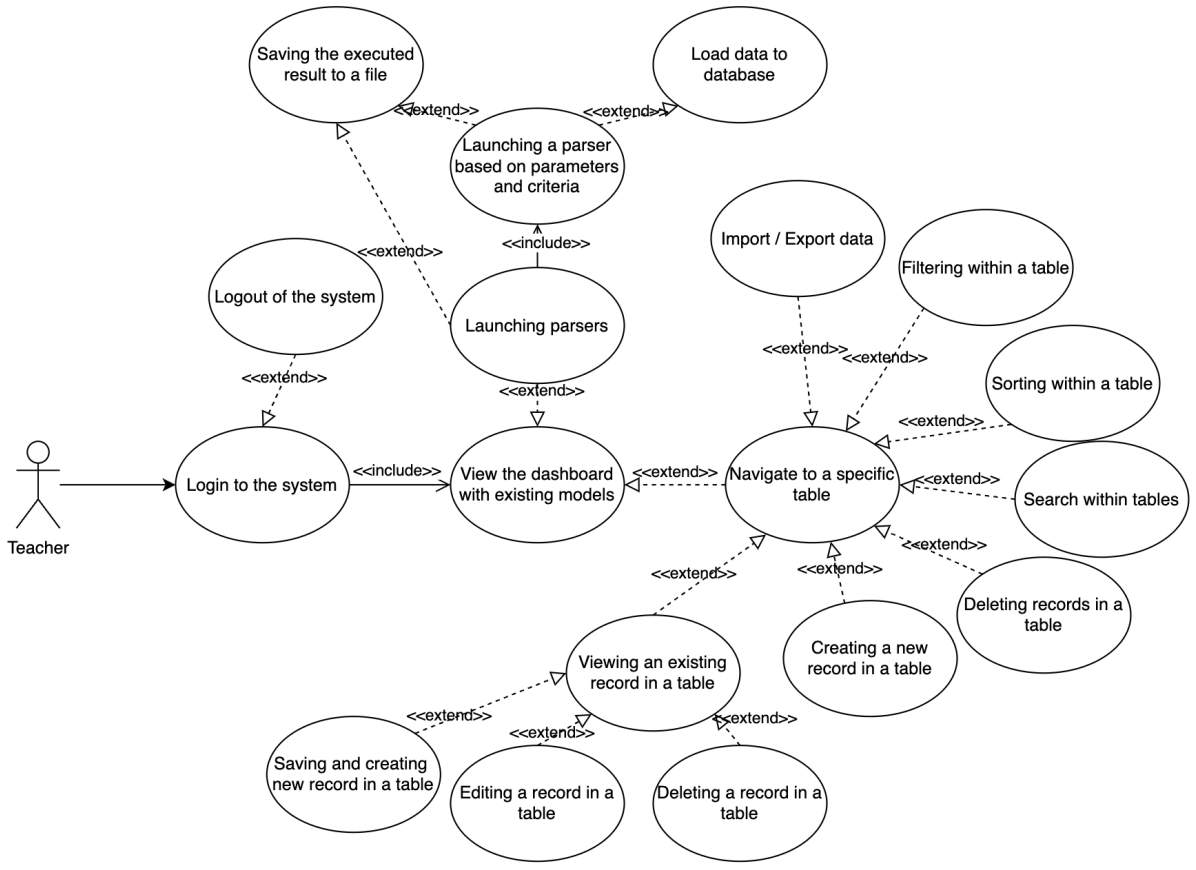


Рисунок 3.1 — Діаграма сценаріїв для адміністративної панелі

3.3 Структура бази даних

Визначаючи структуру освітньої програми, було досліджено з яких компонентів складається програма. Базуючись на цьому, можемо побудувати структуру бази даних.

Розглянемо таблиці бази даних та їх поля.

1) Term — таблиця Семестр, створена для того, щоб не створювати окремо семестр для кожної дисципліни, в яких вона вивчається:
- number — Enum поле з числовими значеннями від 1 до 12 (1-8 семестри для бакалаврату та 9-12 для магістрів), що відповідає номеру семестру;
- value — значення, яке відповідає числу в таблиці.

2) Discipline — таблиця Дисципліна. Тут зберігатиметься детальна інформація про кожен предмет (його код, назва та в якому семестрі він вивчається):
- code — CharField(max_length=15), унікальний код дисципліни;
- name — CharField(max_length=255), назва дисципліни;
- term — ManyToManyField(to=Term), це поле зі зв’язком n-to-n до таблиці Семестр.

3) SelectiveBlock — таблиця Вибірковий Блок (кафедра)**:**
- name — models.CharField(max_length=255) — назва блоку (наприклад, Інтелектуальні інформаційні технології);
- disciplines — ManyToManyField(to=Discipline) поле зі зв’язком n-to-n до таблиці Дисципліна. Відповідно, це поле зберігатиме перелік дисциплін, характерних для певної кафедри.

4) SelectiveList — таблиця Вибірковий Перелік дисциплін, що містить в собі групи дисциплін на вибір:
- name — CharField(max_length=255) — назва або номер переліку;
- first_discipline — ForeignKey(to=Discipline) — посилання на першу дисципліну, яка пропонується до вибору;

- second_discipline — ForeignKey(to=Discipline) — посилання на другу дисципліну, яка пропонується до вибору.

5) Specialty — таблиця Спеціальність, що компонує в собі всю необхідну інформацію та складає Освітню програму:

- name — Enum поле (значення на вибір: System Analysis — 124, Applied Mathematics — 113, Informatics — 122, Software Engineering — 121), зберігає інформацію про освітньо-професійну програму;
- year_of_admission — DateField, зберігає інформацію про те, якого року вступники вчаться за даною програмою;
- mandatory_disciplines — ManyToManyField(to=Discipline) — поле зі зв’язком n-to-n до таблиці Дисципліна, список обов’язкових дисциплін;
- selection_by_blocks — ManyToMany(to=SelectiveBlock) — поле зі зв’язком n-to-n до таблиці SelectiveBlock, зберігає список кафедр та дисциплін, що вивчатимуться на кожній з них;
- selection_from_the_list — ManyToMany(to=SelectiveList) — поле зі зв’язком n-to-n до таблиці SelectiveList, зберігає список всіх предметів на вибір.

Відповідно до вищезазначених класів, створено наступну діаграму класів:

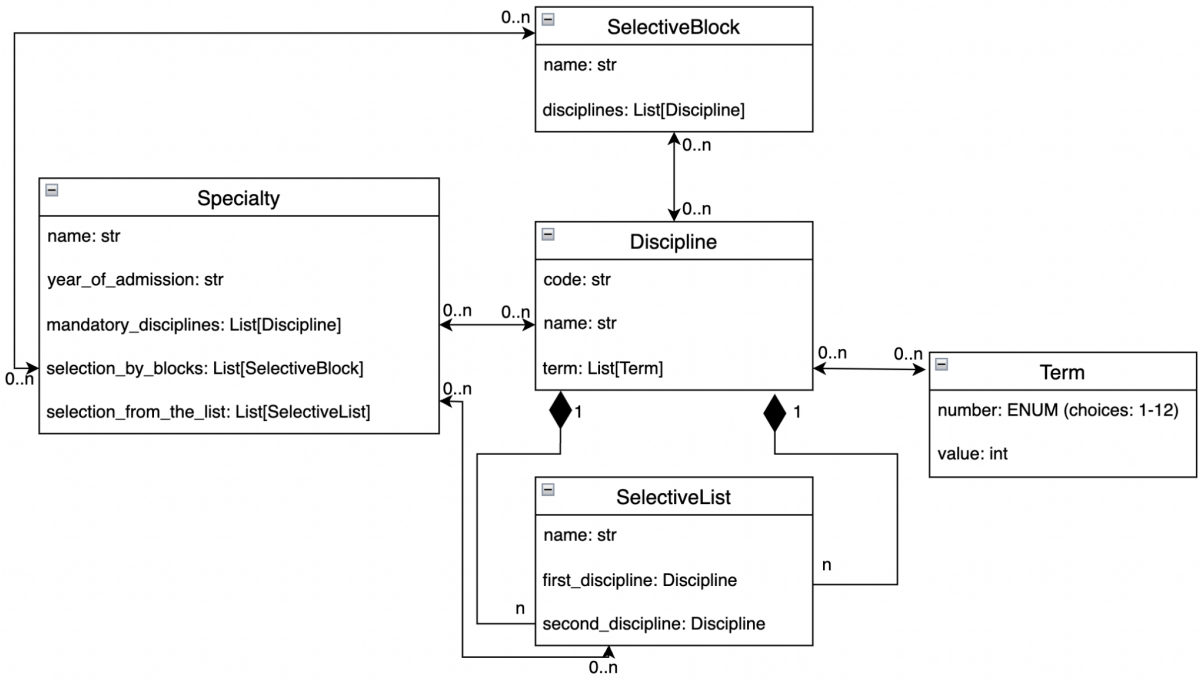


Рисунок 3.2 — Діаграма класів системи

РОЗДІЛ 4 РОЗРОБКА ВЕБДОДАТКУ

4.1 Розробка бази даних та запитів

4.1.1 Підключення бази даних PostgreSQL

Для написання проекту була поставлена мета отримати надійну, масштабовану та легко використовувану базу даних. Тому, як вже було зазначено в розділі 3.2, обрано об'єктно-реляційну систему управління базами даних PostgreSQL та хмарний сервіс управління нею — ElephantSQL.

В першу чергу варто створити порожню базу даних в ElephantSQL. Для цього варто перейти на сайт ElephantSQL та зареєструватись або увійти у вже існуючий акаунт. Далі, на панелі керування, натискаємо кнопку “Create New Instance” для створення нового екземпляру бази даних. Вводимо відповідні дані та обираємо безкоштовний план, цього буде достатньо для даного проекту. Після створення екземпляру, отримано URL та облікові дані для підключення до бази даних, що будуть використані для підключення Django до створеної бази даних.

Наступний крок це підключення Django до створеної бази даних PostgreSQL. Для цього в головній папці проекту знаходимо файл “settings.py”, який відповідає за всі налаштування проекту та його роботи. Знаходимо налаштування бази даних по ключовому слову DATABASES та змінюємо налаштування за замовчуванням на наступні:

```
83  DATABASES = {
84      "default": {
85          "ENGINE": "django.db.backends.postgresql",
86          "NAME": os.getenv("DB_NAME"),
87          "USER": os.getenv("DB_USER"),
88          "PASSWORD": os.getenv("DB_PASSWORD"),
89          "HOST": os.getenv("DB_HOST"),
90          "PORT": os.getenv("DB_PORT"),
91      }
92  }
```

Рисунок 4.1 — Налаштування бази даних у коді

Розглянемо створені налаштування:

- ENGINE — значення яке вказує фреймворку Django на те, яка база даних використовується у проекті;
- USER — ім'я користувача бази даних;
- PASSWORD — пароль користувача;
- HOST — хост або IP-адреса сервера;
- PORT — порт для підключення (для PostgreSQL зазвичай це 5432);
- NAME — ім'я бази даних.

Всі значення, окрім ENGINE, це облікові дані з ElephantSQL.

Варто зазначити, що у даному фрагменті всі секретні дані імпортуються з файлу “.env”. Файл “.env” це текстовий файл, де зберігаються пари “ключ-значення”. Ці пари використовуються для задання різних конфігураційних змінних для проекту. Також було створено файл “.env.sample”, який містить всі змінні з файлу “.env”, але з несправжніми значеннями. Цей файл потрібен для того, щоб інші розробники знали про те, які саме змінні треба створити та в якому форматі мають зберігатись їх значення. Файл “.env.sample” є публічним, на відміну від файлу “.env”, який не завантажується в мережу та має бути створений кожним розробником особисто.

4.1.2 Розробка таблиць бази даних

В розділі 3.3 було розглянуто схему бази даних, включаючи таблиці, поля, типи полів та зв'язків між таблицями. Розглянемо їх реалізацію. Вся структура бази даних написана за допомогою можливостей Django ORM.

За внутрішнім стилем фреймворку Django — всі моделі зберігаються у відповідному модулі в файлі "models.py", в даній роботі також дотримується цей принцип. Для того, щоб створити нову таблицю в базі даних треба створити клас та унаслідуватись від класу models.Model з бібліотеки django.db. Для створення стовпців таблиці та визначення їх типу і властивостей створюються атрибути класу, назва яких відповідатиме назві колонки.

