МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

15.03 — КМР. 1921–“С” 2023.10.23. 01 ПЗ

Колісниченко Дмитро Геннадійович

2023 р.

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 004.9:37.018.43

«ПОГОДЖЕНО»	**«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»**
Декан факультету інформаційних технологій	Завідувач кафедри комп'ютерних наук
Глазунова О.Г., д.п.н., професор	Голуб Б.Л., к.т.н., доцент
_____	_____
____ _____ **202_ р.**	____ _____ **202_ р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Система підтримки прийняття рішень керівництво районної поліклініки

Спеціальність 121 Інженерія програмного забезпечення
(код і назва)

Освітня програма Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

Доцент,кандитат технічних наук _____ Голуб Б.Л
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

Доцент,кандитат технічних наук _____ Бушма О.В
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Колісниченко Д.Г.
(підпис) (ПІБ студента)

КИЇВ-2023

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ)
Інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
Доц.,к.т.н. _____ Голуб Б.Л
(науковий ступінь, вчене звання) (підпис)
(ПІБ)
"____" _____20_____року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Колісниченко Дмитро Геннадійович
(прізвище, ім'я, по батькові)
Спеціальність
121 Інженерія програмного забезпечення
(код і назва)
Освітня програма Програмне забезпечення інформаційних систем
(назва)
Орієнтація освітньої програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Тема магістерської кваліфікаційної роботи: програмне забезпечення автоматизованого збору та аналітичної обробки інформації про навчальні онйлан-курси, затверджена наказом ректора НУБіП України від "30" грудня 2022р. № С
Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)
Вихідні дані до магістерської кваліфікаційної роботи: документ із навантаженням навчального плану на кафедру.
Перелік питань, що підлягають дослідженню:

1. Дослідження предметної області
2. Постановка задачі.
3. Вибір методів дослідження.

Перелік графічного матеріалу (за потреби) допускається.
Дата видачі завдання " 30" грудня 2022 р.
Керівник магістерської кваліфікаційної роботи _____ Бушма О.В.
(підпис) (прізвище та ініціали)
Завдання прийняв до виконання _____ Колісниченко Д.Г.
(підпис) (прізвище та ініціали студента)

Анотація

У цій дипломній роботі було проведене дослідження над об’єктом дослідження, а саме над програмним забезпеченням автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси. Методом дослідження роботи є покращення пошуку інформації та методу аналізу даних.

У даному дослідженні були проведені аналіз сучасного ринку онлайн-курсів, що на даний момент дуже сильно розвивається з початку пандемії. Люди почали масово навчитись вдома, але іноді не знають який курс краще обрати. Для цього є рішення – додаток, який зможе допомогти проаналізувати та вибрати кращий для них курс.

Abstract

This diploma thesis includes research conducted on the subject of automated data collection and analytical processing of information about online educational courses. The research methodology involves improving information search and data analysis methods.

This study involved an analysis of the current online course market, which has experienced significant growth since the beginning of the pandemic. People have started to learn from home in large numbers, but sometimes, they do not know which course is the best choice for them. The solution to this problem is a mobile application that can assist in analyzing and selecting the most suitable course for them.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

SQL – Structured Query Language

JS – JavaScript

JWT – JSON Web Token

UML – Unified Modeling Language

БД – база даних

ІТ – інформаційні технології

API – Application Programming Interface

CI/CD – Continuous Integration and Continuous Delivery

ПЗ – програмне забезпечення

KPI – Key Performance Indicator

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

RAM – Random Access Memory

ВСТУП

Актуальність теми дослідження. На сьогоднішній день це є дуже актуальна тема, тому що обсяг інформації збільшується з кожним днем, годиною, хвилиною, секундою. Часто люди для пошуку якоїсь інформації втрачають час для пошуку і знаходять у деяких випадках не зовсім потрібний та достовірний матеріал для них. На даний момент є рішення. Це автоматизовані програми парсери. Вони за допомогою певних конфігів можуть збирати інформацію з певних сайтів на постійній основі, щоб дає змогу отримувати майже постійно актуальну інформацію.

Об'єктом дослідження є знаходження актуальної інформації на веб-ресурсах та автоматизація її обробки.

Предмет дослідження – це є процес впровадження та перспективи використання парсингу інформації про начальні онлайн курси.

Мета дослідження полягає у створенні, розробці та оцінці комплексної системи для автоматизованого збору інформації про навчальні онлайн курси, яка включає в себе інструменти для збору, обробки, аналізу та збереження даних про курси, їхні властивості, відгуки користувачів, доступність, вартість, оцінки, а також іншу важливу інформацію. Метою є покращення якості та зручності доступу користувачів до інформації про навчальні ресурси в онлайн-середовищі, сприяючи їхньому більш ефективному вибору та використанню. Дослідження передбачає аналіз сучасних практик збору даних про курси, розробку технічних рішень для автоматизації цього процесу, тестування та оцінку розробленої системи, а також рекомендації щодо її подальшого вдосконалення та впровадження в навчальному середовищі. Доцільність дослідження за допомогою D3.js для створення графіків та Puppeteer, Cheerio для скрапингу даних та node-cron для автоматизації самого процесу.

Завдання дослідження.

1. Основна задача полякає в тому, щоб оцінити на скільки це є ефективно та корисно для людей, які хочуть якогомога швидше знайти інформацію про курси, щоб не витрачати свій час марно.
2. Огляд загальнодоступної інформації про дану тему і чи це є актуальним на даний момент.

Методи дослідження.

У своїй роботі використовувались кілька методів дослідження, а саме:

1. Спостереження – систематичне відстеження допитливості людей про пошук курсів, їхні запитання про них.
2. Соціологічні дослідження – опитування та аналіз соціальних явищ та їх процесів.

Наукова новизна.

Можна автоматизовано збирати інформацію про курси з різних джерел та платформ, включаючи їхні назви, описи, ціни, акції, авторів, оцінки тощо. Новизна може полягати в способах ефективного збору інформації та її агрегації в одне зручне для аналізу сховище. Програмне забезпечення може використовувати різні метрики та алгоритми для порівняння курсів на різних платформах. Наприклад, враховуючи якість навчання, популярність, актуальність, оцінки користувачів і ціни, програмне забезпечення може визначити, які курси є найбільш вигідними для користувача. Програмне забезпечення може автоматизовано відстежувати актуальність і оновлювати дані про курси, оскільки ціни і доступність можуть змінюватися впродовж часу. Нововведення може полягати в розробці зручних інтерфейсів та інструментів візуалізації, які допомагають користувачам порівнювати курси на різних платформах.

Апробація результатів дослідження.

ПЗ включає в себе процес апробації та перевірки результатів досліджень через їх представлення та обговорення на спеціалізованих наукових конференціях, семінарах, симпозіумах або публікації в наукових журналах. Цей процес надає

можливість академічній спільноті ознайомитися з результатами досліджень, висувати запитання, пропонувати подальші дослідження та висловлювати свої коментарі.

Процес апробації може включати наступні етапи:

1. Публікація результатів: Результати дослідження можуть бути представлені на наукових конференціях, де викладачі та дослідники можуть ділитися результатами та обмінюватися думками. Зазвичай це супроводжується презентацією або доповіддю, після якої відбувається обговорення та обмін думками.
2. Підготовка наукових статей: Результати дослідження можуть бути описані в наукових статтях. Дослідники можуть написати статтю, яка містить опис методології дослідження, отримані результати, їх аналіз та висновки. Ці статті можуть бути опубліковані в наукових журналах, спеціалізованих у галузі онлайн-навчання та технологій освіти.

Такий підхід сприяє підвищенню якості навчальних курсів і платформи, робить їх більш доступними для академічного співтовариства та сприяє постійному покращенню якості онлайн-навчання.

Структура роботи.

Дипломна робота містить у собі чотири розділи, а саме: системний аналіз предметної області, моделювання системи, розробка системи, аналіз та результати дослідження.

У розділі «системний аналіз» йде опис предметної області, аналіз наявних рішень, які на даний момент представлені аналоги ПЗ, яке ми аналізуємо. Також була постановка завдання, що треба зробити перед моделюванням нашої системи.

У розділі «моделювання системи» було проведено порівняння функціонального моделювання та об’єтно-орієнтованого моделювання. Також були побудовані діаграми активності, прецедентів та послідовності, що відіграє ключову роль перед розробкою додатків.

У «розробці системи» було описано, які інструменти використовувались, яка мова програмування, БД, запобігання від хакерських атак, що може автоматизувати систему, так як це дуже актуально у наш час і збереже ресурси у подальшому.

У «аналізі та результатах дослідження» був проведений аналіз апаратного та програмного забезпечення, що є важливим. Були сформовані звіти для прояснення повної картини, щоб зрозуміти актуальність даної роботи.

СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1Опис предметної області

Предметна область збору та автоматизованої обробки інформації про навчальні онлайн курси це сфера, яка охоплює всі аспекти збору, обробки та аналізу даних, пов'язаних із навчальними онлайн курсами та електронними платформами для навчання. Ця область стає все важливішою в сучасному освітньому середовищі, де інтернет та технології відіграють значущу роль. Ось деякі ключові аспекти цієї предметної області:

- Збір даних.
- Автоматизована обробка даних.
- Аналітика.
- Персоналізована рекомендація.
- Реєстрація.
- Безпека та конфіденційність даних.

Розберемо кожен аспект нашої предметної області.

Збір даних буде здійснюватись за допомогою веб-скрапінгу даних. Що таке скрапінг? Це процес автоматизованого отримання інформації з веб-сайтів. Цей процес включає в себе завантаження веб-сторінок і видобування корисної інформації з HTML-коду сторінки. Тут знадобляться навички програмування та розуміння HTML та CSS для ефективного веб-скрапінгу. На даний момент існує декілька інструментів для збирання даних з веб-ресурсів, наприклад BeautifulSoup для Python, Puppeteer для JavaScript, або різноманітні розширення для браузерів. У нашому випадку будемо використовувати Puppeteer, так як весь продукт буде написаний на JS.

Автоматизована обробка даних - це процес використання програмного забезпечення і алгоритмів для автоматизації обробки великого обсягу даних без

значного втручання людини. Цей процес може включати в себе такі завдання, як збір, аналіз, трансформація, візуалізація і збереження даних. Ось кілька з них:

1. Скрипти мов програмування: створення скриптів або програми на мовах програмування, як Python, Ruby, JavaScript тощо, щоб автоматизувати певні завдання. Наприклад, використовуючи бібліотеки Python, такі як os, subprocess, shutil, ви можете автоматизувати операції з файлами та системними викликами.
2. Системи автоматизації завдань: Існують системи автоматизації, такі як Ansible, Puppet, Chef, які призначені для автоматизації налаштувань та керування інфраструктурою.
3. Системи керування версіями: Використовуючи системи керування версіями, такі як Git, ви можете автоматизувати процес розгортання та відкату змін у коді.
4. Сервіси CI/CD: Для автоматизації процесу розробки та розгортання програмного забезпечення використовуються сервіси неперервної інтеграції/неперервної доставки (CI/CD), такі як Jenkins, Travis CI, CircleCI та інші.
5. Розподілені системи та керування контейнерами: Використовуючи рішення, такі як Docker і Kubernetes, можна автоматизувати процес розгортання та масштабування додатків.
6. Складові керування конфігурацією: За допомогою інструментів, таких як Terraform і Ansible, ви можете автоматизувати налаштування та керування інфраструктурою та ресурсами в хмарних середовищах.
7. Моніторинг і автоматизація відгуків: Використовуючи системи моніторингу, такі як Prometheus, і інші інструменти для автоматичної реакції на помилки та події.
8. Мови декларативного програмування: Інструменти, такі як Ansible, використовують декларативний підхід до автоматизації, де ви описуєте бажаний стан системи, і система забезпечує його виконання.

9. Моніторинг та логування виконання коду: Додавання журналів та моніторингу до автоматизованих процесів допомагає відслідковувати виконання та виявляти помилки.

У нашому додатку будемо використовувати Jenkins, Docker та node-cron для автоматизації самого процесу, щоб розробки сам не витрачав час кожного на запуск скрипта, що буде економити час.

