

Національний університет «Львівська політехніка»
Кафедра програмного забезпечення

КУРСОВА РОБОТА

з дисципліни «Бази даних»

на тему:

«Інформаційна система для складських приміщень книг»

Виконав:

студент спеціальності 121

«Інженерія програмного забезпечення»

групи ПЗ-31

Бабіля О. О.

Керівник:

асистент кафедри програмного забезпечення

Білоіваненко М.В.

Оцінка:

Національна шкала _____

Кількість балів _____ Оцінка ECTS _____

Члени комісії

(підпис)

(підпис)

Білоіваненко М.В.

Цимбалюк Т.М.

Львів – 2024 рік

Зміст

Розділ 1. Аналіз предметної області та постановка завдання.....	3
1. Опис предметної області.....	3
2. Вимоги до обробки даних.....	4
3. Постановка завдання	4
Розділ 2. Розробка моделей зберігання даних	5
1. Концептуальне проектування	5
2. Вимоги до системи накопичення даних	5
3. Логічне проектування схеми бази даних	7
4. Реалізація процедур бізнес-логіки	7
Розділ 3. Розробка програмного коду.....	7
1. Обґрунтування обраної архітектури	7
2. Структурна модель інформаційної системи.....	9
3. Призначення модулів та компонентів системи	9
4. Особливості реалізації та елементи інтерфейсу	10
Розділ 4. Наповнення бази даних	12
1. Опис джерела історичних даних	12
2. Опис процесів генерації тестових даних.....	12
3. Трансформація та первинне завантаження даних	12
Розділ 5. Функціональні можливості для користувача.....	13
1. Опис інтерфейсу у відповідності до бізнес-процесів	13
2. Засоби аналітичного представлення даних	16
3. Засоби експорту даних	17
4. Засоби програмного інтерфейсу.....	17
Висновки.....	18
Список літератури.....	19
Додатки	19
Скрипт створення бази даних	19
Скрипт завантаження історичних даних	47
Інший програмний код	49
Інший код створених моделей	49

Розділ 1. Аналіз предметної області та постановка завдання

1. Опис предметної області

Додаток для управління складськими приміщеннями книг є важливим інструментом у сучасному книготорговельному бізнесі. Враховуючи розвиток технологій та зростання обсягів книжкової продукції, ефективне управління складським простором стає вирішальним для забезпечення якості обслуговування клієнтів та оптимізації бізнес-процесів.

Додаток для складу книг може перетворити щоденні операції на більш ефективні та продуктивні. Від отримання книг на склад, списанню книг, до трансферу книг на інші доступні складські приміщення, кожен етап може бути детально задокументований та оптимізований.

Ключові функції інформаційної системи включають:

1. Інвентаризація книг: Можливість реєструвати та відстежувати кожен примірник книги на складі за допомогою унікальних ідентифікаційних номерів.

2. Управління простором: Можливість відстежувати доступні складські приміщення, їх вміст, щоб оптимізувати інвентаризацію книг.

3. Управління користувачами: Функціонал для додавання нових користувачів та їх груп прав, подальше відстеження дій.

4. Керування подіями: Можливість швидко та зручно зареєструвати подію, пов'язану з ключовими діями на складському приміщенні(прибуття, відправка, трансфер), додати відповідальну особу, відстежити зміни в інвентарі книг.

Специфікація важливих понять:

1. Прийом книг: даний процес включає приймання нових книг на склад.

2. Передача на інший склад: даний процес ідентифікує подію передачі криш з одного складу на інший. Це може бути викликано різними причинами, включаючи потребу в оптимізації запасів або задоволення попиту в інших регіонах.

3. Списання: Ця подія означає вилучення книг зі складу через їх застарілість, пошкодження або інші причини.

Даний додаток дає змогу книготорговельним компаніям ефективно керувати своїми складськими приміщеннями, забезпечуючи швидку обробку подій, точний облік запасів та дій користувачів.

В результаті, підвищується якість обслуговування та ефективність управління, що сприяє збільшенню конкурентоспроможності та успішності бізнесу.

2. Вимоги до обробки даних

Інформаційна система включає в себе наступні бізнес-процеси:

- Робота із даними користувачів та їх груп – внесення нових користувачів, зберігання особистої інформації, додавання їх до відповідних груп, обробка прав груп
- Управління ресурсами складу – зберігання інформації про складське приміщення, наявних книг та їх кількості.
- Управління подіями – створення та відслідковування подій, призначення відповідних людей, складських приміщень відповідних книг та їх кількість.
- Управління книг та залежної інформації – зберігання повної інформації про книги відповідних авторів, жанрів та видавництв.

3. Постановка завдання

Система повинна володіти наступними модулями:

1. Модуль складських приміщень:
 - Додавання в інформаційну систему складських приміщень.
 - Збереження історії подій списання, перевезення та отримання книг.
 - Можливість відслідковування історії відповідних подій.
2. Модуль управління персоналом:
 - Внесення та облік особистих даних працівників.
 - Облік групи та спеціалізацій співробітників та відповідних прав.
 - Відслідковування подій конкретної особи або групи осіб.
3. Модуль подій:

- Створення та управління подій списання, перевезення та отримання книг.
 - Виконання подій.
4. Модуль управління книг:
- Створення, видалення та редагування книг, відповідних авторів та видавництв.
 - Зберігання історії змін книг відповідних авторів та видавництв.

Розділ 2. Розробка моделей зберігання даних

1. Концептуальне проектування

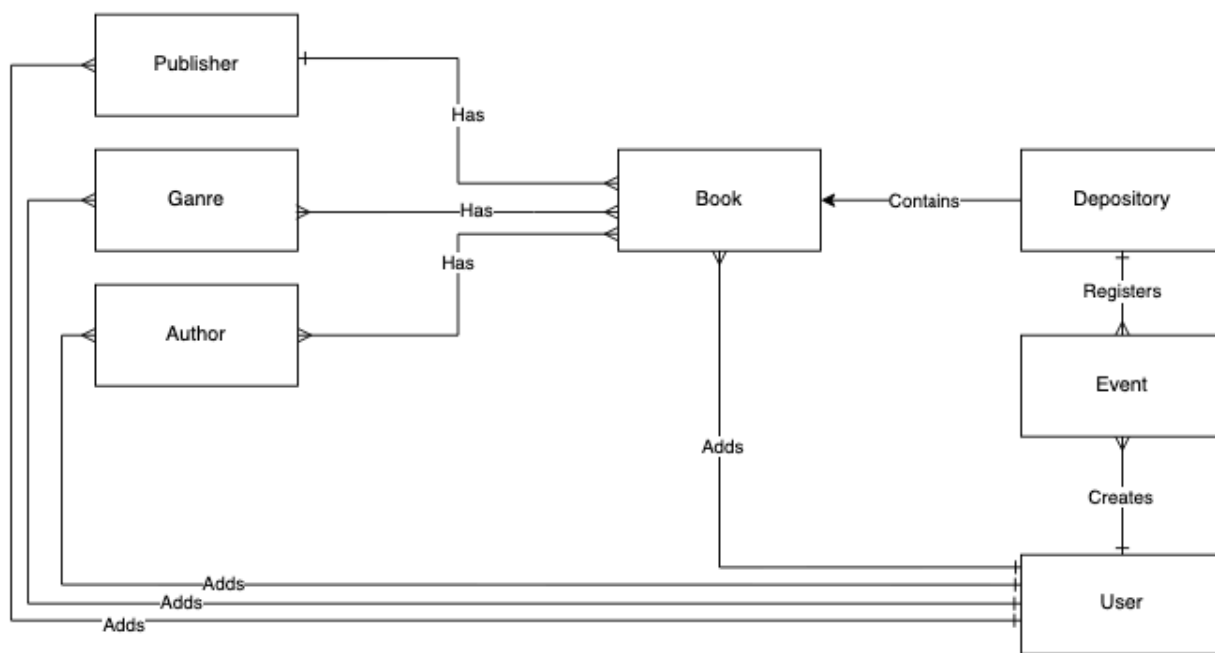


Рис. 2.1. ER-діаграма бази даних

2. Вимоги до системи накопичення даних

Найбільший обсяг даних припадатиме на дані про книги та події. Інформація про авторів та видавництв це теж доволі об'ємна частина системи, оскільки кожна книга має відповідного автора та книгу.

Найменший обсяг даних описуватиме наявні складські приміщення, оскільки це найбільший фізичний об'єкт. В залежності від величини підприємства обсяги даних можуть змінюватись.

Інформація, що додаватиметься часто - це події.

Інформація, що додаватиметься періодично: книги, автори та видавництва.

Інформація, що додаватиметься рідко: працівники та їх групи.

Очікуваними є запити:

- Додавання даних книг, авторів, видавництв, складів, подій та звітів.
- Зміна даних книг, авторів, видавництв, складів.

Очікувані вибірки:

- Вибірка книг, авторів, видавництв
- Вибірка подій за певною датою
- Вибірка подій за певним типом
- Вибірка подій за відповідним користувачів
- Вибірка користувачів
- Вибірка складських приміщень за відповідним фільтром

Обмеження доступу реалізовано на основі трьох основних ролей.

Консультант.

Керує інформацією про книг, видавництв та авторів.

Менеджер.

Має всі права консультанта та додатково має можливість керувати складами та подіями.

Адміністратор.

Має всі права менеджера та додатково має можливість керувати користувачами та їх групами, логами.