Розглянемо приклад створення таблиці Discipline:

```
1  from django.db import models
2
3
4  class Discipline(models.Model):
5      code = models.CharField(max_length=15, unique=True)
6      name = models.CharField(max_length=255)
7      term = models.ManyToManyField('Term')
```

Рисунок 4.2 — Приклад створення таблиці

На даному фрагменті коду на першому рядку виконується імпорт файлу models з бібліотеки django.db. Рядок 4 — оголошення таблиці в базі даних з назвою Discipline. Рядки 5-7 — створення стовпців таблиці, тип поля зазначається після слова models. В даному випадку поля code та name це текстові поля, а поле term — поле зв'язку типу n-to-n до таблиці Term. Всередині полів можна побачити обмеження та властивості, наприклад для поля code максимальне значення запису в рядку має бути 15 символів, і воно має бути унікальним. Інші моделі, що були створені для бази даних, будуть додані до Додатку А.

Останній крок — створення таблиць в підключеній базі даних. Для цього потрібно спершу під’єднатись до бази даних за допомогою команди “python manage.py migrate”.

Після цього треба відтворити зміни з коду, який написаний на мові програмування Python на мову запитів SQL. Це робиться за допомогою команди “python manage.py makemigrations”. В цілому ця команда використовується для створення нових міграцій на основі змін, які були внесли у модель. Django автоматично генерує скрипти міграції, що описують, як внести ці зміни в базу даних. Міграції створюються в окремих файлах в директорії “migrations” кожного Django додатка.

Далі треба застосувати створені міграції до бази даних командою “python manage.py migrate”.

4.1.3 Запити до бази даних

Основна робота для виконання даного проекту полягає у покращенні та перероблені адміністративної панелі Django. Адміністративна панель Django є потужним інструментом для управління даними вебсайту, що дозволяє взаємодіяти з ними через графічний інтерфейс. До основних можливостей адміністративної-панелі Django належить виконання CRUD операцій, авторизація та аутентифікація.

Адміністративна панель Django дозволяє виконувати CRUD операції на даних моделей проекту. Авторизація та аутентифікація включає в себе вбудовану систему аутентифікації, яка дозволяє контролювати доступ до різних частин вебсайту. Всі операції в панелі Django виконуються через SQL запити до бази даних. Коли виконуються певні дії в панелі, Django ORM автоматично генерує відповідні SQL запити до бази даних.

Крім того, адміністративна панель Django включає в себе ряд інших функцій, які можна налаштувати та розширити, включаючи налаштування вигляду та додавання нових форм. Власне, це і є основним завданням проекту.

Отже, більшість запитів вже є виконана, тому тут буде описано створення та реалізація власних налаштованих запитів, які не покриваються вбудованим функціоналом.

Перелік власних налаштованих запитів, які додано до проекту:

- завантаження даних до бази даних;
- вивантаження даних з бази даних;
- парсинг даних з сайту;
- завантаження даних з парсеру до бази даних;
- завантаження даних з парсеру до файлу.

Налаштування власних запитів буде розглянуто в наступних пунктах цього розділу.

Розглянемо архітектуру Django. Django працює на основі архітектури MVT, яка є модифікацією шаблону MVC. Основні складові Django MVT:

- model — модель Django представляє структуру даних. Це місце, де визначається схема бази даних, включаючи таблиці, їх поля та взаємозв'язки між ними;
- view — відповідає за обробку HTTP-запитів та відповідей. Види використовують моделі для отримання даних, які потім передаються в шаблон. Види можуть бути написані як функції або як класи;
- template — шаблони Django, які використовуються для генерації HTML-відповідей. Шаблони можуть включати змінні, які будуть замінені даними, що передаються з видів. Django має власну мову шаблонів, яка включає теги та фільтри, що дозволяють динамічно генерувати HTML.

4.2 Розробка адміністративної панелі та інтерфейсу користувача

4.2.1 Реєстрація таблиць в адміністративній панелі

Адміністративна панель Django — це вбудований інструмент, який дозволяє управляти контентом вебсайту, його наповненням бази даних.

Адміністративна панель має зручний інтерфейс, який може використовуватися для створення, зміни, видалення і навіть пошуку записів, збережених в базі даних.

Для того, щоб в адміністративній панелі з'явились певні таблиці бази даних необхідно їх зареєструвати у файлі "admin.py". Якщо просто зареєструвати модель, то з'являється можливість переглянути таблицю та її записи, видалити або редагувати їх, а також створювати нові записи. Проте, додамо можливість робити пошук по таблиці, додамо фільтрацію та сортування по полям.

Розглянемо приклад реєстрації моделі Specialty:

```
36  class SpecialtyAdmin(admin.ModelAdmin):
37      list_display = ["name", "year_of_admission"]
38      list_filter = ["name", "year_of_admission"]
39      search_fields = ["name"]
40      ordering = ["name", "year_of_admission"]
41
42
43  admin.site.register(Specialty, SpecialtyAdmin)
```

Рисунок 4.3 — Реєстрація таблиці до адміністративної панелі

На вищевказаному прикладі коду рядок 36 відповідає за оголошення класу реєстрації SpecialtyAdmin, який наслідується від класу admin.ModelAdmin і, таким чином, переймає базові властивості для можливості виконання CRUD операцій. Рядок 43 — безпосередня реєстрація таблиці Specialty за зазначеними властивостями класу реєстрації SpecialtyAdmin. Рядки 37-40 відповідають за додаткові налаштування:

- list_display — атрибут, який відповідає за те, які саме поля класу будуть відображатись на сторінці;
- list_filter — зазначення полів, за якими може здійснюватись фільтрація;
- search_fields — назви полів, за якими може виконуватись пошук;
- ordering — назви полів, за якими можна змінювати сортування значень в таблиці.

За таким самим принципом до адміністративної панелі реєструється кожна таблиця.

Після реєстрації таблиць переходимо за посиланням адміністративної панелі: “/admin” та можемо побачити, що таблиці додались до сторінки. Розглянемо таблицю Specialty, яка з’явилась у списку зареєстрованих моделей:

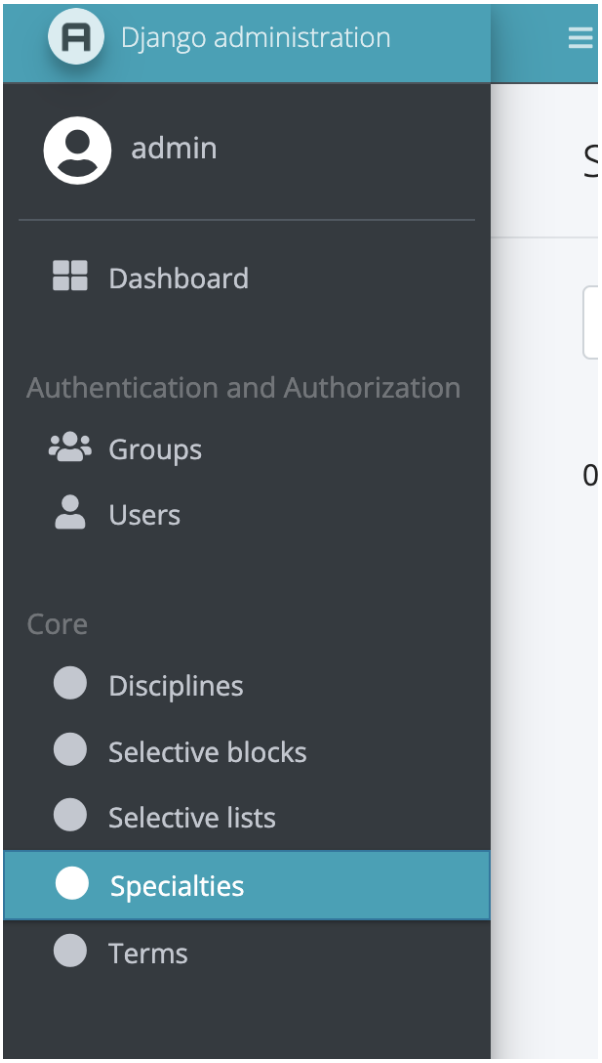


Рисунок 4.4 — Слайдер адміністративної панелі

Якщо перейти за даною таблицею, то можна побачити, що з’явився пошук, фільтрація та сортування:

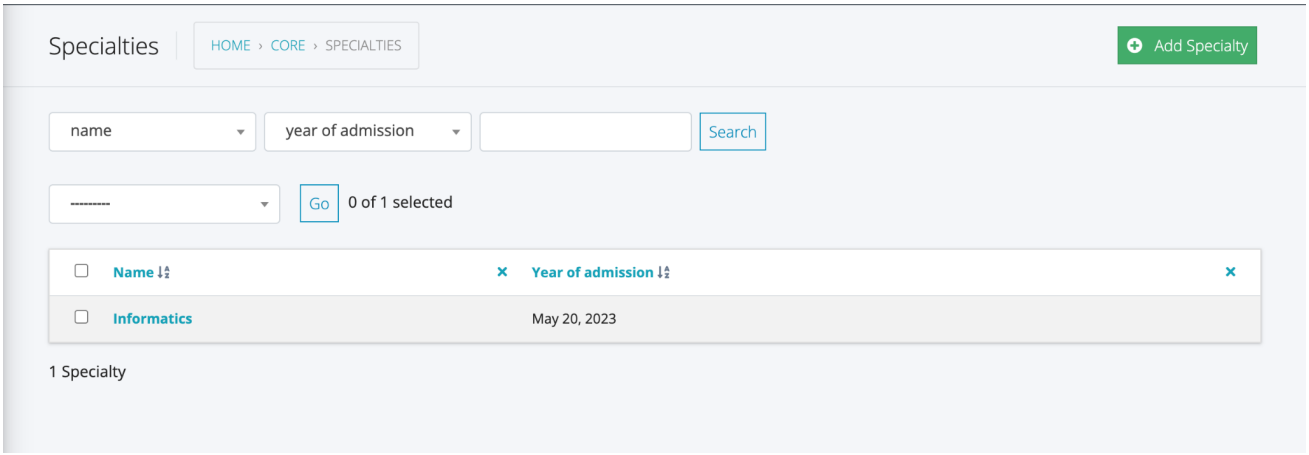


Рисунок 4.5 — Детальна сторінка таблиці

Зауважте, на зображенні присутній лише один запис в табличці, оскільки наповнення бази даних справжніми даними буде розглянуто в розділі 4.3.

4.2.2 Стилізація адміністративної панелі

За замовчуванням адміністративна панель Django має наступний вигляд:

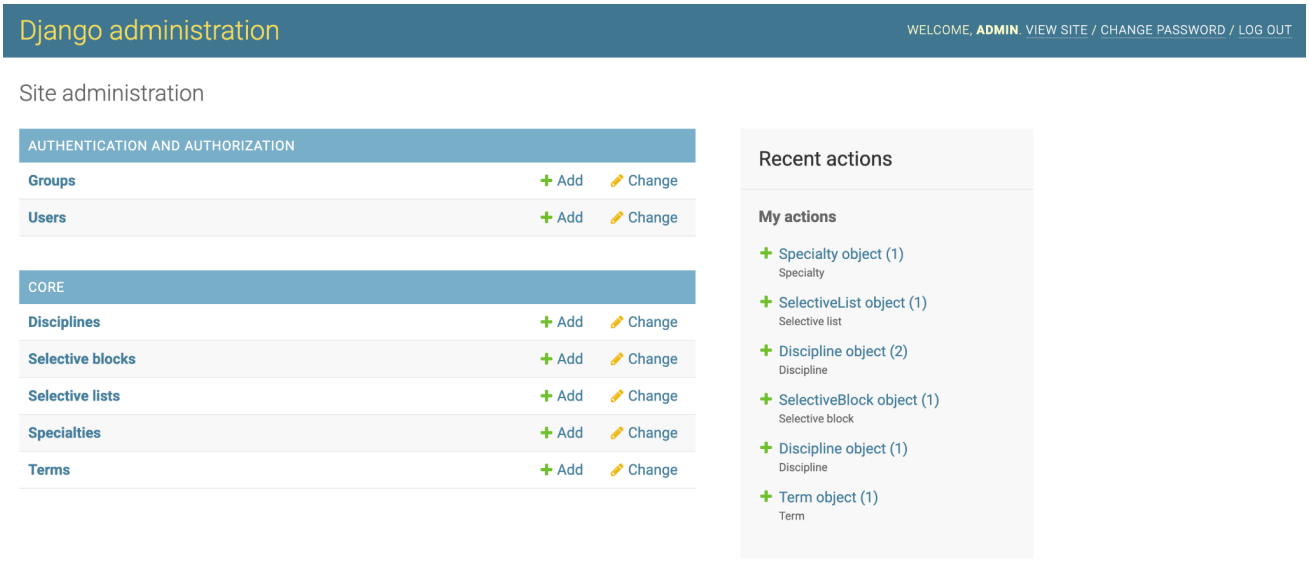


Рисунок 4.6 — Зовнішній вигляд панелі за замовчуванням

Даний зовнішній вигляд є прийнятним, але не сучасним та приємним. Тут дотримано мінімалістичність деталей та простоту в дизайні, але для вдалого проекту варто покращити зовнішній вигляд системи. Для цього використовуватимемо сторонню бібліотеку Django jazzmin [12].

