МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є
"_____"_____2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Розробка *web*-додатку для пошуку та підбору втомобілів

Виконавець: Демчук В.І.

Керівник: старший викладач Сябрук І.М.

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри
Литвиненко О. Є.
«_____»_____2021р.

ЗАВДАННЯ
на виконання дипломної роботи (проєкту)

Демчука Владислава Ігоровича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи (проєкту) Розробка *web*-додатку для пошуку та підбору автомобілів

затверджена наказом ректора від «_04_» лютого_2021р. № 135/ст

2. Термін виконання роботи (проєкту): з _13.04.2021____ по __31.05.2021___

3. Вихідні дані до роботи (проєкту): *web*-додаток для пошуку та підбору автомобілів

4. Зміст пояснювальної записки:

1. аналіз веб-додатків та їх реалізація
2. веб-скрапінг
3. розробка веб-додатку

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1. Схема алгоритму пошуку авто
2. Схема алгоритму скрапінгу інформації
3. Діаграми класів
4. Екранна форма інтерфейсу користувача

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Проведення огляду та аналізу існуючих скраперів	17.05.2021-20.05.2021	
2	Написання першого розділу	21.05.2021-23.05.2021	
3	Проаналізувати літературні джерела за темою дипломної роботи та вивчити основні етапи створення веб-додатків	24.05.2021-27.05.2021	
4	Написання другого розділу	28.05.2021-01.06.2021	
5	Реалізувати веб-додаток для пошуку та підбору автомобілів	02.06.2021-06.06.2021	
6	Оформити пояснювальну записку	07.06.2021-08.06.2021	
7	Підготовка графічних та ілюстративних матеріалів	09.06.2021-17.06.2021	

7. Дата видачі завдання: “ 17 ” травня 2021р.

Керівник дипломної роботи (проєкту) _____ Сябрук І.М.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Демчук В.І.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту «Веб-додаток для пошуку та підбору автомобілів»: 53 сторінки, 14 рисунків, 3 додатки, 12 літературних джерел.

Ключові слова: ВЕБ-ДОДАТОК, *NODE.JS*, ПОШУК АВТО, СКРАПЕР, ПІДБІР АВТО.

Об’єкт дослідження – веб-додаток для пошуку та підбору автомобілів.

Предмет дослідження – створення веб-додатку для пошуку та підбору автомобілів.

Мета дипломної роботи – розробити веб-додаток для пошуку та підбору автомобілів. Готовий проєкт має виконувати наступні функції: пошук та збереження інформації про автомобілі з інших сайтів, можливість підбору авто .

Важливість розробки даного продукту полягає у створенні вільного веб-додатку для підбору авто, актуальним є створення безкоштовного веб-додатку, який водночас буде простий і зрозумілий у використанні.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

HTML – Hyper Text Markup Language — мова розмітки гіпертексту.

CSS – Cascading Style Sheets – спеціальна мова, що використовується для опису зовнішнього вигляду сторінок, написаних мовами розмітки даних

БД – база даних

Node.js – платформа з відкритим кодом для виконання високопродуктивних мережевих веб-додатків

REST – Representational State Transfer – архітектурний стиль взаємодії компонентів розподіленого додатку в мережі

JS – JavaScript – об'єктно-орієнтована прототипна мова програмування

Геотаргетинг – метод видачі вмісту, що відповідає географічному положенню відвідувача.

ВСТУП

На сьогоднішній день, темпи розширення просторів мережі Інтернет набули колосальних темпів, постійно створюються нові статичні та динамічні сайти, соціальні мережі, месенджери, інтернет-магазини – усе це, одним словом веб-додатки. Веб-додатком вважається клієнт-серверна програма, що виконує певну функцію з елементами інтерактиву, використовуючи у якості клієнта виступає веб-браузер. Додатки можуть бути прості, наприклад як звичайна дошка оголошень або контактна інформація на певному сайті, або ж більш складними, такими як текстові процесори або багатокористувацькі мобільні ігрові додатки.

Основні етапи реалізації веб-додатку:

1. Визначення цілей та задач проєкту – на даному етапі формулюють бізнес-цілі створюваного проєкту, визначаються вимоги, які він повинен задовольняти, розробляється загальна концепція;
2. Розробка структури – містить все, що відноситься до змісту та інформаційної тактики, яка визначає, організацію подачі інформації, щоб прийдешні користувачі сайту могли швидко та з легкістю її знайти. Терміновим завданням на даній фазі є створення мапи сайту, що відображає взаємозалежність типових сторінок та їх найбільш важливі функціональні можливості;
3. Розробка дизайн-макету. Дизайн-макет – це графічне відображення фундаментальних елементів сайту. Дизайн-макет цілком втілює візуальне зображення сайту;
4. *HTML*-розмітка. *HTML*-верстка макету є виконується після створення дизайну сайту. Верстка – перетворення створених дизайнером макетів сторінок в *HTML*-код, що буде відображатися у браузері в точній копії з макетом дизайнера;

5. Програмування та контроль якості - це будування функціональних інструментів для заповнення і опрацювання даних. Програмування з’ясовує наскільки стійким і захищеним буде робота сайту;
6. Підключення готової або створення своєї системи керування вмістом – як правило сучасні веб-проєкти містять в собі підсистему адміністрування контентом – *CMS* (*content management system*), або ж будуються навколо неї. Призначення *CMS* полягає в тому, щоб давати можливість публікувати контент на сторінці людині, що не має спеціальних знаннь в області створення *HTML*-документів, а також істотно скоротити час, використовуваний на ввід даних;
7. Підключення баз даних та наповнення їх контентом;
8. Запуск та супровід. Головна мета супроводу – підтримка стабільної роботи веб-ресурсу. Неодмінною умовою професійного супроводу веб-сайту є захист інформації, що передбачає у собі антивірусний захист і захист БД на сервері.

Основною перевагою веб-додатків є їх мультиплатформеність, що звільняє розробника від необхідності створення декількох додатків для різних операційних систем (*Mac OS, Windows, Unix*-подібні операційні системи) чи пристроїв.

Функціонал веб-додатку, звичайно, залежить від області застосування, але існують загальні види додатків, одними з таких є:

ERP *(Enterprise Resource Planning* – планування ресурсів підприємства) такі додатки створюються для великих підприємств. Головними перевагами є уніфікація звітності, оптимізація внутрішніх процесів, покращення співпраці між підрозділами, контроль вирішення завдань, синхронізація процесів.

Електронна комерція – *E-commerce* це веб-додаток що надає змогу виробникам і/або постачальникам рекомендувати свої товари можливій аудиторії відразу в мережі. Такі додатки направлені на організацію процесів, що спрямовані на обробку, керування та спостереження за замовленнями.

Використання таких веб-додатків виробниками та постачальниками відразу вирішує такі завдання як, швидке виведення товару на ринок, пониження виплат, потрібних на підписання/закриття угоди, значно скорочує «шлях», який проходить продукт від постачальника до клієнта, що інколи суттєво знижує кінцеву ціну для клієнта.

CRM *(Customer relationship management* – Управління відносинами з клієнтами) – такі веб-додаток призначені для полегшення зв'язків з клієнтами. Вони дозволяють швидко та якісно вирішувати різноманітні задачі, до яких належать контроль і планування.

Також вирішують такі задачі як зберігання бази клієнтів та високошвидкісного доступ до неї, точна та якісна аналітика за усіма замовленнями, оптимізація роботи працівників організації, велике зниження застосування паперових документів.

Зупинимося на такому видові додатків як електронна комерція (*E-commerce*) – на сьогоднішній день існує велика кількість сайтів для покупки/продажу автомобілів, для ефективності пошуку бажаного авто користувачу необхідно спочатку знайти певну кількість сайтів, окремо у кожному зробити пошук бажаного авто, виписати ціни, зайти на наступний сайти і повторити дії. Після чого серед великої кількості сайтів в ручну така робота буде доволі тяжка. Для цього можна створити веб додаток який може зібрати дані з різних сайтів, відобразити їх на одній сторінці та за необхідності зробити сортування.

Метою роботи є створення веб-додатку для пошуку та підбору автомобілів.

Об'єктом дослідження являється аналіз існуючих веб-додатків для підбору автомобілів, їх порівняння, визначення можливостей та переваг.

Предмет дослідження – веб-додаток для пошуку та підбору автомобілів, створений на програмній платформі *Node.js*.

Результатом роботи є програмне забезпечення, тобто веб-додаток для підбору автомобіля з різних сайтів продажі автомобілів.

РОЗДІЛ 1
АНАЛІЗ ВЕБ-ДОДАТКІВ ТА ЇХ РЕАЛІЗАЦІЯ

Веб-додаток – це будь який сайт, що містить в собі елементи інтерактивності. Це означає, що користувач має можливість взаємодіяти з даними та функціями: натискання кнопок, заповнення форм, здійснення покупки і так далі. Фактично будь-який інтернет-ресурс входить до числа веб-додатків, це можуть бути пошукові системи, стрімінгові платформи, соціальні мережі, інтернет магазини, біржі і т. д.

1.1. Чим відрізняється веб-додаток від сайту

Якщо брати звичайний сайт, це в першу чергу щось інформаційне та статичне: візитки компаній, сайт рецептів, портал міста чи вікипедії. У будь якому випадку, це набір заранння підготовлених *HTML*-файлів, що розміщені десь на віддаленому сервері та віддаються браузерові (клієнтові) по запиту.

Сайти можуть містити статистику але вона не генерується в реальному часі а всього лиш, як і *HTML* файли, готова лежить на сервері. Частіше всього використовуються картинки, *CSS*-файли, *JS*-скрипти, можуть використовуватися і інші файли наступних типів: як *mp3, mov, csv, pdf.*

Блоги та візитки з формою для контакту, лендінги з великою кількістю ефектів теж можна віднести до звичайних сайтів. Хоча на відміну від зовсім статичних сайтів, вони вже містять в собі певну бізнес логіку.