Аналітика даних за допомогою JavaScript може бути виконана за допомогою бібліотек та фреймворків, що допомагають вам обробляти, аналізувати і візуалізувати дані. Ось декілька способів, як ви можете використовувати JavaScript для аналітики даних:

1. Веб-додатки для аналізу даних: створення веб-додатку за допомогою фреймворків, таких як React, Angular або Vue.js, і використовувати бібліотеки для візуалізації даних, такі як D3.js або Chart.js. Веб-додаток може завантажувати дані, аналізувати їх та надавати користувачам можливість взаємодіяти з графіками та діаграмами.
2. Обробка даних на стороні клієнта: Використання JavaScript для обробки даних на стороні клієнта. Наприклад, виконувати фільтрацію, сортування та агрегацію даних в браузері користувача.
3. Запити до API і баз даних: Використання JavaScript для виконання запитів до API або баз даних, щоб отримувати дані для подальшої аналітики. Вбудовані методи для виконання HTTP-запитів (наприклад, fetch) або бібліотеки, такі як Axios.
4. Бібліотеки для аналітики даних: Є багато бібліотек та інструментів для аналітики даних в JavaScript, такі як Pandas.js, Stats.js і інші. Вони надають різні функції для аналізу та обробки даних, включаючи обчислення статистики, агрегацію, фільтрацію тощо.
5. Використання Node.js для аналізу на сервері: Використання JavaScript за допомогою платформи Node.js для аналізу даних на сервері. Node.js надає

доступ до різних модулів для обробки і аналізу даних, і ви можете створювати автоматизовані завдання для обробки даних.

6. Візуалізація даних: Бібліотеки для візуалізації даних, такі як D3.js, для створення інтерактивних графіків і діаграм.
7. Машинне навчання та аналіз даних: Бібліотеки для машинного навчання, такі як TensorFlow.js, для аналізу даних та створення моделей машинного навчання на стороні клієнта або на сервері.

JavaScript має багато інструментів та бібліотек для аналізу даних. У проєкті буде використовуватись D3.js для створення графіків та діаграм.

Персоналізована рекомендація буде рекомендувати користувачам, який курс є актуальним на даний момент. Тут буде використовуватись звичайний фільтр, як на більшості сайтів – пошук по полярності.

Багато сайті з курсами мають особистий кабінет, де студент платформи може одразу ж проходити обраний курс та отримувати сповіщення про оновлення курсу, тобто додавання новий матеріалів, тестування і тому подібне. Наприклад, як це зроблено на популярних платформах: Udemy, Udacity, Coursera, CourseHunter, Prometheus. У додатку над яким йде дослідження буде кабінет через який користувач зможе отримувати розсилку та мати більш розгорнуту статистику проаналізованого курсу.

Безпека та конфіденційність даних дуже важлива у наш час, так як багато людей не слідкує за інформаційною гігієною. Для цього всі дані користувачів будуть шифруватись за допомогою криптографії. Шифрування паролів: зберігання паролів користувачів у зашифрованому вигляді в базі даних, наприклад, використовуючи хеш-функції, такі як bcrypt. Такий підхід допоможе захистити паролі навіть у випадку, коли база даних стає доступною зловмисникам. Використання сесій і токенів: при роботі з веб-додатками використовуйте сесії або токени для аутентифікації користувачів. Зашифрування інформації в сесіях або токенах, щоб зберегти її конфіденційність. Шифрування даних на стороні клієнта: у деяких випадках, якщо дані повинні бути доступні на

стороні клієнта (наприклад, в мобільних додатках), використання механізмів шифрування на стороні клієнта для захисту даних від несанкціонованого доступу. Використання JWT (JSON Web Tokens): JWT - це структурований формат для представлення заявок, які включають в себе дані користувача. Вони можуть бути підписані та/або зашифровані для захисту вмісту.

1.2 Аналіз наявних рішень

Наразі існує кілька вже наявних рішень для автоматизованого збору інформації про навчальні онлайн-курси. Ці рішення використовують різні технології та методи, але в цілому вони спрямовані на те, щоб зробити процес збору інформації про курси більш ефективним і масштабованим.

Одним з найпоширеніших підходів до автоматизованого збору інформації про онлайн-курси є використання веб-скраперів.

Іншим підходом до автоматизованого збору інформації про онлайн-курси є використання API. API - це інтерфейси програмування додатків, які дозволяють програмним програмам взаємодіяти з веб-сайтами. API можуть використовуватися для збирання інформації про курси, включаючи список курсів, інформацію про курси та рейтинги курсів.

Ще одним підходом до автоматизованого збору інформації про онлайн-курси є використання штучного інтелекту. ШІ може використовуватися для аналізу тексту та коду курсів, щоб отримати інформацію про них. Наприклад, ШІ може використовуватися для визначення тем, які охоплюються курсом, або для визначення рівня складності курсу.

Нижче наведено деякі конкретні приклади вже наявних рішень для автоматизованого збору інформації про навчальні онлайн-курси:

- Coursera API - це API, який дозволяє програмним програмам взаємодіяти з веб-сайтом Coursera. Coursera API можна використовувати для збирання інформації про курси, які доступні на платформі Coursera.

- edX API - це API, який дозволяє програмним програмам взаємодіяти з веб-сайтом edX. edX API можна використовувати для збирання інформації про курси, які доступні на платформі edX.
- Udemy API - це API, який дозволяє програмним програмам взаємодіяти з веб-сайтом Udemy. Udemy API можна використовувати для збирання інформації про курси, які доступні на платформі Udemy.
- Course Finder - це веб-сайт, який збирає інформацію про навчальні онлайн-курси з різних джерел. Course Finder можна використовувати для пошуку курсів за темою, типом курсу або ціною.
- Coursera Analyzer - це інструмент, який використовує ШІ для аналізу курсів Coursera. Coursera Analyzer можна використовувати для визначення тем, які охоплюються курсом, або для визначення рівня складності курсу.

Ці рішення можуть використовуватися для різних цілей, таких як:

- Пошук навчальних онлайн-курсів
- Порівняння навчальних онлайн-курсів
- Аналіз навчальних онлайн-курсів

З розвитком технологій автоматизованого збору інформації про навчальні онлайн-курси стає все простіше знайти, порівняти та проаналізувати навчальні онлайн-курси.

1.3 Постановка завдання

Були поставлені такі завдання:

1. Пошук та вибір технологій на яких розробляти ПЗ.
2. Вивчення наявних конкурентів, які вже представили на ринку свій продукт.
3. Знаходження актуальних рішень автоматизації.
4. Написання скрипту, який буде збирати дані про навчальні онлайн курси та заноси дані до БД.

5. Підготування віддаленого серверу.
6. Налаштування Docker та Jenkins на віддаленому сервері.
7. Створення додатку для користувачів.
8. Створення авторизації на сайті для користувачів.
9. Створення графіків за допомогою додатка для користувачів.
10. Створення порівняння певних курси з іншими.
11. Створення форми для зворотнього зв’язку з користучем.

Висновок першого розділу:

Система дозволяє автоматично збирати та систематизувати інформацію про різні онлайн курси, включаючи їхні назви, описи, авторів, тривалість, вартість та інші характеристики. Це полегшує пошук та порівняння курсів для студентів.

Покращена аналітика та рекомендації: Система надає можливість аналізувати дані про курси та користувачів, що дозволяє створювати персоналізовані рекомендації для студентів. Це сприяє покращенню їхнього досвіду навчання та вибору оптимальних курсів для свого розвитку.

Можливості подальшого розвитку системи, включаючи розширення функціональності, інтеграцію з іншими освітніми платформами та використання аналітичних методів для покращення навчального процесу.

МОДЕЛЮВАННЯ СИСТЕМИ

У цьому розділі розберемо два види моделювання системи та проаналізуємо їх для того, щоб зрозуміти, який краще всього імпонуватиме нам. Існує функціональне та об’єктне. Переглянемо опис цих моделювань та порівняємо їх. Також при моделюванні системи зазвичай створюються такі UML діаграми, як: активності, послідовності, прецедентів. Вони дозволяють перейти до етапів проектування, які будуть у наступному розділі.

2.1 Функціональне моделювання

Функціональне моделювання - це методологія розробки програмного забезпечення, яка зосереджена на описі функціональних аспектів системи або програми без глибокого вдавання в деталі реалізації. Цей підхід орієнтований на визначення, які функції повинна виконувати система, які дані вона обробляє і які результати повинна генерувати, без конкретизації, як саме ці функції будуть реалізовані в коді.

Основні принципи функціонального моделювання:

1. Функціональні блоки.
2. Модульність.
3. Взаємодія.
4. Нотація.
5. Абстракція.
6. Валідація.

Функціональне моделювання програмного забезпечення для автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси - це важливий етап у розробці системи. Це допомагає визначити основні функції та функціональність програмного продукту. Нижче наведено приклад функціональної моделі для такої системи:

1. Збір інформації про курси:

- Можливість збирати дані з онлайн-платформ та веб-сайтів, що надають навчальні курси.
- Автоматичне оновлення інформації про курси з регулярністю.
- Система для аналізу HTML-сторінок та витягання важливих даних, таких як назви курсів, описи, автори, тривалість тощо.

2. Зберігання та кешування даних:

- Зручне сховище для зберігання зібраної інформації про курси.
- Можливість кешування даних для швидкого доступу.

3. Аналітика даних:

- Система для аналізу зібраної інформації та виділення ключових метрик.
- Створення звітів та графіків для візуалізації аналітики.

4. Система рекомендацій:

- Визначення методів та алгоритмів для рекомендацій студентам на основі їхніх інтересів та попереднього навчання.

5. Інтерфейс користувача:

- Веб-інтерфейс для адміністраторів для налаштування та керування системою.
- Веб-інтерфейс для студентів для перегляду рекомендацій та пошуку курсів.

6. Управління користувачами та авторизація:

- Система управління ролями та правами доступу для адміністраторів, викладачів та студентів.
- Авторизація та аутентифікація користувачів.

7. Регулярне оновлення даних:

- Механізми автоматичного оновлення інформації про курси відповідно до встановленого розкладу.

8. Заходи безпеки та конфіденційності:

- Захист даних користувачів та інформації про курси від несанкціонованого доступу та витоків даних.

9. Інтеграція з іншими системами:

- Можливість інтеграції з іншими освітніми платформами та системами управління навчанням (LMS).

Ця функціональна модель допомагає визначити обсяг та основні функції системи збору та автоматизованої обробки інформації про навчальні онлайн курси. Кожна з цих функцій грає важливу роль у реалізації системи та її успішному функціонуванні. Дана модель може бути використана для розробки специфікацій системи, планування робіт та подальшої розробки програмного забезпечення для цієї мети.

2.2Об’єктне моделювання

Об'єктно-орієнтоване моделювання (ООМ) - це парадигма розробки програмного забезпечення, яка базується на ідеї моделювання програми як набору взаємодіючих об'єктів. У цьому підході об'єкти представляють собою конкретні сутності або речі з реального світу, які мають стан, характеристики та поведінку. Об'єкти можуть взаємодіяти один з одним, надсилаючи повідомлення та виконуючи методи, що модифікують їх стан.

Основні поняття об'єктно-орієнтованого моделювання включають:

1. Об'єкти.

   Об'єкти - це екземпляри класів, які представляють конкретні сутності. Кожен об'єкт має стан у вигляді полів або властивостей та поведінку у вигляді методів, які операційно змінюють стан об'єкта або взаємодіють з іншими об'єктами.

2. Класи.

   Класи визначають структуру та характеристики об'єктів. Вони визначають поля (властивості) та методи, які спільні для всіх об'єктів цього класу. Клас служить шаблоном для створення об'єктів.

3. Інкапсуляція.

   Інкапсуляція - це ідея, що дані та функції, які працюють з цими даними, повинні бути об'єднані в одному об'єкті, і доступ до них повинен контролюватися через інтерфейс об'єкта. Це приховує деталі реалізації від користувача об'єкта.

4. Спадкування.

   Спадкування дозволяє створювати нові класи на основі існуючих. Підкласи успадковують властивості та методи батьківського класу і можуть розширювати або перевизначати їх.

5. Поліморфізм.

   Поліморфізм дозволяє використовувати об'єкти різних класів зі спільним інтерфейсом однаковим чином. Це спрощує створення універсальних методів, які працюють з різними типами об'єктів.

Об'єктно-орієнтоване моделювання зазвичай застосовується для розробки складних систем, оскільки дозволяє виражати взаємодію різних компонентів системи у вигляді об'єктів та їх взаємодій. Цей підхід сприяє структурній організації коду, спрощує його підтримку та розширення, а також полегшує відновлення іншим розробникам, які можуть працювати з об'єктами, не знаючи всіх деталей їхньої реалізації.

1. Різниця між об'єктно-орієнтованим моделюванням та функціональним. це два різних підходи до розробки програмного забезпечення, які мають свої власні особливості та переваги. Ось деякі основні відмінності між ними: Підхід до даних:

   - Функціональне моделювання орієнтоване на функції та операції, які виконуються над даними. У цьому підході дані вважаються

незмінними, і змінність досягається шляхом створення нових даних на основі старих.

- Об'єктне моделювання орієнтоване на об'єкти, які містять дані та функції, що працюють з цими даними. Об'єкти представляють собою самодостатні сутності, які можуть бути оновлені та взаємодіяти один з одним.

