

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка
до дипломного проєкту (роботи)
бакалавр

(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ВЕБСАЙТУ ДЛЯ МОНІТОРИНГУ
ЦІН КОНКУРЕНТІВ

WEBSITE DEVELOPMENT FOR MONITORING THE
COMPETITORS' PRICES

Виконав: студент(ка) 4 курсу, групи КНТ-228
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

Гальченко В.В.

(прізвище та ініціали)

Керівник Каплієнко Т.І.

(прізвище та ініціали)

Рецензент Шитікова О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Гальченко Валерії Віталіївни

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розроблення вебсайту для моніторингу цін конкурентів. Website Development for Monitoring the Competitors' Prices

керівник проєкту (роботи) Каплієнко Тетяна Ігорівна, к.т.н., доцент,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “27” квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 07 червня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Опис сайту. 3. Розробка сайту.

4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Каплієнко Т.І., доцент		
Нормоконтроль	Камінська Ж.К., асистент		

7. Дата видачі завдання « 04 » квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Гальченко В.В.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Каплієнко Т.І.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
123 с., 64 рис., 3 табл., 22 джерел.

MODEL-VIEW-CONTROLLER, АВТОМАТИЗАЦІЯ, БАЗА ДАНИХ, БІЗНЕС-ПРОЦЕСИ, ВЕБСАЙТ, МОНІТОРИНГ ЦІН, ПАРСЕР, ПОСЛУГА, ТОВАР, ФРЕЙМВОРК.

Об’єкт дослідження – методи і способи автоматизації бізнес-процесів, зокрема методи і підходи спостереження за ціноутворенням на товари і послуги у мережі Інтернет.

Предмет дослідження – програмні засоби для пошуку, обробки, накопичення інформації щодо товарів і послуг у мережі Інтернет.

Мета роботи – створення вебсайту для спостереження за цінами на товари і послуги компаній для прийняття відповідних бізнес-рішень.

Матеріали, методи та технічні засоби: структурне та об’єктно-орієнтоване програмування, мови програмування Python, JavaScript, бази даних MySQL, персональний комп’ютер з процесором Intel Core i3, оперативною пам’яттю 16 Гб, операційною системою Linux або Windows.

Висновки. Розроблено вебсайт для спостереження за цінами на товари і послуги компаній у мережі Інтернеті. Його використання дозволить знайти актуальну інформацію щодо цінових пропозицій на потрібні товари та прийняти відповідні бізнес-рішення щодо формування цінової політики власної компанії.

Практичне значення полягає у розробці інструменту для автоматизації бізнес-процесів підрозділу або компанії, які потребують актуальну інформацію на товари та послуги. Використання вебсайту надає користувачам можливість переглядати динаміку зміни цін та іншої інформації товари і послуги, обирати формат представлення для кращого розуміння інформації.

ABSTRACT

Explanatory note to the final qualifying work of bachelor: 123 pages, 64 figures, 3 tables, 22 sources.

MODEL-VIEW-CONTROLLER, AUTOMATION, DATABASE,
BUSINESS PROCESSES, WEBSITE, PRICE MONITORING, PARSER,
SERVICE, PRODUCTS, FRAMEWORK.

Object of study is methods and ways of business processes automation, especially methods for products and services price monitoring in the Internet.

Subject of study is software for searching, processing, gathering of products and services information in the Internet.

Purpose of the work is to create a website for products and services information gathering and make appropriate business decisions.

Materials, methods and tools: structured and object-oriented programming, Python programming languages, JavaScript, MySQL databases, personal computer with an Intel Core i3 processor, a 16 GB RAM, that runs under the control of Linux or Windows operating system.

Results. A website has been developed to monitor products and services price of companies in the Internet. Users are able to search actual information about prices of products and make important business decisions about price policy of own company.

Practical importance of the work is a tool for automation business processes of department or company, which needs up-to-date information about products and services. It allows users to view a dynamics of prices and other information changing, select a template of given information appearance.