3. Логічне проектування схеми бази даних

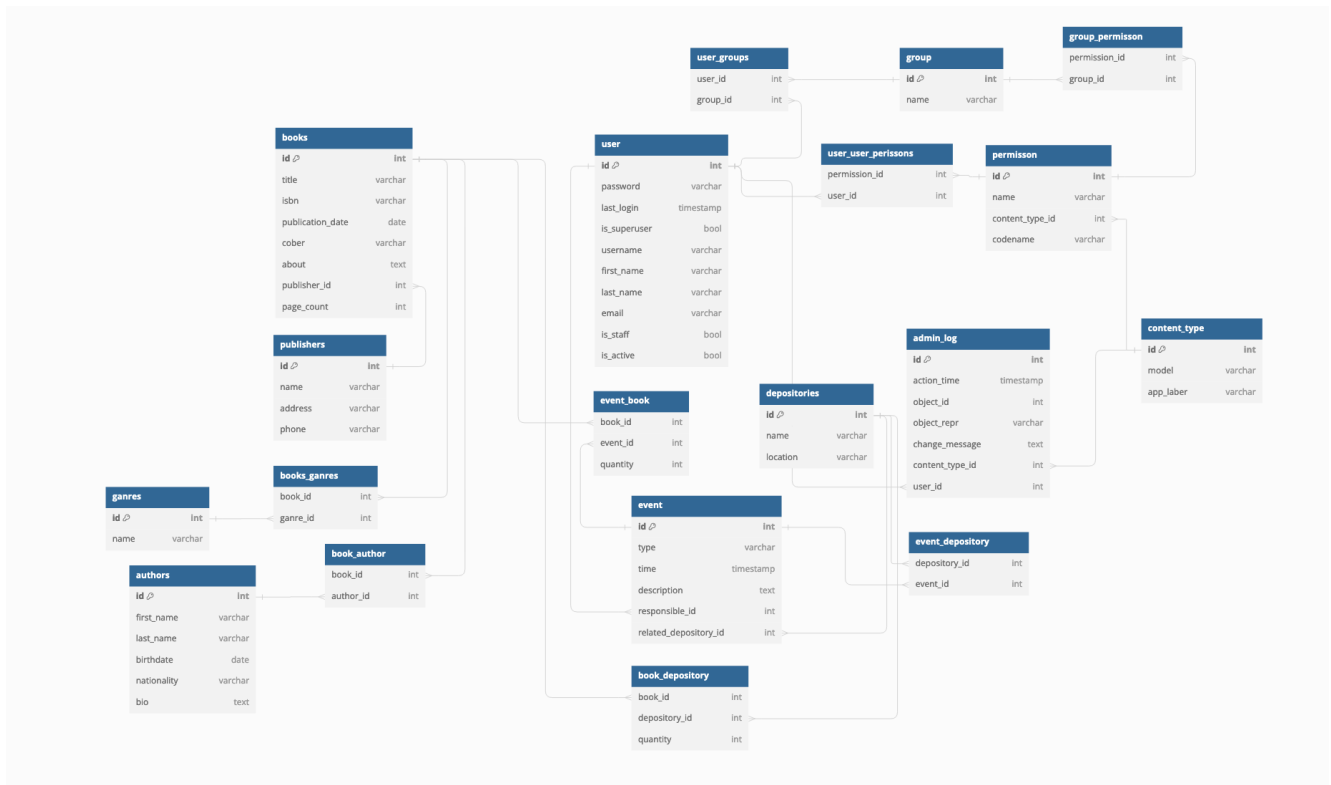


Рис. 2.2. Схема бази даних

4. Реалізація процедур бізнес-логіки

Процедури:

- Додавання нового автора
- Додавання нової книги
- Додавання нового видавництва

Розділ 3. Розробка програмного коду

1. Обґрунтування обраної архітектури

Для розробки інформаційної системи було обрана архітектура MVP (Minimum Viable Product) та подальша реалізація в клієнт-серверної архітектури, за допомогою MVC (Model-View-Controller). Ці рішення були обрані роблячи акцент на створенні базової версії продукту, фокусуючись на основних функціях продукту, необхідних для релізу.

Основним інструментом для виконання була мова програмування Python та її фреймворки Django та Django REST Framework, для створення архітектури, яка підпадає під привила REST. Через велику кількість засобів реалізованих цими фреймворками, а саме:

1. Базові класи для створення моделей, вюсетів, серіалайзерів, роутерів і т. д..

2. Django ORM(Object-Relational-Mapping), надає зручний спосіб взаємодії з базою даних, за допомогою об'єктно-орієнтованого підходу. Управління міграціями запитами та сигналами.

3. Базова адміністративна панель, яка має змогу розширюватись кастомним кодом.

4. Зручна серіалізація та валідація даних.

Також в екосистемі цих технологій вже реалізовано багато функцій, які допоможуть нам зберегти час при розробці функціоналу додатку, який дотримується MVP архітектури.

Для зберігання даних була обрана СУБД PostgreSQL. Цей вибір був зроблений, базуючись на гнучкості та масштабованості, що є головним фактором при зберіганні та опрацюванні великої кількості даних у нашому додатку. Відкритий код PostgreSQL допомагає спільноті користувачів, швидко реагувати на помилки та прогалини в системі. Також, у порівнянні з конкурентами, дане рішення має велику кількість вбудованих функцій, та засобів відгадки.

Базуючись на RESTful архітектурі було обрано JSON формат та протокол HTTP, для комунікації з сервісами. Запити виконують за допомогою основних методів, а саме: GET, POST, PUT, PATCH, DELETE.

2. Структурна модель інформаційної системи

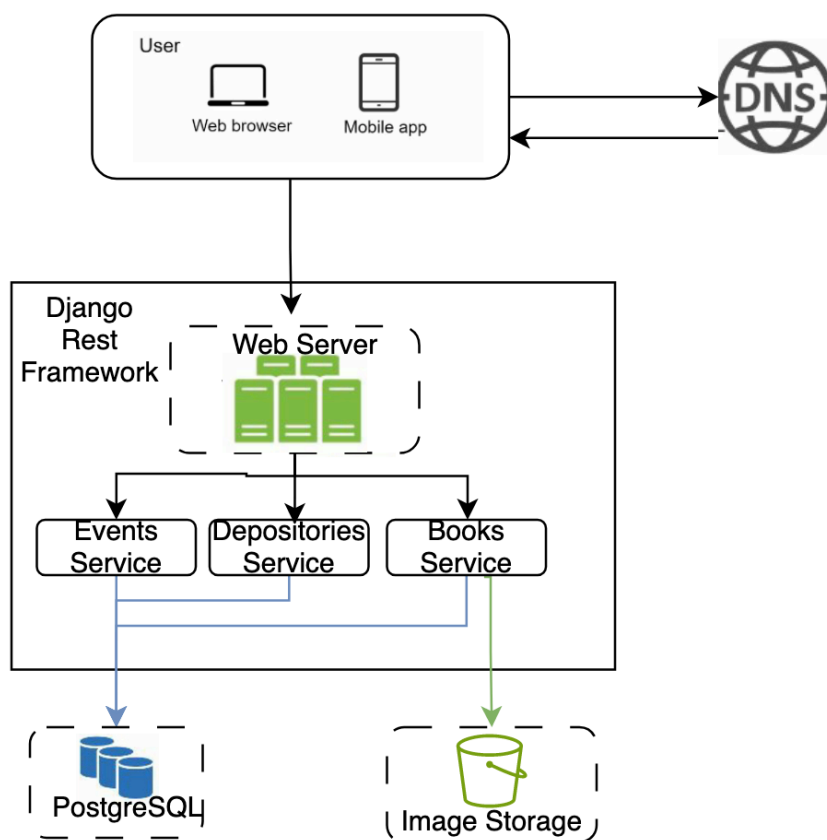


Рис. 3.1. Структура системи

3. Призначення модулів та компонентів системи

Система розділена на три модулі.

Events Service

Даний сервіс відповідає за взаємодію з подіями(прийом, списання, трансфер і так далі). Дані зберігаються в базі, тому даний сервіс взаємодіє з нею.

Depositories Service

Сервіс, який відповідає за складські приміщення, взаємодія з базою забезпечує зберігання книг які відносяться до складу.

Books Service

Цей сервіс відповідає за взаємодію з книгами, їх авторами та видавництвами. Інформація зберігається в СУБД. Оскільки книга має обкладинку, тому необхідно зберігати зображення. Для цього створена взаємодія з файловим сховищем.

4. Особливості реалізації та елементи інтерфейсу

Головною особливістю інтерфейсу є динамічне будування сторінки, в залежності від користувача та його групи. На серверній частині та на стороні СУБД були реалізовані 3 групи: адміністратор, менеджер та консультант (більш детально описано в пункті 2.2). Користувач після автентифікації до системи, бачить лише ті модулі, до яких він має доступ. Перевага такого підходу є простий захист інформації від користувачів сторонніх груп. Користувач не зможе змінити того, чого не бачить. Але якщо користувач спробує відправити запит самостійно, без взаємодії з інтерфейсом, то в такому випадку вже на серверній частині буде перевірка прав та доступів відповідно групи користувачів до яких він належить.

Було реалізовано наступні елементи інтерфейсу відповідно до ролей:

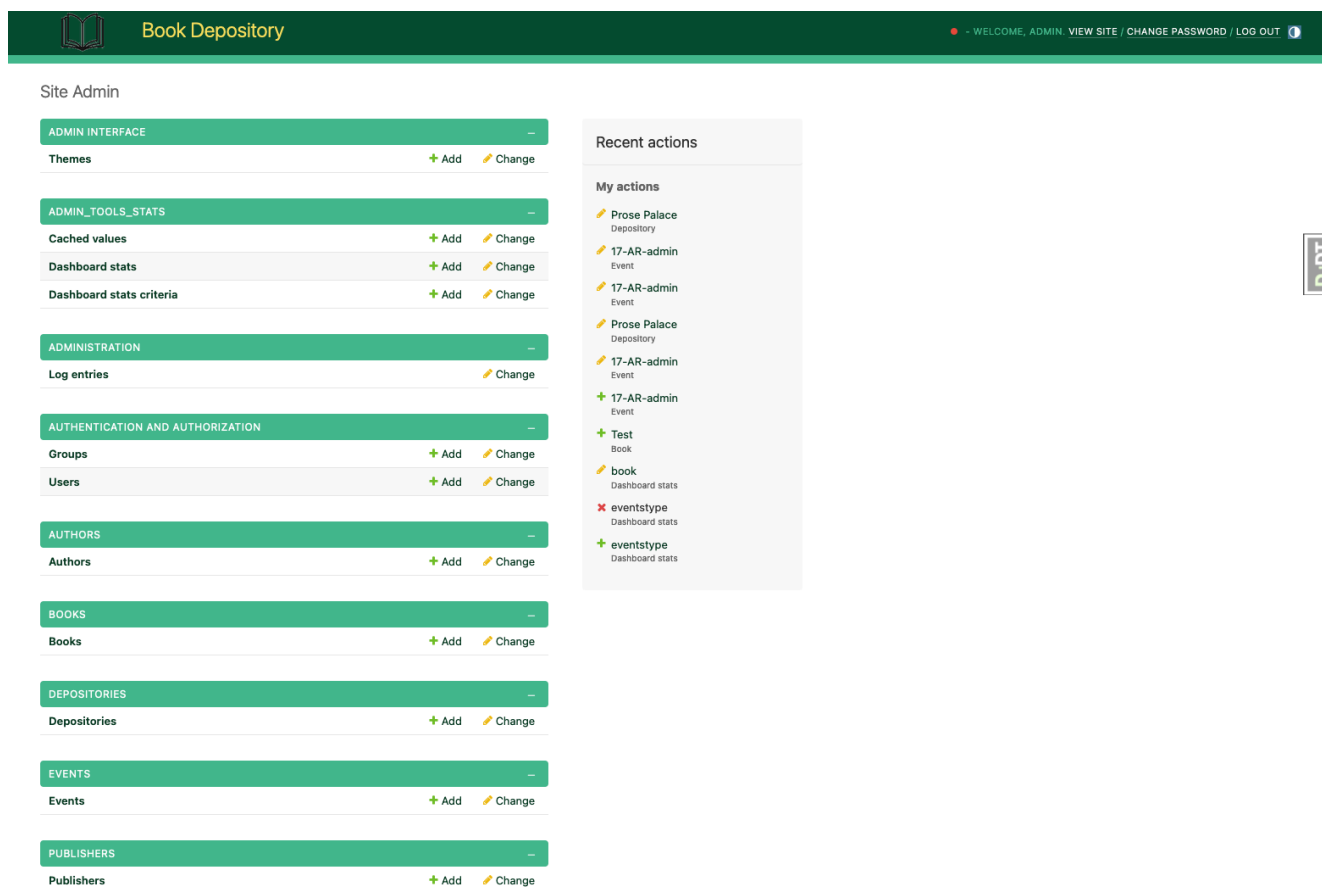


Рис. 3.2. Інтерфейс для взаємодії адміністратора(є доступ до всіх модулів додатку)

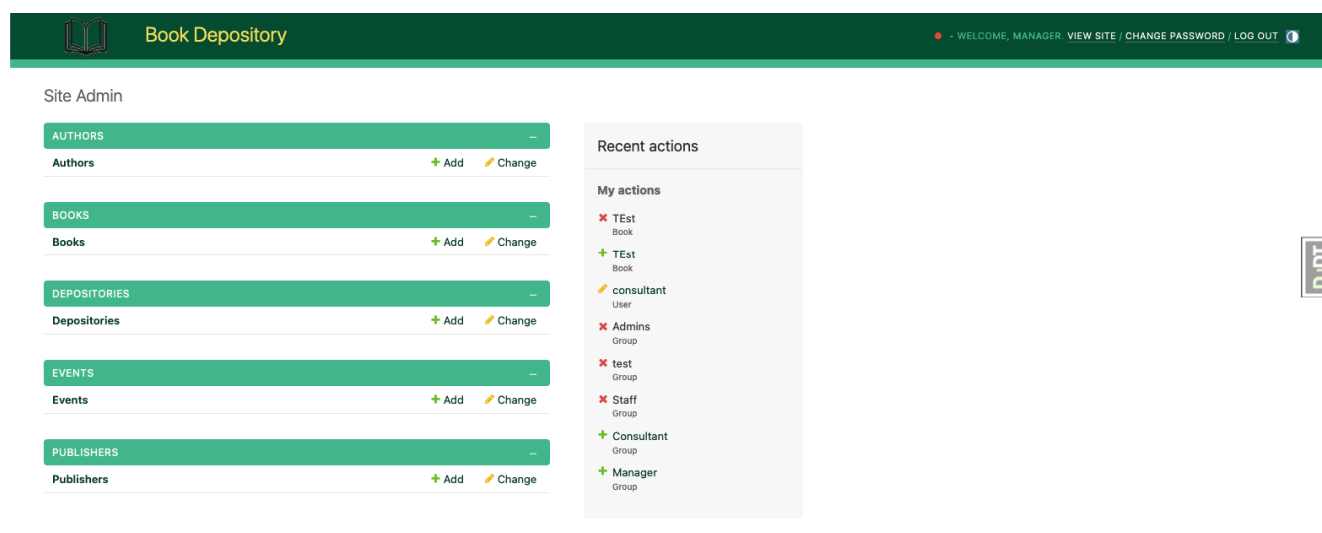


Рис. 3.3. Інтерфейс для взаємодії менеджера(нема доступу до логів, статистики, користувачів та їх груп)

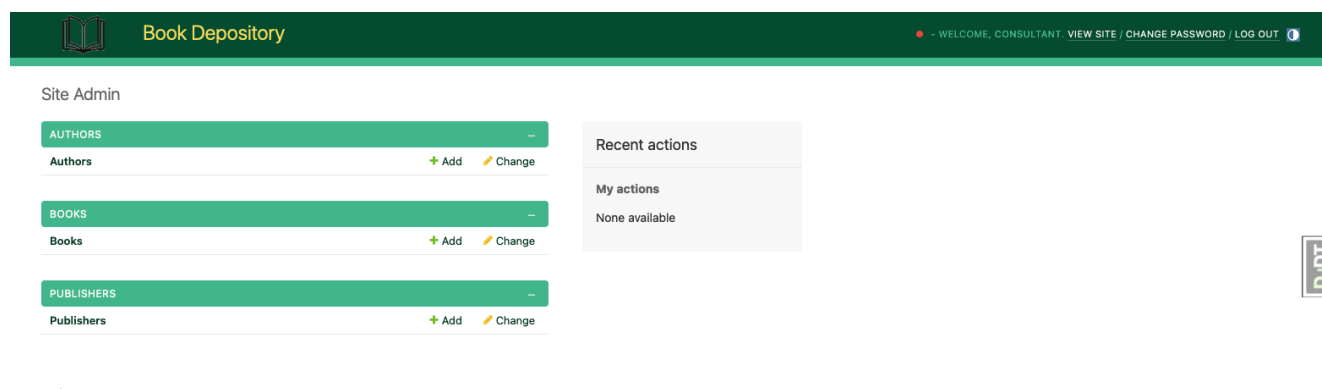


Рис. 3.4. Інтерфейс для взаємодії консультанта(можливо лише керувати книгами, авторами та видавництвами)

Розділ 4. Наповнення бази даних

1. Опис джерела історичних даних

Джерелом історичних даних був відкритий сервіс Google Books та його Google Books API(<https://www.googleapis.com/books/v1/>). Використовувались GET запити, щоб отримати інформацію по книгам, їх авторам та видавництвам.

З API було отримано такі данні як: назва книги, автор, видавництво, жанр, рік публікації, кількість сторінок, обкладинка.

2. Опис процесів генерації тестових даних

Тестові дані генерувались для інших моделей, яких не було можливість отримати з відкритого API. Випадковим чином було згенеровано національність автора, його дата народження, локація видавництва її номер телефону та багато інших полів.

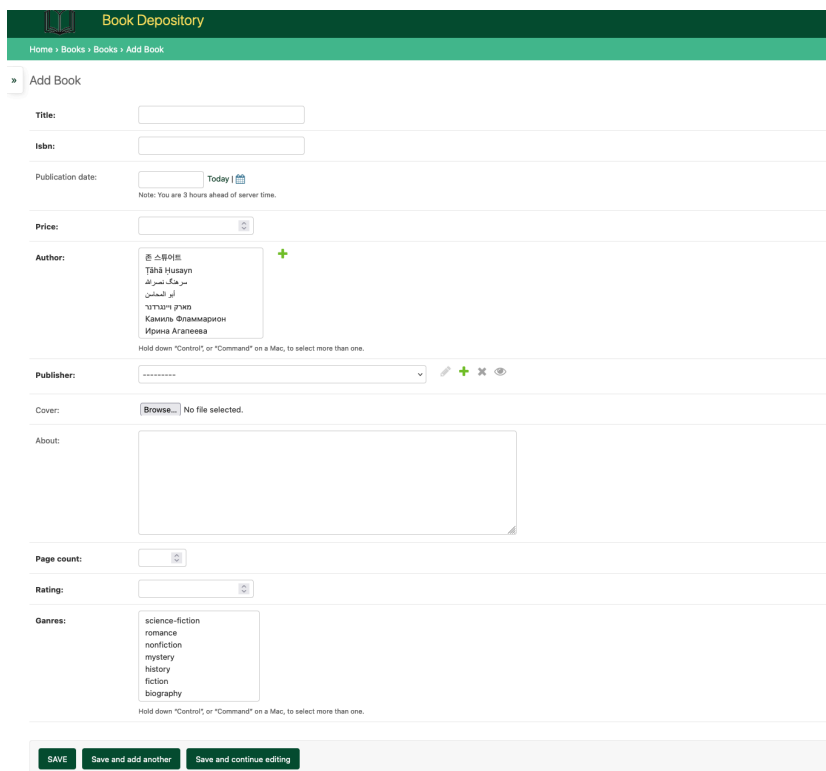
3. Трансформація та первинне завантаження даних

Після отримання та генерації тестових даних настав процес трансформації. Оскільки дані з API приходили в форматі, який не підтримувався системою, була необхідність в обробці кожного поля. Також ключові сутності при завантаженні залежать від додаткових, наприклад: Щоб додати книгу, потрібно додати відповідного автора та видавництво. Тому при додаванні відбувалась перевірка на існування додаткових даних та подальше їх створення, перед створення ключових сутностей.