2. Інкапсуляція:
    - Об'єктне моделювання сприяє інкапсуляції, що означає, що дані та функції, що їх обробляють, знаходяться в одному об'єкті, і доступ до них контролюється через інтерфейс об'єкта.
    - Функціональне моделювання не так сильно підтримує інкапсуляцію, оскільки функції та дані можуть бути розділені.
3. Повторне використання:
    - Об'єктне моделювання полегшує повторне використання коду, оскільки об'єкти можуть бути легко використані в різних частинах програми або навіть в інших програмах.
    - Функціональне моделювання також дозволяє повторне використання коду, але менше сприяє створенню самодостатніх, перевикористовуваних модулів.
4. Зміна даних:
    - В об'єктному моделюванні змінність даних досягається за допомогою методів об'єктів, що модифікують їх стан.
    - У функціональному моделюванні змінність даних досягається за допомогою створення нових даних на основі старих, а не безпосередньої зміни даних.
5. Простота і читабельність:

- Функціональне програмування може бути більш простим та читабельним для деяких завдань, оскільки функції можуть бути досить прямими та декларативними.
- Об'єктне програмування може бути корисним для складних систем, де розподілення функцій та даних на об'єкти полегшує управління складністю програми.

Для нашого проєкту більше підходить об'єктне моделювання.

Для моделювання використовуються UML діаграми, що відіграють головну роль при проектуванні проєкту.

UML (Unified Modeling Language) - це стандартизована мова для моделювання програмних систем та процесів, яка широко використовується в галузі розробки програмного забезпечення. UML надає набір графічних символів і конвенцій для створення діаграм, які допомагають розробникам, аналітикам та іншим учасникам проекту легше розуміти, аналізувати і документувати програмні системи. UML дозволяє створювати візуальні моделі, які включають в себе різні аспекти системи, такі як її структура, поведінка, взаємодія та інше.

UML є потужним інструментом для аналізу та проектування програмних систем і допомагає створювати зрозумілі і документовані моделі для спілкування між різними учасниками проекту. Він широко використовується в галузі розробки програмного забезпечення та є стандартом в цій галузі.

Діаграма прецедентів (Use Case Diagram) є одним із видів діаграм UML і використовується для моделювання функціональності системи з точки зору зовнішніх акторів і взаємодії між ними. Діаграма прецедентів допомагає визначити, як користувачі (актори) взаємодіють з системою та як система реагує на їхні запити.

Основні елементи діаграми прецедентів включають в себе:

1. Прецедент (Use Case): Прецедент представляє сценарій взаємодії між акторами і системою. Кожен прецедент має назву і опис того, що відбувається в цьому сценарії.

2. Актор (Actor): Актори представляють зовнішніх сутностей, які взаємодіють з системою. Це можуть бути користувачі, інші системи, зовнішні об'єкти або ролі користувачів.
3. Взаємозв'язок (Association): Взаємозв'язки показують, які прецеденти доступні для якого актора. Вони вказують на взаємодію між акторами і прецедентами.
4. Система (System): У діаграмі прецедентів систему зазвичай представляють як прямокутник, який містить всі прецеденти і акторів. Система - це той компонент, який надає функціональність для акторів.

Діаграма прецедентів допомагає створити зрозумілу модель функціональності системи, і вона використовується на ранніх етапах проектування для з'ясування вимог користувачів і зовнішніх систем. Ця діаграма спрощує комунікацію між розробниками, аналітиками та іншими учасниками проекту і допомагає визначити, які функції повинна виконувати система. Нижче наведена діаграма прецедентів.

Опис діаграми:

Система має 4 акторів та 10 прецедентів. Актори та прецеденти мають між собою взаємозв’язки. Актори :

1. Користувач
2. Модератор
3. Аналітик
4. Керівник платформи

Користувач має зв’язки з такими прецедентами, як:

- Створення особистого кабінету
- Пошук курсів на платформі
- Перегляд курсів на платформі
- Ознайомлення з вибраним курсом

- Купівля курсу

Модератор пов’язаний з:

- Управлінням інформаціх про курс
- Підтриманням зворотнього зв’язку між користувачами

Аналітик може:

- Формувати звіти, KPI
- Переглядати звіти, KPI

Керівник має можливості:

- Приймати рішення щодо управління платформою
- Переглядати звіти, KPI

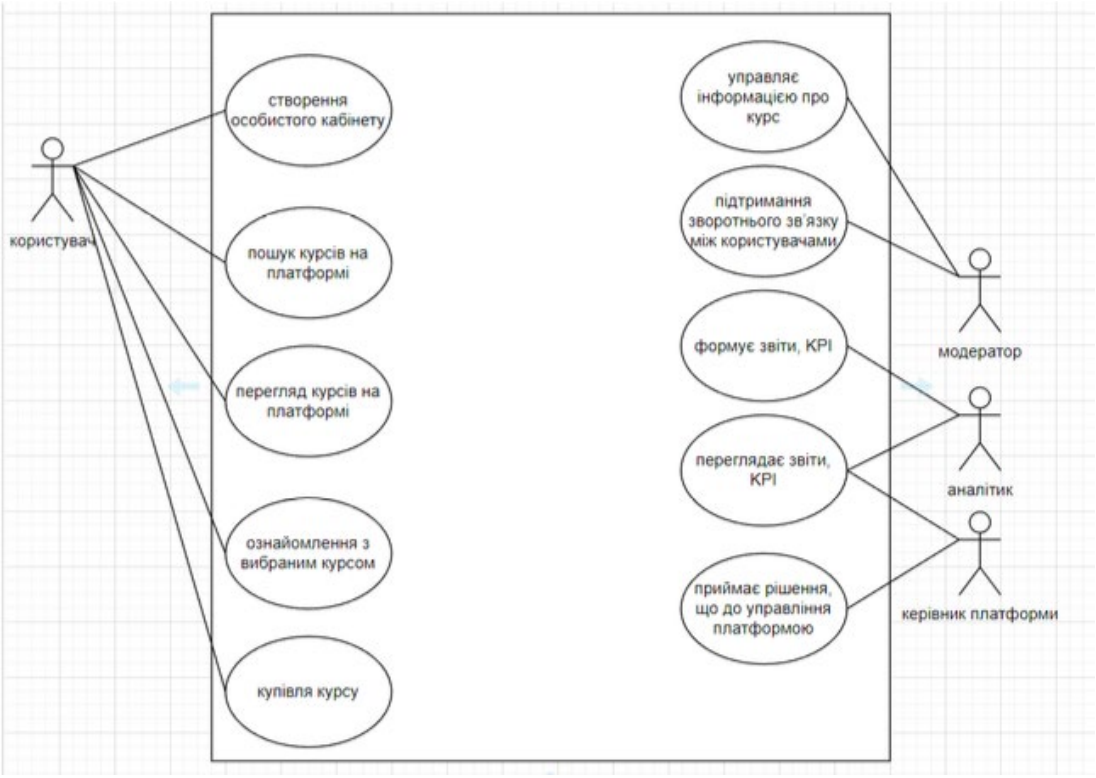

Рис. 1 діаграма прецедентів

Діаграма послідовності (Sequence Diagram) є однією з діаграм UML і використовується для моделювання послідовності виконання операцій і обміну

повідомленнями між об'єктами в системі. Вона допомагає ілюструвати, як об'єкти взаємодіють один з одним в зазначеному порядку, і яким чином відбувається виконання певної функціональності.

Основні елементи діаграми послідовності включають в себе:

1. Об'єкти: Об'єкти, які беруть участь у послідовності виконання, представлені на діаграмі великими прямокутниками з назвами. Це можуть бути екземпляри класів або інші об'єкти системи.
2. Лінії життя (Lifelines): Лінії життя відображають існування об'єктів протягом певного інтервалу часу. Вони зображаються у вигляді вертикальних ліній, які перетинаються з повідомленнями.
3. Повідомлення (Message): Повідомлення показують обмін інформацією і виклик методів між об'єктами. Повідомлення можуть бути синхронними (прямими викликами методів) або асинхронними (повідомленнями, які відправляються і обробляються пізніше).
4. Фрагменти (Fragments): Фрагменти, такі як умовні ітерації або альтернативи, використовуються для моделювання різних умов та можливих шляхів виконання послідовності.
5. Оператори часу (Time Constraints): Оператори часу можуть бути додані до повідомлень, щоб вказати часові обмеження на виконання операцій.

Діаграма послідовності допомагає розробникам і аналітикам легше розуміти взаємодію між об'єктами та порядок виконання операцій в системі. Вона особливо корисна для аналізу та документування послідовностей подій в системі, і вона часто використовується на етапі проектування програмного забезпечення.

Рис. 2 діаграма послідовності

Діаграма активності (Activity Diagram) є однією з діаграм UML і використовується для моделювання послідовності дій або процесів в системі. Вона дозволяє ілюструвати різноманітні аспекти виконання дій, включаючи взаємодію між об'єктами, різні шляхи виконання, умови та обробку помилок.

Основні елементи діаграми активності включають в себе:

1. Вузли (Nodes): Вузли відображають окремі дії або операції, які виконуються в системі. Їх представляють у вигляді прямокутників або еліпсів з назвами.
2. Ребра (Edges): Ребра показують переходи між вузлами і вказують послідовність виконання дій. Ребра можуть мати напрямок, що вказує на потік виконання.
3. Вилки і злиття (Forks and Joins): Вилки (forks) використовуються для розділення потоку виконання на багато паралельних гілок, тоді як злиття (joins) об'єднують ці гілки назад.
4. Вибори (Decisions): Вибори використовуються для визначення умовних переходів в залежності від певних умов.

5. Введення і виведення (Input and Output): Ці елементи вказують на вхідні та вихідні дані в процесі виконання.
6. Об'єкти підпроцесів (Subprocess Objects): Діаграма активності може містити в собі об'єкти підпроцесів для моделювання виконання інших активностей на вищому рівні.

Діаграма активності допомагає розробникам та аналітикам уявити послідовність дій та процесів в системі. Вона особливо корисна для моделювання бізнес-процесів, алгоритмів або інших складних послідовностей. Діаграма активності допомагає зрозуміти, як система реагує на події та умови та яким чином виконуються операції в системі.

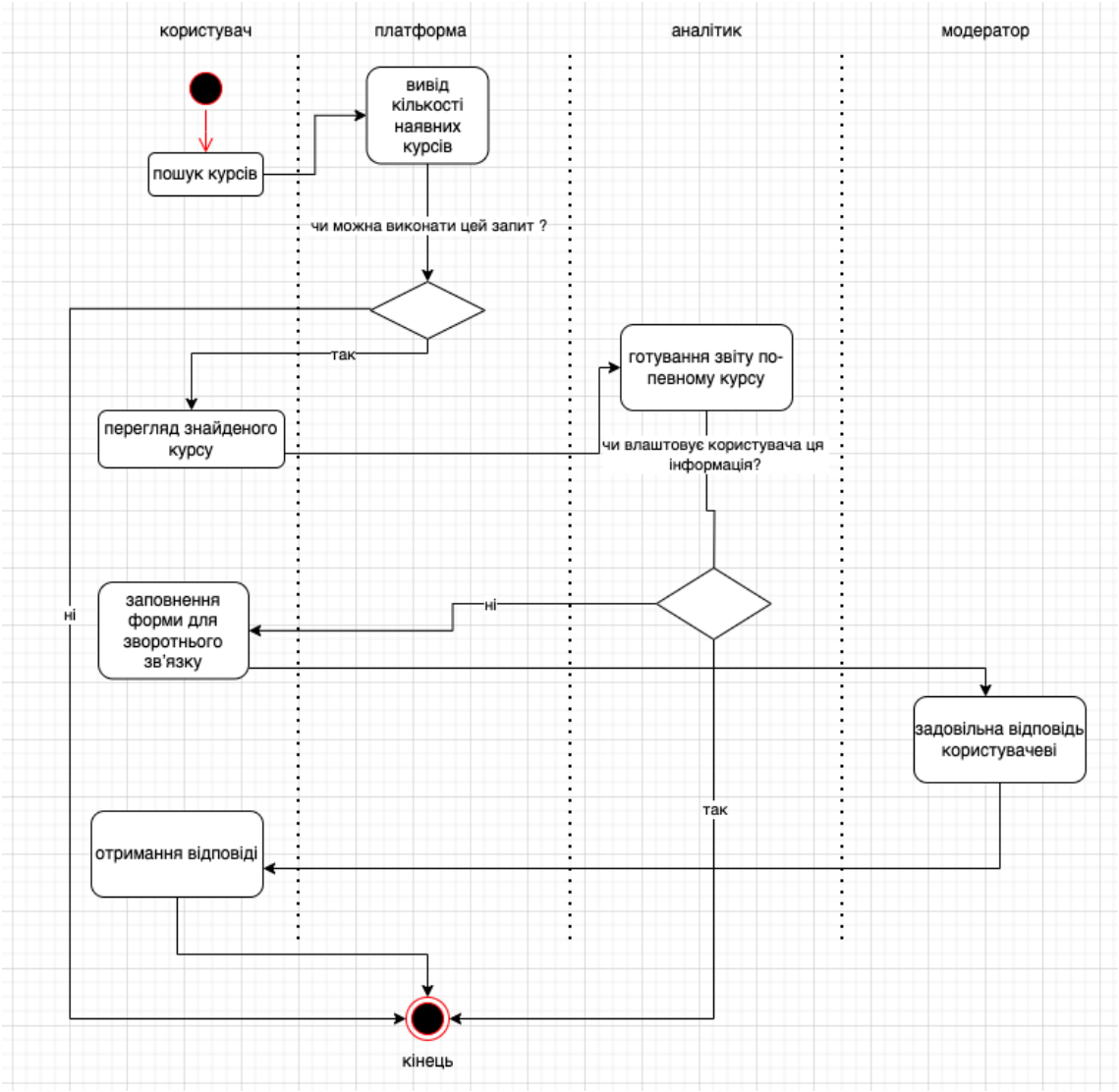

Рис. 3 Діаграма активності

Ця об'єктна модель відображає структуру системи та її ключові компоненти. Об'єкти взаємодіють між собою для забезпечення функціональності збору та аналітичної обробки інформації про навчальні онлайн курси. Кожен клас має свої методи та властивості, які допомагають здійснити конкретні операції та функції в системі.