Спочатку встановлюємо дану бібліотеку за інструкцією, яку можна переглянути в офіційній документації. Наступним кроком варто додати встановлену бібліотеку до списку встановлених додатків у налаштуваннях проекту в файлі “settings.py”:

```
34  # Application definition
35  INSTALLED_APPS = [
36      "jazzmin",
37
38      "django.contrib.admin",
```

Рисунок 4.7 — Додавання встановленого пакету до налаштувань

Тепер адміністративна панель має наступний вигляд:

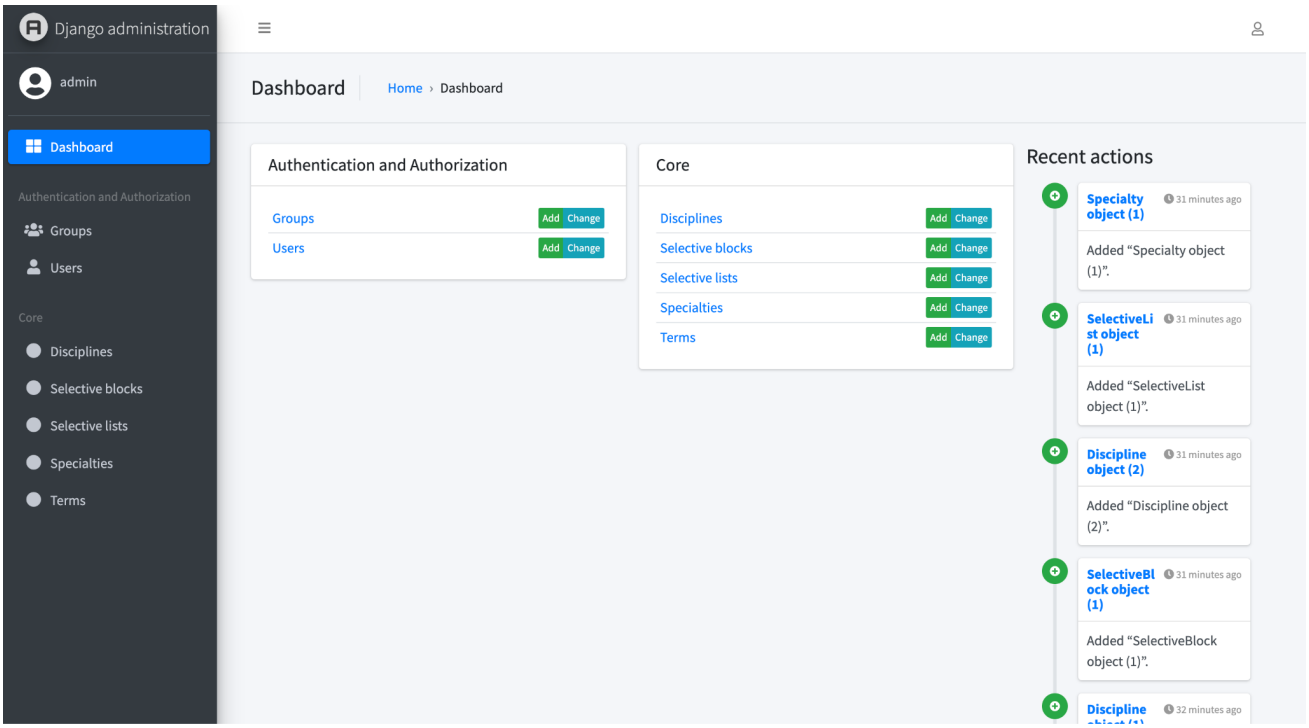


Рисунок 4.8 — Вигляд панелі з використанням бібліотеки

Такий вигляд панелі є сучаснішим та цікавим, проте все ще недосконалий. Далі будемо власноруч змінювати стилізацію панелі.

Першим кроком оберемо кольори та характер зовнішнього вигляду панелі. Кольори грають важливу роль у визначенні настрою і атмосфери вебсайту. У випадку системи управління наповненням освітніх програм, вибір синьо-голубого і сірого кольорів може мати певні переваги.

Синьо-голубий колір вважається спокійним і стабільним, асоціюючись з міццю, надійністю і мудрістю. Він також викликає асоціації з небом і океаном, що викликає відчуття відкритості і свободи. Це може сприяти комфортному користуванню панеллю управління, допомагаючи користувачам відчувати себе заспокоєними і зосередженими.

Сірий колір вважається нейтральним і збалансованим. Він може допомогти підкреслити інші кольори і забезпечити простір, який не втомлює очі при довгому використанні. Крім того, сірий колір також асоціюється з професіоналізмом і формальністю, що може підсилити враження високої якості та надійності платформи.

Що стосується характеру інтерфейсу, то він повинен бути дружнім, зручним та інтуїтивно зрозумілим. Користувачі повинні з легкістю знайти потрібну інформацію або виконати потрібні дії без необхідності вивчати складні інструкції. Він повинен бути як можна більш відкритим і доступним, з усіма основними функціями, які відображаються на видному місці.

Отже, налаштовуємо кольори та розміри за визначеними вище критеріями. Бібліотека надає можливість обрати вже готові теми та налаштувати під власні потреби. Використовуючи дану можливість, створюємо налаштування:

```
JAZZMIN_UI_TWEAKS = {
    "navbar_small_text": True,
    "footer_small_text": False,
    "body_small_text": False,
    "brand_small_text": True,
    "brand_colour": "navbar-info",
    "accent": "accent-info",
    "navbar": "navbar-cyan navbar-dark",
    "no_navbar_border": False,
    "navbar_fixed": False,
    "layout_boxed": False,
    "footer_fixed": False,
    "sidebar_fixed": False,
```

Рисунок 4.9 — Фрагмент налаштування стилізації панелі

Зміни прописуються в файлі “settings.py”. Після застосування даного коду зовнішній вигляд системи змінився наступним чином:

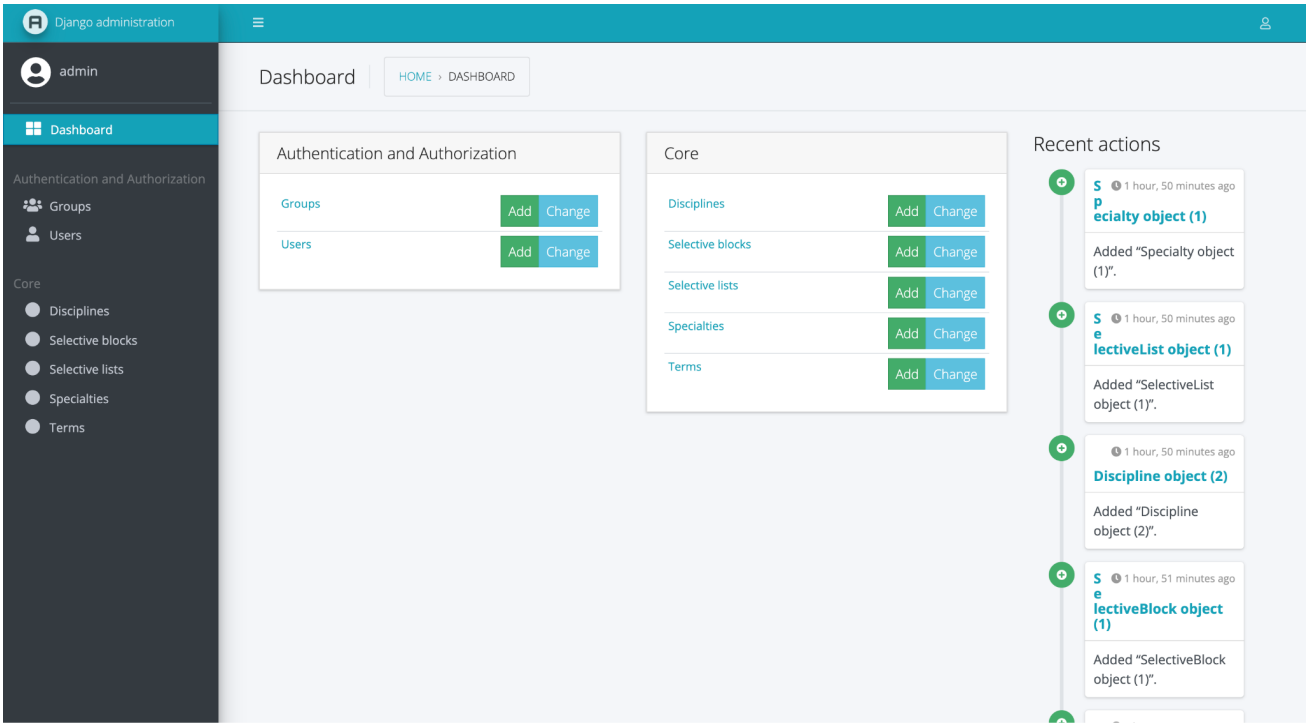


Рисунок 4.10 — Панель з налаштованим власним стилем

Для повного завершення налаштування зовнішнього вигляду адміністративної панелі змінимо назви, винесемо деякі таблиці окремо та додамо функціонал:

```python
JAZZMIN_SETTINGS = {
    "site_title": "Admin",
    "show_ui_builder": False,
    "site_header": "EMS",
    "site_logo_classes": "img-circle",
    "welcome_sign": "Welcome to the Educational Management System admin panel!",
    "search_model": "core.discipline",
    "topmenu_links": [
        {"name": "Home", "url": "admin:index", "permissions": ["auth.view_user"]},
        {"model": "core.discipline"},
        {"model": "core.specialty"},
    ],
    "show_sidebar": True,
    "navigation_expanded": True,
    "changeform_format": "collapsible",
    "copyright": "Educational Management System admin-panel. Made by"
    " <a href='https://github.com/Nattalli' target='_blank'>Nataliia Zakharchuk</a>",
}
```

Рисунок 4.11 — Додаткові налаштування панелі

Після внесення останніх змін зовнішній вигляд системи змінився до наступного:

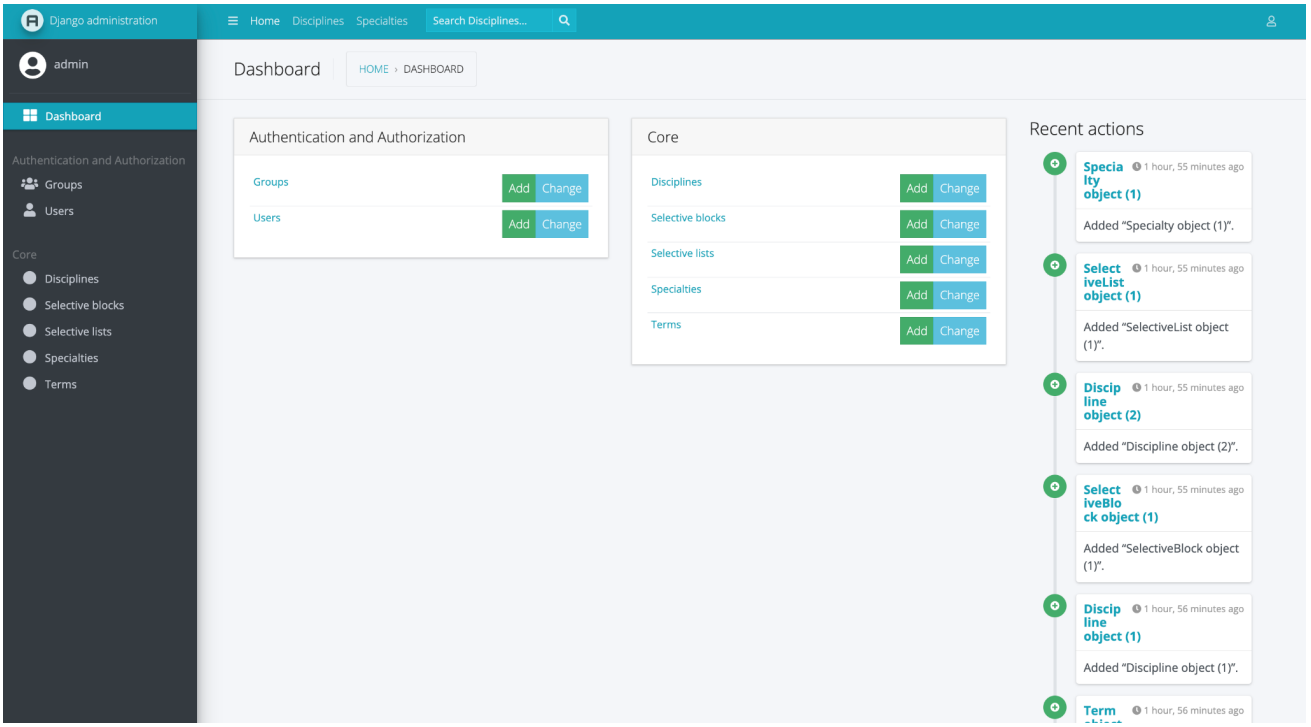


Рисунок 4.12 — Кінцева стилізація адміністративної панелі

Тепер система виглядає більш сучасною та наповненою, зручною для використання.

4.3 Реалізація додаткових функцій та можливостей

4.3.1 Створення парсерів для збирання інформації щодо навчальних програм минулих років

Як вже було зазначено в розділі 1, увагу буде зосереджено на дослідженні програми навчальних дисциплін факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка. Для заповнення бази даних справжніми даними будемо використовувати дані, які розміщені на вебсайті факультету. Для того, щоб не заповнювати інформацію вручну та для уникнення помилок — автоматизуємо даний процес за допомогою створення парсерів, які при запуску будуть зчитувати HTML-розмітку сторінки, фільтрувати отримані дані та за певними критеріями залишати лише необхідну інформацію. Парсер — програма, що забезпечує парсинг, тобто синтаксичний аналіз контенту в мережі по певній математичній моделі [13].

На жаль, у факультету комп'ютерних наук та кібернетики кожна спеціальність має різну структуру відображення на сайті. Тому доведеться створити щонайменше чотири парсери для спеціальностей, а також ще один для загальної сторінки з посиланнями на всі навчальні програми. Отже, для повного виконання завдання даного проекту буде написано п'ять парсерів.

Винесемо всі парсери в окремий модуль "parsers". Оскільки в даному випадку на сторінках відсутня пагінація та немає необхідності переходити на сторінки по посиланням та зчитувати інформацію зсередини, то можна використати сторонню бібліотеку BeautifulSoup, яка забезпечує базові функції та можливості для парсингу.

Розглянемо детально написання парсеру для головної сторінки. Всі інші сторінки з спеціальностями будуть написані по аналогії, лише будуть відрізнятись деякі параметри та структура відповіді.

Обов'язково треба визначити за яким посиланням відбуватиметься парсинг даних. Краще посилання винести у окрему зміну як константу, оскільки це кращі практики Python:

```
6   BASE_URL = "https://csc.knu.ua/uk/"
7   HOME_URL = urljoin(BASE_URL, "programs")
```

Рисунок 4.13 — Визначення посилання для парсеру

Тепер створимо основну функцію, при запуску якої відбуватиметься парсинг. Назвемо її get_main_page. Робимо запит на зазначену сторінку та зберігаємо HTML-відповідь у змінну. Це відповідає рядкам 43-45 коду:

```
43  def get_main_page() -> dict[str, list]:
44      page = requests.get(HOME_URL).content
45      soup = BeautifulSoup(page, "html.parser")
46      bachelor_degree, master_degree, *_ = get_all_programs(soup)
47      bachelor_programs = create_programs(
48          fix_names(get_degree_program_names(bachelor_degree)[:4]),
49          get_programs_links(bachelor_degree)[:4],
50      )
51      master_programs = create_programs(
52          fix_names(get_degree_program_names(master_degree)[:7]),
53          get_programs_links(master_degree)[:7],
54      )
55      return {
56          "bachelor": bachelor_programs,
57          "master": master_programs,
58      }
```

Рисунок 4.14 — Основна функція парсеру

Наступні рядки працюють безпосередньо з обробкою та форматуванням тексту.

Рядок 46 відповідає за відбір лише тих елементів сторінки, які мають клас стилізації “.col-sm-6”. Цю частину функціоналу винесено в окрему функцію для дотримання принципу єдиної відповідальності. Це один з принципів SOLID, що є набіром принципів об'єктно-орієнтованого програмування, розроблених для покращення читабельності, гнучкості та масштабованості програмного коду. Протягом усього проекту дотримувалось виконання даних принципів.

```
10  def get_all_programs(page_soup: BeautifulSoup) -> list:
11      return page_soup.select(".col-sm-6")
```

Рисунок 4.15 — Допоміжна функція для парсингу

Рядки 47 - 50 відповідають за безпосереднє винесення інформації про навчальні програми бакалаврату зі сторінки до струтурованого об’єкту. Аналогічно в рядках 51-54 створюється об’єкт навчальних програм магістратури.

Виведемо частину результату виконання функції get_main_page у консоль:

```
bachelor
Інформатика
('http://csc.knu.ua/media/study/bachelor-degree/inf/inf-courses-2018-2019.html', 'Робочі програми за планом 2018')
('http://csc.knu.ua/media/study/bachelor-degree/inf/inf-courses-2019-2020.html', 'Робочі програми за планом 2019')
('http://csc.knu.ua/media/study/bachelor-degree/inf/inf-courses-2020-2021.html', 'Робочі програми за планом 2020')
('http://csc.knu.ua/media/study/bachelor-degree/inf/inf-courses-2022.html', 'Робочі програми за планом 2022')
('http://csc.knu.ua/media/study/bachelor-degree/inf/inf-courses-2018-zao.html', 'Робочі програми заочного відділення\xa0')

Прикладна математика
('http://csc.knu.ua/media/study/bachelor-degree/pm/pm-courses-2018-2019.html', 'Робочі програми за планом 2018 року')
('http://csc.knu.ua/media/study/bachelor-degree/pm/pm-courses-2019-2020.html', 'Робочі програми за планом 2019 року')
('http://csc.knu.ua/media/study/bachelor-degree/pm/pm-courses-2020-2021.html', 'Робочі програми за планом 2020 року')
('http://csc.knu.ua/media/study/bachelor-degree/pm/pm-courses-2022-2023.html', 'Робочі програми за планом 2022 року')

Системний аналіз
('http://applstat.univ.kiev.ua/ukr/?templ=studies', 'Робочі програми за ОПП 2018')
('http://applstat.univ.kiev.ua/ukr/?templ=studies', 'Робочі програми за ОПП 2019')
('http://applstat.univ.kiev.ua/ukr/?templ=studies', 'Робочі програми за ОПП 2020, 2021')
('http://applstat.univ.kiev.ua/ukr/3.php', 'Робочі програми за ОПП 2022')

Програмна інженерія
('http://csc.knu.ua/media/study/bachelor-degree/courses-2016-2018.html', 'Робочі програми за планом 2018')
('http://csc.knu.ua/media/study/bachelor-degree/courses-2019-202x.html', 'Робочі програми за планом 2019')
```

Рисунок 4.16 — Результат роботи парсера

За таким самим принципом написані парсери і для кожної спеціальності окремо.

4.3.2 Додавання парсерів до адміністративної панелі

Додавання парсерів до адміністративної панелі вимагає переписання вбудованого коду панелі, створення HTML-сторінок, виду для виклику парсеру та шляху до цього виду. Розглянемо покроково кожен етап для створення

сторінки з парсером get_main_page, який було розглянуто у розділі 4.3.1, інші парсери будуть додані до адміністративної панелі за таким самим принципом.

Спершу варто створити вид, який викликатиме функцію парсеру, зберігатиме результат її виконання та передаватиме у відповідну HTML-сторінку. Назвемо цей вид parse_main_page:

```
8    def parse_main_page(request: Request) -> HttpResponse():
9        data = get_main_page()
10       context = {"data": data}
11 <>    return render(request, "data_display.html", context)
```

Рисунок 4.17 — Вид для виклику парсера

Тут у 9 рядку викликається функція парсеру get_main_page та зберігається у змінну data. Рядок 10 — зміна формату відповіді таким чином, щоб це був словник з ключем та значенням. Наступний рядок це повернення HTML-сторінки, в яку передаєтсья результат виконання функції.

Наступним кроком треба створити шлях до даного виду за допомогою його оголошення у файлі urls.py відповідного модулю.

```
4  urlpatterns = [
5      path("parse-and-display-data/", parse_main_page, name="parse_and_display_data"),
6  ]
```

Рисунок 4.18 — Реєстрація нового виду

Після реєстрації відповідного запиту необхідно додати його до адміністративної панелі. В даному проекті це зроблено за допомогою додавання кнопки “Спарсити навчальні програми” на сторінці таблиці Disciplines. Для того, щоб створити дану кнопку, необхідно до вже існуючої сторінки таблиці додати власний HTML-код.

Для додавання кнопки необхідно створити новий файл у папці templates, яка створилась при ініціалізації проекту. Назвемо цей файл “disciplines_parser.html”:

```
{% extends "admin/change_list.html" %}

{% load i18n admin_urls %}

{% block object-tools-items %}
      <a href="{% url 'parse_and_display_data' %}" class="btn btn-primary">
        {% trans "Parse and Display Data" %}
      </a>
    {{ block.super }}
{% endblock %}
```

Рисунок 4.19 — Додавання кнопки до сторінки

Тут шаблон наслідується від вбудованого файлу “admin/change_list.html”. Далі зазначається де та що буде додано: кнопка, при натисканні на яку буде зроблено переадресування за посиланням “parse_and_display_data” — посилання створенного виду для функції парсеру.