Розділ 5. Функціональні можливості для користувача

1. Опис інтерфейсу у відповідності до бізнес-процесів

Додавання нової книги та додаткової інформації:

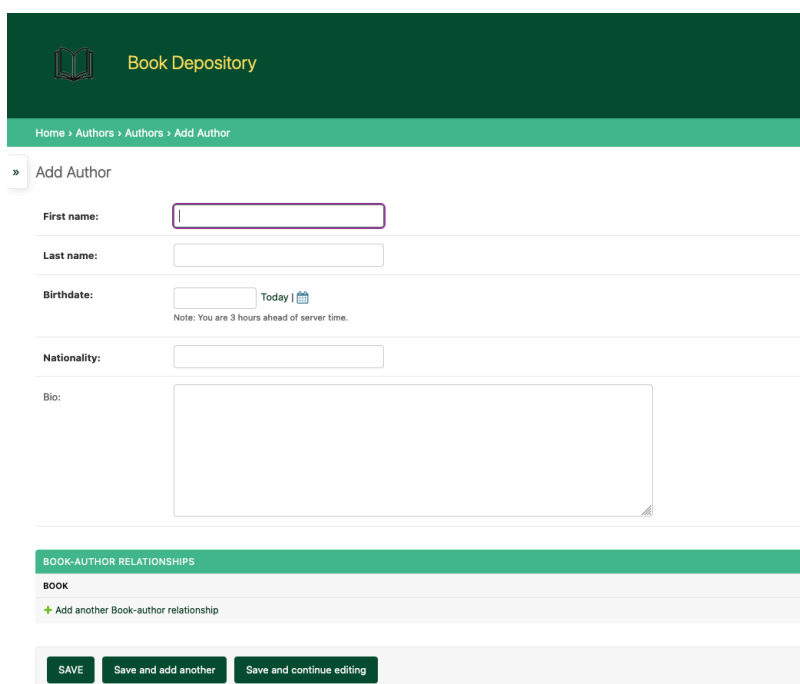


The screenshot shows the 'Add Book' form in the Book Depository system. The form is titled 'Add Book' and includes the following fields and options:

- Title:** A text input field.
- Isbn:** A text input field.
- Publication date:** A date picker set to 'Today'. A note below states: 'Note: You are 3 hours ahead of server time.'
- Price:** A text input field with a currency symbol (€) on the right.
- Author:** A dropdown menu showing a list of authors: '존 스튜어트' (John Stewart), 'Tahā Husayn', 'سرهنگ حسام', 'أبي الحسن', 'Հովսեփ Մկրտչ', 'Камилъ Фламмарикон', and 'Ирина Агапеева'. A green plus icon is next to the dropdown.
- Publisher:** A dropdown menu with a search icon, a green plus icon, a close icon, and an eye icon.
- Cover:** A 'Browse...' button and the text 'No file selected.'
- About:** A large text area for the book's description.
- Page count:** A text input field with a currency symbol (€) on the right.
- Rating:** A text input field with a currency symbol (€) on the right.
- Genres:** A dropdown menu showing a list of genres: 'science-fiction', 'romance', 'nonfiction', 'mystery', 'history', 'fiction', and 'biography'. A note below states: 'Hold down "Control", or "Command" on a Mac, to select more than one.'

At the bottom of the form, there are three buttons: 'SAVE', 'Save and add another', and 'Save and continue editing'.

Рис. 5.1. Додавання нової книги




The screenshot shows the 'Add Author' form in the Book Depository system. The form is titled 'Add Author' and includes the following fields and options:

- First name:** A text input field.
- Last name:** A text input field.
- Birthdate:** A date picker set to 'Today'. A note below states: 'Note: You are 3 hours ahead of server time.'
- Nationality:** A text input field.
- Bio:** A large text area for the author's biography.

Below the form, there is a section titled 'BOOK-AUTHOR RELATIONSHIPS' with a sub-section 'BOOK'. It includes a green plus icon and the text 'Add another Book-author relationship'.

At the bottom of the form, there are three buttons: 'SAVE', 'Save and add another', and 'Save and continue editing'.

Рис. 5.2. Додавання нового автору



Book Depository

Home > Publishers > Publishers > Add Publisher

» Add Publisher

Name:

Address:


Phone:

BOOKS			
TITLE	ISBN	PUBLICATION DATE	PRICE
+ Add another Book			

SAVE Save and add another Save and continue editing

Рис. 5.3. Додавання нового видавництва

Додавання нового складу:



Book Depository

Home > Depositories > Depositories > Add Depository

» Add Depository

Name:

Location:

BOOK DEPOSITORYS	
BOOK	QUANTITY
+ Add another Book depository	

EVENT DEPOSITORYS	
EVENT	
+ Add another Event depository	

SAVE Save and add another Save and continue editing

Рис. 5.4. Створення нового складу

Реєстрація нової події:

Add event

Event type:

▼

Description:

Responsible:

▼

+

Related depository:

▼

+

EVENT BOOKS

BOOK

QUANTITY

+ Add another Event book

SAVE

Рис. 5.5. Реєстрація нової події

Система пошуку книг характеристиками:

Select Book to change

Q

Search

Action:

Go

0 of 100 selected

<input type="checkbox"/> TITLE	ISBN	PUBLICATION DATE	PRICE	IMAGE TAG
<input type="checkbox"/> 香榧貴公子	9789862403570	June 12, 2012	49.0	<div>No Image Available</div>
<input type="checkbox"/> 笑問儒	9789862403211	June 8, 2012	11.0	<div>No Image Available</div>
<input type="checkbox"/> 少壯錄保羅	9789862402900	June 12, 2012	28.0	<div>No Image Available</div>
<input type="checkbox"/> 威猛先生	9789862402894	June 8, 2012	14.0	<div>No Image Available</div>
<input type="checkbox"/> 蘇拉曼的靈魂	9781527311916	July 27, 2022	48.0	<div>No Image Available</div>

IMPORT

ADD BOOK

FILTER

By title

All

▼

By isbn

All

▼

By publication date

Any date

▼

By price

All

▼

Рис. 5.6. Фільтрація та пошук книг

Фільтрація та пошук подій:



Рис. 5.6. Фільтрація та пошук подій

2. Засоби аналітичного представлення даних

Для аналітичного представлення був обраний зручний звіт за допомогою логів та графіки:

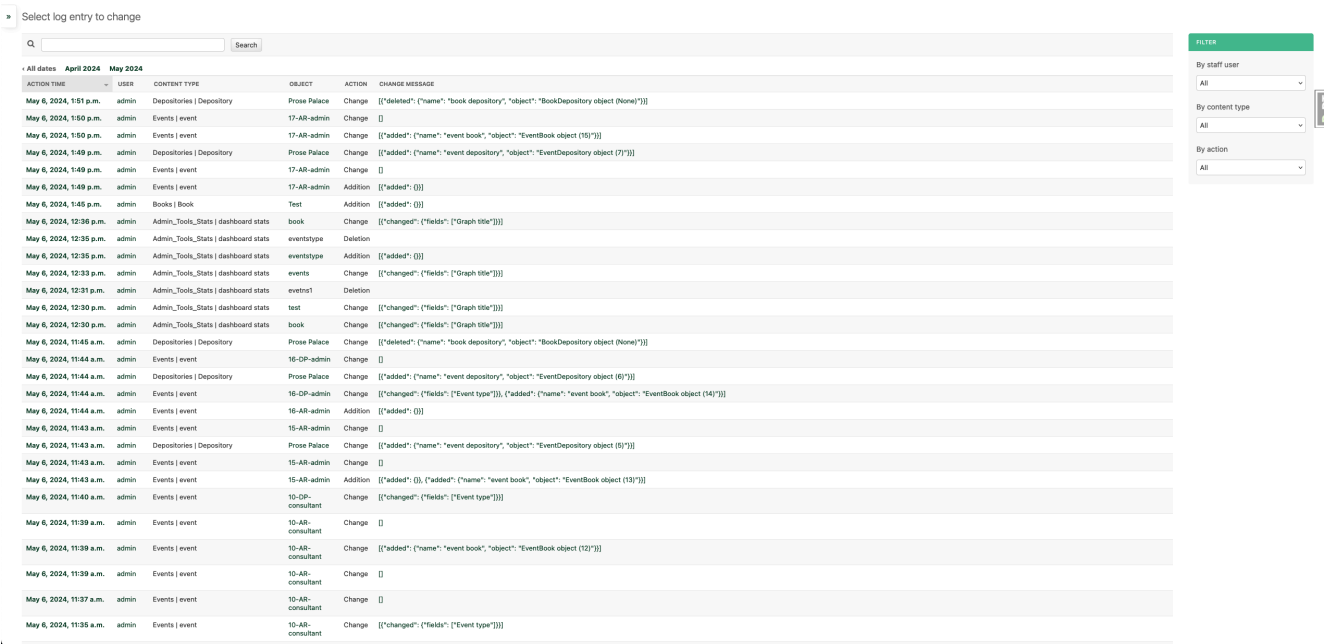


Рис. 5.7. Логи для аналітики

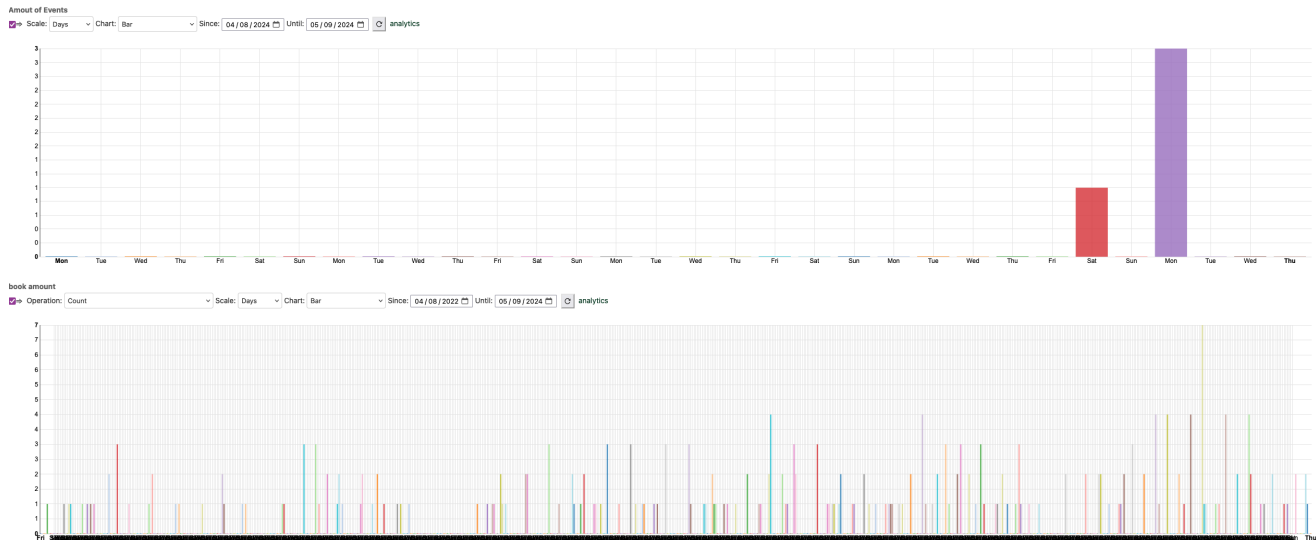


Рис. 5.8. Графіки візуалізації

3. Засоби експорту даних

Із системи доступний експорт даних про авто у форматі csv, xls, xlsx, tsv, ops, json, yaml, html.