А веб-додаток – технічно більш складний. Тут *HTML*-сторінки генеруються в реальному часі в залежності від запитів користувача.

1.2. Які бувають веб-додатки

Веб-додатки можна розділити на декілька типів, в залежності від його складових:

- *Backend* (бекенд або серверна сторона додатку) працює на віддаленому комп'ютері (сервері), що може знаходитися де завгодно. Може бути написана на різних мовах програмування: *PHP*, *Python*, *Ruby*, *C#* та інші. Якщо при створенні додатку використовувати лише серверну частину, то у результаті будь яких переходів між розділами, відправкою форм, оновленні даних, сервером буде генеруватися новий *HTML*-файл та сторінка браузера буде перезавантажуватися;
- *Frontend* (фронтенд або ж клієнтська частина додатку) виконується в браузері користувача. Ця частина пишеться на мові програмування *JavaScript*. Додаток може складатися з клієнтської частини, у випадках, коли немає необхідності зберігати дані користувача, більше однієї сесії. Це можуть бути фото-редактори або ж просто ігри;
- *Single page application* (*SPA* або ж односторінковий додаток). В даному випадку використовується і бекенд і фронтенд. За допомогою їх поєднання можливо створити додаток, що буде працювати зовсім без перезавантажень сторінки у браузері. Або ж в спрощеному варіанті, коли перехід між розділами викликає перезавантаження сторінки, а будь-які дії в розділі обходять без перезавантаження.

1.3. Як клієнт та сервер спілкуються між собою

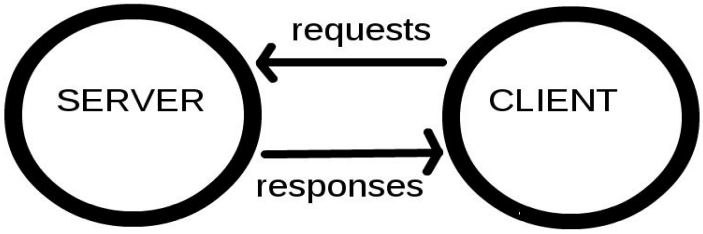

Рис. 1.1. Схема спілкування клієнта з сервером

Спілкування клієнта з сервером відбувається за протоколом *HTTP* (*HyperText Transfer Protocol*). Основа даного протоколу – запит від клієнту до сервера та відповідь сервера клієнту.

Для запитів, в основному, використовують методи *GET* та *POST* запити. *GET* запит використовується у випадках коли ми бажаємо отримати певні дані. *POST* запит використовується для того щоб змінити певні дані. Також в запитах вказується *Host* (домен сайту), тіло запиту (якщо це *PSOT*-запит) та багато додаткової технічної інформації.

Сучасні веб-додатки використовують протокол *HTTPS*, розширену версію *HTTP* з підтримкою шифрування *SSL*(*Secure Sockets Layer*)/*TLS*(*Transport Layer Security*). Використання шифрованого каналу передачі даних, незалежно від важливості цих даних, стало хорошим тоном в інтернеті.

Ще один запит, що виконується перед *HTTP*. Це *DNS* (*domain name system*) запит. Він потрібен для отримання *ip*-адреси, до якої прив'язаний домен. Оскільки в інтернеті реальні адреси не зручні та складаються з чисел, їх замінили доменними іменами, наприклад: 142.250.186.142 – це *ip*-адреса доменного імені *google.com.*

1.4. Технічні особливості веб-додатків

Головною особливістю веб-додатку є його кросплатформність, це означає, що усі функції будуть працювати коректно на будь якій операційній системі (*Microsoft Windows*, *Mac OS X*, *GNU/Linux* чи іншій) чи пристрої (комп'ютер,

телефон, планшет і т. д.). Якщо з роботою проблем не виникає, то з коректним відображенням одного і того ж додатку на різних пристроях можуть виникнути проблеми. Наприклад текст в 14 пікселів, на екранах комп'ютерів вважається текстом середнього розміру, а на екрані телефону це великий шрифт. Тому при написанні веб-додатку часто використовують відносні шрифти що задаються у величинах *vw* – залежить від висоти екрану та *vh* – залежить від ширини екрану. Деякі браузери мають власні особливості пов'язані з відображенням того чи іншого елементу розмітки, тому кожен веб-додаток необхідно тестувати у різних браузерах для уникнення незручностей у користувачів.

1.5. Динамічні та статичні сторінки, їх обробка

Сторінки поділяються на статичні та динамічні в залежності від поведінки браузера. Документи також поділяються на статичні та динамічні в залежності способу їх створення.

По поведінці сторінки у браузері користувача, розділяють:

- Статичні *HTML*-сторінки – сторінки що завжди виглядають однаково, не залежно від впливу користувача. Наприклад, меню створене організованими посиланнями а не випадаючим меню;
- Динамічні *HTML*-сторінки. Це сторінки, що змінюються під впливом користувача. Наприклад по натисканні на певний текст може випливати інший блок тексту по верх попереднього.

Динаміка на веб-сторінках реалізовується за допомогою скриптів, що при відкритті сторінки чи використанні певного тригеру, виконуються браузером. Велика кількість компонентів мови *HTML* підтримують певні обробники подій. Наприклад, можна задати обробку події «натискання клавіші миші» на картинку чи певний блок. Якщо, користувач натисне на елемент до якого прив'язана подія, то запускається обробник даної події.

Документи також поділяються на статичні та динамічні.

Статичними називають документи, що лежать на сервері в готовому вигляді *HTML*-файлу.

Динамічними називають файли, що генеруються на сервері по запиту клієнта. Зазвичай у процесі динамічного створення файлу бере участь база даних та сервер.

Якщо коротко розглядати як працює процес динамічного генерування сторінки, то це виглядатиме наступним чином:

1) Браузер робить запит на сервер для отримання документу;
2) Сервер з'ясовує, що документ являється скриптом та запускає його виконання;
3) Скрипт генерує *HTML* сторінку;
4) Сервер надсилає готову згенеровану *HTML* сторінку браузеру.

Існує велика кількість мов програмування, які використовуються для написання скриптів, що генеруватимуть «динамічні» файли. Поширеними є: *Java, JavaScript, PHP*, *ASP*, *Python*, *SSI*, *Perl*. Кожна з цих мов програмування має свої особливості застосування. Написати скрипт можна на будь якій з них, але необхідно знати сильні та слабкі сторони кожної з поданих мов, для того щоб обрати необхідну.

Як правило, якщо посилання сторінки закінчується на *.html* або ж *.php* то це статичні сторінки. Якщо ж вказано інше розширення, наприклад на мові PHP закінчення файлів створених скриптами .php, на мові *Perl-.pl, ASP- .aspx*. На сьогоднішній день на серверах часто задають відносні посилання, тобто користувач не буде бачити у посиланні назву надісланого йому файлу.

1.6. Компоненти веб-додатку

Частіше за все веб-додаток складається з мінімум трьох основних компонентів:

- Клієнтська частина додатку – це графічний інтерфейс. Це те що бачить користувач на сторінці. Взаємодія з додатком відбувається саме через графічний інтерфейс, що відображає браузер, за допомогою посилань та кнопок;
- Серверна частина додатку – це програма чи скрипт на сервері, що оброблює запити клієнта (браузера). При кожному переході користувача по посиланні браузер відправляє запит серверу;
- База даних – програмне забезпечення на сервері, що займається збереженням даних та їх видачею в необхідний момент. У випадку блогу чи форуму дані – це пости, коментарі, новини, якщо це інтернет магазин – дані про товар, фото, коментарі. Серверна частина додатку звертається до бази даних, витягує необхідну інформацію для формування файлу на основі запиту сформованого користувачем.

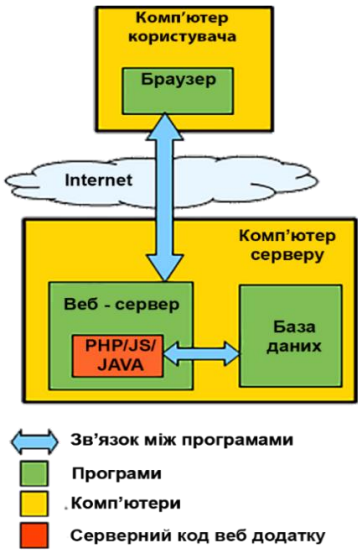

Рис. 1.2. Графічне зображення схеми взаємодії основних компонентів веб-додатку

1.7. Структура веб-додатку

Структурні схеми сторінок (*wireframes*) – головний результат робіт з проєктування. Вони в деталях показують, яка інформація і елементи керування мають відображатися на кожній сторінці системи. А також розставляють акценти – які з елементів сторінки більш, а які – менш важливі. *Wireframes* також описують поведінку динамічних і *AJAX*-елементів сторінок – як вони повинні реагувати на дії користувача. Варто пам'ятати, що схеми сторінок – це не остаточний дизайн системи.

Призначення структурних схем полягає:

- Постановка завдання дизайнеру. При створенні макетів сторінок дизайнер має враховувати усі зазначені в *wireframes* елементи і брати до уваги розставлені пріоритети;
- Постановка завдання розробникам. Розробникам необхідно дотримуватися *wireframes* при створенні прообразу системи;
- Демонстрація інвесторам і потенційним користувачам.

Веб-додаток, включає декілька тематичних розділів, об’єднаних між собою гіперпосиланнями. Посилання на всі частини сайту з не великим описом їх вмісту приводяться на першій (головній) сторінці, зазвичай такій сторінці задають назву *index.html*.

Логічна структура сайту – це сукупність всіх сторінок на сайті, розташованих з урахуванням ієрархії.

Структуру сайту треба побудувати таким чином, щоб спростити користувачам перехід від загальних тем до конкретної інформації, за якою вони прийшли. Від цього залежить зручність користувача, час, який він проведе на сайті, цільові дії, які він зробить. Заплутана система навігації і надмірне структурування контенту ні до чого доброго не приведе.