Висновки до розділу.

У даному розділі був проведений аналіз між двома типами моделювання (об’єктним та функціональним). Також були побудовані діаграми послідовності, активності та прецедентів.

РОЗРОБКА СИСТЕМИ

Розробка системи - це процес створення програмного забезпечення або апаратно-програмних комплексів з метою вирішення конкретних завдань або вироблення функціональності для певної мети. Цей процес включає в себе кілька етапів і може бути спрямованим на створення різноманітних систем, включаючи програмні додатки, веб-сервіси, операційні системи, бази даних, інформаційні системи, та інше.

3.1 Основні етапи розробки

Розробка програмного забезпечення для автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси може включати в себе кілька основних етапів. Нижче наведено опис кожного з цих етапів:

1. Аналіз вимог:
    - Збір вимог від користувачів та зацікавлених сторін, таких як викладачі, студенти та адміністратори.
    - Визначення функціональних і нефункціональних вимог для системи.
    - Встановлення критеріїв успіху та мети проекту.
2. Проектування:
    - Розробка архітектурної концепції системи, включаючи базу даних, інтерфейс користувача, логіку програми та інше.
    - Визначення структури бази даних для збереження інформації про курси, студентів, викладачів і т. д.
    - Розробка дизайну користувальницького інтерфейсу.
3. Реалізація (програмування):
    - Написання програмного коду, який забезпечує функціональність для збору інформації про курси, реєстрації студентів і викладачів, ведення журналу подій і т. д.

- Створення бази даних і реалізація системи для збереження та керування інформацією.

4. Тестування:
    - Проведення тестів для впевненості в тому, що програмне забезпечення працює правильно та задовольняє вимоги.
    - Виявлення та виправлення помилок та недоліків у програмному коді.
    - Тестування взаємодії між різними компонентами системи.
5. Впровадження:
    - Встановлення програмного забезпечення на серверах або хостингу для доступу в Інтернеті.
    - Налаштування системи для роботи в реальних умовах.
    - Запуск системи та навчання користувачів, включаючи викладачів і студентів.
6. Підтримка та супровід:
    - Підтримка та обслуговування системи в режимі реального часу.
    - Виправлення помилок, оновлення та розвиток програмного забезпечення відповідно до нових вимог і потреб користувачів.
    - Моніторинг і аналітика роботи системи для забезпечення її ефективності та вдосконалення.

Ці етапи розробки можуть вимагати співпраці між різними командами фахівців, такими як програмісти, аналітики, дизайнери, тестувальники та інші спеціалісти. Проведення систематичного тестування та забезпечення безпеки даних також важливі у контексті розробки системи для автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-

3.2 Аналіз вимог

Розпочнемо з аналізу вимог до нашого програмного забезпечення

Аналіз вимог до програмного забезпечення для автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси є важливим етапом у розробці такої системи. Перед початком розробки необхідно чітко визначити, які функції та характеристики має мати система, щоб задовольнити потреби користувачів та досягти поставлених цілей. Ось кілька ключових аспектів аналізу вимог:

1. Функціональні вимоги:
    - Збір і обробка даних.
    - Реєстрація користувачів.
    - Пошук курсів.
    - Аналітика.

Збір і обробка даних буде реалізуватись за скриптів на JS. Для цього буде використовуватись puppeteer. Puppeteer - це Node.js бібліотека, яка дозволяє автоматизувати браузер Google Chrome, включаючи збір та обробку даних з веб-сторінок. За допомогою Puppeteer ви можете виконувати такі завдання, як витягування даних з веб-сайтів, заповнення форм, взаємодія з веб-сторінками, а також генерація знімків сторінок. Інформація з сторінок збирається за допомогою HTML-селекторів. Якщо на сайті є пагінація, то за допомогою вбудованої функції він може автоматично заходити на наступні сторінки, але за їх наявності.

```
const puppeteer = require('puppeteer'),
      fs = require('fs');


(async () => {
   const browser = await puppeteer.launch( options: {
      headless: true,
      defaultViewport: false,
      userDataDir: './tmp'
   });
   const page = await browser.newPage();

   await page.goto( url: "https://courseforyou.com.ua/ua/dlsfjjklsadf/c80004/producer=javascript/",
         options: {waitUntil: "load"});

   const listingHandles = await page.$$( selector: 'rz-grid.ng-star-inserted li');

   for (listingHandle of listingHandles) {
      let isButtonEnabled = false;
      while (!isButtonEnabled){

          let title = 'null',
              price = '0',
              link = 'null';
```

Рис.1 написання скрипту для парсингу інформації з певного сайту.

Рис.2 частина коду, де використовуються селектори для діставання певної інформації.

Реєстрація користувачів буде максимально простою. Для цього буде створена форма, яка буде запитувати у користувача логін та пароль для реєстрації, також для входу. У коді буде поєднуватись EJS це шаблонізатор, який дає змогу писати

скрипти в середині HTML-коду, що є дуже зручним інструментом див.рис.3.

```
<%- include('partials/header'); -%>

<form>
    <h2>Вхід</h2>
    <label for="email">Пошта (голубіна не підійде)</label>
    <input type="email" name="email" placeholder="ваша пошта" required>
    <div class="email error"></div>
    <label for="password">Пароль</label>
    <input type="password" name="password" placeholder="ваш надійний пароль" required>
    <div class="password error"></div>
    <button>Вхід</button>
</form>


<script>
    const form = document.querySelector('form');
    const emailError = document.querySelector('.email.error');
    const passwordError = document.querySelector('.password.error');
    form.addEventListener('submit', async (e) => {
        e.preventDefault();


        emailError.textContent = '';
        passwordError.textContent = '';
```

Рис. 3 код за допомогою якого реалізовувалась авторизація для користувачів.

Також для авторизації використовувався JWT-токен. JWT (JSON Web Token) - це популярний механізм автентифікації та авторизації в веб-додатках. Він дозволяє створювати токени, які містять інформацію про користувача та дозволяють підтверджувати його ідентифікацію без необхідності зберігати стан на сервері. Ось кілька кроків, які вам потрібно виконати для реалізації авторизації з JWT:

1. Генерація JWT: Після успішної аутентифікації користувача на сервері, створіть JWT, який містить інформацію про користувача (ідентифікатор, роль, термін дії тощо). Підпишіть цей токен за допомогою секретного ключа.

2. Надсилання JWT клієнту: Передайте створений JWT клієнту, зазвичай через заголовок відповіді або як частину тіла відповіді після аутентифікації.
3. Збереження JWT на клієнтській стороні: Клієнт повинен зберегти отриманий JWT, зазвичай у сховищі, такому як localStorage або cookies.
4. Надсилання JWT з кожним запитом: При кожному запиті до захищених ресурсів, клієнт повинен включати JWT в заголовок запиту або в інший безпечний спосіб.
5. Перевірка та розшифрування JWT на сервері: На сервері перевірте, чи JWT підписаний справжнім секретним ключем. Якщо перевірка пройшла успішно, розшифруйте токен і отримайте інформацію про користувача.
6. Перевірка прав доступу: Після отримання інформації про користука, перевірте його права доступу до ресурсу. Врахуйте ролі та дозволи, які вказані в JWT.
7. Відповідь на запит: Відповідь на запит відправляється разом з інформацією про користука, яка була отримана з JWT.

Це загальний опис процесу використання JWT для авторизації. JWT може бути використаний для створення безпечного і масштабованого механізму авторизації в вашому веб-додатку. Переконайтеся, що ви дотримуєтеся кращих практик забезпечення JWT, включаючи правильну настройку і зберігання секретних ключів, а також валідацію токенів перед виконанням дій користувача.

Пошук курсів буде зазвичай, як і на всіх інших сайтах, так як це є дуже зручним для користувачів. Кожен зможе ввести назву потрібного їм курсу, що дасть змогу потім бачити готовий для них запит з потрібною інформацією, яку вони шукали.

Аналітика це дуже важливо для користувача та модератора сайту. Для споживача буде аналізуватись курс по ціні, як вона зростала чи падала протягом часу, кількість студентів з усіх доступних платформ, які на них навчаються та яка тенденція їх зростання і кількість зацікавлених осіб у придбанні даного

навчання. Тобто наглядний вигляд попиту на цю чи іншу пропозицію. Модератор буде мати трохи іншу статистику, скільки користувачів зареєструвалось, яка їх тенденція, з яких вони країн, як часто користуються додатком для пошуку потрібної інформації. Для аналітики буде використовуватись D3.js

Рис. 4 D3.js

2. Нефункціональні вимоги:
    - Безпека.
    - Швидкодія.

Насамперед у наш час дуже важлива безпека у інформаційному просторі. Так, як є різні типи хакерів, які можуть заволодіти інформацією. Треба вберегти від SQL-ін’єкцій див.рис. 5. Це поширений вектор атаки, який використовує зловмисний код SQL для маніпулювання серверною базою даних для доступу до інформації, яка не призначена для відображення.

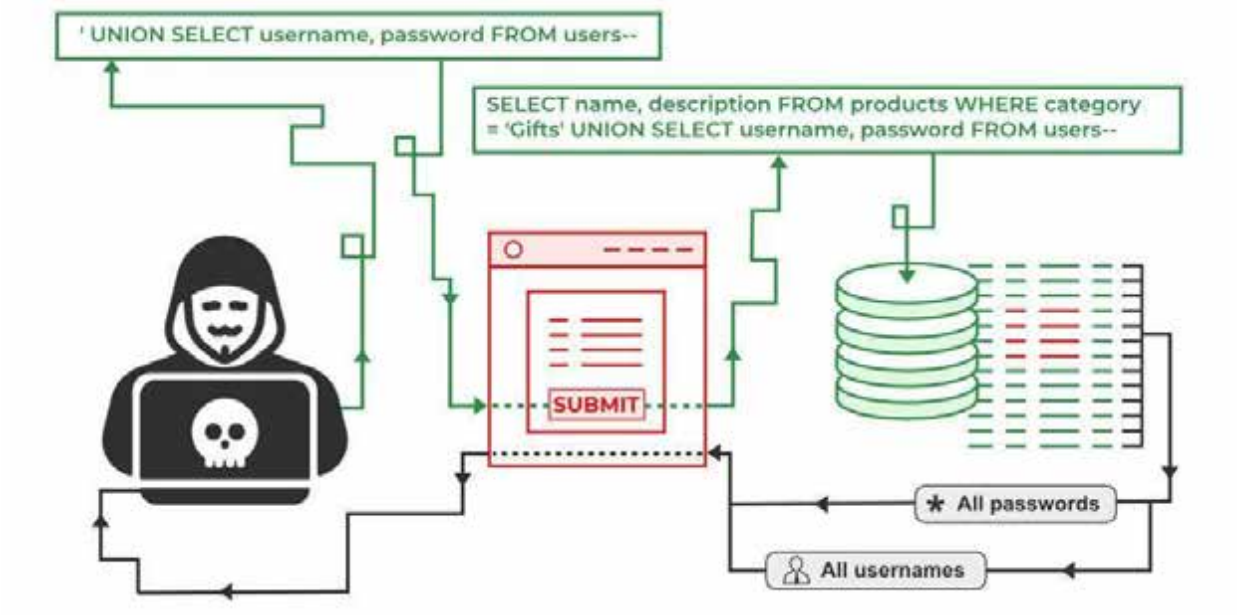

Рис. 5 наглядний приклад роботи sql-ін’єкцій

Для безпеки користувачів, також буде використане шифрування паролів за допомогою bcrypt. Це бібліотека для хешування паролів в програмуванні. Вона допомагає забезпечити безпеку паролів, зберігаючи їх у захищеному вигляді у базі даних. Bcrypt використовує алгоритм хешування, який є спеціально призначеним для збереження паролів та інших конфіденційних даних.

