НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій СТІРЕНКО

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Веб-додаток агрегатор курсів дистанційного навчання»

Виконав:
студент IV курсу, групи ІВ-72
Фурман Костянтин Олегович _____

Керівник:
асистент
Каплунов Артем Володимирович _____

Консультант з н. контроль:
Професор, д.т.н,
Сімоненко Валерій Павлович _____

Рецензент:

_____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Сергій СТІРЕНКО

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Фурману Костянтину Олеговичу

1. Тема проєкту «Веб-додаток агрегатор курсів дистанційного навчання», керівник проєкту Каплунов Артем Володимирович, асистент, затверджені наказом по університету від « 11 » травня 2021 р. № 1139-с

2. Термін подання студентом проєкту 2021 р.

3. Вихідні дані до проєкту технічна документація, теоретичні дані, інтернет публікації на тему роботи

4. Зміст пояснювальної записки: аналіз форм представлення оновлень інформації на сайтах та огляд існуючих агрегаторів онлайн курсів, огляд варіантів рішень та вибір інструментів для реалізації, розробка програмної реалізації агрегатора, тестування функціоналу розробки та інструкція для користувача

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): узагальнена схема роботи системи, uml-діаграма классів бази данних, блок-схема алгоритму роботи модуля-парсера.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Н. контроль	Сімоненко В.П., професор, д.т.н.		

7. Дата видачі завдання 01.09.2020 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Затвердження теми роботи	10.12.2020-15.12.2020	
2.	Вивчення та аналіз завдання	14.12.2020-15.03.2021	
3.	Розробка архітектури та загальної структури системи	15.03.2021-25.03.2021	
4.	Розробка структур окремих підсистем	25.03.2021-5.04.2021	
5.	Програмна реалізація системи	5.04.2021-15.04.2021	
6.	Оформлення пояснювальної записки	15.04.2021-20.05.2021	
7.	Захист програмного продукту	25.04.2021	
8.	Передзахист	23.05.2021	
9.	Захист	18.06.2021	

Студент Костянтин ФУРМАН

Керівник Артем КАПЛУНОВ

АНОТАЦІЯ

Даний дипломний проект присвячений створенню веб-додатка для сбору з різних джерел актуальних курсів онлайн навчання з можливістю пошуку та фільтрації за декількома параметрами.

Веб-додаток являє собою інтерактивний сайт, що дозволяє шукати за ключовими словами або категоріями курси навчання з різних джерел. Для зручності користувачів на сайті існує можливість фільтрації результатів пошуку за декількома критеріями.

У даному дипломному проекті розроблено: архітектуру web-додатку, модуль-парсер HTML-сторінок, дизайн клієнтського інтерфейсу web-додатку.

ANNOTATION

This diploma project is dedicated to the creation of web application for collection from various sources of online learning courses with a ability to search and filter results of aggreagation.

A web application is an interactive website that can search for keywords or categories of courses from a variety of sources. For the convenience of users on the website, it is possible to filter the search results of several authors.

In this diploma project the following are developed: web application architecture, HTML-page parser module, web application client interface design.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ.467100.001 ВП	Відомість проекту	1	
3	А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	60	
5	А4	ІАЛЦ.467100.004 Д1	UML діаграма класів моделі бази даних	1	
6	А4	ІАЛЦ.467100.005 Д2	Структурна схема системи	1	
7	А4	ІАЛЦ.467100.006 Д3	Блок-схема алгоритму роботи модуля-парсера	1	
8	А4	ІАЛЦ.467100.007 Д4	Сирцевий код проєкту	11	

Зм.		№ документа	Підп.	Дата	*ІАЛЦ.467100.001 ВП*		
		Фурман К.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / 1
		Каплунов А.В.					
Н.контр.		Сімоненко В. П.				*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІВ-72*	
Затв.		Стіренко С. Г.					

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи

освітньо-кваліфікаційного рівня бакалавр

на тему: «Веб-додаток агрегатор курсів дистанційного навчання»

Київ – 2021 рік

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Агрегатор курсів онлайн навчання.

Галузь застосування: інформаційні технології.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут» (НТУУ «КПІ»).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Призначенням прозробки є створення ресурсу для зручного та швидкого пошуку користувачем курсів навчання з різних джерел.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Веб-додаток повинен забезпечувати такі основні функції:

1) актуальність курсів онлайн навчання;

2) зручний пошук по результатам агрегації

3) динамічне оновлення вмісту веб-сторінок;

4) фільтрація та категоризація результатів агрегації;

5) простий та зрозумілий дизайн веб-сторінок

5) розробку виконати з використанням веб-фреймворку (на вибір);

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1. пояснювальна записка;
2. програма та методика тестування;
3. інструкція користувача;
4. креслення:
   – «Схема роботи програмної системи»;
   – «UML-діаграма класів моделі бази даних»;
   – «Блок-схема алгоритму роботи модуля-парсера».

6. ЕТАПИ ПРОЕКТУВАННЯ

	Дата
Вивчення та аналіз завдання	15.12.2020
Розробка архітектури та загальної структури системи	15.03.2021
Розробка структур окремих підсистем	25.03.2021
Програмна реалізація системи	05.04.2021
Оформлення пояснювальної записки	15.04.2021

Пояснювальна записка
до дипломного проєкту
на тему: «Веб-додаток агрегатор курсів дистанційного навчання»

Київ - 2021 року

ЗМІСТ

ВСТУП

Сучасний світ надзвичайно стрімко розвивається та змінюється. Людям потрібно за ним встигати та вчасно адаптуватись. Курси дистанційного навчання один з методів швидкого розвитку та навчання. Але існує багато сайтів що пропонують різні онлайн курси та пошук потрібного курсу на багатьох різних сайтах може зайняти багато часу. Пошук за допомогою пошукових систем також не спрощує життя користувача, адже пошукова система не відображає інформацію про курс, а лише дає на нього посилання заставляючи користувача відвідувати безліч вебсайтів в надії знайти найкращий курс. Таким чином набуває актуальності необхідність агрегації та відображення потрібної користувачеві інформації на одному вебсайті.

Технологія агрегації об'єднує контент з багатьох вебсайтів в одну сторінку, яка відображатиме найактуальнішу інформацію зібрану з різних ресурсів. Агрегатори скорочують зусилля і час, необхідні для пошуку по різним вебсайтам потрібної інформації. На відміну від пошукової системи сайт агрегатор відображає лише інформацію пов’язану з онлайн курсами, що зменшує зусилля користувача при пошуку та фільтрації результату. Веб-агрегатор забезпечує швидке та компактне відображення вмісту у веб-браузері та допомагає користувачеві краще та оперативніше орієнтуватись в агрегованій інформації, оскільки результат агрегації відображається в зручному для користувача вигляді. Вебсайти агрегатори є зручними та актуальними рішеннями проблеми, надаючи користувачам доступ до інформації з будь-якого девайсу підключеного до мережі Інтернет.

Завданням даної роботи буде розробка веб-додатку агрегатора курсів дистанційного навчання, який збиратиме дані з провідних сайтів онлайн навчання, зберігатиме їх та відображатиме користувачу в зручній для нього формі.

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

XML	Extensible Markup Language – розширювана мова розмітки.
Content Aggregation	агрегація контенту – збір інформації з різних джерел.
URL	Uniform Resource Locator — єдиний вказівник наресурс.
W3C	World Wide Web Consortium – головна міжнародна організація, яка займається розробкою та впровадженням технологічних стандартів для Всесвітнього павутиння.
IETF	Internet Engineering Task Force — відкрите міжнародне співтовариство проектувальників, учених, мережевих операторів і провайдерів, яке займається розвитком протоколів і архітектури Інтернету.
JSON	JavaScript Object Notation – текстовий формат даних, що базується на мові JavaScript.
MVC	Model-view-controller – архітектурний шаблон “Модель–вигляд–контролер”.
DRY	“Don't repeat yourself” – “не повторюй себе” — принцип розробки програмного забезпечення, що направлений на уникнення дублювання інформації будь-якого вигляду.
HTTP	Hyper Text Transfer Protocol – протокол передачі гіпер-текстових документів.
ORM	Object-relational mapping – об'єктно-реляційна проекція — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базуданих».
CSS	Cascading Style Sheets – каскадні таблиці стилів.
HTML	Hyper Text Transfer Protocol – протокол передачі гіпертекстових документів.
SGML	Standard Generalized Markup Language – стандартна узагальнена мова розмітки.
ISO	International Organization for Standardization – Міжнародна організація зі стандартизації

DOM	Document Object Model – об'єктна модель документа.
AJAX	Asynchronous JavaScript And XML – асинхронний JavaScript та XML – підхід до побудови користувацьких інтерфейсів веб-застосунків.
MOOC	Massive Open Online Courses – масові відкриті онлайн-курси.

РОЗДІЛ 1

АНАЛІЗ ВІДОМИХ ТЕХНОЛОГІЙ ТА АЛГОРИТМІВ ПАРСИНГУ ВЕБСАЙТІВ ТА ОГЛЯД ІСНУЮЧИХ АГРЕГАТОРІВ КУРСІВ НАВЧАННЯ

Парсити - збирати і систематизувати інформацію, розміщену на певних вебсайтах, за допомогою спеціальних програм парсерів, що автоматизують процес та полегшують збір та обробку великої кількості даних.

Парсери вебсайтів - це програмні продукти, частиною яких є синтаксичні аналізатори, які преобразують вхідні дані (зазвичай текст) в структурований формат[1]. Основною функцією веб парсерів є отримання структурованих та готових до подальшої обробки або збереження даних з веб-сторінок.

Інформація на веб-сторінках - набір даних, структурований за допомогою мов тегів, стилів та людських мов. Людською мовою надана інформація, заради яких, власне, люди і користуються Інтернетом. Мови тегів (HTML) та стилів (CSS) визначають вигляд інформації на екрані користувача, розмітку та стиль сторінки. В сучасних веб-сторінках також широко використовуються JavaScript скрипти, які допомагають користувачу веб-сторінки інтерактивніше взаємодіяти з контентом.

1.1 Порівняння засобів представлення інформації на веб-сторінках

1.1.1 HTML

HTML (англ. HyperText Markup Language - «мова розмітки гіпертексту») - стандартизований мова розмітки документів у мережі Інтернет. Майже всі веб-сторінки мережі Інтернет містять опис розмітки на мові HTML (або XHTML). Мова HTML інтерпретується браузерами, форматований текст, отриманий в результаті інтерпретації, відображається на

екрані монітора або мобільного пристрою. Остання версія HTML називається HTML5 і була опублікована 28 жовтня 2014 року.

HTML семантично описує структуру вебсторінки. Елементи HTML є будівельними блоками HTML сторінок. За допомогою HTML конструкцій, зображення та інші об'єкти, наприклад інтерактивні форми, можуть вбудовуватись у візуалізовану сторінку.

HTML дозволяє створювати структуровані документи, за допомогою позначення структурної семантики тексту, наприклад заголовків, абзаців, посиланнь, списків та інших елементів. Елементи HTML виділені тегами, написаними з використанням кутових дужок. Теги містять інформацію про текст документа і можуть включати інші теги як піделементи. Браузери не показують користувачам теги HTML, але використовують їх для розмітки та інтерпретації вмісту сторінки.

Документи HTML складаються з дерева елементів та тексту. Кожен елемент позначається у джерелі початковим тегом, таким як "<body>", і кінцевим тегом, таким як "</body>". (Деякі початкові та кінцеві теги в певних випадках можуть бути опущені, і це передбачається іншими тегами.) Теги повинні бути вкладені таким чином, щоб всі елементи знаходились повністю один в одному, без перекриття[2].