Action: Export selected events Format

✓ ...

Go

3 of 4 selected

EVENT TYPE	RESPONSIBLE	TIMESTAMP	ID	RELATED DEPOSITORY
<input checked="" type="checkbox"/> Arrival	admin	May 6, 2024, 1:49 p.m.	17	-
<input checked="" type="checkbox"/> Departure	admin	May 6, 2024, 11:44 a.m.	16	-
<input checked="" type="checkbox"/> Arrival	admin	May 6, 2024, 11:43 a.m.	15	-
<input type="checkbox"/> Departure	consultant	May 4, 2024, 12:35 p.m.	10	Book Haven

4. Засоби програмного інтерфейсу

Із засобів програмного інтерфейсу реалізовано точку доступу для пошуку ключових сутностей за характеристиками.

Висновки

Під час розробки було встановлено наступні недоліки у реалізації та використаних технологіях:

- Не інтуїтивний інтерфейс – інтерфейс недостатньо наповнений інформацією, щоб користувач одразу розумів всю інформацію на екрані.
- Залежність від функціоналу адміністративної панелі Django.
- Кількість типів подій не покриває всі можливі кейси в функціоналі додатку.
- Існують прогалини на стороні інтерфейсу, які можуть привести до помилок, які не обробляються на стороні серверу.

На мою думку, рішення будувати серверну архітектуру, за допомогою Python та Django Rest Framework було правильним через перевагу даних технологій в архітектурі мінімально життєздатного продукту, що позитивно вплинуло та швидкість розробки додатку.

Очевидним помилковим рішенням стосувалось реалізації клієнтської частини за допомогою Django admin panel. Під час розробки неодноразово була проблема з кастомізацією HTML сторінки. Та в кінцевому продукті були знайдені недоліки, які через обмеженість вибраної технології не вдалось виправити.

Правильним рішенням було б використання сучасного фреймворку для створення інтерфейсу користувача, наприклад React або Angular. Дане рішення розв'язало б руку в створенні фронтенду додатку. Допомогло б повноцінно створити особисті компоненти та перекрити помилки в дизайні.

Рішення використовувати PostgreSQL в ролі СУБД та Django ORM для роботи з базою даних, на мою думку вважається вірним. Під час розробки майже не було знайдено проблем, а ті які були, то швидко вирішувались через велику базу користувачів та спільнот даних технологій.

Список літератури

1. Django Documentation [Електронний ресурс] / Django Software Foundation. – 2023. – Режим доступу до ресурсу: <https://docs.djangoproject.com/en/5.0/>.
2. Django ORM Documentation [Електронний ресурс] // Django Software Foundation. – 2023. – Режим доступу до ресурсу: <https://docs.djangoproject.com/en/5.0/topics/db/queries/>
3. PostgreSQL: Documentation [Електронний ресурс] // PostgreSQL. – 2024. – Режим доступу до ресурсу: <https://www.postgresql.org/docs/current/>.

Додатки

Скрипт створення бази даних

```
--

-- PostgreSQL database dump

--

-- Dumped from database version 16.2
-- Dumped by pg_dump version 16.2

-- Started on 2024-05-09 21:31:45 EEST

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;
```

```
--

-- TOC entry 266 (class 1255 OID 65795)

-- Name: insertbook(bigint, character varying, character varying, date, double precision, character
varying, text, bigint, smallint, double precision); Type: PROCEDURE; Schema: public; Owner: postgres
--

CREATE PROCEDURE public.insertbook(IN in_id bigint, IN in_title character varying, IN in_isbn character
varying, IN in_publication_date date, IN in_price double precision, IN in_cover character varying, IN
in_about text, IN in_publisher_id bigint, IN in_page_count smallint, IN in_rating double precision)

    LANGUAGE plpgsql

    AS $$

BEGIN

    INSERT INTO public.books_book (id, title, isbn, publication_date, price, cover, about,
publisher_id, page_count, rating)

        VALUES (in_id, in_title, in_isbn, in_publication_date, in_price, in_cover, in_about,
in_publisher_id, in_page_count, in_rating);

END;

$$;


ALTER PROCEDURE public.insertbook(IN in_id bigint, IN in_title character varying, IN in_isbn character
varying, IN in_publication_date date, IN in_price double precision, IN in_cover character varying, IN
in_about text, IN in_publisher_id bigint, IN in_page_count smallint, IN in_rating double precision)
OWNER TO postgres;


--

-- TOC entry 267 (class 1255 OID 65796)

-- Name: insertperson(bigint, character varying, character varying, date, character varying, text);
Type: PROCEDURE; Schema: public; Owner: postgres
--

CREATE PROCEDURE public.insertperson(IN in_id bigint, IN in_first_name character varying, IN
in_last_name character varying, IN in_birthdate date, IN in_nationality character varying, IN in_bio
text)

    LANGUAGE plpgsql

    AS $$

BEGIN

    INSERT INTO public.persons (id, first_name, last_name, birthdate, nationality, bio)

        VALUES (in_id, in_first_name, in_last_name, in_birthdate, in_nationality, in_bio);

END;

$$;
```

```
ALTER PROCEDURE public.insertperson(IN in_id bigint, IN in_first_name character varying, IN
in_last_name character varying, IN in_birthdate date, IN in_nationality character varying, IN in_bio
text) OWNER TO postgres;
```

```
--
```

```
-- TOC entry 268 (class 1255 OID 65797)
```

```
-- Name: insertpublisher(bigint, character varying, character varying, character varying); Type:
PROCEDURE; Schema: public; Owner: postgres
```

```
--
```

```
CREATE PROCEDURE public.insertpublisher(IN in_id bigint, IN in_name character varying, IN in_address
character varying, IN in_phone character varying)
```

```
    LANGUAGE plpgsql
```

```
    AS $$
```

```
BEGIN
```

```
    INSERT INTO public.publishers_publisher (id, name, address, phone)
```

```
    VALUES (in_id, in_name, in_address, in_phone);
```

```
END;
```

```
$$;
```

```
ALTER PROCEDURE public.insertpublisher(IN in_id bigint, IN in_name character varying, IN in_address
character varying, IN in_phone character varying) OWNER TO postgres;
```

```
SET default_tablespace = '';
```

```
SET default_table_access_method = heap;
```

```
--
```

```
-- TOC entry 243 (class 1259 OID 24779)
```

```
-- Name: admin_interface_theme; Type: TABLE; Schema: public; Owner: postgres
```

```
--
```

```
CREATE TABLE public.admin_interface_theme (
```

```
    id integer NOT NULL,
```

```

name character varying(50) NOT NULL,

active boolean NOT NULL,

title character varying(50) NOT NULL,

title_visible boolean NOT NULL,

logo character varying(100) NOT NULL,

logo_visible boolean NOT NULL,

css_header_background_color character varying(10) NOT NULL,

title_color character varying(10) NOT NULL,

css_header_text_color character varying(10) NOT NULL,

css_header_link_color character varying(10) NOT NULL,

css_header_link_hover_color character varying(10) NOT NULL,

css_module_background_color character varying(10) NOT NULL,

css_module_text_color character varying(10) NOT NULL,

css_module_link_color character varying(10) NOT NULL,

css_module_link_hover_color character varying(10) NOT NULL,

css_module_rounded_corners boolean NOT NULL,

css_generic_link_color character varying(10) NOT NULL,

css_generic_link_hover_color character varying(10) NOT NULL,

css_save_button_background_color character varying(10) NOT NULL,

css_save_button_background_hover_color character varying(10) NOT NULL,

css_save_button_text_color character varying(10) NOT NULL,

css_delete_button_background_color character varying(10) NOT NULL,

css_delete_button_background_hover_color character varying(10) NOT NULL,

css_delete_button_text_color character varying(10) NOT NULL,

list_filter_dropdown boolean NOT NULL,

related_modal_active boolean NOT NULL,

related_modal_background_color character varying(10) NOT NULL,

related_modal_rounded_corners boolean NOT NULL,

logo_color character varying(10) NOT NULL,

recent_actions_visible boolean NOT NULL,

favicon character varying(100) NOT NULL,

related_modal_background_opacity character varying(5) NOT NULL,

env_name character varying(50) NOT NULL,

env_visible_in_header boolean NOT NULL,

```

```

env_color character varying(10) NOT NULL,

env_visible_in_favicon boolean NOT NULL,

related_modal_close_button_visible boolean NOT NULL,

language_chooser_active boolean NOT NULL,

language_chooser_display character varying(10) NOT NULL,

list_filter_sticky boolean NOT NULL,

form_pagination_sticky boolean NOT NULL,

form_submit_sticky boolean NOT NULL,

css_module_background_selected_color character varying(10) NOT NULL,

css_module_link_selected_color character varying(10) NOT NULL,

logo_max_height smallint NOT NULL,

logo_max_width smallint NOT NULL,

foldable_apps boolean NOT NULL,

language_chooser_control character varying(20) NOT NULL,

list_filter_highlight boolean NOT NULL,

list_filter_removal_links boolean NOT NULL,

show_fieldsets_as_tabs boolean NOT NULL,

show_inlines_as_tabs boolean NOT NULL,

css_generic_link_active_color character varying(10) NOT NULL,

collapsible_stacked_inlines boolean NOT NULL,

collapsible_stacked_inlines_collapsed boolean NOT NULL,

collapsible_tabular_inlines boolean NOT NULL,

collapsible_tabular_inlines_collapsed boolean NOT NULL,

CONSTRAINT admin_interface_theme_logo_max_height_check CHECK ((logo_max_height >= 0)),

CONSTRAINT admin_interface_theme_logo_max_width_check CHECK ((logo_max_width >= 0))

);


ALTER TABLE public.admin_interface_theme OWNER TO postgres;


--

-- TOC entry 242 (class 1259 OID 24778)

-- Name: admin_interface_theme_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres

--

