

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

Марченко Павло Анатолійович

**ІНФОРМАЦІЙНА СИСТЕМА ПІДТРИМКИ ПЕРЕКЛАДАЦЬКОЇ
ДІЯЛЬНОСТІ НА ОСНОВІ ТЕХНОЛОГІЙ ПРОЄКТУВАННЯ .NET CORE**

кваліфікаційна робота

здобувача вищої освіти другого (магістерського) рівня

освітньої програми «Мультимедійні системи»

за спеціальністю 121 Інженерія програмного забезпечення

Особистий підпис _____ Павло МАРЧЕНКО

Науковий керівник _____ Світлана ДОНЧЕНКО,
асистент кафедри інформаційних
технологій та систем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Полтава – 2025

АНОТАЦІЯ

Марченко П. А.

Тема: Інформаційна система підтримки перекладацької діяльності на основі технологій проєктування .NET Core.

Спеціальність: 121 «Інженерія програмного забезпечення».

Установа: ЛНУ імені Тараса Шевченка, 2025р.

Магістерська робота містить: 58 с., 26 рис., 3 табл., 31 джерел.

Об'єкт дослідження – технології проєктування інформаційних систем на платформі .NET Core.

Предмет дослідження – проєктування інформаційних систем на платформі .NET Core.

Мета дослідження – розробка інформаційної системи підтримки перекладацької діяльності на основі технологій проєктування .NET Core.

Методи дослідження – розробка інформаційної системи, вирішення проблем та демонстрація можливостей платформи.

Результати роботи. У роботі досліджено можливості, застосунки платформи .NET Core для проєктування та розробки інформаційних систем.

Проаналізовано проблему відсутності каталогу перекладачів та засобів отримання та пошуку інформації по літературним творам, їх авторів та перекладачів.

Розроблено інформаційну систему на базі платформи .NET Core та мові програмування C# для оцінки та демонстрації можливості технологій. Реалізовано збір даних з відкритих джерел методом парсингу. Розроблено демонстрацію, пошук, фільтрацію, управління зібраними даними.

Реалізовано алгоритм забезпечення захисту системи від неавторизованого доступу шляхом використання хеш-значень паролю.

Ключові слова: ЦИТУВАННЯ, ПЕРЕКЛАДАЧ, ПЕРЕКЛАДАЦЬКА ДІЯЛЬНІСТЬ, ВЕБ-ЗАСТОСУНОК, ІНФОРМАЦІЙНА СИСТЕМА, .NET CORE, C#, ASP.NET CORE.

ANNOTATION

Marchenko Pavlo

Theme: Information system to support translation activities based on .NET Core design technologies.

Speciality: 121 "Software Engineering".

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2025 year.

Master's work of: 58 p., 26 im, 3 tables, 31 sources.

A research object of: Information systems design technologies on the .NET Core platform.

The article of research: information systems design on the .NET Core platform.

An aim of research is: development of an information system to support translation activities based on .NET Core design technologies.

Research methods – development of an information system, problem solving, and demonstration of platform capabilities.

Job performanes: Possibilities, applications of the .NET Core platform for design and development of information systems are investigated.

The problem of lack of a catalog of translators and means of obtaining and searching for information on literary works, their authors and translators is analyzed.

An information system based on the .NET Core platform and the C# programming language has been developed to assess and demonstrate the capabilities of the technology. Implemented data collection from open sources by parsing. Demonstration, search, filtering, management of the collected data are developed.

An algorithm for protecting the system from unauthorized access by using password hash values has been implemented.

Keywords: QUOTATIONS, TRANSLATOR, TRANSLATION ACTIVITY, WEB APPLICATION, INFORMATION SYSTEM, .NET CORE, C#, ASP.NET CORE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОГО СЕРЕДОВИЩА ТА	
МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ.....	9
1.1. Аналіз предметного середовища	9
1.1.1. Опис предметного середовища	9
1.1.2. Опис процесу діяльності	10
1.1.3. Опис функціональної моделі	11
1.1.4. Огляд наявних аналогів	12
1.2. Дослідження математичного забезпечення	14
1.2.1. Змістовна постановка задачі	14
1.2.2. Математична постановка задачі	15
1.2.3. Обґрунтування методу розв'язання.....	15
1.2.4. Опис методу розв'язання.....	17
Висновки до розділу	21
РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА АНАЛІЗ ІНФОРМАЦІЙНОЇ СИСТЕМИ	
НА ПЛАТФОРМІ .NET CORE	23
2.1. Вимоги до технічного забезпечення	23
2.2. Архітектура програмного забезпечення	23
2.2.1. Вхідні дані.....	23
2.2.2. Вихідні дані.....	25
2.2.3. Побудова діаграм варіантів використання (Usecase Diagrams).....	25
2.2.4. Фізичне представлення моделі програмної розробки	26
2.2.5. Діаграма пакетів	28
2.2.6. Діаграма послідовності.....	32
2.2.7. Діаграма компонентів	34
Висновки до розділу	35
РОЗДІЛ 3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДТРИМКИ	
ПЕРЕКЛАДАЦЬКОЇ ДІЯЛЬНОСТІ НА ПЛАТФОРМІ .NET CORE	36

3.1. Обґрунтування вибору засобів розробки.....	36
3.2. Опис структури бази даних	38
3.3. Розробка інтерфейсу	42
Висновки до розділу	52
ВИСНОВКИ	53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	55
ДОДАТОК А	59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	Application Programming Interface
REST	Representational State Transfer
СКБД	Система керування базою даних
IDE	Integrated Development Environment
MVC	Model-View-Controller
MVVM	Model-View-ViewModel
ПЗ	Програмне забезпечення
UML	Unified Model Language

ВСТУП

З кожним роком вимоги до функціональності інформаційних систем зростають. Розробка застосунків потребує високо кваліфікованих спеціалістів з глибоким розумінням та великим досвідом. Час розробки таких систем збільшується разом з складністю. Тому наявність технологій, які б дозволяли спростити, пришвидшити розробку є вкрай необхідним.

У сучасному світі інформація є одним із найцінніших ресурсів, без якого неможливо уявити функціонування жодної сучасної галузі. Неправильна, некоректна або неповна інформація може стати причиною багатьох проблем різного ступеня складності. Саме тому роль перекладача набуває особливого значення, адже його робота стає мостом, що з'єднує людей у дедалі глобалізованішому світі. У зв'язку з цим розробка ефективних інструментів, які допомагали б перекладачам у їхній діяльності, є надзвичайно важливим завданням.

Метою цього дослідження є вивчення можливостей платформи .NET Core для створення та розробки інформаційних систем. Як приклад інформаційної системи пропонується застосунок у формі каталогу перекладачів.

Розробка єдиного каталогу перекладачів значно полегшить їхню роботу. Така система дозволить збирати дані з різних джерел, усувати прогалини в інформації та надавати користувачам можливість отримувати відомості про твори, їхніх авторів і перекладачів.

Зібрана інформація сприятиме глибшому розумінню творчої діяльності перекладачів, а також створить основу для подальших досліджень. Систематизація даних дозволить аналізувати стилі перекладів, проводити порівняння між різними авторами, визначати їхні особливості та виділяти характеристики, притаманні певним категоріям перекладачів.

Розроблена інформаційна система дозволить проаналізувати та продемонструвати різні можливості платформи .NET Core. При розробці застосунку можна зіткнутись та вирішити задачі різної складності з різними цілями та методом вирішення. Даний підхід дозволить продемонструвати

технології платформи .NET Core, які можуть бути використанні для схожих задач.

Об’єкт дослідження – технології проєктування інформаційних систем на платформі .NET Core.

Предмет дослідження – проєктування інформаційних систем на платформі .NET Core.

Мета роботи – розробка інформаційної системи підтримки перекладацької діяльності на основі технологій проєктування .NET Core.

Методи дослідження – розробка інформаційної системи, вирішення проблем та демонстрація можливостей платформи.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати можливості платформи .NET Core на базі існуючої документації;
- дослідити існуючі фреймворки та бібліотеки, їх функціональність та вирішення яких проблем вони надають;
- розробити інформаційну систему на базі зібраної інформації;
- здійснити парсинг наявних електронних каталогів для збору даних, необхідних для функціонування інформаційної системи.
- забезпечити збереження даних, отриманих у результаті парсингу каталогів.
- розробити підсистему для фільтрації, редагування та доповнення даних.
- впровадити можливість групування перекладачів та/або авторів за визначеними критеріями.
- створити підсистему для виведення інформації.

Практична цінність роботи полягає в аналізі потенціалу платформи .NET Core і розробці інформаційної системи, що допомагає вирішувати завдання цитування та збору даних для аналізу стилістичних особливостей перекладачів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОГО СЕРЕДОВИЩА ТА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Аналіз предметного середовища

1.1.1. Опис предметного середовища

Перекладач — це фахівець, основним завданням якого є створення перекладу. Він виступає посередником між джерелом інформації та її користувачем. Потреба у такій проміжній ланці виникає, коли користувач не володіє мовою, якою створено джерело інформації.

Джерелом інформації можуть бути як письмові твори, так і усні виступи. Споживачі використовують цю інформацію для різноманітних цілей: від навчання та розваг до комерційної діяльності чи вирішення повсякденних питань.

Саме тому робота перекладача є надзвичайно важливою. Результати його діяльності можуть впливати на роботу багатьох інших людей. При перекладі літературних творів часто виникає необхідність зазначення цитат авторів.

Цитата – це дослівне відтворення фрагмента якогось тексту з обов’язковим посиланням на джерело [1].

При відсутності інформації про автора цитати чи перекладу є ризик порушити авторські права, так як привласнення авторства на чужий твір науки, літератури, мистецтва або на чуже відкриття, винахід чи раціоналізаторську пропозицію це є плагіатом [1], тому перекладачам слід обережно вказувати цитати. Відсутність інформації про “офіційний переклад” потребує від перекладача робити його власноруч, при цьому враховуючи ритміку для віршованих творів, термінологію для наукових або технічних текстів та інше. Така додаткова робота збільшує складність та час витрачений на переклад.

Роботу може полегшити електронний каталог перекладачів. Система яка містить інформацію про перекладачів та їх роботи, зберігає та дозволяє віднайти точну інформацію про примірник.

Як вже було наведено вище, при роботі перекладачі зустрічаються з творами, для яких не вказано автора та джерела цитати (найчастіше для художніх

текстів). Дана проблема заважає визначити контекст для перекладу, цитування або інших цілей.

При наявності методу розпізнавання стилю автора можна спробувати дізнатися кому належить даний твір. Так використовуючи систему з каталогом авторів, перекладачів та їх творів можна використати існуючі методи розпізнавання стилю автора, та знайти або встановити авторство тексту.

Для пошуку офіційних перекладів, дослідження стилю та впливу перекладача на інформацію перекладу, встановлення, знаходження переліку перекладених за автором творів та іншого необхідна наявність системи, яка представляє функції електронного каталогу.

Електронний каталог – це програмне забезпечення, яке містить впорядковану за певним критерієм інформацію та надає доступ до цієї інформації користувачам [2].

Електронний каталог може міститись на окремому електронному носії надаючи доступ тільки тим користувачам, які володіють цим носієм, та мають пристрій для зчитування інформації з електронного носія. Альтернативою може виступати зберігання каталогу на віддаленому сервері та надання доступу засобами мережі Інтернет.

Зберігання на сервері має значні переваги над зберіганням на окремому електронному носії.

При колективному використанні системи каталогу, необхідно забезпечити консистентність даних – цілісність даних, відсутність внутрішніх протиріч.

1.1.2. Опис процесу діяльності

Існуючі систему пошуку та електронні каталоги містять переважно дані про автора тексту і надають обмежену інформацію або взагалі не надають її про перекладача.

Метою автоматизації в описаній інформаційній системі є процес пошуку і збору даних текстів та надання сукупної інформації про перекладача.

При відсутності наявності реалізованої інформаційної системи пошук зводиться до наступного. Відомі дані потрібно ввести в пошукову систему. Серед

результатів пошукової системи обрани одне джерело. Перевірити джерело на наявність необхідної інформації. При відсутності інформації скористатись іншим джерелом. Якщо ж відсутні будь-які дані про текст, автора чи інше потрібно шукати існуючі каталоги та намагатись знайти інформацію в цьому каталозі.

Схему структурні діяльності до розробки системи наведена в частині графічного матеріалу.

Далі наведено опис процесу діяльності при наявності реалізованої інформаційної системи.

Знаходження авторів можна двома способами.

Перший спосіб – використання каталогу творів. При необхідності можна скористатися пошуком за назвою твору. Якщо назва не відома, твори можна відсортувати. Також при наявності інформації про твір, можна скористатись фільтром, що може значно звузити набір творів для пошуку. Серед творів, які задовольняють вказаним фільтрам обрати необхідний. На сторінці книги можна знайти детальну інформацію про твір, видавництво, авторів та перекладачів книги.

Другий спосіб – використання каталогу авторів. При необхідності можна скористатись пошуком за ім'ям автора. Також список авторів можна відсортувати. Серед знайдених авторів вибрати необхідного. На сторінці автора можна знайти інформацію про написані, або перекладені цим автором книги.

1.1.3. Опис функціональної моделі

Найзручніший спосіб використання системи може надати реалізація як веб застосунку.

При проєктуванні інформаційної системи, було виявлено необхідність таких акторів як користувач, адміністратор.

Система поділена на 2 частини.

Перша частина – сторінки доступні будь-якому користувачу. Ця частина надає доступ на отримання інформації.

Друга частина – кабінет адміністратора. Адміністратор має доступ на редагування інформації.

Для збору інформації використовуються джерела, які знаходяться у вільному доступі. Шляхом парсингу збирається інформація про книги, видавництва, авторів, перекладачів і тд.

Для зберігання інформації використовується база даних.

Схема структурна варіантів використання наведена в частині графічного матеріалу.

1.1.4. Огляд наявних аналогів

Всі знайдені системи виконуються функції електронного каталогу книг, мають обмежені та/чи не повні дані. Тому розглянемо джерела, які можна використати для збору інформації.

Інтернет-магазин «Yakaboo»

Розглянемо електронний ресурс інтернет магазин «Yakaboo» [3] та інформацію, яку можна отримати з даного джерела (рис. 1.1).

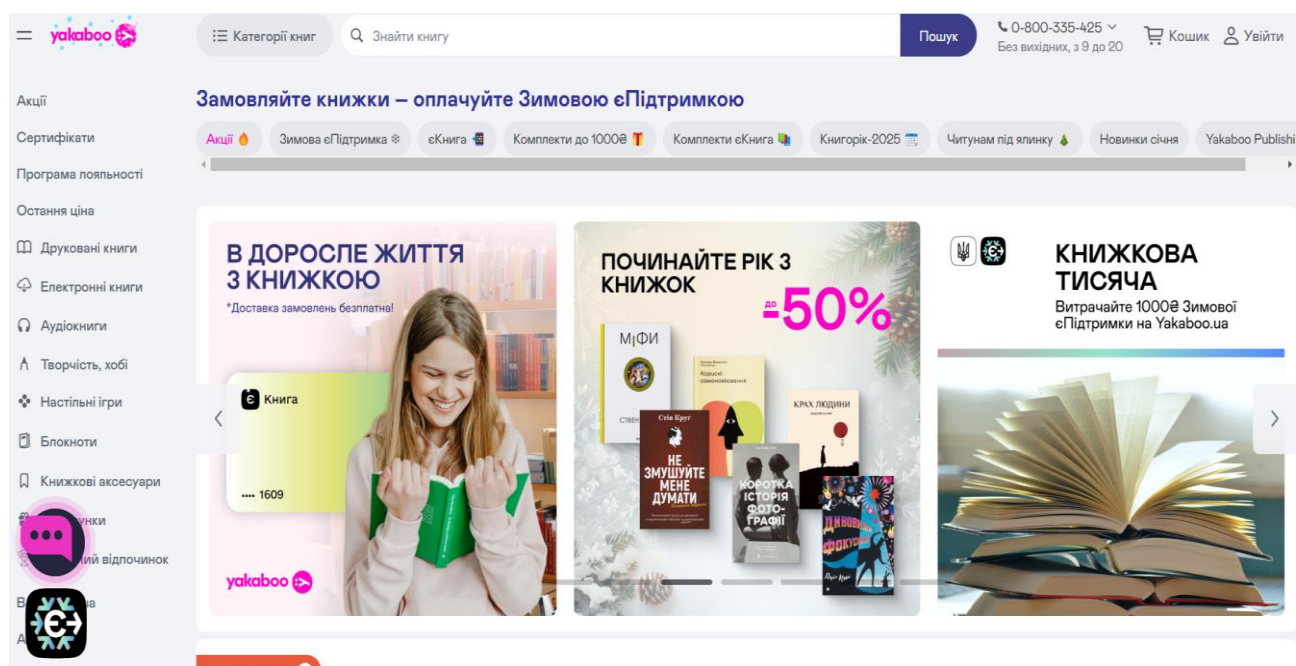


Рис. 1.1. Сторінка з книгами з сайту yakaboo.ua

На даному ресурсі в наявності велика кількість книжок. Значна частина на українській мові. Основна маса книг видання останніх років. Серед авторів

багато і українських, наявність яких не потрібна для інформаційної системи. Відсутня можливість сортувати книги за авторами чи перекладачами.

Інтернет-магазин «Book-ye»

Розглянемо електронний ресурс інтернет магазин «Book-ye» [4] та інформацію, яку можна отримати з даного джерела (рис. 1.2).

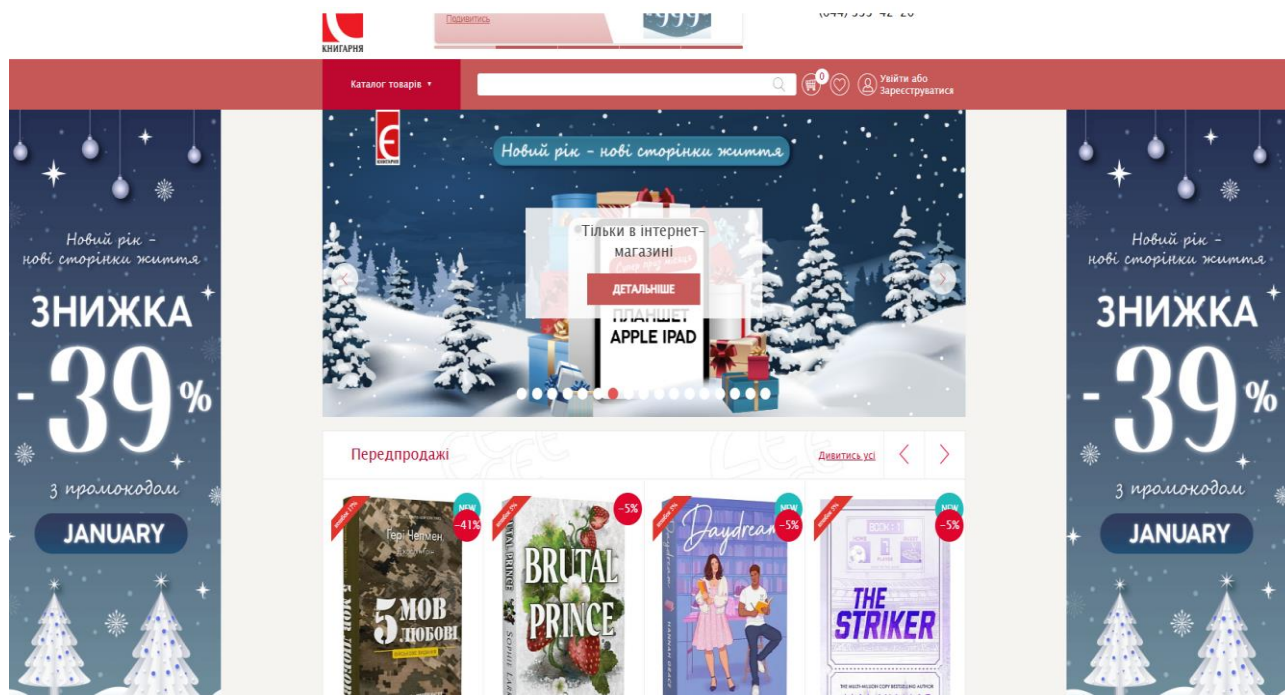


Рис 1.2. Сторінка з книгами з сайту book-ye.com.ua

На даному ресурсі кількість книг на українській мові значно менша ніж на попередньому. Для частини перекладачів вказано не повне ім'я, а лише прізвище та ініціали. Частина перекладених книг взагалі не містять інформації про перекладачів. Відсутня можливість сортувати книги за авторами чи перекладачами.

Національна бібліотека Вернадського

Ресурс є доволі не стабільний, час від часу недоступний, що не дозволяє його використовувати постійно. В системі наявний пошук по авторам, назві та контексту. Пошук по перекладачам відсутній. Можна використовувати пошук по контексту, але результат може містити не потрібні дані. Повний каталог ресурсів відсутній, можна використовувати тільки пошук.

Наявність основних характеристик кожної з систем можна переглянути в таблиці 1.1.

Порівняння існуючих джерел

	Інформація система, що розроблюється	Yakaboo	Book-ye	Національна бібліотека Вернадського
Наявність великої кількості книг	Так	Так	Так	Так
Можливість переглядати всі наявні книги	Так	Так	Так	Ні
Можливість сортувати книги за авторами/ перекладачами	Так	Ні	Ні	Ні

При розробці інформаційної системи враховуються недоліки аналогічних ресурсів. Даний підхід дозволяє створити інструмент, який повністю задовольнить потреби користувачів.

1.2. Дослідження математичного забезпечення

1.2.1. Змістовна постановка задачі

В інформаційній системі виділяється 2 типи користувачів: звичайний користувач та адміністратор. Адміністратор має можливість модифікувати контент інформаційної системи. Для захисту даних від модифікації звичайними користувачами система повинна обмежувати доступ. Шляхом проведення процедури встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора [6]. Тобто запросити у користувача логін та пароль та перевірити їх коректність.

Але зберігати паролі в відкритому виді небезпечно [7]. Якщо злоумисник отримає доступ до бази даних, чи просто якимсь чином побачить дані, які в ній зберігаються, то він отримає доступ до системи.

Адміністратор працює в системі з інформацією про книги, авторів, перекладачів. Так як в система може отримати на вхід не коректні чи не повні дані, то їх потрібно корегувати. Таким чином система має мати функції, які допоможуть адміністратору перевіряти та модифікувати дані.

1.2.2. Математична постановка задачі

Для вирішення проблеми зберігання паролів у відкритому виді використовується хешування. Хеш-функція дозволяє перетворювати паролі на рядок символів та не дозволяє відновити цим даним сам пароль [8]. Інша особливість хеш-функцій в тому, що при однакових вхідних даних функція видає однаковий результат. Дана особливість дозволяє зберігати тільки хеш значення паролів. А при перевірці введеного користувачем паролю, до цих даних застосовуються та сама хеш функція. Тобто звіряється схожість не самих паролів, а хеш значень цих даних.

Для полегшення роботи адміністратора в системі використовується метод визначення відстані Дамерау – Лаванштейна, для визначення схожості слів. Даний алгоритм визначає відстань між 2-ма словами, тобто наскільки ці слова відрізняються. В поняття відстані вкладається кількість операцій, які необхідно виконати для перетворення одного слова в інше.

1.2.3. Обґрунтування методу розв'язання

Для створення хеш-значення паролю потрібно використати саму хеш-функцію та функцію формування ключа.

Паролі потрібно зберігати у виді хеш-значення через те, що для такого значення неможливо встановити початкову інформацію. Хоча це не є правдою, на практиці є способи відновити дані, але ці операцію потребують дуже багато часу. Тому зберігання паролю у вигляді хеш-значення є доволі надійний спосіб забезпечити безпеку системи.

В якості хеш-функції було обрано SHA-256. В якості альтернатив розглянемо наступні функції: MD5, SHA-1, SHA-512. Порівняємо їх по критеріям швидкості, захищеності [9].

MD5 є застарілою функцією, яку не рекомендовано використовувати [10]. Так як функція виконується швидко та існує велика база відомих паролів та їх хеш-значень, обійти захист методом підбору дуже легко. Також проблемою є колізії, велика кількість паролів мають схожі хеш-значення.

Колізією хеш-функції називаються два різні вхідних блоки даних які мають однакове хеш-значення [11].

SHA-1 є швидшою функцією ніж MD5, але також має проблеми з колізіями [12]. Через це використовувати її для визначення хеш-значень паролів може привести до проблем.

SHA-256 та SHA-512 функції сімейства SHA-2. Вони мають значно кращу криптографічну стійкість ніж попередні алгоритми. Обидві функції є повільнішими ніж попередні. Відмінність SHA-256 від SHA-512 полягає в розмірі результуючого набору символів 64 проти 128 та швидкості, перший виконує операції швидше.

З усіх вище наведених функцій було обрано SHA-256. Так як ця функція генерує значення достатньої розмірності. Є досить швидкою, щоб не навантажувати систему та досить повільною, щоб ускладнити можливість підбору. Обрана функція є оптимальним балансом між швидкістю та криптостійкістю.

Для забезпечення надійності паролів також потрібно використовувати сіль. Детальніше наведено в розділі 3.4. Необхідність даної операції полягає в тому, що для людини досить складною задачею є запам'ятовування довгих паролів, які мають різні літери, цифри та спеціальні символи. Тому для забезпечення надійності паролів використовується сіль, яка дозволяє збільшити розмір паролю.

Функція формування ключа є необхідною, так як дозволяє збільшити криптографічну стійкість хеш-значення. Якщо зловмисник отримає значення хешу йому знадобиться значно більше часу на встановлення оригінального паролю.

В якості функції формування ключа було обрано PBKDF2 так як дана функція є частиною PKCS #5 v2.0 який є стандартом RFC 2898 [13].

Для визначення схожості слів було обрано метод визначення відстані Дамеру – Левенштейна. Так як на відміну від звичайного методу Левенштейна, обраний метод також враховує те, що літери в словах можуть бути поміняні

місцями. Тому такий метод може визначити схожість слів, навіть якщо при набірні тексті було введено літери не в правильному порядку.

1.2.4. Опис методу розв'язання

Алгоритм створення хешу полягає в наступному:

- Згенерувати сіль за допомогою псевдовипадкової функції;
- За допомогою функції формування ключа згенерувати ключ;
- Об'єднати отримані та вхідні дані в хеш-значення.

Детальніше ознайомимся з кожним пунктом.

Сіль – модифікатор, рядок даних який передається хеш-функції разом з паролем [14]. Сіль додається до паролю і вже на отриманому рядку застосовується хеш-функція. Якщо зломисник отримав базу з паролями то підбираючи різні функції хешування та можливі паролі можна відновити паролі. Сіль дозволяє збільшити час, який зломисник витратить на відновлення паролів.

Функція формування ключа – функція, що формує один або кілька секретних ключів на основі секретного значення [15].

В якості функції формування ключа використовується PBKDF2

PBKDF2 (Password-Based Key Derivation Function) – стандарт формування ключа на основі пароля [16]. Функція використовує псевдовипадкову функцію для отримання ключа.