1.8. Структура каталогів

Найбільш поширені речі, присутні в будь-якому проєкті сайту: індексний файл *HTML* і папки, що містять зображення, файли стилів і файли скриптів. Виглядає це наступним чином:

- *index.html*: Цей файл зазвичай містить контент домашньої сторінки, тобто текст і зображення, які люди бачать, коли вони вперше потрапляють на ваш сайт;
- Папка *images*: Ця папка буде містити всі зображення, які використовуються на сайті. Папку з ім'ям *images* створюють всередині головної папки проєкту;
- Папка *styles*: Ця папка буде містити *CSS* код, який використовується для стилізації вашого контенту (наприклад, настройка тексту і кольору фону). Папку з ім'ям *styles* створюють всередині головної папки проєкту;
- Папка *scripts*: Ця папка буде містити весь *JavaScript* код, який використовується для додавання інтерактивних функцій на сайті (наприклад, кнопки що завантажують дані);
- Папку з ім'ям *scripts* створюють всередині головної папки проєкту.

1.9. Класифікація веб-додатків

Веб-додатки можна розділити на види в залежності від технологій створення, а також за призначенням.

Розглянемо докладніше більш популярні та необхідні:

AJAX – перевага даного підходу полягає в тому, що веб-сторінки не оновлюються з усіма даними з початку, а лише підвантажують потрібне з сервера, це підвищує рівень інтерактивності та продуктивність додатку. Один з принципів роботи – підвантаження *JavaScript* скриптів. Зручно застосовувати в

інтернет-магазинах, сайтах-каталогах, будь-яких великих інтернет-проєктах, що вимагають обробки великих масивів даних.

Також використовують такі технології, як *ASP.NET, JSP, CGI*.

За призначенням веб-додатки умовно можна розділити в залежності від сфери застосування. Умовно, тому що, у будь-який інтерактивний сайт – це онлайн-додаток. Відповідно, таких сфер може бути безліч:

1) Системи бронювання і покупки: квитків, готелю, товарів, послуг;
2) Розважальні портали;
3) Фінансові та банківські інтернет-портали з функціями замовлення послуг онлайн, калькулятора кредитів, переведення валют, інтернет-банкінгом та іншими;
4) Соціальні мережі;
5) Ігри;
6) Освітні, навчальні канали, сайти телепрограм, газет;
7) Веб-версії програмного забезпечення;
8) Біржі контенту, фрілансу і т.п;
9) *CRM*. Для прикладу детально розглянемо ці популярні сервіси.

CRM – система управління проєктами, спрямована на автоматизацію обробки повного спектру інформації про клієнтів і товари.

Подібні рішення – це комплексний продукт, що поєднує функції баз даних, пошти, календаря, обліку фінансів та інші. У них можуть бути інтегровані, в залежності від потреб, різні модулі: управлінської звітності, бухгалтерії, обліку кадрів і т.д.

CRM є основою бізнесу телемаркетингових компаній і колл-центрів. Незамінні, коли потрібно налаштувати проєктну роботу з чітким поділом за ролями і зонами відповідальності, взаємодія між відділами, роботу з клієнтами. Це актуально для банків, агентств маркетингових комунікацій, компаній-розробників IT, онлайн-магазинів товарів і послуг.

Більш заточений під потреби конкретного бізнесу варіант – це *ERP*. Це *web*-додатки, розроблені для автоматизації процесів управління

внутрішньогосподарської діяльністю великих підприємств з розвиненою філіальною мережею, різними напрямками діяльності, складнопідрядних структурою. Включає модулі виробничого, фінансового управління, закупівлі і тд.

В інтернеті сьогодні представлені всі види бізнесу і категорії споживачів. Веб-додатки допомагають готувати, купувати, вибирати автомобілі, ростити дітей, вчити іноземні мови, досліджувати глибини океану і зірки. Нові технології дають можливість розробникам створювати продукт під будь-який попит, смак і гаманець.

При цьому, у всього різноманіття онлайн-додатків є загальні характерні риси.

- Вони активно підтримують розвиток *E-commerce*: переносять процеси покупки, ділової комунікації, підписання документів в інтернет;
- Це процес *win-win*: переваги отримує і продавець, і покупець;
- Інтернет-додатки допомагають компаніям-продавцям товарів і послуг бути більш мобільними, пропонувати постійно розширюється перелік послуг, обслужити в одиницю часу більше людей, продати супутні сервіси;
- Клієнт може знайти, порівняти, вибрати по набору пріоритетних особисто для нього характеристик, купити, оплатити, отримати через доставку.

1.10. Приклади застосування веб-додатків

Приклад №1.

Процедура співпраці компанії з банківською установою тепер виглядає так:

- через сайт банку можна заповнити заявки на відкриття рахунку, карток, супутніх сервісів;
- співробітник банку зв'яжеться по телефону і уточнить деталі;

– кур'єр привезе на підпис документи;
– фахівці встановлять ПЗ, що дозволяє здійснювати платежі і необхідні операції за допомогою електронно-цифрового підпису;
– можна відразу почати працювати, дозволити працювати віддалено з будь-якого куточка світу.

Приклад №2.

Ще один тренд останніх років – використання crm-програм. Власники малого бізнесу по достоїнству оцінили всі можливості, які отримують при використанні сервісу управління робочим часом і проєктами:

– прозора система планування, постановки завдань, контролю. Завжди видно, хто винен і коли дедлайн;
– впровадження та закріплення стандартів роботи з клієнтами, включаючи пошук, теплі дзвінки, продаж, післяпродажне обслуговування, програму лояльності;
– інтеграція з іншими робочими сервісами з метою розширити і уніфікувати звітність.

1.11. Висновки до розділу

В першому розділі було проведено аналіз веб-додатків та етапів їх створення. Веб-додаток – це програмне забезпечення, яке зберігається на віддаленому сервері і доступно через Інтернет.

Досліджено, що існує два типи веб-додатків – колективні та персональні. Останні орієнтовані на домашні сайти, сайти-візитки або резюме, а колективні підходять для створення сайтів великих компаній з використанням великої кількості інформаційних ресурсів. Наприклад корпоративні сайти, інтернет-магазини, представницькі сайти.

Веб-додаток реалізовується за технологією «клієнт-сервер», клієнтом виступає будь-який браузер. У браузері відображається інтерфейс користувача, формується запит то сервера і обробляє відповіді від нього. Сервер отримує

запити від клієнта, виконує обчислення та формує веб-сторінку в залежності від запиту користувача, після чого відправляє її клієнту через мережу з використанням протоколу *HTTP/HTTPS*. Для збільшення продуктивності та інтерактивності веб додатків, було розроблено нові підходи до розробки веб-додатків, одним з яких є *AJAX*. Даний метод дозволяє підвантажувати дані до сторінки не потребуючи її повного перезавантаження.

Окремо було досліджено типи веб-додатків, а саме *ERP, CRM, E-commerce*. Кожен з наведених типів додатків має великі можливості у різних сферах. *ERP* – використовують великі підприємства для стандартизації звітності та полегшення документообігу. E-commerce – це загальне визначення інтернет-магазинів, такі додатки спрямовані на прийом, обробку, управління та відстеження замовлень. *CRM* – веб-додатоки призначені для спрощення взаємовідносин з клієнтами, покупцями. Вони дозволяють ефективно вирішувати різні завдання, серед яких контроль і планування.

Окремо досліджено питання структурних схем сторінок – основний результат робіт з проєктування. Вони в деталях показують, яка інформація і елементи керування мають відображатися на кожній сторінці системи. А також розставляють акценти – які з елементів сторінки більш, а які – менш важливі.

Проаналізовано шість етапів реалізації веб-додатків – визначення цілей та задач проєкту, розробка структури, розробка дизайн-макету, html-розмітка, програмування та контроль якості, підключення готової або створення своєї системи керування вмістом, підключення баз даних та наповнення їх контентом, запуск та супровід.

РОЗДІЛ 2
ВЕБ-СКРАПІНГ

2.1. Що таке веб-скрапінг

У широкому розумінні веб-скрапінг – це збір даних із різних інтернет-ресурсів. Принцип роботи скраперів наступний: певний код виконує *GET*-запити на цільовому веб-сайті та отримуючи відповідь, парсить (перебирає) *HTML*-документ, шукає дані та перетворює їх у заданий формат.

До категорій корисних даних може належати:

- каталог товарів;
- зображення;
- відео;
- текстовий контент;
- відкриті контактні дані – адреса електронної пошти, телефони тощо.

Сфери застосування інструментів скрапінга:

- Створення списків постачальників, виробників, продавців та інших осіб для комерційного використання. Контактна інформація витягується з різних сайтів;
- Збір таргетованої інформації для маркетингових досліджень;
- Пошук вакансій або співробітників;
- Моніторинг і порівняння цін на товари в різних магазинах.

Щоб розуміти на скільки важливу роль на сьогоднішній день відіграє скрапінг можна розказати про позицію таких великих компаній як *Competera*, *Amazon* и *Walmart*:

За словами генерального директора компанії по оптимізації роздрібних цін *Competera, Amazon* і *Walmart* створюють цілі відділи, які займаються скрапінгом. Інші звертаються до таких компаній як *Competera*. Вони збирають дані про ціни з усього інтернету, починаючи від роздрібної торгівлі взуття компанії *Nine West* і закінчуючи промисловим обладнанням компанії *Deelat*, і використовують алгоритми машинного навчання, щоб допомогти своїм клієнтам вирішити, скільки коштує заплатити за різні продукти.

Скрапінг – це частина роботи в мережі. *Google* і *Bing* теж використовують скрапінг веб-сторінок, щоб індексувати їх для своїх пошукових систем. Академіки і журналісти використовують програмне забезпечення для збору даних.