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>Sample page</title>
 </head>
 <body>
  <h1>Sample page</h1>
  <p>This is a <a href="demo.html">simple</a> sample.
</p>
  <!-- this is a comment -->
 </body>
</html>
```

Рис. 1.1 – приклад HTML документу [2]

HTML до 5-ї версії визначався як додаток SGML (англ. Standard Generalized Markup Language - стандартної узагальненої мови розмітки за стандартом ISO 8879). Специфікації HTML5 формулюються в термінах DOM (об'єктній моделі документа).

У мережі Інтернет HTML-сторінки, зазвичай, браузер отримує від сервера по протоколах HTTP або HTTPS, у вигляді простого або зашифрованого тексту. В HTML можна вбудувати програмний код на мові програмування JavaScript, для управління поведінкою і змістом веб-сторінок. Також включення CSS в HTML описує зовнішній вигляд і макет сторінки.

1.1.2 JavaScript

JavaScript (часто скорочується як JS) - динамічна, об'єктно-орієнтована прототипна мова програмування. Зазвичай використовується для створення сценаріїв веб-сторінок, що дає можливість веб-сторінці на боці клієнта взаємодіяти з користувачем, асинхронно обмінюватися з сервером даними, керувати браузером, змінювати зовнішній вигляд та структуру вебсторінки. JavaScript має синтаксис фігурних дужок, динамічне введення тексту, об’єктну орієнтацію та функції першого класу.

Поряд з HTML та CSS, JavaScript є однією з основних технологій Всесвітньої мережі. Понад 97% вебсайтів використовують його на стороні клієнта для поведінки веб-сторінок, часто включаючи сторонні бібліотеки. Усі основні веб-браузери мають спеціальний механізм для виконання JavaScript коду на пристрої користувача, що може бути використано для проектування / програмування поведінки веб-сторінок при виникненні певної події. JavaScript - це проста у вивченні, а також потужна мова сценаріїв, широко використовувана для контролю поведінки веб-сторінок[3].

Спочатку двигуни JavaScript використовувались лише у веб-браузерах, але зараз вони є основними компонентами інших програмних систем, зокрема серверів та різноманітних додатків.

Мова JavaScript використовується для:

- написання сценаріїв (скриптів) для надання інтерактивності веб-сторінкам
- створення прогресивних, односторінкових вебзастосунків (AngularJS, React, Vue.js)
- програмування на боці сервера (Node.js)
- десктоп застосунків (Electron, NW.js)
- мобільних застосунків (Cordova, React Native)
- всередині PDF-документів

Через схожість назв, мови Java та JavaScript часто сплутують, але це дві різні мови, що мають відмінну семантику, хоча й схожі риси в правилах іменування та стандартних бібліотеках. Синтаксис обох мов схожий до синтаксису мови C, хоча дизайн та семантика JavaScript є результатом впливу мов Scheme та Self.

1.1.3 XML

Інколи веб-сторінки отримують дані від сервера в форматі XML. XML сторінки розпаршуються в веб-браузері користувача за допомогою JavaScript скриптів.

XML (англ. EXtensible Markup Language) - розширювана мова розмітки. Рекомендований консорціумом World Wide Web Consortium (W3C)[4]. Специфікація XML описує XML-документи і описує поведінку XML-процесорів (програм, що читають XML-документи і забезпечують доступ до їх вмісту). XML розроблявся з простим формальним синтаксисом, зручним для створення і обробки документів як програмами так і людиною, з акцентом на використання в мережі Інтернет.

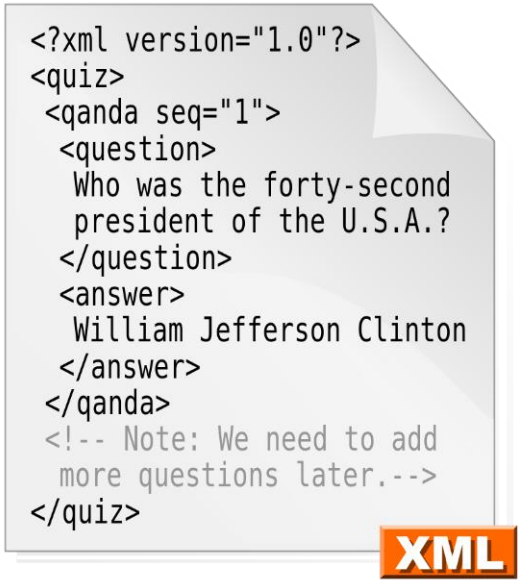


Рис. 1.2 – приклад XML документу [4]

Мова називається розширюваною, оскільки в ній не фіксується розмітка, яка використовується в документах: розробник має свободу створювати розмітку відповідно до потреб конкретної області, будучи обмеженим лише синтаксичними правилами мови. Розширення XML - це конкретна граматика, створена на базі XML і представлена словником тегів і їх атрибутів, а також набором правил, що визначають, які атрибути і елементи можуть входити до складу інших елементів.

1.1.4 JSON

JSON (JavaScript Object Notation) – це текстовий формат обміну даними між комп'ютерами. Це дуже поширений формат даних, із різноманітним набором програм, одним із прикладів є веб-програми, які взаємодіють із сервером.

JSON незалежний від мови програмування формат даних. Він був отриманий з JavaScript, але багато сучасних мов програмування включають бібліотеки та методи для генерації та синтаксичного аналізу даних у форматі JSON. Імена файлів JSON використовують розширення .json[5]. Дуглас Крокфорд розробив і популяризував формат на початку 2000-х.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```


Рис. 1.3 – приклад JSON синтаксису

1.2. Приклади агрегаторів курсів онлайн навчання

1.2.1 Your Skills, y-skills.com

Your Skills агрегатор курсів переважно на російській мові. Проект позиціонує себе як найбільший агрегатор онлайн курсів на ринку СНД. На вебсайті представлено більш ніж 900 курсів по програмуванню та розробці, естетичній красі, бізнесу та маркетингу. Користувач може вибрати між платними та безкоштовними курсами. Проект почався в 2010 році та працює по сьогоднішній день.

Особливості:

На головній сторінці представлене меню пошуку за направленням, категорією та діапазоном ціни. Користувач може обрати категорію та підкатегорію як наприклад Програмирование – 1С; Java; Linux. Результат

категорію можна відфільтрувати за допомогою фільтрів представлених зліва від списку доступних курсів. В правому верхньому куту сторінки є пошук за словами. Пошук по ключовим словам дає користувачеві можливість одразу знайти потрібний йому курс.

На вебсайті наявні такі фільтри:

- фільтр ціни з параметрами - від, до та інтерактивний слайдером діапазону ціни
- фільтр рівня складності – початковий, середній, просунутий
- фільтр наявності сертифікату після закінчення курсу
- фільтр дати початку та кінця курсу
- фільтр тривалості – не більше 3 місяців, від 3 до 6 місяців, більше 6 місяців

Меню сортування над списком курсів дає користувачеві відсортувати результат в потрібному йому порядку, як наприклад найближчі, найдешевші або найдорожчі.

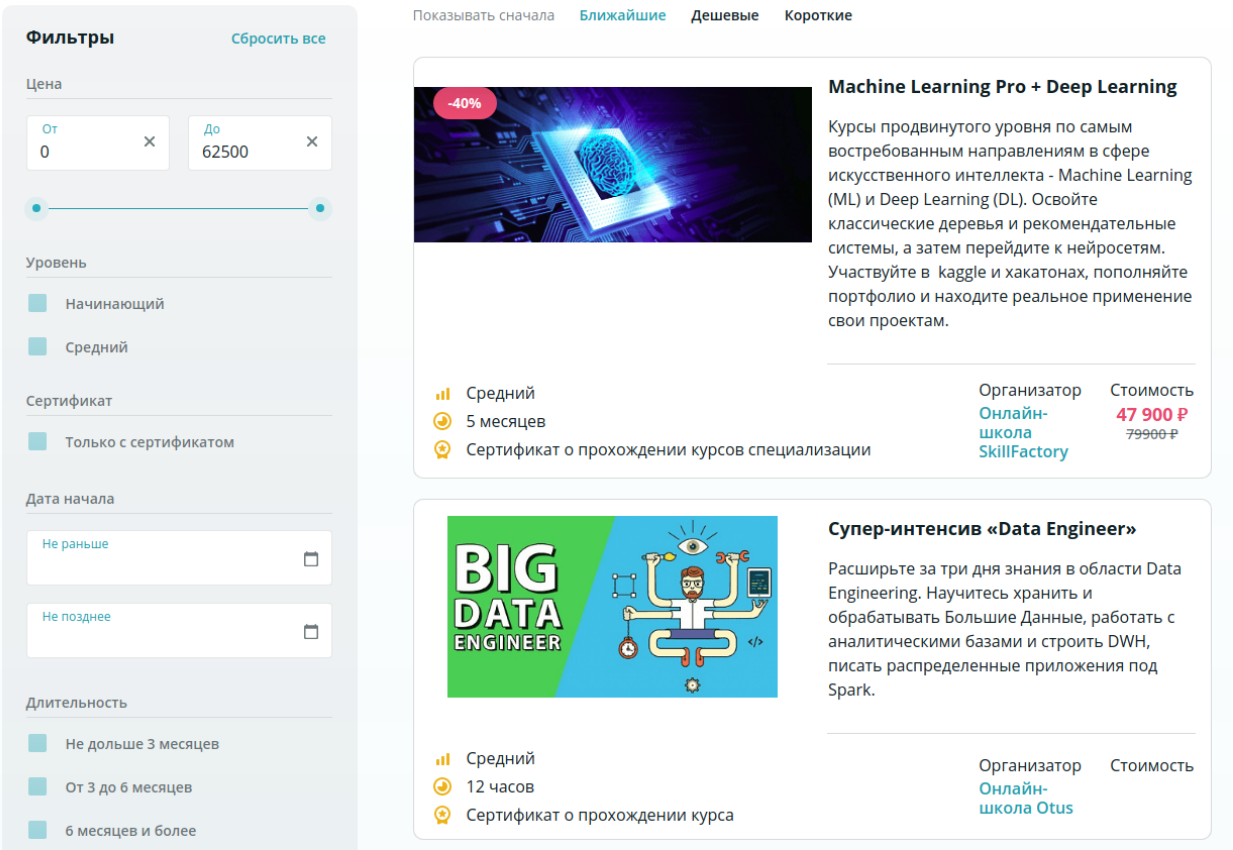


Рис. 1.4 – приклад інтерфейсу агрегатора Your Skills

При виборі курсу користувач перенаправляється на сторінку агрегатора з додатковою інформацією про курс та можливістю перейти на джерело онлайн курсу.

1.2.2 CourseBuffeet, coursebuffet.com

CourseBuffeet агрегатор курсів англійською мовою. Перелічує всі курси з відкритих онлайн-курсів (MOOC), такі як: Coursera, Udacity, edX тощо. Також перелічує курси від таких постачальників, як консорціум OpenCourseWare, Saylor та багато інших.

Особливості:

CourseBuffet використовує власну класифікаційну систему для класифікації кожного курсу. Це означає, що він призначає предмет і рівень кожному курсу, подібному до того, що робиться у багатьох провідних університетах. Курси, що охоплюють той самий або подібний матеріал, матимуть однакову тему та номер курсу. Це дозволяє знати, які курси безпосередньо можна порівняти.

Класифікатор CourseBuffet складається з двох частин. Перша частина - це абревіатура предмета, друга - число, що вказує на рівень.

Число класифікатора, приблизно еквівалентно класифікації курсу в американському університеті. У американськиій системі курсу присвоюється номер на основі його рівня. Курси для першокурсників (1-й рік) складають 100, другий курс (2-й рік) - 200, третій курс (3-й рік) - 300, четвертий курс (4-й рік) - 400. Рівень аспірантури - 500. Курс до коледжу - 99 або нижче.

Приклад того, як працює ця система: “Вступ до мікроекономіки” називається Econ 101, оскільки, як правило, це курс 1-го курсу (або 2-го курсу). “Проміжну мікроекономіку” ми можемо назвати Econ 301, оскільки це, як правило, 3-й курс. Курси, яким присвоєно "0", не визначають рівня. Це, як правило, курси медицини або громадського здоров’я, які можуть не входити до звичайної класифікації.

CourseBuffet надає послугу Degree Paths. Це список курсів на спеціальність бакалавра. Сервіс використовує безкоштовні онлайн курси з сотень університетів, щоб повністю відтворити традиційний університетський ступінь.

Користувач має змогу зареєструватись на сервісі. Реєстрація можлива через профіль соціальної мережі Facebook, або за допомогою електронної адреси. Це дозволяє користувачеві відстежувати та зберігати цікаві йому курси, бачити пройдені курси та отримані Degree Paths.

На головній веб-сторінці агрегатора представлений пошук за ключовими словами, посилання на список категорій та посилання на реєстрацію для Degree Paths. Агрегатор надає класифікацію за 16 основними категоріями з багатьма підкатегоріями, наприклад Мистецтво та дизайн, Бізнес, Економіка, тощо. В описані курсів користувач бачить класифікатор CourseBuffet, назву курсу, короткий опис, теги, доступність та джерело курсу.

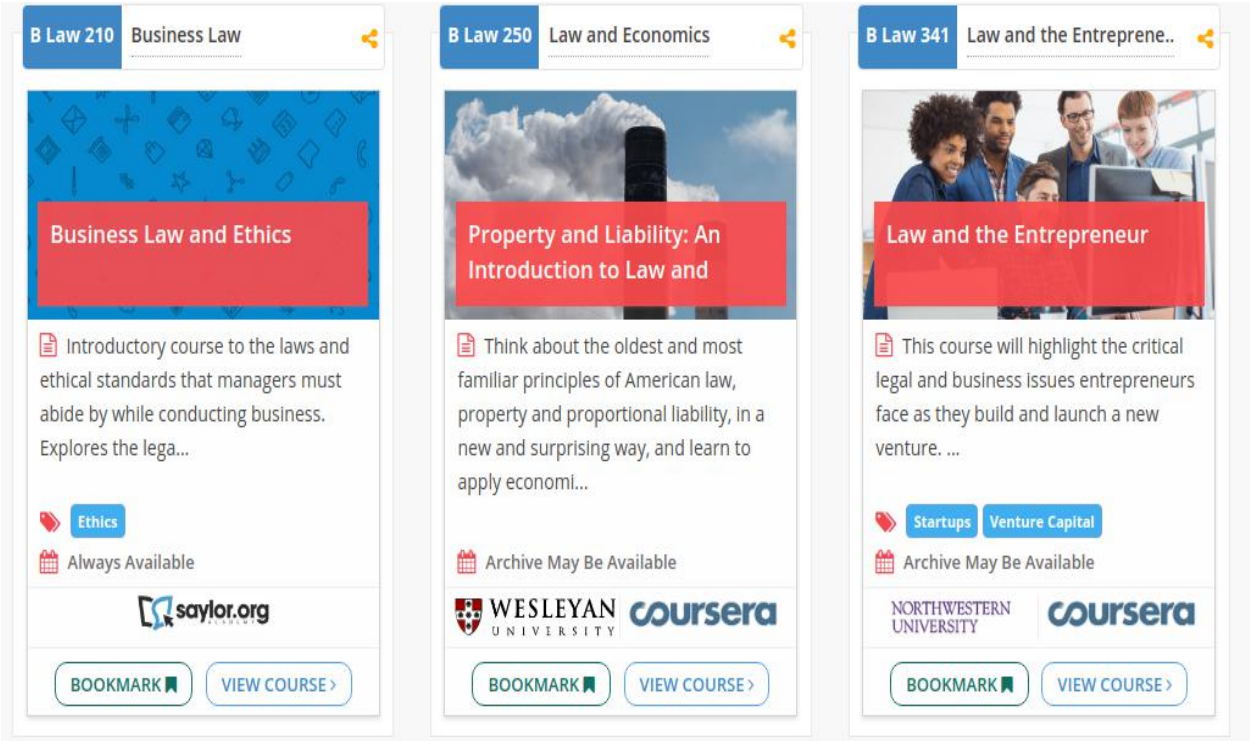


Рис. 1.5 – приклад списку курсів CourseBuffet

1.2.3 Class Central, classcentral.com

Class Central - це агрегатор онлайн-курсів переважно на англійській мові. Він збирає курси багатьох постачальників, щоб полегшити пошук найкращих курсів майже з будь-якої теми, де б вони не існували. Class Central зосереджений насамперед на безкоштовних (або безкоштовних для прослуховування) курсах університетів, що пропонуються на масових платформах відкритих онлайн-курсів (MOOC).

Через Class Central користувач може знайти курси; переглянути курси, які він пройшов; прочитати відгуки інших людей; стежити за університетами, предметами та курсами, щоб отримувати персоналізовані оновлення; а також планувати та відстежувати своє навчання. На веб-ресурсі зосереджено більше 30 тисяч курсів з більш ніж 900 університетів.

Дхавал Шах заснував Class Central наприкінці 2011 року, щоб відстежувати онлайн-класи різних університетів, які він хотів взяти. Кожен університет перераховує курси на своєму вебсайті, і Шах хотів одну сторінку, де він міг би відсортувати кожен курс за датою початку.

Особливості:

На головній сторінці користувачу доступне поле пошуку за ключовими словами та меню категорій.

Меню категорій містить список тринадцяти основних категорій (наприклад Програмування, Інженерія, Математика), кожна з яких має декілька підкатегорій (наприклад Програмування – Веб програмування, Мобільна розробка, Бази даних). Також в меню категорій доступні курси певного університету та список найкращих курсів за версією користувачів.

Користувачі можуть переглянути список курсів, які незабаром розпочнуться, або здійснити пошук курсів за ключовим словом або тематичною областю. Після того, як вони визначили курс, який вони мають пройти, вони можуть натиснути кнопку «Перейти до класу», яка спрямовує їх безпосередньо на вебсайт, де розміщений цей курс. Якщо користувачі хочуть зберегти інформацію про курси, які потрібно пройти пізніше, вони можуть

зареєструватися в обліковому записі Class Central. Реєстрація можлива за допомогою облікового запису Google або електроної скриньки. Зареєстровані користувачі зможуть створювати персоніфікований каталог курсів, організованих за рівнем інтересу, статусом завершення, темою та мовою курсу. Якщо користувачі додають до свого каталогу курси, які наразі не пропонуються, Class Central повідомить їх про повторне пропонування курсу. Користувачі також можуть оцінювати та переглядати курси, які вони пройшли, а також читати відгуки користувачів, щоб допомогти вирішити, які курси брати.

ВИСНОВКИ ДО РОЗДІЛУ 1

Огляд форм представлення контенту на вебсторінках показав, що зазвичай інформація представлена в HTML тегах. Однак все більше сайтів використовують JavaScript для взаємодії користувача з вебсторінкою, який динамічно завантажує додатковий контент у форматах HTML, XML, JSON.

Ознайомлення з популярними web-агрегаторами показало, що більшість з них має великий каталог доступних курсів, левова частка яких доступна лише англійською мовою або іншими іноземними мовами. Користувачеві який не володіє іноземними мовами на високому рівні, доступний набагато скудніший вибір. Варто зауважити, що на всіх проаналізованих агрегаторах відсутні курси провідних українських платформ: Prometheus, Edera, ВУМonline, Wisecow.

В цілому в ході вивчення можливостей, запропонованих даними агрегаторами можна виокремити наступні типові особливості:

- наявність системи авторизації користувача з її відповідними перевагами – наприклад створенням власних підбірок курсів з обраних джерел, перегляд завершених курсів
- зручний інтерфейс переліку курсів;
- можливість вибору способу відображення агрегованих курсів на сайті агрегатора, або перехід на сайт-джерело за бажанням;
- гнучкі фільтри агрегованого контенту.

Як наслідок, на етапах проєктування та розробки готового рішення, пріоритетною ціллю має бути реалізація наведених особливостей та надання курсів з провідних українських платформ.

РОЗДІЛ 2

ОБГРУНТОВАНИЙ ВИКЛАД РІШЕННЯ ПРОБЛЕМИ АГРЕГАЦІЇ ІНФОРМАЦІЇ З РІЗНИХ ДЖЕРЕЛ

Аналіз існуючих рішень агрегаторів веб-контенту дав зрозуміти, що найбільш оптимальним рішенням, як для розробника проекту, так і для потенційних користувачів, буде саме розробка агрегатора в якості веб-додатку. Сьогодні існує велика кількість різних сайтів агрегаторів в мережі і причиною цього являються переваги та зручність WEB розробки, а саме:

- легкий доступ на будь-яких платформах при наявності веб-браузера та підключення до мережі Інтернет
- доступу – користувачу потрібно лише ввести URL
- відсутність непотрібних завантажень чи встановлень на пристрій користувача
- завжди найактуальніша інформація та найсвіжіша версія застосунку без зусиль зі сторони користувача
- якість та швидкість розробки – сучасні фреймворки дозволяють розробнику не докладати зусиль для вирішення тривіальних проблем, розробник має більше часу для продумання бізнес-логіки, архітектури та взаємодії компонентів системи

Для описання організації архітектури найкраще підійде один з популярних об'єктно-орієнтованих програмних шаблонів ,метою яких являється можливість гнучкої розробки компонентів системи та простота підтримки подальшої роботи застосунку, прикладом може слугувати перехід на нову технологію, зміна однієї зі структурних одиниць програми і т.д. Зазвичай об'єктно-орієнтовані шаблони показують взаємодії і відносини між класами або об'єктами, без визначення того, які кінцеві класи або об'єкти додатку будуть використовуватися. В даному проекті буде реалізований архітектурних шаблон MVC (Model View Controller).

2.1 Огляд архітектурного шаблону MVC

Модель-вигляд-контролер, зазвичай відомий як MVC (англ. Model-view-controller) - це шаблон дизайну програмного забезпечення, який зазвичай використовується для розробки інтерфейсів користувача, що розділяє відповідну логіку програми на три взаємопов'язані елементи: модель, вигляд і контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно[6]. Концепція MVC була сформульована Трюгве Реенскаугом (Trygve Reenskaug) в результаті його роботи в Xerox PARC в 1978/79 роках. Як правило створення MVC пов'язують з мовою SmallTalk, але це не зовсім так. Насправді Реенскауг працював в групі, що займалася розробкою портативного комп'ютера "для дітей різного віку" Dynabook під керівництвом Алана Кея (Alan Kay).

Традиційно використовуваний для GUI інтерфейсів (графічних інтерфейсів користувачів ПК), цей шаблон набув популярність у проектуванні веб-додатків. Популярні мови програмування, такі як Python, JavaScript, Perl, Object Pascal / Delphi, PHP, Ruby, Java, C #, Swift та Elixir, мають фреймворки MVC, які використовуються для розробки веб або мобільних додатків прямо зараз.

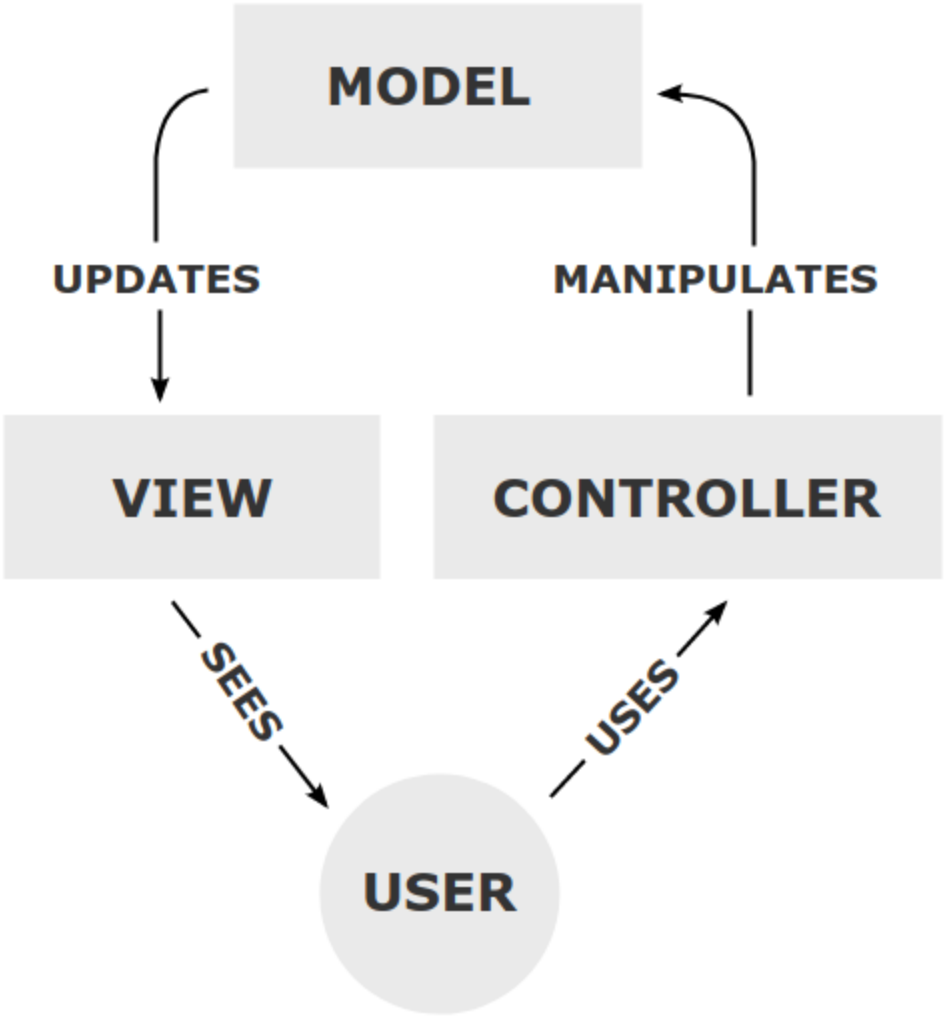


Рис. 2.1 – схематичне зображення взаємодії компонентів MVC [6]

Концепція MVC дозволяє розділити модель, вигляд і контролер на три окремих компоненти:

- Модель представляє собою дані і методи роботи з ними: запити до бази даних, перевірка на коректність. Модель безпосередньо керує даними, правилами та логікою програми, не залежно від вигляду (не знає як дані візуалізувати) і контролера (не має точок взаємодії з користувачем), лише надаючи доступ до даних і можливість управління ними. Будова моделі влаштована таким чином, шо вона змінює свій стан при відповіді на запити отримані від контролера.

- Вигляд відповідає за отримання та відправку необхідних даних користувачу. Дані він отримує з моделі. Вигляд не обробляє жодних даних що ввів користувач.
- Контролер перевіряє та направляє дані від користувача до системи і навпаки , забезпечуючи «зв'язок» між користувачем і системою. Для реалізацій необхідних дій Контролер використовує модель та її уявлення.

Оскільки MVC не має суворої реалізації, то реалізований він може бути по-різному. Немає загальноприйнятого визначення, де повинна розташовуватися бізнес-логіка[7]. Вона може знаходитися як в контролері, так і в моделі. В останньому випадку, модель буде містити всі бізнес-об'єкти з усіма даними і функціями.

Деякі фреймворки жорстко задають місце розташовування бізнес-логіки, інші не мають таких правил. Також не вказано, де повинна знаходитися перевірка введених користувачем даних. Проста валідація може зустрічатися навіть у вигляді, але частіше вони зустрічаються в контролері чи моделі.

Інтернаціоналізація і форматування даних також не має чітких вказівок по розташуванню.

Переваги MVC:

- Швидший процес розвитку: MVC підтримує швидкий і паралельний розвиток. За допомогою MVC один програміст може працювати над поданням, а інший - над контролером, щоб створити ділову логіку веб-програми. Додаток, розроблений за допомогою MVC, може бути втричі швидше реалізований, ніж додаток, розроблений з використанням інших патернів розробки.
- Можливість надання декількох виглядів: у моделі MVC ви можете створити кілька виглядів для моделі. Дублювання коду в

MVC зустрічається доволі рідко, оскільки шаблон відокремлює дані та ділову логіку від вигляду.

- Підтримка асинхронності: MVC також підтримує асинхронність яка дозволяє розробляти додаток, який завантажується дуже швидко.
- Простота модифікації: оскільки між моделями, виглядами чи контролерами відсутня велика зв'язність, це дозволяє простіше модифікувати додаток
- Модель MVC повертає дані без форматування: шаблон MVC повертає дані без застосування будь-якого форматування, тому ті самі компоненти можуть бути використані та викликані для використання з будь-яким інтерфейсом.
- Простіша розробка за допомогою TTD (розробка через тестування). За допомогою архітектури MVC можна легко розробити додаток, використовуючи Test Driven Development. Це допомагає писати та виконувати тести для окремих модулів.
- Легко підтримує будь-який плагін JavaScript. Шаблон може поєднуватися з будь-якою структурою JavaScript. Наприклад, ви можете поєднувати програму на Angular з MVC.