Функцію задається наступною формулою

$$DK = PBKDF2(PRF, P, S, c, dkLen) \quad (1.1)$$

де PRF – псевдовипадкова функція, з виходом довжини $hLen$

P – майстер-пароль – пароль, для якого необхідно отримати хеш-значення;

S – сіль;

c – кількість ітерацій, має бути додатнім числом;

$dkLen$ – бажана довжина ключа (не більше $(2^{32} - 1) * hLen$)

Вихідний параметр: DK – згенерований ключ довжини $dkLen$.

Дана функція (1) використовується наступним чином. При генерації хешу на вхід функція отримує випадковим чином згенеровану сіль розміром 128 біт, пароль, кількість ітерацій в розмірі 10000, бажаний розмір ключа 256 бітів та

функцію хешування. В якості хеш функції використовується HMAC-SHA256. В результаті виконання функція повертає згенерований ключ розмірності 256 бітів.

Після отримання всіх значень формується остаточне хеш-значення за допомогою якого можна буде перевірити введений користувачем пароль на достовірність. Остаточне значення займає 61 байт або 488 бітів. В це значення входять:

- тип шифрування – 1 байт;
- тип хеш-функції – 4 байти;
- кількість ітерацій – 4 байти;
- розмір солі – 4 байти;
- значення солі – 16 байти;
- значення отриманого ключа – 32 байти.

Але для запису в базу даних значення кодується за допомогою base64.

Base64 – таблиця з 64 символів [17]. Маємо рядок бітів. При кодуванні дані беруться у 6 бітовому форматі, так як максимальне число 64 це 2 в 6 степені. Тобто з рядку бітів виділяються кожні 6 цифр та за допомогою таблиці визначається символ який відповідає цьому числу.

Для перевірки введеного користувачем паролю проводяться наступні операції.

За допомогою введеного логіна в базі знаходиться рядок зі даними адміністратора. Дістається хеш-значення паролю. Перевіряється чи кількість байтів співпадає.

Після цього подібно методу шифрування перевіряється значення. Береться тип шифрування, щоб дізнатись яким алгоритмом було проведено шифрування. Береться тип хеш-функції, кількість ітерацій та розмір солі. Перевіряється чи розмір солі співпадає з записаними в системі.

За допомогою значення розміру солі, з хеш-значенню паролю виділяється значення солі.

Всі байти, що залишились після вище названих операцій, припадають на ключ, який було отримано при виконанні псевдовипадкової функції. За

допомогою тієї ж самою функцією та з отриманими вище даними визначається ключ.

Якщо отриманий ключ співпадає з ключем записаним в базі то пароль введено вірний.

Розглянемо детально як працює функція формування ключа PBKFD2.

Хід обчислень складається з 3 етапів.

1 етап.

Маємо розмір бажаного ключа $dkLen$, та розмір хешу $hLen$, який отримуємо використавши псевдовипадкову функцію PRF .

Виділяємо l блоків.

$$l = \left\lceil \left(\frac{dkLen}{hlen} \right) \right\rceil$$

значення округляємо вгору. Визначаємо кількість байтів в останньому блоці

$$r = dkLen - (l - 1) * hLen.$$

2 етап

Маючи значення P – майстер пароль, S – сіль, c – кількість ітерацій введемо наступну функцію.

$$F(P, S, c, i) = U_1 \oplus U_2 \oplus \dots \oplus U_c \quad (1.2)$$

$$U_1 = PRF(P, S \parallel INT(i))$$

$$U_2 = PRF(P, U_1)$$

.....

$$U_c = PRF(P, U_{c-1})$$

Операція \oplus називається хог або виключна диз'юнкція [18]. Тобто функція (2) визначена, як операція хог над першими c ітераціями хеш-функції PRF . В хеш функцію передається значення парою P , значення солі S , і номер блоку в форматі integer, тобто 4-байтове ціле число.

Виключна диз'юнкція – логічна або бітова операція, що приймає значення істина тоді і лише тоді, коли значення «істина» має рівно один з її операндів.

Введена функція (2) виконується для кожного з блоків наступним чином

$$T_1 = F(P, S, c, 1)$$

$$T_2 = F(P, S, c, 2)$$

.....

$$T_l = F(P, S, c, l)$$

3 етап

Значення T для кожного блоку потрібно об'єднати для створення ключа. Ключ DK розмірності $dkLen$. При поділу блоків було визначено значення r – кількість байтів в останньому блоці. При об'єднанні всі значення T записуються по порядку, а від останнього T_l береться тільки перші r байтів.

Далі наведено опис роботи алгоритму визначення відстані Дамерау – Лаванштейна [19].

Даний алгоритм знаходить кількість операцій, які необхідно виконати для того, щоб перетворити одне слово в інше. В якості операцій можуть бути використані вставка, видалення, заміна та транспозиція.

Операція вставки представляє собою операція, в результаті якої слово буде мати на одну літеру більше. Так в обраному місці додається обрана літера слова, з яким порівнюється початкове слово.

Операція заміни навпаки зменшує кількість літер в слові. Якщо в слові, яке перевіряється немає на обраному місці літери, то вона може бути видалена з початкового слова.

Операція заміни має в результаті виконання операції вставки та заміни, але рахується як одна операція. Тобто літера слова, яке перевіряється, замінює собою літеру з початкового слова.

Операція транспозиції полягає в тому, що дві літери, які йдуть одна за одною, міняються між собою місцями. Тобто якщо в слові, яке перевіряється, знаходяться наприклад літери 'аб', а в початковому слові 'ба'. То дана перестановка буде розцінюватись як одна операція.

Таким чином формується матриця, на якій визначається ціна переходу для відповідних частин слова. Якщо довжина першого слова дорівнює n , а довжина

другого слова m , то для визначення відстані будується матриця розмірності $(n+1) \times (m+1)$.

При обході матриці, визначається ціна для кожного елемента матриці. Позначається ціна $D(i, j)$, де i номер стовпця, а j номер рядка матриці.

При обчисленні ціни використовуються наступні правила

$$D(i, j) = 0, i = 0, j = 0$$

$$D(i, j) = i, i > 0, j = 0$$

$$D(i, j) = j, i = 0, j > 0$$

Для обчислення D коли $i, j > 0$ нижче наведена формула. Але спочатку введемо позначення.

$S_1[i]$ - набір символів першого слова, $i = 1..n$

$S_2[j]$ - набір символів другого слова, $j = 1..m$

C_3 - ціна заміни

C_T - ціна транспозиції

$$D(i, j) = \min \begin{cases} D(i, j-1) + 1 \\ D(i-1, j) + 1 \\ D(i-1, j-1) + C_3 \\ D(i-2, j-2) + C_T \end{cases}$$

$$C_3 = \begin{cases} 1, & S_1[i] \neq S_2[j] \\ 0 & \end{cases}$$

$$C_T = \begin{cases} 1, & S_1[i] = S_2[j-1] \text{ і } S_1[i-1] = S_2[j] \\ \infty & \end{cases}$$

Таким чином при обчисленні значення елемента матриці використовуються попередні елементи матриці. Результуючим значенням буде $D(n+1, m+1)$, тобто значення матриці в правому нижньому куті.

Якщо значення відстані менше 3 можна вважати, що дані слова схожі.

Висновки до розділу

В даному розділі було описано предметну область, проблеми, які вирішує інформаційна система, що розробляється. Описано аналоги системи та порівняння цих аналогів з даною системою. Описано процес діяльності до автоматизації та після впровадження розробленої інформаційної системи.

Також в розділі описано математичне забезпечення, яке використовується в інформаційній системі. Було детально описано алгоритм хешування паролів, обґрунтовано використання хеш-функції та функції формування ключа. Було наведено формули по обчисленню.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА АНАЛІЗ ІНФОРМАЦІЙНОЇ СИСТЕМИ НА ПЛАТФОРМІ .NET CORE

2.1. Вимоги до технічного забезпечення

Для коректної роботи інформаційної системи технічне забезпечення повинно відповідати наступним вимогам:

- процесор з кількістю ядер 2 або більше, з тактовою частотою 1.8 ГГц або вище
- оперативна пам'ять з ємністю 4 ГБ або більше
- один з наступних браузерів
 - 1) Google Chrome версії 57 або вище
 - 2) Mozilla FireFox версії 52 або вище
 - 3) Safari версії 11 або вище
 - 4) Microsoft Edge версії 16 або вище

2.2. Архітектура програмного забезпечення

2.2.1. Вхідні дані

Інформаційна система припускає декілька можливих способів введення даних, а саме:

- автоматичний збір інформації з інших каталогів та/або джерел;
- введення даних адміністратором системи;
- завантаження даних від користувача.

Всі вхідні дані впливають на роботу інформаційної системи. Для загальної характеристики розглянемо кожен шлях надходження даних окремо.

Автоматичний збір інформації з інших каталогів та/або джерел – представляє собою дані зібрані з доступних у веб середовищі відкритих джерел, таких як інтернет-магазини, бібліотеки, каталоги.

Нижче наведено структура даних з поділом на більш загальні групи інформації:

- інформація про книгу:
 - 1) назва;
 - 2) код ISBN;

- 3) посилання на джерело з якого було отримані дані;
- 4) опис книги;
- 5) мова;
- інформація про автора:
 - 1) ім'я
 - 2) по-батькові (або друге ім'я);
 - 3) прізвище;
 - 4) повне ім'я;
 - 5) інформація про те, чи є дана персона перекладачем або автором оригінального твору;
- інформація про обкладинки книги або друкованого видання:
 - 1) посилання на картинку;
- інформація про видавництво:
 - 1) назва видавництва;
 - 2) дата видання.

Після завершення етапу збору або оновлення даних, інформація є доступною для адміністратора. Адміністратор може фільтрувати, редагувати та вносити зміни, додавати нові дані. Тобто вхідними даними від адміністратора є наведені вище групи вхідної інформації.

Функції фільтрування та редагування даних Адміністратором необхідні для забезпечення цілісності та несуперечливості даних, первинної обробки інформації. Так однією з важливих функцій Адміністратора системи є поєднання даних про авторів або перекладачів, яких можна вважати за одного автора. Наприклад, авторів, які мають псевдоніми, або групу перекладачів, які працюють разом.

Для роботи з системою користувач може обрати наступні дані для фільтрів, що забезпечують доступ до необхідної йому інформації:

- назва твору;
- категорія сортування;
- тип сортування;

- ім'я автора та інше.

2.2.2. Вихідні дані

Вихідні дані представляють собою інформацію про книгу або особу.

Інформація про книгу має наступні параметри:

- назва книги;
- код ISBN;
- список авторів;
- список перекладачів;
- дата публікації;
- назва видавництва
- зображення палітурки;
- опис книги.

Інформація про особу має наступні параметри:

- ім'я особи;
- інші імена особи, або осіб пов'язані з нею;
- книги, над якими працювала особа.

2.2.3. Побудова діаграм варіантів використання (Usecase Diagrams)

Користувач взаємодіє з інтерфейсом (User Interface), через нього він обирає сторінку та отримує дані. В свою чергу інтерфейс взаємодіє з сервером (Server). Інтерфейс передає команди від користувача до сервера. Після відповіді сервера, інтерфейс відображає отриману інформацію. Сервер взаємодіє з базою даних (Database). Так як всі дані зберігаються в базі даних. Сервер формує запит, відповідно до отриманих команд. Інформацію, отриману від бази даних, сервер оброблює та відправляє інтерфейсу.

Адміністратор взаємодіє з інтерфейсом. Для доступу до функцій адміністратора, йому потрібно пройти авторизацію. Інтерфейс потребує введення логіну та паролю. Введені дані інтерфейс передає до сервера. Сервер формує запит до бази даних, відповідно до введеного логіну. Отримане хеш-значення паролю звіряється з обчисленим значенням вхідного паролю. Результат

порівнянь сервер передає інтерфейсу. Відповідно чи успішно було пройдено перевірку, інтерфейс дає доступ до функцій адміністратора.

Для того, щоб адміністратор міг отримати інформацію про книги та перевірити її, фоновому сервісу (Background Service) необхідно завантажити дані. Фоновий сервіс посилає запит до Джерела інформації за необхідною інформацією. Після отримання інформації фоновий сервіс зберігає її до бази даних. Фоновий сервіс також перевіряє чи в базі містяться вже перевірені дані і при наявності оброблює їх та зберігає в базі даних.

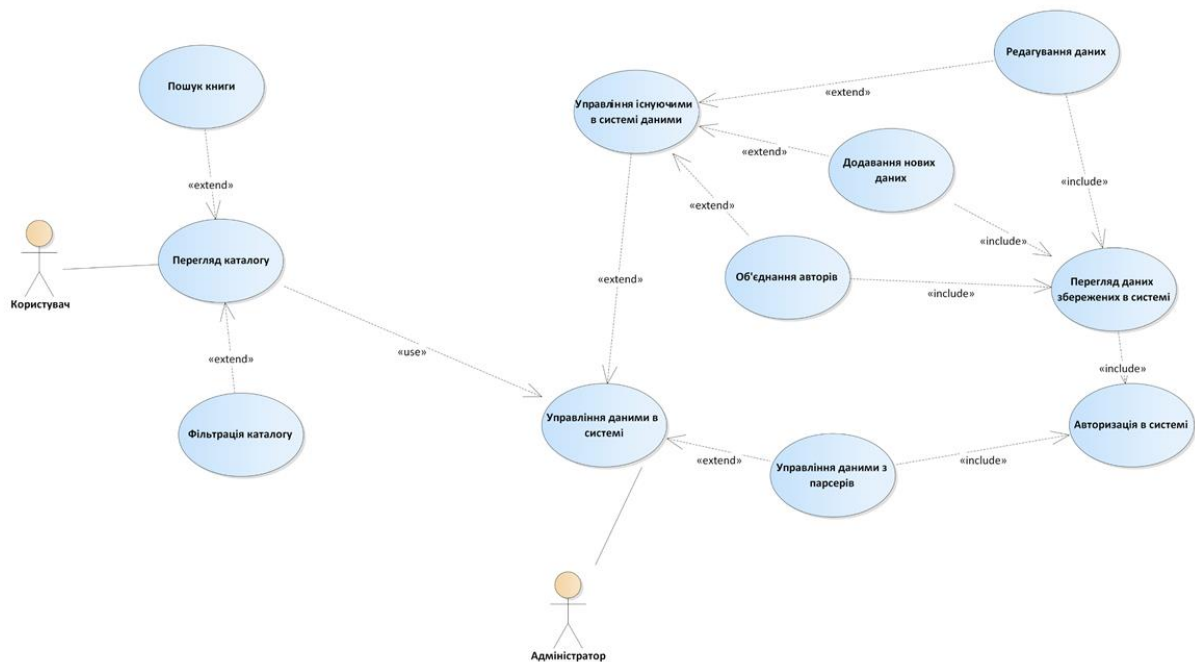


Рис. 2.1. Діаграма варіантів використання системи

2.2.4. Фізичне представлення моделі програмної розробки

Діаграми станів визначають всі можливі стани, в яких може перебувати конкретний об'єкт, а також процес зміни станів об'єкта в результаті настання деяких подій.

На діаграмі є два спеціальних стану - початкове (start) і кінцеве (stop). Початковий стан виділено чорною точкою, він відповідає стану об'єкта, коли він тільки що був створений. Кінцевий стан позначається чорною точкою в білому кружку, він відповідає стану об'єкта перед його знищенням. На діаграмі станів може бути один і тільки один початковий стан.

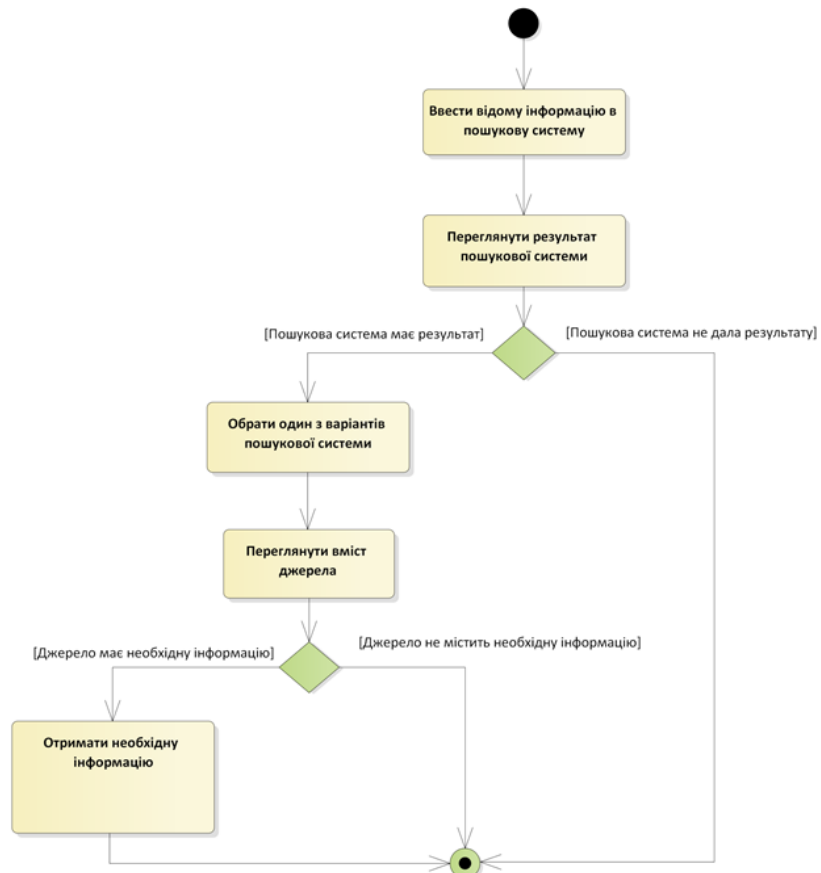
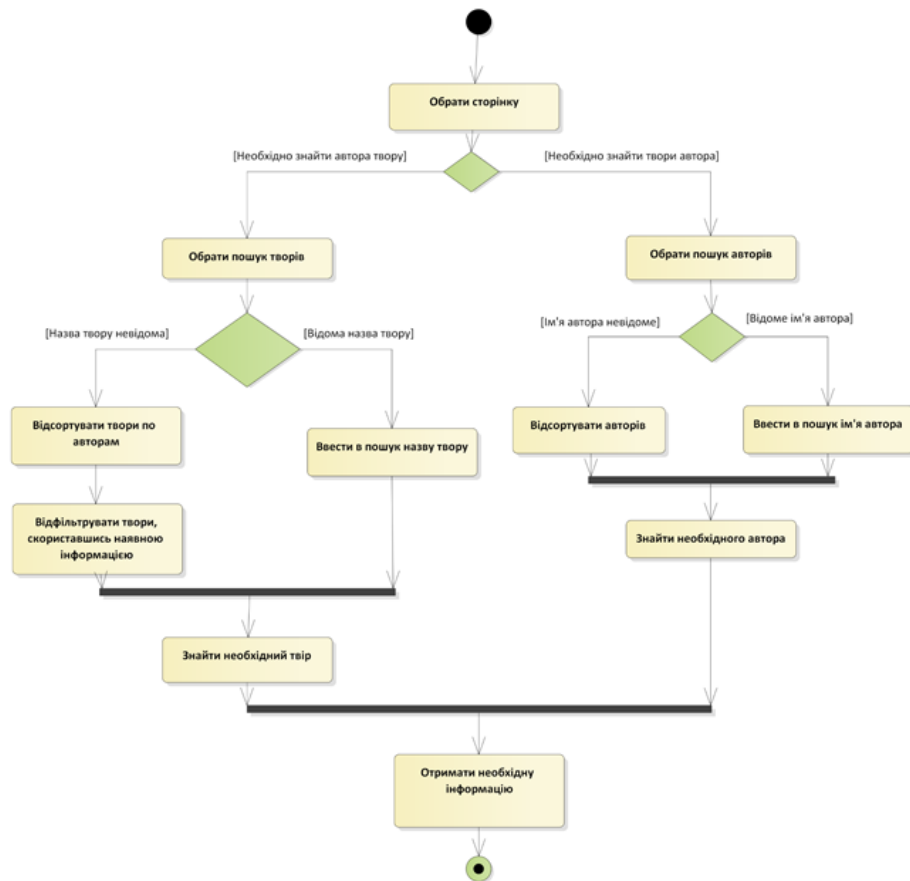


Рис. 2.2. Діаграма станів

2.2.5. Діаграма пакетів

Схема структурна пакетів програмного забезпечення наведена в частині графічного матеріалу.

Інформаційна система складається з 8 пакетів.

- TranslatorCatalog

Даний пакет є вхідною точкою системи. В цьому пакеті налаштовуються всі конфігурації, залежності, формуються головні Dependency Injection контейнери.

- 1) TranslatorCatalog.Models

В даному просторі імен знаходяться моделі, які необхідні для реалізації Арі серверу.

- 2) TranslatorCatalog.Controllers

В даному просторі імен знаходяться контролери для реалізації Арі серверу.

- TC.DataSaver

- 1) TC.DataSaver.Entities

Простір імен який описує сутності, які зберігаються в базі даних.

- 2) TC.DataSaver.Savers, TC.DataSaver.Abstraction

В наведених просторах імен зберігається основні класи, функціями яких є зберігання даних. Тобто головні класи до яких звертаються при необхідності, а вони, в свою чергу, вже викликають створюють та використовують інші класи.

- 3) TC.DataSaver.Savers.SaveChain,
TC.DataSaver.Savers.SaveChain.ChainComponents,
TC.DataSaver.Abstraction.SaveChain

В наведених вище просторах імен зберігаються інтерфейси та їх реалізації необхідні для створення зв'язаного ланцюга класів, які оброблюють дані отримані від парсерів та записують до бази відповідно до типу даних. Даний набір класів є реалізацію патерну Ланцюжок відповідальностей (Chain of responsibility) [30]. Суть патерну полягає в тому, що є класи, послідовно зв'язаних між собою, які оброблюють одне повідомлення.

- 4) TC.DataSaver.Repos, TC.DataSaver.Repos.Abstraction,
TC.DataSaver.Repos.ConcreteRepos,
TC.DataSaver.Repos.Abstraction.ConcreteRepos

В наведених вище просторах імен зберігається інтерфейси та їх реалізації необхідні для створення репозиторіїв. Репозиторій – шаблон проєктування, який реалізує інтерфейс для доступу до даних [29]. В даному випадку, репозиторії полегшують доступ до сутностей, які зберігаються в базі даних.

- TC.Migrations

В даному пакеті зберігається історія міграцій бази даних. Міграція – це інформація про зміни внесені в базу даних [31].

- TC.Authorization

В даному пакеті зберігаються класи необхідні для збереження даних адміністраторів, таких як логін та хеш паролю, та подальшого надання доступу адміністраторам при вході до системи.

- TC.BackgroundService

Пакет в якому зберігаються сервіси, які створюються при першому запуску програми і виконують деяку роботу в фоні. Один з сервісів збирає з парсерів дані, та записує до бази. Інший сервіс працює по таймеру і при запуску дістає з бази прийняті адміністратором книги та передає їх на подальшу обробку.

- TC.Core

В даному пакеті містяться інтерфейси та класи, які використовуються в більшості інших пакетах та дозволяють поєднувати пакети між собою, без явного вказування.

- 1) TC.Core.Repository, TC.Core.Repository.Base

В даних просторах імен містяться інтерфейси та базові реалізації патерну Репозиторій [22].

- 2) TC.Core.Helpers, TC.Core.Helpers.Attributes

В даних просторах імен зберігаються допоміжні класи.

- 3) TC.Core.Savers

В даному просторі імен зберігаються інтерфейси для класів, які зберігають дані. Класи реалізації містяться в інших пакетах.

4) TC.Core.Models, TC.Core.Models.Base

В даних просторах імен містяться моделі основних сутностей.

5) TC.Core.Parser

В даному просторі імен зберігаються інтерфейси для парсерів.

6) TC.Core.Combiner

В даному просторі імен зберігаються класи, які містять інформацію різну інформацію про книгу та комбінують, зберігаючи це в одній сутності.

– TC.Parsers

Даний пакет містить реалізації парсерів.

1) TC.Parsers.DependencyInjection

В даному просторі імен містяться допоміжні класи для реалізації Denendency Injection.

2) TC.Parsers.Services

В даному просторі імен зберігаються сервіси, які використовуються в усіх парсерах.

3) TC.Parsers.Abstraction, TC.Parsers.Abstraction.Infos

В даних просторах імен містяться інтерфейси та абстрактні класи, які мають бути реалізовані в кожному парсері.

4) TC.Parsers.YakabooParser,
TC.Parsers.YakabooParser.ModelParsers,
TC.Parsers.YakabooParser.Info

В даних просторах імен містяться класи парсера сайту yakaboo.ua.

5) TC.Parsers.BookyeParser, TC.Parsers.BookyeParser.Info,
TC.Parsers.BookyeParser.ModelParsers

В даних просторах імен містяться класи парсера сайту book-ye.com.ua

– TC.CommonUI

Даний пакет відповідає за інтерфейс системи.

1) TC.CommonUI.Pages

В даному просторі імен містяться сторінки, які може відвідати користувач.

2) TC.CommonUI.Pages.Admin

- В даному просторі імен містяться сторінки, які може відвідати тільки адміністратор.

1) TC.CommonUI.Shared

Даний простір імен містить класи, спільні для всіх сторінок.

2) TC.CommonUI.Shared.Components

В даному просторі імен містяться компоненти, які використовуються для побудови сторінок.

3) TC.CommonUI.Models

В даному просторі імен містяться моделі даних, які відображаються на сторінках.

4) TC.CommonUI.Models.Admin

- В даному просторі імен містяться моделі даних, які відображаються на сторінках адміністратора.

1) TC.CommonUI.Services, TC.CommonUI.Services.Abstraction,
TC.CommonUI.Services.Realizations

В даних просторах імен містяться інтерфейси та їх реалізації, необхідні для доступу сторінок до даних.

2) TC.CommonUI.Helpers

В даному просторі імен зберігаються допоміжні класи.