Щоб програма розуміла для якого саме файлу написаний цей фрагмент коду варто зазначити це в налаштуваннях вигляду таблиці у файлі “admin.py”:

```
@admin.register(Discipline)
class DisciplineAdmin(admin.ModelAdmin):
    change_list_template = "disciplines_parser.html"
```

Рисунок 4.20 — Підключення HTML-сторінки

Тепер додано кнопку парсеру на сторінку таблиці Discipline:

Рисунок 4.21 — Сторінка з новою кнопкою

Останній крок — написання HTML-сторінки, на яку має перейти користувач при натисканні на дану кнопку. Код даної сторінки зі стилізацією додано до Додатку Б. Результат виконання коду сторінки:

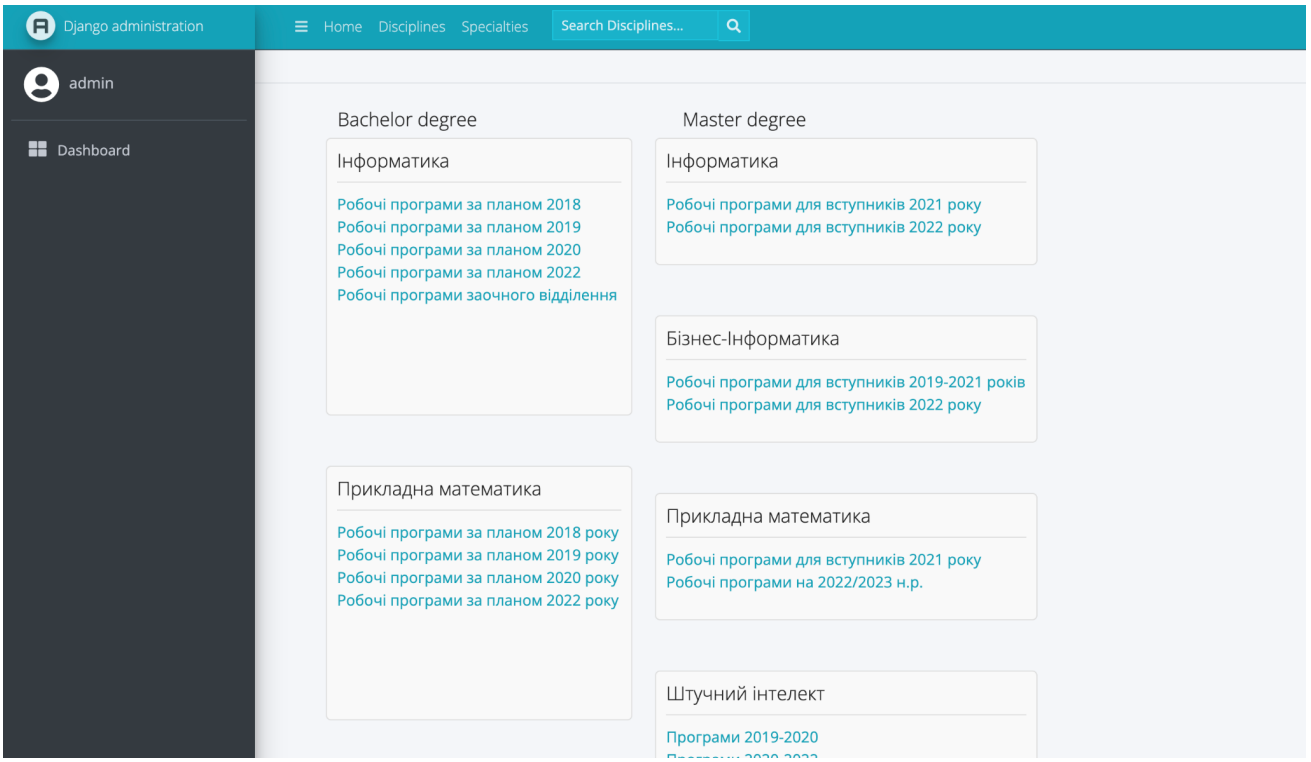


Рисунок 4.22 — Сторінка виконання роботи парсеру

4.3.3 Можливість завантажити інформацію з парсеру у файл

З метою збільшення користі від створених парсерів, було вирішено додати можливість завантаження інформації у файл CSV формату. Для виконання цього етапу потрібно створити функцію, яка завантажує дані до файлу та додати кнопку, яка викликає створену функцію.

Створюємо новий вид, який зберігатиме дані з парсеру до файлу, назвемо його “download_main_page_to_csv”:

```
15  def download_main_page_to_csv(request: Request) -> HttpResponse():
16      data = get_main_page()
17      response = HttpResponse(content_type="text/csv")
18      response["Content-Disposition"] = 'attachment; filename="data.csv"'
19
20      writer = csv.writer(response)
21      for degree, value_list in data.items():
22          for name, links in value_list:
23              for url, description in links:
24                  writer.writerow([degree, name, url, description])
25
26      return response
```

Рисунок 4.23 — Вид для збереження інформації до файлу

З виклику функції дані приходять у JSON-форматі, а для сприйнятливого вигляду необхідно, щоб отримані дані були структуровані, мали стовпці та рядки значень. Для цього у рядках 20-24 було зроблено ітерацію по отриманим значенням та записано їх у рядок списком. Отже, download_main_page_to_csv view створює новий CSV файл, записує в нього потрібні дані та відправляє його як відповідь на HTTP-запит.

Після створення виду важливо зареєструвати його в “urls.py” файлі, щоб програма мала доступ до виду та при здійснені запиту на нього відбувалась певна дія:

```
26  urlpatterns = [
27      path("admin/parse-and-display-data/", parse_main_page, name="parse_and_display_data"),
28      path("admin/download-csv/", download_main_page_to_csv, name="download_csv"),
```

Рисунок 4.24 — Реєстрація видів

Останній крок — створення кнопки, яка відповідатиме за створення запиту на вид download_main_page_to_csv. Додамо цю функцію до сторінки таблиці, що відображає інформацію з парсеру. Для створення нової кнопки було достатньо додати наступний код з посиланням на зареєстрований шлях виду:

```
58  <a href="{% url 'download_csv' %}" class="btn btn-primary">
59      Download to CSV
60  </a>
```

Рисунок 4.25 — Додавання кнопки на сторінку

Також додано стилі для кнопки, щоб вона гармонійно вписувалась до зовнішнього вигляду сторінки:

```
30      .btn-primary {
31          margin-left: 50px;
32          margin-bottom: 20px;
33          height: 40px;
34      }
```

Рисунок 4.26 — Стилізація кнопки

Тепер продемонструємо результат виконаного етапу та яким чином зберігається файл:

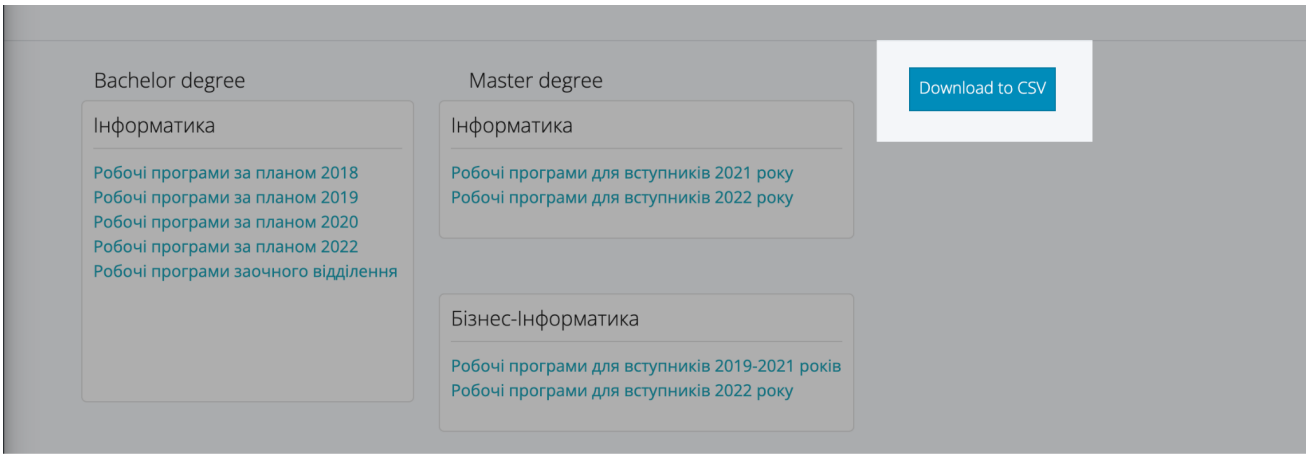


Рисунок 4.27 — Кнопка завантаження на сторінці

При натисканні на кнопку спочатку пропонується обрати шлях, куди буде завантажено файл, та створити назву файлу, за замовчуванням це значення

дорівнює “data.csv”:

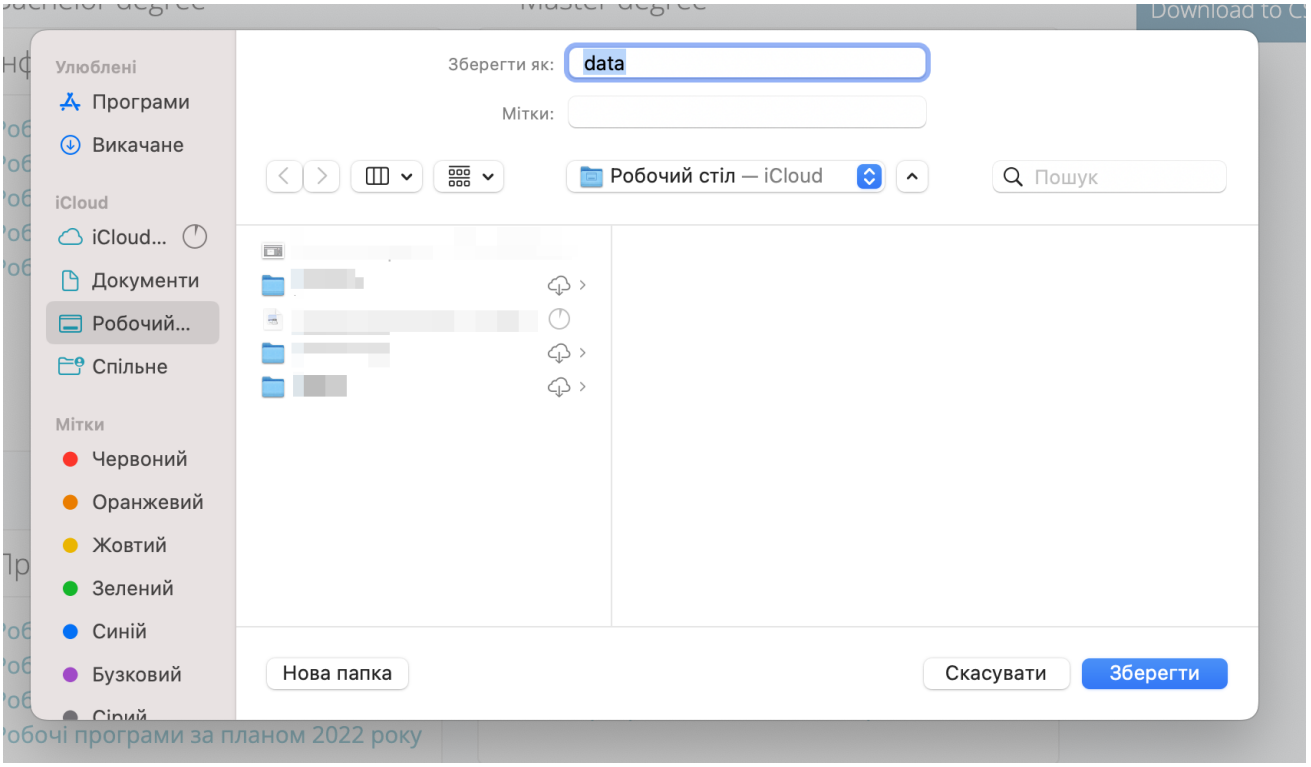


Рисунок 4.28 — Збереження файлу

Тепер на головному екрані з’явився відповідний файл з назвою “data.csv”. При його відкриті можна побачити наступну інформацію:

data

	Degree	Specialty	Link	Title
2	bachelor	Інформатика	http://csc.knu.ua/media/study/bachelor-degree/inf/inf-courses-2019-2020.html	Робочі програми за планом 2019
3	bachelor	Інформатика	http://csc.knu.ua/media/study/bachelor-degree/inf/inf-courses-2020-2021.html	Робочі програми за планом 2020
4	bachelor	Інформатика	http://csc.knu.ua/media/study/bachelor-degree/inf/inf-courses-2022.html	Робочі програми за планом 2022
5	bachelor	Інформатика	http://csc.knu.ua/media/study/bachelor-degree/inf/inf-courses-2018-zao.html	Робочі програми заочного відділення
6	bachelor	Прикладна математика	http://csc.knu.ua/media/study/bachelor-degree/pm/pm-courses-2018-2019.html	Робочі програми за планом 2018 року
7	bachelor	Прикладна математика	http://csc.knu.ua/media/study/bachelor-degree/pm/pm-courses-2019-2020.html	Робочі програми за планом 2019 року
8	bachelor	Прикладна математика	http://csc.knu.ua/media/study/bachelor-degree/pm/pm-courses-2020-2021.html	Робочі програми за планом 2020 року
9	bachelor	Прикладна математика	http://csc.knu.ua/media/study/bachelor-degree/pm/pm-courses-2022-2023.html	Робочі програми за планом 2022 року
10	bachelor	Системний аналіз	http://applstat.univ.kiev.ua/ukr/?templ=studies	Робочі програми за ОПП 2018
11	bachelor	Системний аналіз	http://applstat.univ.kiev.ua/ukr/?templ=studies	Робочі програми за ОПП 2019
12	bachelor	Системний аналіз	http://applstat.univ.kiev.ua/ukr/?templ=studies	Робочі програми за ОПП 2020, 2021
13	bachelor	Системний аналіз	http://applstat.univ.kiev.ua/ukr/3.php	Робочі програми за ОПП 2022
14	bachelor	Програмна інженерія	http://csc.knu.ua/media/study/bachelor-degree/courses-2016-2018.html	Робочі програми за планом 2018
15	bachelor	Програмна інженерія	http://csc.knu.ua/media/study/bachelor-degree/courses-2019-202x.html	Робочі програми за планом 2019
16	master	Інформатика	http://csc.knu.ua/media/study/master-degree/122-mag-inf-courses-2020-2021_PROLONG.html	Робочі програми для вступників 2021 року
17	master	Інформатика	http://csc.knu.ua/media/study/master-degree/122-mag-inf-courses-2022.html	Робочі програми для вступників 2022 року
18	master	Бізнес-Інформатика	http://csc.knu.ua/media/study/master-degree/122-bizinf-master-2021-prol.html	Робочі програми для вступників 2019-2021 років
19	master	Бізнес-Інформатика	http://csc.knu.ua/media/study/master-degree/122-bizinf-master-2022.html	Робочі програми для вступників 2022 року
20	master	Прикладна математика	http://csc.knu.ua/media/study/master-degree/113-pm-master-2021-2022.html	Робочі програми для вступників 2021 року

Рисунок 4.29 — Збережений файл

4.3.4 Завантаження даних з парсерів до бази даних

На даному етапі створення програми вже додана можливість парсити дані про навчальні системи, проглядати їх на сторінках адміністративної панелі та вивантажувати у файл. Наступним кроком буде можливість завантаження даних, що повертає парсер (окрім парсеру головної сторінки), до бази даних.