Існує маса рішень для скрпаінга веб-сайтів. Серед них:

- Окремі сервіси, які працюють через *API* або мають веб-інтерфейс (*Embedly, DiffBot* і ін.);
- Проєкти з відкритим кодом, на різних мовах програмування (*Goose, Scrapy - Python; Goutte - PHP; Readability, Morph - Ruby*).;

Вивчивши найрізноманітніші рішення, можна сміливо заявити: ідеального скрапера не існує, через велику кількість можливих проблем на шляху скрапера:

- Жоден сайт не має ідеальної верстки з точки зору догматів веб-дизайну. Саме це робить кожен сайт унікальним і привабливим для користувачів;
- Кожен веб-розробник (якщо він не працює в солідній IT компанії зі своїми правилами і стайл-гайдами) пише код під себе або просто, як вміє. Далеко не завжди код виходить грамотним і якісним. Найчастіше в ньому можна знайти величезну кількість помилок. У тому числі граматичних. Все це робить "самописний" код абсолютно нечитабельним для скраперів (мова, в першу чергу, йде про верстку);
- Маса веб-ресурсів використовує *HTML5*, де кожен елемент може бути абсолютно унікальним;

- Деякі ресурси містять різноманітні системи захисту від копіювання даних, а значить і від скрапінга. Це виражається в багаторівневої верстці, використанні *JavaScript* для рендерингу контенту, перевірки *user-agent* і т.д.;
- Залежно від сезону або тематики цільового матеріалу на сайті можуть бути використані різні макети. Періодично це стосується навіть типових сторінок (сезонні акції, преміум статті і т.д.);
- Крім корисних блоків, веб-сторінка часто рясніє "сміттям" у вигляді реклами, коментарів, додаткових елементів навігації і т.д.;
- Вихідний код може містити посилання на одні й ті ж картинки різних розмірів, наприклад - для попереднього перегляду;
- Сайт може визначити країну, в якій знаходиться ваш сервер і віддати інформацію не необхідною мовою;
- У всіх сайтів може бути різне кодування, що не отримується у відповіді на запит.

Перераховані вище фактори серйозно ускладнюють процес веб-скрапінга. В результаті якість контенту може впасти до 20% і навіть до 10%, що абсолютно неприйнятно. Не забуваємо – повнота інформації є її найважливішим критерієм.

Оскільки проблем справді багато, в даній ситуації можна зробити висновок, наступним чином:

- При необхідності отримувати дані з невеликої кількості джерел, краще написати свій скрапер і налаштувати його під потрібні сайти (якість одержуваного контенту - близько 100%);
- Використовувати комплексний підхід у виборі рідера (скрапера), якщо потрібно отримувати інформацію з великої кількості джерел (до 95% якості).

Варто розрізняти два типи скраперів це любительський та професійний. Для першого, пишеться програма-скрапер для збору інформації лише одним методом. У випадку створення професійного скраперу, пишуть декілька скраперів які згодом використовуються в різних ситуаціях.

Це працює наступним чином, готується механізм отримання *HTML* коду по *GET* запиту. Далі розглядаємо *DOM* структуру цільового сайту і визначаємо вузли з цікавою для нас інформацією. Після цього створюємо обробник вузлів і виводимо дані в нормалізованому вигляді (за бажанням замовника або зручності обробки результатів - наприклад, в форматі *JSON*).

Розглянемо систему скрапігну «*Duck System*».

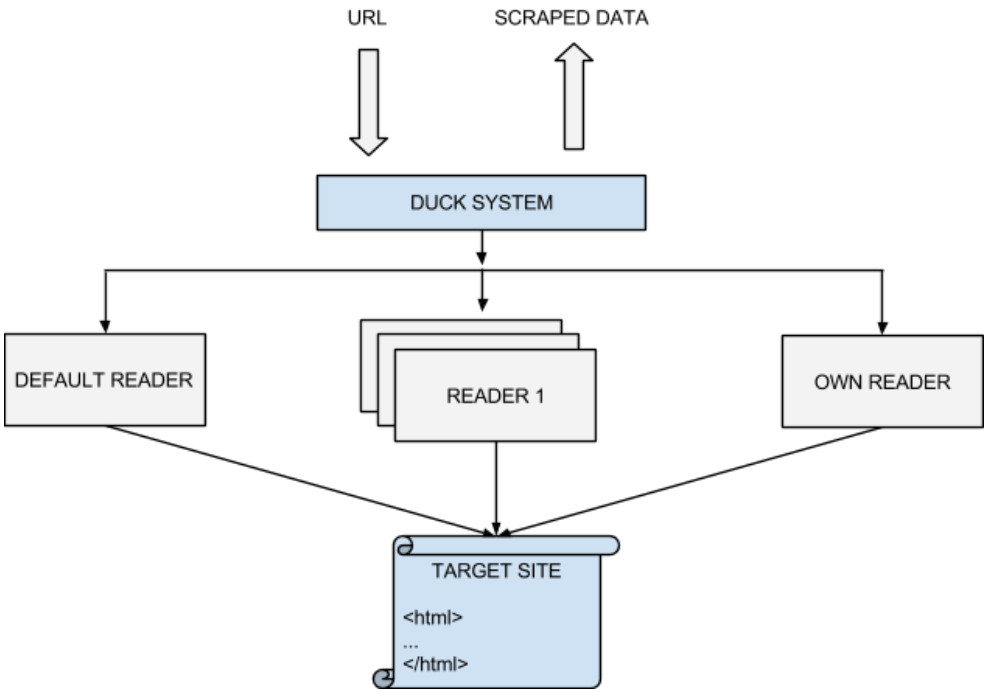

Рис. 2.1. Схематичне зображення роботи системи «*Duck System*»

На вході наша система отримує *URL* цільової сторінки, а на виході буде створено нормалізовані дані (наприклад, у форматі *JSON*). Отримавши *URL*, системі необхідно визначити якого зчитувача(рідера) слід віддати заданий *URL* на обробку (ми точно знаємо, що цей рідер читав сторінки сайту з найвищому якістю і під нього були внесені відповідні налаштування). Якщо ж рідер для цільового сайту відсутній, його читає рідер, який використовується за умовчанням (як правило це або найстабільніший скрапер, або сторонній сервіс). Отримавши від *Duck System* завдання, рідер її виконує.

На схемі представлений ще один скрапер (праворуч). Це додатковий, постійно допрацьовуваний рідер. Він призначений для читання сайтів, які не зміг

обробити скрапер, який використовується за умовчанням. Причому доопрацьовуватися він може як розробниками, коли ситуація дуже складна і скрапер написати досить важко, так і спеціально навченим адміністратором системи, якому достатньо скопіювати *XPATH* (*XPATH* – це мова запитів до елементів *xml* або *xhtml* документа. Також як *SQL*, *xpath* є декларативною мовою запитів) контейнера з контентом і контейнерів, які потрібно вирізати з результату.

Такий підхід дозволяє підняти якість одержуваного контенту на максимальну висоту. Однак за все доводиться платити. У нашому випадку платою за використання системного підходу є час обробки і ресурси сервера. А також підписка на сторонній сервіс веб-скрапінга – вона коштує грошей. Ці витрати можуть перевищити вкладення в серверну інфраструктуру і роботу всіх разом узятих програмістів.

Переваги власних рішень для невеликої кількості сайтів полягають в їх швидкості. Обробка однієї сторінки – від 7*ms*. Існує проблема обмеження швидкості інтернету та розмірів завантажувальних файлів. Вирішити це питання можна за допомогою асинхронного завантаження медіа та основного контенту в бекграунд. В результаті файли розміром від 100 *Мб* вантажаться неймовірно швидко, а результати на виході дають стовідсоткову точність.

2.2. Варіанти побудови власного веб-скраперу

Існує два типи контенту – статичний та динамічний. Статичний контент це готовий *HTML*-файл, який не є змінним, динамічний контент – це одно-сторінкові сайти, інформація у яких динамічно підвантажується не перезавантажуючи сторінку в цілому, зазвичай це реалізовано за допомогою *JavaScript* коду.

Якщо планується скрапінг веб-сторінки, потрібно незначна комп'ютерна потужність і мінімум часу.

Однак це працює тільки в тому випадку, якщо вихідний *HTML* код містить дані, на які необхідно орієнтуватися. Щоб перевірити це в *Chrome*, необхідно натиснути правою кнопкою миші сторінку і виберіть «Перегляд коду сторінки». Відобразиться вихідний код *HTML*.

Як тільки знайдено дані, обираємо *CSS* селектор, що належить wrapping елементу, щоб пізніше мати посилання на даний елемент.

Для реалізації відправляємо *HTTP*-запит *GET* на *URL*-адресу сторінки і отримати назад вихідний *HTML* код.

У *Node.js* можна використовувати інструмент *CheerioJS*, щоб парсити необроблений *HTML* і отримувати дані за допомогою селектора. Код буде виглядати так:

```
const fetch = require('node-fetch');
const cheerio = require('cheerio');
const url = 'https://example.com/';
const selector = '.example';
fetch(url)
  .then(res => res.text())
  .then(html => {
    const $ = cheerio.load(html);
    const data = $(selector);
    console.log(data.text());  });
```

У багатьох випадках отримати доступ до інформації не можливо з необробленого *HTML*-коду, тому що *DOM* керувався *JavaScript*, виконуваним у фоновому режимі. Типовим прикладом цього є *SPA* (односторінковий додаток), де *HTML*-документ містить мінімальний обсяг інформації, а *JavaScript* заповнює його під час виконання.

У цій ситуації рішення полягає в тому, щоб створити *DOM* і виконати сценарії, розташовані в вихідному коді *HTML*, як це робить браузер. Після цього дані можуть бути вилучені з цього об'єкта за допомогою селекторів. Перечислимо варіанти скрапінгу динамічних сторінок:

1) *Headless* браузери

Headless браузер це те ж саме, що і звичайний браузер, тільки без користувальницького інтерфейсу. Він працює у фоновому режимі, і ви можете керувати ним програмно замість того, щоб клацати мишею і друкувати з клавіатури.