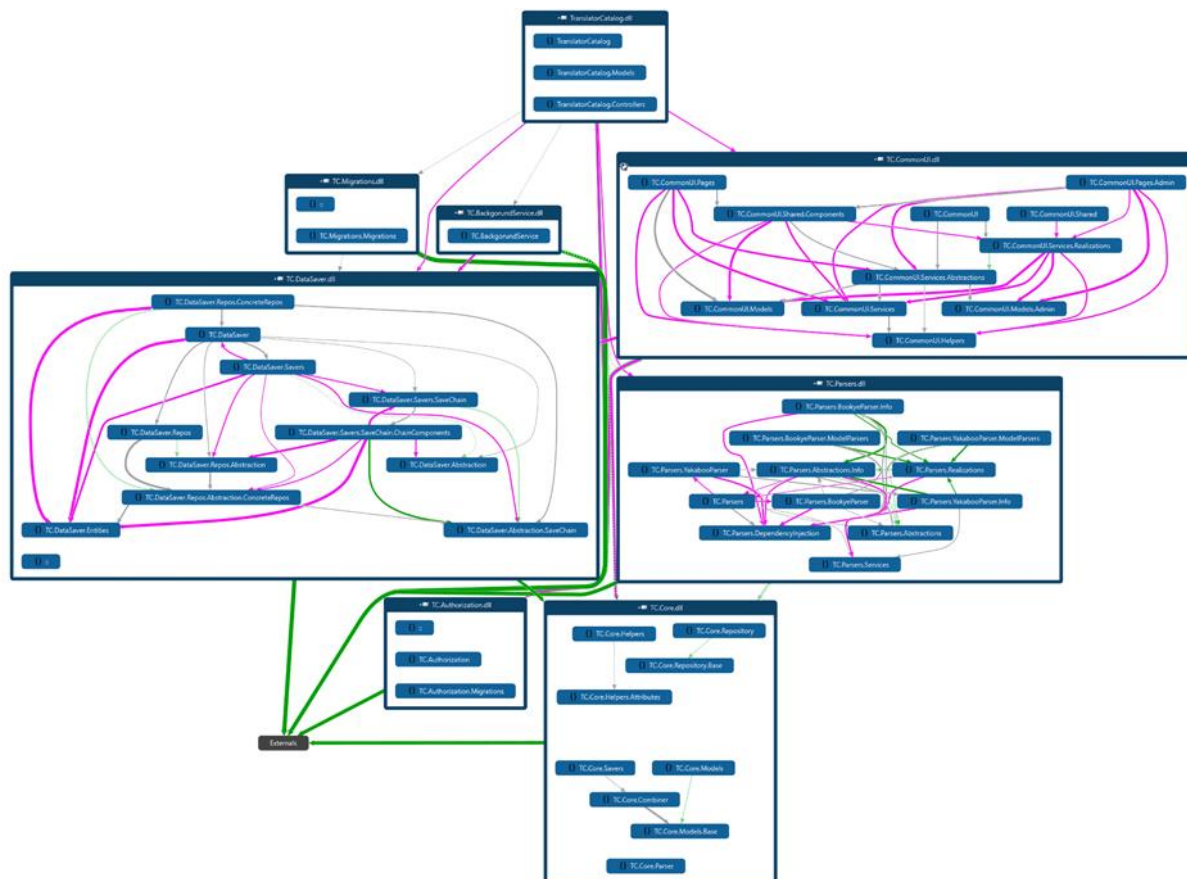


Рис. 2.3. Діаграма пакетів

2.2.6. Діаграма послідовності

Схема структурна послідовності наведені в частині графічного матеріалу.

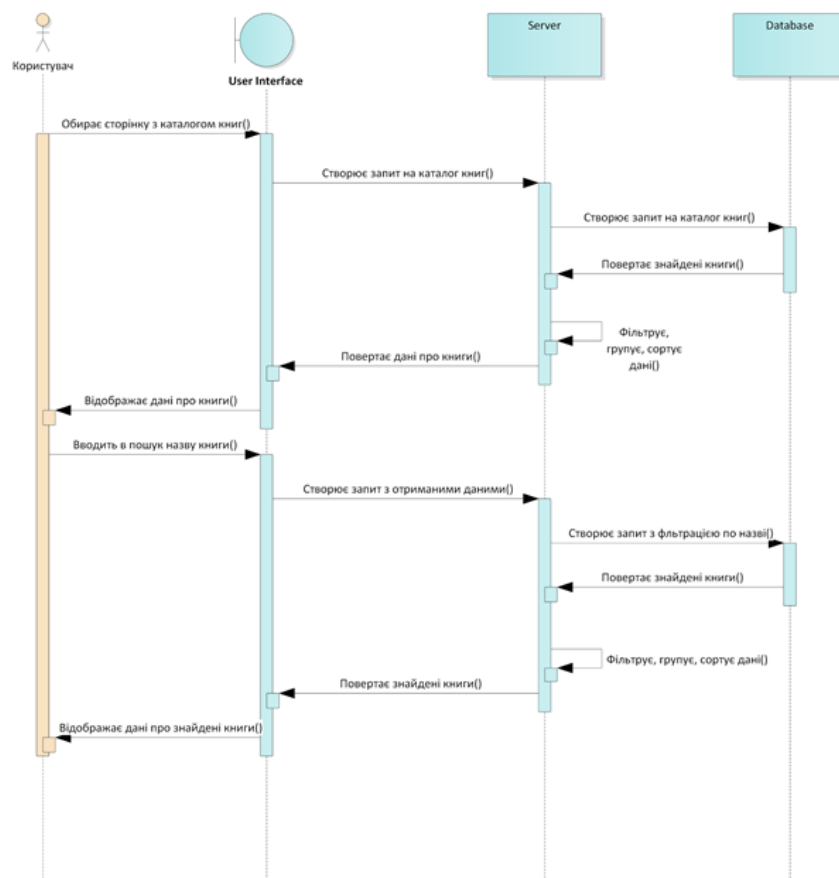
Так як функції користувача та адміністратора в дані системі відрізняються, тому наведено 2 діаграми послідовності.

Користувач взаємодіє з інтерфейсом (User Interface), через нього він обирає сторінку та отримує дані. В свою чергу інтерфейс взаємодіє з сервером (Server). Інтерфейс передає команди від користувача до сервера. Після відповіді сервера, інтерфейс відображає отриману інформацію. Сервер взаємодіє з базою даних (Database). Так як всі дані зберігаються в базі даних. Сервер формує запит, відповідно до отриманих команд. Інформацію, отриману від бази даних, сервер оброблює та відправляє інтерфейсу.

Адміністратор взаємодіє з інтерфейсом. Для доступу до функцій адміністратора, йому потрібно пройти авторизацію. Інтерфейс потребує введення логіну та паролю. Введені дані інтерфейс передає до сервера. Сервер формує запит до бази даних, відповідно до введеного логіну. Отримане хеш-

значення паролю звіряється з обчисленим значенням вхідного паролю. Результат порівнянь сервер передає інтерфейсу. Відповідно чи успішно було пройдено перевірку, інтерфейс дає доступ до функцій адміністратора.

Для того, щоб адміністратор міг отримати інформацію про книги та перевірити її, фоновому сервісу (Background Service) необхідно завантажити дані. Фоновий сервіс посилає запит до Джерела інформації за необхідною інформацією. Після отримання інформації фоновий сервіс зберігає її до бази даних. Фоновий сервіс також перевіряє чи в базі містяться вже перевірені дані і при наявності оброблює їх та зберігає в базі даних.



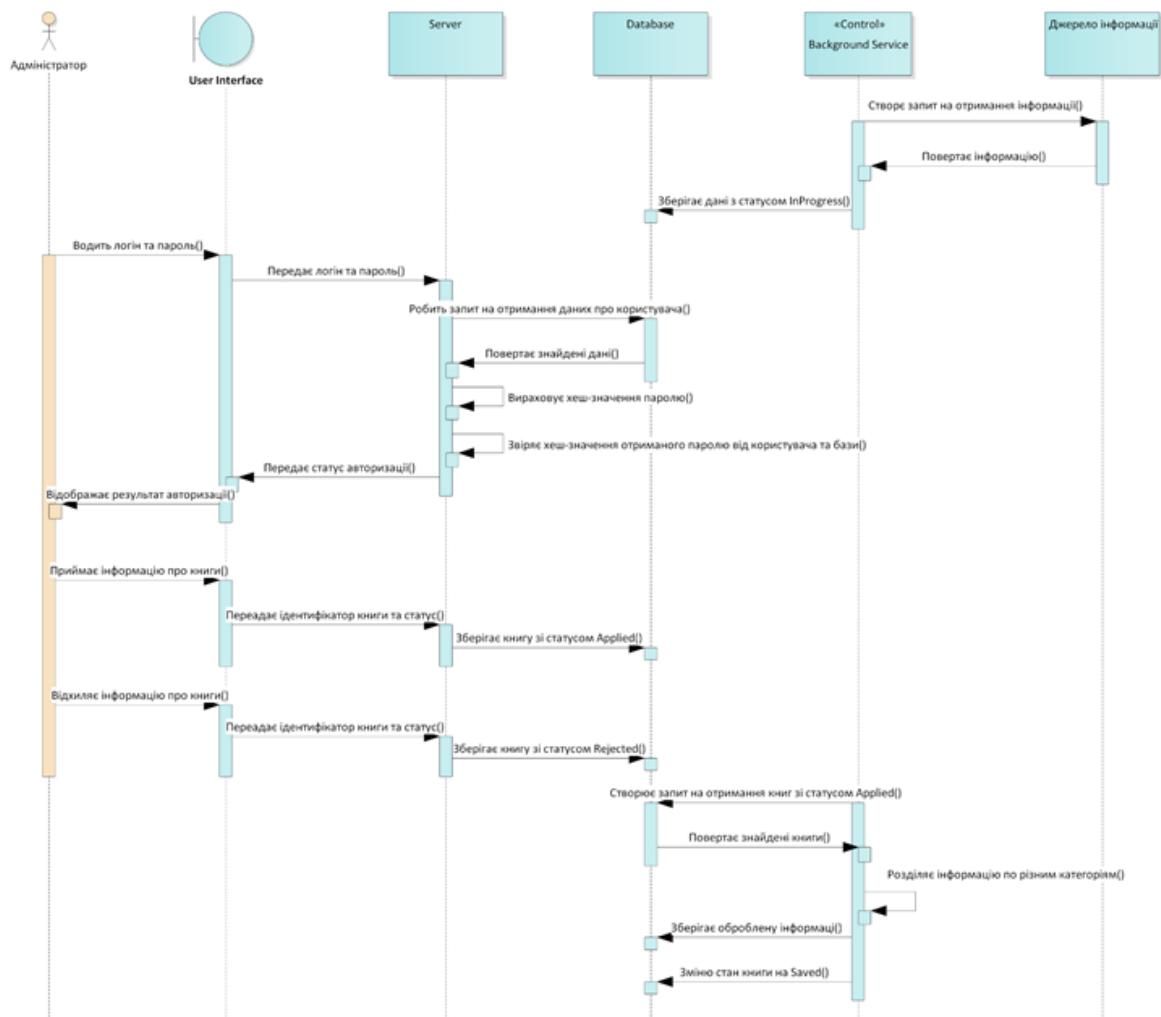


Рис. 2.4. Діаграма послідовності

2.2.7. Діаграма компонентів

Схема структурна компонентів програмного забезпечення наведена в частині графічного матеріалу.

Інформаційна система складається з 3 частин: сервер, інтерфейс та база даних.

Інтерфейс відображає інформації користувачу. За допомогою протоколу WebSocket інтерфейс взаємодіє з сервером, отримує від нього дані та передає команди від користувача.

Сервер взаємодіє з базою даних за допомогою технології Entity to SQL. Реалізацію технології забезпечує фреймворк Entity Framework Core. Суть технології в тому, що код на C# переводиться в код на SQL. Таким чином з бази дістається необхідна інформація та записується нова.

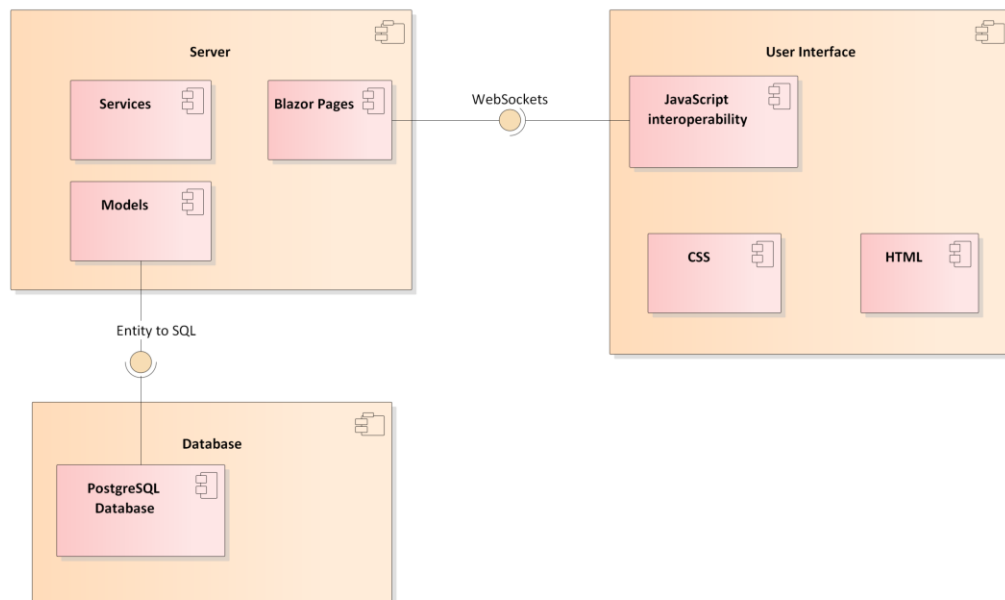


Рис. 2.5. Діаграма компонентів

Висновки до розділу

В даному розділі було описано вимоги до технічного забезпечення та наведено діаграми варіантів, станів, послідовностей, пакетів, компонентів.

РОЗДІЛ 3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДТРИМКИ ПЕРЕКЛАДАЦЬКОЇ ДІЯЛЬНОСТІ НА ПЛАТФОРМІ .NET CORE

3.1. Обґрунтування вибору засобів розробки

Для розробки було використано такі програми, як Visual Studio 2019 Community Edition, Enterprise Architect 14, Visio, DBeaver.

Для розробки інформаційної системи було використано мову програмування C# 8 версії [20][21].

Платформа розробки .NET Core. Більш детальна інформація про версії компонентів:

- SDK версії 3.1.201
- Runtimes:
 - 1) Microsoft.AspNetCore.All версії 2.2.8
 - 2) Microsoft.AspNetCore.App версії 3.1.3
 - 3) Microsoft.NetCore.App версії 3.1.3

Обрана платформа .NET Core розроблена компанією Microsoft та є продуктом з відкритим кодом [22]. Платформа підтримується такими операційними системами як Windows, Linux, macOS. Тому розроблена інформаційна система може бути розгорнута на більшості сучасному комп'ютерному обладнанні.

Для розробки було використано наступні фреймворки:

- ASP.NET Core [23]

Фреймворк для розробки серверної частини інформаційної системи.

- Entity Framework Core [24]

Фреймворк, який реалізує технологію ORM [25]. Надає доступ та полегшує взаємодію з базою даних.

- Blazor

Фреймворк для розробки клієнтської частини інформаційної системи. Фреймворк може бути використаний для генерації HTML, CSS, JavaScript коду на сервері та подальшої передачі на комп'ютер клієнта, так звана ServerSide. Або у вигляді WebAssembly та подальшого розгортання в браузері на комп'ютері

клієнта. На момент розроблення інформаційно системи версія значиться як 3.2.0 Preview 5. Слово Preview в назві версії означає, що фреймворк не рекомендовано використовувати для розробки систем для повного використання. Тому було обрано ServerSide версію фреймворку.

Також при розробці було використано архітектурний шаблон MVVM та архітектурний стиль взаємодії компонентів розподіленого застосунку REST Api [26].

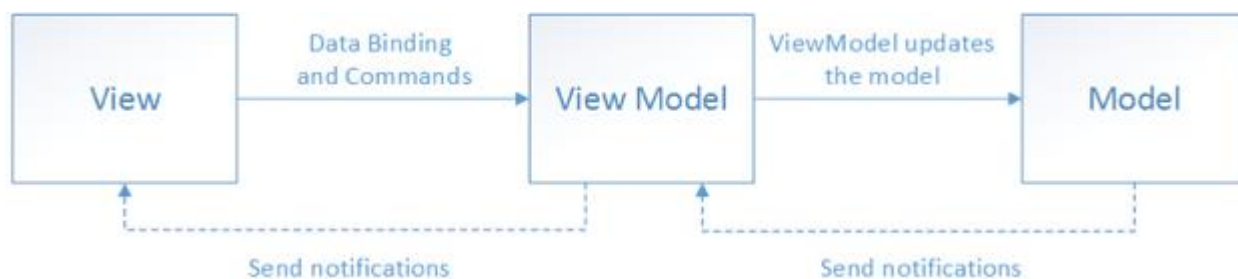


Рис. 3.1. Модель взаємодії шаблону MVVM [27]

MVVM – Model-View-ViewModel шаблон проєктування [28]. Суть полягає в розділенні даних, логіки та інтерфейсу на різні частини.

Model відповідає за дані, які необхідні для відображення інтерфейсу. В моделі може міститись логіка взаємодії та перевірка даних.

ViewModel відповідає за передачу даних на інтерфейс. Тобто такий собі посередник, який фіксує дії користувача та повідомляє про них систему. Також при оновленні даних в Моделі передає результат на інтерфейс.

View інтерфейс, який отримує дані та відображає їх. Фіксує взаємодію користувача з інтерфейсом та повідомляє про це систему через ViewModel.

REST Api – архітектурний стиль для взаємодії веб-сервісів. Тобто деякий набір правил, обмежень, рекомендації як потрібно проєктувати мережеве спілкування систем.

Використовують REST на базі протоколу HTTP 1.1.

Api – application programming interface – інтерфейс який має набір функцій за допомогою яких можна взаємодіяти з системою.

Для взаємодії можуть використовуватись різні формати даних, такі як HTML, JSON, XML. В розробленій інформаційній системі використовується JSON.

Для збереження даних використовується база даних PostgreSQL. PostgreSQL базується на мові SQL і підтримує більшість можливостей з стандарту SQL:2011.

Сильними сторонами PostgreSQL вважаються:

- продуктивність та надійні механізми транзакцій і реплікації;
- підтримка слабоструктурованих даних у форматі JSON.

3.2. Опис структури бази даних

Для зберігання даних було обрано об'єктно-реляційну систему керування базами даних PostgreSQL [5].

Схема бази даних наведена в частині графічного матеріалу.

База даних складається з наступних сутностей (табл. 3.1).

Таблиця 3.1.

Сутності бази даних

№	Назва таблиці	Назва сутності
1	Books	Книга
2	Authors	Автор
3	AuthorNames	Ім'я автора
4	AuthorType	Тип автора
5	Languages	Мова
6	Files	Файл
7	PairsOfSameName	Об'єднані імена авторів
8	Publishers	Видавництво
9	PublsheDates	Дата видавництва
10	IntermdiateModelPacks	Проміжний набір моделей отриманих від парсерів
11	ModelPackStates	Статус проміжного набору моделей
12	NotSaved	Не збережені набори моделей
13	Users	Користувачі

В таблиці 3.2 детально описано представлені сутності у вигляді таблиць та параметри таблиць.

Таблиця 3.2

Опис таблиць бази даних

Назва таблиці	Назва параметру	Тип даних	Опис
Books	Id	int4	Ідентифікатор книги
	Title	text	Назва книги
	Link	text	Посилання на джерело звідки було отримано книгу
	Description	text	Опис книги
	LanguageId	int4	Ідентифікатор мови
	OriginalBookId	int4	Ідентифікатор оригінального твору
Authors	Id	int4	Ідентифікатор автора
	AuthorNameId	int4	Ідентифікатор імені автора
	AuthorTypeId	int4	Ідентифікатор типу автора
AuthorBooks	Id	int4	Ідентифікатор пари автор книга
	BookId	int4	Ідентифікатор книги
	AuthorId	int4	Ідентифікатор автора
AuthorNames	Id	int4	Ідентифікатор імені автора
	FirstName	text	Ім'я автора
	MiddleName	text	Друге ім'я автора
	LastName	text	Прізвище автора
	FullName	text	Повне ім'я автора

Назва таблиці	Назва параметру	Тип даних	Опис
AuthorTypes	Id	int4	Ідентифікатор типу автора
	Code	text	Назва типу
	Description	text	Опис типу
Files	Id	int4	Ідентифікатор файлу
	Title	text	Назва файлу
	Path	text	Шлях до файлу
	Extension	text	Розширення файлу
	Link	text	Посилання на джерело, звідки отримано файл
BookCovers	Id	int4	Ідентифікатор обкладинки
	BookId	int4	Ідентифікатор книги
	FileId	int4	Ідентифікатор файлу
Languages	Id	int4	Ідентифікатор мови
	Code	text	Назва мови
	Description	text	Опис мови
Publisher	Id	int4	Ідентифікатор видавництва
	Title	text	Назва видавництва
PublishDates	Id	int4	Ідентифікатор дати видання
	Date	timestamp	Дата видання
	BookId	int4	Ідентифікатор книги
	PublisherId	int4	Ідентифікатор видавництва

Назва таблиці	Назва параметру	Тип даних	Опис
IntermediateModelPacks	Id	int4	Ідентифікатор проміжного набору моделей
	Content	text	Набір моделей в форматі JSON
	Created	timestamp	Дата створення
	StateId	int4	Стан набору
ModelPackState	Id	int4	Ідентифікатор стану набору моделей
	Code	text	Назва стану
	Description	text	Опис стану
NotSaved	Id	int4	Ідентифікатор не збереженого набору моделей
	Data	text	Набір моделей в форматі JSON
	Error	text	Помилка, через яку не вдалось зберегти
	Created	timestamp	Дата створення
Users	Id	int4	Ідентифікатор користувача
	Login	text	Логін користувача
	PasswordHash	text	Хеш-значення паролю

На рисунку 3.2 зображено концептуальну модель даних.

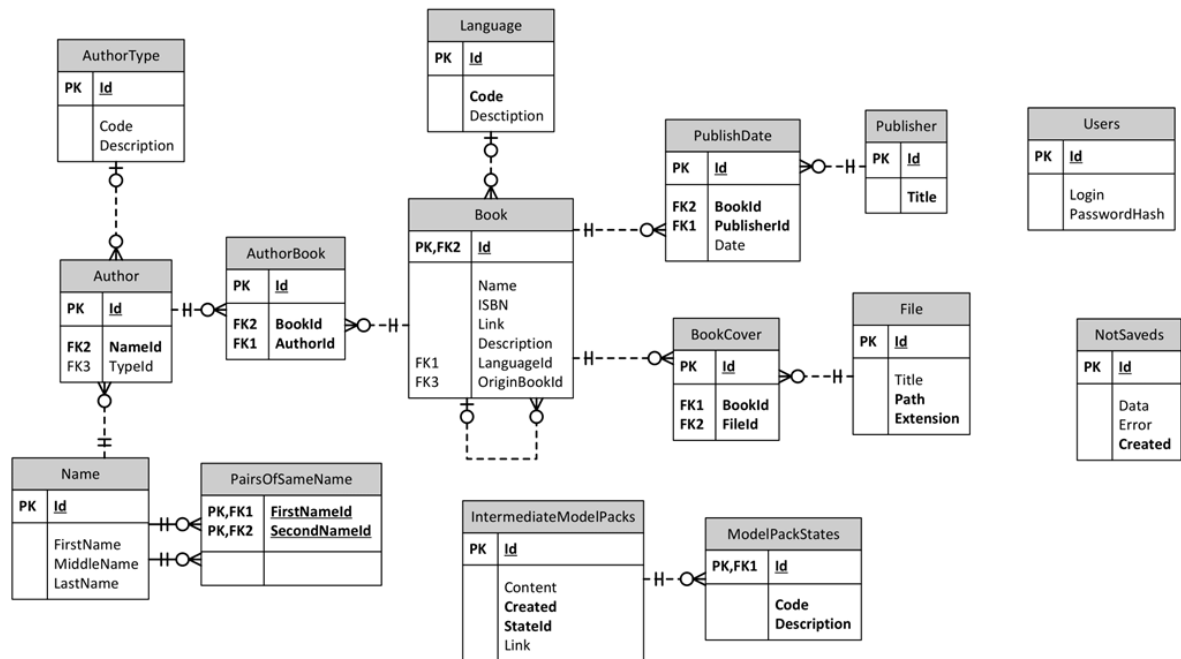


Рис. 3.2. Концептуальна модель даних

3.3. Розробка інтерфейсу

Інтерфейс – це інструмент за допомогою якого користувач може взаємодіяти з розробленою інформаційною системою. Роль інтерфейсу є не менш важливим ніж функціональної частини інформаційної системи. Від того, який досвід отримує користувач даної системи, залежить чи з'явиться в нього бажання і надалі використовувати програму.

Розроблена система дозволяє перекладачам знаходити необхідну інформацію по творах, їх авторам та перекладачам, для подальшого дослідження. Модифікація та керування даними відбувається за допомогою окремого кабінету адміністратора.

Інформаційна система має наступні можливості:

- Пошук книги по ключовим словам
- Пошук за авторами/перекладачами
- Перегляд детальної інформації по книзі
- Фільтрація, сортування списку книг
- Модифікація результатів парсингу
- Корегування інформації про кожну книгу
- Додавання інформації, яку не було знайдено за допомогою парсингу

- Захист від неавторизованого редагування інформації

В системі розрізняються 2 ролі: користувач та адміністратор. Тому для кожної ролі окремо розглянуто процес взаємодії з системою.

Після запуску програми користувач потрапляє на сторінку з каталогом книг (рис. 3.3).

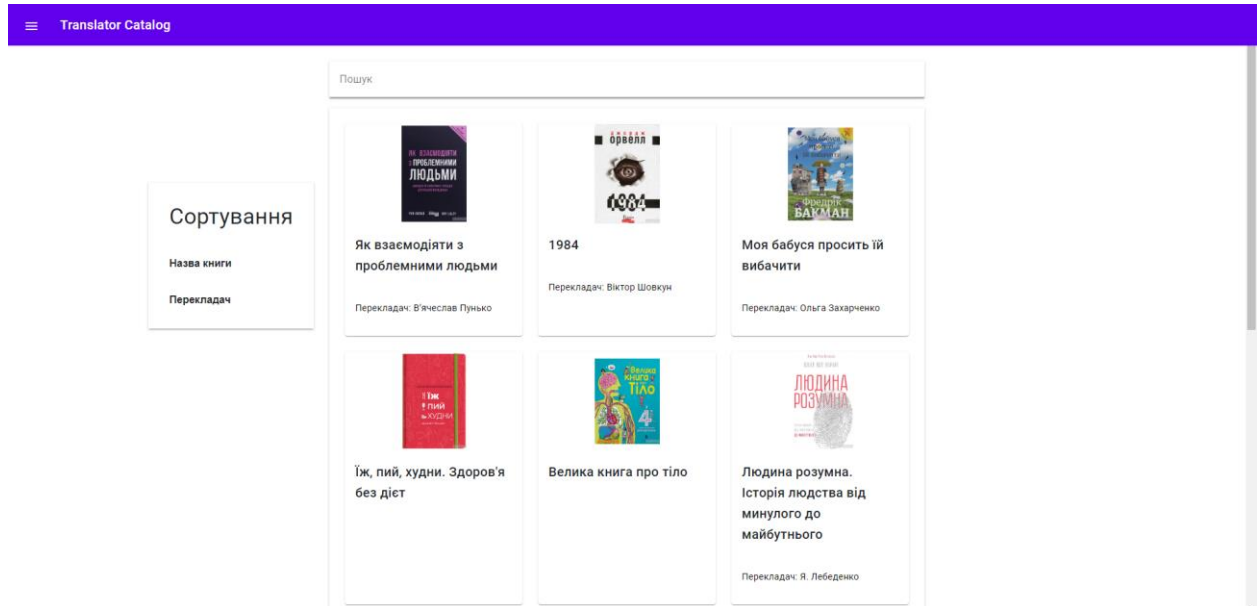


Рис. 3.3. Сторінка з каталогом книг

На даній сторінці користувач може переглядати наявні в системі книги. Якщо користувач знає назву необхідної йому книги, він може скористатись пошуком (рис. 3.4).