Послідовність дій для виконання цього завдання схожа з послідовністю з попереднього розділу. Потрібно створити та зареєструвати функції, які завантажують дані з парсеру, та додати кнопки. На головній сторінці з навчальними програмами додамо селектор для того, щоб обрати яку саме спеціальність парсити. Це необхідно тому що кожна спеціальність має різну структуру навчальної програми на сайті, а тому і парсери для них різні. Отже, додаємо випадаючий список з назвами спеціальностей та кнопку для переходу на детальну сторінку спеціальності:

```
<form method="post" action="{% url 'choose_parser' %}">
    {% csrf_token %}
    <div class="form-group">
        <label for="specialty">Choose a specialty:</label>
        <select name="specialty" id="specialty">
            <option value="System Analysis — 124">System Analysis — 124</option>
            <option value="Applied Mathematics — 113">Applied Mathematics — 113</option>
            <option value="Informatics — 122">Informatics — 122</option>
            <option value="Software Engineering — 121">Software Engineering — 121</option>
        </select>
    </div>
    <button type="submit" class="btn btn-primary btn-go-to-the-next-page">Go to this page!</button>
</form>
```

Рисунок 4.30 — Створення списку

Так це виглядає на сайті:

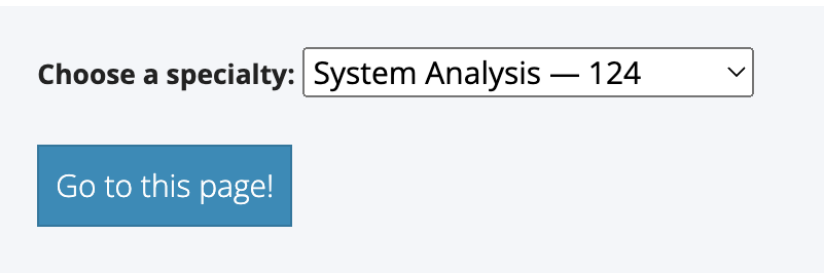


Рисунок 4.31 — Список спеціальностей на сторінці

Тепер додамо вид та зареєструємо його. В залежності від вибору спеціальності буде викликатись певний парсер:

```
@csrf_exempt
def choose_parser(request: Request) -> HttpResponse():
    specialty_name = request.POST.get("specialty", "")
    if specialty_name == "Informatics – 122":
        return show_inform_disciplines_with_parser_data(request)
    elif specialty_name == "Software Engineering – 121":
        return show_se_disciplines_with_parser_data(request)
    elif specialty_name == "Applied Mathematics – 113":
        return show_pm_disciplines_with_parser_data(request)
    elif specialty_name == "System Analysis – 124":
        return show_sa_disciplines_with_parser_data(request)
    else:
        return HttpResponse("Specialty not found!")
    return HttpResponse("OK")
```

Рисунок 4.32 — Вид для вибору парсера

Розглянемо наступні кроки за однією спеціальністю — “Інформатика”. Створюємо вид, де викликається відповідний парсер та повертається назва шаблону сторінки, яка має повернутись як результат.

```
def show_inform_disciplines_with_parser_data(request: Request, year: int = 19) -> HttpResponse():
    data = get_inform_page(year)
    context = {"data": data}
    return render(request, "inform_disciplines.html", context)
```

Рисунок 4.33 — Виклик парсеру спеціальності

Додаємо функціїї для кнопок “Завантаження інформації до файлу” та “Заповнення бази даних даними”. Реєструємо функції та додаємо до шаблону сторінки:

```
<a href="{% url 'download_inform_csv' %}" class="btn btn-primary">
  Download to CSV
</a>

<a href="{% url 'download_inform_data_to_db' %}" class="btn btn-primary">
  Download data to DB
</a>
```

Рисунок 4.34 — Додавання кнопок на сторінку

Тепер сторінка з парсингом спеціальності “Інформатика” має наступний вигляд:

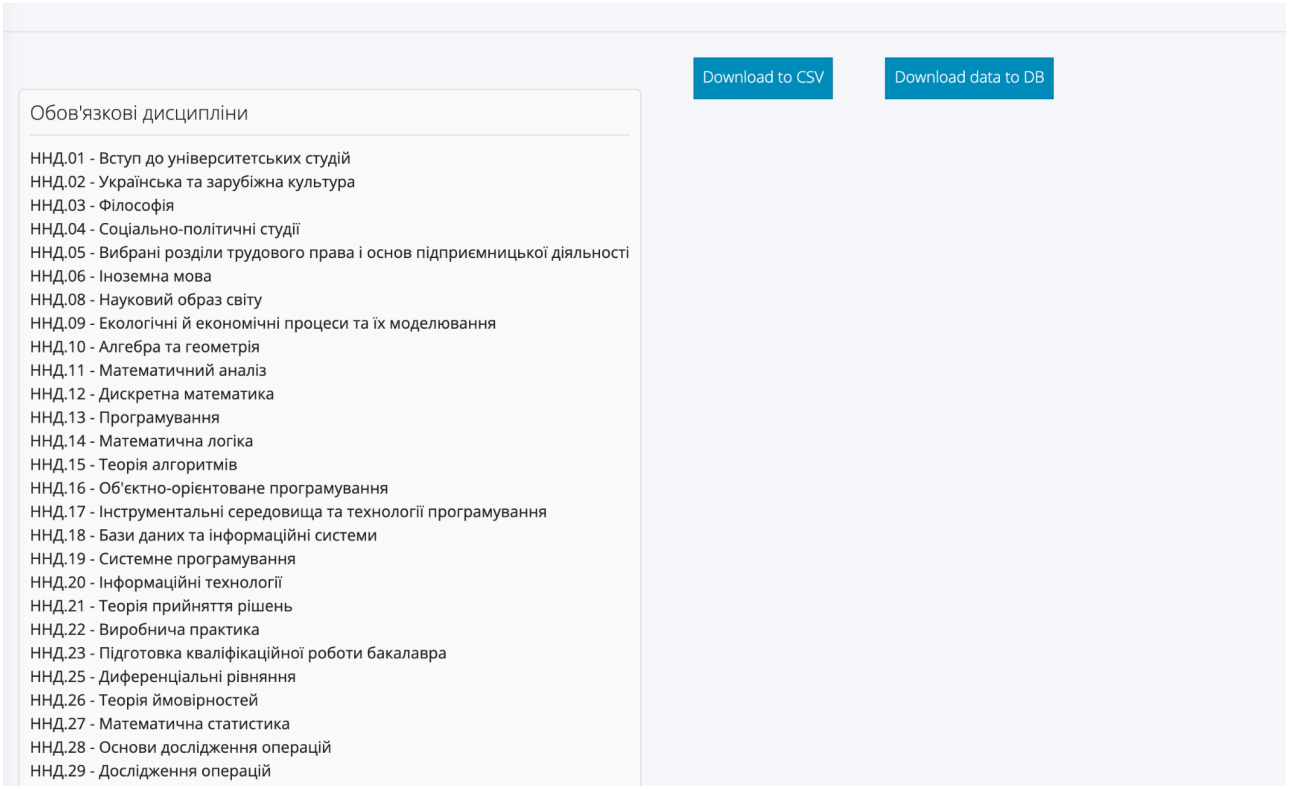


Рисунок 4.35 — Вигляд сторінки конкретної спеціальності

Перевіримо роботу кнопки “Завантажити дані до бази даних”. Ось сторінка таблиці Disciplines до того, як натиснути на кнопку:

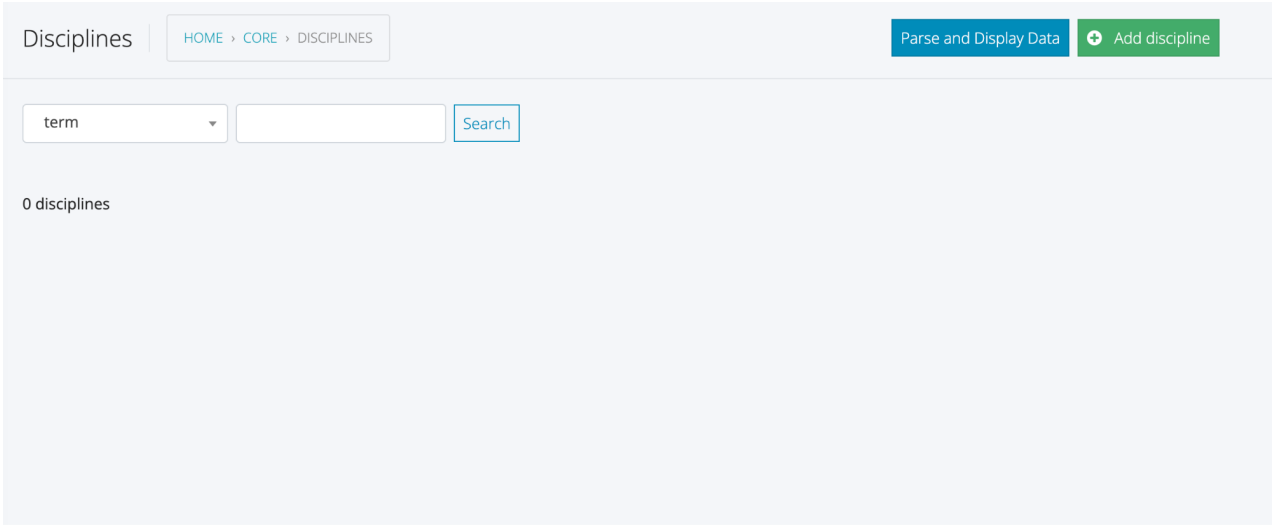


Рисунок 4.36 — Порожня таблиця спеціальності

Сторінка після запуску кнопки:

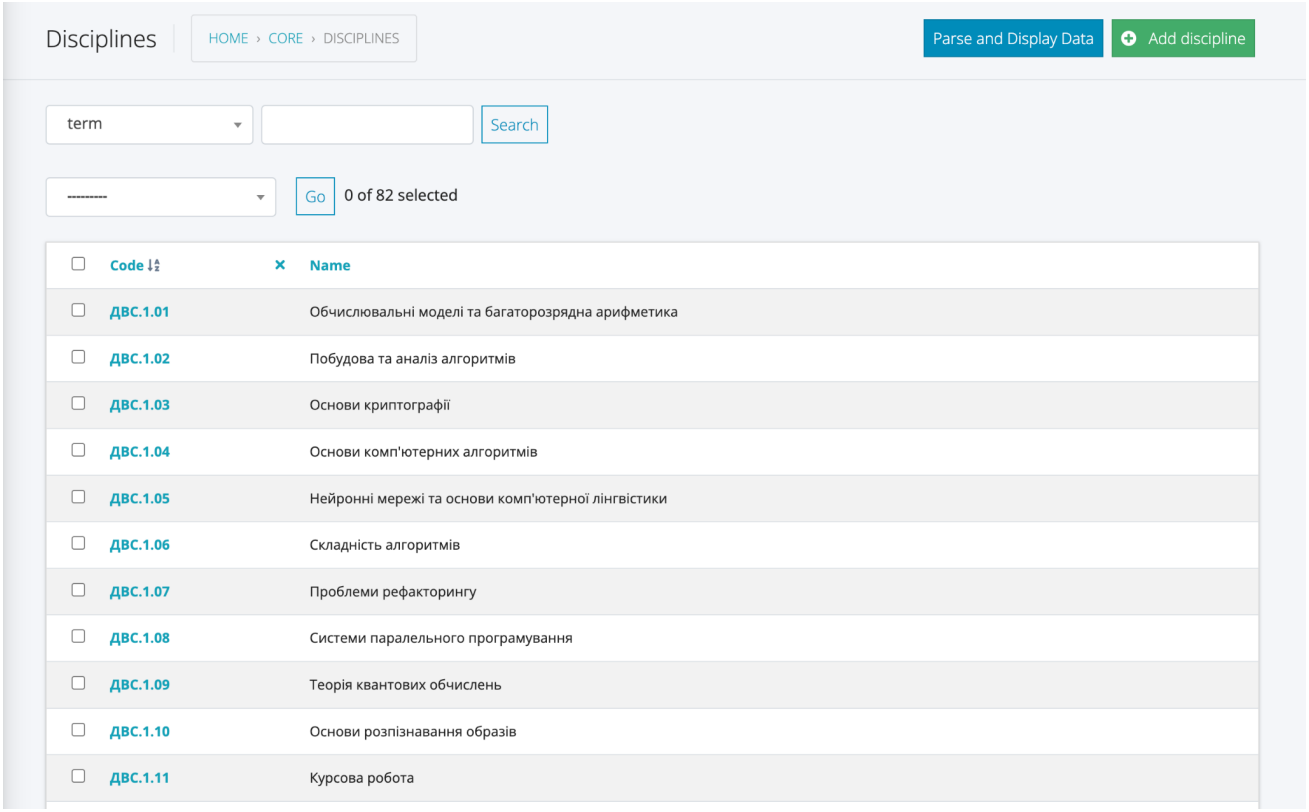


Рисунок 4.37 — Таблиця, заповнена даними з парсеру

4.3.5 Завантаження власних даних з файлу до бази даних

Для покращення системи буде корисною можливістю завантаження та вивантаження файлу до або з бази даних. Для виконання даного етапу використовуватимемо сторонню бібліотека django-import-export, оскільки за допомогою даного інструменту завантаження файлу може бути у різних форматах.

Спочатку варто встановити дану бібліотеку та додати “import_export” до INSTALLED_APPS у “settings.py” файлі. Створюємо файл “resources.py”, де прописуватимуться необхідні налаштування для кожної моделі, імпорт та експорт якої планується.

Створємо ресурси для кожної моделі. Приклад створення ресурсів для двох моделей:

```python
from import_export import resources, fields
from import_export.widgets import ForeignKeyWidget, ManyToManyWidget
from .models import Term, Discipline, SelectiveBlock, SelectiveList, Specialty


class TermResource(resources.ModelResource):
    class Meta:
        model = Term
        fields = ('id', 'number', 'value')


class DisciplineResource(resources.ModelResource):
    class Meta:
        model = Discipline
        skip_unchanged = True
        report_skipped = False
        import_id_fields = ("code",)
        fields = ("code", "name", "term")
        use_bulk = True
```

Рисунок 4.38 — Створення ресурсів для табличок

У вказаному коді існують обов’язкові поля, де вказується модель та поля, які імпортуються та експортуються, а також додаткові — яке поле вважати за унікальний ідентифікатор, чи пропускати ті записи, які вже є в базі даних та незмінені, тощо.

Для того щоб дані зміни почали виконуватись, необхідно також зареєструвати дані класи відповідно до кожної моделі у “admin.py” файлі. Наведемо приклад, яким чином це додається:

```python
@admin.register(Term)
class TermAdmin(ImportExportModelAdmin):
    resource_class = TermResource
    list_display = ["number", "value"]
    list_filter = ["number"]
    search_fields = ["number", "value"]
    ordering = ["number"]
```

Рисунок 4.39 — Реєстрація ресурсу

Тут рядки 7-9 такі, що зазнали змін. Робимо аналогічний крок з кожною моделлю. Тепер на кожній сторінці таблиці з’явилось дві кнопки:

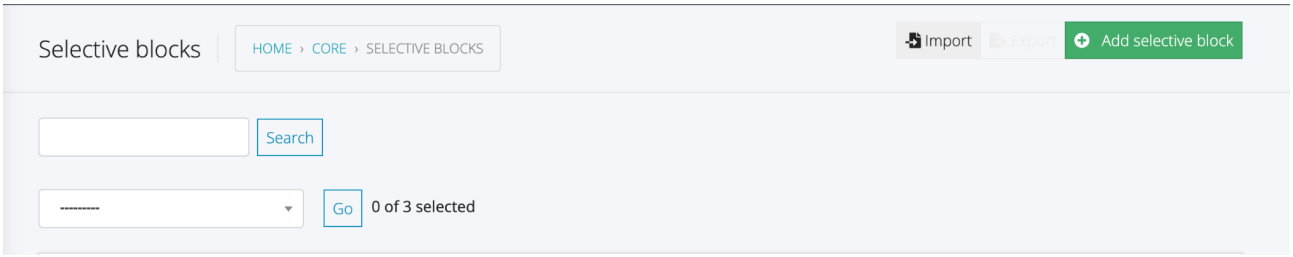


Рисунок 4.40 — Вигляд кнопки імпорту та експорту

Якщо натиснути на них, то отримаємо форму з вибором формату файлу:

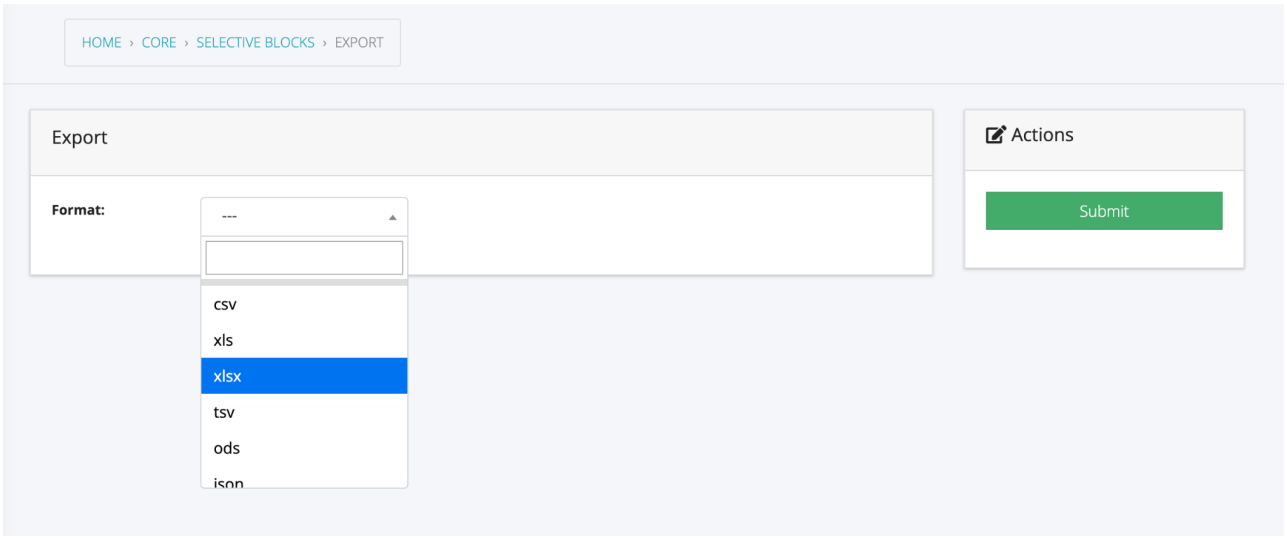


Рисунок 4.41 — Імпорт даних у файл

Вигляд збережених даних у файл формату .xlsx для таблиці Selective blocks:

	A	B	C	D	E	F	G	H
1	**disciplines**	**id**	**name**					
2	120,121,122,123,124,125,126,127,128,129,130	2	Вибірковий блок "Інтелектуальні інформаційні технології"					
3	142,143,144,145,146,147,148,149,150,151,152	4	Вибірковий блок "Інформаційні технології та системи"					
4	131,132,133,134,135,136,137,138,139,140,141	3	Вибірковий блок "Теорія та технологія програмування"					
5								
6								
7								

Рисунок 4.42 — Збережений файл

РОЗДІЛ 5 ТЕСТУВАННЯ

5.1 Методи та підходи до тестування системи

Тестування системи — важливий аспект будь-якого процесу розробки програмного забезпечення. Це процес перевірки функціональності програми для забезпечення того, що вона виконує свої завдання правильно. У контексті даного проекту, можуть бути використані декілька методів та підходів до тестування системи, які можуть бути особливо корисними:

- юніт-тестування: полягає у перевірці окремих компонентів програми. Це найнижчий рівень тестування і він сфокусований на перевірці правильності окремих функцій та методів. Django має вбудований фреймворк для юніт-тестування, який базується на стандартному фреймворку Python unittest. Це забезпечує універсальний набір інструментів для тестування різноманітних аспектів вашого вебдодатку;
- інтеграційне тестування: зосереджується на перевірці взаємодії між компонентами системи. Зокрема, в контексті Django це може включати тестування запитів до бази даних, маршрутизації URL, шаблонів та інших аспектів, які взаємодіють один з одним;
- функціональне тестування: цей вид тестування зосереджений на перевірці того, чи відповідає система своїм вимогам. Це включає перевірку того, що користувач може виконувати необхідні дії та досягати бажаних результатів;
- тестування продуктивності: при великому обсязі даних або при значному навантаженні важливо переконатись, що ваш вебсайт здатний ефективно працювати;
- тестування безпеки: Django є досить безпечним веб-фреймворком, але все одно важливо перевірити додаток на потенційні проблеми безпеки. Це може включати перевірку на вразливості до SQL-ін'єкцій та інших загроз.

В рамках даного проекту буде виконано модульне, функціональне та інтеграційне тестування, оскільки вони є найважливішими в контексті розробки веб-додатків, особливо на ранніх стадіях проекту.

5.2 Результати модульного тестування

Модульне тестування це процес перевірки окремих частин коду (модулів, функцій, методів, класів). Це найнижчий рівень тестування, який забезпечує, що кожна окрема частина коду працює правильно в ізоляції. Модульне тестування дає впевненість, що кожна частина коду працює правильно окремо. Це корисно для виявлення і виправлення помилок на ранній стадії розробки, що може суттєво заощадити час та зусилля у майбутньому.

Модульне тестування зазвичай здійснюється для функцій або методів, які виконують якесь конкретне завдання. Тому напишемо тести для видів.

Будемо використовувати функцію “mock” для симулювання відповідей від функцій парсингу.