Puppeteer один з найпопулярніших *headless* браузерів. Це проста у використанні бібліотека *Node.js*, яка надає *API* високого рівня для управління *Chrome* в автономному режимі. Наступний код буде працювати і з динамічними сторінками:

```
const puppeteer = require('puppeteer');
async function getData(url, selector){
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto(url);
  const data = await page.evaluate(selector => {
    return document.querySelector(selector).innerText;
  }, selector);
  await browser.close();
  return data;
}
const url = 'https://example.com';
const selector = '.example';
getData(url,selector)
  .then(result => console.log(result));
```

Звичайно, можна робити більш цікаві речі з *Puppeteer*, оскільки він являє собою повноцінний браузер, тут можливо навіть робити скріншот сторінки, для цього необхідно використовувати функцію «*takeScreenshot(url, path*);», де *url* – посилання на сторінку, *path* – шлях куди зберігати зображення.

Для роботи браузера потрібно набагато більше обчислювальної потужності, ніж для відправки простого запиту *GET* і аналізу відповіді. Тому

виконання щодо повільне. Також додавання браузера в якості залежності робить пакет масивним.

З іншого боку, цей метод дуже гнучкий. Його можна використовувати для навігації по сторінках, імітації роботи миші і використання клавіатури, заповнення форм, створення скріншотів або створення *PDF*-сторінок, виконання команд в консолі, вибору елементів для вилучення текстового вмісту. В принципі, все, що може бути зроблено вручну в браузері.

2) Побудова *DOM*

Оскільки модулювати цілий браузер лише для створення *DOM* – це не зовсім ефективно з точки зору витрат ресурсів, то можна використовувати *Jsdom*.

Jsdom - бібліотека *Node.js*, яка аналізує переданий *HTML*, як це робить браузер. Однак це не браузер, а інструмент для побудови *DOM* із заданого вихідного *HTML* коду, а також для виконання коду *JavaScript* в цьому *HTML*. Завдяки цій абстракції *Jsdom* може працювати швидше, ніж *headless* браузер.

Але у використанні *Jsdom* є мінуси, вони полягають у тому, що усі скрипти на сторінці виконуються асинхронно, що виключає можливість визначити точний час коли уся сторінка буде згенерована, через це є можливість, що не уся інформація буде вилучена.

Приклад написання скраперу с використанням бібліотеки *Jsdom*:

```
const jsdom = require("jsdom");
const { JSDOM } = jsdom;
async function getData(url,selector,timeout) {
  const virtualConsole = new jsdom.VirtualConsole();
  virtualConsole.sendTo(console, { omitJSDOMErrors: true });
  const dom = await JSDOM.fromURL(url, {
    runScripts: "dangerously",
    resources: "usable",
    virtualConsole
  });
```

```
  const data = await new Promise((res,rej)=>{
    const started = Date.now();
    const timer = setInterval(() => {
      const element = dom.window.document.querySelector(selector)
      if (element) {
        res(element.textContent);
        clearInterval(timer);
      }
      else if(Date.now()-started > timeout){
        rej("Timed out");
        clearInterval(timer);
      }
    }, 100);
  });
  dom.window.close();
  return data;
}
const url = "https://example.com/";
const selector = ".example";
getData(url,selector,2000).then(result => console.log(result));
```

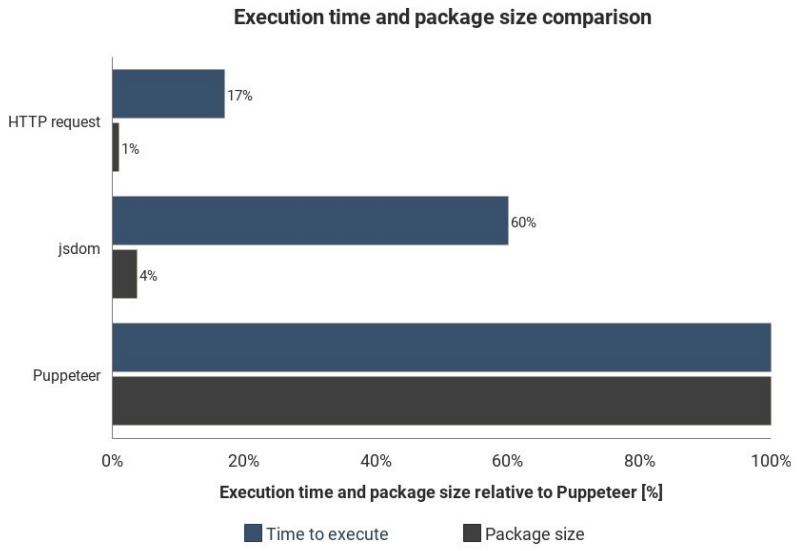

Рис. 2.2. Діаграма часу виконання і розміру пакета

3) Реверс-інжиніринг

Веб-сторінка, яку необхідно парсити, складається з тих же *HTML* і *JavaScript*. Таким чином, якщо знайти фрагмент коду, з якого були отримані цільові дані, можна повторити ту ж операцію, щоб отримати той же результат.

Якщо все спростити, дані, які необхідно знайти, можуть бути:

- частиною вихідного *HTML* коду;
- частиною статичного файлу, на який є посилання в документі *HTML* (наприклад, рядок в файлі *JavaScript*);
- відповіддю на мережевий запит (наприклад, деякий код *JavaScript* відправив запит *AJAX* на сервер, який відповів рядком *JSON*).

Ці джерела даних можуть бути доступні за допомогою мережевих запитів. Не має значення, який протокол використовує веб-сторінка *HTTP*, *WebSockets* або будь-якої інший комунікаційний протокол.

Знайшовши ресурс, що містить дані, можна відправити аналогічний мережевий запит на той же сервер, що і вихідна сторінка. В результаті отримуємо відповідь, що містить цільові дані, які можна легко витягти за допомогою регулярних виразів, строкових методів, *JSON.parse* і т. д.

Можна взяти ресурс, на якому розташовані дані, замість того, щоб обробляти і завантажувати весь матеріал. Таким чином, проблема, показана в інших методах, може бути вирішена за допомогою одного *HTTP*-запиту замість управління браузером або складним об'єктом *JavaScript*.

Це рішення здається простим в теорії, але в більшості випадків воно може бути трудомістким і вимагає досвіду роботи з веб-сторінками та серверами.

Необхідно розпочати з спостереження за мережевим трафіком. Хороший інструмент для цього – вкладка *Network* в *Chrome DevTools*. Тут видно усі вихідні запити з відповідями (включаючи статичні файли, *AJAX*-запити і т.д.), щоб виконувати їх ітерацію і пошук даних.

Якщо відповідь буде змінена будь-яким кодом перед відображенням на екрані, процес буде повільнішим. В цьому випадку необхідно знайти цю частину коду і зрозуміти, що в ній відбувається.

Отже, такий спосіб може потребувати набагато більше роботи, ніж методи, описані вище. З іншого боку, це забезпечує найкращу продуктивність.

2.3. Популярні веб-скрапери

2.3.1. *Scraper API*

Scraper API дозволяє отримати вміст *HTML* з будь-якої сторінки за допомогою виклику *API*. З Scraper *API* можна з легкістю працювати з браузерами і проксі-серверами і обходити перевірочні код *CAPTCHA*. Єдине на що необхідно зосередитися це перетворення веб-сайтів в цінну інформацію. З цим інструментом практично неможливо бути заблокованим, так як він змінює *IP*-адреси при кожному запиті, автоматично повторює невдалі спроби і вирішує *CAPTCHA*.

2.3.2. *Octoparse*

Octoparse це безкоштовний інструмент призначений для веб скрапінга. Він дозволяє отримувати дані з інтернету без рядка коду і перетворювати веб-сторінки в структуровані дані всього за один клік. Завдяки автоматичної ротації *IP*-адрес для запобігання блокування та можливості планування подальшого скрапінга цей інструмент є одним з найефективніших.

2.3.3. *DataOx*

DataOx – справжній експерт в області скрапінга веб-сторінок. Інструменти пропонуються компанією *DataOx* забезпечують великомасштабні збори даних і

надають комплексні рішення адаптовані до потреб клієнтів. Цій компанії можуть довіряти як стартапи, що створюють продукти на основі даних, так і великі підприємства, які вважають за краще доручати збір власних даних професіоналам.

2.3.4. *ScrapingBot*

ScrapingBot пропонує потужний *API* для отримання *HTML*-вмісту. Компанія пропонує *API*-інтерфейси для збору даних в області роздрібної торгівлі (опис продукту, ціна, валюта, відгук) і нерухомості (ціна покупки або оренди, площа, місце розташування). Доступні тарифні плани, *JS*-рендеринг, парсинг з веб-сайтів на *Angular JS, Ajax, JS, React JS*, а також можливість геотаргетинга роблять цей продукт незамінним помічником для збору даних.

2.3.5. *Wintr*

Wintr це *API* для парсинга веб-сторінок, що використовує обертові резидентні проксі, який дозволяє добувати і аналізувати будь-які дані доступні в мережі. Простий у використанні і повністю налаштовується *Wintr* включає безліч інструментів для збору даних навіть з найскладніших веб-сайтів. Наприклад, можна легко витягти вміст з загальнодоступною веб-сторінки за допомогою ротаційної *IP*-адреси або автоматизувати аутентифікацію за допомогою *JavaScript* рендеринга, а потім витягти особисті дані за допомогою файлів cookie і постійного *IP*-адреси.

2.3.6. *Import.io*

Import.oi – SaaS платформа, яка дозволяє перетворювати напів-структуровані веб-дані в структуровані. Платформа надає можливість вилучення даних в реальному часі за допомогою *API*-інтерфейсів *JSON REST* і потокової

передачі, а також легко інтегрується з багатьма мовами програмування і інструментами для аналізу даних.

2.3.7. *Webhose.io*

Webhose.io це розширений *API* сервіс для вилучення веб даних в реальному часі. Це інструмент дуже часто використовують для вилучення історичних даних, моніторингу ЗМІ, бізнес-аналітики, фінансового аналізу, а також для академічних досліджень.