- Зіставлення URL-адрес. В шаблоні є потужні методи відображення та маршрутизації URL-адрес. Це корисно, коли ви зосереджуєтесь на оптимізації пошукових систем.

Недоліки MVC:

- Іноді це стає складною архітектурою для налагодження
- Неефективність доступу до даних з вигляду
- Потрібні знання з багатьох технологій
- Розробник повинен знати код на стороні клієнта та HTML код

2.2 Стуктура та схема вирішення проблеми розробки веб-додатку агрегатора онлайн курсів навчання

Для реалізації веб-додатку з використанням архітектурного шаблону MVC знадобиться Модель (представлена базою даних та сутностями класів), Вигляд (клієнтська частина, з нею буде взаємодіяти користувач) та Контролер (логіка взаємодії Вигляду та Моделі). В випадку додатка агрегатора існує також необхідність збору контенту з різних джерел мережі Інтернет з можливістю постійних оновлень, яку буде виконувати окремий модуль парсер реалізований за допомогою фреймворку Scrapy. Парсер буде періодично запускатись за допомогою планувальника, збирати дані та напряму зберігати їх в базі даних.

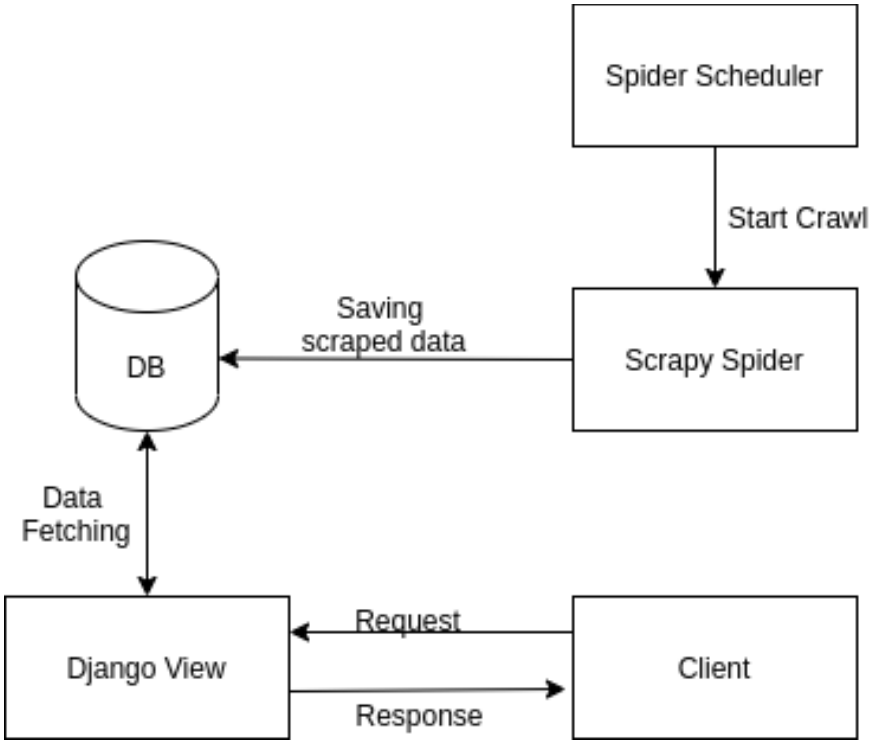


Рис. 2.2 — загальна схема пропонованого рішення

Приблизний алгоритм роботи додатку:

1. Планувальник запускає парсер при необхідності
2. Клієнт (Вигляд) надсилає запит з URL-адресою веб-сторінки

3. Контролер отримує потрібні дані з бази даних, після чого повертає їх Клієнту.
4. В залежності від Вигляду дані будуть представлені користувачу.

2.3 Реалізація шаблону MVC

Для створення додатку з функціоналом необхідним за вказівками шаблону MVC була вибрана мова Python3 та безкоштовний веб-фреймворк Django. Django - це високорівневий веб-фреймворк Python, який допомагає швидко розвивати додатки та створювати чистий, прагматичний дизайн. Фреймворк використовує архітектурний патерн Model Template View (MTV, укр. Модель-Шаблон-Вигляд), що по-суті являється аналогією MVC. Основна відмінність між MVC та MTV полягає в тому, що у шаблоні MVC розробник повинен написати весь код, який відповідає конкретному елементу управління, а в MVT про частину контролера дбає сам фреймворк. Вибір даного фреймворку можна обгрунтувати основною метою Django – полегшення реалізації та створення додатків керованих базами даних. Він визначає себе як веб-фреймворк, що включає повторне використання компонентів, з надійністю та простотою, що допомагає веб-розробникам писати чистий, ефективний та потужний код. Це один із найвідоміших веб-фреймворків у світі, а також один із найбільш часто використовуваних фреймворків. Також Django надає інтерфейс управління базою даних через панель адміністратора. Панель адміністратора Django генерується автоматично з коду Python. Завдяки стороннім програмам на адміністративній панелі Django є багато місця для налаштування. Крім того, Django дозволяє змінювати інтерфейс за допомогою сторонніх обгорток та додавати інформаційні унікальні панелі, для потреб розробника.

Django підтримує багато систем баз даних. SQLite дійсно хороший для тестування та розробки, оскільки його можна використовувати безпосередньо без необхідності встановлювати додаткове програмне забезпечення. Для “бойової” версії додатку можливо перейти на MySQL або

PostgreSQL, також веб-фреймворк підтримує бази даних NoSQL, наприклад розробник може використовувати MongoDB з Django.

З вбудованих компонентів Django варто приділити увагу шаблонам, моделям та вигляду.

Шаблони Django (Django Templates) - шаблонний шар використовується для відокремлення даних від способу їх фактичного представлення та перегляду користувачем[10]. Шаблонний шар подібний до рівня MVC View. Якщо ви знайомі з шаблонуванням на інших мовах, це приблизно те саме в Django; ви використовуєте синтаксис, подібний до HTML, який згодом компілюється в HTML із усіма відповідними введеними даними. Звичайно, існують формати для інших шаблонів, крім HTML, якщо ви хочете створювати XML-документи, файли JSON тощо. DRY - це один із основних принципів дизайну шаблонів Django, і це шаблон дизайну, що означає «Не повторюйся сам». Це саме те, що це звучить, це означає, що ви не повинні, принаймні в більшості випадків, копіювати та вставляти код. Натомість, ваш шаблон, наприклад, слід розділити на компоненти, що використовуються багаторазово, такі як бічна панель навігації, основна панель навігації, верхній колонтитул сторінки, нижній колонтитул сторінки тощо. Це мінімізує повторення та робить для написання ефективного та більш чистого коду.

Модель Django(Django Model) - моделі визначають структуру збережених даних, включаючи типи полів, їх максимальний розмір, значення за замовчуванням, параметри списку вибору, текст довідки для документації, текст мітки для форм тощо. Визначення моделі не залежить від типу бази даних – база даних задається як частина налаштувань проекту. Розробнику не потрібно спілкуватися з базою даних безпосередньо, він лише пише структуру своєї моделі та інший код, і Django виконує всю “брудну роботу” спілкування з базою даних.

Вигляд Django (Django View) - вигляд в Django - це рівень бізнес-логіки. Він відповідає за обробку запиту користувача та надсилання

правильної відповіді. Він отримує дані з моделі, надає кожному шаблону доступ до певних даних для відображення, або він може виконати певну обробку даних заздалегідь. На сьогодні подання Django можуть бути функціями, які обробляють запит і повертають відповідь, або можуть бути класами, здатними набагато більше, подібним чином до контролерів Laravel і Rails.

Маршрутизатор URL (URL router) - він у Django складніший, ніж в інших фреймворках, скажімо Rails або Laravel. Проблема в тому, що він використовує регулярні вирази, які непросто використовувати для початківців.

2.4 Реалізація модуля парсера вебсайтів

Оскільки “бек-енд” частина додатку написана на мові Python, що має в свому арсеналі велику кількість бібліотек та модулів для роботи з веб-сторінками та мережею, в тому числі і для парсингуінтернет сторінок, правильно було б використати одну з бібліотек мови Python для написання модуля парсера. Вибір пав на одну з найпопулярніших та найпотужніших python бібліотек для парсингу – Scrapy, фреймворк з відкритим кодом, створеним для вилучення потрібних даних із вебсайтів. Scrapy має надзвичайну кількість вбудованих компонентів та функціональностей, що полегшують роботу та не відволікають розробника рутиними та тривіальними проблемами.

Огляд компонентів Scrapy:

- Двигун (Scrapy Engine) - механізм що відповідає за контроль потоку даних між усіма компонентами системи та ініціює події, коли відбуваються певні дії.
- Планувальник (Scheduler) - планувальник отримує запити від двигуна і ставить їх у чергу для подальшої подачі (також до двигуна), коли двигун запитує їх.

- Завантажувач (Downloader) - відповідає за завантаження веб-сторінок та їх подачу в двигун, який, у свою чергу, подає їх павукам.
- Павуки (Spiders) - це власні класи, написані користувачами Scrapy для аналізу завантажених веб-сторінок та вилучення з них даних або додаткових запитів на виконання.
- Конвеєр даних(Item Pipeline) - відповідає за обробку даних після того, як їх витягнули (спарсили) павуки. Типові завдання конвеєру включають очищення даних, зберігання в базі даних або файлі, перевірка даних, тощо.
- Посередник завантажувача (Downloader middlewares) - це специфічні хуки, які розташовані між двигуном і завантажувачем і обробляють запити, коли вони переходять з двигуна на завантажувач, і відповіді, які передаються із завантажувача на двигун. Застосовуються для обробки запитів безпосередньо перед відправленням у Завантажувач (тобто безпосередньо перед тим, як Scrapy надсилає запит на вебсайт), зміни отриманої відповіді перед передачею її павуку, надсилання нового запиту замість передачі отриманої відповіді павуку, і т.д.

Scrapy написано за допомогою Twisted - популярного мережевого фреймворку для Python, керованого подіями. Таким чином, це реалізовано з використанням асинхронний коду для паралельності, що надає фреймворку неймовірну перевагу в порівнянні з іншими. Завдяки асинхронності Scrapy має можливість дуже швидко обробляти дані та відправляти нові запити.

Архітектура Scrapy зав’язана навколо павуків, які являються автономними скраперами зі набором власних інструкцій. Аналогічно до Django, Scrapy також побудований за принципом DRY, що дозволило розробникам фреймворку реалізувати зручну архітектуру для створення користувачами Scrapy масштабних проектів з сотнями парсерів та мінімальним повторенням коду. Scrapy також забезпечує shell оболнку для

веб-сканування, яку розробники можуть використовувати для перевірки та аналізу поведінки сайту.

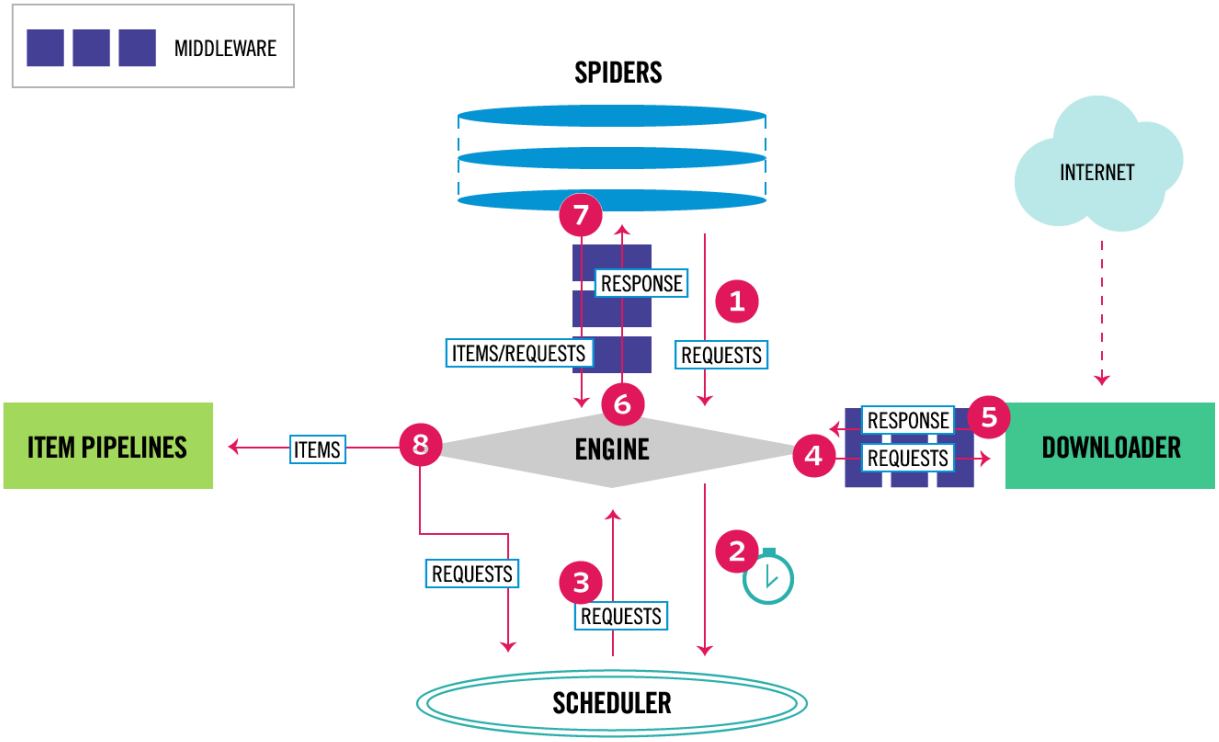


Рис. 2.3 — Схема потоку даних в Scrapy [11]

Scrapy також забезпечує контроль над швидкістю сканування за допомогою декількох налаштувань. Можливо виконувати такі дії як задання кількості одночасних запитів на IP-адресу або на домен, встановлення затримки завантаження між кожним запитом та рандомізація затримки, і навіть використовування розширення AutoThrottle, яке динамічно виставляє обмеження на кількість одночасних запитів та затримку між запитами базуючись аналізі завантаженості сервера.

Scrapy фреймворк надає велику кількість потужних функцій для полегшення та більш ефективного парсингу, таких як:

- підтримка “з коробки” вибору та вилучення даних із джерел HTML / XML за допомогою Xpath виразів та розширених селекторів CSS;

- наявність допоміжних методів для вилучення даних за допомогою регулярних виразів;
- інтерактивна консоль оболонки (IPython shell) для випробування виразів CSS та XPath для витягування даних, дуже корисна під час написання або налагодження павуків.
- вбудована підтримка для генерації експорту даних у форматах (XML, JSON, CSV) та зберігання їх у декількох серверних базах (S3, FTP, локальна файлова система)
- потужна підтримка розширюваності, що дозволяє підключити власну функціональність за допомогою сигналів та чітко визначеного API (проміжні програми, розширення та конвеєри).
- широкий асортимент вбудованих розширень та проміжних засобів для обробки (наприклад заміна заголовка User-Agent, файли cookie та обробка сеансів, функції HTTP, такі як автентифікація, стиснення, кешування, керування глибиною сканування вебсайту, аналіз та робота з файлом robots.txt)
- консоль Telnet для підключення до консолі Python для аналізу та налагодження скрапера
- інші особливості, такі як канали мультимедіа для автоматичного завантаження зображень (або будь-якого іншого мультимедіа), пов'язаного зі скребковими елементами, павуки багаторазового використання для сканування сайтів із файлів Sitemaps та XML / CSV, кешуючий DNS вирішувач і т.д.
- сигнали scrapy для сповіщення про певні події, наприклад сигнал запуску/завершення scrapy engine, перехід павука у режим idle (що означає що у павука більше немає запитів та ітемів для обробки)
- вбудована обробка Sitemap (мап сайтів).

Мапи сайту - це простий спосіб для веб-майстрів інформувати пошукові системи про сторінки на своїх сайтах, доступні для сканування[14].

У найпростішій формі файл Sitemap - це XML-файл, в якому перелічені URL-адреси вебсайту, а також додаткові метадані про кожну URL-адресу (коли вона востаннє оновлювалась, як часто вона зазвичай змінюється та наскільки вона важлива щодо інших URL-адрес на сайті), щоб пошукові системи могли якісніше сканувати сайт. Веб-сканери зазвичай виявляють сторінки за посиланнями на вебсайті та з інших сайтів. Мапи сайтів доповнюють ці дані, щоб дозволити сканерам, які підтримують мапи сайтів, підібрати всі URL-адреси на мапі сайту та дізнатися про ці URL-адреси за допомогою пов'язаних метаданих. Використання протоколу Sitemap не гарантує включення веб-сторінок у пошукові системи, але дає підказки веб-сканерам щодо кращого сканування вашого сайту.

Для зручності керування павуками та збору логів запуск павуків буде виконуватись за допомогою додатку Scrapyd. Scrapyd – додаток що прослуховує запити для запуску парсерів-павуків і створює процес для кожного з павуків при отриманні відповідного запиту. Зазвичай додаток запускається на окремому порту сервера у фоновому режимі. Scrapyd постачається з мінімалістичним веб-інтерфейсом, який значно спрощує розробку проекту, для моніторингу запущених павуків, доступу до логів та планування нових запусків. Доступ до веб-інтерфейсу можна отримати за адресою http://localhost:6800, за бажанням порт можна змінити на будь-який вільний.

Керувати періодичним запуском павуків буде спеціальний павук-планувальник, який за допомогою idle режиму та виклику винятку DontCloseSpider не буде завершуватись та з визначеною періодичністю надсилатиме запит до scrapyd на запуск відповідних павуків.

Зберігати спаршені дані та взаємодіяти з Моделю Django буде так званий конвеєр ("Pipepline"). Після того, як дані спарсив павук, вони надсилаються на до конвеєра, який обробляє їх через кілька послідовних функцій[13]. Зазвичай конвеєр представляють як клас Python, який реалізує прості методи. Вони отримують предмет і виконують над ним дію, також

вирішуючи, чи повинен він продовжувати рух по конвеєру, або його можна “викинути” і більше не обробляти.

2.5 Підходи до розробки клієнтського інтерфейсу

Добре розроблений клієнтський інтерфейс надзвичайно важливий, адже він забезпечує зручність використання програми. В більшості інтерактивних систем використовується посібник Шнайдермана щодо гарного дизайну - «Вісім золотих правил дизайну інтерфейсу». Шнайдерман запропонував цей набір принципів ще у 1985 році. Якоб Нільсен, Джефф Джонсон та інші розширили ці правила та включили їхні варіації. Сьогодні використовується вже шоста версія посібника.

Ось перелік правил[15]:

- Прагніть до послідовності. У подібних ситуаціях слід вимагати послідовності дій; слід використовувати однакову термінологію в підказках, меню та екранах довідки; сумісні кольори, макети, використання великих літер, шрифтів тощо, слід застосовувати на всьому протязі. Винятки, такі як необхідне підтвердження команди видалення або відсутність повторення паролів, повинні бути зрозумілими та обмеженими за кількістю
- Шукайте універсальність та зручність використання. Визнати потреби різноманітних користувачів та створити пластичний дизайн, полегшуючи трансформацію вмісту. Дизайн повинен задоволняти початківців та експертів, незамежно від віку, інвалідності, міжнародних відмінностей та технологічного різноманіття кожен із них збагачує спектр вимог, якими керується дизайн. Додавання функцій для початківців, таких як пояснення, та функцій для експертів, таких як ярлики, збагачує дизайн інтерфейсу та покращує його якість
- Запропонуйте інформативні відгуки. Для кожної дії користувача повинен бути інтерфейс зворотнього зв'язку. На часті та незначні дії

реакція може бути скромною, тоді як на рідкісні та серйозні дії реакція повинна бути більш суттєвою.

- Створіть діалогові вікна про завершення роботи. Послідовності дій слід організовувати в групи з початком, серединою та кінцем. Інформативний зворотний зв'язок при завершенні групи дій дає користувачам задоволення від досягнень, відчуття полегшення та показник для підготовки до наступної групи дій. Наприклад, вебсайти електронної комерції переходять користувачів від вибору продуктів до каси, закінчуючи чіткою сторінкою підтвердження, яка завершує транзакцію.
- Запобігайте помилкам. Наскільки це можливо, розробляйте інтерфейс так, щоб користувачі не могли робити серйозних помилок; наприклад, звиділення сірим кольором невідповідних пунктів меню і заборона на використання буквених символів в полях введення цифр. Якщо користувачі припускаються помилки, інтерфейс повинен пропонувати прості, конструктивні і конкретні інструкції по відновленню. Наприклад, користувачі не повинні заново набирати всю форму з ім'ям і адресою, якщо вони ввели невірний поштовий індекс, а повинні бути спрямовані на виправлення тільки несправної частини. При помилкових діях стан інтерфейсу має залишатися незмінним, або інтерфейс повинен давати інструкції по відновленню стану.
- Дозволяти легко скасовувати дії. Наскільки це можливо, дії повинні бути оборотними. Ця функція знімає тривогу, оскільки користувачі знають, що помилки можна скасувати, і заохочує вивчення незнайомих варіантів. Одиницями оборотності може бути окрема дія або ціла група дій.
- Тримайте користувачів під контролем. Досвідчені користувачі дуже хочуть відчути, що вони відповідають за інтерфейс і що інтерфейс реагує на їх дії. Вони не хочуть сюрпризів чи змін у звичній поведінці, і

їх дратують нудні послідовності введення даних, труднощі в отриманні необхідної інформації та неможливість отримати бажаний результат.

- Зменшіть короткочасне навантаження на пам’ять користувача. Обмежена здатність людей обробляти інформацію в короткочасній пам’яті вимагає від дизайнерів уникати інтерфейсів, в яких користувачі повинні запам’ятовувати інформацію з одного дисплея, а потім використовувати цю інформацію на іншому дисплеї. Це означає, що мобільні телефони не повинні вимагати повторного введення телефонних номерів, розташування вебсайтів повинні залишатися видимими, а довгі форми слід ущільнювати, щоб вмістити в один дисплей.

Ці основні принципи необхідно тлумачити, вдосконалювати та розширювати для кожного середовища. У них є свої обмеження, але вони є хорошою відправною точкою для мобільних, десктоп та веб-дизайнерів.

Правила для дизайнерів та розробників інтерфейсів, згадані вище, досить непогано висвітлюють ці три ключові компоненти досвіду користувача - зручності, корисності та бажаності, а це означає, що логічно було б використати їх при розробці клієнтського інтерфейсу застосунку.

2.6 Вибір технологічної бази для розробки користувацького інтерфейсу

За інтерфейс клієнта в контексті веб-розробки з використанням шаблону MVC відповідає Вигляд (View). Вигляд також відомий як фронтенд (з англ. “front-end” — “передня частина”). Найпоширенішими інструментами для розробки фронтенду є HTML, CSS та JavaScript.

2.6.1 HTML

Hyper Text Markup Language (HTML) – основа будь-якого вебсайту, без якого існування сайту не можливе. Мова розмітки вказує браузеру, на що

саме перетворити текст, будь то зображення, таблиці, посилання чи інші подання.

В контексті розробки фронтенду HTML зазвичай генерується шаблонно. Web-фреймворк Django, забезпечує цілу мову для спрощення переходу від серверних даних до їх відображення у браузерному вікні. Django визначає стандартний API для незалежного від бекенду завантаження та візуалізації шаблонів.

Системи шаблонів дозволяють вказати структуру вихідного документа, використовуючи наповнювачі для даних, які будуть вставлені при генеруванні сторінки. Шаблони часто використовуються для створення HTML, але також можуть створювати інші типи документів.