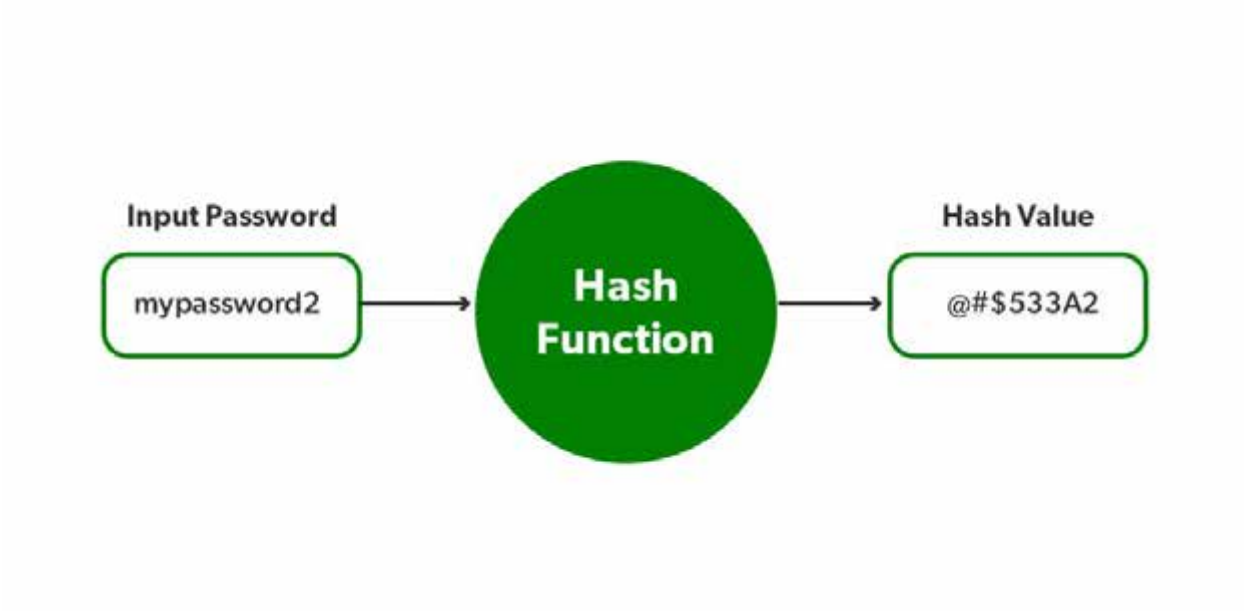

Рис. 6 як працює хешування паролю.

Швидкодія є дуже важливим аспектом і її треба забезпечити для комфорту споживача. Ось які способи можна використати для реалізації цього:

- Оптимізація запитів до бази даних: Якщо ваш додаток взаємодіє з базою даних, важливо оптимізувати запити до бази даних, використовуючи індекси, кешування, агрегацію даних та інші техніки для покращення швидкодії запитів.
- Кешування даних: Використання кешування для збереження проміжних результатів або найбільш запитуваних даних може значно прискорити додаток. Кешування можна реалізувати на різних рівнях, від кешування в пам'яті до використання систем кешування, таких як Redis.
- Мінімізація завантаження сторінки: Для веб-додатків важливо мінімізувати завантаження сторінок. Використовуйте компресію даних, зображень і статичних ресурсів. Мінімізуйте кількість запитів на сторінку та використовуйте кешування клієнта.
- Використання CDN (Content Delivery Network): Використання CDN дозволяє розподіляти статичні ресурси по всьому світові, зменшуючи час завантаження сторінок для користувачів у різних географічних областях.
- Асинхронність та паралельність: Використання асинхронних запитів та операцій може покращити відгукливість додатку. Розгляньте використання розробки подій, обіцінок або потоків, якщо це необхідно.
- Масштабування: Якщо ваш додаток має високе навантаження, розгляньте можливість масштабування додатку на багато серверів або в хмарному середовищі для розділення навантаження.

- Профілювання та оптимізація коду: Використовуйте інструменти профілювання для визначення найбільших точок оптимізації в коді. Оптимізуйте ці частини коду для покращення продуктивності.
- Контроль пам'яті: Уникайте утечок пам'яті та зайвого використання пам'яті. Пам'ять важлива для швидкодії та стабільності додатку.
- Стиснення даних: Використовуйте стиснення для передачі даних між клієнтом і сервером, що зменшу

3. Вимоги до інтерфейсу користувача:
   - Розробка інтерфейсів для користувачів.
   - Забезпечення зручного та інтуїтивно зрозумілого користування системою.
4. Вимоги до бази даних:
   - Проектування структури бази даних для збереження інформації.

У нашому додатку будемо використовувати реляційну та нереляційну базу даних. Пояснення та різниця між ними буде відображена у наступних реченнях.

Реляційна база даних (РБД) - це тип бази даних, яка організована у вигляді таблиць (реляцій), де дані зберігаються у вигляді рядків та стовпців. Кожна таблиця представляє собою конкретний вид даних або сутність, а кожен рядок у таблиці відповідає конкретному запису або об'єкту даних, а стовпці визначають атрибути цієї сутності. Основні поняття та характеристики реляційних баз даних включають: таблиці: база даних складається з різних таблиць, де кожна таблиця має унікальне ім'я і визначає тип даних, які в ній зберігаються. рядки (кортежі): кожен рядок в таблиці представляє собою окремий запис або кортеж. Кожен кортеж має унікальний ідентифікатор або ключ, який допомагає звертатися до нього. Стовпці (атрибути): стовпці таблиці визначають атрибути даних, які зберігаються в таблиці. Кожен стовпець має ім'я і визначений тип даних.

Ключі: реляційні бази даних використовують ключі для ідентифікації та пов'язування даних між таблицями. Основні види ключів включають первинний ключ (Primary Key) і зовнішній ключ (Foreign Key). запити SQL: Для взаємодії з реляційною базою даних використовують мову структурованих запитів SQL (Structured Query Language). SQL дозволяє виконувати різноманітні операції, такі як додавання, оновлення, видалення та витягування даних. Нормалізація: реляційні бази даних дотримуються принципів нормалізації для уникнення дублювання даних та підтримки цілісності даних. Нормалізація допомагає розподілити дані на окремі таблиці для забезпечення їхньої ефективної організації. Транзакції: реляційні бази даних підтримують транзакції для забезпечення атомарності, цілісності та узгодженості даних. Транзакції дозволяють виконувати групу операцій як атомарну одиницю.

Популярні реляційні бази даних включають такі системи, як MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server і SQLite. Реляційні бази даних широко використовуються для збереження структурованих даних у багатьох видх програм, від веб-додатків до корпоративних систем.

Нереляційна база даних (NoSQL база даних) - це тип бази даних, який відрізняється від реляційних баз даних (РБД) тим, що в них дані не зберігаються у вигляді табличних структур, але надходять у більш різноманітних форматах, таких як документи, ключ-значення, колекції, графи тощо. Цей підхід дозволяє більше гнучкості та швидкодії при зберіганні та опрацюванні даних.

Основні характеристики нереляційних баз даних включають: Різновиди типів даних: нереляційні бази даних підтримують різноманітні типи даних, такі як документи, JSON, XML, ключ-значення, колекції графів тощо. Це дозволяє зберігати навіть неструктуровані дані. Гнучкість схеми: нереляційні бази даних зазвичай допускають гнучку схему, що означає, що різні записи в одній колекції можуть мати різні поля без обов'язковості

визначення статичної схеми, як у РБД. Швидкодія: нереляційні бази даних нерідко мають декілька оптимізацій, які дозволяють досягти високої швидкодії при роботі з даними, особливо при великих обсягах даних. Розподілені системи: деякі NoSQL бази даних підтримують розподілені системи, що дозволяє їм масштабуватися горизонтально на багато серверів. Гнучке зберігання даних: NoSQL бази даних надають змогу зберігати дані в більш природному для додатку форматі, спрощуючи розробку. Підтримка для різних структур даних: NoSQL бази даних можуть підтримувати структури даних, такі як дерева, графи та інші, що робить їх підходящими для різних видів додатків. Популярні NoSQL бази даних включають MongoDB (документ-орієнтована), Redis (ключ-значення), Neo4j (графова), Cassandra (колонкова) та багато інших. Вибір між реляційною і нереляційною базою даних залежить від конкретних вимог вашого додатку. Реляційні бази даних найбільш підходять для додатків з визначеною структурою даних, тоді як NoSQL бази даних дозволяють зберігати та обробляти дані з більшою гнучкістю.

У нашому випадку будемо використовувати реляційну базу даних MySQL та нереляційну MongoDB.

На рис.7 зображена спроектована схема бази даних, яка реалізована у MySQL. Ця схема використовувала такі типи зв’язків, а саме:

1. Один-до-одного (One-to-One): У цьому виді зв'язку один запис в одній таблиці відповідає одному запису в іншій таблиці. Наприклад, це може бути відношення між користувачем і його паспортом, де у кожного користувача є лише один паспорт, і навпаки.
2. Один-до-багатьох (One-to-Many): У цьому випадку один запис в одній таблиці відповідає багатьом записам в іншій таблиці. Наприклад, відношення між користувачем і його замовленнями, де кожен користувач може мати багато замовлень, але кожне замовлення належить лише одному користувачеві.

3. Багато-до-багатьох (Many-to-Many): У цьому виді зв'язку багато записів в одній таблиці відповідають багатьом записам в іншій таблиці. Зазвичай для реалізації такого зв'язку використовується додаткова таблиця, яка містить пари ідентифікаторів з обох таблиць. Наприклад, відношення між студентами і курсами, де кожен студент може вивчати багато курсів, і кожен курс може мати багато студентів.
4. Самовідношення (Self-Referencing): Цей вид зв'язку виникає, коли таблиця посилається на себе. Наприклад, в таблиці "організація" може бути поле, яке посилається на інший запис в тій же таблиці для визначення батьківської організації.
5. Зовнішній ключ (Foreign Key): Зовнішній ключ - це зв'язок, який використовується для створення зв'язку між таблицею та іншою таблицею. Зовнішній ключ вказує на первинний ключ іншої таблиці та допомагає встановити зв'язок між ними.

   Зв'язки в реляційних базах даних допомагають встановлювати взаємозв'язки між різними таблицями, що дозволяє виконувати складні запити та отримувати зв'язану інформацію з різних джерел. Правильне визначення і належне управління зв'язками допомагає зберігати та використовувати дані в реляційних базах даних ефективно та логічно.

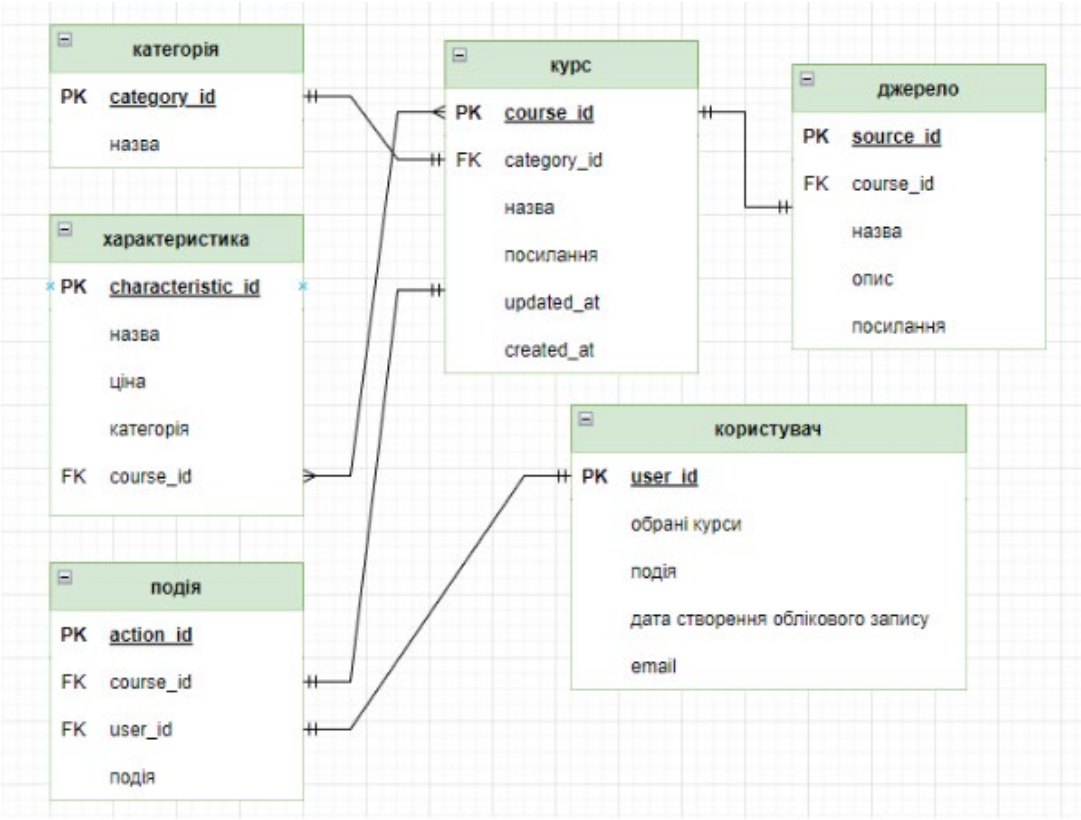

Рис. 7 схема реляційної бази даних

MongoDB буде використовуватись для збреження інформації про авторизованих користувачів та буде мати їхні захешовані паролі. Для цього було створено кластер на mongodb.com на віддаленому сервері.