В першу чергу створимо клас для тестування та метод “setUp” — метод, який викликається перед кожним тестом. В цьому методі створюються екземпляри класів RequestFactory і Client, які використовуються для створення тестових запитів і взаємодії зі стороною сервера.

```
9  class TestViews(TestCase):
10     def setUp(self):
11         self.factory = RequestFactory()
12         self.client = Client()
```

Рисунок 5.1 — Створення класу для тестування

Напишемо декілька тестів на виконання GET-запит до URL:

```python
    def test_parse_main_page(self):
        response = self.client.get(reverse('parse_and_display_data'))
        self.assertEqual(response.status_code, 200)

    def test_download_main_page_to_csv(self):
        response = self.client.get(reverse('download_csv'))
        self.assertEqual(response.status_code, 200)
```

Рисунок 5.2 — Тести на GET-запит

Також, додамо тести на виконання POST-запитів:

```python
    def test_choose_parser(self):
        response = self.client.post(reverse('choose_parser'), {'specialty': 'Informatics – 122'})
        self.assertEqual(response.status_code, 200)
```

Рисунок 5.3 — Тест на POST-запит

Напишемо тест з використанням мок-функції. Цей тест використовуватиме “@patch” декоратор для мокування функції “get_main_page”. Мок-об'єкт “mock_get_main_page” замінює справжню функцію “get_main_page” у контексті тесту. Далі у коді задаються значення, які поверне мок-об'єкт, і виконується функція “download_main_page_to_csv”. Потім перевіряється, чи код відповіді є 200, а також вміст відповіді, який містить певні дані:

```python
    @patch('core.views.get_main_page')
    def test_download_main_page_to_csv_with_mock(self, mock_get_main_page):
        mock_get_main_page.return_value = {
            "Bachelor": [("Name1", [("URL1", "Desc1")]), ("Name2", [("URL2", "Desc2")])],
            "Master": [("Name3", [("URL3", "Desc3")]), ("Name4", [("URL4", "Desc4")])],
        }
        request = self.factory.get('download_csv')
        response = download_main_page_to_csv(request)
        self.assertEqual(response.status_code, 200)
        content = response.content.decode('utf-8')
        self.assertIn("Bachelor,Name1,URL1,Desc1", content)
        self.assertIn("Bachelor,Name2,URL2,Desc2", content)
        self.assertIn("Master,Name3,URL3,Desc3", content)
        self.assertIn("Master,Name4,URL4,Desc4", content)
```

Рисунок 5.4 — Тест з використанням мок-функції

Напишемо ще декілька тестів з використанням мок-функцій. Наприклад, тест, який також використовує мок-об'єкт “mock_get_main_page” для мокування

функції “get_main_page”. У коді задається значення, яке поверне мок-об'єкт, і виконується функція “choose_parser”. Перевіряється, чи код відповіді є 400, що вказує на помилку у запиті, оскільки не було передано значення спеціальності.

```
52      @patch('core.views.get_main_page')
53      def test_choose_parser_with_no_specialty(self, mock_get_main_page):
54          mock_get_main_page.return_value = "Your mock value"
55          request = self.factory.post('choose_parser')
56          response = choose_parser(request)
57          self.assertEqual(response.status_code, 400)
```

Рисунок 5.5 — Тест, який очікує неуспішний статус виконання функції

Це приклади основних тестів, було написано ще схожі тести за таким самим принципом. Також, було додано тести для парсерів. Результат виконання тестів:

```
(venv) ➜  educationalManagementSystem git:(main) ✗ python manage.py test
Found 22 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
......................
----------------------------------------------------------------------
Ran 22 tests in 0.459s

OK
Destroying test database for alias 'default'...
```

Рисунок 5.6 — Результат виконання тестів

5.3 Функціональне тестування

Під час розробки та реалізації системи управління освітніми програмами було проведено функціональне тестування для перевірки відповідності реалізованого програмного продукту вимогам, вказаним в розділі 3, що були поставлені за мету. Ці вимоги охоплювали функціональну адміністративну систему.

Функціональне тестування було здійснено за допомогою модульного тестування, використовуючи бібліотеку unittest, яка є частиною стандартної

бібліотеки Python. Кожен тест був створений для окремої функції або методу, який повинен був виконати певну задачу в рамках системи.

Адміністративна підсистема була тестована з метою перевірки ефективності управління базою даних, включаючи запуск парсингу, роботу з інформацією в адміністративній панелі та обробку інформації (фільтрування, сортування та пошук). Було також проведено тестування на можливість створення нових користувачів системи з адміністраторськими правами та доступами.

Тестування системи включало в себе перевірку реакції на неправильні дані, наприклад як відсутність вибору спеціальності в тесті "test_choose_parser_with_no_specialty", що дозволило переконатися в коректності обробки помилок системою.

Також було проведено мок-тестування, яке дозволило імітувати відповіді від зовнішніх ресурсів та сервісів, як, наприклад, в тестах "test_download_main_page_to_csv_with_mock" та "test_choose_parser_with_mock".

Загалом, розроблена система успішно пройшла всі стадії функціонального тестування, підтверджуючи свою готовність до впровадження та використання в освітньому процесі.

Важливо відзначити, що процес тестування був ітеративним: при виявленні будь-яких проблем або відхилень від вимог, відповідні частини системи були кориговані та повторно тестовані до досягнення бажаного результату.

Усі тести проходять успішно, що свідчить про високу якість реалізованого програмного продукту та його відповідність вимогам технічного завдання.

5.4 Виконання інтеграційного тестування

В ході розробки системи управління освітніми програмами було проведено інтеграційне тестування для перевірки взаємодії різних компонентів системи та їх відповідність вимогам, визначеним в розділі 3.

Інтеграційне тестування полягало у перевірці правильності взаємодії між рівнями системи: рівнем даних, рівнем бізнес-логіки та рівнем представлення.

Під час інтеграційного тестування було перевірено наступні сценарії:

а) запуск системи та звернення до бази даних:

- перевірка з'єднання з базою даних та доступу до неї;
- виконання запитів до бази даних для отримання та зміни інформації;

б) взаємодія між рівнем бізнес-логіки та рівнем представлення:

- передача та обробка даних між серверними компонентами та клієнтськими компонентами;
- відображення та оновлення даних на стороні клієнта через інтерфейс користувача;

в) інтеграція з зовнішніми сервісами:

- перевірка звернення до зовнішніх ресурсів, таких як сервіс ElephantSQL для роботи з базою даних;
- перевірка коректного отримання та обробки даних з зовнішніх джерел під час парсингу.

В процесі інтеграційного тестування переконалися в тому, що всі компоненти системи взаємодіють правильно, дані передаються без помилок та зберігаються коректно в базі даних. Було також перевірено реакцію системи на неправильні дані та ситуації, що можуть виникнути в реальному використанні.

В результаті інтеграційного тестування було підтверджено, що різні компоненти системи працюють разом як очікувалося, відповідають вимогам поставленого завдання та забезпечують коректну та надійну роботу системи управління освітніми програмами.

ВИСНОВКИ

Розроблено систему управління освітніми програмами, яка дозволяє викладачам з правами адміністратора зберігати, обробляти та оновлювати інформацію про освітні програми, дисципліни та іншу пов'язану інформацію.

Використано трирівневу архітектуру, що включає рівні даних, бізнес-логіки та представлення. Рівень даних реалізовано з використанням PostgreSQL як реляційної СУБД. Рівень бізнес-логіки реалізовано з використанням мови програмування Python та фреймворку Django. Рівень представлення забезпечено за допомогою адміністративної панелі Django, налаштованої за допомогою бібліотеки jazzmin.

Проведено модульне тестування для того, щоб перевірити, що результати відповідають очікуванням. Написання даних тестів необхідно для уникнення помилок в майбутньому та для збереження стану програми, щоб не було збоїв за рахунок необачних змін.

Здійснено функціональне тестування, яке підтвердило відповідність розробленої системи вимогам технічного завдання. Успішно протестовано всі функції та задачі, які були заплановані для реалізації.

Проведено інтеграційне тестування для перевірки взаємодії різних компонентів системи. В результаті тестування підтверджено правильну взаємодію між рівнями системи та інтеграцію з зовнішніми сервісами.

Розроблена система управління освітніми програмами відповідає всім вимогам, поставленим на початку виконання роботи та готова до впровадження в реальне середовище. Система є гарним інструментом для ефективного управління освітніми програмами та забезпечує зручний доступ до інформації для користувачів.

Розроблена система може бути використана в реальних умовах та сприятиме ефективному управлінню освітніми програмами.

Перспективи виконаної роботи:

- підвищення гнучкості системи задля можливості використання в різних закладах вищої освіти;
- забезпечення можливостей внесення пропозицій щодо оновлення освітніх програм;
- інтеграція з іншими системами, які вже впроваджені.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Освітні програми [Електронний ресурс] // Державна служба якості освіти України. – Режим доступу: https://sqe.gov.ua/diyalnist/osvitni-programi/.
2. Програми навчальних дисциплін [Електронний ресурс] // Факультет комютерних наук та кібернетики. – Режим доступу: https://csc.knu.ua/uk/programs.
3. Методичні рекомендації щодо перегляду та оновлення освітніх програм у Полтавському національному педагогічному університеті ім. В.Г. Короленка / Затверджено вченою радою ПНПУ імені В.Г. Короленка, протокол 11 від 24 березня 2022 року — 7 с.;
4. Що таке компетентнісний підхід у навчанні — відповідає Державна служба якості освіти [Електронний ресурс] // Нова українська школа. — Режим доступу: https://nus.org.ua/questions/zo-take-kompetentnisnyj-pidhid-u-navchanni-vidpovidaye-derzhavna-sluzhba-yakosti-osvity.
5. Moodle [Електронний ресурс] // Moodle. – Режим доступу: https://moodle.com/.
6. Організація дистанційного навчання в Moodle [Електронний ресурс] // Освіта.ua. – Режим доступу: https://osvita.ua/vnz/high_school/72285/.
7. Anthology + Blackboard [Електронний ресурс] // Blackboard. – Режим доступу: https://www.blackboard.com/.
8. What Is Blackboard Learn? [Електронний ресурс] // Blackboard Help Center. – Режим доступу: https://help.blackboard.com/Learn/Instructor/Ultra/Getting_Started/What_Is_Blackboard_Learn.
9. Розробка ecommerce проектів technologies postgreSQL [Електронний ресурс] // Brander. – Режим доступу: https://brander.ua/technologies/postgresql.

10. ElephantSQL [Електронний ресурс] // ElephantSQL. – Режим доступу: https://www.elephantsql.com/.
11. Why We Use Django Framework & What Is Django Used For [Електронний ресурс] // Djangostars. – Режим доступу: https://djangostars.com/blog/why-we-use-django-framework/.
12. Django Jazzmin Documentation [Електронний ресурс] // Jazzmin. – Режим доступу: https://django-jazzmin.readthedocs.io/.
13. Що це і де вони використовуються? Парсери [Електронний ресурс] // Avada Media TV. – Режим доступу: https://avada-media.ua/ua/services/parser/.

ДОДАТОК А
Приклад створених моделей

```python
class SelectiveBlock(models.Model):
    name = models.CharField(max_length=255)
    disciplines = models.ManyToManyField(Discipline)


class SelectiveList(models.Model):
    name = models.CharField(max_length=255)
    first_discipline = models.ForeignKey(
        Discipline, related_name="first_discipline", on_delete=models.CASCADE
    )
    second_discipline = models.ForeignKey(
        Discipline, related_name="second_discipline", on_delete=models.CASCADE
    )


class Specialty(models.Model):
    class SpecialtyChoices(models.TextChoices):
        SYSTEM_ANALYSIS = "System Analysis – 124"
        APPLIED_MATHEMATICS = "Applied Mathematics – 113"
        INFORMATICS = "Informatics – 122"
        SOFTWARE_ENGINEERING = "Software Engineering – 121"

    name = models.CharField(max_length=255, choices=SpecialtyChoices.choices)
    year_of_admission = models.DateField()
    mandatory_disciplines = models.ManyToManyField(
        Discipline, related_name="mandatory_disciplines"
    )
    selection_by_blocks = models.ManyToManyField(SelectiveBlock)
    selection_from_the_list = models.ManyToManyField(SelectiveList)

    class Meta:
        verbose_name = "Specialty"
        verbose_name_plural = "Specialties"
```

ДОДАТОК Б
Шаблон HTML-сторінки

```
{% extends "admin/base_site.html" %}

{% block extrastyle %}
<style>
    .grid-container {
        display: grid;
        grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
        gap: 20px;
    }
    .grid-item h5 {
        border-bottom: 1px solid #ddd;
        padding-bottom: 10px;
        margin-bottom: 10px;
    }
    .grid-item ul {
        padding: 0;
        list-style-type: none;
    }
    .for-title {
        margin-right: -150px;
        margin-left: 50px;
    }
</style>
{% endblock %}

{% block content %}
{% for key, value_list in data.items %}
    <h5 class="for-title">{{ key|title }} degree</h5>
    <div class="grid-container">
        {% for name, links in value_list %}
            <div class="grid-item">
              <h5>{{ name }}</h5>
                <ul>
                    {% for url, description in links %}
                        <li><a href="{{ url }}">{{ description
}}</a></li>
                    {% endfor %}
                </ul>
            </div>
        {% endfor %}
    </div>
{% empty %}
    <p>Даних не знайдено.</p>
{% endfor %}

{% endblock %}
```