2.3.8. *Zyte*

Zyte (раніше *ScrapingHub*) – спеціалізується на швидкому й ефективному отриманін даних з використанням технологій з відкритим вихідним кодом. Інструмент обробляє понад 3 мільярдів веб-сторінок на місяць. Сьогодні *Zyte* пропонує чотири різних типи інструментів для парсинга веб-джерел – *Smart Proxy Manager* (раніше *Crawlera*), *AutoExtract, Scrapy Cloud і Splash.*

2.3.9. *ParseHub*

ParseHub – безкоштовний інструмент для парсинга веб-сторінок з простим і зрозумілим інтерфейсом. За допомогою *ParseHub* можна автоматично завантажувати очищені дані в будь-якому форматі для подальшого аналізу.

2.3.10. *Mozenda*

Mozenda це корпоративне програмне забезпечення розроблене для всіх видів завдань по вилученню даних. Цій компанії довіряють тисячі підприємств і більше 30% компаній зі списку *Global Fortune 500*. Це один із кращих інструментів для парсинга веб-сторінок, який допоможе за лічені хвилини

створити скрапер агента. *Mozenda* також пропонує функції *Job Sequencer and Request Blocking* для збору веб-даних в реальному часі і кращий сервіс для роботи з клієнтами.

2.3.11. *Luminati*

Luminati пропонує інструмент для збору даних нового покоління, який дозволяє отримувати автоматизований і настроюється потік даних за допомогою однієї простої панелі управління. Необхідно тільки відправити запит, а всім іншим: *IP*-адресами, заголовками, файлами *cookie*, *CAPTCHA* буде керувати система. З *Luminati* ви отримаєте структуровані дані з будь-якого веб-сайту, від тенденцій в електронній комерції та даних з соціальних мереж до досліджень ринку і конкурентної розвідки.

2.3.12. *Outwit*

Outwit - цей інструмент для збору даних дозволяє легко отримувати будь-яку інформацію з інтернету. Для отримання даних з сайтів за допомогою *Outwit Hub* не потрібні навички програмування. Одним клацанням миші можна запустити парсинг на сотнях веб-сторінок. На вибір пропонуються три пакети: *Pro, Expert і Enterprise*.

2.4. Висновки до розділу

В другому розділі було досліджено, що таке скрапер та скрапінг. Веб-скрапінг – це збір даних із різними інтернет-ресурсами. *Web scraper* – це програма для автоматичного вилучення інформації з веб-сайтів. Після вилучення інформації її структурують в необхідному порядку, після чого вона може бути як для звичайного перенесення в іншу БД та відображення на певній сторінці, так і для аналітики.

Також в розділі проаналізовано методи створення веб-скраперів. Вибір методу повинен залежати від того, які сторінки будуть піддаватися скануванні – динамічні чи статичні. Для динамічних сторінок варто використовувати такі методи, як *Headless* браузери, побудова *DOM* (наприклад за допомогою *Jsdom*), реверс-інжиніринг.

Не дивлячись на те, що існує велика кількість скраперів, кожен з них має власні переваги та недоліки, при цьому майже усі вони платні. Тому як вже зазначалося у розділі, найкращим варіантом є написання власного скраперу, який хоч і буде вузько-направленим, але якість збереженої інформації буде на найвищому рівні.

У результаті проведення аналізу обраної сфери, можна прийти до висновку, що для виконання парсингу інформації автомобілів, варто створити власний веб-додаток для збереження та відображення інформації. При цьому додаток повинен бути простий та зрозумілий у використанні.

РОЗДІЛ 3
РОЗРОБКА ВЕБ-ДОДАТКУ

3.1. Основні використовувані компоненти

В ході розробки «Веб-додатку для пошуку та підбору автомобілів» використовувалися наступні технології:

- середовище розробки *Visual Studio Code*;
- мова програмування *JavaScript*;
- мова розмітки *HTML*;
- структура даних *JSON*;
- сервер *Node.js;*
- фреймворк *Express.js*;
- СУБД *MongoBD*.

Файлова структура додатку:

- *Scraper*
  - *saveCar.html*
  - *node.js*
  - *package.json*
  - *package-lock.json*
- *WebSite*
  - *index.html*
  - *index.js*
  - *main.css*

– *package.json*

– *package-lock.json*

У *Node*.js- *package.json* файл, який зберігає конфігурацію для веб- додатку. Менеджер пакетів *Node.js (npm*) використовуватиме це для встановлення будь-яких залежностей або модулів, які будуть використовуватись. В даному випадку буде використовуватись:

Express – популярний *Node.js framework.*

Mongoose – моделювання об'єктів для БД *MongoDB.*

Axios – це широко відома *JavaScript*-бібліотека. Вона представляє собою *HTTP*-клієнт, заснований на продуктах і призначений для браузерів та для *Node.js*.

Cheerio – *JS*-бібліотекка, містить у собі *HTML*-парсер, *API* якого схожий, як *API jQuery*)

Body-parser – модуль проміжного програмного забезпечення, *Цей* модуль аналізує дані *JSON*, буфера, рядки і *URL*-адреси, передані з використанням запиту *HTTP POST*.

EJS – шаблонізатор для платформи *Node.js.*

```
PS D:\new\scraping\TEST NPM> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (test-npm)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to D:\new\scraping\TEST NPM\package.json:

{
  "name": "test-npm",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes)
PS D:\new\scraping\TEST NPM>
```

Рис. 3.1. Ініціалізація пакетів *npm* у проєкті

Розпочати створення веб-додатку на мові *JavaScript* необхідно з встановлення програмного пакету *Node.js*, після чого відкриваємо термінал та переходимо у кореневу директорію для ініціалізації проєкту – для цього прописуємо команду *npm init*. Саме після запуску цієї команди у корені проєкту з'являється файл *package.json* в якому *npm* зберігає усі свої налаштування підключених пакетів.

Наступним кроком є встановлення додаткових пакетів, що були перераховані на початку даного розділу. Для цього необхідно ввести послідовно наступні команди *npm install mongoose, npm install express, npm install body-parser, npm install axios, npm install cheerio* або ж об'єднавши це в одну команду *npm install express mongoose body-parser axios cheerio.*

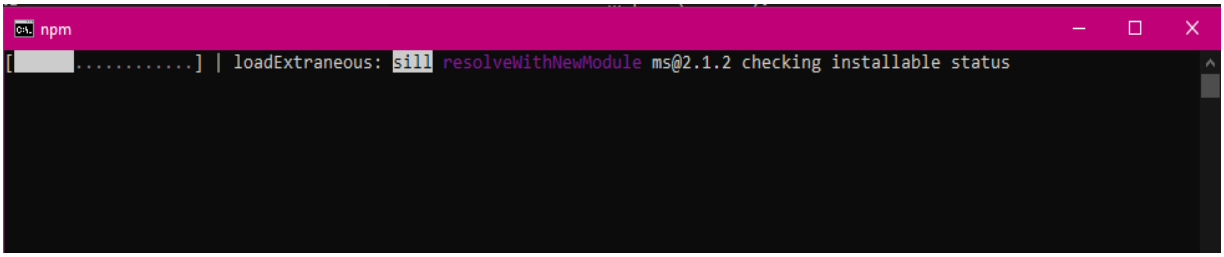

Рис. 3.2. Встановлення додаткових пакетів *NPM*

Додаток для зручності функціонування буде розділений на дві частини – перша частина у якій буде відбуватися власне скрапінг та збереження даних у базу даних з головним файлом *node.js*, та друга частина яка буде відповідати за відображення та пошук даних у БД, з головним файлом *website.js*.

Перша частина додатку, файл *node.js,* містить в собі:

– Налаштування скраперу;

– Підключання до БД;

– Створення моделі *Mongoose;*

– Визначення шляху до зовнішнього інтерфейсу.

Після встановлення усіх пакетів їх необхідно підключити до проєкту, робиться це за допомогою наступних рядків коду:

const axios = require('axios');

const cheerio = require('cheerio');

const mongoose = require('mongoose');

const bodyParser = require("body-parser");

const express = require('express');

const router = express.Router();

Тепер використовуючи підключений модуль *'express'*, необхідно реалізувати локальний сервер, виглядає це наступним чином:

const { createServer } = require('http');

const app = express();

const port = 3000;

const server = createServer(app);

server.listen(port, () => console.log(`server is up. port: ${port}`));

Сервер готовий, тепер при виконанні файлу *node.js* він буде запускатися та прослуховувати 3000-ний порт.

Рис. 3.3. Порт веб-серверу

3.2. Створення *Node.JS API*

Створення *RESTful* дозволить нам мати *API*, який буде отримувати інформацію від користувача, які сторінки необхідно надати для парсину. Він буде повертати всю цю інформацію в форматі *JSON* та записувати її у БД.

Створюємо модель *Car* для *MongoDB*:

const CarsSchema = new mongoose.Schema({

title:{ type: String, required: true, },

money:{ type: Number,: true, },

characteristic:{ type: String,: true, },

descriptions:{type: String, required: true, },

```
    link:{type: String, required: true, unique:true    },
    imgsrc:{ type: String, required: true,    }});
const Cars = mongoose.model('Cars', CarsSchema);
```

Створюємо шляхи *Express* для обробки *API* викликів.