ЗМІСТ

Перелік скорочень та умовних познач	9
Вступ	9
1 Аналіз предметної області	10
1.1 Обґрунтування ефективності аналізу цінових пропозицій	10
1.2 Методи і засоби вирішення задачі	12
1.3 Програмні засоби автоматизації	13
1.3.1 Сервіс “Price Control”	14
1.3.2 Сервіс “Z-Price”	15
1.3.3 Сервіс “PriceCop”	17
1.3.4 Сервіс “Competera”	18
1.4 Постановка задачі	20
1.5 Постановка завдань роботи	22
1.6 Висновки за розділом 1	22
2 Розробка архітектури вебсайту	23
2.1 Вимоги до функціоналу	23
2.2 Вимоги до дизайну сторінок	24
2.3 Вибір інструментарію для розробки проєкту	27
2.3.1 Вибір мови програмування на BE	28
2.3.2 Вибір фреймворку для створення BE	29
2.3.3 Вибір інструментів для створення верстки макетів	30
2.4 Структура проєкту	31
2.5 Вибір середовища розробки	33
2.6 Висновки за розділом 2	34
3 Основні рішення щодо реалізації компонентів вебсайту	35
3.1 Модуль управління обліковими записами	35
3.1.1 Створення облікових записів	35
3.1.2 Вхід до системи	37
3.1.3 Вихід з системи	38
3.1.4 Відновлення доступу до облікових записів	38

3.1.5 Перегляд облікових записів	40
3.1.6 Оновлення облікових записів	41
3.2 Модуль підтримки користувачів	42
3.3 Модуль управління даними	43
3.3.1 Введення даних користувачем	43
3.3.2 Введення даних з парсеру	44
3.3.3 Виведення інформації з бази даних	45
3.3.4 Встановлення інтервалу оновлення даних дашборду	46
3.4 Модуль безпеки даних	47
3.4.1 Політика парольного захисту	47
3.4.2 Політика рольового доступу	48
3.4.3 Активація облікових записів	50
3.4.4 Моніторинг дій користувачів	51
3.4.5 Блокування доступу користувачам	52
3.4.6 Відключення користувачів від системи	53
3.5 Структура бази даних	53
3.6 Висновки за розділом 3	64
4 Експлуатація, тестування та експериментальне дослідження вебсайту	65
4.1 Вимоги до клієнтської частини	65
4.1.1 Апаратне забезпечення	65
4.1.2 Програмне забезпечення	65
4.2 Інструкція по роботі з програмою	66
4.2.1 Розгортання серверної частини	66
4.2.2 Запуск програми	66
4.2.3 Методики роботи з вебсайтом	67
4.3 Висновки за розділом 4	75
Висновок	76
Перелік джерел посилання	78
Додаток А Текст програми	81
Додаток Б Слайди презентації	116

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface
BE	– Backend
CRM	– Customer Relationship Management
FE	– Frontend
HTML	– Extensible Hypertext Markup Language
JS	– JavaScript
MVC	–Model-View-Controller
SaaS	– Software as Service
WSG	– Web Server Gateway Interface
XML	– Extensible Markup Language
БД	– база даних
ПЗ	– програмне забезпечення
РРЦ	– рекомендована роздрібна ціна

ВСТУП

Ресурси – один з важливих атрибутів будь-якої розвинутої та незалежної країни. В сучасному світі їх кількість визначена та обмежена, що є основною причиною боротьби за них. За історичними даними [1], близько 40% війн світу велися за ресурси.

Однак війна є крайнім засобом боротьби за ресурси й здійснюється на загально національному рівні. В сучасних цивілізованих умовах вказане протистояння трансформоване у бізнес. Саме тому у боротьбі за ресурси може примати участь будь-хто від приватної особи до консалтингових, технологічних і промислових корпорацій, які є частиною економіки будь-якої країни. Основними критеріями для ведення успішної боротьби є достатній рівень знань та капіталу (ресурсів), які необхідні для успішного ведення боротьби [2, 3].

Користування послугами великої кількості кваліфікованих фахівців не завжди є економічно обґрунтованим, однак потребує значних фінансових вкладень. Тому більша частина компаній, а тим більше приватних осіб, не мають можливості користуватися їх послугами і отримувати очікуваний прибуток на постійній (довготривалій) основі. Вказана обставина є критичною для успішності бізнесу, особливо в період економічного спаду [4-5]. З метою збереження та збільшення потужностей бізнесу, а також очікуваного рівня прибутку, керівники компаній приймають рішення щодо скорочення кількості операційних витрат [6-7], а як наслідок кваліфікованих кадрів, і впровадження автоматизації бізнес-процесів з використанням технічних засобів [8].