```

```

ALTER TABLE public.admin_interface_theme ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

    SEQUENCE NAME public.admin_interface_theme_id_seq

    START WITH 1

    INCREMENT BY 1

    NO MINVALUE

    NO MAXVALUE

    CACHE 1

);

```

```

--

-- TOC entry 255 (class 1259 OID 33103)

-- Name: admin_tools_stats_cachedvalue; Type: TABLE; Schema: public; Owner: postgres

--

```

```

CREATE TABLE public.admin_tools_stats_cachedvalue (

    id bigint NOT NULL,

    date timestamp with time zone NOT NULL,

    time_scale character varying(90) NOT NULL,

    operation character varying(90),

    operation_field_name character varying(90),

    filtered_value character varying(512),

    value double precision,

    dynamic_choices jsonb NOT NULL,

    multiple_series_choice_id bigint,

    stats_id bigint NOT NULL,

    is_final boolean NOT NULL,

    "order" integer

);

```

```

ALTER TABLE public.admin_tools_stats_cachedvalue OWNER TO postgres;

```



```

--
-- TOC entry 254 (class 1259 OID 33102)
-- Name: admin_tools_stats_cachedvalue_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.admin_tools_stats_cachedvalue ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.admin_tools_stats_cachedvalue_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);

--
-- TOC entry 253 (class 1259 OID 33067)
-- Name: admin_tools_stats_criteriatostatm2m; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.admin_tools_stats_criteriatostatm2m (
    id bigint NOT NULL,
    "order" integer,
    prefix character varying(255) NOT NULL,
    use_as character varying(90) NOT NULL,
    criteria_id bigint NOT NULL,
    stats_id bigint NOT NULL,
    default_option character varying(255) NOT NULL,
    choices_based_on_time_range boolean NOT NULL,
    count_limit integer,
    recalculate boolean NOT NULL,
    CONSTRAINT admin_tools_stats_criteriatostatm2m_count_limit_check CHECK ((count_limit >= 0)),
    CONSTRAINT admin_tools_stats_criteriatostatm2m_order_check CHECK (("order" >= 0))
);

```

```
ALTER TABLE public.admin_tools_stats_criteriatostatm2m OWNER TO postgres;
```

```
--
```

```
-- TOC entry 252 (class 1259 OID 33066)
```

```
-- Name: admin_tools_stats_criteriatostatm2m_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
```

```
--
```

```
ALTER TABLE public.admin_tools_stats_criteriatostatm2m ALTER COLUMN id ADD GENERATED BY DEFAULT AS  
IDENTITY (
```

```
    SEQUENCE NAME public.admin_tools_stats_criteriatostatm2m_id_seq
```

```
    START WITH 1
```

```
    INCREMENT BY 1
```

```
    NO MINVALUE
```

```
    NO MAXVALUE
```

```
    CACHE 1
```

```
);
```

```
--
```

```
-- TOC entry 222 (class 1259 OID 24599)
```

```
-- Name: auth_group; Type: TABLE; Schema: public; Owner: postgres
```

```
--
```

```
CREATE TABLE public.auth_group (
```

```
    id integer NOT NULL,
```

```
    name character varying(150) NOT NULL
```

```
);
```

```
ALTER TABLE public.auth_group OWNER TO postgres;
```

```
--
```

```
-- TOC entry 221 (class 1259 OID 24598)
```

```
-- Name: auth_group_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
```

```
--
```

```
ALTER TABLE public.auth_group ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
```

```
    SEQUENCE NAME public.auth_group_id_seq
```

```
    START WITH 1
```

```
    INCREMENT BY 1
```

```
    NO MINVALUE
```

```
    NO MAXVALUE
```

```
    CACHE 1
```

```
);
```

```
--
```

```
-- TOC entry 224 (class 1259 OID 24607)
```

```
-- Name: auth_group_permissions; Type: TABLE; Schema: public; Owner: postgres
```

```
--
```

```
CREATE TABLE public.auth_group_permissions (
```

```
    id bigint NOT NULL,
```

```
    group_id integer NOT NULL,
```

```
    permission_id integer NOT NULL
```

```
);
```

```
ALTER TABLE public.auth_group_permissions OWNER TO postgres;
```

```
--
```

```
-- TOC entry 223 (class 1259 OID 24606)
```

```
-- Name: auth_group_permissions_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
```

```
--
```

```
ALTER TABLE public.auth_group_permissions ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
```

```
    SEQUENCE NAME public.auth_group_permissions_id_seq
```

```

        START WITH 1

        INCREMENT BY 1

        NO MINVALUE

        NO MAXVALUE

        CACHE 1

);

--

-- TOC entry 220 (class 1259 OID 24593)
-- Name: auth_permission; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.auth_permission (

    id integer NOT NULL,

    name character varying(255) NOT NULL,

    content_type_id integer NOT NULL,

    codename character varying(100) NOT NULL

);

ALTER TABLE public.auth_permission OWNER TO postgres;

--

-- TOC entry 219 (class 1259 OID 24592)
-- Name: auth_permission_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.auth_permission ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

    SEQUENCE NAME public.auth_permission_id_seq

    START WITH 1

    INCREMENT BY 1

    NO MINVALUE

    NO MAXVALUE

```

```

    CACHE 1

);

--

-- TOC entry 226 (class 1259 OID 24613)
-- Name: auth_user; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.auth_user (

    id integer NOT NULL,

    password character varying(128) NOT NULL,

    last_login timestamp with time zone,

    is_superuser boolean NOT NULL,

    username character varying(150) NOT NULL,

    first_name character varying(150) NOT NULL,

    last_name character varying(150) NOT NULL,

    email character varying(254) NOT NULL,

    is_staff boolean NOT NULL,

    is_active boolean NOT NULL,

    date_joined timestamp with time zone NOT NULL

);

ALTER TABLE public.auth_user OWNER TO postgres;

--

-- TOC entry 228 (class 1259 OID 24621)
-- Name: auth_user_groups; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.auth_user_groups (

    id bigint NOT NULL,

    user_id integer NOT NULL,

```

```

        group_id integer NOT NULL
    );

ALTER TABLE public.auth_user_groups OWNER TO postgres;

--
-- TOC entry 227 (class 1259 OID 24620)
-- Name: auth_user_groups_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.auth_user_groups ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

    SEQUENCE NAME public.auth_user_groups_id_seq

    START WITH 1

    INCREMENT BY 1

    NO MINVALUE

    NO MAXVALUE

    CACHE 1

);

--
-- TOC entry 225 (class 1259 OID 24612)
-- Name: auth_user_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.auth_user ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

    SEQUENCE NAME public.auth_user_id_seq

    START WITH 1

    INCREMENT BY 1

    NO MINVALUE

    NO MAXVALUE

    CACHE 1

);

```

```

--
-- TOC entry 230 (class 1259 OID 24627)
-- Name: auth_user_user_permissions; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.auth_user_user_permissions (
    id bigint NOT NULL,
    user_id integer NOT NULL,
    permission_id integer NOT NULL
);

ALTER TABLE public.auth_user_user_permissions OWNER TO postgres;

--
-- TOC entry 229 (class 1259 OID 24626)
-- Name: auth_user_user_permissions_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.auth_user_user_permissions ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.auth_user_user_permissions_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);

--
-- TOC entry 234 (class 1259 OID 24714)
-- Name: authors_author; Type: TABLE; Schema: public; Owner: postgres

```

--

```
CREATE TABLE public.authors_author (
    id bigint NOT NULL,
    first_name character varying(30) NOT NULL,
    last_name character varying(30) NOT NULL,
    birthdate date NOT NULL,
    nationality character varying(75) NOT NULL,
    bio text
);
```

```
ALTER TABLE public.authors_author OWNER TO postgres;
```

--

```
-- TOC entry 233 (class 1259 OID 24713)
-- Name: authors_author_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--
```

```
ALTER TABLE public.authors_author ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.authors_author_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
```

--

```
-- TOC entry 238 (class 1259 OID 24730)
-- Name: books_book; Type: TABLE; Schema: public; Owner: postgres
--
```



```

CREATE TABLE public.books_book (
    id bigint NOT NULL,
    title character varying(55) NOT NULL,
    isbn character varying(13) NOT NULL,
    publication_date date,
    price double precision NOT NULL,
    cover character varying(100),
    about text,
    publisher_id bigint NOT NULL,
    page_count smallint NOT NULL,
    rating double precision NOT NULL,
    CONSTRAINT books_book_page_count_check CHECK ((page_count >= 0)),
    CONSTRAINT isbn_length CHECK (((length((isbn)::text) = 13) AND ((isbn)::text ~ '^([0-9]+$)::text'))),
    CONSTRAINT non_negative_price CHECK ((price >= (0)::double precision)),
    CONSTRAINT non_negative_rating CHECK ((rating >= (0)::double precision))
);

```

```

ALTER TABLE public.books_book OWNER TO postgres;

```

```

--
-- TOC entry 240 (class 1259 OID 24738)
-- Name: books_book_author; Type: TABLE; Schema: public; Owner: postgres
--

```

```

CREATE TABLE public.books_book_author (
    id bigint NOT NULL,
    book_id bigint NOT NULL,
    author_id bigint NOT NULL
);

```

```

ALTER TABLE public.books_book_author OWNER TO postgres;

```

```
--
-- TOC entry 239 (class 1259 OID 24737)
-- Name: books_book_author_id_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.books_book_author ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.books_book_author_id_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
```

```
--
-- TOC entry 259 (class 1259 OID 41215)
-- Name: books_book_ganres; Type: TABLE; Schema: public; Owner: postgres
--
```

```
CREATE TABLE public.books_book_ganres (
    id bigint NOT NULL,
    book_id bigint NOT NULL,
    ganre_id bigint NOT NULL
);
```

```
ALTER TABLE public.books_book_ganres OWNER TO postgres;
```

```
--
-- TOC entry 258 (class 1259 OID 41214)
-- Name: books_book_ganres_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--
```

```

ALTER TABLE public.books_book_ganres ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

    SEQUENCE NAME public.books_book_ganres_id_seq

    START WITH 1

    INCREMENT BY 1

    NO MINVALUE

    NO MAXVALUE

    CACHE 1

);