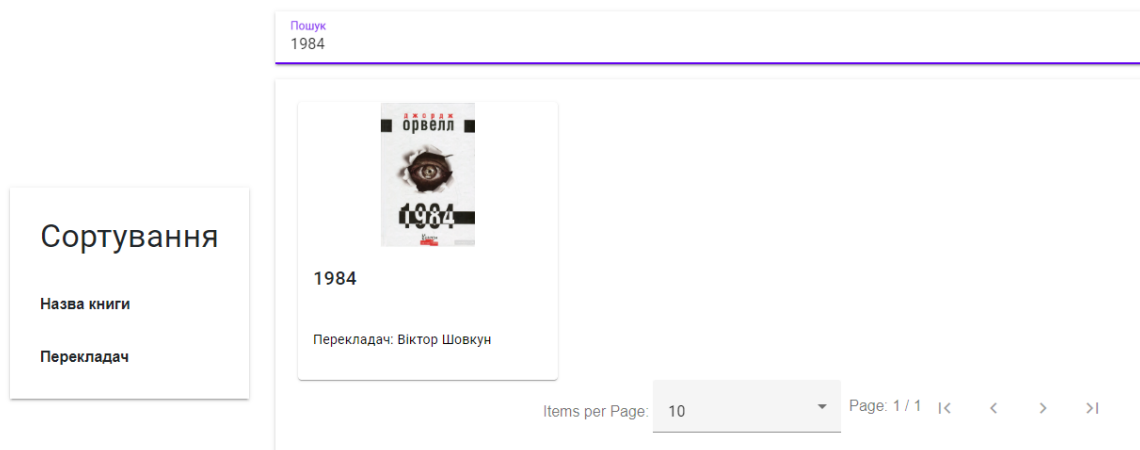


Рис. 3.4. Пошук книги за назвою

Таким чином користувач може фільтрувати книги. Якщо він не володіє достатньою інформацією, то може спробувати ввести частину назви, та обрати книгу з отриманих варіантів (рис. 3.5).

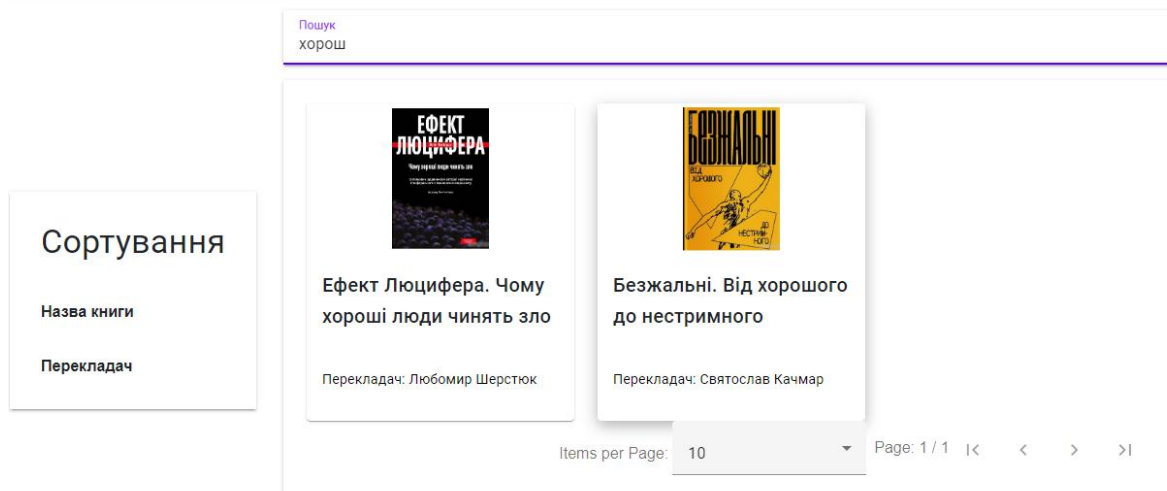


Рис. 3.5. Пошук книги за частиною назви

Для зручного пошуку користувач може скористатись сортуванням. Система надає можливість сортувати книги за назвою та за перекладачем (рис. 3.6).

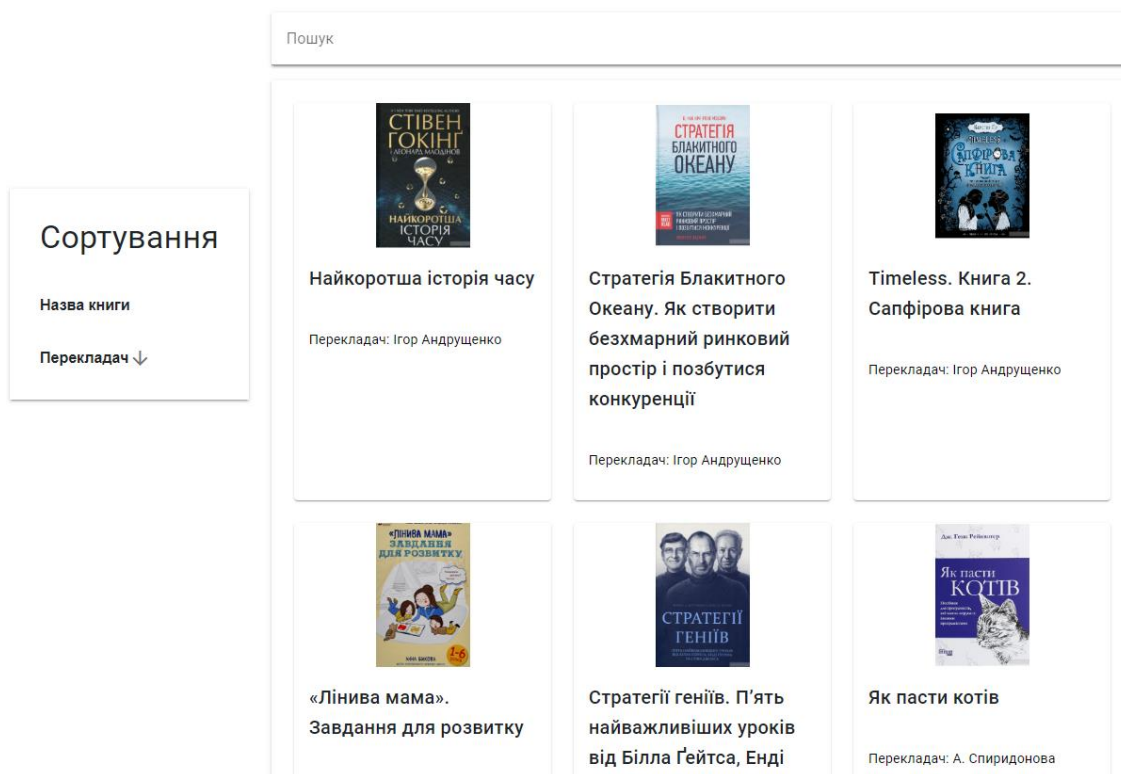


Рис. 3.6. Відсортовані за перекладачем книги

Таким чином система надає можливість шукати необхідну інформацію. Надає можливість знайти книгу за обкладинкою, якщо користувачу відомо як вона виглядає, але не відома назва.

Якщо користувач натисне на книгу він потрапить на сторінку з детальною інформацією про цю книгу (рис. 3.7).

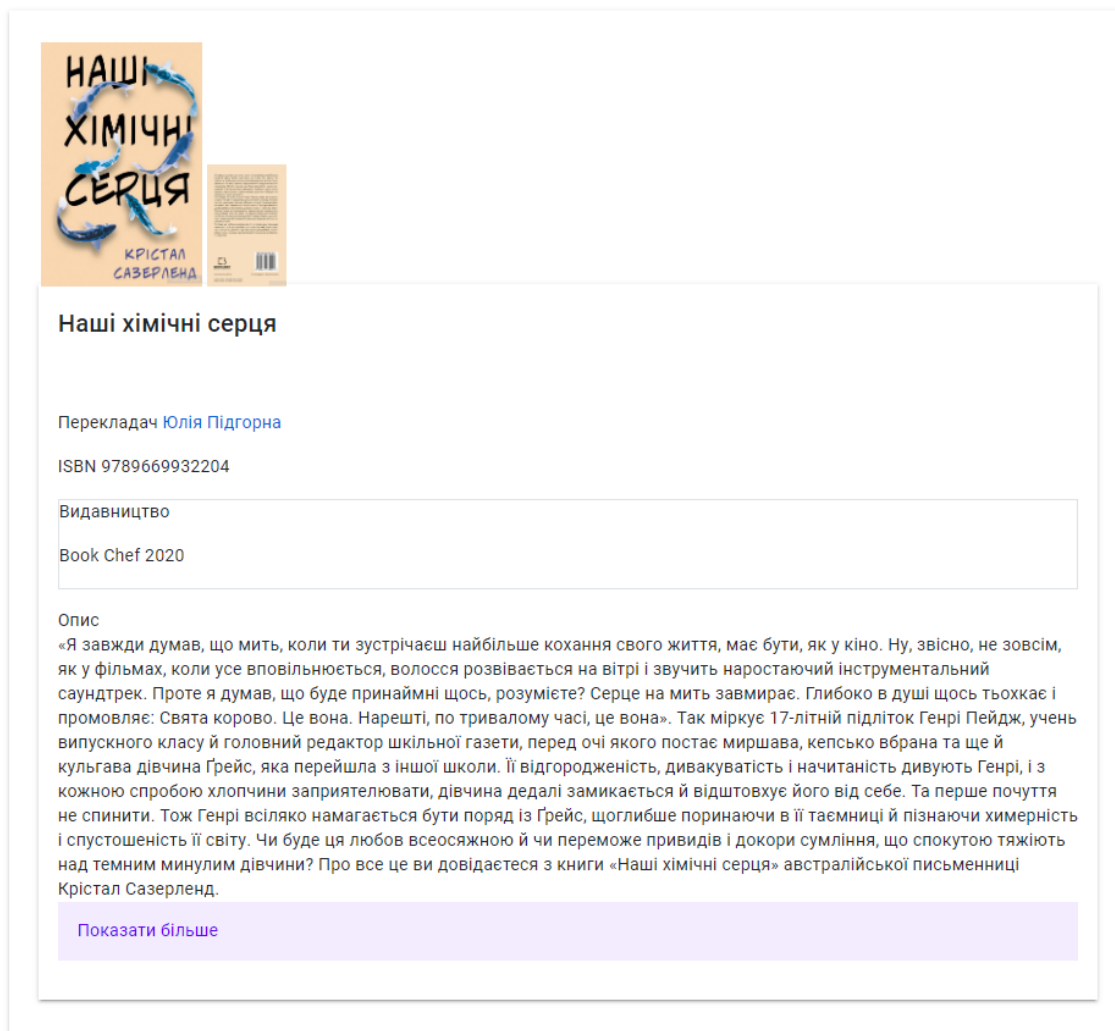


Рис. 3.7. Сторінка з інформацією про книгу

На цій сторінці користувач може знайти інформацію про назву книги, її унікальний номер ISBN, про перекладачів книги та видавництво.

Якщо користувач натисне на ім'я перекладача то перейде на сторінку з книгами, які даний перекладач перевів (рис. 3.8).

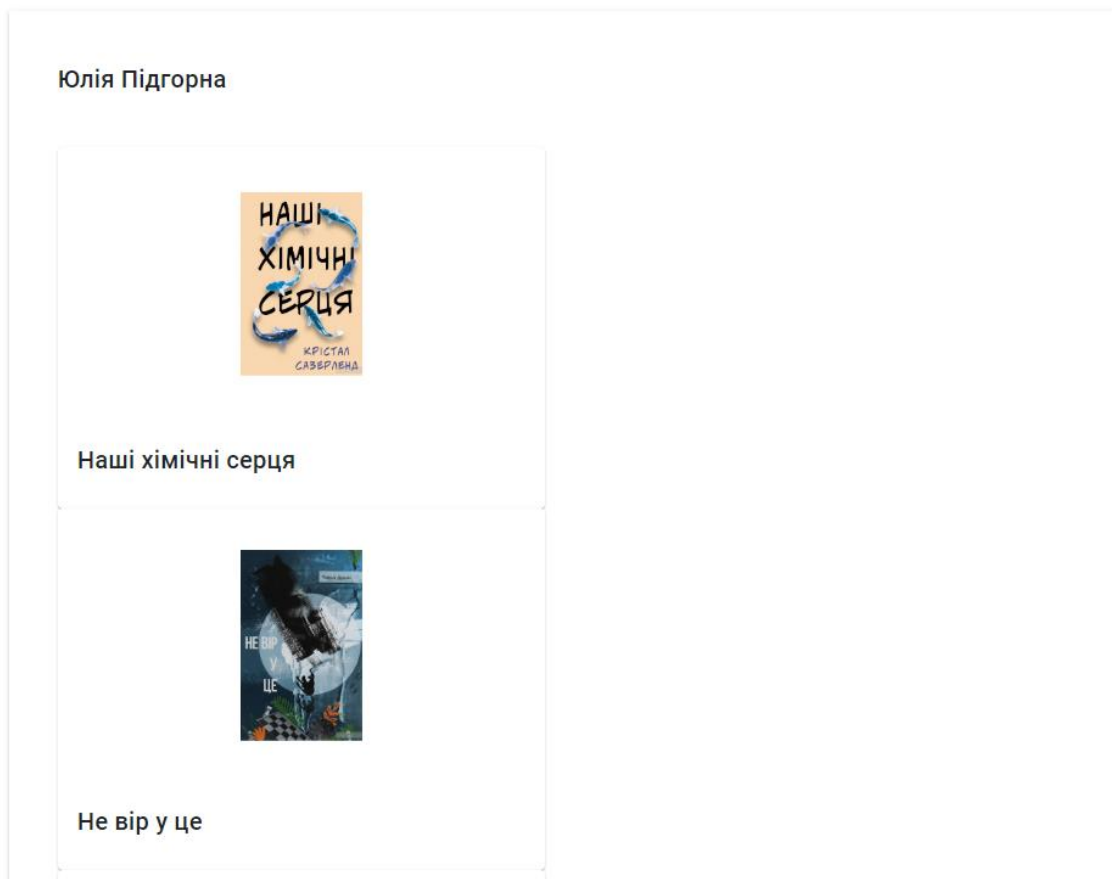


Рис. 3.8. Сторінка з книгами автора

В лівому верхньому вуглі на сторінці розміщена кнопка. При натисканні з'являється навігаційне меню (рис. 3.9).

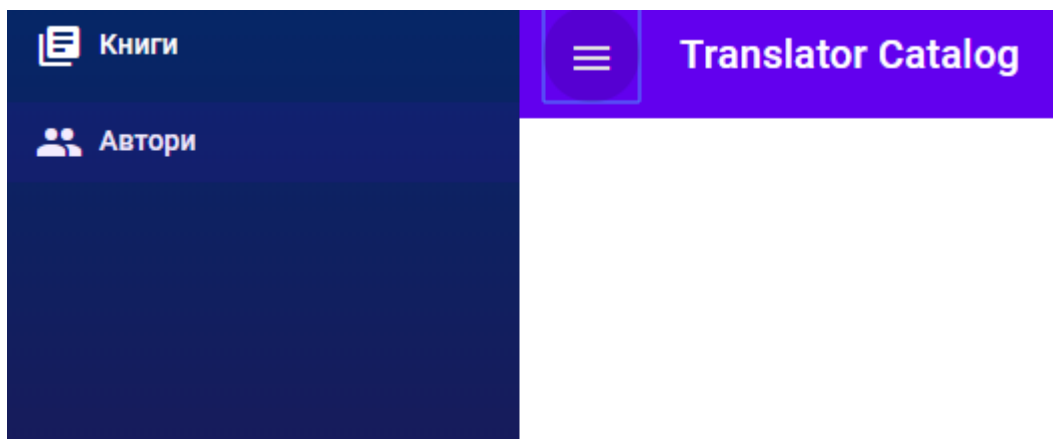


Рис. 3.9. Навігаційне меню

Якщо користувач натисне на кнопку «Автори» в меню то потрапить на сторінку з каталогом авторів (рис. 3.10).

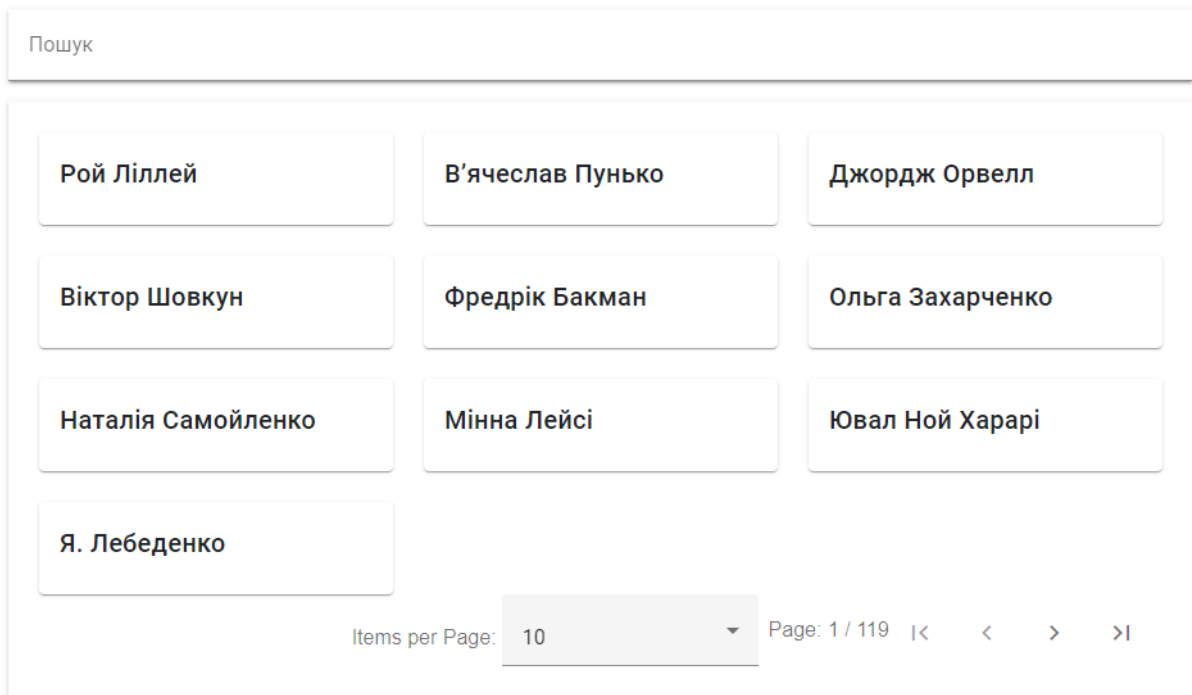


Рис. 3.10. Сторінка з каталогом авторів

Для доступу до функцій адміністратора, користувачу необхідно ввести логін та пароль (рис. 3.11).

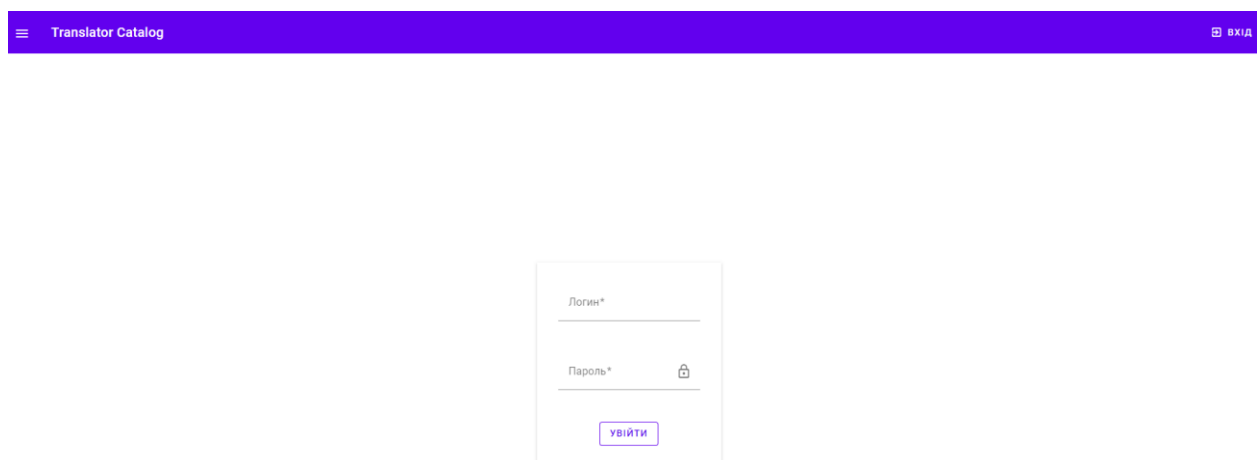


Рис. 3.11. Сторінка авторизації

При неправильному введенні паролю чи логіну користувачу буде повідомлено за допомогою спливаючого вікна (рис.3.12).

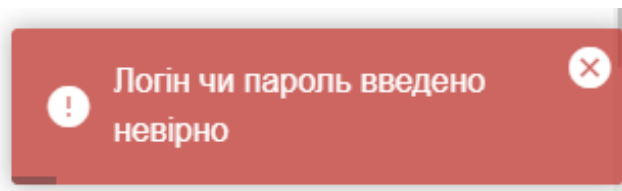


Рис. 3.12. Повідомлення про невірно введені дані

При правильно введеному логіні та паролі система дозволить перейти на сторінки кабінету адміністратора (рис.3.13).

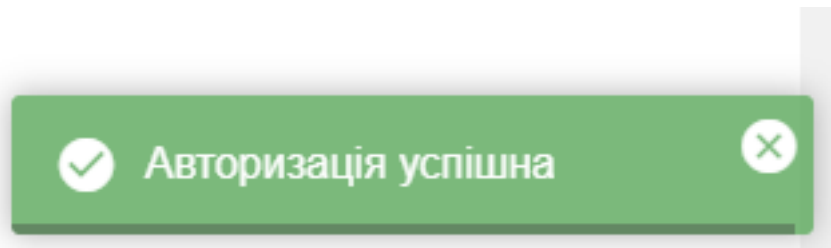


Рис. 3.13. Повідомлення про успішну авторизацію

Після запуску система створює фонові процеси, які збирають дані з зовнішніх джерел. Шляхом парсингу дістаються дані про книги. Зібрані дані зберігаються в базі зі статусом In Progress. Даний статус означає, що дані ще не були перевірені адміністратором, та не можуть бути остаточно збережені в системі. Адміністратор в своєму кабінеті може переглянути всі зібрані з парсерів дані (рис. 3.14).

Посилання	Дата створення	Статус	
https://www.yakaboo.ua/ua/velika-magija.html	13.12.2024 21:56:52	InProgress	РЕДАГУВАТИ
https://www.yakaboo.ua/ua/misto-divchat.html	13.12.2024 21:47:10	Saved	РЕДАГУВАТИ
https://www.yakaboo.ua/ua/charlie-and-the-chocolate-factory.html	13.12.2024 21:47:13	Saved	РЕДАГУВАТИ

Рис. 3.14. Сторінка з даними які отримані з парсерів

Після натискання кнопки «Редагувати» адміністратор потрапляє на сторінку з всією зібраною інформацією про дану книгу (рис. 3.15). Адміністратор може перевірити та модифікувати дані.

Книги

Назва
Велика магія

ISBN
9786176794141

Опис
Пропонуємо вашій увазі книгу Елізабет Гілберт «Велика магія» українською мовою від «Видавництва Старого Лева»!
Про книгу:
Тема творчості та творення сильно сакралізована, тому серед найрізноманітніших авторів

Посилання
<https://www.yakaboo.ua/ua/velika-magija.html>

☐ Оригінальний твір

Назва
Big Magic

Рис. 3.15. Сторінка редагування даних від парсерів

В низу сторінки адміністратор може побачити 3 кнопки (рис. 3.16).

Посилання
https://img.yakaboo.ua/media/catalog/product/cache/1/image/546x/00c1a1eab9920e00d38dc8798e6142c9/i/m/img_0122_5.jpg

Посилання
https://img.yakaboo.ua/media/catalog/product/cache/1/image/546x/00c1a1eab9920e00d38dc8798e6142c9/i/m/img_0123_3.jpg

ПРИЙНЯТИ

ВІДМОВИТИ

НАЗАД

Рис. 3.16. Кнопки для зберігання даних в системі

Кнопка «Прийняти» переводить дані про книгу зі стану In Progress в статус Applied. Таким чином повідомляючи системі, що дані можна зберегти. Фоновий

процес відбирає з усіх даних об'єкти зі статусом Applied та зберігає всі дані до відповідних таблиць. Після чого статус даних переводиться в Saved.

Кнопка «Відмовити» переводить стани даних в Rejected. Повідомляючи системі, що дані не правильні. Такі дані не будуть збережені в системі.

Кнопка «Назад» повертається до попередньої сторінки з усіма даними з парсерів.

В кабінеті адміністратора більше функцій ніж у звичайного користувача. Тому у навігаційному меню з'являються додаткові варіанти (рис. 3.17).

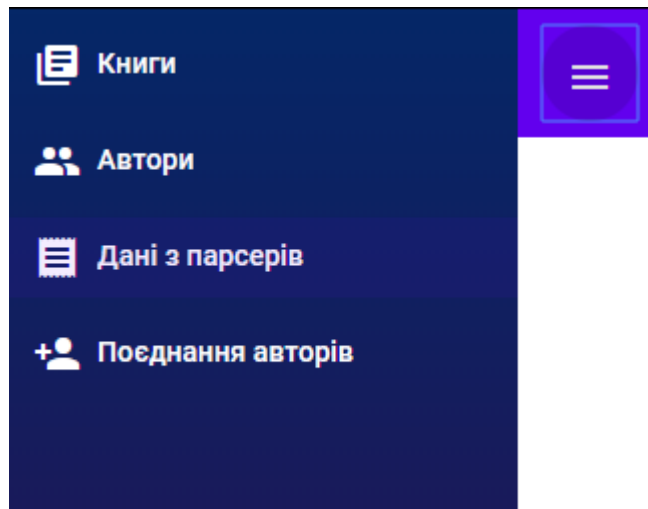


Рис. 3.17. Навігаційне меню адміністратора

При натисканні на «Поєднання авторів» адміністратор потрапляє на сторінку з іменами авторів, які було об'єднано в одну сутність (рис. 3.18).