Актуальність обраної теми визначається значущістю цінової політики, починаючи від грамотно продуманої та розробленої стратегії до прибутку та швидкості окупності початкових капіталовкладень.

Актуальність виконаної роботи пов'язана зі прогресом на ринку праці, ростом конкуренції на ринку товарів і послуг, потребами малого та середнього бізнесу у технічній підкованості та аналізу ринкових процесів з метою прийняття стратегічних рішень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

З метою визначення найбільш необхідних базових характеристик розроблюваного вебсайту та задоволення потреб кінцевих користувачів, у даному розділі будуть виконані обґрунтування ефективності аналізу цінових пропозицій, способи і методи його проведення, огляд сервісів автоматизації, вибір функцій які необхідно реалізувати на вебсайті.

1.1 Обґрунтування ефективності аналізу цінових пропозицій

Спостереження за подіями, явищами або процесами оточуючого середовища дозволяє отримати інформацію, яка є досить корисною для обґрунтування бізнес рішень у бізнесі. Під час спостереження керівник має можливість виконати оцінку ринку, керувати активом і його станом у залежності від впливу зовнішніх факторів. Вказані процеси закладені у поняття “моніторингу” [9].

У контексті даної праці розглянуто поняття “моніторингу цінових пропозицій”, яке асоціюється з необхідністю збору, обробки, аналізу та зберігання даних щодо цінових пропозицій на товари і послуги учасників ринку (конкурентів). У подальшому їх використання дозволить сформулювати цінову політику компанії і створити власну стратегію ведення бізнесу у мережі Інтернет. Результатом вказаного процесу є отримання інформації для оптимальної зміни цін за короткий проміжок часу [10].

У зв'язку зі стрімким розвитком сфери продажів і її переміщення у мережу Інтернет, розміри ринку, часу його функціонування, кількість учасників і пропозицій суттєво збільшилися. З'явилася необхідність визначати стратегію ведення бізнесу конкурентів і адаптувати бізнес-процеси власних компаній у режимі реального часу [10].

Промислові компанії, як правило, не займаються збутом своєї продукції та користуються послугами різноманітних рекламних і маркетингових

компаній, які займаються пошуком ринків збуту вказаної продукції. У зв'язку з цим, кількість найменувань товарів і послуг в останніх стрімко зростає і маркетологам складно охопити усі ринки збуту, не кажучи про аналіз цінових пропозицій в умовах мінливості ринку [10].

Вказана інформація також є важливою для постачальників і логістичних компаній для формування ефективної цінової політики. Тому для отримання необхідних даних маркетологи користуються офісними програмними засобами та спеціалізованими сервісами для відстежування цінових пропозицій на декількох Інтернет-ресурсах. Це дозволяє скоротити час на збір даних, витратити його на аналіз ринку і прийняття відповідних бізнес-рішень [10].

Спеціалізовані сервіси, які надають подібні послуги відносяться до класу “software as service” (SaaS). Вони дозволяють автоматизувати усі процеси пов'язані з відстежуванням і аналізом цінових пропозицій конкурентів на одній та більше Інтернет-платформах. Вказане дозволяє не лише знайти отримати інформацію про наявних конкурентів, а також про їх товари, послуги, ціни, оперативно реагувати на їх зміну та коригувати власні цінові пропозиції на ринку [10].

Доступ до вказаних сервісів здійснюється за допомогою вебсайтів, які представлені у вигляді електронного майданчика з даними про товари та послуги з фотокартками, їх детальним описом, технічними характеристиками та відповідними цінами збуту. В демоверсіях, на сайтах розміщуються лише переліки товарів з цінами і посиланнями на сайти постачальників або торгових майданчиків [10].