Виклик сторінки для збереження даних:

```
app.get('/, loadCarSavePage);
function loadCarSavePage(err, res){
    res.sendFile(__dirname + "/saveCars.html");}
```

Функція виклику та передачі даних у парсер:

```
app.post('/loadcar', async function (req, res){
    if(!req.body) return res.sendStatus(400);
    console.log(req.body);
    var form = req.body;
    const urlStart = form.URL;
    const startPage =  Number(form.startPage);
    const endPage =  Number(form.endPage);
     SaveToBDAutoria(urlStart, startPage, endPage);
     await new Promise(resolve => setTimeout(resolve, 5000));
    res.redirect("http://localhost:5000/test");});
function SaveToBDAutoria(urlStart, startPage, endPage){
    console.log(startPage !== endPage);
    if(startPage !== endPage+1){
       console.log(url + "");
    var url = urlStart + "?page=" + startPage;
    axios.get(url)
    .then(response => {
       getData(response.data)    })
    .catch( error => {        console.log(error);    })
    let getData = html => {
       var data = [];
```

```
const $ = cheerio.load(html);
$('.content-bar').each((i, elem) => {
  console.log(url + "");
  data.push({
    title : $(elem).find('.head-ticket').text(),
    money : $(elem).find('.size22').text().replace(/[^\d]/g, ''),
    characteristic: $(elem).find('.characteristic').text(),
    descriptions : $(elem).find('.descriptions-ticket').text(),
    link : $(elem).find('a.m-link-ticket').attr('href'),
    imgsrc: $(elem).find('img.outline').attr('src')
  });          });
data.forEach(function(entry) {
  Cars.create({
    title: entry.title,
    money: entry.money,
    characteristic: entry.characteristic,
    descriptions: entry. descriptions,
    link: entry.link,
    imgsrc: entry.imgsrc          });       });    }
startPage++;
SaveToBDAutoria(urlStart, startPage, endPage);}}
```

Функція відображення інформації на сторінці:

```
app.get("/cars",Loading);
async function Loading(req, res){
  var carsDB = await Cars.find().limit().sort({money: -1});
  ejs.renderFile( __dirname + "/index.html", { carsDB: carsDB }, function(
err,str) {
    if (err) console.log(err);
    res.send(str);   });}
```

Функція власного пошуку, сортування та відображення інформації:

app.get("/customsort", CustomLoading);

async function CustomLoading(req, res){

var sort = req.query.sort;

var find = req.query.find;

if(find == ""){ var carsDB = await Cars.find().limit().sort({money: sort});

}else{

var carsDB = await Cars.find({$text: {$search: find}}).limit().sort({money: sort}); } ejs.renderFile(__dirname + "/index.html", { carsDB: carsDB }, function(err,str) {

if (err) console.log(err);

res.send(str); });}

На основі цих шляхів можна побудувати таблицю, яка буде пояснювати, як зовнішній додаток повинен робити запити даних з *API* (табл. 3.1).

Таблиця 3.1

Запити даних з *API*

HTTP-назва	*URL*	Опис
GET	/	Отримати форму для заповнення даних
POST	/*loadcar*	Відправити дані для скраперу
GET	/*cars*	Відображення інформації
GET	/*customsort*	Функція власного пошуку та сортування

3.3. Створення інтерфейсу додатку

Усі налаштування інтерфейсу додатку знаходяться у файлах *index.html* та *saveCar.html*.

У файлі *saveCar.html* знаходиться інтерфейс для скраперу. Він складається з однієї форми, що включає в себе одне текстове поле для посилання, два поля типу *number* для вибору початкової та кінцевої сторінки для скраперу та кнопка для відправлення форми.

Форма налаштувань скраперу:

<form method="POST" action="/savecar" id="form">

Введіть посилання на розділ: *<input type="text" size="50" id="url" name="URL" required>
*

Введіть початкову сторінку*: <input type="number" size="1" id="start" name="startPage" required*

*style="width: 3vw;">
*

Введіть кінцеву сторінку: *<input type="number" size="1" id="end" name="endPage" required*

*style="width: 3vw;">
*

<button type="button" style="margin: 25px; padding: 8px;

background-color: black; color: white; border-radius: 10px;

font-weight: 700; font-size: 17px; " onclick='Start()'>Почати збереження

даних*</button>*

</form>

У файлі *index.html* знаходиться шаблон для відображення інформації з БД, файл *index.html* обробляється на сервері за допомогою *EJS* та відправляється кінцевому користувачу.

Шаблон файлу index.html:

* *

<% for(let car of carsDB){%>

```
<div class="content">
 <div class="image">
  <a href="<%=car.link%>" target="_blank" class="a-mobile">
   <img src="<%=car.imgsrc%>"> </a>
 </div>
 <div class="info">
  <a href="<%=car.link%>" target="_blank">
   <p id="carName"> <%=car.title%> </p>          </a>
  <div class="information">
   <p id="price"> Ціна: <%=car.money%>$</p>
  </div>
  <div style=" overflow: hidden; padding-left: 25px; padding-right:
25px;">
   <p> Короткі характеристики:<br><%=car.characteristic%></p>
   <p> Опис від продавця: <br> <%=car.descriptions%> </p>
  </div>
  <div class="link">
   <a href="<%=car.link%>" id="link" target="_blank">Перейти на
сторінку продажі</a>
  </div>       </div>       </div>
 <%}%>
</div>
```

3.4. Діаграми станів

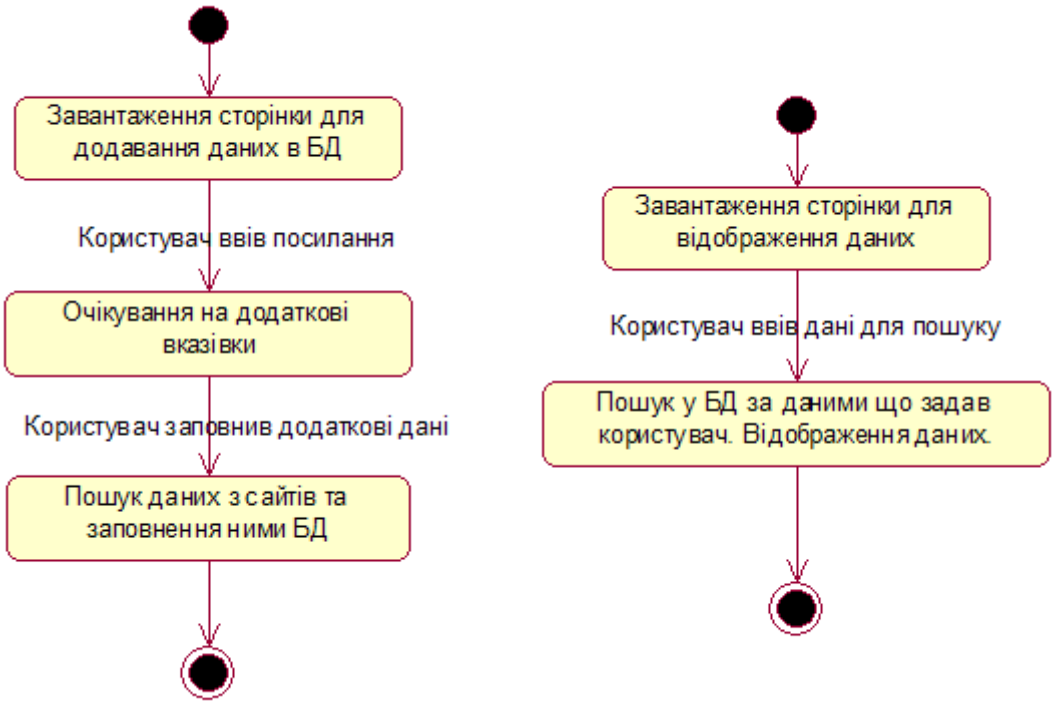

Рис. 3.4. Діаграми станів

3.5. Діаграма послідовності

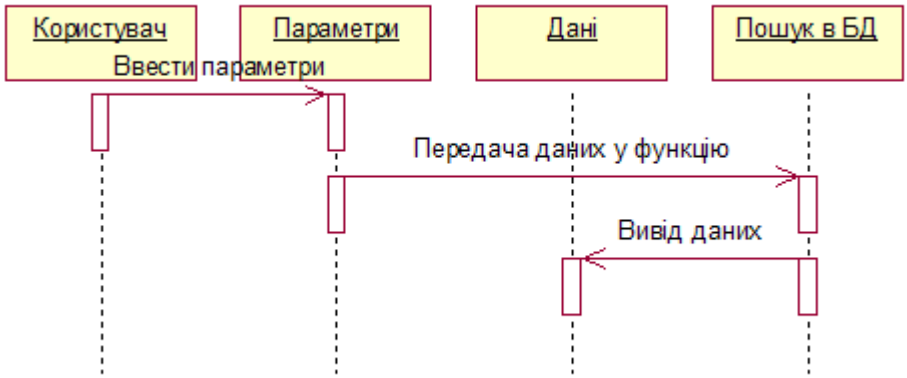

Рис. 3.5. Діаграма послідовності

3.6. Вимоги до технічного забезпечення

Для функціонування програми достатньо комп'ютера з технічними характеристиками не меншими ніж:

Процесор – *Intel Pentium 4 / Athlon 64* або пізнішої версії з підтримкою *SSE2*;

Вільний простір на жорсткому – 350 *Мб*;

Оперативна пам'ять - 512 *Мб*;

Операційна система - *Windows XP* з пакетом оновлення 2 +, *Windows Vista Windows 7, Windows 8, Windows 10;*

3.7. Робота користувача з додатком

Для запуску веб-додатку необхідно у браузері перейти за посиланням *http://localhost:3000:*

Рис. 3.6. Виклик додатку у браузері

Після виклику веб-додатку у браузері відкриється головна сторінка, яка представлена на рис. 3.5.

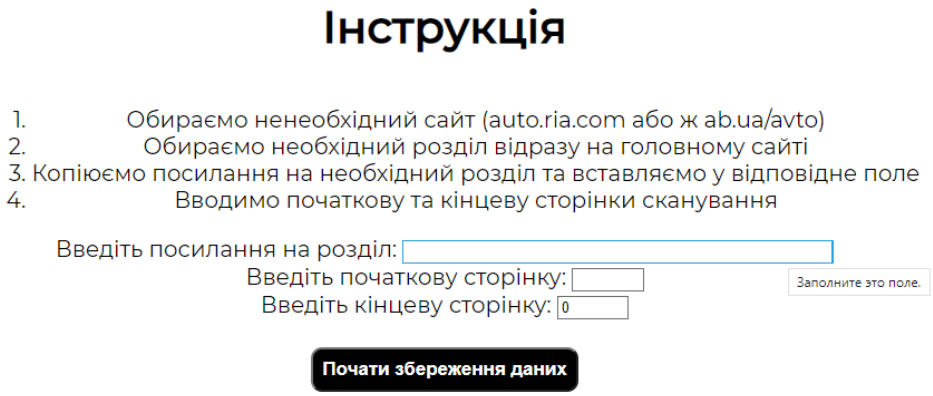

Рис. 3.7. Головне вікно додатку

Щоб розпочати збір даних про автомобілі необхідно вказати у перше поле посилання на необхідний розділ сайту, з якого буде виконуватися збір даних та обрати початкову та кінцеву сторінки цільового сайту.