ДОДАТИ	
Я. Лебеденко	Ярослав Лебеденко
Ярослав Лебеденко	Я. Лебеденко
І. Іванченко	Ірина Іванченко
А. Новікова	Світлана Новікова
Н. Лавська	Наталія Вікторівна Лавська
Ірина Іванченко	І. Іванченко
Світлана Новікова	А. Новікова
Наталія Вікторівна Лавська	Н. Лавська

Рис. 3.18. Сторінка з поєднаними авторами

Якщо натиснути на кнопку «Додати» то адміністратору зможе створити нову пару імен авторів, які будуть об'єднані в одну сутність (рис. 3.19).

Перший автор

Другий автор

ДОДАТИ

Рис. 3.19. Спосіб створення пари авторів

Висновки до розділу

В даному розділі було наведено засоби розробки та обґрунтування їх вибору. Описано структуру бази даних необхідну для зберігання зібраної інформації та надання можливостей інформаційної системи. Описано процес роботи з системою за допомогою користувацького інтерфейсу. Наведено скріншоти програми.

ВИСНОВКИ

В даній роботі було розглянуто та описано створення інформаційної системи підтримки перекладацької діяльності на основі технологій проєктування .NET Core.

Для досягнення поставленої мети були вирішені наступні задачі:

- Проаналізовано проблему відсутності каталогу перекладачів та засобів отримання та пошуку інформації по літературним творам, їх авторів та перекладачів;
- Проаналізовано можливості платформи .NET Core, фреймворків та бібліотек;
- Розроблено інформаційну систему на базі платформи .NET Core та мові програмування C# для оцінки та демонстрації можливості технологій;
- Реалізовано збір даних з відкритих джерел методом парсингу.
- Розроблено демонстрацію, пошук, фільтрацію, управління зібраними даними.
- Реалізовано алгоритм забезпечення захисту системи від неавторизованого доступу шляхом використання хеш-значень паролю.

В першому розділі проаналізовано предметне середовище інформаційної системи. Було наведено загальну інформацію про проблеми, які вирішує дана система та описано способи вирішення. Розглянуто існуючі аналоги. Описано математичне забезпечення, яке було використано в даній роботі. Розглянуто проблеми, які необхідно було вирішити в процесі розробки інформаційної системи. Описано можливі шляхи вирішення проблем. Описано алгоритми.

В другому розділі наведено структуру інформаційної системи. Описано вимоги до технічного забезпечення необхідного для розробки та користування інформаційної системи на платформі .NET Core. Наведено опис пакетів та компонентів системи та їх призначення.

В третьому розділі описано технології, які було використано при розробці системи, їх призначення та використання. Описано структуру бази даних необхідну для зберігання зібраної інформації та надання можливостей інформаційної системи. Розглянуто процес роботи системи, описані функції та продемонстровано можливості системи.

Розроблена система дозволяє збирати інформацію з відкритих джерел шляхом парсингу. Після редагування зібраної інформації її можна використовувати для різних цілей. Система дозволяє групувати, об'єднувати авторів. Таким чином інформаційна система може бути використана для наступних цілей:

- Єдиний каталог перекладачів;
- Набір інформації про перекладачів для дослідження стилю авторів;
- Набір даних про групи перекладачів для подальших досліджень.

Дана інформаційна система демонструє можливості платформи .NET Core для проектування та розробки аналогічних систем. Було реалізовано та вирішено проблеми різної складності для дослідження наявних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кудінов І. О. Основи наукового цитування [Електронний ресурс]// Режим доступу: <https://www.donnu.edu.ua/wp-content/uploads/sites/8/2019/08/Kudinov-I.O.-Osnovi-naukovogo-czitivannya.pdf>.
2. Електронний каталог [Електронний ресурс]// Режим доступу: https://uk.wikipedia.org/wiki/%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B8%D0%B9_%D0%BA%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3
3. Інтернет магазин «Yakaboo» [Електронний ресурс]// Режим доступу: <https://www.yakaboo.ua/>
4. Інтернет магазин «Book-ye» [Електронний ресурс]// Режим доступу: <https://book-ye.com.ua/>
5. PostgreSQL [Електронний ресурс]// Режим доступу: <https://uk.wikipedia.org/wiki/PostgreSQL>
6. Автентифікація [Електронний ресурс]// Режим доступу: <https://uk.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F>
7. Why do you need to Salt and Hash passwords? [Електронний ресурс]// Режим доступу: <https://culttt.com/2013/01/21/why-do-you-need-to-salt-and-hash-passwords/>
8. Хеш-функція [Електронний ресурс]// Режим доступу: <https://uk.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F>
9. MD5, SHA-1, SHA-256 and SHA-512 speed performance [Електронний ресурс]// Режим доступу: <https://automationrhapsody.com/md5-sha-1-sha-256-sha-512-speed-performance/>
- 10.3 Reasons why MD5 is not Secure [Електронний ресурс]// <https://www.md5online.org/blog/why-md5-is-not-safe/>

11. At death's door for years, widely used SHA1 function is now dead [Електронний ресурс]// Режим доступу: <https://arstechnica.com/information-technology/2017/02/at-deaths-door-for-years-widely-used-sha1-function-is-now-dead/>
12. Колізія геш-функції [Електронний ресурс]// Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D1%96%D0%B7%D1%96%D1%8F_%D0%B3%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%97
13. RFC2898 [Електронний ресурс]// Режим доступу: <https://tools.ietf.org/html/rfc2898>
14. Сіль (криптографія) [Електронний ресурс]// Режим доступу: [https://uk.wikipedia.org/wiki/%D0%A1%D1%96%D0%BB%D1%8C_\(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%8F\)](https://uk.wikipedia.org/wiki/%D0%A1%D1%96%D0%BB%D1%8C_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%8F))
15. Функція формування ключа [Електронний ресурс]// Режим доступу: https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BA%D0%BB%D1%8E%D1%87%D0%B0
16. PBKDF2 [Електронний ресурс]// Режим доступу: <https://uk.wikipedia.org/wiki/PBKDF2>
17. Base64 [Електронний ресурс]// Режим доступу: <https://uk.wikipedia.org/wiki/Base64>
18. Виключна диз'юнкція [Електронний ресурс]// Режим доступу: https://uk.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%BB%D1%8E%D1%87%D0%BD%D0%B0_%D0%B4%D0%B8%D0%B7%27%D1%8E_%D0%BD%D0%BA%D1%86%D1%96%D1%8F
19. Расстояние Дамерау – Левенштейна [Електронний ресурс]// Режим доступу: <https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%81%D1%82>

[%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5 %D0%94%D0%B0%D0%BC%D0%B5%D1%80%D0%B0%D1%83 %E2%80%94 %D0%9B%D0%B5%D0%B2%D0%B5%D0%BD%D1%88%D1%82%D0%B5%D0%B9%D0%BD%D0%B0](#)

- 20.Троелсен Е. Язык программирования C# 7 и платформы .NET и .NET Core / Троелсен Е., Джепикс Ф. – К. : Диалектика, 2018. — 1328 с.
- 21.Ріхтер Дж. CLR via C# / Ріхтер Дж. – К. : Microsoft Press, 2012. – 896 с.
- 22.Лок Е. ASP.NET Core in Action / Лок Е. – К. : Manning Publications, 2018. – 712 с.
- 23.Чаберс Дж. ASP.NET Core Application Development: Building an application in four sprints / Чамерс Дж., Паккет Д., Тіммс С. – К. : Microsoft Press, 2016. – 432 с.
- 24.Леман Дж. Programming Entity Framework: Code First / Леман Дж., Міллер Р. – К. : O'Reilly Media, 2011. – 192 с.
- 25.Об'єктно-реляційне відображення [Електронний ресурс]// Режим доступу: [https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B5 %D0%B2%D1%96%D0%B4%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F](#)
- 26.REST [Електронний ресурс]// Режим доступу: [https://uk.wikipedia.org/wiki/REST](#)
- 27.Шаблон Model-View-ViewModel [Електронний ресурс]// Режим доступу: [https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/enterprise-application-patterns/mvvm](#)
- 28.Model-View-ViewModel [Електронний ресурс]// Режим доступу: [https://uk.wikipedia.org/wiki/Model-View-ViewModel](#)
- 29.Паттерн «Репозиторий». Основі и разьяснения [Електронний ресурс]// Режим доступу: [https://habr.com/ru/post/248505/](#)

30.Ланцюжок відповідальностей [Електронний ресурс]// Режим доступу:

https://uk.wikipedia.org/wiki/%D0%9B%D0%B0%D0%BD%D1%86%D1%8E%D0%B6%D0%BE%D0%BA_%D0%B2%D1%96%D0%B4%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9

31.Міграція бази даних [Електронний ресурс]// Режим доступу

https://uk.wikipedia.org/wiki/%D0%9C%D1%96%D0%B3%D1%80%D0%B0%D1%86%D1%96%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85

ДОДАТОК А

Лістинг коду

TC.Core

```
namespace TC.Core.Combiner
{
    public class ModelPack
    {
        private readonly Dictionary<Type, IList<IModel>> modelStorage = new Dictionary<Type,
IList<IModel>>();

        public string Link { get; set; }

        public void AddModel(IModel model)
        {
            var type = model.GetType();
            if(!this.modelStorage.TryGetValue(type, out IList<IModel> models))
            {
                models = new List<IModel>();
                this.modelStorage.Add(type, models);
            }
            models.Add(model);
        }

        public IEnumerable<IModel> GetModelOrDefault(Type type)
        {
            this.modelStorage.TryGetValue(type, out IList<IModel> model);
            return model;
        }

        public void AddModels(IEnumerable<IModel> models)
        {
            foreach(var m in models)
            {
                AddModel(m);
            }
        }

        internal Dictionary<Type, IList<IModel>> GetAll()
            => this.modelStorage;
    }
}

namespace TC.Core.Combiner
{
    internal class ModelPackJsonEntity
    {
        public string FullType { get; set; }
        public IList<IModel> Models { get; set; }
    }

    internal class ModelPackJsonEntityWithObjects
    {
        public string FullType { get; set; }
        public IList<object> Models { get; set; }
    }
}

namespace TC.Core.Combiner
{
    public static class ModelPackSerializer
    {
        public static string ToJson(ModelPack modelPack)
        {

```

```

        var modelEntities = modelPack.GetAll().Select(m => new
ModelPackJsonEntityWithObjects
    {
        FullType = m.Key.FullName,
        Models = m.Value.Select(m => (object)m).ToList()
    }).ToList();
        return JsonSerializer.Serialize(modelEntities);
    }

    public static ModelPack ToModelPack(string json)
    {
        JsonDocument jsonDocument = JsonDocument.Parse(json);
        var elements = jsonDocument.RootElement.EnumerateArray();
        var entities = elements.Select(e => ConverJsonElement(e)).ToList();
        jsonDocument.Dispose();

        ModelPack modelPack = new ModelPack();
        foreach (var e in entities)
        {
            modelPack.AddModels(e.Models);
        }
        return modelPack;
    }
    private static ModelPackJsonEntity ConverJsonElement(JsonElement element)
    {
        var typeName = element.GetProperty("FullType").GetString();
        var assembly = typeof(IModel).Assembly;
        var type = assembly.GetType(typeName);

        var modelElements = element.GetProperty("Models").EnumerateArray();
        var models = modelElements.Select(m =>
m.ToObject(type)).Cast<IModel>().ToList();

        return new ModelPackJsonEntity
        {
            FullType = typeName,
            Models = models
        };
    }
}
internal static class JsonExtensions
{
    public static T ToObject<T>(this JsonElement element, JsonSerializerOptions options
= default)
    {
        var bufferWriter = WriteElement(element);
        return JsonSerializer.Deserialize<T>(bufferWriter.WrittenSpan, options);
    }

    public static T ToObject<T>(this JsonDocument document, JsonSerializerOptions
options = default)
    {
        if (document == null)
            throw new ArgumentNullException(nameof(document));
        return document.RootElement.ToObject<T>(options);
    }

    public static object ToObject(this JsonElement element, Type type,
JsonSerializerOptions option = default)
    {
        var bufferWriter = WriteElement(element);
        return JsonSerializer.Deserialize(bufferWriter.WrittenSpan, type, option);
    }
    public static object ToObject(this JsonDocument document, Type type,
JsonSerializerOptions options = default)
    {

```

```

        if (document == null)
            throw new ArgumentNullException(nameof(document));
        return document.RootElement.ToObject(type, options);
    }

    private static ArrayBufferWriter<byte> WriteElement(JsonElement jsonElement)
    {
        var bufferWriter = new ArrayBufferWriter<byte>();
        using (var writer = new Utf8JsonWriter(bufferWriter))
        {
            jsonElement.WriteTo(writer);
            writer.Flush();
        }
        return bufferWriter;
    }
}

namespace TC.Core.Helpers
{
    public static class FromHierarchyExtension
    {
        public static IEnumerable<TSource> FromHierarchy<TSource>(
            this TSource source,
            Func<TSource, TSource> nextItem,
            Func<TSource, bool> canContinue)
        {
            for (var current = source; canContinue(current); current = nextItem(current))
            {
                yield return current;
            }
        }

        public static IEnumerable<TSource> FromHierarchy<TSource>(
            this TSource source,
            Func<TSource, TSource> nextItem)
            where TSource : class
        {
            return FromHierarchy(source, nextItem, s => s != null);
        }
    }
}

namespace TC.Core.Helpers
{
    public static class NameHelper
    {
        public static (string, string, string) ParseName(string fullName)
        {
            string firstName = null;
            string middleName = null;
            string lastName = null;
            if (!string.IsNullOrEmpty(fullName))
            {
                var names = fullName.Split(new string[] { " " },
                    StringSplitOptions.RemoveEmptyEntries);
                if (names.Length > 0)
                {
                    firstName = names[0];
                }
                if (names.Length == 2)
                {
                    lastName = names[1];
                }
                else if (names.Length >= 2)
                {

```

```

        middleName = names[1];
        lastName = names[2];
    }
}
return (firstName, middleName, lastName);
}
}
}
namespace TC.Core.Helpers
{
    public static class SequentiallyTaskExecution
    {
        public static async IAsyncEnumerable<TResult> RunSequentially<TResult>(this
IEnumerable<Task<TResult>> tasks)
        {
            foreach(var t in tasks)
            {
                yield return await t;
            }
        }
    }
}
namespace TC.Core.Models
{
    public class Author : IModel
    {
        public string FirstName { get; set; }
        public string MiddleName { get; set; }
        public string LastName { get; set; }
        public string FullName { get; set; }
        public bool IsTranslator { get; set; }
    }
}
namespace TC.Core.Models
{
    public class Book : IModel
    {
        public string Title { get; set; }
        public string ISBN { get; set; }
        public string Link { get; set; }
        public string Description { get; set; }
        public string Language { get; set; }
        public bool IsOriginalBook { get; set; }
    }
}
namespace TC.Core.Models
{
    public class BookCover: IModel
    {
        public IList<string> Links { get; set; }
    }
}
namespace TC.Core.Models
{
    public class Publisher : IModel
    {
        public string Name { get; set; }
        public DateTime? Date { get; set; }
    }
}
namespace TC.Core.Repository.Base
{
    public interface IRepository<TEntity> where TEntity : class
    {
        TEntity GetById(int id);
    }
}

```

```

        Task<TEntity> GetByIdAsync(int id);
        IEnumerable<TEntity> GetAll();
        Task<IEnumerable<TEntity>> GetAllAsync();

        void Add(TEntity entity);
        Task AddAsync(TEntity entity);

        void AddRange(IEnumerable<TEntity> entities);
        Task AddRangeAsync(IEnumerable<TEntity> entities);

        void Remove(TEntity entity);
        void RemoveRange(IEnumerable<TEntity> entities);

        Task<int?> IsExistAsync(TEntity entity);
    }
}
namespace TC.Core.Repository.Base
{
    public interface IUnitOfWork
    {
        void SaveChanges();
        Task SaveChangesAsync();
    }
}

namespace TC.Core.Repository
{
    public class Repository<TEntity> : IRepository<TEntity>, IUnitOfWork where TEntity :
class
    {
        protected readonly DbContext dbContext;
        public Repository(DbContext dbContext)
        {
            this.dbContext = dbContext;
        }
        public void Add(TEntity entity)
        {
            this.dbContext.Add(entity);
        }

        public async Task AddAsync(TEntity entity)
        {
            await this.dbContext.AddAsync(entity);
        }

        public void AddRange(IEnumerable<TEntity> entities)
        {
            this.dbContext.AddRange(entities);
        }

        public async Task AddRangeAsync(IEnumerable<TEntity> entities)
        {
            await this.dbContext.AddRangeAsync(entities);
        }

        public IEnumerable<TEntity> GetAll()
        {
            return this.dbContext.Set<TEntity>().ToList();
        }

        public async Task<IEnumerable<TEntity>> GetAllAsync()
        {
            return await this.dbContext.Set<TEntity>().ToListAsync();
        }
    }
}

```

```

    public TEntity GetById(int id)
    {
        return this.dbContext.Set<TEntity>().Find(id);
    }

    public async Task<TEntity> GetByIdAsync(int id)
    {
        return await this.dbContext.Set<TEntity>().FindAsync(id);
    }

    public virtual Task<int?> IsExistAsync(TEntity entity)
    {
        return null;
    }

    public void Remove(TEntity entity)
    {
        this.dbContext.Set<TEntity>().Remove(entity);
    }

    public void RemoveRange(IEnumerable<TEntity> entities)
    {
        this.dbContext.Set<TEntity>().RemoveRange(entities);
    }

    public void SaveChanges()
    {
        this.dbContext.SaveChanges();
    }

    public async Task SaveChangesAsync()
    {
        await this.dbContext.SaveChangesAsync();
    }
}
}
namespace TC.Core.Savers
{
    public interface ISaver
    {
        Task SaveAsync(ModelPack modelPack);
    }
}
namespace TC.Core.Savers
{
    public interface ISaverFactory
    {
        ISaver Create();
    }
}

namespace TC.Core.Parser
{
    public interface IParser
    {
        Task ParseAsync();
    }
}
TC.DataSaver

namespace TC.DataSaver.Abstraction
{
    internal interface IFileSaver
    {
        Task<string> SaveFileFromUri(Uri uri);
    }
}

```



```

    }
}

namespace TC.DataSaver.Abstraction.SaveChain
{
    internal interface ISaveChainBuilder
    {
        IChainComponent Build();
    }
}

namespace TC.DataSaver.Abstraction.SaveChain
{
    internal interface IComponentMessage
    {
    }
}

namespace TC.DataSaver.Abstraction.SaveChain
{
    internal abstract class AbstractChainComponent : IChainComponent
    {
        protected readonly IChainComponent next;
        protected AbstractChainComponent(IChainComponent next)
        {
            this.next = next;
        }

        public abstract Task RunAsync(IComponentMessage message);
    }
}

namespace TC.DataSaver.Savers.SaveChain
{
    internal class ModelPackComponentMessage : IComponentMessage
    {
        public ModelPack ModelPack { get; set; }
        public List<int> BookIds { get; set; } = new List<int>();
        public int? OriginBookIds { get; set; }
        public List<int> OriginAuthorsIds { get; set; } = new List<int>();
        public List<int> TranslatorIds { get; set; } = new List<int>();
    }
}

namespace TC.DataSaver.Savers
{
    internal class FileSaver : IFileSaver
    {
        private const string FILE_DIRECTORY = @"Content\Files\Img\";
        private IHttpClientFactory httpClientFactory;

        public FileSaver(IHttpClientFactory httpClientFactory)
        {
            this.httpClientFactory = httpClientFactory;
        }

        public async Task<string> SaveFileFromUri(Uri uri)
        {
            var host = uri.Host;
            var scheme = uri.Scheme;
            var client = this.httpClientFactory.CreateClient();
            client.BaseAddress = new Uri(scheme + "://" + host);
            var filePath = GenereteFilePath();

            using (var response = await client.GetAsync(uri))
            {
                if (response.IsSuccessStatusCode)
                {
                    var content = response.Content;
                    var mimeType = content.Headers.ContentType.MediaType;
                }
            }
        }
    }
}

```

```

        var extension = GetDefaultExtension(mimeType);
        filePath += extension;
        if (string.IsNullOrEmpty(extension)) throw new
ContentTypeNotFoundException();

        using(var source = await response.Content.ReadAsStreamAsync())
        {
            using(var target = File.OpenWrite(filePath))
            {
                await source.CopyToAsync(target);
            }
        }
    }
    else
    {
        throw new BadResponseException(uri.ToString());
    }
}
return filePath;
}

private static object locker = new object();
private string GetAndCheckFolder()
{
    var currentDirectory = Environment.CurrentDirectory;
    string imgDirectory = currentDirectory + FILE_DIRECTORY;
    if (!Directory.Exists(imgDirectory))
    {
        lock (locker)
        {
            if (!Directory.Exists(imgDirectory))
            {
                Directory.CreateDirectory(imgDirectory);
            }
        }
    }

    return imgDirectory;
}

private string GenereteFilePath()
{
    var folderPath = GetAndCheckFolder();
    lock (locker)
    {
        var guid = Guid.NewGuid();
        while (File.Exists(folderPath + guid))
        {
            guid = Guid.NewGuid();
        }
        return folderPath + guid;
    }
}

public static string GetDefaultExtension(string mimeType)
{
    string result;
    RegistryKey key;
    object value;

    key = Registry.ClassesRoot.OpenSubKey(@"MIME\Database\Content Type\" + mimeType,
false);
    value = key != null ? key.GetValue("Extension", null) : null;
    result = value != null ? value.ToString() : string.Empty;

    return result;
}

```

```

    }
}
internal class ContentTypeNotFoundException : Exception
{
    public ContentTypeNotFoundException()
    {
    }

    public ContentTypeNotFoundException(string message) : base(message)
    {
    }

    public ContentTypeNotFoundException(string message, Exception innerException) :
base(message, innerException)
    {
    }

    protected ContentTypeNotFoundException(SerializationInfo info, StreamingContext
context) : base(info, context)
    {
    }
}
internal class BadResponseException : Exception
{
    public BadResponseException(string message) : base(message)
    {
    }

    public BadResponseException(string message, Exception innerException) :
base(message, innerException)
    {
    }

    public BadResponseException()
    {
    }

    protected BadResponseException(SerializationInfo info, StreamingContext context) :
base(info, context)
    {
    }
}
}
namespace TC.DataSaver.Savers.SaveChain.ChainComponents
{
    internal abstract class MainChainComponent : AbstractChainComponent
    {
        protected readonly ILogger logger;

        protected MainChainComponent(IChainComponent next, ILogger logger)
            :base(next)
        {
            this.logger = logger;
        }

        public override async Task RunAsync(IComponentMessage message)
        {
            logger.LogDebug("Start save at {ChainComponent}", this.GetType().Name);
            ModelPackComponentMessage modelPackMessage = (ModelPackComponentMessage)message;

            await OperateModelPack(modelPackMessage);

            logger.LogDebug("Finish save at {ChainComponent}", this.GetType().Name);

            if (this.next != null)

```

```

        {
            logger.LogDebug("Call next '{NextChainComponent}' at {ChainComponent}",
this.next.GetType().Name, this.GetType().Name);
            await this.next.RunAsync(modelPackMessage);
        }
    }

    protected abstract Task OperateModelPack(ModelPackComponentMessage
modelPackMessage);
}

namespace TC.DataSaver.Savers.SaveChain.ChainComponents
{
    internal class AuthorBooksComponent : MainChainComponent
    {
        private readonly IRepoFacade repos;

        public AuthorBooksComponent(IRepoFacade repos,
[KeyFilter(nameof(AuthorBooksComponent))]IChainComponent next, ILogger<AuthorBooksComponent>
logger)
            : base(next, logger)
        {
            this.repos = repos;
        }

        protected override async Task OperateModelPack(ModelPackComponentMessage
modelPackMessage)
        {
            var origniAuthors = modelPackMessage.OriginAuthorsIds;
            var originBooks = new List<int>();
            if (modelPackMessage.OriginBookIds.HasValue)
            {
                originBooks.Add(modelPackMessage.OriginBookIds.Value);
            }

            await SaveAuthorBooks(origniAuthors, originBooks);

            var translatorAuthors = modelPackMessage.TranslatorIds;
            var books = modelPackMessage.BookIds;

            await SaveAuthorBooks(translatorAuthors, books);
        }

        private async Task SaveAuthorBooks(IEnumerable<int> authorIds, IEnumerable<int>
bookIds)
        {
            var authorBooks = authorIds.SelectMany(a => bookIds.Select(b =>
ToAuthorBookEntity(a, b)));

            var isExists = await authorBooks.Select(ab =>
this.repos.AuthorBookRepo.IsExistAsync(ab)).RunSequentially().ToListAsync();
            var notExists = authorBooks.Where((ab, i) => !isExists[i].HasValue).ToList();

            await this.repos.AuthorBookRepo.AddRangeAsync(notExists);
            await this.repos.AuthorBookRepo.SaveChangesAsync();
        }

        private Entities.AuthorBook ToAuthorBookEntity(int authorId, int bookId)
        {
            return new Entities.AuthorBook
            {
                AuthorId = authorId,
                BookId = bookId
            }
        }
    }
}

```

```

        };
    }
}
namespace TC.DataSaver.Savers.SaveChain.ChainComponents
{
    internal class AuthorComponent : MainChainComponent
    {
        private readonly IRepoFacade repos;

        public AuthorComponent(IRepoFacade repos,
[KeyFilter(nameof(AuthorComponent))]IChainComponent next, ILogger<AuthorComponent> logger)
            :base(next, logger)
        {
            this.repos = repos;
        }

        protected override async Task OperateModelPack(ModelPackComponentMessage
modelPackMessage)
        {
            var list =
modelPackMessage.ModelPack.GetModelOrDefault(typeof(Core.Models.Author));
            if (list == null) return;

            var authors = list.Cast<Core.Models.Author>();

            var originAuthorIds = await SaveOriginAuthors(authors);
            var transaltorAuthorIds = await SaveTranslatorAuthors(authors);

            modelPackMessage.OriginAuthorsIds = originAuthorIds.ToList();
            modelPackMessage.TranslatorIds = transaltorAuthorIds.ToList();
        }

        private async Task<IEnumerable<int>>
SaveTranslatorAuthors(IEnumerable<Core.Models.Author> authors)
        {
            var translators = authors.Where(a => a.IsTranslator);
            var translatorNamesIds = await SaveNamesAsync(translators);
            var ids = await SaveAuthorsAsync(translatorNamesIds,
Entities.AuthorTypeEnum.Translator);

            return ids;
        }

        private async Task<IEnumerable<int>>
SaveOriginAuthors(IEnumerable<Core.Models.Author> authors)
        {
            var originAuthors = authors.Where(a => !a.IsTranslator);
            var originNamesIds = await SaveNamesAsync(originAuthors);
            var ids = await SaveAuthorsAsync(originNamesIds,
Entities.AuthorTypeEnum.Original);

            return ids;
        }

        private async Task<IEnumerable<int>> SaveNamesAsync(IEnumerable<Core.Models.Author>
authors)
        {
            var names = authors.Select(a => ToAuthorNameEntity((Core.Models.Author)a));

            var isExists = await names.Select(ab =>
this.repos.AuthorNameRepo.IsExistAsync(ab)).RunSequentially().ToListAsync();

            var notExists = names.Where((n, i) => !isExists[i].HasValue).ToList();