До переваг подібних рішень можуть бути віднесені наступні функції [10]:

- отримання актуальної і достовірної інформації щодо цінових пропозицій конкурента у будь-який час;
- зменшення витрат часу на збір, обробку та систематизування даних;
- можливість візуалізації зібраних даних для виявлення відхилень у діяльності конкурента;

- можливість створення та введення звітів для проведення нарад і роботи компанії офлайн.
- надання доступу до особистого кабінету користувача для кастомного налаштування послуг сервісу та відстежування динаміки змін цінових пропозицій бізнес-партнерів і конкурентів;
- підключення SaaS рішення до власної системи “customer relationship management” (CRM) та обміну даними на основі “application programming interface” (API);
- забезпечення безперебійної роботи системи для компаній, які не можуть дозволити собі розгортання вказаних рішень у середині компанії.

З іншого боку вказані системи мають недоліки, які несуть проблеми для застосування рішень і безпеки даних компаній-користувачів:

- SaaS-рішення не підтримують кастомізацію відповідно до особливостей структури і потреб компаній (шаблонне рішення);
- можливість витоку даних зібраних системою та конфіденційної інформації компаній-користувачів (регламентується публічним договором між постачальником і користувачем сервісу);
- можливість втрати доступу до зібраних сервісом даних через відсутність зв'язку з мережею Інтернет і виходом з ладу обладнання з боку постачальника послуг.

1.2 Методи і засоби вирішення задачі

Узагальнений алгоритм для відстеження динаміки ціноутворення на ринку включає наступні кроки.

Крок 1. Пошук всіх конкурентів, що надають аналогічні товари чи послуги в Інтернет.

Крок 2. Вибір тих, хто є конкурентоспроможним своїми на онлайн-ринку.

Крок 3. Процес відстеження цін.

Крок 4. Формування стратегії про ціноутворення, згідно з результатами моніторингу.

Спостереження цін інтернет-магазинів можна провести декількома методами: ручним і автоматизованим.

Ручний метод має високий рівень людського чинника. Такий спосіб має як плюси (розширення погляду на нові закономірності), так і мінуси (велика ймовірність помилки, вартість вище). Для проведення ручного моніторингу знадобиться участь окремого працівника. Час проведення розвідки – це вартість послуги, яку необхідно буде сплатити. При цьому варто враховувати, що працівник буде працювати з великим обсягом даних. Відтак час витрачений на збір та обробку інформації може зайняти декілька днів або тижні. Інший спосіб - прайс-агрегатори, які є досить корисним інструментом підготовки оглядової статті для блогу та ілюстрації її даними ринку. Більш складним методом є синтаксичний аналіз або парсинг. В ідеалі, звісно, використовувати спеціально написану програму, але вартість її створення може бути невиправдано високою [11].

Автоматизована система моніторингу цін – метод відстеження з урахуванням людського чинника. За допомогою такого методу можливо промоніторити ціни всіх конкурентів певного інтернет-магазину, відстежувати тих, що вказують ціни нижче рекомендованих. Набагато ефективніше використовувати сервіс моніторингу і отримувати інформацію про цінові зміни щодня або кілька разів протягом дня. Також подібний спосіб дозволяє заощадити значні ресурси [10].

1.3 Програмні засоби автоматизації

Для автоматизації моніторингу цінових пропозицій у мережі Інтернет створено велику кількість онлайн-сервісів. Вони надають послуги з пошуку та обробки даних, формування автоматичних звітів для проведення порівняльного аналізу цінових пропозицій, що в свою чергу дозволяє маркетологам стежити за

конкурентами і оперативно реагувати на зміни ринку.

1.3.1 Сервіс “Price Control”

Сервіс “Price control” [11] дозволяє отримувати інформацію щодо діяльності конкурентів, на основі якої користувачі можуть проводити аналіз ринку і створювати власні стратегії ціноутворення (рис. 1.1).