```
{% extends "base_generic.html" %}

{% block title %}{{ section.title }}{% endblock %}

{% block content %}
<h1>{{ section.title }}</h1>

{% for story in story_list %}
<h2>
  <a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a>
</h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock %}
```

Рис. 2.4 — приклад файлу з Django шаблоном

Шаблон - це текстовий файл. Він може генерувати будь-який текстовий формат (HTML, XML, CSV тощо). Шаблон містить змінні, які замінюються значеннями при оцінці шаблону, і теги, які контролюють логіку шаблону. Окрім цього, в шаблоні підтримується фільтрація та доступ до атрибутів об’єктів. Загалом для статичних сторінок (в контексті сайту агрегатора курсів навчання) функціоналу шаблонів Django має вистачати.

2.6.2 CSS

Каскадні таблиці стилів (CSS) - це мова таблиць стилів, яка використовується для опису представлення документа, написаного на мові розмітки, такому як HTML. CSS є одним із стовпів технології World Wide Web, поряд з HTML і JavaScript [16].

CSS розроблений для відокремлення подання від змісту, включаючи макет, кольори і шрифти. Таке відокремлення може поліпшити доступність змісту, забезпечити більшу гнучкість і контроль у визначенні характеристик уявлення, дозволити кільком веб-сторінок спільно використовувати форматування шляхом вказівки відповідного CSS в окремому .css файлі, що зменшує складність і повторення структурного змісту, а також дозволяє кешувати .css файл для поліпшення швидкості завантаження сторінкок, які спільно використовують цей файл і його форматування.

```
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
```

Рис. 2.5 — приклад CSS таблиці стилів

CSS файли в контексті роботи з фреймворком Django зазвичай розташовуються в окремій директорії (наприклад static) та довантажуються до HTML шаблонів за допомогою спеціальної команди {% load static %}.

2.6.3 JavaScript

JavaScript - імперативна мова програмування на основі подій, яка використовується для перетворення статичної HTML сторінки в динамічний інтерфейс. Використовуючи техніку веб-розробки AJAX (англ. “Asynchronous JavaScript and XML” - “Асинхронний JavaScript та XML”), JavaScript код також може активно отримувати дані з Інтернету (без перезавантаження сторінки, незалежно від початкового отримання HTML-сторінок), а також реагувати на події зі сторони сервера, що робить веб-сторінки надзвичайно динамічними. Файли з JavaScript кодом в ієрархії проекту Django, в залежності від їх розміру, розміщуються або в шаблонах або окремими файлами в папці static, в такому випадку, принцип їх завантаження аналогічний довантаженню файлів CSS. Адміністраторська система Django використовує фреймворк jQuery для збільшення можливостей інтерфейсу адміністратора. Потрібно зауважити, що Django надає можливість тестування javascript скриптів у консолі браузера чи у командному рядку.

Велика кількість вебсайтів використовують JavaScript бібліотеки або веб-фреймворки для розробки та запуску скриптів на сторінках клієнтів. Серед бібліотек для розробки виділяється jQuery (яку використовує django) – найпопулярніша бібліотека, майже 7 сайтів з 10 використовують її[17], що беззаперечно доводить її попелярність, чого не скажеш про такі бібліотеки як Prototype JavaScript Framework чи MooTools. На основі jQuery також базується більшість JavaScript та CSS фреймворків для веброзробки, наприклад React та Bootstrap. Варто зауважити, при розробці клієнтського інтерфейсу використання бібліотеки jQuery в комбінації з HTML5 та CSS є дуже доречним, особливо при застосуванні підходів та рекомендацій з пункту 2.5.

ВИСНОВКИ ДО РОЗДІЛУ 2

В ході аналізу рішення проблеми побудови агрегатора курсів онлайн навчання на базі web-додатку стало зрозуміло, що найбільш зручним по ряду причин способом реалізації необхідних можливостей застосунку являється шаблон проєктування MVC. Слід зауважити що за загальноприйнятою практикою в веб-розробці сайт складатиметься з двох окремих частин – клієнтської та серверної. Для розробки серверної частини була обрана мова Python та її фреймфорк Django. Модуль-парсер буде реалізований також на Python з використатнням бібліотеки з відкритим кодом Scrapy, одна з переваг котрої зручна інтеграція в екосистему Django.

При розробці клієнтської частини, слід користуватись “золотими” правилами дизайну інтерфейсів, та використовувати можливості, бібліотеки та фреймворки мови JavaScript для створення зручного та динамічного інтерфейсу користувача.

РОЗДІЛ 3

ОПИС РОЗРОБКИ ВЕБ-ДОДАТКУ КУРСІВ НАВЧАННЯ

В даному розділі помодульно показана розробка веб-додатку для агрегації курсів онлайн навчання з різних ресурсів з використанням python фреймворку Django. Основним принципом функіонування агрегатора є регулярне наповнення БД актуальними курсами. Спеціально написані парсери для кожного окремого ресурсу періодично збирають та зберігають дані в БД. Далі за запитом користувача веб-додатку збережені в БД дані можна відобразити в зручному вигляді.

3.1 Загальна структура проєкту

Як будь-який звичайний веб сайт, веб-додаток агрегатор складається з серверної та клієнтської частини. Для злагодженої роботи додатку також потребується окремий сервер для модуля парсера. Наступна структурна схема системи представляє більш деталізовану версію рис. 2.2.

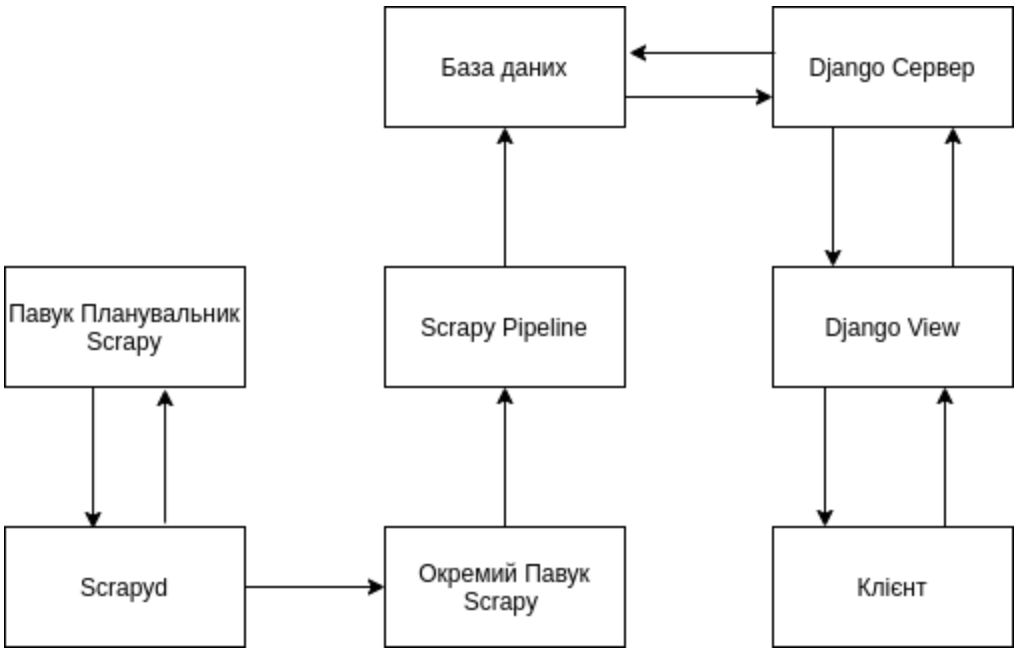


Рис. 3.1 — структурна схема системи

Варто зазначити, що алгоритм роботи системи майже повністю співвідноситься з теоретичним алгоритмом, що був поданий в другому пункті другого розділу. Умовно систему можна розділити на дві окремі частини які зв'язує база даних – це модуль парсер та веб-додаток.

Схема дії модуля парсера досить тривіальна.

1. Павук планувальник запущений у scrapyd в режимі idle (що запобігає завершенню процесу павука) надсилає запит до scrapyd на запуск відповідних павуків;
2. Scrapyd при отримані запиту запускає процес павука;
3. Павук згідно зі своєю логікою збирає дані та відправляє їх до пайплайну;
4. Пайплайн зберігає дані у БД.

Алгоритм роботи веб-додатку наступний:

1. Клієнт заходить на сторінку каталог курсів;
2. відновідний Django View робить запит на сервер з усіма фільтрами що зазначив користувач;
3. сервер отримує відфільтровані курси з БД;
4. сервер повертає до Django View колекцію об'єктів курсів
5. Django View генерує шаблон для відображення html сторінки для клієнта.

Слід зазначити, що актуальність оновлень даних в базі може бути поганою при задані занадто великого проміжку часу між оновленнями. Адже "свіжість" збережених даних буде в межах заданого вікна графіку, як рішення вікно можна зробити доволі малим, однак варто мати на увазі, що при збільшенні кількості джерел та зменшенні часу між запусками парсерів збільшується і навантаження як на модуль-парсер так і на базу даних, тому слід заздалегідь подбати про поступовий (каскадний) запуск павуків та

оновлення інформації в базі. Як видно зі алгоритмів вказаних вище, основні операції системи послідовні та прозорі. В наступному пункті будуть викладені деталі та тонкощі реалізації кожного з елементів системи.

3.2 Модель данних

Основною ланкою в базі є Course – представлення курсу. Кожен курс зв'язаний з SourceSite – сайтом джерелом за допомогою зовнішнього ключа. Зв'язки продемонстровані на рис. 3.2

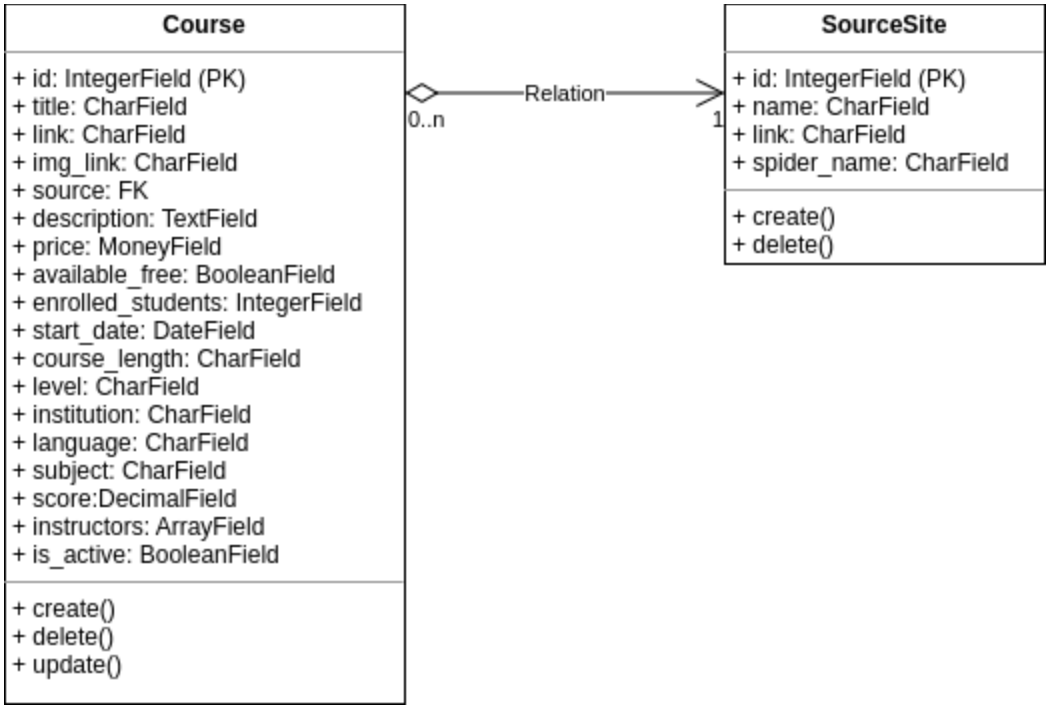


Рис. 3.2 — UML діаграма класів моделі django

Приклад задавання Django моделі в вигляді класу для Course. Методи такі як геттери та сеттери для атрибутів, а також методи save(), update() та delete() не вказані, адже вони реалізовані у класі django.db.models.Model за замовчуванням та наслідуються класом Course. Також django.db.models.Model за замовчуванням використовує поле id в якості первинного ключа, тому він також не вказаний у класі Course. За допомогою атрибуту unique_together класу Meta визначається список полів моделі що визначають унікальність кожного запису в базі.

Метод __str__ визначає представлення екземпляру даних при приведені його до типу str, наприклад щоб вивести його на екран за допомогою функції print().