```

```

--

-- TOC entry 237 (class 1259 OID 24729)

-- Name: books_book_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres

--

```

```

ALTER TABLE public.books_book ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

    SEQUENCE NAME public.books_book_id_seq

    START WITH 1

    INCREMENT BY 1

    NO MINVALUE

    NO MAXVALUE

    CACHE 1

);

```

```

--

-- TOC entry 257 (class 1259 OID 41209)

-- Name: books_ganre; Type: TABLE; Schema: public; Owner: postgres

--

```

```

CREATE TABLE public.books_ganre (

    id bigint NOT NULL,

    name character varying(25) NOT NULL

);

```

```
ALTER TABLE public.books_ganre OWNER TO postgres;
```

```
--
```

```
-- TOC entry 256 (class 1259 OID 41208)
```

```
-- Name: books_ganre_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
```

```
--
```

```
ALTER TABLE public.books_ganre ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.books_ganre_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
```

```
--
```

```
-- TOC entry 251 (class 1259 OID 33028)
```

```
-- Name: dash_stats_criteria; Type: TABLE; Schema: public; Owner: postgres
```

```
--
```

```
CREATE TABLE public.dash_stats_criteria (
    id bigint NOT NULL,
    criteria_name character varying(90) NOT NULL,
    criteria_fix_mapping jsonb,
    dynamic_criteria_field_name character varying(90),
    criteria_dynamic_mapping jsonb,
    created_date timestamp with time zone NOT NULL,
    updated_date timestamp with time zone NOT NULL
);
```

```
ALTER TABLE public.dash_stats_criteria OWNER TO postgres;
```

```
--
```

```
-- TOC entry 250 (class 1259 OID 33027)
```

```
-- Name: dash_stats_criteria_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
```

```
--
```

```
ALTER TABLE public.dash_stats_criteria ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.dash_stats_criteria_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
```

```
--
```

```
-- TOC entry 249 (class 1259 OID 33018)
```

```
-- Name: dashboard_stats; Type: TABLE; Schema: public; Owner: postgres
```

```
--
```

```
CREATE TABLE public.dashboard_stats (
    id bigint NOT NULL,
    graph_key character varying(90) NOT NULL,
    graph_title character varying(90) NOT NULL,
    model_app_name character varying(90) NOT NULL,
    model_name character varying(90) NOT NULL,
    date_field_name character varying(90) NOT NULL,
    operation_field_name character varying(90),
    type_operation_field_name character varying(90),
    is_visible boolean NOT NULL,
    created_date timestamp with time zone NOT NULL,
```

```

updated_date timestamp with time zone NOT NULL,

user_field_name character varying(90),

default_chart_type character varying(90) NOT NULL,

default_time_period integer NOT NULL,

default_time_scale character varying(90) NOT NULL,

y_axis_format character varying(90),

"distinct" boolean NOT NULL,

default_multiseries_criteria_id bigint,

show_to_users boolean NOT NULL,

allowed_chart_types character varying(1000),

allowed_time_scales character varying(1000) NOT NULL,

allowed_type_operation_field_name character varying(1000),

cache_values boolean NOT NULL,

CONSTRAINT dashboard_stats_default_time_period_check CHECK ((default_time_period >= 0))

);

```

```

ALTER TABLE public.dashboard_stats OWNER TO postgres;

```

```

--
-- TOC entry 248 (class 1259 OID 33017)
-- Name: dashboard_stats_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

```

```

ALTER TABLE public.dashboard_stats ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

    SEQUENCE NAME public.dashboard_stats_id_seq

    START WITH 1

    INCREMENT BY 1

    NO MINVALUE

    NO MAXVALUE

    CACHE 1

);

```

```

--
-- TOC entry 247 (class 1259 OID 24833)
-- Name: depositories_bookdepository; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.depositories_bookdepository (

    id bigint NOT NULL,

    quantity integer NOT NULL,

    book_id bigint NOT NULL,

    depository_id bigint NOT NULL

);


ALTER TABLE public.depositories_bookdepository OWNER TO postgres;


--
-- TOC entry 246 (class 1259 OID 24832)
-- Name: depositories_bookdepository_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.depositories_bookdepository ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

    SEQUENCE NAME public.depositories_bookdepository_id_seq

    START WITH 1

    INCREMENT BY 1

    NO MINVALUE

    NO MAXVALUE

    CACHE 1

);


--
-- TOC entry 245 (class 1259 OID 24825)
-- Name: depositories_depository; Type: TABLE; Schema: public; Owner: postgres
--

```

```
CREATE TABLE public.depositories_depository (
    id bigint NOT NULL,
    name character varying(55) NOT NULL,
    location character varying(125) NOT NULL
);
```

```
ALTER TABLE public.depositories_depository OWNER TO postgres;
```

```
--
```

```
-- TOC entry 244 (class 1259 OID 24824)
```

```
-- Name: depositories_depository_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
```

```
--
```

```
ALTER TABLE public.depositories_depository ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.depositories_depository_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
```

```
--
```

```
-- TOC entry 232 (class 1259 OID 24685)
```

```
-- Name: django_admin_log; Type: TABLE; Schema: public; Owner: postgres
```

```
--
```

```
CREATE TABLE public.django_admin_log (
    id integer NOT NULL,
    action_time timestamp with time zone NOT NULL,
    object_id text,
```



```

    object_repr character varying(200) NOT NULL,

    action_flag smallint NOT NULL,

    change_message text NOT NULL,

    content_type_id integer,

    user_id integer NOT NULL,

    CONSTRAINT django_admin_log_action_flag_check CHECK ((action_flag >= 0))

);


ALTER TABLE public.django_admin_log OWNER TO postgres;


--
-- TOC entry 231 (class 1259 OID 24684)
-- Name: django_admin_log_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.django_admin_log ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

    SEQUENCE NAME public.django_admin_log_id_seq

    START WITH 1

    INCREMENT BY 1

    NO MINVALUE

    NO MAXVALUE

    CACHE 1

);


--
-- TOC entry 218 (class 1259 OID 24585)
-- Name: django_content_type; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.django_content_type (

    id integer NOT NULL,

    app_label character varying(100) NOT NULL,

```

```

        model character varying(100) NOT NULL
    );

ALTER TABLE public.django_content_type OWNER TO postgres;

--
-- TOC entry 217 (class 1259 OID 24584)
-- Name: django_content_type_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.django_content_type ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.django_content_type_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);

--
-- TOC entry 216 (class 1259 OID 24577)
-- Name: django_migrations; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.django_migrations (
    id bigint NOT NULL,
    app character varying(255) NOT NULL,
    name character varying(255) NOT NULL,
    applied timestamp with time zone NOT NULL
);

```

```
ALTER TABLE public.django_migrations OWNER TO postgres;
```

```
--
```

```
-- TOC entry 215 (class 1259 OID 24576)
```

```
-- Name: django_migrations_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
```

```
--
```

```
ALTER TABLE public.django_migrations ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.django_migrations_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
```

```
--
```

```
-- TOC entry 241 (class 1259 OID 24763)
```

```
-- Name: django_session; Type: TABLE; Schema: public; Owner: postgres
```

```
--
```

```
CREATE TABLE public.django_session (
    session_key character varying(40) NOT NULL,
    session_data text NOT NULL,
    expire_date timestamp with time zone NOT NULL
);
```

```
ALTER TABLE public.django_session OWNER TO postgres;
```

```
--
```

```
-- TOC entry 261 (class 1259 OID 57595)
```

```
-- Name: events_event; Type: TABLE; Schema: public; Owner: postgres
```

--

```
CREATE TABLE public.events_event (
    id bigint NOT NULL,
    event_type character varying(2) NOT NULL,
    "timestamp" timestamp with time zone NOT NULL,
    description text,
    responsible_id integer NOT NULL,
    related_depository_id bigint
);
```

```
ALTER TABLE public.events_event OWNER TO postgres;
```

--

```
-- TOC entry 260 (class 1259 OID 57594)
```

```
-- Name: events_event_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
```

--

```
ALTER TABLE public.events_event ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.events_event_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
```

--

```
-- TOC entry 265 (class 1259 OID 57649)
```

```
-- Name: events_eventbook; Type: TABLE; Schema: public; Owner: postgres
```

--

```
CREATE TABLE public.events_eventbook (
    id bigint NOT NULL,
    quantity integer NOT NULL,
    book_id bigint NOT NULL,
    event_id bigint NOT NULL,
    CONSTRAINT positive_quantity CHECK ((quantity >= 0))
);
```

```
ALTER TABLE public.events_eventbook OWNER TO postgres;
```

```
--
-- TOC entry 264 (class 1259 OID 57648)
-- Name: events_eventbook_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--
```

```
ALTER TABLE public.events_eventbook ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.events_eventbook_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
```

```
--
-- TOC entry 263 (class 1259 OID 57609)
-- Name: events_eventdepository; Type: TABLE; Schema: public; Owner: postgres
--
```

```
CREATE TABLE public.events_eventdepository (
    id bigint NOT NULL,
    depository_id bigint NOT NULL,
```

```

        event_id bigint NOT NULL
    );

ALTER TABLE public.events_eventdepository OWNER TO postgres;

--
-- TOC entry 262 (class 1259 OID 57608)
-- Name: events_eventdepository_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.events_eventdepository ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.events_eventdepository_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);

--
-- TOC entry 236 (class 1259 OID 24722)
-- Name: publishers_publisher; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.publishers_publisher (
    id bigint NOT NULL,
    name character varying(75) NOT NULL,
    address character varying(255) NOT NULL,
    phone character varying(15) NOT NULL
);

```

```

ALTER TABLE public.publishers_publisher OWNER TO postgres;

--

-- TOC entry 235 (class 1259 OID 24721)

-- Name: publishers_publisher_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres

--

ALTER TABLE public.publishers_publisher ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

    SEQUENCE NAME public.publishers_publisher_id_seq

    START WITH 1

    INCREMENT BY 1

    NO MINVALUE

    NO MAXVALUE

    CACHE 1

);