Рис. 8 створений кластер на mongodb.com.

Також треба у коді реалізувати підключення до нашої NoSQL бази даних. Для цього використовувався Mongoose. Mongoose ODM (Object Data Modeling) - це популярна бібліотека на JavaScript, яка надає схемно-орієнтоване рішення для моделювання і взаємодії з MongoDB, базою даних NoSQL. MongoDB - це документова база даних, яка зберігає дані у гнучкому форматі, схожому на JSON,

і Mongoose спрощує роботу з MongoDB, додаючи структуру і валідацію до ваших даних.

Ключові функції та концепції Mongoose ODM включають:

1. Визначення схеми: За допомогою Mongoose ви визначаєте схему для моделей даних. Схема визначає структуру документів у колекції, вказуючи поля, типи даних і правила валідації.
2. Створення моделі: Mongoose дозволяє створювати моделі для ваших даних, які відповідають схемі. Модель використовується для взаємодії з даними в базі даних, включаючи створення, читання, оновлення та видалення документів.
3. З'єднання з MongoDB: Mongoose забезпечує можливість підключення до бази даних MongoDB і взаємодії з нею за допомогою моделей і схем.
4. Підтримка Middleware: Mongoose надає можливість встановлення middleware, які дозволяють виконувати певні дії перед або після збереження, оновлення, видалення тощо документів.

Mongoose ODM допомагає розробникам створювати структуровані та валідовані моделі даних для MongoDB і спрощує взаємодію з цією документовою базою даних з використанням JavaScript. На рис.9 можна побачити реалізацію у коді.

Рис. 9 використання Mongoose.

5. Інші специфічні вимоги:

- Вимоги до забезпечення сумісності з різними платформами та пристроями.

Сумісність буде можлива майже для всіх пристроїв: планшет, телефон, ПК. Для цього застосовувалась Кросплатформена розробка (Cross-Platform Development). це підхід до розробки програмного забезпечення, який дозволяє створювати додатки, які працюють на різних операційних системах та платформах без необхідності переписувати весь код для кожної окремої платформи. Ключовою метою кросплатформеної розробки є максимізація використання коду та ресурсів для розробки додатків для різних цільових платформ, таких як iOS, Android, Windows, macOS і інші. Для цього був використаний підхід - веб-програмування: веб-додатки, написані з використанням HTML, CSS та JavaScript, можуть бути доступні на різних платформах завдяки веб-браузерам. Такі додатки можна запускати на комп'ютерах, планшетах і смартфонах без необхідності переписування коду.

Після проведення аналізу вимог рекомендується створити документ зі специфікацією вимог (Software Requirements Specification), який буде служити основою для подальшої розробки системи. Такий документ допомагає зрозуміти та узгодити очікування всіх зацікавлених сторін і забезпечити успішну розробку програмного забезпечення для автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси.

3.3 Проектування системи.

У проектування системи використовувалась розподілена архітектура (Distributed Architecture).

Розподілена архітектура (Distributed Architecture) - це архітектурний підхід, в якому система розбивається на окремі компоненти, які розташовані на різних фізичних або логічних вузлах та взаємодіють між собою через мережу. Основна ідея розподіленої архітектури полягає в тому, щоб розділити функціональність

та обов'язки системи на менші компоненти, які можуть співпрацювати для досягнення загальних цілей.

Основні характеристики розподіленої архітектури включають:

1. Розділення функціональності: Розподілена система складається з різних компонентів, кожен з яких відповідає за певний аспект функціональності. Наприклад, можуть бути окремі сервіси для обробки запитів від клієнтів, для збереження даних, для аутентифікації тощо.
2. Комунікація: Компоненти взаємодіють між собою через мережу, використовуючи різні протоколи та механізми комунікації. Це може включати в себе HTTP для веб-сервісів, RPC (Remote Procedure Call) для віддалених викликів, або спеціалізовані протоколи обміну даними.
3. Масштабованість: Розподілена архітектура дозволяє легко масштабувати окремі компоненти системи для вирішення зростаючого обсягу роботи. Це може бути досягнуто шляхом додавання нових екземплярів компонентів або розподілення навантаження.
4. Надійність: Розподілена архітектура може бути більш надійною через можливість резервного копіювання даних і компонентів, а також через відновлення в разі виникнення помилок або збоїв в одному з компонентів.
5. Гнучкість: Розподілена архітектура дозволяє гнучко вносити зміни в окремі компоненти без впливу на інші частини системи.

Прикладами систем, які використовують розподілену архітектуру, є сучасні веб-додатки, мікросервісні архітектури, розподілені бази даних, системи для обробки великих обсягів даних тощо. Розподілена архітектура дозволяє ефективно робити великі та складні системи більш масштабованими, надійними та гнучкими.

Розподілена архітектура для програмного забезпечення автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси може бути створена на основі різних компонентів, які взаємодіють між собою через мережу. Ось можливий приклад архітектури для такої системи:

1. Веб-інтерфейс (Web Interface):
    - Це веб-додаток, який дозволяє користувачам шукати та реєструватися на онлайн-курси.
    - Веб-інтерфейс може бути реалізований як окремий компонент або навіть як набір мікросервісів.
2. Сервери інформації про курси (Course Information Servers):
    - Ці сервери містять дані про доступні курси, включаючи інформацію про назви курсів, викладачів, тривалість, вартість тощо.
    - Дані можуть зберігатися в розподіленій базі даних або іншому сховищі.
3. Система для аналітичної обробки (Analytical Processing System):
    - Ця система відповідає за аналітичну обробку інформації про користувачів та їх активність на курсах.
    - Вона може включати в себе компоненти для збору та аналізу даних, в тому числі інструменти для створення звітів та візуалізації даних.
4. Сервіс аутентифікації та авторизації (Authentication and Authorization Service):
    - Цей сервіс відповідає за перевірку ідентифікації користувачів і надає доступ до системи на основі їх прав доступу.
5. Мікросервіси для інтеграції з іншими системами:
    - Якщо ваша система потребує інтеграції з іншими послугами або зовнішніми джерелами даних, то можна використовувати мікросервіси для цієї мети.
6. База даних:
    - Система може включати в себе розподілену базу даних або сховище даних для зберігання інформації про користувачів, курси, взаємодію тощо.
7. Розподілені сховища для медіа-файлів:

- Якщо на курсах використовуються відео, аудіо або інші медіа-файли, то їх можна зберігати в розподілених сховищах та надавати доступ до них через CDN (Content Delivery Network).

8. Інші компоненти:
    - Залежно від конкретних потреб системи, можуть бути інші компоненти, такі як системи оплати, інтеграція з LMS (Learning Management System), системи для обробки іншої інформації, яка пов'язана з навчанням.

Ця розподілена архітектура дозволяє розширювати та масштабувати систему, а також забезпечувати надійну та ефективну роботу з великою кількістю користувачів та курсів. Кожен компонент може бути розгорнутий та масштабований окремо залежно від навантаження.

3.4 Реалізація на основі архітектури

Розподілена архітектура в контексті вашого проекту може включати розподілену обробку даних та задач для скрапінгу та аналітичної обробки інформації про навчальні онлайн-курси. Це може бути особливо корисно, якщо ви маєте велику кількість даних для обробки та потребуєте високої масштабованості. Ось загальний план, як ви можете це реалізувати:

1. Мікросервіси: розподілення системи на окремі мікросервіси, які будуть відповідати за різні функціональні аспекти. Наприклад, мати можливість створити окремий мікросервіс для скрапінгу, окремий мікросервіс для збереження даних в базі даних і окремий мікросервіс для аналітичної обробки. Див. рис. 9.

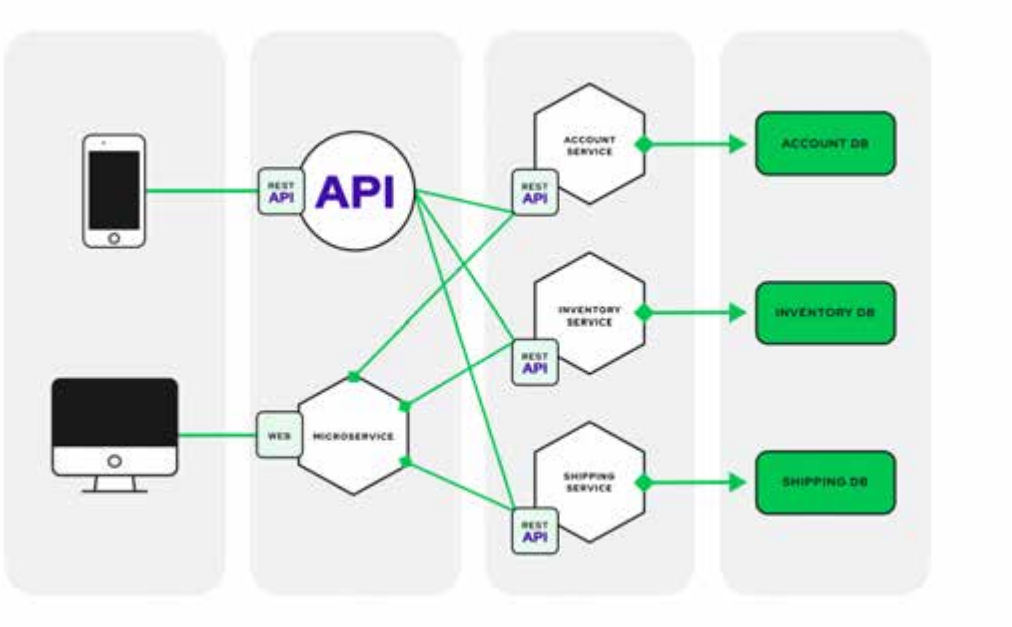

Рис.10 схема роботи мікросервісу.

2. Комунікація між мікросервісами: використання механізмів мікрослужб для комунікації між мікросервісами. REST API, протоколи, такі як gRPC або MQTT, або інші механізми для обміну даними між компонентами.
3. Node.js для реалізації мікросервісів: Використовуйте Node.js для реалізації кожного мікросервісу. Node.js добре підходить для створення високопродуктивних мікросервісів завдяки своєму неблокуючому асинхронному підходу.
4. Node-cron для планування задач: Використовуйте бібліотеку node-cron для планування скрапінгу та інших асинхронних завдань у ваших мікросервісах. Кожен мікросервіс може мати свій план виконання задач.
5. Завдання для мікросервісів: Кожен мікросервіс може мати завдання, які виконуються у відповідності до розкладу node-cron. Наприклад, скрапінг нових даних за допомогою puppeteer з онлайн-курсів може бути завданням одного мікросервісу, а аналітична обробка може бути завданням іншого мікросервісу.
6. Моніторинг та керування: Важливо встановити механізми моніторингу та керування для кожного мікросервісу, щоб виявляти помилки,

відновлювати випущені завдання та забезпечувати стабільну роботу системи.

7. Запуск та оркестрація: Використовуйте інструменти для оркестрації мікросервісів, такі як Docker та Kubernetes, для керування і масштабування мікросервісів.

Ця архітектура дозволить вам створити масштабовану та ефективну систему для збору та аналітичної обробки інформації про навчальні онлайн-курси, де кожен компонент відповідає за свою функціональність та взаємодіє з іншими через мережу.

Судячи з аналізу деяких інструментів, можна створити автоматизацію.

Для автоматизації запуску скрапінг-скрипта на віддаленому сервері за допомогою Jenkins та Docker, можна виконати наступні кроки:

1. Підготувати скрапінг-скрипт і створити Docker-контейнер:
    - Розробити ваш скрапінг-скрипт та пакети, які потрібні для його виконання.
    - Створити Docker-контейнер для скрипта, який включає всі необхідні залежності.
2. Завантажити Docker-контейнер на віддалений сервер:
    - Використати Docker Hub або інший репозиторій контейнерів для зберігання нашогоDocker-контейнера.
    - Встановити Docker на віддаленому сервері та завантажити контейнер з репозиторію.
3. Налаштувати Jenkins на віддаленому сервері:
    - Встановити та налаштувати Jenkins на віддаленому сервері.
    - Переконатись, що Jenkins може взаємодіяти з Docker і використати його для запуску контейнера.
4. Налаштувати завдання в Jenkins:

- Створітити нове завдання (Job) в Jenkins для автоматичного запуску скрапінг-скрипта.
- У налаштуваннях завдання, вказати команду для запуску Docker-контейнера з скриптом. Наприклад див. рис. 10.

```
~ ❯ docker run -d --name my-scraper-container your-scraper-image
```

Рис. 11 команда запуску контейнеру

- Налаштувати запуск завдання о 9 годині ранку щодня за допомогою node-cron, як описано в попередньому відповіді.