```
class Course(models.Model):
    title = models.CharField(max_length=256)
    link = models.CharField(max_length=256)
    img_link = models.CharField(max_length=512)
    source = models.ForeignKey(SourceSite, on_delete=models.CASCADE)
    description = models.TextField()
    price = MoneyField(max_digits=8, decimal_places=2, null=True, default_currency=None)
    available_free = models.BooleanField(default=True)
    enrolled_students = models.IntegerField(null=True)
    start_date = models.DateField()
    course_length = models.CharField(max_length=256, null=True)
    level = models.CharField(max_length=256, null=True)
    institution = models.CharField(max_length=256, null=True)
    language = models.CharField(max_length=256, null=True)
    subject = models.CharField(max_length=256, null=True)
    score = models.DecimalField(max_digits=2, decimal_places=1, null=True)
    instructors = ArrayField(JSONField(null=False), null=True)
    is_active = models.BooleanField(default=True)

    class Meta:
        unique_together = ('title', 'source', 'language')

    def __str__(self):
        return f'{self.source.name} | {self.title}'
```

Рис. 3.3 — клас моделі Course

3.3 Модуль-парсер

Кожен ресурс з якого береться інформація про курси навчання потребує власного павука Scrapy. В даному проєкті написані 4 павука для таких ресурсів: coursera.org, edx.org, prometheus.org, vumonline.ua. Для прикладу буде розібраний алгоритм павука для coursera.org, інші павуки мають схожий принцип. На даному ресурсі є так звана мапа сайту – sitemap в якій вказані адреси існуючих веб сторінок для полегшення роботи парсерів та кращої індексації сайту роботами веб-пошуковиків. Вона буде використовуватись парсером та для цього в Scrapy є спеціальний клас SitemapSpider що дозволяє наслідуватись від нього та в якому реалізована логіка витягування посилань з мап.

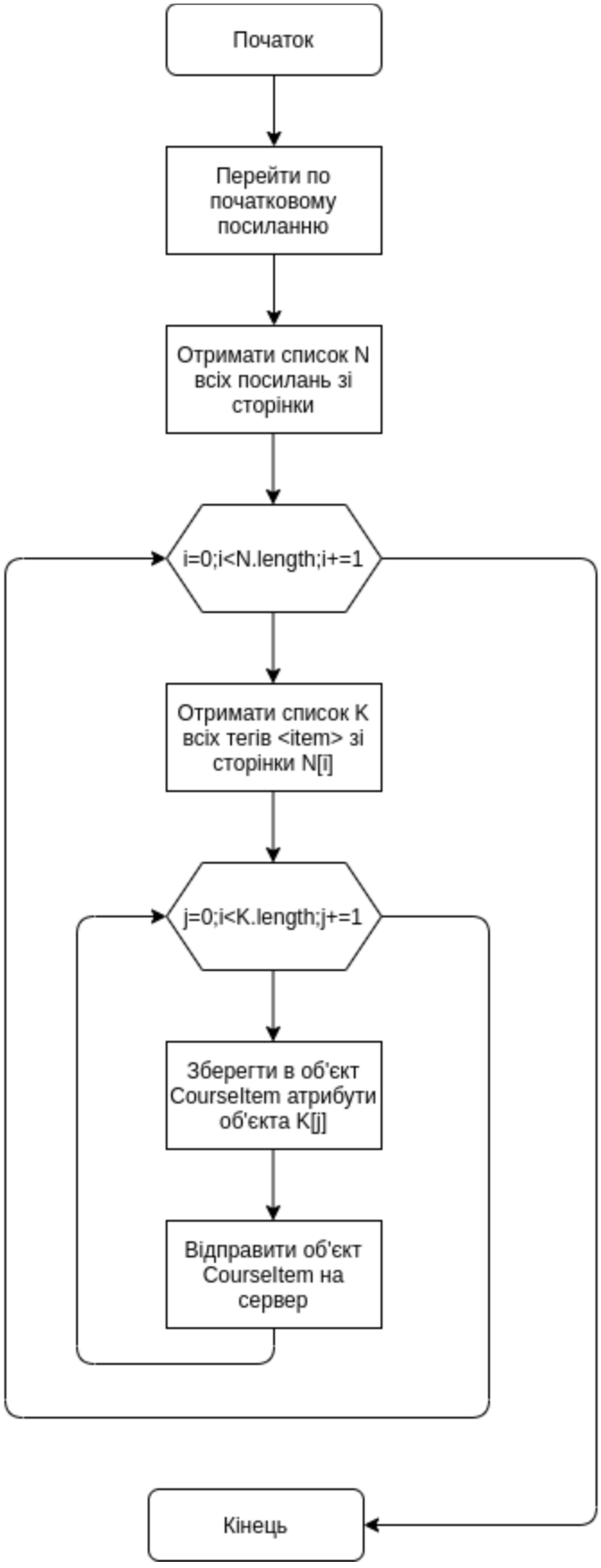


Рис. 3.4 — алгоритм роботи модуля-парсера

У SitemapSpider вже вбудовані стартовий метод, який являється обов'язковим для кожного павука, start_requests (зображений на рисунку 3.5) та обробник респонсу з вебсайту який повертається на запит від метода start_requests – метод _parse_sitemap.

```
def start_requests(self):
    for url in self.sitemap_urls:
        yield Request(url, self._parse_sitemap)
```

Рис. 3.5 — метод start_requests класа SitemapSpider

Метод обробник керуючись спеціальними атрибутами класу sitemap_urls та sitemap_rules дістає з тіла http відповіді посилання, які можуть бути посиланнями на наступні мапи або ж посиланнями на звичайні html сторінки вебсайту. sitemap_urls задає список посилань з яких спайдер розпочне роботу, тоді як в sitemap_rules – список правил для визначення методу який викликається для обробки респонсу з урлом що відповідає правилу.

На рисунку 3.6 видно як вказуються правила та початкові адреси для павука.

```
class CourseraSpider(SitemapSpider):
    name = 'coursera'

    custom_settings = {...}

    sitemap_urls = ('https://www.coursera.org/sitemap~www~courses.xml',)

    sitemap_rules = [('/learn/', 'parse_course')]
```

Рис. 3.6 — частина коду класа CourseraSpider

В методі parse_course відбувається вибірка потрібних даних та приведення їх до відповідного формату. Результатом роботи метода є об'єкт

CourseItem що відправляється далі до так званого Pipeline – конвеєра, який збереже його до бази даних.

```
yield CourseItem({
    'source_id': self.source_id,
    'title': response.xpath('//h1/text()').get(),
    'link': response.url,
    'img_link': response.xpath('//div[@class="_1m6egy8n"]//img/@src').get(),
    'description': response.xpath('//div[@class="m-t-1 description"]//p/text()').get(),
    'price': None,
    'available_free': True,
    'enrolled_students': enrolled_students,
    'start_date': timezone.now().date(),
    'course_length': course_length,
    'institution': response.xpath('//div[@class="p-b-1s p-r-1"]//@alt').get(),
    'language': self.languages_from_list(primary_langs, subtitle_langs),
    'level': level[0].lower(),
    'score': response.xpath('//span[@class="_16ni8zai m-b-0 rating-text number-rating number-rating-expertise"]/text()').get(),
    'instructors': self.instructors(response),
    'subject': response.xpath('//div[@aria-current="page"]/a/text()').get(),
})
```

Рис. 3.7 — вигляд об’єкта CourseItem перед відправленням до конвеєра

На рисунку 3.7 ключами словника що передається в CourseItem є назви полів моделі Course. Фактично CourseItem являє собою оболонку python словника зі списоком полів що він може прийняти. Конструкція response.xpath(...).get() повертає відповідний зміст з тегів (параметр методу xpath) у HTML-дереві сторінки, що парситься.

```
def process_item(self, item, spider):
    if isinstance(item, BaseItem):
        _instance = self.create_db_instance(item)
        _instance.save()
    return item

def create_db_instance(self, item):
    instance_data = {}
    for key in item.fields.keys():
        instance_data[key] = item[key]
    return self._db_model.objects.update_or_create(is_active=True, defaults=instance_data)
```

Рис. 3.8 — методи конвеєра для збереження елемента до бази даних

Функція process_item на рис. 3.8 створює екземпляр даних моделі Course з об’єкта CourseItem, що віддається парсером, та зберігає новостворений екземпляр до бази даних.

За регулярний запуск павуків відповідає спеціальний павук працюючий у режимі idle. Метод spider_idle павука приведений на рис. 3.9, він викликається кожні 3 секунди та перевіряє кількість часу з останнього запуска павуків. Якщо кількість годин перевищує атрибут hours_period, викликається метод що надсилає запити до scrapyd на запуск павуків. Запуск відбувається завдяки python бібліотеці Scrapyd-API, яка дає можливість взаємодіяти з додатком scrapyd напряму з коду павука.

```
def spider_idle(self):
    if not self.last_run or self._hours_from_last_run() >= self.hours_period:
        self.schedule_all_spiders()
        self.last_run = datetime.now()

    raise DontCloseSpider
```

Рис. 3.9 — метод павука планувальника котрий запускає інших павуків

3.4 Django Views та проектування інтерфейсу користувача

Приділяючи увагу порадами-правилам розробки дизайну інтерфейсів, що були викладені в пункті 2.5 була обрана наступня структура:

- головна сторінка з полем для пошуку та декількома найкращими курсами за рейтингом користувачів цих курсів
- сторінка каталогу курсів з фільтрами прапорцями зліва від списку курсів та пагінацією знизу сторінки.

Така структура сайту заслужено вважають стандартом та вона добре знайома кожному користувачу мережі.

```python
class HomePageView(TemplateView):
    template_name = 'index.html'

    def get(self, request, *args, **kwargs):
        context = self.get_context_data(**kwargs)
        return self.render_to_response(context)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['best_courses'] = Course.objects.filter(score__isnull=False,
                                                        institution__isnull=False,
                                                        is_active=True).order_by('-score')[:12]
        return context
```

Рис. 3.9 — клас HomePageView, що відповідає за повернення курсів користувачу на домашній сторінці ресурсу

Метод get класу HomePageView викликає відповідає на запит користувача генеруючи контекст та повертаючи метод render_to_response, що повертає HttpResponce — відповідь, що була побудована згідно шаблону, визначеного атрибутом template_name (на рис 3.9 це “index.html”) зі згенерованим контекстом в якості параметру. Метод генеруючий контекст – get_context_data повертає відфільтровані об’єкти з бази даних за допомогою менеджера моделі Course в вигляді словника. Важливим моментом в роботі з методом get_context_data є необхідність викликати цей метод у батьківському класі, для об’єднання даних контексту всіх батьківських класів з даними поточног класу.

```
<body>
<h1>Search Results</h1>

<ul>
    {% for course in object_list %}
        <li>
            {{ course.source.name }} <a href={{ course.link }}>{{ course.title }}</a>
        </li>
    {% endfor %}
</ul>
</body>
</html>
```

Рис. 3.10 — тіло шаблону сторінки search.html

На рисунку 3.10 можна побачити тіло однієї зі сторінок, а саме тіло сторінки з результатом пошуку за словом.