```

```

        await this.repos.AuthorNameRepo.AddRangeAsync(notExists);
        await this.repos.AuthorNameRepo.SaveChangesAsync();

        var existsIds = isExists.Where(i => i.HasValue).Select(i => i.Value);
        return notExists.Select(n => n.Id).Concat(existsIds);
    }

    private async Task<IEnumerable<int>> SaveAuthorsAsync(IEnumerable<int> namesId,
Entities.AuthorTypeEnum authorType)
    {
        var authors = namesId.Select(n => ToAuthorEntity(n, authorType));

        var isExists = await authors.Select(a =>
this.repos.AuthorRepo.IsExistAsync(a)).RunSequentially().ToListAsync();

        var notExists = authors.Where((a, i) => !isExists[i].HasValue).ToList();

        await this.repos.AuthorRepo.AddRangeAsync(notExists);
        await this.repos.AuthorRepo.SaveChangesAsync();

        var existsIds = isExists.Where(i => i.HasValue).Select(i => i.Value);
        return notExists.Select(n => n.Id).Concat(existsIds);
    }

    private Entities.Author ToAuthorEntity(int authorNameId, Entities.AuthorTypeEnum
authorType)
    {
        return new Entities.Author
        {
            AuhtorNameId = authorNameId,
            AuthorTypeId = (int)authorType
        };
    }

    private Entities.AuthorName ToAuthorNameEntity(Core.Models.Author author)
    {
        var entityAuthor = new Entities.AuthorName
        {
            FirstName = author.FirstName,
            MiddleName = author.MiddleName,
            LastName = author.LastName,
            FullName = author.FullName
        };
        return entityAuthor;
    }
}

namespace TC.DataSaver.Savers.SaveChain.ChainComponents
{
    internal class BookComponent : MainChainComponent
    {
        private readonly IRepoFacade repos;

        public BookComponent(IRepoFacade repos,
[KeyFilter(nameof(BookComponent))]IChainComponent next, ILogger<BookComponent> logger)
            : base(next, logger)
        {
            this.repos = repos;
        }

        protected override async Task OperateModelPack(ModelPackComponentMessage
modelPackMessage)
        {
            var books =
modelPackMessage.ModelPack.GetModelOrDefault(typeof(Core.Models.Book));

```

```

        var originBook = books.FirstOrDefault(b =>
((Core.Models.Book)b).IsOriginalBook);
        int? originBookId = null;
        if (originBook != null)
        {
            originBookId = await SaveOriginBook((Core.Models.Book)originBook);
            modelPackMessage.OriginBookIds = originBookId;
        }

        var ids = await books.Where(b => b != originBook)
            .Select(b => SaveTranslatedBook((Core.Models.Book)b,
originBookId))
            .RunSequentially()
            .ToListAsync();
        modelPackMessage.BookIds = ids.ToList();
    }

    private async Task<int> SaveTranslatedBook(Core.Models.Book book, int? originBookId)
    {
        var dbBook = await ModelBookToEntity(book, isTranslated: true);
        dbBook.OriginBookId = originBookId;
        return await SaveBookToDb(dbBook);
    }

    private async Task<int> SaveOriginBook(Core.Models.Book book)
    {
        var dbBook = await ModelBookToEntity(book, isTranslated: false);
        return await SaveBookToDb(dbBook);
    }

    private async Task<Entities.Book> ModelBookToEntity(Core.Models.Book book, bool
isTranslated)
    {
        int? languageId = null;
        if (!string.IsNullOrEmpty(book.Language))
        {
            languageId = await
this.repos.LanguageRepo.GetLanguageIdOrEmptyAsync(book.Language);
        }
        var dbBook = new Entities.Book()
        {
            LanguageId = languageId,
            Description = book.Description,
            ISBN = book.ISBN,
            Link = book.Link,
            Title = book.Title,
            IsTranslated = isTranslated
        };
        return dbBook;
    }

    private async Task<int> SaveBookToDb(Entities.Book book)
    {
        var isExist = await this.repos.BookRepo.IsExistAsync(book);
        if (isExist.HasValue)
        {
            return isExist.Value;
        }

        await this.repos.BookRepo.AddAsync(book);
        await this.repos.BookRepo.SaveChangesAsync();
        return book.Id;
    }
}

namespace TC.DataSaver.Savers.SaveChain.ChainComponents
{

```

```

internal class BookCoverComponent : MainChainComponent
{
    private readonly IInternalRepoFacade repos;
    private readonly IFileSaver fileSaver;

    private IComponentMessage currentMessage;

    public BookCoverComponent(IInternalRepoFacade repos,
        IFileSaver fileSaver,
        [KeyFilter(nameof(BookCoverComponent))]IChainComponent next,
        ILogger<BookCoverComponent> logger)
        :base(next,logger)
    {
        this.repos = repos;
        this.fileSaver = fileSaver;
    }

    protected override async Task OperateModelPack(ModelPackComponentMessage
modelPackMessage)
    {
        this.currentMessage = modelPackMessage;

        var list =
modelPackMessage.ModelPack.GetModelOrDefault(typeof(Core.Models.BookCover));
        if (list == null) return;
        var bookCovers = list.Cast<Core.Models.BookCover>();

        //string filePath = null;
        var links = bookCovers.SelectMany(b => b.Links);
        var books = modelPackMessage.BookIds;

        await SaveBookCover(links, books);
    }
    private async Task<LinkFilePathInfo> SaveLink(string link)
    {
        LinkFilePathInfo info = new LinkFilePathInfo { Link = link};
        string filepath = null;
        try
        {
            filepath = await fileSaver.SaveFileFromUri(new Uri(link));
        }
        catch (Exception ex)
        {
            string error = string.Join(Environment.NewLine, ex.FromHierarchy(e =>
e.InnerException).Select(e => e.Message));
            await this.repos.NotSavedRepo.AddComponentMessageAsync(currentMessage,
error);
        }
        info.FilePath = filepath;
        return info;
    }
    private async Task SaveBookCover(IEnumerable<string> links, IEnumerable<int>
bookIds)
    {
        var combinedInfos = await links.Select(l =>
SaveLink(l)).RunSequentially().ToListAsync();
        combinedInfos =
combinedInfos.Where(f=>!string.IsNullOrEmpty(f.FilePath)).ToList();

        var fileIds = await SaveFiles(combinedInfos);
        var bookCovers = fileIds.SelectMany(f => bookIds.Select(b =>
ToBookCoverEntity(b, f))).ToList();
    }
}

```



```

        var isExists = await bookCovers.Select(bc =>
this.repos.BookCoverRepo.IsExistAsync(bc)).RunSequentially().ToListAsync();

        var notExists = bookCovers.Where((bc, i) => !isExists[i].HasValue).ToList();

        await this.repos.BookCoverRepo.AddRangeAsync(notExists);
        await this.repos.BookCoverRepo.SaveChangesAsync();
    }

    private async Task<IEnumerable<int>> SaveFiles(IEnumerable<LinkFilePathInfo>
filePatheAndLinks)
    {
        var files = filePatheAndLinks.Select(f =>
ToFileEntity(f.FilePath, f.Link)).ToList();

        var isExists = await files.Select(f =>
this.repos.FileRepo.IsExistAsync(f)).RunSequentially().ToListAsync();

        var notExists = files.Where((f, i) => !isExists[i].HasValue).ToList();

        await this.repos.FileRepo.AddRangeAsync(notExists);
        await this.repos.FileRepo.SaveChangesAsync();

        var existsIds = isExists.Where(i => i.HasValue).Select(i => i.Value).ToList();
        return notExists.Select(n => n.Id).Concat(existsIds).ToList();
    }

    private Entities.File ToFileEntity(string filePath, string link)
    {
        string filename = filePath.Split(new string[] { "/", "\\" },
StringSplitOptions.RemoveEmptyEntries).Last();
        var parts = filename.Split('.');
        var title = parts.Length > 0 ? parts[0] : string.Empty;
        var extension = parts.Length > 1 ? parts[1] : string.Empty;
        return new Entities.File
        {
            Extension = extension,
            Path = filePath,
            Title = title,
            Link = link
        };
    }

    private Entities.BookCover ToBookCoverEntity(int bookId, int fileId)
    {
        return new Entities.BookCover
        {
            BookId = bookId,
            FileId = fileId
        };
    }

    private class LinkFilePathInfo
    {
        public string Link { get; set; }
        public string FilePath { get; set; }
    }
}

namespace TC.DataSaver.Savers.SaveChain.ChainComponents
{
    internal class PublisherComponent : MainChainComponent
    {
        private readonly IRepoFacade repos;
    }
}

```

```

        public PublisherComponent(IRepoFacade repos,
[KeyFilter(nameof(PublisherComponent))]IChainComponent next, ILogger<PublisherComponent>
logger)
        {
            :base(next, logger)
            {
                this.repos = repos;
            }

            protected override async Task OperateModelPack(ModelPackComponentMessage
modelPackMessage)
            {
                var list =
modelPackMessage.ModelPack.GetModelOrDefault(typeof(Core.Models.Publisher));
                if (list == null) return;
                var publishers = list.Cast<Core.Models.Publisher>();

                var publisherIds = await publishers.Select(p =>
this.repos.PublisherRepo.GetPublisherOrCreate(p.Name))
                    .RunSequentially()
                    .ToListAsync();

                var publishDates = publishers.Where(p => p.Date.HasValue).Select(p =>
p.Date.Value);
                var books = modelPackMessage.BookIds;
                await SavePublishDates(publisherIds, publishDates, books);
            }

            private async Task SavePublishDates(IEnumerable<int> publisherIds,
IEnumerable<DateTime> publishDate, IEnumerable<int> bookIds)
            {
                var publishDates = bookIds.SelectMany(b => publisherIds.SelectMany(p =>
publishDate.Select(pd => ToEntity(p, pd, b))));

                var isExists = await publishDates.Select(pd =>
this.repos.PublishDateRepo.IsExistAsync(pd))
                    .RunSequentially()
                    .ToListAsync();

                var notExists = publishDates.Where((pd, i) => !isExists[i].HasValue).ToList();

                await this.repos.PublishDateRepo.AddRangeAsync(notExists);
                await this.repos.PublishDateRepo.SaveChangesAsync();
            }

            private Entities.PublishDate ToEntity(int publisherId, DateTime publishDate, int
bookId)
            {
                return new Entities.PublishDate
                {
                    PublisherId = publisherId,
                    Date = publishDate,
                    BookId = bookId
                };
            }
        }
    }
}
namespace TC.DataSaver.Savers.SaveChain
{
    internal class ChainBuilder : ISaveChainBuilder
    {
        private readonly IChainComponent first;

        public ChainBuilder([KeyFilter(ChainConst.FIRST_KEY)]IChainComponent chainComponent)
        {

```

```

        this.first = chainComponent;
    }

    public IChainComponent Build()
    => first;
}
internal class ChainModule: Autofac.Module
{
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterChainComponent()
            .AddChainComponent<BookComponent>()
            .AddChainComponent<AuthorComponent>()
            .AddChainComponent<AuthorBooksComponent>()
            .AddChainComponent<BookCoverComponent>()
            .AddChainComponent<PublisherComponent>()
            .Configure();
    }
}
internal static class ChainRegistratorExtension
{
    public static ChainRegistrator RegisterChainComponent(this ContainerBuilder builder)
    {
        builder.RegisterType<ChainBuilder>().As<ISaveChainBuilder>().WithAttributeFiltering();
        return new ChainRegistrator(builder);
    }
}
internal class ChainRegistrator
{
    private ContainerBuilder builder;
    private List<Type> chainComponentTypes;

    public ChainRegistrator(ContainerBuilder builder)
    {
        this.builder = builder;
        chainComponentTypes = new List<Type>();
    }

    public ChainRegistrator AddChainComponent<TChainComponent>() where TChainComponent :
IChainComponent
    {
        this.chainComponentTypes.Add(typeof(TChainComponent));
        return this;
    }

    private void Registrater(Type current, Type next)
    {
        builder.RegisterType(next)
            .As<IChainComponent>()
            .Keyed<IChainComponent>(current.Name)
            .WithAttributeFiltering();
    }
    private void RegistraterFirst(Type first)
    {
        builder.RegisterType(first)
            .As<IChainComponent>()
            .Keyed<IChainComponent>(ChainConst.FIRST_KEY)
            .WithAttributeFiltering();
    }

    public ContainerBuilder Configure()
    {

```

```

        var current = chainComponentTypes.FirstOrDefault() ?? throw new Exception("Add
chain component");
        RegisterFirst(current);

        for(int i = 1; i < chainComponentTypes.Count; i++)
        {
            var next = chainComponentTypes[i];
            Register(current, next);
            current = next;
        }
        Register(current, typeof(LastChainComponent));

        return builder;
    }
}
internal static class ChainConst
{
    public const string FIRST_KEY = "first chain component";
}
internal class LastChainComponent : IChainComponent
{
    public void Run(IComponentMessage message)
    {
    }

    public Task RunAsync(IComponentMessage message)
    {
        return Task.CompletedTask;
    }
}
}
namespace TC.DataSaver.Savers
{
    class DbSaver : IDbSaver
    {
        private readonly ISaveChainBuilder chainBuilder;
        private readonly IInternalRepoFacade repos;
        private readonly ILogger logger;

        public DbSaver(ISaveChainBuilder chainBuilder, IInternalRepoFacade repoFacade,
ILogger<DbSaver> logger)
        {
            this.chainBuilder = chainBuilder;
            this.repos = repoFacade;
            this.logger = logger;
        }

        public async Task SaveAsync(ModelPack modelPack)
        {
            logger.LogDebug("Start save model pack");
            var messageComponent = new ModelPackComponentMessage
            {
                ModelPack = modelPack
            };
            try
            {
                await chainBuilder.Build().RunAsync(messageComponent);
            }
            catch (Exception ex)
            {
                logger.LogError(ex, "While save model pack");
                string error = string.Join(Environment.NewLine,
ex.FromHierarchy(e => e.InnerException).Select(e => e.Message));
                await
this.repos.NotSavedRepo.AddComponentMessageAsync(messageComponent, error);
            }
        }
    }
}

```

```

    }
}
namespace TC.DataSaver.Savers
{
    class JsonSaver : ISaver
    {
        private readonly GeneralDbContext dbContext;

        public JsonSaver(GeneralDbContext dbContext)
        {
            this.dbContext = dbContext;
        }

        public async Task SaveAsync(ModelPack modelPack)
        {
            var byLink = await this.dbContext.IntermediateModelPacks.AsQueryable()
                                                                    .Where(i =>
!string.IsNullOrEmpty(i.Link))
                                                                    .FirstOrDefaultAsync(i
=> string.Equals(i.Link, modelPack.Link));
            if (byLink == null)
            {
                string json = ModelPackSerializer.ToJson(modelPack);

                var entity = new IntermediateModelPack { Content = json, Created =
DateTime.Now, StateId = (int)ModelPackStateEnum.InProgress, Link = modelPack.Link };

                await this.dbContext.IntermediateModelPacks.AddAsync(entity);
                await this.dbContext.SaveChangesAsync();
            }
        }
    }
}
namespace TC.DataSaver
{
    public class GeneralDbContext : DbContext
    {
        public GeneralDbContext(DbContextOptions<GeneralDbContext> options)
            : base(options)
        {
        }
        #region DbSets
        public DbSet<Book> Books { get; set; }
        public DbSet<Author> Authors { get; set; }
        public DbSet<AuthorName> AuthorNames { get; set; }
        public DbSet<AuthorType> AuthorTypes { get; set; }
        public DbSet<Language> Languages { get; set; }
        public DbSet<PairOfSameName> PairsOfSameName { get; set; }
        public DbSet<PublishDate> PublishDates { get; set; }
        public DbSet<Publisher> Publishers { get; set; }
        public DbSet<File> Files { get; set; }
        public DbSet<BookCover> BookCovers { get; set; }
        public DbSet<NotSaved> NotSaveds { get; set; }
        public DbSet<IntermediateModelPack> IntermediateModelPacks { get; set; }
        public DbSet<ModelPackState> ModelPackStates { get; set; }
        #endregion
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Book>()
                .HasOne(b => b.Language)
                .WithMany(l => l.Books)
                .HasForeignKey(b => b.LanguageId);
            modelBuilder.Entity<Book>()
                .HasOne(b => b.OriginBook)

```

```

        .WithMany(o => o.TranslatedBook)
        .HasForeignKey(b => b.OriginBookId);

modelBuilder.Entity<Author>()
    .HasOne(a => a.AuthorName)
    .WithMany(an => an.Authors)
    .HasForeignKey(a => a.AuthorNameId);
modelBuilder.Entity<Author>()
    .HasOne(a => a.AuthorType)
    .WithMany(at => at.Authors)
    .HasForeignKey(at => at.AuthorTypeId);

modelBuilder.Entity<AuthorBook>()
    .HasOne(ab => ab.Author)
    .WithMany(a => a.AuthorBooks)
    .HasForeignKey(ab => ab.AuthorId);
modelBuilder.Entity<AuthorBook>()
    .HasOne(ab => ab.Book)
    .WithMany(b => b.AuthorBooks)
    .HasForeignKey(ab => ab.BookId);

modelBuilder.Entity<BookCover>()
    .HasOne(bc => bc.Book)
    .WithMany(b => b.BookCovers)
    .HasForeignKey(bc => bc.BookId);
modelBuilder.Entity<BookCover>()
    .HasOne(bc => bc.File)
    .WithMany(f => f.BookCovers)
    .HasForeignKey(bc => bc.FileId);

modelBuilder.Entity<PublishDate>()
    .HasOne(pd => pd.Book)
    .WithMany(b => b.PublishDates)
    .HasForeignKey(pd => pd.BookId);
modelBuilder.Entity<PublishDate>()
    .HasOne(pd => pd.Publisher)
    .WithMany(p => p.PublishDates)
    .HasForeignKey(pd => pd.PublisherId);

modelBuilder.Entity<PairOfSameName>()
    .HasKey(p => new { p.FirstNameId, p.SecondNameId });

modelBuilder.Entity<PairOfSameName>()
    .HasOne(p => p.FirstName)
    .WithMany(an => an.SameNames);

modelBuilder.Entity<PairOfSameName>()
    .HasOne(p => p.SecondName);

FillDb(modelBuilder);
AddLanguages(modelBuilder);
AddStates(modelBuilder);
}
private void FillDb(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<AuthorType>().HasData(
        new AuthorType { Id = 1, Code = "originalAuthor", Description = "Автор" },
        new AuthorType { Id = 2, Code = "translator", Description = "Перекладач" }
    );
}
private void AddLanguages(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Language>().HasData(

```

```

        new Language { Id = 1, Code = "uk-UA", Description = "Українська" },
        new Language { Id = 2, Code = "ru-RU", Description = "Російська" },
        new Language { Id = 3, Code = "en-US", Description = "Англійська" }
    );
}

private void AddStates(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<ModelPackState>().HasData(
        new ModelPackState { Id = 1, Code = "InProgress", Description = "Model pack
wait while administrator managed it" },
        new ModelPackState { Id = 2, Code = "Applied", Description = "Can be stored"
    },
        new ModelPackState { Id = 3, Code = "Rejected", Description = "Incorrect
data" },
        new ModelPackState { Id = 4, Code = "Saved", Description = "Saved to db" }
    );
}
}
}

```

TC.BackgroundService

```

namespace TC.BackgorundService
{
    public class BackgroundParser : BS
    {
        private readonly IEnumerable<IParser> parsers;

        public BackgroundParser(IEnumerable<IParser> parsers)
        {
            this.parsers = parsers;
        }

        protected override async Task ExecuteAsync(Cancellation_token stoppingToken)
        {
            foreach (var p in parsers)
            {
                await p.ParseAsync();
            }
        }
    }
}

namespace TC.BackgorundService
{
    public class BackgroundSaver : BS
    {
        private const double PERIOD_IN_MILLISECONS = 1000 * 60 * 60;
        private Timer timer;
        private readonly IServiceProvider serviceProvider;

        public BackgroundSaver(IServiceProvider serviceProvider)
        {
            this.serviceProvider = serviceProvider;
        }

        protected override Task ExecuteAsync(Cancellation_token stoppingToken)
        {
            timer = new Timer(
                async (obj) => await DoWork(stoppingToken),
                null,
                TimeSpan.Zero,
                TimeSpan.FromMilliseconds(PERIOD_IN_MILLISECONS));
            return Task.CompletedTask;
        }
    }
}

```

```

private async Task DoWork(CancellationTokentoken cancellationToken)
{
    using(var scope = this.serviceProvider.CreateScope())
    {
        SaveJob job = scope.ServiceProvider.GetRequiredService<SaveJob>();
        await job.Save(cancellationToken);
    }
}

public override Task StopAsync(CancellationTokentoken cancellationToken)
{
    this.timer?.Change(Timeout.Infinite, 0);
    return base.StopAsync(cancellationToken);
}
public override void Dispose()
{
    base.Dispose();
    this.timer.Dispose();
}
}
}
namespace TC.BackgorundService
{
    class SaveJob
    {
        private readonly IDbSaver dbSaver;
        private readonly GeneralDbContext dbContext;
        private readonly ILogger<SaveJob> logger;

        public SaveJob(IDbSaver dbSaver, GeneralDbContext dbContext, ILogger<SaveJob>
logger)
        {
            this.dbSaver = dbSaver;
            this.dbContext = dbContext;
            this.logger = logger;
        }

        public async Task Save(CancellationTokentoken cancellationToken)
        {
            var modelPacks = await dbContext.IntermediateModelPacks.AsQueryable()
                                                                    .Where(i => i.StateId ==
(int)ModelPackStateEnum.Applied)
                                                                    .ToListAsync();

            foreach(var m in modelPacks)
            {
                if (cancellationToken.IsCancellationRequested)
                {
                    break;
                }
                var isSuccessSaved = await SaveModelPack(m);
                if (isSuccessSaved)
                {
                    m.StateId = (int)ModelPackStateEnum.Saved;
                    await this.dbContext.SaveChangesAsync();
                }
            }
        }

        private async Task<bool> SaveModelPack(IntermediateModelPack jsonModelPack)
        {
            if (jsonModelPack == null) return false;
            var modelPack = ModelPackSerializer.ToModelPack(jsonModelPack.Content);

```



```

        bool isSuccess = true;
        try
        {
            await this.dbSaver.SaveAsync(modelPack);
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "While saving json model pack " + jsonModelPack.Id);
            isSuccess = false;
        }
        return isSuccess;
    }
}
}
namespace TC.BackgorundService
{
    public class SaverFactory : ISaverFactory
    {
        private readonly IServiceScopeFactory serviceScopeFactory;

        public SaverFactory(IServiceScopeFactory serviceScopeFactory)
        {
            this.serviceScopeFactory = serviceScopeFactory;
        }

        public ISaver Create()
        {
            var scope = serviceScopeFactory.CreateScope();
            return scope.ServiceProvider.GetRequiredService<ISaver>();
        }
    }
}

```

TC.Authorization

```

namespace TC.Authorization
{
    public class AdminAccount
    {
        public int Id { get; set; }
        public string Login { get; set; }
        public string PasswordHash { get; set; }
    }
}
namespace TC.Authorization
{
    public class AuthorizationDbContext : DbContext
    {
        public DbSet<AdminAccount> Users { get; set; }
        public AuthorizationDbContext(DbContextOptions<AuthorizationDbContext> options)
            : base(options)
        {
        }
    }
}
namespace TC.Authorization
{
    public static class ClaimsExtension
    {
        public static ClaimsPrincipal ToClaim(this AdminAccount adminAccount)
        {
            bool isAuthethenticated = false;

```

```

        List<Claim> claims = new List<Claim>();
        if (adminAccount.Id > 0)
        {
            claims.Add(new Claim(ClaimTypes.NameIdentifier,
adminAccount.Id.ToString()));
            isAuthenticated = true;
        }
        if (!string.IsNullOrEmpty(adminAccount.Login))
        {
            claims.Add(new Claim(ClaimTypes.Name, adminAccount.Login));
            isAuthenticated = true;
        }

        ClaimsIdentity identity;
        if (isAuthenticated)
        {
            identity = new ClaimsIdentity(claims, "ApplicationCookie", ClaimTypes.Name,
ClaimsIdentity.DefaultRoleClaimType);
        }
        else
        {
            identity = new ClaimsIdentity();
        }

        ClaimsPrincipal principal = new ClaimsPrincipal(identity);
        return principal;
    }

    public static AdminAccount ToAccount(this ClaimsPrincipal claimsPrincipal)
    {
        AdminAccount account = new AdminAccount();
        var claims = claimsPrincipal.Claims;
        var claimId = claims.Where(c => c.Type ==
ClaimTypes.NameIdentifier).FirstOrDefault();
        if (claimId != null)
        {
            var id = claimId.Value;
            int idVal = int.Parse(id);
            account.Id = idVal;
        }
        var claimName = claims.FirstOrDefault(c => c.Type == ClaimTypes.Name);
        if (claimName != null)
        {
            account.Login = claimName.Value;
        }
        return account;
    }
}
namespace TC.Authorization
{
    public interface IAuthorizeService
    {
        Task<ResultDto> SignIn(string login, string password);

        Task SignOut();
    }

    class AuthorizeService : IAuthorizeService
    {
        private readonly AuthorizationDbContext dbContext;
        private readonly SignInManager<AdminAccount> signInManager;
        private readonly ILogger<AuthorizeService> logger;

        public AuthorizeService(AuthorizationDbContext dbContext,
SignInManager<AdminAccount> signInManager, ILogger<AuthorizeService> logger)

```

```

{
    this.dbContext = dbContext;
    this.signInManager = signInManager;
    this.logger = logger;
}

public async Task<ResultDto> SignIn(string login, string password)
{
    var result = new ResultDto { IsSuccess = false };

    if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password))
        return result;

    var user = await GetUserByLogin(login);
    if (user == null) return result;

    var signInResult = await signInManager.CheckPasswordSignInAsync(user, password,
lockoutOnFailure: false);
    if (signInResult.Succeeded)
    {
        var claims = user.ToClaim();
        await SignOut();
        try
        {
            await this.signInManager.SignInAsync(user, isPersistent: true);
        }
        catch (Exception ex)
        {
            this.logger.LogError(ex, "While try to add cookie");
        }
        result.IsSuccess = true;
    }

    return result;
}

public async Task SignOut()
{
    try
    {
        await this.signInManager.Context.SignOutAsync();
        await this.signInManager.SignOutAsync();
    }
    catch (Exception ex)
    {
        this.logger.LogError(ex, "While try to sign out");
    }
}

private async Task<AdminAccount> GetUserByLogin(string login)
{
    login = login.ToLower();
    return await this.dbContext.Users.FirstOrDefaultAsync(u =>
string.Equals(u.Login.ToLower(), login));
}
}
namespace TC.Authorization
{
    class InitDb
    {
        private const string MAIN_ADMIN_NAME = "admin";
        private const string MAIN_ADMIN_PASSWORD = "Qwe12345!";

        private readonly AuthorizationDbContext dbContext;
        private readonly UserManager<AdminAccount> userManager;
    }
}