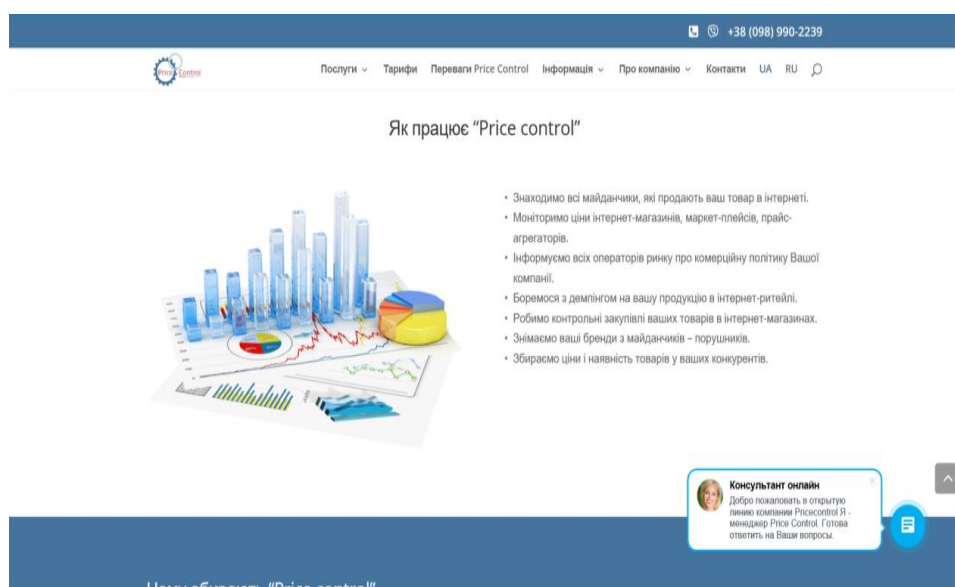


Рисунок 1.1 – Price Control [11]

Відповідно до результатів моніторингу, користувач отримує можливість вивчити динаміку ринкових процесів, які впливають на діяльність компанії і оперативно коригувати її маркетингову стратегію. З метою досягнення вказаного результату, команда “Price control” пропонує користувачам наступні послуги:

- пошук всіх майданчиків, які продають подібні товари у мережі Інтернет;
- моніторинг цінових пропозицій розміщених в Інтернет-магазинах, маркетплейсах, прайс-агрегаторах;
- зворотний зв'язок з операторами ринку для уточнення комерційної політики компанії користувача;

- здійснення контрольних закупівель товарів користувача в Інтернет-магазинах;
- збір у мережі Інтернет і накопичення інформації щодо наявного у конкурентів товару і відповідних цінових пропозицій;
- захист та вирішення проблем щодо прав інтелектуальної власності в інтернет-магазинах;
- зняття товарів з продажу на недобросовісних ресурсах (при необхідності);
- участь в судових справах щодо захисту торгових марок.

Таким чином, сервіс “Price control” допомагає збирати та відстежувати інформацію щодо дій конкурентів, підвищити якість моніторингу даних. Кожний місяць, відповідно до обраного користувачем пакету послуг, система проводить повну актуалізацію усіх Інтернет-магазинів за для виправлення помилок і помилок під час збору даних через регулярні зміни на сайтах Інтернет-магазинів. З метою розуміння повної картини цін на ринку сервіс збирає інформацію щодо цінових пропозицій як на сайтах інтернет-магазинів, так і на прайс-агрегаторах, маркетплейсах і рекламних майданчиках.

1.3.2 Сервіс “Z-Price”

Рішення українського функціонального автоматичного онлайн-сервісу “Z-Price” [11] дозволяють виконувати збір інформації на вебсайтах Інтернет-магазинів та аналізувати їх цінові пропозиції за для контролю цінової політики користувача (рис. 1.2).