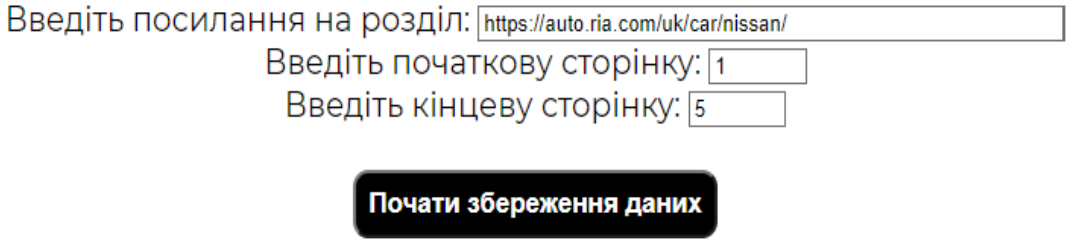

Рис. 3.8. Вигляд правильно заповнених полів

Після заповнення даних, натискаємо кнопку «Почати збереження даних». По завершенню процедури, відбудеться автоматична переадресація на сторінку з збереженими даними.

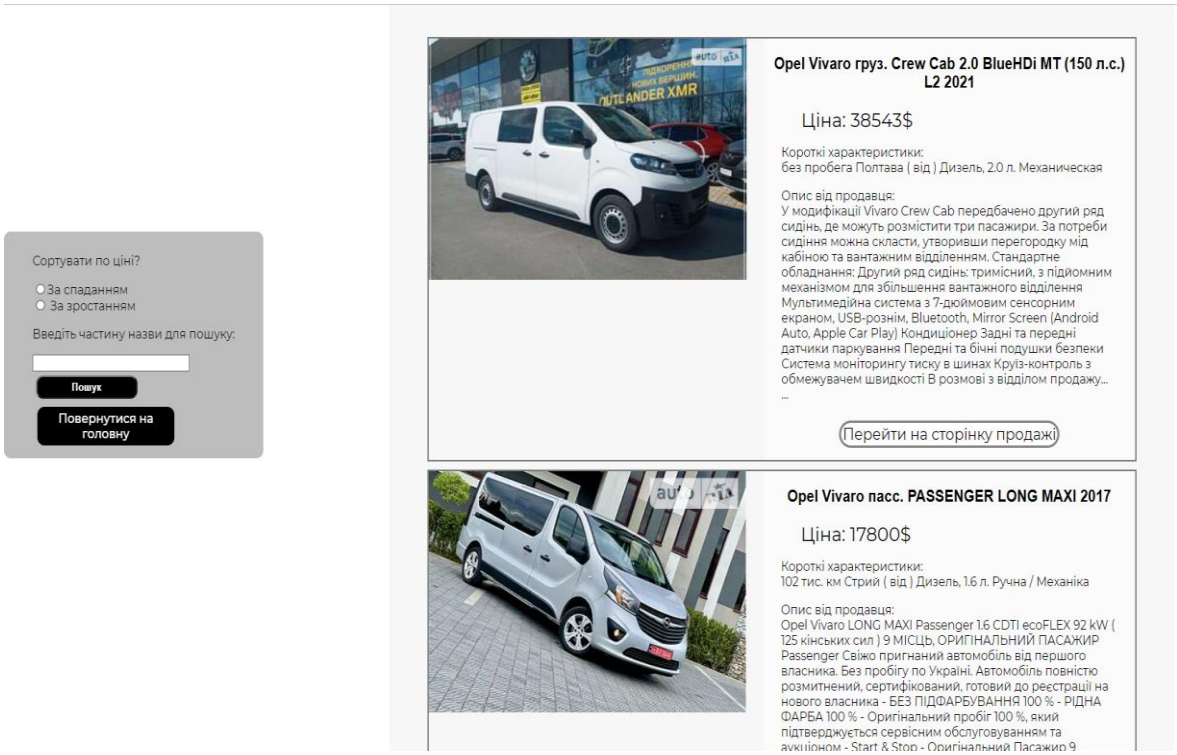

Рис. 3.9. Вигляд сторінки з збереженими авто

На даній сторінці є можливість виконувати текстовий пошук бажаного авто та сортування знайденого за ціною.

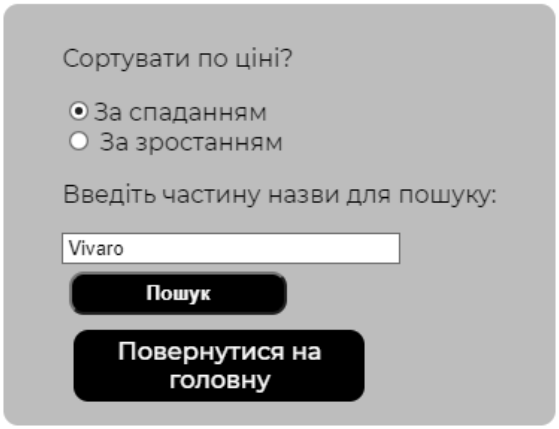

Рис. 3.10. Вікно налаштувань для пошуку

3.8. Висновки до розділу

В третьому розділі була проведена робота по створенню веб-додатку для пошуку та підбору автомобілів. Проаналізовані та застосовані технології для створення додатку, такі як мова розмітки *HTML*, таблиця стилів *CSS*, мова програмування *JavaScript*, фреймворк *Node.js*, база даних *MongoBD*.

В ході проєктування були встановлені *Express, Mongoose, Axios, Cheerio* створені файли *Node.js*, який містить основні налаштування *Node.js,* та *package.json*, який містить конфігурацію веб-додатку. Був запущений сервер з портом 3000. Створений *Node.js API* за технологією *RESTful* та його робота с базою даних.

Створено файл *index.js,* який з використанням модулю – *body-parser,* виконує рендерінг файлу *index.html* перед відправленням клієнтові.

Розроблений зовнішній інтерфейс веб-додатку з використанням *HTML* та *CSS*.

Також було досліджено та розгорнуто представлено мінімальні вимоги до програмного та технічного забезпечення. Створена інструкція користування веб-додатком для користувача.

ВИСНОВКИ

Під час виконання дипломного проєкту були розв’язані наступні задачі:

- проведено аналіз веб-додатків та етапи їх реалізації;
- описана предметна область скраперів;
- проаналізовано існуючі на даний момент в інтернеті скрапери;
- реалізовано веб-додаток для пошуку та підбору автомобілів з використанням серверу *Node.js API.*

У дипломній роботі розкрито питання веб-додатків, їх різноманітність, архітектура, класифікація, структурні та технічні особливості. Розглянуто роботу веб-сервера з динамічними та статичними сторінками. При роботі з статичними сторінками веб-сервер відправляє клієнту готовий файл, що лежить на сервері, у випадку роботи з динамічною сторінкою сервер спочатку рендерить файл в залежності від запиту, і лише потім відправляє файл клієнту.

Крім цього на сьогоднішній день існує велика кількість готових, але платних додатків для веб-скрапінгу. Було проведено їх аналіз, визначено можливості та категорії у яких варто їх використовувати.

Проведено аналіз способів написання скраперів (*Headless* браузери, Побудова *DOM*, Реверс-інжиніринг, *GET* запити).

Розроблений «Веб-додаток для пошуку та підбору автомобілів», який є простий у використанні, може використовуватися як для пошуку авто які уже існують у базі даних проєкту, так і для додавання нових авто з інших сайтів. Додаток може бути відкритий як на комп’ютері так і на смартфоні.

Зовнішній інтерфейс додатку мінімалістичний, лаконічний та зрозумілий, адаптивний для різних розмірів екрану комп’ютерів. Усі дії при роботі с додатком, такі як, створити завдання, перемістити, виконати чи видалити - інтуїтивно зрозумілі.

Розроблений програмний продукт відрізняється від аналогів тим, що є безкоштовним, мультиплатформенний та простим.

Результати дипломного проєкту можна розділити на дві групи: теоретичні (аналіз і етапи реалізація веб-додатків) та практичні (розробка веб-додатку на основі новітніх технологій). Теоретичні результати можна використовувати при розробці веб-додатків, практичні – веб-додаток можна використовувати з власних потреб.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ ГОСТ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.»
2. *Eric Freeman, Elisabeth Robson. Head First HTML and CSS*. – *O'Reilly Media*, 2005. – 694 с.
3. *htmlbook.ru* [Електронний ресурс] – Режим доступу до ресурсу: *http://htmlbook.ru/html5*
4. *Node.js* в действии. 2-е издание – Кантелон М., Хартер М., Головайчук Т., Райлих Н., 2018. – 432с.
5. *github.com/nodejs* [Електронний ресурс] – Режим доступу до ресурсу: *https://github.com/nodejs*
6. *habr.com/ru/post/301426/* [Електронний ресурс] – Режим доступу до ресурсу: *https://habr.com/ru/post/301426/*
7. *JavaScript* Подробное руководство – Дэвид Флэнаган, 2008. – 986с.
8. Д. Флэнаган. *JavaScript*. Подробное руководство, 5-е издание. – М: Символ-Плюс, 2004. – 992 с.
9. *Jennifer Niederst. Web Design in a Nutshell: A Desktop Quick Reference. – O'Reilly Media*, 2006. – 826 с.
10. *Eric Elliott. Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries*. – *O'Reilly Media*, 2014. – 254 с.
11. Браун И. Веб-разработка с применением *Node.js* и *Express*. Полноценное использование стека. – Санкт-Петербург:Питер, 2017. - 336 с.
12. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проєкти) випускників Національного авіаційного університету. – К.:НАУ, 2017.-63с.