```

```

        public InitDb(UserManager<AdminAccount> userManager, AuthorizationDbContext
dbContext)
        {
            this.userManager = userManager;
            this.dbContext = dbContext;
        }

        public async Task Initialize()
        {
            var admin = await dbContext.Users.FirstOrDefaultAsync(u =>
string.Equals(u.Login.ToLower(), MAIN_ADMIN_NAME));
            if (admin == null)
            {
                AdminAccount account = new AdminAccount
                {
                    Login = MAIN_ADMIN_NAME
                };
                await this.userManager.CreateAsync(account, MAIN_ADMIN_PASSWORD);
            }
        }
    }
}
namespace TC.Authorization
{
    public class ResultDto
    {
        public bool IsSuccess { get; set; }
    }
}

namespace TC.Authorization
{
    class UserStore : IUserStore<AdminAccount>, IUserPasswordStore<AdminAccount>,
IRoleStore<IdentityRole>
    {
        private readonly AuthorizationDbContext dbContext;
        private readonly ILogger<UserStore> logger;
        public UserStore(AuthorizationDbContext dbContext, ILogger<UserStore> logger)
        {
            this.dbContext = dbContext;
            this.logger = logger;
        }

        public async Task<IdentityResult> CreateAsync(AdminAccount user, CancellationToken
cancellation_token)
        {
            try
            {
                await this.dbContext.Users.AddAsync(user);
                await this.dbContext.SaveChangesAsync();
            }
            catch (Exception ex)
            {
                this.logger.LogError(ex, "While Create user");
                return IdentityResult.Failed();
            }
            return IdentityResult.Success;
        }

        public Task<IdentityResult> CreateAsync(IdentityRole role, CancellationToken
cancellation_token)
        {
            return Task.FromResult(IdentityResult.Success);
        }
    }
}

```

```

        public async Task<IdentityResult> DeleteAsync(AdminAccount user, CancellationToken
cancellationToken)
        {
            try
            {
                this.dbContext.Users.Remove(user);
                await this.dbContext.SaveChangesAsync();
            }
            catch (Exception ex)
            {
                this.logger.LogError(ex, "While deleting user");
                return IdentityResult.Failed();
            }
            return IdentityResult.Success;
        }

        public Task<IdentityResult> DeleteAsync(IdentityRole role, CancellationToken
cancellationToken)
        {
            return Task.FromResult(IdentityResult.Success);
        }

        public void Dispose()
        {
            this.dbContext.Dispose();
        }

        public async Task<AdminAccount> FindByIdAsync(string userId, CancellationToken
cancellationToken)
        {
            var id = int.Parse(userId);
            var user = await this.dbContext.Users.FirstAsync(i => i.Id == id);
            return user;
        }

        public async Task<AdminAccount> FindByNameAsync(string normalizedUserName,
CancellationToken cancellationToken)
        {
            return await this.dbContext.Users.FirstOrDefaultAsync(a =>
string.Equals(a.Login.ToLower(), normalizedUserName));
        }

        public Task<string> GetNormalizedRoleNameAsync(IdentityRole role, CancellationToken
cancellationToken)
        {
            return Task.FromResult(role.NormalizedName);
        }

        public Task<string> GetNormalizedUserNameAsync(AdminAccount user, CancellationToken
cancellationToken)
        {
            return Task.FromResult<string>(user?.Login?.ToLower());
        }

        public Task<string> GetPasswordHashAsync(AdminAccount user, CancellationToken
cancellationToken)
        {
            return Task.FromResult(user.PasswordHash);
        }

        public Task<string> GetRoleIdAsync(IdentityRole role, CancellationToken
cancellationToken)
        {
            return Task.FromResult(role.Id);
        }

```

```

        public Task<string> GetRoleNameAsync(IdentityRole role, CancellationToken
cancellationToken)
        {
            return Task.FromResult(role.Name);
        }

        public Task<string> GetUserIdAsync(AdminAccount user, CancellationToken
cancellationToken)
        {
            return Task.FromResult(user.Id.ToString());
        }

        public Task<string> GetUserNameAsync(AdminAccount user, CancellationToken
cancellationToken)
        {
            return Task.FromResult(user.Login);
        }

        public Task<bool> HasPasswordAsync(AdminAccount user, CancellationToken
cancellationToken)
        {
            return Task.FromResult(!string.IsNullOrEmpty(user.PasswordHash));
        }

        public Task SetNormalizedRoleNameAsync(IdentityRole role, string normalizedName,
CancellationToken cancellationToken)
        {
            role.NormalizedName = normalizedName;
            return Task.CompletedTask;
        }

        public Task SetNormalizedUserNameAsync(AdminAccount user, string normalizedName,
CancellationToken cancellationToken)
        {
            user.Login = normalizedName;
            return Task.CompletedTask;
        }

        public Task SetPasswordHashAsync(AdminAccount user, string passwordHash,
CancellationToken cancellationToken)
        {
            user.PasswordHash = passwordHash;
            return Task.CompletedTask;
        }

        public Task SetRoleNameAsync(IdentityRole role, string roleName, CancellationToken
cancellationToken)
        {
            role.Name = roleName;
            return Task.CompletedTask;
        }

        public Task SetUserNameAsync(AdminAccount user, string userName, CancellationToken
cancellationToken)
        {
            user.Login = userName;
            return Task.CompletedTask;
        }

        public async Task<IdentityResult> UpdateAsync(AdminAccount user, CancellationToken
cancellationToken)
        {
            this.dbContext.Attach(user);
            await this.dbContext.SaveChangesAsync();
            return IdentityResult.Success;
        }

```

```

        public Task<IdentityResult> UpdateAsync(IdentityRole role, CancellationToken
cancellationToken)
        {
            return Task.FromResult(IdentityResult.Success);
        }

        Task<IdentityRole> IRoleStore<IdentityRole>.FindByIdAsync(string roleId,
CancellationToken cancellationToken)
        {
            return Task.FromResult(new IdentityRole("admin"));
        }

        Task<IdentityRole> IRoleStore<IdentityRole>.FindByNameAsync(string
normalizedRoleName, CancellationToken cancellationToken)
        {
            return Task.FromResult(new IdentityRole("admin"));
        }
    }
}

```

TC.Parsers

```

namespace TC.Parsers.Abstractions
{
    internal interface IModelPackParser
    {
        Task<ModelPack> ParseAsync();
    }
}
namespace TC.Parsers.Abstractions
{
    internal interface IModelParser
    {
        Task<IEnumerable<IModel>> ParseAsync(Uri link);
    }
}
namespace TC.Parsers.Abstractions
{
    internal interface IPageParser
    {
        Task<IEnumerable<ItemInfo>> ParsePageAsync(PageInfo pageInfo);
    }
}
namespace TC.Parsers.Abstractions
{
    internal interface ISiteParser
    {
        Task<IEnumerable<PageInfo>> ParseSiteAsync(SiteInfo info);
    }
}
namespace TC.Parsers.Abstractions.Info
{
    internal interface IInfoFactory
    {
        ItemInfo CreateItem(string link);
        PageInfo CreatePage(string link);
    }
}
namespace TC.Parsers.Abstractions.Info
{
    internal abstract class ItemInfo : IModelPackParser
    {
        protected readonly HttpService httpService;
        protected readonly IEnumerable<IModelParser> modelParsers;
        protected readonly ILogger logger;
    }
}

```

```

        public string Link { get; set; }
        public ItemInfo(IServiceFactory serviceFactory, HttpService httpService, ILogger
logger)
        {
            this.httpService = httpService;
            this.modelParsers = serviceFactory.GetServices<IModelParser>();
            this.logger = logger;
        }
        public async Task<ModelPack> ParseAsync()
        {
            this.logger.LogDebug("Start parse item at {ItemInfo} for {Link}",
this.GetType().Name, Link);

            string pageHtml = await this.httpService.GetPageAsync(Link);
            ModelPack modelPack = new ModelPack();
            modelPack.Link = Link;
            var taskResult = await ParsePackAsync();

            this.logger.LogDebug("Finish parse item at {ItemInfo} for {Link}",
this.GetType().Name, Link);

            var models = taskResult.Where(t => t != null).ToList();
            modelPack.AddModels(models);
            return modelPack;
        }
        private async Task<IEnumerable<IModel>> ParsePackAsync()
        {
            var tasks = modelParsers.Select(m => m.ParseAsync(new Uri(Link)));
            var t = Task.WhenAll(tasks);
            var models = await t;
            return models.Where(m => m != null).SelectMany(m => m.Select(m2 => m2));
        }
    }
}
namespace TC.Parsers.Abstractions.Info
{
    internal abstract class PageInfo
    {
        protected readonly HttpService httpService;
        protected readonly ILogger logger;

        protected PageInfo(HttpService httpService, ILogger logger)
        {
            this.httpService = httpService;
            this.logger = logger;
        }

        public string Link { get; set; }
        public IDocument Document { get; protected set; }

        protected abstract IEnumerable<ItemInfo> ParseDocument();

        public async Task<IEnumerable<ItemInfo>> ParsePageAsync()
        {
            this.logger.LogDebug("Start parse Page at {PageInfo} for {Link}",
this.GetType().Name, Link);

            if (Document == null)
            {
                await CreateDocument();
            }

            this.logger.LogDebug("Finish parse Page at {PageInfo} for {Link}",
this.GetType().Name, Link);

```



```

        return ParseDocument();
    }

    public abstract bool IsPageCorrect();

    public async Task CreateDocument()
    {
        string pageHtml = await this.httpService.GetPageAsync(Link);

        var context = AngleSharpFactory.Create();
        var document = await context.OpenAsync(req => req.Content(pageHtml));
        this.Document = document;
    }
}

namespace TC.Parsers.Abstractions.Info
{
    internal abstract class SiteInfo
    {
        protected readonly IInfoFactory infoFactory;
        protected readonly ILogger logger;
        public string BaseAddress { get; set; }
        public SiteInfo(IServiceFactory serviceFactory, ILogger logger)
        {
            this.infoFactory = serviceFactory.GetService<IInfoFactory>();
            this.logger = logger;
        }

        public abstract string PageLink(int pageNum);

        public async IAsyncEnumerable<PageInfo> ParseSiteAsync()
        {
            this.logger.LogDebug("Start parse Site at {SiteInfo}", this.GetType().Name);
            int counter = 0;
            bool isValid = true;
            while (isValid)
            {
                counter++;
                var link = PageLink(counter);
                var page = infoFactory.CreatePage(link);

                this.logger.LogDebug("At {SiteInfo} create PageInfo for {Link}",
this.GetType().Name, link);

                await page.CreateDocument();
                isValid = page.IsPageCorrect();
                yield return page;
            }
            this.logger.LogDebug("Finish parse Site at {SiteInfo}", this.GetType().Name);
        }
    }
}

namespace TC.Parsers.DependencyInjection
{
    public interface ICustomDependencyResolver
    {
        TResolved Resolve<TResolved>();
    }

    public class CustomDependencyResolver : ICustomDependencyResolver
    {
        private readonly ILifetimeScope lifetimeScope;

        public CustomDependencyResolver(ILifetimeScope lifetimeScope)
        {
            this.lifetimeScope = lifetimeScope;
        }
    }
}

```

```

    }

    public TResolved Resolve<TResolved>()
        => lifetimeScope.Resolve<TResolved>();
}
namespace TC.Parsers.DependencyInjection
{
    public interface IServiceFactory
    {
        TService GetService<TService>() where TService : class;
        IEnumerable<TService> GetServices<TService>() where TService : class;
    }
    public class ServiceFactory : IServiceFactory
    {
        private readonly ICustomDependencyResolver resolver;
        private readonly string targetKey;
        private readonly string targetValue;

        public ServiceFactory(ICustomDependencyResolver resolver, string targetKey, string
targetValue)
        {
            this.resolver = resolver;
            this.targetKey = targetKey;
            this.targetValue = targetValue;
        }

        public TService GetService<TService>() where TService : class
        {
            var services = this.resolver.Resolve<IEnumerable<Meta<TService>>>();
            var service = services.FirstOrDefault(s =>
targetValue.Equals($"{s.Metadata[targetKey]}", StringComparison.OrdinalIgnoreCase)).Value;
            return service ?? throw new InvalidOperationException();
        }

        public IEnumerable<TService> GetServices<TService>() where TService : class
        {
            var services = this.resolver.Resolve<IEnumerable<Meta<TService>>>()
                .Where(s => targetValue.Equals($"{s.Metadata[targetKey]}",
StringComparison.OrdinalIgnoreCase))
                .Select(s => s.Value);
            return services;
        }
    }
}
namespace TC.Parsers.Realizations
{
    internal abstract class MainAbstractModelParser : IModelParser
    {
        protected readonly HttpService httpService;
        protected readonly ILogger logger;

        protected Uri link;
        protected IDocument document;

        public MainAbstractModelParser(HttpService httpService, ILogger logger)
        {
            this.httpService = httpService;
            this.logger = logger;
        }
        protected async Task GetDocumentAsync(Uri link)
        {
            this.link = link;
            var html = await httpService.GetPageAsync(link.AbsoluteUri);

```

```

        var context = AngleSharpFactory.Create();
        document = await context.OpenAsync(req => req.Content(html));
    }

    protected abstract IEnumerable<IModel> ParseDocument();

    public async Task<IEnumerable<IModel>> ParseAsync(Uri link)
    {
        logger.LogDebug("Start parse at {ModelParser} for a {Link}",
this.GetType().Name, link.AbsoluteUri);
        await GetDocumentAsync(link);
        logger.LogDebug("Finish parse at {ModelParser} for a {Link}",
this.GetType().Name, link.AbsoluteUri);
        return ParseDocument();
    }
}
}
namespace TC.Parsers.Realizations
{
    internal class MainInfoFactory : IInfoFactory
    {
        protected readonly IServiceFactory serviceFactory;

        public MainInfoFactory(IServiceFactory serviceFactory)
        {
            this.serviceFactory = serviceFactory;
        }

        public PageInfo CreatePage(string link)
        {
            var pageInfo = this.serviceFactory.GetService<PageInfo>();
            pageInfo.Link = link;
            return pageInfo;
        }

        public ItemInfo CreateItem(string link)
        {
            var itemInfo = this.serviceFactory.GetService<ItemInfo>();
            itemInfo.Link = link;
            return itemInfo;
        }
    }
}
namespace TC.Parsers.Realizations
{
    internal abstract class MainParser : IParser
    {
        protected readonly IServiceFactory serviceFactory;
        protected readonly ISaverFactory saverFactory;
        protected readonly HttpService httpService;
        protected readonly SiteInfo siteInfo;
        protected readonly ILogger logger;

        public MainParser(
            IServiceFactory serviceFactory,
            HttpService httpService,
            ISaverFactory saverFactory,
            ILogger logger)
        {
            this.serviceFactory = serviceFactory;
            this.httpService = httpService;
            this.siteInfo = this.serviceFactory.GetService<SiteInfo>();
            this.saverFactory = saverFactory;
            this.logger = logger;
        }
    }
}

```

```

public async Task ParseAsync()
{
    logger.LogDebug("Start parsing at {Parser}", this.GetType().Name);
    await foreach (var pageInfo in siteInfo.ParseSiteAsync())
    {
        var itemsInfo = await pageInfo.ParsePageAsync();
        foreach (var i in itemsInfo)
        {
            try
            {
                var modelPack = await i.ParseAsync();
                await SaveModelPack(modelPack);
            }
            catch (Exception ex)
            {
                this.logger.LogError(ex, "Error at {Parser}", this.GetType().Name);
            }
        }
    }
    logger.LogDebug("Finish parsing at {Parser}", this.GetType().Name);
}

private async Task SaveModelPack(ModelPack modelPack)
{
    ISaver saver = saverFactory.Create();
    await saver.SaveAsync(modelPack);
}
}
namespace TC.Parsers.Services
{
    public class HttpService
    {
        private static readonly TimeSpan timeout = TimeSpan.FromMinutes(5);

        private readonly IMemoryCache memoryCache;
        private readonly IHttpClientFactory httpClientFactory;
        public HttpService(IHttpClientFactory httpClientFactory, IMemoryCache memoryCache)
        {
            this.httpClientFactory = httpClientFactory;
            this.memoryCache = memoryCache;
        }
        public async Task<string> GetPageAsync(string url)
        {
            if (!memoryCache.TryGetValue(url, out string content))
            {
                content = await GetFromClientAsync(url);
                memoryCache.Set(url, content, timeout);
            }
            return content;
        }
        private async Task<string> GetFromClientAsync(string url)
        {
            var client = CreateClient(new Uri(url));
            try
            {
                var response = await client.GetAsync(url);
                if (response.IsSuccessStatusCode)
                {
                    return await response.Content.ReadAsStringAsync();
                }
                else throw new Exception("Bad response. Link " + url);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
    }
}

```

```

    }
}

private HttpClient CreateClient(Uri uri)
{
    var host = uri.Host;

    return httpClientFactory.CreateClient(host);
}
}
namespace TC.Parsers
{
    using AngleSharp;
    internal class AngleSharpFactory
    {
        public static IBrowsingContext Create()
        {
            var config = Configuration.Default;
            var contex = BrowsingContext.New(config);

            return contex;
        }
    }
}
namespace TC.Parsers.YakabooParser.Info
{
    internal class YakabooPageInfo : PageInfo
    {
        private readonly IInfoFactory infoFactory;

        public YakabooPageInfo(
            [KeyFilter(YakabooConst.META_VALUE)]IServiceFactory serviceFactory,
            HttpService httpService,
            ILogger<YakabooPageInfo> logger)
            :base(httpService, logger)
        {
            this.infoFactory = serviceFactory.GetService<IInfoFactory>();
        }

        public override bool IsPageCorrect()
        {
            var article = this.Document.QuerySelector("article[role=\"main\"]");
            if (article == null) return false;
            return !article.ToHtml().Contains("У даній категорії немає товарів.");
        }

        protected override IEnumerable<ItemInfo> ParseDocument()
        {
            var divCategoryProducts = this.Document.QuerySelector("div.category-products");
            var liItems = divCategoryProducts.QuerySelectorAll("li.item.last");
            var links = liItems.Select(
                li =>
                li.QuerySelectorAll("*")
                .Where(e => e.LocalName == "div")
                .Where(e => e.ClassList.Contains("content"))
                .FirstOrDefault()
                .QuerySelector("a")
                .GetAttribute("href")
            );
            return links.Select(l => this.infoFactory.CreateItem(l));
        }
    }
}

```

```

    }
}

namespace TC.Parsers.YakabooParser.Info
{
    internal class YakabooSiteInfo : SiteInfo
    {
        private const string base_address = "https://www.yakaboo.ua";
        private const string first_page = "/ua/knigi.html?book_lang=Ukrainskij";

        private string primalPageLink = base_address + first_page;

        public YakabooSiteInfo(
            [KeyFilter(YakabooConst.META_VALUE)]IServiceFactory serviceFactory,
            ILogger<YakabooSiteInfo> logger)
            : base(serviceFactory, logger)
        {
        }

        public override string PageLink(int pageNum)
        {
            if (pageNum <= 0) throw new Exception("Page num need to high than 0");
            return primalPageLink + "&p=" + pageNum;
        }
    }
}

namespace TC.Parsers.YakabooParser.ModelParsers
{
    internal abstract class AbstractModelParser : MainAbstractModelParser
    {
        public AbstractModelParser(HttpService httpService, ILogger logger)
            : base(httpService, logger)
        {
        }

        protected string GetFromTable(string rowName)
        {
            var table = document.QuerySelector("table#product-attribute-specs-table");
            var div = table.QuerySelectorAll("div").Where(t =>
t.InnerHtml.Contains(rowName)).FirstOrDefault();
            if (div == null) return null;

            var row = table.QuerySelectorAll("tr").Where(t =>
t.Contains(div)).FirstOrDefault()?.QuerySelectorAll("td").ToList();
            string value = null;
            if (row != null && row.Count > 1)
            {
                value = row[1].Text();
            }
            return value?.Trim();
        }
    }
}

namespace TC.Parsers.YakabooParser.ModelParsers
{
    internal class AuthorModelParser : AbstractModelParser, IModelParser
    {
        protected string rowName = "Автор";
        public AuthorModelParser(HttpService httpService, ILogger<AuthorModelParser> logger)
            : base(httpService, logger)
        {
        }

        protected override IEnumerable<IModel> ParseDocument()
    }
}

```

```

    {
        var fullAuthorName = GetFromTable(rowName);
        if (fullAuthorName == null) return new List<IModel>();

        var authorsNames = fullAuthorName.Split(new string[] { "," },
StringSplitOptions.RemoveEmptyEntries).ToList();
        var authors = authorsNames.Select(n => CreateAutor(n)).ToList();
        return authors;
    }
    protected Author CreateAutor(string fullName)
    {
        if (fullName == null) return null;
        fullName = fullName.Trim();
        var names = NameHelper.ParseName(fullName);
        string firstName = names.Item1;
        string middleName = names.Item2;
        string lastName = names.Item3;

        Author author = new Author()
        {
            FirstName = firstName,
            MiddleName = middleName,
            LastName = lastName,
            FullName = fullName,
            IsTranslator = rowName.Equals("Перекладач")
        };
        return author;
    }
}
}
namespace TC.Parsers.YakabooParser.ModelParsers
{
    internal class BookCoverModelParser : AbstractModelParser
    {
        public BookCoverModelParser(HttpService httpService, ILogger<BookCoverModelParser>
logger)
            : base(httpService, logger)
        {
        }

        protected override IEnumerable<IModel> ParseDocument()
        {
            List<string> links = new List<string>();
            var li = this.document.QuerySelector("li#media_popup_photos");
            if (li != null)
            {
                var imgs = li.QuerySelectorAll("img");
                if (imgs != null && imgs.Length > 0)
                {
                    links = imgs.Select(l => l.GetAttribute("data-original")).Where(l => l
!= null).ToList();
                }
            }

            return new List<BookCover> { new BookCover { Links = links } };
        }
    }
}
namespace TC.Parsers.YakabooParser.ModelParsers
{
    internal class BookModelParser : AbstractModelParser, IModelParser
    {

```

```

public BookModelParser(HttpService httpService, ILogger<BookModelParser> logger)
    : base(httpService, logger)
{
}

protected override IEnumerable<IModel> ParseDocument()
{
    var isbn = GetFromTable("ISBN")?.Replace("-", "");
    var title = GetTitle();
    var language = GetFromTable("Moba");
    var description = GetDescription();

    Book book = new Book()
    {
        Link = this.link.AbsoluteUri,
        ISBN = isbn,
        Description = description,
        Language = language,
        Title = title,
        IsOriginalBook = false
    };
    return new List<Book> { book };
}

private string GetTitle()
{
    string title = null;
    var titleDiv = document.QuerySelector("div#product-title");
    if (titleDiv != null)
    {
        var titleDivAttribute =
titleDiv.QuerySelector("[itemprop=\"description\"]");
        if (titleDivAttribute != null)
        {
            var text = titleDivAttribute.Text();
            title = text.Trim();
        }
    }
    return title;
}

private string GetDescription()
{
    string description = null;

    var div = document.QuerySelector("div.description-
shadow[itemprop=\"description\"]");
    if (div != null)
    {
        description = div.Text().Trim();
    }
    return description;
}
}

namespace TC.Parsers.YakabooParser.ModelParsers
{
    internal class OriginalBookModelParser : AbstractModelParser, IModelParser
    {
        public OriginalBookModelParser(HttpService httpService,
ILogger<OriginalBookModelParser> logger)
            : base(httpService, logger)

```



```

    {
    }

    protected override IEnumerable<IModel> ParseDocument()
    {
        var title = GetTitle();
        if (title == null) return null;
        Book book = new Book()
        {
            Title = title,
            IsOriginalBook = true
        };
        return new List<Book> { book };
    }
    private string GetTitle()
    {
        string title = null;
        var englishNameDiv = document.QuerySelector("div.english-name");
        if (englishNameDiv != null)
        {
            var englishName = englishNameDiv.Text();
            if (!string.IsNullOrEmpty(englishName))
            {
                title = englishName;
            }
        }
        return title;
    }
}

namespace TC.Parsers.YakabooParser.ModelParsers
{
    internal class PublisherModelParser : AbstractModelParser, IModelParser
    {
        public PublisherModelParser(HttpService httpService, ILogger<PublisherModelParser>
logger)
            : base(httpService, logger)
        {
        }

        protected override IEnumerable<IModel> ParseDocument()
        {
            var publishName = GetFromTable("Видавництво");
            var publishYear = GetFromTable("Рік видання");
            DateTime? date = null;
            if (!string.IsNullOrEmpty(publishYear))
            {
                try
                {
                    date = DateTime.ParseExact(publishYear, "yyyy",
CultureInfo.CurrentCulture);
                }
                catch
                {
                    date = null;
                }
            }

            Publisher publisher = new Publisher()
            {
                Name = publishName,
                Date = date
            };

            return new List<Publisher> { publisher };
        }
    }
}

```

```

    }
}
namespace TC.Parsers.YakabooParser.ModelParsers
{
    internal class TranslatorModelParser : AuthorModelParser
    {
        public TranslatorModelParser(HttpService httpService, ILogger<TranslatorModelParser>
logger)
            : base(httpService, logger)
        {
            this.rowName = "Перекладач";
        }
    }
}
namespace TC.Parsers.BookyeParser.Info
{
    internal class BookyePageInfo : PageInfo
    {
        private const string BASE_ADDRESS = "https://book-ye.com.ua";

        private readonly IInfoFactory infoFactory;
        public BookyePageInfo(
            [KeyFilter(BookyeConst.META_VALUE)]IServiceFactory serviceFactory,
            HttpService httpService,
            ILogger<BookyePageInfo> logger)
            : base(httpService, logger)
        {
            this.infoFactory = serviceFactory.GetService<IInfoFactory>();
        }

        public override bool IsPageCorrect()
        {
            var a = this.Document.QuerySelector("a.pagination__item.pagination__item--
next.icon-arrow_right");
            if (a == null) return false;
            var href = a.GetAttribute("href");
            return string.IsNullOrEmpty(href);
        }

        protected override IEnumerable<ItemInfo> ParseDocument()
        {
            var divCategoryProducts = this.Document.QuerySelector("div.products-wrap");
            var aItems = divCategoryProducts.QuerySelectorAll("a.product__name");
            var links = aItems.Select(
                a =>
                a.GetAttribute("href")
                .Insert(0, BASE_ADDRESS)
            );
            return links.Select(l => this.infoFactory.CreateItem(l)).ToList();
        }
    }
}
namespace TC.Parsers.BookyeParser.Info
{
    internal class BookyeSiteInfo : SiteInfo
    {
        private const string BASE_ADDRESS = "https://book-ye.com.ua";
        public BookyeSiteInfo(
            [KeyFilter(BookyeConst.META_VALUE)]IServiceFactory serviceFactory,
            ILogger<BookyeSiteInfo> logger)
            : base(serviceFactory, logger)
        {