```python
class CatalogListView(ListView):
    model = Course
    template_name = 'class_catalog.html'
    paginate_by = 30

    def get_queryset(self):
        object_list = self.model.objects.filter(is_active=True)

        sources = self.request.GET.getlist('source')
        if sources:
            object_list = object_list.filter(source__name__in=sources)

        levels = self.request.GET.getlist('level')
        if levels:
            levels = [i.lower() for i in levels]
            object_list = object_list.filter(level__in=levels)

        free = self.request.GET.get('free')
        if free:
            object_list = object_list.filter(available_free=bool(int(free)))

        subjects = self.request.GET.getlist('subject')
        if subjects:
            object_list = object_list.filter(subject__in=subjects)

        query = self.request.GET.get('q')
        if query:
            object_list = object_list.filter(
                Q(title__icontains=query) | Q(source__name__icontains=query)
            )

        return object_list

    def get_context_data(self, **kwargs):
        ctx = super().get_context_data(**kwargs)
        ctx['sources'] = [i['source__name'] for i in self.model.objects.values('source__name').distinct()]

        levels = self.model.objects.filter(level__isnull=False).values('level').distinct()
        ctx['levels'] = [i['level'].capitalize() for i in levels if i]

        subjects = self.model.objects.filter(subject__isnull=False).values('subject').distinct()
        ctx['subjects'] = [i['subject'] for i in subjects]

        ctx['courses_count'] = self.model.objects.all().count()
        ctx['courses_show_count'] = self.get_queryset().all().count()
        return ctx
```

Рис. 3.11 — клас CatalogListView

На рисунку 3.11 зображений клас CatalogListView представляючий Контролер з шаблону MVC. Клас наслідується від базового ListView бібліотеки Django, який має вбудовані методи пагінації, та рендерингу відповіді. Метод get_context_data повертає python словник з набором атрибутів, які будуть використані шаблоном як фільтри доступні для користувача. Після вибору фільтрів користувачем та відправлення запиту, список відфільтрованих курсів поверне метод get_query_set, звертаючись напряму до менеджера моделі бази даних. Фільтри передаються в якості параметрів url адреси запиту та доступні завдяки методам get та getlist атрибута requests.GET.

ВИСНОВКИ ДО РОЗДІЛУ 3

В данному розділі були представлені деталі розробки web-додатку агрегатора курсів онлайн навчання з вибірковими прикладами. Була продемонстрована пара типових сценаріїв використання системи.

В результаті проведеної роботи щодо розробки веб-додатку агрегатора можна виокремити наступне:

- надано структурну схему роботи системи з детальним описом (див. пункт 3.1)
- показано UML діаграму класів моделі та описані її атрибути та методи (див. пункт 3.2)
- реалізовано моделі Django відповідно до розробленої UML моделі
- показано алгоритм роботи модуля-парсера (див. рис. 3.4)
- реалізовано та наведено детальний приклад роботи модуля-парсера з демонстрацією коду (див. пункт 3.3)
- реалізовано інтерфейс користувача за допомогою Django Views та Django Templates (див. пункт 3.4)

Були приведені скріншоти розробки та опис ключових моментів для детальної демонстрації.

РОЗДІЛ 4

ТЕСТУВАННЯ РОЗРОБКИ ТА ПРИКЛАД ЇЇ ВИКОРИСТАННЯ

В цьому розділі буде покроково розглянутий функціонал розробленого веб-додатку.

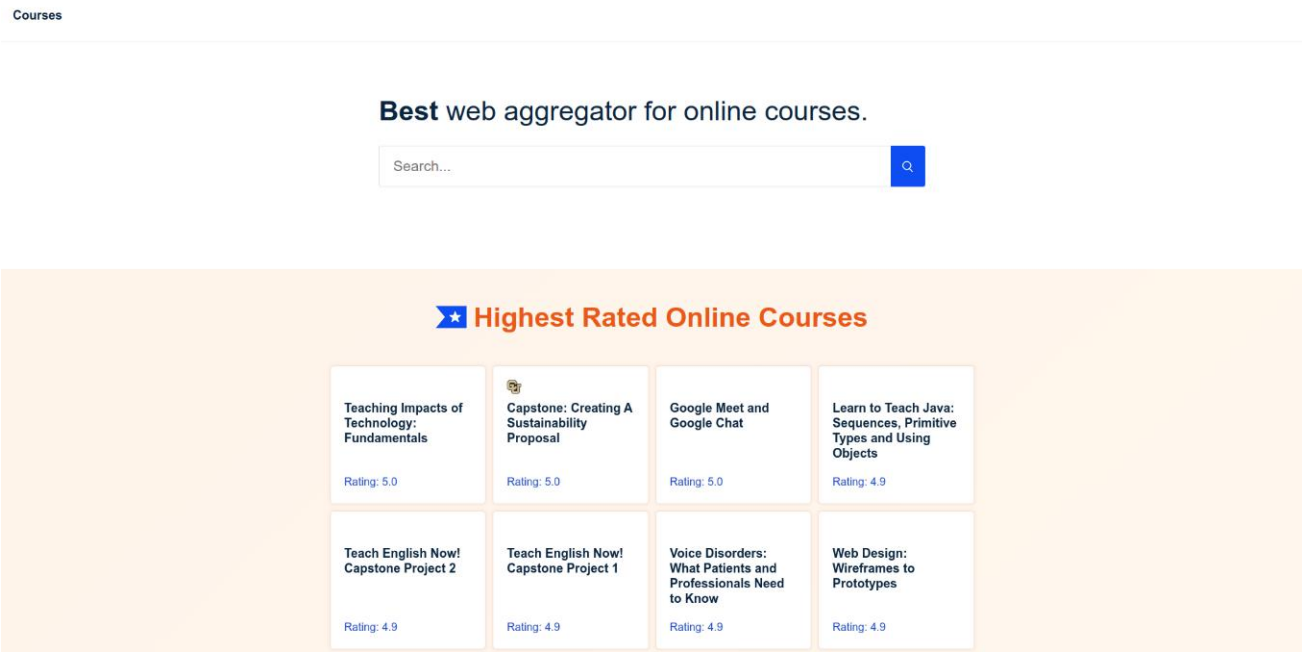


Рис. 4.1 – вид головної сторінки веб-додатку

На рисинку 4.1 продемонстрована головна сторінка, яку користувач бачить при першому відвідувані додатку. У верхньому лівому куті гіперпосилання на каталог всіх курсів, по середині поле пошуку за ключовими словами, під ним сітка з 8 курсів з найкращим рейтингом, користувач бачить назву курсу, зображення (при наявності) та оцінку інших користувачів по п’ятибальній шкалі, при кліку на “плитку” з курсом користувач буде перенаправлений на відповідне посилання.

Був обраний мінімалістичний стиль інтерфейсу з приємними оку кольорами та зручним розташуванням компонентів вебсайту.

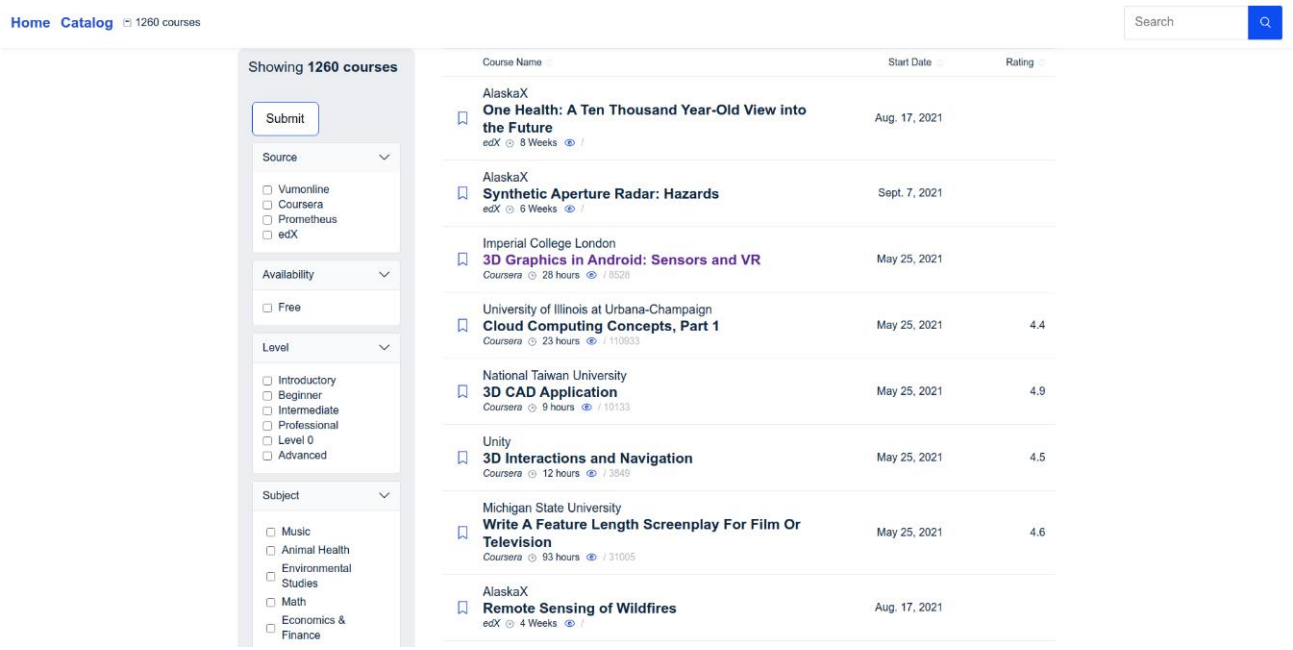


Рис. 4.2 – вид сторінки каталогу курсів

На рисунку 4.2 зображений загальний вигляд сторінки курсів при переході за посиланням “Courses” (див. рис. 4.1).

Рис. 4.3 – вигляд пагінації сторінок

На рисунках 4.2 та 4.3 показаний вигляд сторінки каталогу зі всіма доступними курсами. Згори є два гіперпосилання на головну сторінку та на сторінку каталог, зправа від них кількість доступних курсів. У верхньому правому кутку поле пошуку. В центрі показаний список курсів та така інформація про кожний курс: назва курсу (виділена жирним шрифтом), інститут (над назвою курсу), джерело (під назвою), приблизний час

проходження (справа від джерела та знизу від назви, може вказуватись в кількості занять або в приблизному часі витраченому на курс вцілому), кількість зареєстрованих студентів (справа від часу проходження та знизу від назви), дата початку курса (справа від назви) та його оцінка іншими студентами (при наявності, справа від дати початку курса). При наведені курсора на курс, текст зміститься вліво виділяючи обраний курс серед інших. Зліва від списку курсів є доступні фільтри, такі як джерела курсів, безкоштовність, рівень складності та предмет вивчення. Знизу гіперпосилання пагінації сторінок.

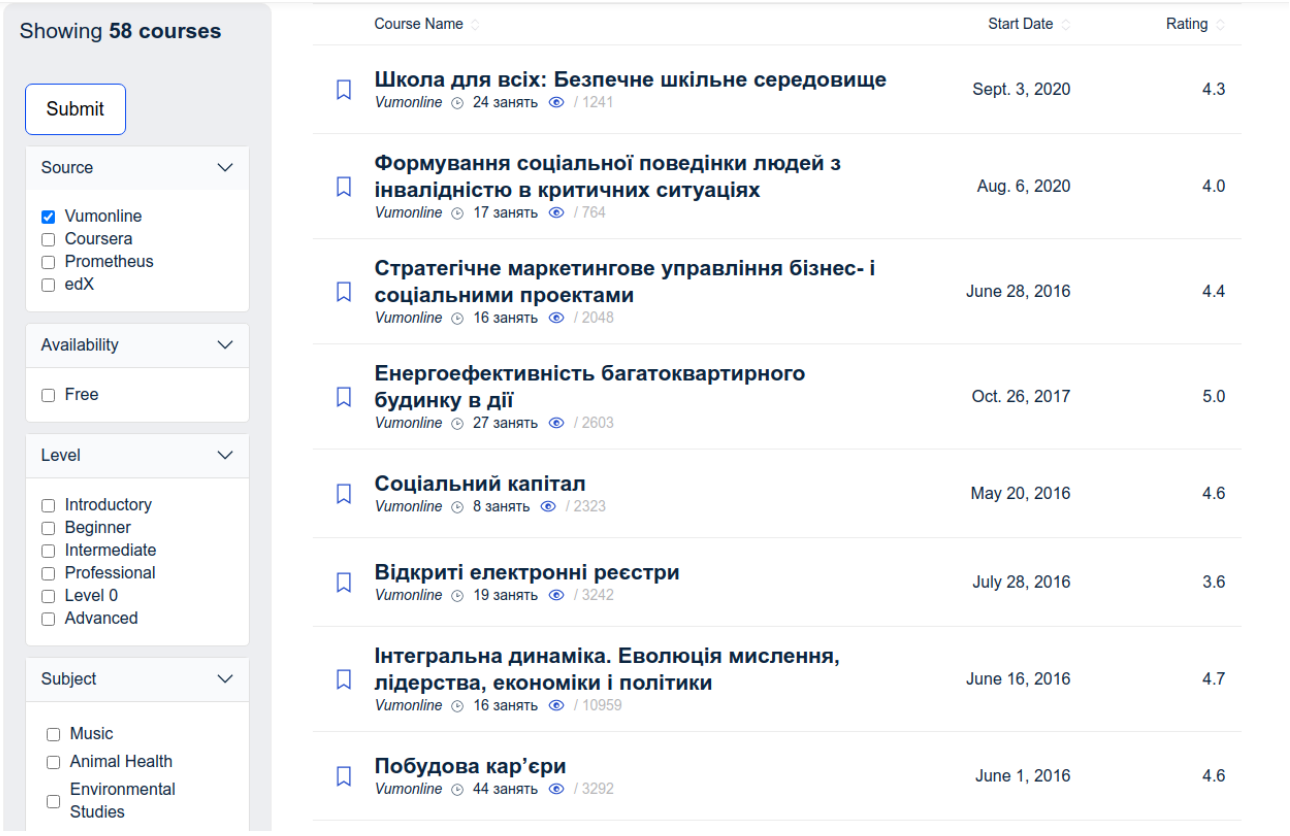


Рис. 4.4 – вид результату фільтрації

На рисунку 4.4 продемонстрований результат фільтрації за джерелом курсу, на ньому видно що в результаті відфільтровано 58 курсів та джерело на представлених курсах співпадає з джерелом фільтрації.

Рис. 4.5 – вид результату пошуку

Щоб дізнатись що відбувається з модулем-парсером під час роботи, можна подивитись логи павуків у зручній веб-консолі модуля Scrapyd (див. рис. 4.6 та 4.7)

Jobs

Go up

Project	Spider	Job	PID	Start	Runtime	Finish	Log	Cancel
Pending								
Running								
default	scheduler	2021-05-28_5228c694bfaa11eb8a8e0242ac140003	129	2021-05-28 11:46:20	0:01:36		Log	Cancel
default	coursera	2021-05-28_54f43a98bfaa11eb8a8e0242ac140003	135	2021-05-28 11:46:25	0:01:31		Log	Cancel
default	edx_courses	2021-05-28_562f4740bfaa11eb8a8e0242ac140003	144	2021-05-28 11:46:31	0:01:26		Log	Cancel
Finished								
default	prometheus	2021-05-28_5748f82ebfaa11eb8a8e0242ac140003		2021-05-28 11:46:35	0:00:02	2021-05-28 11:46:37	Log	
default	vumonline	2021-05-28_5841cb5cbfaa11eb8a8e0242ac140003		2021-05-28 11:46:40	0:00:23	2021-05-28 11:47:03	Log	

Рис. 4.6 — вигляд веб-консолі scrapyd

На рисунку 4.6 показаний статус процесів з часом початку та завершення. Кожен процес зберігає файл-журнал з деталями його роботи, котрий можна переглянути за гіперпосиланням “Log” в рядку відповідного процесу. Гарною функцією є можливість зупинити павука за допомогою кнопки “Cancel”, яка посилає павуку сигнал завершення.