5. Налаштувати моніторинг і логування:
    - Додати моніторинг та журналювання виконання завдання, щоб ви могли відстежувати статус та виявляти помилки.

Завдяки цій системі, ваш скрапінг-скрипт буде автоматично запускатися на віддаленому сервері о 9 годині ранку щодня за розкладом, який налаштовано у Jenkins. Docker дозволитьізолювати середовище виконання скрипта, що спростить управління залежностями та допоможе забезпечити надійну роботу.

3.5 Тестування

Тестування розподіленого продукту, який включає мікросервіси, базу даних, планування завдань та інші компоненти, може бути складним завданням, але важливим для забезпечення надійності та функціональності продукту. Ось деякі кроки та підходи до тестування такого продукту:

1. Одиницеве тестування (Unit Testing): Почніть з написання тестів для окремих компонентів, таких як мікросервіси та бізнес-логіка. Використовуйте бібліотеки для тестування, такі як Mocha, Chai або Jest для створення і запуску тестів. Одиницеві тести допоможуть вам перевірити коректність роботи окремих частин системи.
2. Інтеграційне тестування (Integration Testing): Використовуйте інтеграційне тестування, щоб перевірити, як компоненти взаємодіють один з одним.

Тести можуть охоплювати сценарії взаємодії між мікросервісами та їх завданнями.

3. Тестування бази даних: Створіть тести для бази даних, щоб переконатися, що дані правильно зберігаються та витягуються з бази даних. Використовуйте фіктивну базу даних або засіб для мокування бази даних під час тестування.
4. Тестування планування завдань: Переконайтеся, що планування завдань працює правильно, відтворюючи різні сценарії планування. Використовуйте інструменти для тестування планування завдань або встановіть засіб для мокування планування.
5. Автоматизоване тестування: Автоматизуйте тестування, щоб можливо найшвидше виявити помилки та забезпечити надійність системи під час регулярних змін та оновлень.
6. Тестові середовища: Створіть тестове середовище, яке відтворює реальне виробниче середовище. Використовуйте контейнеризацію (наприклад, Docker) для створення ізольованих тестових середовищ та роботи з тестовими базами даних та мікросервісами.
7. Стрес-тестування та навантаження: Проводьте стрес-тестування та тестування на навантаження, щоб визначити межі масштабованості та виявити проблеми, пов'язані з продуктивністю та ресурсами.
8. Моніторинг та логування: Забезпечте моніторинг та логування в системі, щоб ви могли відстежувати стан продукту та аналізувати помилки в реальному часі.
9. Тестування збійостійкості: Проводьте тестування збійостійкості, шляхом відключення окремих мікросервісів або баз даних, щоб переконатися, як система реагує на відмови.
10. Тестування безпеки: Проводьте тестування безпеки для переконання в захищеності системи від потенційних атак та загроз.

11. Тестування сценаріїв відновлення: Проводьте тести сценаріїв відновлення, щоб переконатися, що система може відновити роботу після виникнення помилок чи збоїв.

Загалом, тестування розподіленого продукту вимагає комплексного підходу та включення різних видів тестів для різних компонентів системи. Тестування повинно бути невід'ємною частиною процесу розробки та оновлення системи, щоб забезпечити її стабільність та надійність.

АНАЛІЗ ТА РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Апаратні вимоги для програмного забезпечення автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси

Апаратні вимоги для програмного забезпечення автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси варіюються в залежності від обсягу та складності обробки даних, а також від рівня автоматизації та обсягу даних, які вам необхідно обробляти. Нижче наведені загальні вимоги та рекомендації для апаратних засобів:

1. Система: Для програмного забезпечення, яке займається автоматизованим збором та аналітичною обробкою інформації про навчальні онлайн-курси, рекомендована операційна система - Windows, macOS або Linux. Вибір залежить від ваших вподобань та потреб.
2. Процесор: Швидкодія процесора має значення для ефективності обробки даних. Рекомендовані процесори Intel Core i5 або еквівалентні.
3. Оперативна пам'ять (RAM): Обсяг оперативної пам'яті також є важливим чинником. Для невеликих обсягів даних може бути достатньо 4-8 ГБ RAM, але для обробки великих обсягів даних рекомендується 16 ГБ RAM і більше.
4. Місце на диску: Обсяг вільного місця на жорсткому диску залежить від обсягу даних, які ви збираєте і обробляєте. Рекомендовано мати як мінімум 100 ГБ вільного місця.
5. Графічна карта: Для аналітики і візуалізації даних, якщо це необхідно, рекомендується використовувати графічну карту з підтримкою GPU.
6. Інтернет-з'єднання: Швидкість та стабільність інтернет-з'єднання важливі для збору даних з онлайн-курсів та їх аналізу. Використовуйте високошвидкісне з'єднання.

7. Відображення: Розширення та якість екрану мають значення, особливо якщо ви працюєте з візуалізацією даних. Рекомендовано монітори з Full HD або вищим розширенням.
8. Периферійні пристрої: Наявність клавіатури та миші є обов'язковою. Ви також можливо захочете використовувати графічний планшет або інші периферійні пристрої в залежності від завдань.

Це загальні вимоги для апаратного забезпечення, але кожен конкретний проект може вимагати своїх унікальних параметрів. Наявність резервного живлення, відмовостійкість, захист від перегріву тощо також може бути важливою в залежності від умов використання програмного забезпечення.

Програмні вимоги для програмного забезпечення автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси з використанням JavaScript

При розробці програмного забезпечення для автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси, використання мови програмування JavaScript є ключовим аспектом, оскільки JavaScript є однією з основних мов для розробки веб-додатків та взаємодії з веб-сторінками. Нижче наведені загальні програмні вимоги, зокрема з використанням JavaScript:

1. Використання середовища виконання JavaScript: Для створення програмного забезпечення з використанням JavaScript, вам потрібне середовище виконання JavaScript. Один із популярних варіантів - це середовище виконання в браузері, а також розробка на серверному боці, наприклад, використовуючи Node.js.
2. Бібліотеки та фреймворки: Для зручності розробки можна використовувати різні бібліотеки та фреймворки на основі JavaScript. Наприклад, для веб-скрапінгу та роботи з HTTP-запитами можна використовувати бібліотеки, такі як Axios, Fetch API або більш потужні фреймворки, такі як Express.js для серверної розробки.

3. Засоби роботи з DOM: Якщо ваше програмне забезпечення взаємодіє з веб-сторінками, вам потрібні знання роботи з Document Object Model (DOM) в браузері. Для цього можна використовувати вбудовані засоби JavaScript.
4. Робота з базами даних: Для зберігання та обробки даних можна використовувати різні бази даних, які підтримують JavaScript. Наприклад, ви можете використовувати NoSQL бази даних, такі як MongoDB або SQLite для локального зберігання даних.
5. Аналітичні бібліотеки: Для обробки та аналізу даних можна використовувати аналітичні бібліотеки, такі як D3.js для візуалізації даних або бібліотеки для математичних обчислень, наприклад, Math.js.
6. Автоматизація завдань: Для автоматизації процесів та запуску завдань визначте, якій буде використовувати таскранери, такі як Gulp або Grunt, або використовуйте вбудовані засоби Node.js.
7. Засоби безпеки: Забезпечте безпеку програмного забезпечення, включаючи валідацію даних, захист від атак, таких як ін'єкція SQL або міжсайтовий сценарій (XSS), і захист даних.
8. Моніторинг та відладка: Забезпечте можливість відстеження і відладки програмного забезпечення, включаючи ведення журналу подій та використання інструментів, таких як вбудований консоль браузера.
9. Інтеграція з іншими системами: Передбачте можливість інтеграції з іншими системами, включаючи внутрішні та зовнішні API, для обміну даними та інформацією.
10. Документація: Надайте користувачам та розробникам докладну документацію щодо використання та налаштування програмного забезпечення.

Це загальні програмні вимоги для програмного забезпечення автоматизованого збору та аналітичної обробки інформації про навчальні онлайн-курси, з фокусом на використанні JavaScript. Конкретні вимоги можуть варіюватися в залежності від обсягу та складності вашого проекту.

Була створена авторизація для користувачів. Вони можуть реєструватись та входити на свій особистий кабінет за допомогою електронної пошти та паролю. Якщо користувач вводить на дійсну пошту, то форма автоматично повідомлятиме, що пошта не дійсна. Аналогічна ситуація і з паролем. Паролі автоматично зберігаються у нереляційній базі даних і пошта користувачів також. Пароль одразу хешуються. Це зроблено для безпеки користувачів, якщо їхні ключі будуть викрадені.