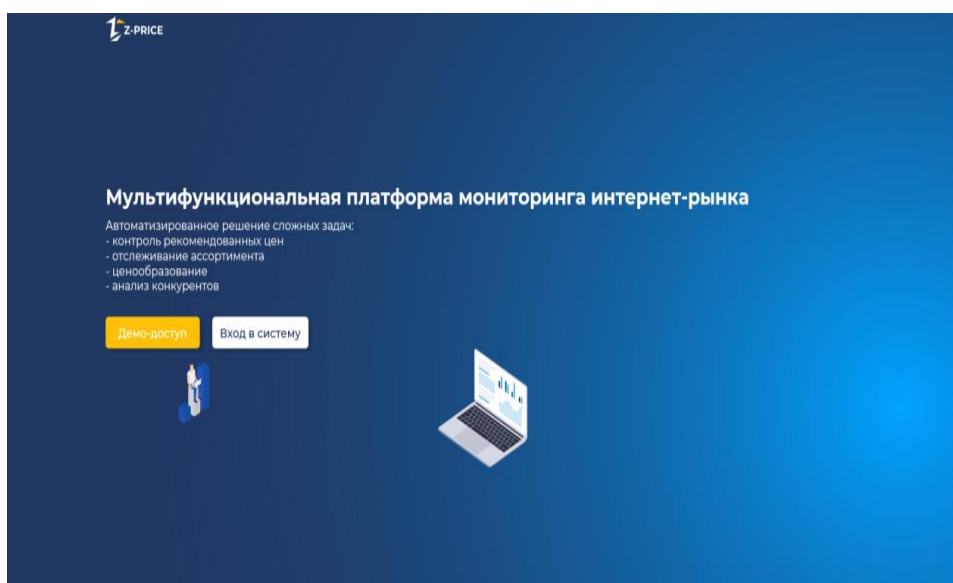


Рисунок 1.2 – Z-Price [11]

У залежності від обраного бренду або Інтернет-магазину, вказаний сервіс надає різноманітні можливості з урахуванням потреб продуктових компаній. Наприклад, дозволяє відстежувати цінові пропозиції лише маркетплейсів і не дозволяє працювати з Інтернет-магазинами.

Для брендів надаються стандартні послуги зі збору цінових пропозицій, порівняння з рекомендованою роздрібною ціною, розсилки листів порушникам, завантаження звітів із рекомендованою роздрібною ціною (РРЦ), отримання інформації щодо відхилень цінових пропозицій товарів і послуг кожного Інтернет-магазину та збереження історії її змін.

Технічні фахівці компанії “Z-Price” ретельно стежить за пропонованими акціями, знижками та подарунками. У разі появи відхилень від РРЦ менеджери сервісу оперативно повідомляють про появу магазинів-порушників. Таймфрейм оновлення даних сервісом здійснюється кожні 2 години, що дозволяє виявляти постачальників активних у нічний час.

Окремо, у якості бонусу, компанія “Z-Price” надає можливість доступу до особистого кабінету, його інтеграції з власним сайтом, створювати стандартні та кастомізовані звіти, завантажувати їх у різних форматах. Однак, вказаний сервіс має той самий недолік, що й інші компанії даного класу – необхідність додаткової оплати за парсинг кожного додаткового конкурента. З

метою попередження втручання у роботу системи [11].

Таким чином, Z-Price функціонує за рахунок багатопотокової системи оновлення даних, що дозволяє їй надавати лише актуальну інформацію та оновлювати дані 2 рази на день. Отримання більшої кількості функцій можливе лише у разі надання інформації щодо походження користувача (представник бренду або інтернет-магазину), країни користувача.

1.3.3 Сервіс “PriceCop”

На відміну від попереднього сервісу, продукт української компанії “PriceCop” [11] надає компаніям-виробникам можливість відстежувати демпінгових продавців (рис. 1.3).



Рисунок 1.3 – PriceCop [11]

Даний сервіс регулярно надає актуальну інформацію щодо цін на товари виробника на місцевому агрегаторі цін Hotline. Також можна відстежувати й обмежену кількість конкретних продавців. Парсинг цін із сайту кожного продавця налаштовується індивідуально (близько 3 днів).

До переваг вказаного рішення можна віднести наступне:

- можливість створення великої кількості звітів (щоденно);

- можливість керування номенклатурою компаній;
- можливість збереження історії змін цінових пропозицій;
- можливість представлення інформації у графічному вигляді;
- можливість з власною системою з використанням API;
- наявність механізму автоматичної відправки сповіщень щодо демпінгу.

До недоліків даного рішення можна віднести наступне:

- менеджери не втрачають можливості продати послугу та інтернет-магазинам,
- проєкт заточений виключно під бренди, що призводить до появи невдоволених користувачів;
- наявність мінімальних лімітів за кількістю позицій доступних для моніторингу;
- наявність великої кількості неінформативних звітів;
- відсутність інструментів для переоцінки товарів і послуг.
- проведення презентацій рішення з використанням акаунтів реальних клієнтів.