```
2021-05-28 12:10:06 [scrapy.utils.log] INFO: Scrapy 2.5.0 started (bot: scrapers_module)
2021-05-28 12:10:06 [scrapy.utils.log] INFO: Versions: lxml 4.6.3.0, libxml2 2.9.10, cssselect 1.1.0, parsel 1.6.0, w3lib
pyOpenSSL 20.0.1 (OpenSSL 1.1.1j  16 Feb 2021), cryptography 3.4.7, Platform Linux-4.15.0-66-generic-x86_64-with
2021-05-28 12:10:06 [scrapy.utils.log] DEBUG: Using reactor: twisted.internet.epollreactor.EPollReactor
2021-05-28 12:10:06 [scrapy.crawler] INFO: Overridden settings:
{'BOT_NAME': 'scrapers_module',
 'COMMANDS_MODULE': 'scrapers_module.commands',
 'LOG_FILE': 'logs/default/vumonline/2021-05-28_9d669520bfad11eb8a6e0242ac140003.log',
 'LOG_LEVEL': 'INFO',
 'NEWSPIDER_MODULE': 'scrapers_module.spiders',
 'SPIDER_MODULES': ['scrapers_module.spiders'],
 'USER_AGENT': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, '
               'like Gecko) Chrome/86.0.4240.198 Safari/537.36'}
2021-05-28 12:10:06 [scrapy.extensions.telnet] INFO: Telnet Password: db119b0450809292
2021-05-28 12:10:06 [scrapy.middleware] INFO: Enabled extensions:
['scrapy.extensions.corestats.CoreStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.memusage.MemoryUsage',
 'scrapy.extensions.logstats.LogStats']
2021-05-28 12:10:06 [scrapy.core.engine] INFO: Spider opened
2021-05-28 12:10:06 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2021-05-28 12:10:06 [scrapy.extensions.telnet] INFO: Telnet console listening on 127.0.0.1:6029
2021-05-28 12:10:27 [scrapy.core.engine] INFO: Closing spider (finished)
2021-05-28 12:10:27 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 32654,
 'downloader/request_count': 81,
 'downloader/request_method_count/GET': 81,
 'downloader/response_bytes': 1378390,
 'downloader/response_count': 81,
 'downloader/response_status_count/200': 80,
 'downloader/response_status_count/301': 1,
 'elapsed_time_seconds': 21.389175,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2021, 5, 28, 12, 10, 27, 531316),
 'httpcompression/response_bytes': 4400110,
 'httpcompression/response_count': 80,
 'item_scraped_count': 73,
 'log_count/INFO': 10,
 'memusage/max': 74514432,
 'memusage/startup': 74514432,
 'request_depth_max': 1,
 'response_received_count': 80,
 'scheduler/dequeued': 81,
 'scheduler/dequeued/memory': 81,
 'scheduler/enqueued': 81,
 'scheduler/enqueued/memory': 81,
 'start_time': datetime.datetime(2021, 5, 28, 12, 10, 6, 142141)}
2021-05-28 12:10:27 [scrapy.core.engine] INFO: Spider closed (finished)
```

Рис. 4.7 — приклад вмісту файлу-журналу павука

На рисунку 4.7 представлений файл-журнал завершення роботи павука, зі детальною статистикою щодо виконаного процесу. Функція ведення журналу дійсно стає у нагоді для програміста при пошуку несправностей, відладки роботи модуля та для аналізу навантаження на модуль.

ВИСНОВКИ ДО РОЗДІЛУ 4

В данному розділі було оглянуто та протестовано результат розробки агрегатора курсів онлайн навчання, реалізованого у вигляді web-додатку.

Були продемонстровані головна сторінка додатку з функцією пошуку та відображенням кращих курсів за рейтингом студентів, сторінки пошуку та каталогу доступних курсів з можливістю фільтрації та пагінації результатів — одні з необхідних умов будь-якого агрегатора, приділена увага мінімалістичному та лаконічному вигляду сторінок

Також була зазначена та представлена на рисунках зручна властивість додатку scrapyd, а саме можливість переглядати журнал для кожного завдання, отриманого модулем-парсером. Ця властивість дійсно стає впригоді та спрощує відладку системи.

ВИСНОВКИ

В представленому дипломному проєкті була зроблена спроба вирішити проблему агрегації курсів навчання (в загальному випадку будь-яких оновлень сайту).

В першому розділі були розглянуті основні способи представлення інформації на сайтах, а саме HTML, XML, JSON. Оскільки формою кінцевої реалізації проєкту обраний вебсайт агрегатор, то відповідно розглянуто приклади сайтів-агрегаторів та сайтів джерел онлайн курсів з виділенням основних властивостей кожного з них.

В другому розділі були представлені інструменти та підходи до розробки з обґрунтуванням причин вибору певних інструментів та технік. Для реалізації серверу був обраний python фреймворк Django, клієнтський інтерфейс був розроблений згідно рекомендацій по дизайну з використанням django видів та html шаблонів, для реалізації модуля-парсера за основу була обрана python бібліотека Scrapy, для керування та відладки модуля використовувався додаток scrapyd.

В третьому розділі даної роботи були продемонстровані деталі та тонкощі розробки системи. Наведено опис та перелік компонентів системи з детальним поясненням роботи кожного компонента.

В четвертому розділі протестовано та продемонстровано роботу веб-додатка агрегатора.

В даному проєкті були реалізовані основні функції агрегатора кусів навчання, такі як актуальність курсів, можливість фільтрації результатів, пошук за ключовими словам. Надалі буде доцільно сконцентруватись на наданні нового функціоналу та покращені досвіду користувача при використанні системи, так званих QoL (англ. “Quality-of-Life” — “якість життя”) властивостях. Наприклад створенню категорій курсів, вдосконаленню доступних фільтрів, наданні можливості залишити відгук, відмітити пройдений курс тощо.

ЛІТЕРАТУРА

1. Jacobs C. Parsing Techniques - Second Edition / C. Jacobs, D. Grune., 2008
2. HTML Standard [Електронний ресурс] – Режим доступу до ресурсу: https://html.spec.whatwg.org/.
3. About JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.
4. Extensible Markup Language (XML) [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/XML/.
5. The JSON data interchange syntax [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/.
6. The DCI Architecture: A New Vision of Object-Oriented Programming [Електронний ресурс]. – 2009. – Режим доступу до ресурсу: https://web.archive.org/web/20090323032904/https://www.artima.com/articles/dci_vision.html.
7. MVC Application Design [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/cd/A97329_03/core.902/a95101/da_desig.htm.
8. Офіційна документація фреймворку Django версії 3.0 [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/3.0/.
9. Django Template language [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/3.0/ref/templates/language/.
10. Scrapy Architecture overview [Електронний ресурс] – Режим доступу до ресурсу: https://docs.scrapy.org/en/latest/topics/architecture.html.
11. Офіційна документація бібліотеки Scrapy [Електронний ресурс] – Режим доступу до ресурсу: https://docs.scrapy.org/.
12. Scrapy Item Pipeline [Електронний ресурс] – Режим доступу до ресурсу: https://docs.scrapy.org/en/latest/topics/item-pipeline.html.

13. Sitemap Explanation [Електронний ресурс] – Режим доступу до ресурсу: https://www.sitemaps.org/index.htm.
14. Designing the User Interface: Strategies for Effective Human-Computer Interaction (5th Edition) / B.Shneiderman, C. Plaisant, M. Cohen, S. Jacobs., 2009. – 624 с
15. Nielsen J. 10 Usability Heuristics for User Interface Design [Електронний ресурс] / J. Nielsen, R. Molich. – 1994. – Режим доступу до ресурсу: https://www.nngroup.com/articles/ten-usabilityheuristics/.
16. Огляд можливостей CSS [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/css/index.htm.
17. Usage statistics of JavaScript libraries for websites [Електронний ресурс] – Режим доступу до ресурсу: https://w3techs.com/technologies/ overview/javascript_library.
18. Greenfeld D. R. Two Scoops of Django: Best Practices for Django 1.8 / D. R. Greenfeld, A. R. Greenfeld., 2015. – 532 с
19. Офіційна документація API додатку Scrapyd [Електронний ресурс] – Режим доступу до ресурсу: https://scrapyd.readthedocs.io/en/stable/api.html
20. Gaston C. Hillar “Django RESTful Web Services”, 2018
21. Hajba G. L. Website Scraping with Python: Using BeautifulSoup and Scrapy / G bor L szl Hajba., 2015. – 241 с. – (1st).

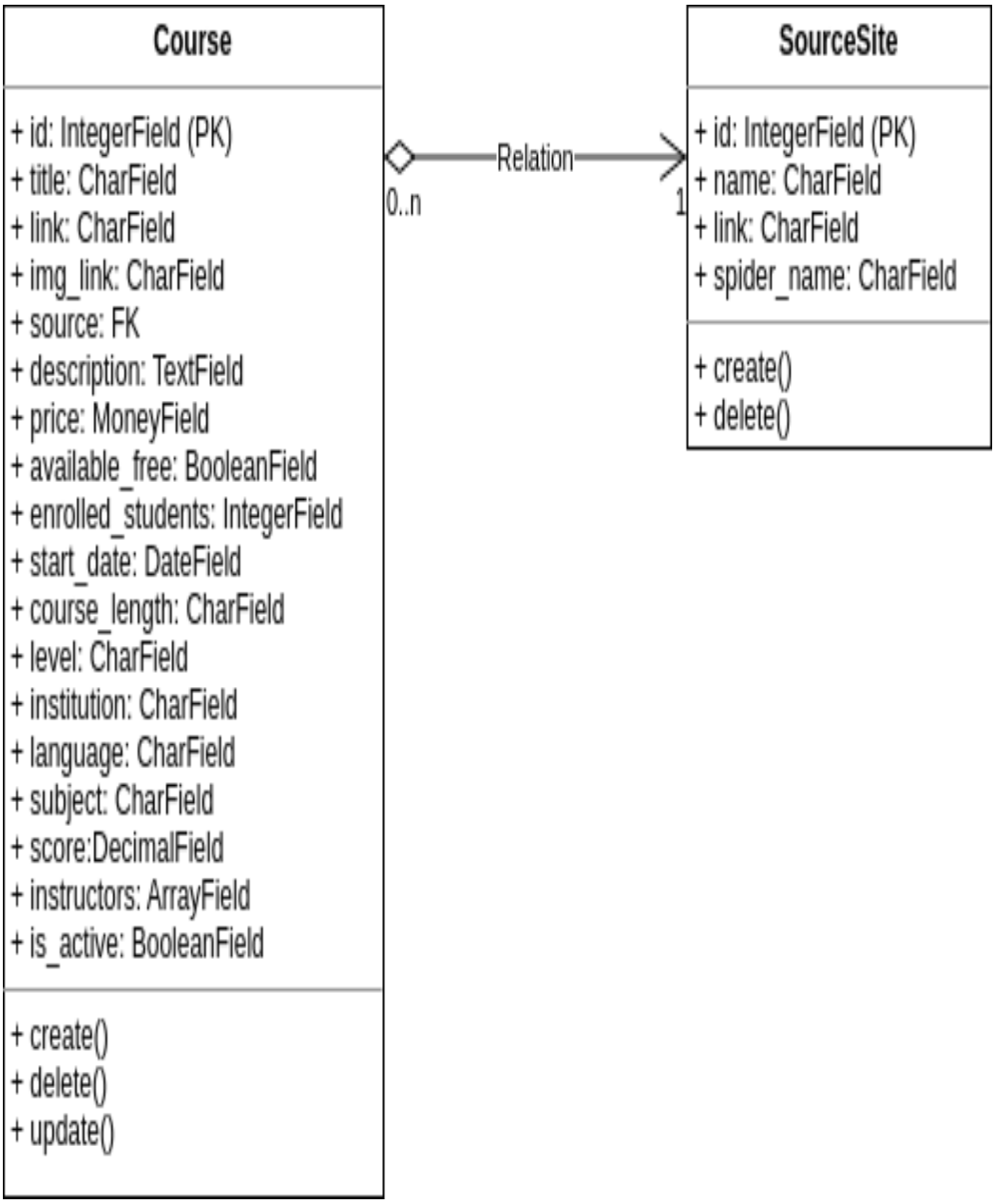
ДОДАТОК 1

Веб-додаток агрегатор курсів дистанційного навчання

UML діаграма класів моделі бази даних
ІАЛЦ.467100.004 Д1

Аркушів 1

Київ 2021 р.



					ІАЛЦ.467100.004 Д1	
Зм.		№ документа	Підп.	Дата		
Розробив		Фурман К.О.			Веб-додаток агрегатор курсів дистанційного навчання UML діаграма класів моделі бази даних	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Каплунов А.В.				
Н.контр.		Сімоненко В. П.				НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІВ-72
Затверд.		Стіренко С. Г.				

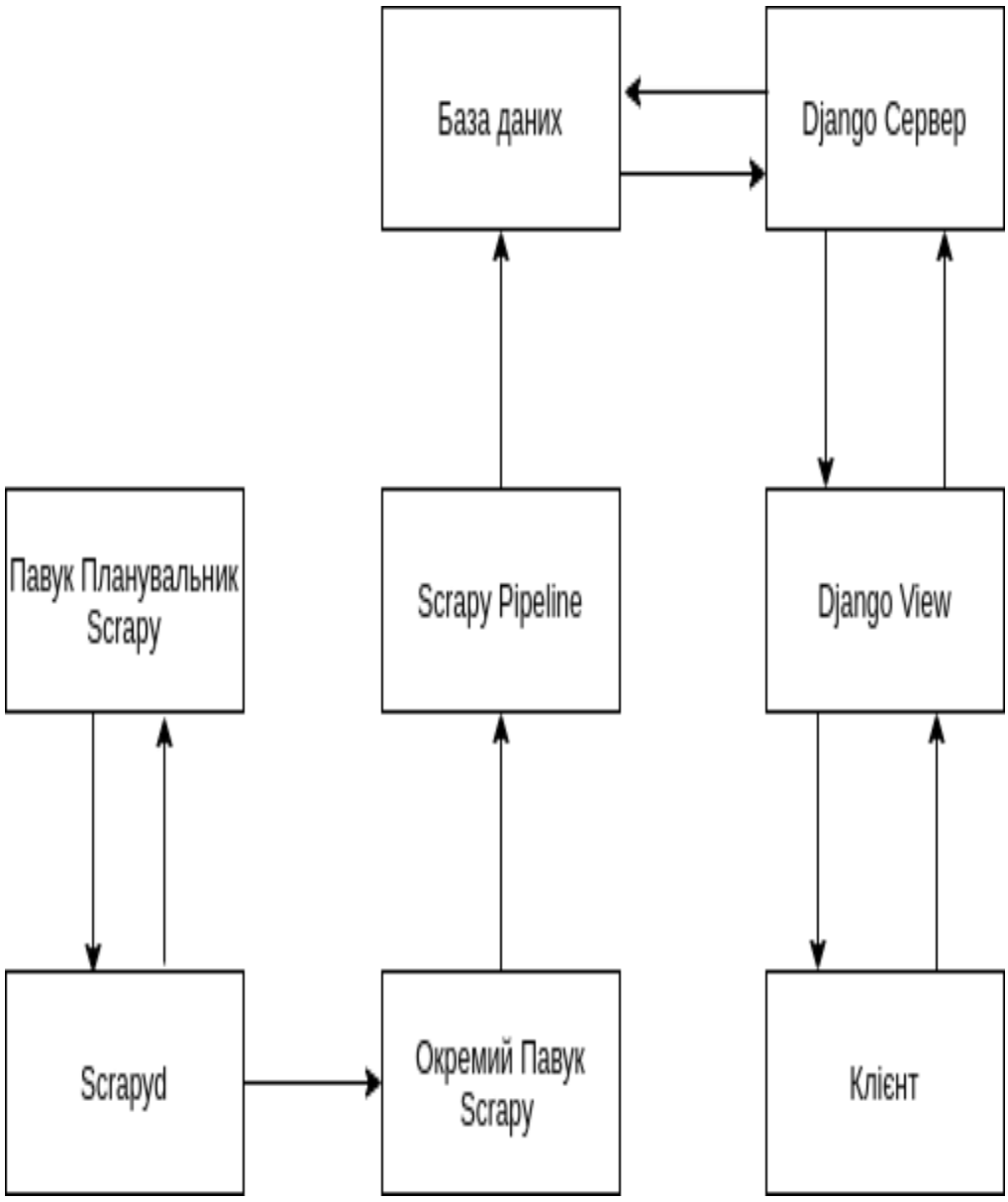
ДОДАТОК 2

Веб-додаток агрегатор курсів дистанційного навчання

Структурна схема системи
ІАЛЦ.467100.005 Д2

Аркушів 1

Київ 2021р.


База даних
Django Сервер
Павук Планувальник Scrapy
Scrapy Pipeline
Django View
Scrapyd
Окремий Павук Scrapy
Клієнт

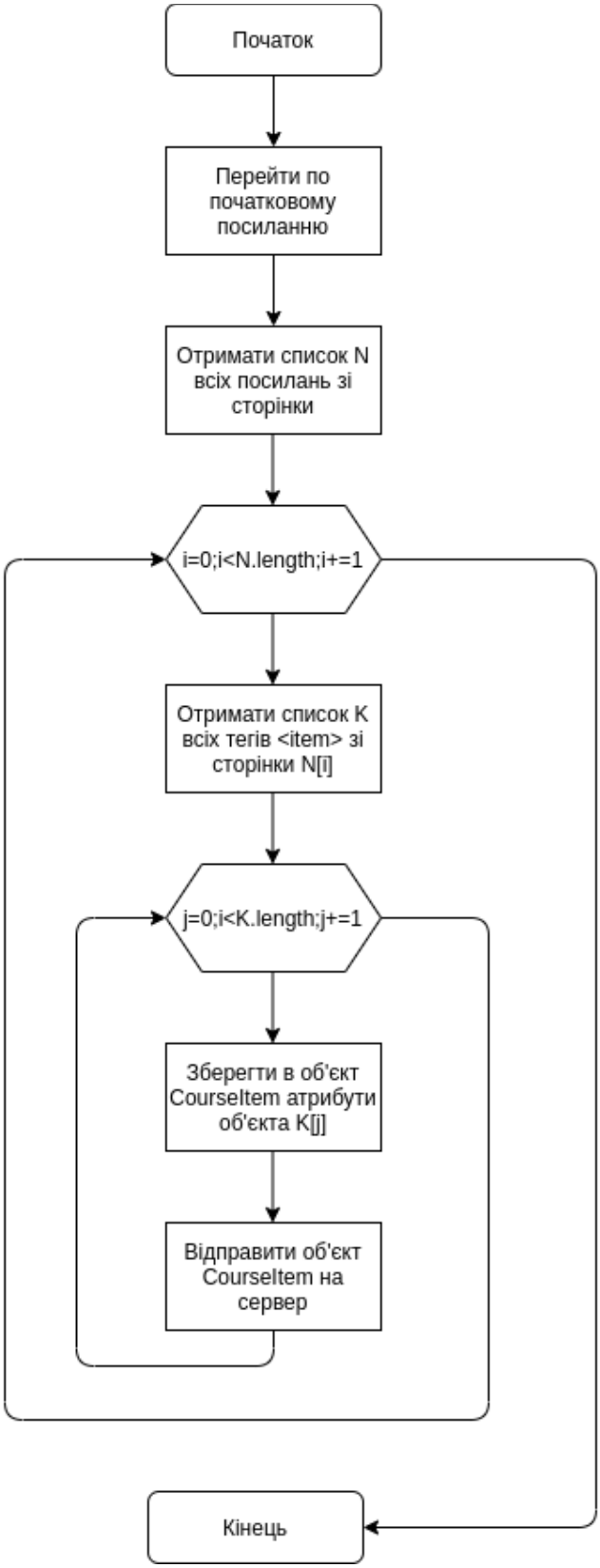
ДОДАТОК 3

Веб-додаток агрегатор курсів дистанційного навчання

Блок-схема алгоритму роботи модуля-парсера
ІАЛЦ.467100.006 Д3

Аркушів 1

Київ 2021 р.



					ІАЛЦ.467100.006 Д3	
		№ документа	*Підп.*	*Дата*		
Зм.		*Фурман К.О.*			*Веб-додаток агрегатор курсів дистанційного навчання Блок-схема алгоритму роботи модуля-парсера*	*Літ.* *Аркуш* 1 1
		Каплунов А.В.				
Н.контр.		*Сімоненко В. П.*				*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІВ-72*
Затв.		*Стіренко С. Г.*				

ДОДАТОК 4

Веб-додаток агрегатор курсів дистанційного навчання

Сирцевий код проєкту

ІАЛЦ.467100.007 Д4

Аркушів 11

Київ 2021 р.