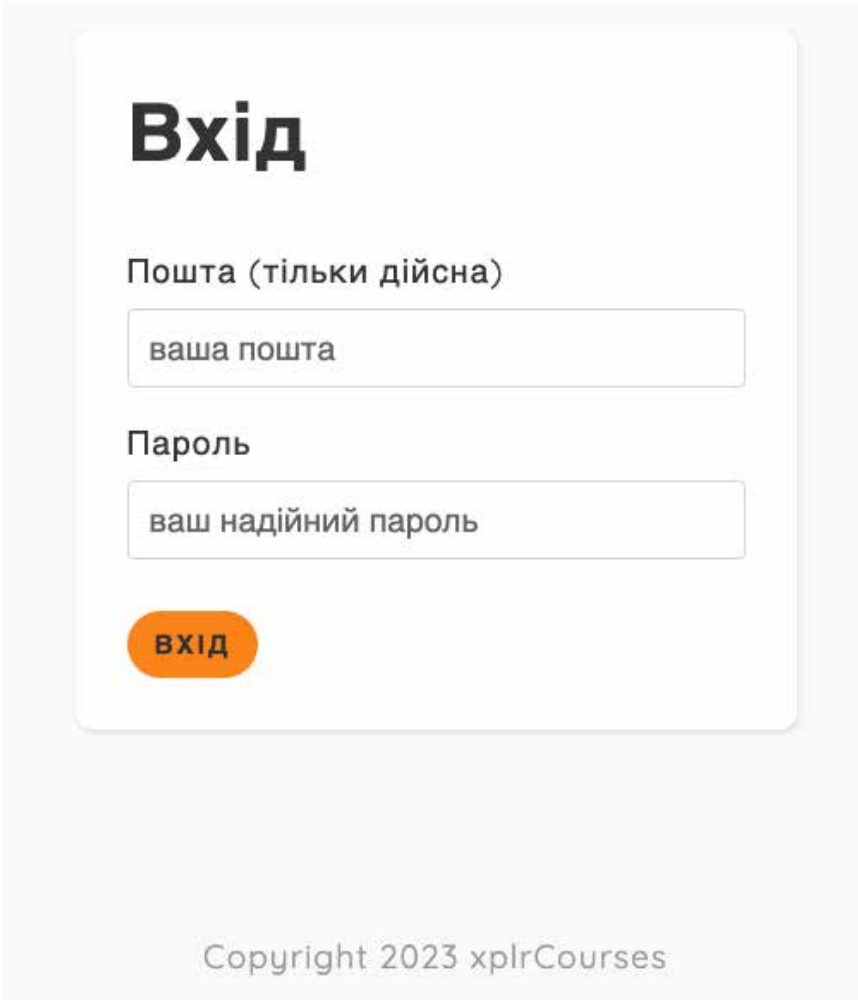

Рис. 1 форма входу

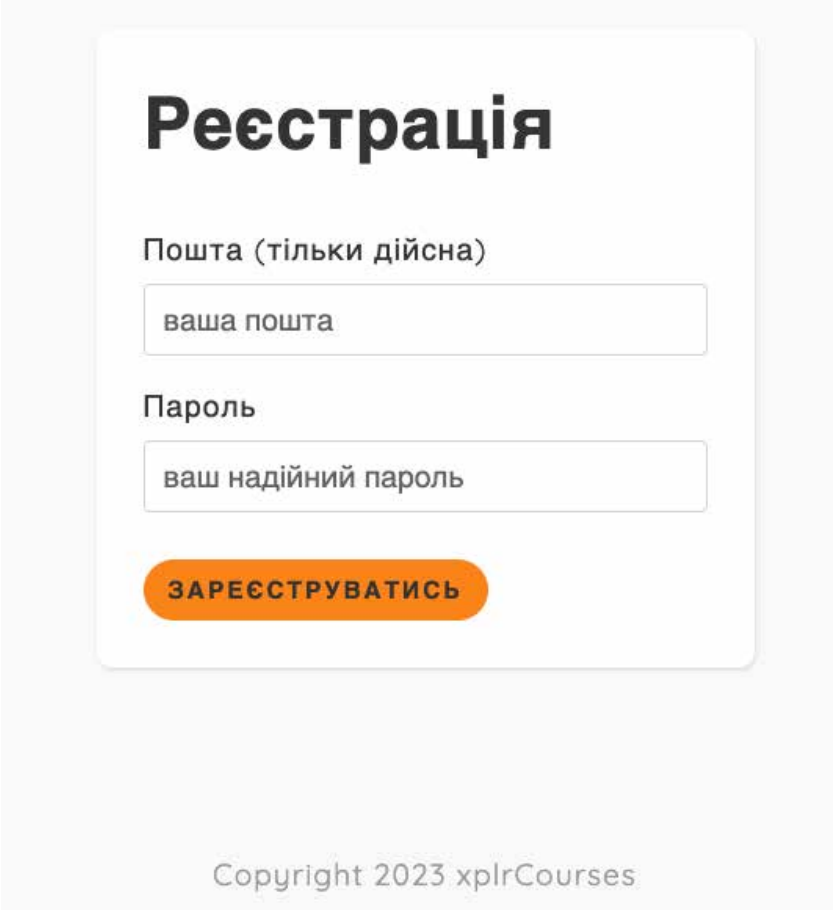

Рис. 2 форма реєстрації


```
_id: ObjectId('63e63d640d0ca88792490df8')
email: "ipz18dqwerty@it.edu.nubip.ua"
password: "$2b$10$.YnWSoGKBCPokzNGYjxK9.e0d7LgBC0qehtbLH1LrfxFoQDRPzD6G"
__v: 0

_id: ObjectId('63e643c269d9050b87cc7824')
email: "qwerty@gmail.com"
password: "$2b$10$lPJa76ImrdP6b8i4YlZAZ.MxfAkpiGfCjAn6vwGXrMhYf7uT19oxy"
__v: 0

_id: ObjectId('63eb447244049b4c714d3351')
email: "qwerty1@gmail.com"
password: "$2b$10$i9EDeelsh1xvonWnR.q39.xi9gGdo4no46m3iQ4BQB1oOneyQJq0q"
__v: 0
```

Рис. 3 збережені дані в NoSQL

Користувач знайшовши курс може його переглянути. Сторінка курсу містить в собі:

- Опис курсу
- Ілюстрацію
- Скільки є наявних пропозицій на різних платформах
- Динаміка ціни
- Графік попиту

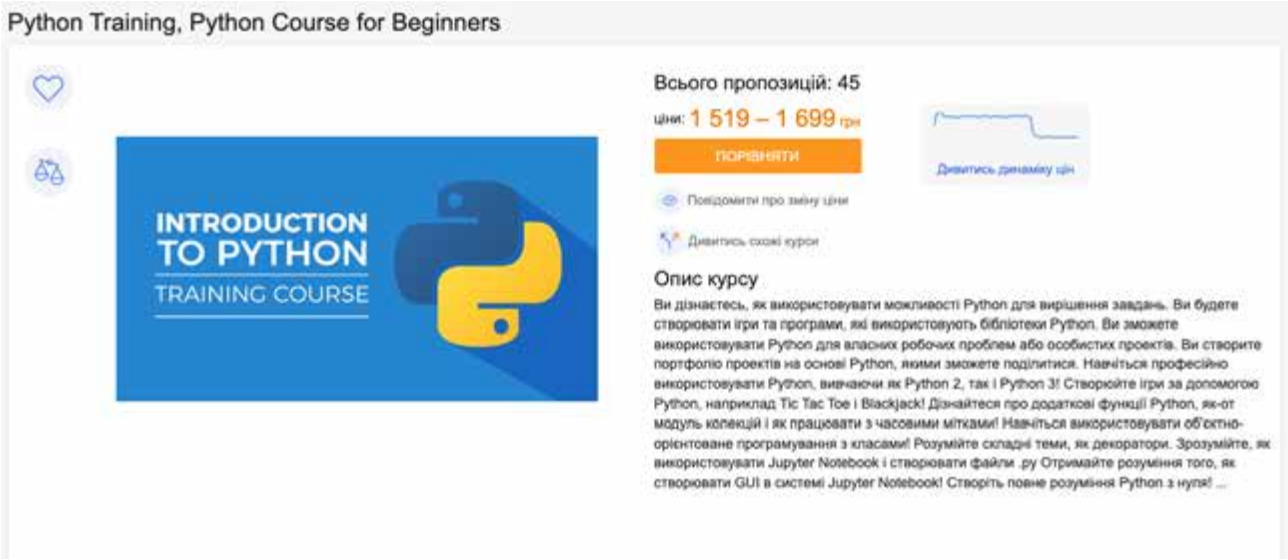

Рис. 4 сторінка курсу

Динаміка цін є дуже важливим, так як користувач може подивись, як змінювалась ціна протягом певних проміжків часу.

Рис.5 графік динаміки ціни

Також є проаналізований попит на певних проміжках часу.

Рис.6 графік попиту на курс

Саме головне аналіз на скільки актуально у наш час це програмне забезпечення. Були розроблені дешбори, які мають аналітику актуальності. Модератор може побачити статиску у яких країнах більше всього цікавляться курсами та звідки приходить цільова аудиторія

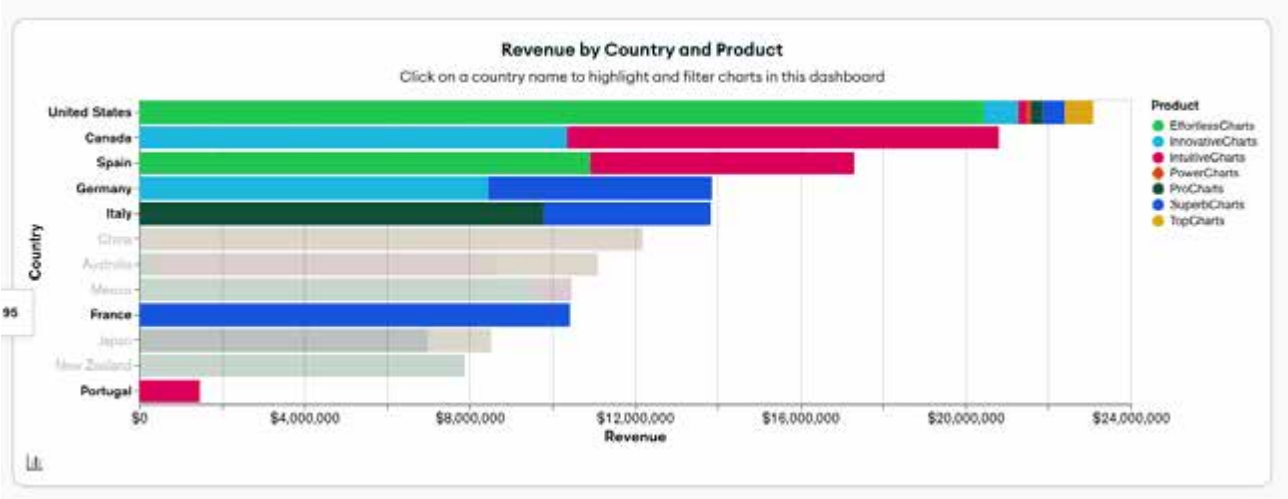

Рис. 7 гістограма кількості користувачів у різних країнах

Також він може переглянути графік скільки коштів приносить наш додаток певним компаніям. На графіку зображено скільки прибутку отримує компанія від нашого додатку протягом певних проміжків часу.

Рис. 6 графік прибутку компанії завдяки додатку

ВИСНОВКИ

Дослідження показало, що автоматизація збору інформації про курси може мати ряд переваг, зокрема:

- Покращення якості та зручності доступу до інформації про курси. Автоматизована система може збирати інформацію з різних джерел та платформ, що дозволяє користувачам отримати більш повну та актуальну інформацію про курси.
- Покращення ефективності вибору курсів. Автоматизована система може використовувати різні метрики та алгоритми для порівняння курсів на різних платформах, що допомагає користувачам знайти курси, які відповідають їхнім потребам.
- Зниження витрат часу та зусиль на пошук курсів. Автоматизована система може збирати інформацію про курси на постійній основі, що дозволяє користувачам отримувати актуальну інформацію без необхідності самостійного пошуку.

Дослідження також показало, що існують деякі виклики, пов'язані з автоматизацією збору інформації про курси. Зокрема, це може бути складно зібрати інформацію з усіх джерел та платформ, а також забезпечити актуальність інформації.

Незважаючи на ці виклики, автоматизація збору інформації про навчальні онлайн курси є перспективним напрямком досліджень. Вона може допомогти користувачам ефективніше знаходити курси, які відповідають їхнім потребам.

Додаткові ідеї для розвитку висновку

У висновку можна розглянути такі додаткові ідеї:

- Порівняння різних підходів до автоматизації збору інформації про курси.
- Аналіз потенційних переваг і недоліків автоматизації збору інформації про курси.
- Пропозиції щодо подальших досліджень у цьому напрямку.

Наприклад, можна розглянути порівняння таких підходів до автоматизації збору інформації про курси:

- Скріпінг веб-сайтів курсів.
- Використання API курсів.
- **Використання штучного інтелекту для обробки інформації про курси.

Також можна розглянути потенційні переваги і недоліки автоматизації збору інформації про курси, такі як:

- Переваги:
    - Покращення якості та зручності доступу до інформації про курси.
    - Покращення ефективності вибору курсів.
    - Зниження витрат часу та зусиль на пошук курсів.
- Недоліки:
    - Складність зібрати інформацію з усіх джерел та платформ.
    - Необхідність забезпечення актуальності інформації.

Нарешті, можна запропонувати напрямки для подальших досліджень у цьому напрямку, такі як:

- Розробка більш ефективних алгоритмів для збору інформації про курси.
- Розробка інструментів для аналізу та відображення інформації про курси.
- Розробка систем для автоматизації вибору курсів.

Ці ідеї можна використовувати як основу для подальшого розвитку висновку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке онлайн-курс? [Електронний ресурс] URL: https://samoosvita.in.ua/scho-take-onlayn-kurs/,
2. JavaScript [Електронний ресурс] URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript,
3. Npm [Електронний ресурс] URL: https://www.npmjs.com/,
4. Nodejs[Електронний ресурс] URL: https://nodejs.org/en,
5. UML [Електронний ресурс] URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
6. Puppeteer [Електронний ресурс] URL: https://pptr.dev/,
7. Docker [Електронний ресурс] URL: https://docs.docker.com/?_gl=1*19qtrzv*_ga*NjA3MzQ0NDI3LjE2OTkwNTM4ODQ.*_ga_XJWPQMJYHQ*MTY5OTA1Mzg4My4xLjEuMTY5OTA1Mzg4NC41OS4wLjA.,
8. Jenkins [Електронний ресурс] URL: https://www.jenkins.io/doc/book/,
9. Kubernetes [Електронний ресурс] URL: https://kubernetes.io/docs/home/,
10. SQL-injection [Електронний ресурс] URL: https://owasp.org/www-community/attacks/SQL_Injection,
11. SQL [Електронний ресурс] URL: https://www.w3schools.com/sql/sql_intro.asp,
12. Різниця між ODM. та ORM [Електронний ресурс] URL: https://www.seacomp.com/resources/oem-vs-odm-manufacturing,
13. Mongoose [Електронний ресурс] URL: https://mongoosejs.com/docs/index.html,
14. MySQL[Електронний ресурс] URL: https://dev.mysql.com/doc/,
15. MongoDB [Електронний ресурс] URL: https://www.mongodb.com/,
16. Мікросервісна архітектура[Електронний ресурс] URL: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d,

17. Хешування паролів [Електронний ресурс] URL: https://doc.nette.org/uk/security/passwords,
18. Що таке криптографія? [Електронний ресурс] URL: https://tsecrypto.com/article/shho-take-kryptografiya/
19. Різниця між скрапингом та парсингом [Електронний ресурс] URL: https://stackoverflow.com/questions/590888/%D0%92-%D1%87%D0%B5%D0%BC-%D1%80%D0%B0%D0%B7%D0%BD%D0%B8%D1%86%D0%B0-%D0%BC%D0%B5%D0%B6%D0%B4%D1%83-%D0%BF%D0%B0%D1%80%D1%81%D0%B8%D0%BD%D0%B3%D0%BE%D0%BCparsing-%D0%B8-%D1%81%D0%BA%D1%80%D0%B5%D0%B9%D0%BF%D0%B8%D0%BD%D0%B3%D0%BE%D0%BCweb-scraping,
20. Що таке API? [Електронний ресурс] URL: https://dev.ua/news/chto-takoe-api-prostym-yazykom,
21. HTTP [Електронний ресурс] URL: https://developer.mozilla.org/en-US/docs/Web/HTTP.