```

Скрипт завантаження історичних даних

```

import requests
import random
import pprint
import json

from faker import Faker # Import Faker library for generating fake data
from datetime import datetime, timedelta

fake = Faker()

UNKNOWN_AUTHOR_ID = 132
UNKNOWN_PUBLISHER_ID = 74

GANRES_IDS = {
    "fiction": 1,
    "nonfiction": 2,
    "science-fiction": 3,
    "mystery": 4,
    "romance": 5,
    "history": 6,
    "biography": 7
}

def transform_date(date_str):
    try:
        # Parse the date string into a datetime object
        date_obj = datetime.fromisoformat(date_str)

        # Format the datetime object into "YYYY-MM-DD" format
        formatted_date = date_obj.strftime("%Y-%m-%d")
        return formatted_date
    except:
        return ""

def generate_random_birthdate():
    start_date = datetime(1900, 1, 1)
    end_date = datetime.now() - timedelta(days=365*18) # Assuming authors are at least 18
years old
    random_date = start_date + timedelta(days=random.randint(0, (end_date - start_date).days))

```

```

    return random_date.strftime("%Y-%m-%d")

def generate_random_nationality():
    nationalities = ["American", "British", "Canadian", "French", "German", "Japanese",
"Russian", "Chinese", "Indian"]
    return random.choice(nationalities)

def generate_random_address():
    return fake.address()

def generate_random_phone_number():
    return fake.phone_number()

def fetch_random_books(num_books, ganre):
    base_url = "https://www.googleapis.com/books/v1/volumes"
    books = []
    iterations = 0

    while len(books) < num_books:
        # Generate a random index to get a random page of books
        random_index = str(random.randint(1, 100))
        params = {
            "q": f"subject:{ganre}", # You can adjust the query to fit your needs
            "startIndex": random_index,
            "maxResults": min(40, num_books - len(books)),
            # "orderBy": "relevance"
            "orderBy": "newest"
        }
        response = requests.get(base_url, params=params)
        iterations += 1

        if response.status_code == 200:
            data = response.json()
            books.extend(data.get("items", []))
        else:
            print("Failed to fetch books:", response.status_code)
            break

    all_books_data = []

    for i, book in enumerate(books[:num_books], start=1):
        title = book['volumeInfo']['title']
        authors_info = book['volumeInfo'].get('authors', ['Unknown'])
        authors = ", ".join(authors_info)
        publisher = book['volumeInfo'].get('publisher', 'Unknown Publisher')
        published_date = transform_date(book['volumeInfo'].get('publishedDate', '1000-01-01'))
        isbn_13 = next((identifier['identifier'] for identifier in
book['volumeInfo'].get('industryIdentifiers', []) if identifier['type'] == 'ISBN_13'), 'Unknown')
        page_count = book['volumeInfo'].get('pageCount', 0)
        average_rating = book['volumeInfo'].get('averageRating', 0)

        # Generate random birthdate and nationality for each author
        authors_birthdates = [generate_random_birthdate() for _ in authors_info]
        authors_nationalities = [generate_random_nationality() for _ in authors_info]

        publisher_address = generate_random_address()
        publisher_phone = generate_random_phone_number()

        book_data = {
            "title": title,
            "authors": [{"name": author, "birthdate": birthdate, "nationality": nationality}
for author, birthdate, nationality in zip(authors_info, authors_birthdates, authors_nationalities)],
            "publisher": {"name": publisher, "address": publisher_address, "phone":
publisher_phone},
            "publication_date": published_date,
            "isbn": isbn_13,
            "page_count": page_count,
            "rating": average_rating,
            "price": random.randint(4, 50)
        }

        all_books_data.append(book_data)

    return all_books_data

def add_publishers(publisher):
    response = requests.post('http://127.0.0.1:8000/api/v1/publishers/', publisher)
    print(f"{response.status_code} | {response.text}")

def add_authors(authors):

```



```

for author in authors:
    name = author['name'].split()
    if len(name) < 2:
        name.append("")
    body = {
        "first_name": name[0],
        "last_name": name[1],
        "birthdate": author["birthdate"],
        "nationality": author["nationality"]
    }
    response = requests.post('http://127.0.0.1:8000/api/v1/authors/', body)
    print(f"{response.status_code} | {response.text}")

def add_books(book_payload, ganre):
    authors_json = dict()
    publishers_json = dict()

    response = requests.get('http://127.0.0.1:8000/api/v1/authors/')
    response_json = response.json()

    for author in response_json:
        authors_json[f"{author['first_name']} {author['last_name']}"] = author["id"]

    response = requests.get('http://127.0.0.1:8000/api/v1/publishers/')
    response_json = response.json()

    for publisher in response_json:
        publishers_json[publisher["name"]] = publisher["id"]

    book_payload['publisher'] = publishers_json.get(book_payload['publisher']['name'],
UNKNOWN_PUBLISHER_ID)

    name = book_payload["authors"][0]["name"].split()
    if len(name) < 2:
        name.append("")
    authors_list = []
    authors_list.append(authors_json.get(f"{name[0]} {name[1]}", UNKNOWN_AUTHOR_ID))
    book_payload['author'] = authors_list
    book_payload['ganres'] = [GANRES_IDS[ganre]]

    response = requests.post('http://127.0.0.1:8000/api/v1/books/', book_payload)
    print(f"{response.status_code} | {response.text}")

def set_ganre():
    response = requests.get('http://127.0.0.1:8000/api/v1/books/')
    books_json = response.json()

    books_ids= [book["id"] for book in books_json]

    for id in books_ids:
        response = requests.patch(f"http://127.0.0.1:8000/api/v1/books/{id}/", {"ganres": [1]})
        print(f"{response.status_code} | {response.text}")

ganres = ["mystery", "romance", "history", "biography"]

for ganre in ganres:
    books = fetch_random_books(2000, ganre)
    for book in books:
        add_publishers(book['publisher'])
        add_authors(book['authors'])

    add_books(book, ganre)

```

Інший програмний код

Програмний код всієї системи знаходиться в [репозиторії](#).

Інший код створених моделей

```

from django.db import models
from django.core.exceptions import ValidationError

```

```

class Author(models.Model):
    first_name = models.CharField(max_length=30, db_index=True)
    last_name = models.CharField(max_length=30, db_index=True)
    birthdate = models.DateField()
    nationality = models.CharField(max_length=75, db_index=True)
    bio = models.TextField(blank=True, null=True)

    def save(self, *args, **kwargs):
        if Author.objects.filter(first_name=self.first_name,
last_name=self.last_name).exists():
            raise ValidationError("Author with this first name and last name already exists.",
code='author_exists')
        super().save(*args, **kwargs)

    def clean(self):
        if Author.objects.filter(first_name=self.first_name,
last_name=self.last_name).exists():
            raise ValidationError("Author with this first name and last name already exists.",
code='author_exists')

    class Meta:
        verbose_name = "Author"
        verbose_name_plural = "Authors"
        ordering = ["-first_name"]

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

from django.db import models
from authors.models import Author
from publishers.models import Publisher

class Genre(models.Model):
    name = models.CharField(max_length=25)

    class Meta:
        verbose_name = "Genre"
        verbose_name_plural = "Genres"
        ordering = ["-name"]

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

class Book(models.Model):
    title = models.CharField(max_length=55, db_index=True)
    isbn = models.CharField(max_length=13, unique=True, db_index=True)
    publication_date = models.DateField(db_index=True, null=True, blank=True)
    price = models.FloatField(db_index=True)
    author = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher, on_delete=models.DO_NOTHING)
    cover = models.ImageField(default='default_book_image.jpg', null=True, blank=True)
    about = models.TextField(blank=True, null=True)
    page_count = models.PositiveIntegerField()
    rating = models.FloatField()
    genres = models.ManyToManyField(Genre)

    class Meta:
        verbose_name = "Book"
        verbose_name_plural = "Books"
        ordering = ["-title"]

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

from django.db import models

from books.models import Book

class Depository(models.Model):
    name = models.CharField(max_length=55, db_index=True)
    location = models.CharField(max_length=125)

    class Meta:
        verbose_name = "Depository"
        verbose_name_plural = "Depositories"
        ordering = ["-name"]

```

```

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

class BookDepository(models.Model):
    book = models.ForeignKey(Book, on_delete=models.CASCADE)
    depository = models.ForeignKey(Depository, on_delete=models.CASCADE)
    quantity = models.IntegerField()

    class Meta:
        unique_together = ('book', 'depository')

from django.contrib.auth.models import User
from django.db import models
from depositories.models import Depository
from books.models import Book

class Event(models.Model):
    ARRIVAL = 'AR'
    DEPARTURE = 'DP'
    TRANSFER_FROM = 'TF'
    TRANSFER_TO = 'TO'

    EVENT_CHOICES = [
        (DEPARTURE, 'Departure'),
        (ARRIVAL, 'Arrival'),
        (TRANSFER_TO, 'Transfer from this to Another'),
        (TRANSFER_FROM, 'Transfer to this From Anothet'),
    ]

    event_type = models.CharField(max_length=2, choices=EVENT_CHOICES, db_index=True)
    timestamp = models.DateTimeField(auto_now_add=True, db_index=True)
    description = models.TextField(blank=True, null=True)
    responsible = models.ForeignKey(User, on_delete=models.CASCADE, db_index=True)
    related_depository = models.ForeignKey(Depository, on_delete=models.CASCADE, null=True,
blank=True)

    def __str__(self):
        return f"{self.id}-{self.event_type}-{self.responsible}"

class EventDepository(models.Model):
    event = models.ForeignKey(Event, on_delete=models.CASCADE)
    depository = models.ForeignKey(Depository, on_delete=models.CASCADE)

    class Meta:
        unique_together = ('event', 'depository')

class EventBook(models.Model):
    event = models.ForeignKey(Event, on_delete=models.CASCADE)
    book = models.ForeignKey(Book, on_delete=models.CASCADE)
    quantity = models.IntegerField()

from django.db import models

class Publisher(models.Model):
    name = models.CharField(max_length=75, db_index=True, unique=True)
    address = models.CharField(max_length=255)
    phone = models.CharField(max_length=15)

    class Meta:
        verbose_name = "Publisher"
        verbose_name_plural = "Publishers"
        ordering = ["-name"]

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

```