```

```

    {
    }

    public override string PageLink(int pageNum)
    {
        return "https://book-ye.com.ua/catalog/vydavnytstva/filter/perekladna-is-true/?PAGEN_1=" + pageNum;
    }
}
namespace TC.Parsers.BookyeParser.ModelParsers
{
    internal abstract class AbstractModelParser : MainAbstractModelParser
    {
        public AbstractModelParser(HttpService httpService, ILogger logger)
            : base(httpService, logger)
        {
        }
        protected string GetFromTable(string rowName)
        {
            var table = document.QuerySelector("div.card__params.js-params");
            var div = table.QuerySelectorAll("div.card__info").Where(t =>
t.InnerHtml.Contains(rowName)).FirstOrDefault();
            if (div == null) return null;

            string divValue = div.Text();
            string value = null;
            if (divValue != null)
            {
                var parts = divValue.Split(new string[] { ":" },
StringSplitOptions.RemoveEmptyEntries);
                if (parts.Length > 1)
                {
                    value = parts[1];
                }
            }
            return value?.Trim();
        }
    }
}
namespace TC.Parsers.BookyeParser.ModelParsers
{
    internal class AuthorModelParser : AbstractModelParser
    {
        protected string rowName = "Автор";

        public AuthorModelParser(HttpService httpService, ILogger<AuthorModelParser> logger)
            : base(httpService, logger)
        {
        }

        protected override IEnumerable<IModel> ParseDocument()
        {
            var fullAuthorName = GetFromTable(rowName);
            if (fullAuthorName == null) return new List<IModel>();

            var authorsNames = fullAuthorName.Split(new string[] { "," },
StringSplitOptions.RemoveEmptyEntries).ToList();
            var authors = authorsNames.Select(n => CreateAutor(n)).ToList();
            return authors;
        }

        protected Author CreateAutor(string fullName)

```

```

        {
            if (fullName == null) return null;
            fullName = fullName.Trim();
            var names = NameHelper.ParseName(fullName);
            string firstName = names.Item1;
            string middleName = names.Item2;
            string lastName = names.Item3;

            Author author = new Author()
            {
                FirstName = firstName,
                MiddleName = middleName,
                LastName = lastName,
                FullName = fullName,
                IsTranslator = rowName.Equals("Перекладач")
            };
            return author;
        }
    }
}
namespace TC.Parsers.BookyeParser.ModelParsers
{
    internal class BookCoverModelParser : AbstractModelParser
    {
        private const string BASE_URL = "https://book-ye.com.ua/";

        public BookCoverModelParser(HttpService httpService, ILogger<BookCoverModelParser>
logger)
            : base(httpService, logger)
        {
        }

        protected override IEnumerable<IModel> ParseDocument()
        {
            List<string> links = new List<string>();
            var li = this.document.QuerySelectorAll("div.owl-item");
            if (li != null)
            {
                var imgs = li.SelectMany(l => l.QuerySelectorAll("img"));
                if (imgs != null && imgs.Count() > 0)
                {
                    links = imgs.Select(l => l.GetAttribute("src")).Where(l => l !=
null).Select(l => BASE_URL + l).ToList();
                }
            }

            return new List<BookCover> { new BookCover { Links = links } };
        }
    }
}
namespace TC.Parsers.BookyeParser.ModelParsers
{
    internal class BookModelParser : AbstractModelParser, IModelParser
    {
        public BookModelParser(HttpService httpService, ILogger<BookModelParser> logger)
            : base(httpService, logger)
        {
        }

        protected override IEnumerable<IModel> ParseDocument()
        {
            string isbn = GetFromTable("ISBN")?.Replace("-", "");
            var language = GetFromTable("Moba");

            Book book = new Book()
            {

```

```

        Link = this.link.AbsoluteUri,
        ISBN = isbn,
        Description = GetTitle(),
        Language = language,
        Title = GetDescription(),
        IsOriginalBook = false
    };
    return new List<Book> { book };
}

private string GetTitle()
{
    var h = this.document.QuerySelector("h1.card__title");
    string value = null;
    if (h != null)
    {
        value = h.Text().Trim();
    }
    return value;
}

private string GetDescription()
{
    var anotations = this.document.QuerySelectorAll("p.article__description");
    string value = null;
    if (anotations != null && anotations.Length > 0)
    {
        var text = anotations.Select(a =>
a.Text()).Where(t => !string.IsNullOrEmpty(t));
        value = string.Join(Environment.NewLine, text).Trim();
    }
    return value;
}
}
}

namespace TC.Parsers.BookyeParser.ModelParsers
{
    internal class PublisherModelParser : AbstractModelParser
    {
        public PublisherModelParser(HttpService httpService, ILogger<PublisherModelParser>
logger)
            : base(httpService, logger)
        {
        }

        protected override IEnumerable<IModel> ParseDocument()
        {
            var publishName = GetFromTable("Видавництво");
            var publishYear = GetFromTable("Рік видання");
            DateTime? date = null;
            if (!string.IsNullOrEmpty(publishYear))
            {
                try
                {
                    date = DateTime.ParseExact(publishYear, "yyyy",
CultureInfo.CurrentCulture);
                }
                catch
                {
                    date = null;
                }
            }

            Publisher publisher = new Publisher()
            {
                Name = publishName,
                Date = date
            }
        }
    }
}

```

```

        };
        return new List<Publisher> { publisher };
    }
}
namespace TC.Parsers.BookyeParser.ModelParsers
{
    internal class TranslatorModelParser : AuthorModelParser
    {
        public TranslatorModelParser(HttpService httpService, ILogger<TranslatorModelParser>
logger)
            : base(httpService, logger)
        {
            this.rowName = "Перекладач";
        }
    }
}
TC.CommonUI

```

```

namespace TC.CommonUI.Helpers
{
    public static class Paginator
    {
        public static IQueryable<TModel> TakePage<TModel>(this IQueryable<TModel> models,
PageInfo pageInfo)
        {
            => models.Skip((pageInfo.PageNumber - 1) * pageInfo.PageSize)
                .Take(pageInfo.PageSize);
        }
    }

    public class PageInfo
    {
        private int currentPageNumber = 1;
        private int pageSize = 1;
        private int totalItems = 0;
        public int PageNumber
        {
            get { return currentPageNumber; }
            set
            {
                if (value < 1 || value > TotalPages)
                    throw new ArgumentException("PageNumber must be hire than 0, and less
then TotalPages");
                currentPageNumber = value;
            }
        }
        public int PageSize
        {
            get { return pageSize; }
            set
            {
                if (value < 1)
                    throw new ArgumentException("PageSize must be hire than 0");
                pageSize = value;
            }
        }
        public int TotalItems
        {
            get { return totalItems; }
            set
            {

```

```

        {
            if(value < 0)
                throw new ArgumentException("TotalItems must be hire or equal than 0");
            totalItems = value;
        }
    }
    public int TotalPages
    {
        get { return (int)Math.Ceiling((decimal)TotalItems / PageSize); }
    }
}
namespace TC.CommonUI.Services
{
    public class Filter
    {
        public string SerachParameter { get; set; }
        public PageInfo PageInfo { get; set; }
    }
}
namespace TC.CommonUI.Services
{
    public class OrderInfo
    {
        public OrderType OrderType { get; set; } = OrderType.None;
        public OrderObject OrderObject { get; set; } = OrderObject.None;
    }

    public enum OrderType
    {
        None = 0,
        Asc = 1,
        Desc = 2
    }

    public enum OrderObject
    {
        None,
        Author,
        Translator,
        Book
    }
}
namespace TC.CommonUI.Services.Realizations
{
    class AdminAuthorizeService : IAdminAuthorizeService
    {
        private const string BASE_URL = "https://localhost:5001";
        private const string SIGN_IN_URL = "/api/authorize/signin";
        private const string SIGN_OUT_URL = "/api/authorize/signout";
        private const string GET_INFO_URL = "/api/authorize/getuser";
        private readonly IHttpClientFactory httpClientFactory;

        private JsonSerializerOptions jsonOptions;

        public AdminAuthorizeService(IHttpClientFactory httpClientFactory)
        {
            this.httpClientFactory = httpClientFactory;

            this.jsonOptions = new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            };
        }
    }
}

```

```

        public async Task<AdminAccount> GetUser()
        {
            AdminAccount account = new AdminAccount();
            try
            {
                var client = httpClientFactory.CreateClient();
                var url = BASE_URL + GET_INFO_URL;
                var response = await client.GetAsync(url);
                var responseMessage = await response.Content.ReadAsStringAsync();
                account = JsonSerializer.Deserialize<AdminAccount>(responseMessage,
jsonOptions);
            }
            catch (Exception ex)
            {
            }
            return account;
        }

        public async Task<bool> SignIn(string login, string password)
        {
            bool isSucceed = false;
            try
            {
                var url = BASE_URL + SIGN_IN_URL + "?login=" + login + "&password=" +
password;
                var client = this.httpClientFactory.CreateClient();
                var response = await client.GetAsync(url);
                var responseMessage = await response.Content.ReadAsStringAsync();
                var result = JsonSerializer.Deserialize<ResultDto>(responseMessage,
jsonOptions);
                isSucceed = result.IsSuccess;
            }
            catch (Exception ex)
            {
                System.Diagnostics.Debug.WriteLine(ex.Message);
            }
            return isSucceed;
        }

        public async Task SignOut()
        {
            var url = BASE_URL + SIGN_OUT_URL;
            var client = this.httpClientFactory.CreateClient();
            await client.GetAsync(url);
        }
    }
}
namespace TC.CommonUI.Services.Realizations
{
    class AdminService : IAdminService
    {
        private readonly GeneralDbContext dbContext;

        public AdminService(GeneralDbContext dbContext)
        {
            this.dbContext = dbContext;
        }

        private IQueryable<IntermediateModelPack> GetQuery()
            => this.dbContext.IntermediateModelPacks.AsNoTracking();
        private IQueryable<IntermediateModelPack>
ApplyFilter(IQueryable<IntermediateModelPack> query, Filter filter)
        {

```



```

        return query;
    }
    public async Task<int> GetCount(Filter filter)
    {
        var query = GetQuery();
        query = ApplyFilter(query, filter);
        return await query.CountAsync();
    }

    public async Task<AdminModelPack> GetModelPackById(int id)
    {
        var modelPackJson = await this.dbContext.IntermediateModelPacks.AsNoTracking()
            .Where(i => i.Id
== id)
.FirstOrDefaultAsync();
        AdminModelPack adminModelPack = null;
        if (modelPackJson != null)
        {
            var modelPack = ModelPackSerializer.ToModelPack(modelPackJson.Content);
            adminModelPack = new AdminModelPack(modelPack,
(ModelPackStateEnum)modelPackJson.StateId);
        }
        return adminModelPack;
    }

    public async Task<IList<ModelPackSmallItem>> GetWithFilter(Filter filter)
    {
        var query = GetQuery();
        query = query.TakePage(filter.PageInfo);
        query = ApplyFilter(query, filter);
        return await query.Select(ModelPackSmallItem.Projection).ToListAsync();
    }

    private async Task GetFromDb()
    {
        var modelPacksJson = await this.dbContext.IntermediateModelPacks.AsNoTracking()
            .Where(i =>
i.StateId == (int)ModelPackStateEnum.InProgress)
            .ToListAsync();

        var modelPacks = modelPacksJson.Select(m =>
ModelPackSerializer.ToModelPack(m.Content)).ToList();
    }

    public async Task Save(int id, AdminModelPack adminModelPack)
    {
        var modelPackJson = await
this.dbContext.IntermediateModelPacks.AsQueryable().FirstOrDefaultAsync(i => i.Id == id);
        if (adminModelPack.Status == ModelPackStateEnum.Applied)
        {
            var modelPack = adminModelPack.ToModelPack();
            var json = ModelPackSerializer.ToJson(modelPack);
            modelPackJson.Content = json;
            modelPackJson.StateId = (int)ModelPackStateEnum.Applied;
        }
        else if (adminModelPack.Status == ModelPackStateEnum.Rejected)
        {
            modelPackJson.StateId = (int)ModelPackStateEnum.Applied;
        }

        await this.dbContext.SaveChangesAsync();
    }
}
}
namespace TC.CommonUI.Services.Realizations

```

```

{
    class AuthorNameService : IAuthorNameService
    {
        private readonly GeneralDbContext dbContext;

        public AuthorNameService(GeneralDbContext dbContext)
        {
            this.dbContext = dbContext;
        }

        public async Task<IList<AuthorNameCombined>> GetCombinedNames()
        {
            try
            {
                var authorNamesQuery = this.dbContext.AuthorNames.Include(a =>
a.SameNames).AsQueryable();

                authorNamesQuery = authorNamesQuery.Where(a => a.SameNames != null).Where(a
=> a.SameNames.Count() > 0);

                var combinedNameQuery = authorNamesQuery.AsQueryable()
                    .Select(AuthorNameCombined.AuthorNameProjection);
                return await combinedNameQuery.ToListAsync();
            }
            catch (Exception ex)
            {
            }
            return new List<AuthorNameCombined>();
        }

        public async Task<IList<AuthorNameItem>> GetAllNames()
        {
            return await
this.dbContext.AuthorNames.Select(AuthorNameItem.Projection).ToListAsync();
        }

        public async Task<bool> IsPairExist(int firstNameId, int secondNameId)
        {
            return (await this.dbContext.PairsOfSameName.AsQueryable()
                .FirstOrDefaultAsync(p => p.FirstNameId == firstNameId &&
p.SecondNameId == secondNameId))
                != null;
        }

        public async Task AddPair(int firstNameId, int secondNameId)
        {
            var first = new PairOfSameName
            {
                FirstNameId = firstNameId,
                SecondNameId = secondNameId
            };
            var second = new PairOfSameName
            {
                FirstNameId = secondNameId,
                SecondNameId = firstNameId
            };

            await this.dbContext.AddRangeAsync(first, second);
            await this.dbContext.SaveChangesAsync();
        }
    }
}
namespace TC.CommonUI.Services.Realizations
{
    class AuthorService : IAuthorService
    {

```

```

private readonly GeneralDbContext dbContext;

public AuthorService(GeneralDbContext dbContext)
{
    this.dbContext = dbContext;
}

private IQueryable<Author> Query()
    => this.dbContext.Authors.AsNoTracking();

private IQueryable<Author> WithName(IQueryable<Author> query)
    => query.Include(a => a.AuthorName);

private IQueryable<Author> WithBook(IQueryable<Author> query)
    => query.Include(a => a.AuthorBooks);

private IQueryable<Author> WithSameNames(IQueryable<Author> query)
    => query.Include(a => a.AuthorName)
        .ThenInclude(an => an.SameNames)
        .ThenInclude(p => p.SecondName);

public async Task<AuthorWithOtherNames> GetAuthorById(int id)
{
    var query = Query();
    query = WithName(query);
    query = WithBook(query);
    query = WithSameNames(query);
    return await query
        .Select(AuthorWithOtherNames.Projection)
        .FirstOrDefaultAsync(a => a.Id == id);
}

public async Task<IList<AuthorWithOtherNames>> GetFilteredAuthor(Filter filter)
{
    var query = Query();
    query = query.TakePage(filter.PageInfo);
    query = WithName(query);
    query = WithBook(query);
    query = WithSameNames(query);
    query = ApplyFilter(query, filter);
    return await query.Select(AuthorWithOtherNames.Projection)
        .ToListAsync();
}

public async Task<int> GetCountOfAuthors(Filter filter)
{
    var query = Query();
    query = WithName(query);
    query = ApplyFilter(query, filter);
    return await query.CountAsync();
}

private IQueryable<Author> ApplyFilter(IQueryable<Author> query, Filter filter)
{
    if (!string.IsNullOrEmpty(filter.SerachParameter))
    {
        query = query.Where(a =>
a.AuthorName.FullName.ToLower().Contains(filter.SerachParameter.ToLower()));
    }

    return query;
}

public async Task<IList<BookSmallItem>> GetSmallBooksByAuthor(int authorId)
{
    var authorQuery = Query();

```

```

        authorQuery = WithName(authorQuery);
        authorQuery = WithSameNames(authorQuery);
        var author = await authorQuery.FirstOrDefaultAsync(a => a.Id == authorId);

        List<int> authorNameIds = new List<int> { author.AuthorNameId };
        var otherAuthorIds = author.AuthorName.SameNames.Select(p => p.SecondNameId);
        authorNameIds.AddRange(otherAuthorIds);

        var smallBooks = await dbContext.Authors
            .Include(an => an.Authors)
            .ThenInclude(a => a.AuthorBooks)
            .ThenInclude(ab => ab.Book)
            .ThenInclude(b => b.BookCovers)
            .Where(an => authorNameIds.Contains(an.Id))
            .SelectMany(an => an.Authors)
            .SelectMany(a => a.AuthorBooks)
            .Select(ab => ab.Book)
            .Select(BookSmallItem.Projection)
            .ToListAsync();

        smallBooks = smallBooks.GroupBy(b => b.Id).Select(g =>
g.FirstOrDefault()).ToList();
        return smallBooks;
    }
}
namespace TC.CommonUI.Services.Realizations
{
    class BookService : IBookService
    {
        private readonly GeneralDbContext dbContext;

        public BookService(GeneralDbContext dbContext)
        {
            this.dbContext = dbContext;
        }
        private IQueryable<Book> GetOnlyBookQuery()
            => dbContext.Books.AsNoTracking();

        private IQueryable<Book> WithCovers(IQueryable<Book> books)
            => books
                .Include(b => b.BookCovers)
                .ThenInclude(bc => bc.File);

        private IQueryable<Book> WithAuthor(IQueryable<Book> books)
            => books
                .Include(b => b.AuthorBooks)
                .ThenInclude(ab => ab.Author)
                .ThenInclude(a => a.AuthorName);

        private IQueryable<Book> WithPublish(IQueryable<Book> books)
            => books.Include(b => b.PublishDates)
                .ThenInclude(pd => pd.Publisher);

        public async Task<BookItem> GetBookItemById(int id)
        {
            var books = GetOnlyBookQuery();
            books = WithAuthor(books);
            books = WithCovers(books);
            books = WithPublish(books);
            return await books
                .Select(BookItem.Projection)
                .FirstOrDefaultAsync(b => b.Id == id);
        }

        private IQueryable<Book> GetSmallBookPageQuery(PageInfo pageInfo)

```

```

    {
        var books = GetOnlyBookQuery();
        books = books.TakePage(pageInfo);
        books = WithAuthor(books);
        books = WithCovers(books);
        books = books.Where(b => b.IsTranslated);
        return books;
    }
    public async Task<IList<BookSmallItem>> GetSmallBookPage(PageInfo pageInfo)
        => await GetSmallBookPageQuery(pageInfo)
            .Select(BookSmallItem.Projection)
            .ToListAsync();

    public async Task<int> GetTotalBookCount(Filter filter)
    {
        var query = GetOnlyBookQuery();
        query = WithAuthor(query);
        query = WithCovers(query);
        query = ApplyFilter(query, filter);

        query = query.Where(b => b.IsTranslated);

        return await query.CountAsync();
    }

    public async Task<IList<BookSmallItem>> GetSamllBookPageWithFilter(Filter filter)
    {
        var query = GetSmallBookPageQuery(filter.PageInfo);
        query = ApplyFilter(query, filter);
        return await query.Select(BookSmallItem.Projection).ToListAsync();
    }

    private IQueryable<Book> ApplyFilter(IQueryable<Book> query, Filter filter)
    {
        if (filter == null) return query;
        if (!string.IsNullOrEmpty(filter.SerachParameter))
        {
            query = query.Where(b =>
b.Title.ToLower().Contains(filter.SerachParameter.ToLower()));
        }
        return query;
    }

    public async Task<IList<BookSmallItem>> GetSmallIBookFilterAndOrder(Filter filter,
OrderInfo orderInfo)
    {
        try
        {
            var query = GetOnlyBookQuery();
            query = WithAuthor(query);
            query = WithCovers(query);
            query = ApplyFilter(query, filter);

            query = query.Where(b => b.IsTranslated);

            var books = await query.ToListAsync();

            if (orderInfo.OrderObject == OrderObject.Book)
            {
                books = OrderByTitle(books.AsQueryable(), orderInfo.OrderType).ToList();
            }
            else if (orderInfo.OrderObject == OrderObject.Author)
            {
                books = OrderByAuthor(books.AsQueryable(),
orderInfo.OrderType).ToList();
            }
        }
    }

```

```

        else if (orderInfo.OrderObject == OrderObject.Translator)
        {
            books = OrderByTranslator(books.AsQueryable(),
orderInfo.OrderType).ToList();
        }
        books = books.AsQueryable().TakePage(filter.PageInfo).ToList();
        var smallBooks =
books.AsQueryable().Select(BookSmallItem.Projection).ToList();
        return smallBooks;
    }
    catch (Exception ex)
    {
    }
    return new List<BookSmallItem>();
}

private IQueryable<Book> OrderByTranslator(IQueryable<Book> query, OrderType
orderType)
{
    query = query.Where(b => b.AuthorBooks.Where(ab => ab.AuthorId >
0).FirstOrDefault(a => a.Author.AuthorTypeId == (int)AuthorTypeEnum.Translator) != null);
    if (orderType == OrderType.Asc)
    {
        query = query.OrderBy(b => b.AuthorBooks.FirstOrDefault(a =>
a.Author.AuthorTypeId == (int)AuthorTypeEnum.Translator).Author.AuthorName.FullName ?? "");
    }
    else if (orderType == OrderType.Desc)
    {
        query = query.OrderByDescending(b => b.AuthorBooks.FirstOrDefault(a =>
a.Author.AuthorTypeId == (int)AuthorTypeEnum.Translator).Author.AuthorName.FullName ?? "");
    }
    return query;
}

private IQueryable<Book> OrderByAuthor(IQueryable<Book> query, OrderType orderType)
{
    query = query.Where(b => b.AuthorBooks.Where(ab => ab.AuthorId >
0).FirstOrDefault(a => a.Author.AuthorTypeId == (int)AuthorTypeEnum.Original) != null);
    if (orderType == OrderType.Asc)
    {
        query = query.OrderBy(b => b.AuthorBooks.FirstOrDefault(a =>
a.Author.AuthorTypeId == (int)AuthorTypeEnum.Original).Author.AuthorName.FullName);
    }
    else if (orderType == OrderType.Desc)
    {
        query = query.OrderByDescending(b => b.AuthorBooks.FirstOrDefault(a =>
a.Author.AuthorTypeId == (int)AuthorTypeEnum.Original).Author.AuthorName.FullName);
    }
    return query;
}

private IQueryable<Book> OrderByTitle(IQueryable<Book> query, OrderType orderType)
{
    if (orderType == OrderType.Asc)
    {
        query = query.OrderBy(b => b.Title);
    }
    else if (orderType == OrderType.Desc)
    {
        query = query.OrderByDescending(b => b.Title);
    }
    return query;
}

```

```

    }
}
namespace TC.CommonUI.Services.Realizations
{
    public class IdentityAuthenticationStateProvider : AuthenticationStateProvider
    {
        private readonly IAdminAuthorizeService adminAuthorizeService;

        public IdentityAuthenticationStateProvider(IAdminAuthorizeService
adminAuthorizeService)
        {
            this.adminAuthorizeService = adminAuthorizeService;
        }

        public async Task Login(string login, string password)
        {
            await adminAuthorizeService.SignIn(login, password);
            NotifyAuthenticationStateChanged(GetAuthenticationStateAsync());
        }

        public async Task SignOut()
        {
            await adminAuthorizeService.SignOut();
            NotifyAuthenticationStateChanged(GetAuthenticationStateAsync());
        }

        public async Task<AdminAccount> GetUserInfo()
        {
            var user = await adminAuthorizeService.GetUser();
            return user;
        }

        public override async Task<AuthenticationState> GetAuthenticationStateAsync()
        {
            var user = await adminAuthorizeService.GetUser();
            var claim = user.ToClaim();
            var state = new AuthenticationState(claim);
            return state;
        }
    }
}

```

TranslatorCatalog

```

namespace TranslatorCatalog.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AuthorizeController : ControllerBase
    {
        private readonly IAuthorizeService authorizeService;

        public AuthorizeController(IAuthorizeService authorizeService)
        {
            this.authorizeService = authorizeService;
        }

        [HttpGet("[action]")]
        public async Task<ResultDto> SignIn(string login, string password)
        {
            var result = await this.authorizeService.SignIn(login, password);

            return result;
        }
    }
}

```

```

    }

    [HttpGet("[action]")]
    public Task<AdminAccount> GetUser()
    {
        var user = this.User.ToAccount();
        return Task.FromResult(user);
    }

    [HttpGet("[action]")]
    public async Task<IActionResult> SignOut()
    {
        await this.authorizeService.SignOut();
        return Ok();
    }
}

namespace TranslatorCatalog
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var logger =
NLog.Web.NLogBuilder.ConfigureNLog("NLog.config").GetCurrentClassLogger();
            try
            {
                logger.Debug("init main");
                CreateHostBuilder(args).Build().Run();
            }
            catch (Exception exception)
            {
                //NLog: catch setup errors
                logger.Error(exception, "Stopped program because of exception");
                throw;
            }
            finally
            {
                // Ensure to flush and stop internal timers/threads before application-exit
(Avoid segmentation fault on Linux)
                NLog.LogManager.Shutdown();
            }
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .UseServiceProviderFactory(new AutofacServiceProviderFactory())
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                })
                .ConfigureLogging(logging =>
                {
                    logging.ClearProviders();
                    logging.SetMinimumLevel(Microsoft.Extensions.Logging.LogLevel.Debug);
                })
                .UseNLog();
    }
}

namespace TranslatorCatalog
{
    public class Startup
    {
        public Startup(IConfiguration configuration)

```



```

    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the
    container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddCustomAuthentication(Configuration);
        services.AddHttpClient();
        services.AddParser();
        services.AddDataSaver();
        services.AddCommonUI();

        services.AddHttpClient();

        services.AddControllers();
        services.AddRazorPages()
            .AddRazorRuntimeCompilation();
        services.AddServerSideBlazor();
        //services.AddReports();

        var serviceDescriptors = services.Where(descriptor => descriptor.ServiceType ==
typeof(AuthenticationStateProvider)).ToList();
        serviceDescriptors.ForEach(s => services.Remove(s));

        services.AddScoped<AuthenticationStateProvider,
IdentityAuthenticationStateProvider>();

        //services.AddBackgroundService();
    }
    public void ConfigureContainer(ContainerBuilder builder)
    {
        // Register your own things directly with Autofac, like:
        builder.RegisterModule(new ParserModule());
        builder.RegisterModule(new DataSaverModule());
    }

    // This method gets called by the runtime. Use this method to configure the HTTP
    request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            //app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days. You may want to change this for
            production scenarios, see https://aka.ms/aspnetcore-hsts.
            app.UseHsts();
        }
        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthentication();
        app.UseAuthorization();

```

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapDefaultControllerRoute();
    endpoints.MapControllers();

    endpoints.MapBlazorHub();
    endpoints.MapFallbackToPage("/_Host");
});
}
}
```