```
from django.db import models
from djmoney.models.fields import MoneyField
from django.contrib.postgres.fields import ArrayField, JSONField


class SourceSite(models.Model):
    name = models.CharField(max_length=256, unique=True)
    link = models.CharField(max_length=256)
    spider_name = models.CharField(max_length=256)

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


class Course(models.Model):
    title = models.CharField(max_length=256)
    link = models.CharField(max_length=256)
    img_link = models.CharField(max_length=512)
    source = models.ForeignKey(SourceSite, on_delete=models.CASCADE)
    description = models.TextField()
    price      =      MoneyField(max_digits=8,      decimal_places=2,      null=True,
default_currency=None)
    available_free = models.BooleanField(default=True)
    enrolled_students = models.IntegerField(null=True)
    start_date = models.DateField()
    course_length = models.CharField(max_length=256, null=True)
    level = models.CharField(max_length=256, null=True)
    institution = models.CharField(max_length=256, null=True)
    language = models.CharField(max_length=256, null=True)
    subject = models.CharField(max_length=256, null=True)
    score = models.DecimalField(max_digits=2, decimal_places=1, null=True)
    instructors = ArrayField(JSONField(null=False), null=True)
    is_active = models.BooleanField(default=True)

    class Meta:
        unique_together = ('title', 'source', 'language')

    def __str__(self):
        return f'{self.source.name} | {self.title}'

from django.db.models import Q

# Create your views here.
from django.views.generic import TemplateView, ListView

from .models import *


class HomePageView(TemplateView):
    template_name = 'index.html'

    def get(self, request, *args, **kwargs):
        context = self.get_context_data(**kwargs)
        return self.render_to_response(context)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['best_courses'] = Course.objects.filter(score__isnull=False,
```

```
                                                    institution__isnull=False,
                                                    is_active=True).order_by('-
score')[:12]
        return context


class CatalogListView(ListView):
    model = Course
    template_name = 'class_catalog.html'
    paginate_by = 30

    def get_queryset(self):
        object_list = self.model.objects.filter(is_active=True)

        sources = self.request.GET.getlist('source')
        if sources:
            object_list = object_list.filter(source__name__in=sources)

        levels = self.request.GET.getlist('level')
        if levels:
            levels = [i.lower() for i in levels]
            object_list = object_list.filter(level__in=levels)

        free = self.request.GET.get('free')
        if free:
            object_list = object_list.filter(available_free=bool(int(free)))

        subjects = self.request.GET.getlist('subject')
        if subjects:
            object_list = object_list.filter(subject__in=subjects)

        query = self.request.GET.get('q')
        if query:
            object_list = object_list.filter(
                Q(title__icontains=query) | Q(source__name__icontains=query)
            )

        return object_list

    def get_context_data(self, **kwargs):
        ctx = super().get_context_data(**kwargs)
        ctx['sources']        =        [i['source__name']        for        i        in
self.model.objects.values('source__name').distinct()]

        levels                                                                    =
self.model.objects.filter(level__isnull=False).values('level').distinct()
        ctx['levels'] = [i['level'].capitalize() for i in levels if i]

        subjects                                                                  =
self.model.objects.filter(subject__isnull=False).values('subject').distinct()
        ctx['subjects'] = [i['subject'] for i in subjects]

        ctx['courses_count'] = self.model.objects.all().count()
        ctx['courses_show_count'] = self.get_queryset().all().count()
        return ctx

from django import template

register = template.Library()
```

```
@register.simple_tag(takes_context=True)
def url_replace(context, **kwargs):
    d = context['request'].GET.copy()
    for k, v in kwargs.items():
        d[k] = v
    for k in [k for k, v in d.items() if not v]:
        del d[k]
    return d.urlencode()
```

```
from django.urls import path

from .views import *

urlpatterns = [
    path('search/', CatalogListView.as_view(), name='search_results'),
    path('', HomePageView.as_view(), name='home'),
    path('catalog/', CatalogListView.as_view(), name='catalog_list')
]
```

```
from .items import BaseItem
from core.models import Course


class PostgresSqlPileline:
    _db_model = Course

    def open_spider(self, spider):
        self.deactivate_saved_items(spider)

    def deactivate_saved_items(self, spider):
        deactivated                                                                    =
self._db_model.objects.filter(source__spider_name=spider.name).update(is_active=False)
        spider.logger.info(f'Deactivated {deactivated} items in db')

    def process_item(self, item, spider):
        if isinstance(item, BaseItem):
            _instance = self.create_db_instance(item)
            _instance.save()
        return item

    def create_db_instance(self, item):
        instance_data = {}
        for key in item.fields.keys():
            instance_data[key] = item[key]
        return                 self._db_model.objects.update_or_create(**instance_data,
defaults={'is_active': True})
```

```
import scrapy

class BaseItem(scrapy.Item):
    pass

class CourseItem(BaseItem):

    title = scrapy.Field()
    source_id = scrapy.Field()
    link = scrapy.Field()
    img_link = scrapy.Field()
```

```
    description = scrapy.Field()
    price = scrapy.Field()
    available_free = scrapy.Field()
    enrolled_students = scrapy.Field()
    start_date = scrapy.Field()
    course_length = scrapy.Field()
    institution = scrapy.Field()
    language = scrapy.Field()
    level = scrapy.Field()
    score = scrapy.Field()
    instructors = scrapy.Field()
    subject = scrapy.Field()

import scrapy
import re
import json
from scrapy.spiders import SitemapSpider
from django.utils import timezone

from ..items import CourseItem
from core.models import SourceSite


class CourseraSpider(SitemapSpider):
    name = 'coursera'

    custom_settings = {
        'LOG_LEVEL': 'INFO',
        'DOWNLOADER_MIDDLEWARES': {
            # 'scrapers_module.middlewares.LuminatiProxyUs': 1000
        },
        'ITEM_PIPELINES': {
             'scrapers_module.pipelines.PostgresSqlPileline': 300,
        }
    }

    sitemap_urls = ('https://www.coursera.org/sitemap~www~courses.xml',)

    sitemap_rules = [('/learn/', 'parse_course')]

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.source_id = SourceSite.objects.get(spider_name=self.name).id

    def parse_course(self, response):
        course_approx_len           =           response.xpath('//span[contains(text(),
"Approx.")]/text()').get() or ''
        course_length   =   course_approx_len.replace('Approx.   ',   '').replace('   to
complete', '')

        script_data = response.xpath('/html/body/script[1]').get()
        primary_langs                                                              =
json.loads(re.findall('"primaryLanguages":\{"type":"json","json":(.+?)\}',
script_data)[0])
        subtitle_langs                                                             =
json.loads(re.findall('"subtitleLanguages":\{"type":"json","json":(.+?)\}',
script_data)[0])

        level = re.findall(',"level":"(\w+)",', script_data)
```

```
        level = level[0].lower() if level else None

        enrolled_students          =          response.xpath('string(//div[@class="rc-
ProductMetrics"]//strong)').get() or None
        if enrolled_students:
            enrolled_students = int(enrolled_students.replace(',', ''))

        yield CourseItem({
            'source_id': self.source_id,
            'title': response.xpath('//h1/text()').get(),
            'link': response.url,
            'img_link': response.xpath('//div[@class="_1m6egy8n"]//img/@src').get(),
            'description':                          response.xpath('//div[@class="m-t-1
description"]//p/text()').get(),
            'price': None,
            'available_free': True,
            'enrolled_students': enrolled_students,
            'start_date': timezone.now().date(),
            'course_length': course_length,
            'institution': response.xpath('//div[@class="p-b-1s p-r-1"]//@alt').get(),
            'language': self.languages_from_list(primary_langs, subtitle_langs),
            'level': level,
            'score':   response.xpath('//span[@class="_16ni8zai   m-b-0   rating-text
number-rating number-rating-expertise"]/text()').get(),
            'instructors': [],
            'subject': response.xpath('//div[@aria-current="page"]/a/text()').get(),
        })

    def instructors(self, response):
        instructors = []
        for div in response.xpath('//div[@class="instructor-wrapper"]'):
            instructors.append({
                'name': div.xpath('.//h3/text()').get(),
                'title': div.xpath('.//span[@class="instructor-title"]/text()').get(),
                'institution':          div.xpath('.//div[@class="instructor-department
caption-text color-black"]/text()').get()
            })
        return instructors or None

from pydispatch import dispatcher

from scrapy import Spider, signals
from scrapy.exceptions import DontCloseSpider
from scrapyd_api import ScrapydAPI

from datetime import datetime

from core.models import Course


class SchedulerSpider(Spider):
    name = 'scheduler'

    custom_settings = {
        'LOG_LEVEL': 'WARNING',
        'ITEM_PIPELINES': {}
    }

    def __init__(self, *args, **kwargs):
```

```
        super().__init__(*args, **kwargs)
        dispatcher.connect(self.spider_idle, signals.spider_idle)
        self.scrapyd = ScrapydAPI('http://localhost:6801')
        self.scrapyd_project_name = 'default'

        self.hours_period = 24
        self.last_run = None

    def spider_idle(self):
        if not self.last_run or self._hours_from_last_run() >= self.hours_period:
            self.schedule_all_spiders()
            self.last_run = datetime.now()

        raise DontCloseSpider

    def schedule_all_spiders(self):
        for spider in self.list_spiders():
            if not self.spider_is_running(spider):
                self.schedule_spider(spider)
            else:
                self.logger.info(f'Spider {spider} is already in process')

    def schedule_spider(self, spidername):
        self.scrapyd.schedule(self.scrapyd_project_name, spidername)
        self.logger.info(f'Scheduled {spidername}')

    def spider_is_running(self, spidername):
        all_jobs_dict = self.scrapyd.list_jobs(self.scrapyd_project_name)
        not_finished_jobs = all_jobs_dict['pending'] + all_jobs_dict['running']
        for job in not_finished_jobs:
            if spidername == job['spider']:
                return True
        return False

    def list_spiders(self):
        return self.scrapyd.list_spiders(self.scrapyd_project_name)

    def _hours_from_last_run(self):
        last_run_delta = datetime.now() - self.last_run
        return last_run_delta.seconds // 3600

import scrapy

from ..items import CourseItem
from core.models import SourceSite

from datetime import datetime


class VumonlineSpider(scrapy.Spider):
    name = 'vumonline'
    allowed_domains = ['vumonline.ua']

    custom_settings = {
        'LOG_LEVEL': 'INFO',
        'DOWNLOADER_MIDDLEWARES': {
            'scrapers_module.middlewares.LuminatiProxyUs': 1000
        },
        'ITEM_PIPELINES': {
```

```
            'scrapers_module.pipelines.PostgresSqlPileline': 300,
        }
    }

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.source_id = SourceSite.objects.get(spider_name=self.name).id

    def start_requests(self):
        for i in range(1, 8):
            yield scrapy.Request(f'https://vumonline.ua/courses/page/{i}/')

    def parse(self, response, **kwargs):
        for course in response.xpath('//div[@class="col-lg-4 col-md-6 col-sm-6 single-
cource"]'):
            url = course.xpath('.//h2/a/@href').get()
            img_link = course.xpath('./div[@class="image-cource"]/img/@src').get()
            yield scrapy.Request(url, self.parse_course, meta={'img_link': img_link})

    def parse_course(self, response):
        extracted_date          =          response.xpath('//span[@class="start-cource-
time"]/text()').get()
        return CourseItem({
            'source_id': self.source_id,
            'title': response.xpath('//h1/text()').get(),
            'link': response.url,
            'img_link': response.meta['img_link'],
            'description':                      response.xpath('//div[@class="visible-
content"]/p/text()').get(),
            'price': None,
            'available_free': True,
            'enrolled_students':    response.xpath('string(//div[@class="stats-numbers-
wrap"]/div[1])').get().strip(),
            'start_date': datetime.strptime(extracted_date, '%d.%m.%Y'),
            'course_length':       response.xpath('string(//div[@class="stats-numbers-
wrap"]/div[2])').get().strip(),
            'level': 'BEGINNER'.lower(),
            'language': 'UA',
            'score': response.xpath('//div/@data-rateit-value').get(),
            'instructors': None,
            'subject': None,
            'institution': None
        })

import re

from scrapy.spiders import SitemapSpider

from ..items import CourseItem
from core.models import SourceSite

from djmoney.money import Money

from datetime import datetime


class EdxCoursesSpider(SitemapSpider):
    name = 'edx_courses'
```

```
    custom_settings = {
        'LOG_LEVEL': 'INFO',
        'DOWNLOADER_MIDDLEWARES': {
            'scrapers_module.middlewares.LuminatiProxyUs': 1000
        },
        'ITEM_PIPELINES': {
            'scrapers_module.pipelines.PostgresSqlPileline': 300,
        }
    }

    handle_httpstatus_list = [404, 400]

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.source_id = SourceSite.objects.get(spider_name=self.name).id

    sitemap_urls = ('https://www.edx.org/sitemap.xml',)
    sitemap_rules = [('org\/course\/[\w-]+\/?$', 'parse_course')]

    def parse_course(self, response):
        if (not response.xpath('//h1/text()').get(default="").strip()) \
                or (not '/course/' in response.url) or 'This course is archived' in
response.text:
            return

        price = self.price(response)
        img_link_xpath = '//img[@class="header-image d-none d-sm-inline"]/@src' \
                         '| //img[@class="video-thumb"]/@src'

        return CourseItem({
            'source_id': self.source_id,
            'link': response.url,
            'img_link': response.xpath(img_link_xpath).get(),
            'title': response.xpath('//h1/text()').get(),
            'description': self.course_description(response),
            'price': Money(price['price'], price['currency']),
            'available_free': self.available_free(response),
            'enrolled_students': self.enrolled_students(response),
            'start_date': self.start_date(response),
            'course_length': self._course_property(response, 'Length'),
            'institution': self._course_property(response, 'Institution'),
            'language': self._course_property(response, 'Language'),
            'subject': self._course_property(response, 'Subject'),
            'level': self._course_property(response, 'Level').lower(),
            'instructors': self.instructors(response),
            'score': None,
        })

    def start_date(self, response):
        extracted_string = response.xpath('//div[contains(text(), "Starts ")]/text()'
                                          '|      //div[contains(text(),      "Started
")]/text()').get()
        if not extracted_string:
            self.logger.warning(f'No start date {response.url}')
            return

        date_string = re.findall('[A-Za-z]+ \d+', extracted_string)[0]
        parsed_date = datetime.strptime(date_string, '%b %d')
        parsed_date = parsed_date.replace(year=datetime.now().year)
```

```
        return parsed_date

    def course_description(self, response):
        raw_description         =         response.xpath('string(//div[@class="course-
description"])').get(default="")
        description = re.sub('\s{2,}', ' ', raw_description)
        description = re.sub('\n', ' ', description)
        return description

    def instructors(self, response):
        instructors = []
        for div in response.xpath('//div[contains(@class, "instructor-details")]'):
            instructors.append({
                'name': div.xpath('./*[contains(@class, "name")]/text()').get(),
                'title': div.xpath('./*[contains(@class, "title")]/text()').get(),
                'institution': div.xpath('./*[contains(@class, "org")]/text()').get()
            })
        return instructors or None

    def enrolled_students(self, response):
        xp = '//div[@id="js-number-enrolled-label"]//span/text()'
        count = response.xpath(xp).get()
        if count:
            return count.replace(',', '')

    def available_free(self, response):
        price_prop = self._course_property(response, 'Price')
        return 'free' in price_prop.lower()

    def price(self, response):
        price_prop = self._course_property(response, 'Price')
        price = re.findall('[$\d]+', price_prop)

        if not price:
            return
        if '$' not in price[0]:
            self.logger.warning(f'No currency in "{price_prop}", url {response.url}')
            return

        return {'price': price[0].replace('$', ''),
                'currency': 'USD'}

    def _course_property(self, response, name):
        xp          =          '//span[contains(text(),          "{}")]/../following-
sibling::div//text()'.format(name)
        return ' '.join(response.xpath(xp).extract())

import scrapy

from ..items import CourseItem
from core.models import SourceSite

from djmoney.money import Money
from datetime import datetime

class PrometheusSpider(scrapy.Spider):
    name = 'prometheus'
    allowed_domains = ['prometheus.org']
    start_urls = ['http://prometheus.org/']
```

```
    custom_settings = {
         'LOG_LEVEL': 'INFO',
        'DOWNLOADER_MIDDLEWARES': {
            #'scrapers_module.middlewares.LuminatiProxyUs': 1000
        },
        'ITEM_PIPELINES': {
            #'scrapers_module.pipelines.PostgЯresSqlPileline': 300,
        },
        'COOKIES_ENABLED': True
    }

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.source_id = SourceSite.objects.get(spider_name=self.name).id

    def start_requests(self):
        url = 'https://prometheus.org.ua/wp-admin/admin-ajax.php'
        headers = {'Accept': '*/*',
                   'X-Requested-With': 'XMLHttpRequest',
                   'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'}
        for i in range(1, 10):
            yield scrapy.FormRequest(url,
                                     headers=headers,
                                     formdata=formdata)

    def parse(self, response, **kwargs):
        for course in response.json()['results']:
            price = course['first_enrollable_paid_seat_price']
            languages = ', '.join(set(['UA'] + course['transcript_languages']))

            course_level = course.get('type')
            if course_level:
                course_level = course_level.lower()

            yield CourseItem({
                'source_id': self.source_id,
                'title': course['title'],
                'link': course['marketing_url'],
                'img_link': course['image'].get('src'),
                'description': course['short_description'],
                'price': Money(price, 'UAH') if price else None,
                'available_free': False if price else True,
                'enrolled_students': course['enrollment_count'],
                'start_date': datetime.fromisoformat(course['start']),
                'course_length':  '{}  Weeks'.format(course['weeks_to_complete'])  if
course['weeks_to_complete'] else None,
                'language': languages,
                'level': course_level,
                'institution': None,
                'score': None,
                'instructors': None,
                'subject': None
            })
```