Таким чином, сервіс “PriceCop” є платформою, що дозволяє відстежувати вартість товару чи послуги як в інтернет-магазинах, так і в прайс-агрегаторах. Крім цього, дозволяє дізнаватися про проведення акцій у конкурентів та відстежувати наявність або відсутність конкретних виробів. Крім цього, автоматично визначаються нові товари у магазинів, що відстежуються.

1.3.4 Сервіс “Competera”

Програмний продукт компанії “Competera” [11] — це платформа, заснована на штучному інтелекті, яка допомагає онлайн та офлайн ритейлерам досягати очікуваних фінансових показників та збільшувати лояльність покупців за рахунок правильних цін (рис. 1.4).

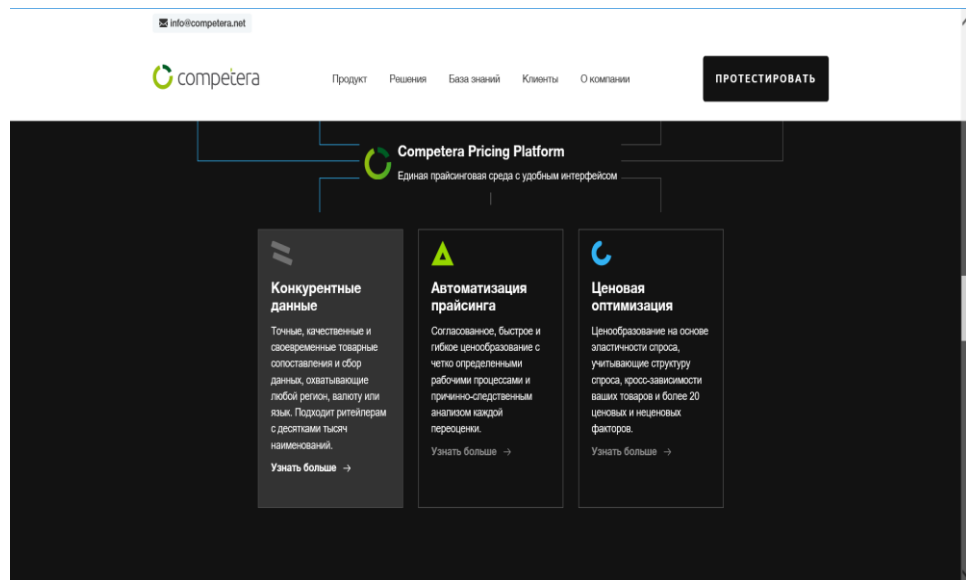


Рисунок 1.4 – Competera [11]

Вказаний сервіс буде корисним для великих компаній з окремим бюджетом на інформаційні технології (ІТ), які мають потребу в користуванні послугами особистого менеджера [3].

Компанія “Competera” не пропонує універсальне рішення, яке підійде усім клієнтам, однак їх продукт може бути адаптований під кожного клієнта. Це дозволяє розробникам співпрацювати як з великими клієнтами (на національному рівні), так і з клієнтами, які можуть придбати будь-який пакет послуг. До переваг даного сервісу можна віднести:

- функцію збору цінових пропозицій, на сайтах вказаних користувачем;
- функцію конкурентної аналітики;
- функцію ціноутворення на товари основного попиту та штучного інтелекту.

До недоліків вказаного сервісу можна віднести:

- високі ціни (при першому контакті менеджер уточнює, який саме пакет послуг може дозволити собі користувач);
- жорсткі вимоги до наданої клієнтом інформації (необхідно надати дані щодо продажів за кілька останніх періодів, що є комерційною таємницею).

Враховуючи вищевказане, “Competera” представляє хмарний сервіс

ціноутворення для офлайн і онлайн продавців. Його спеціалісти пропонують звичайну послугу з відстеження цін, а також більш серйозний функціонал, який може бути використаний для прийняття бізнес рішень.

1.4 Постановка задачі

Використовуючи інформацію з п.1.1 можна зробити висновок, що однією з ключових проблем ведення ефективної конкуренції на ринку є моніторинг цінових пропозицій. Однак бізнес висуває досить жорсткі вимоги до обсягу інформації, точності та швидкості її обробки для прийняття оперативних рішень. Основною проблемою, при цьому, є значний вплив людського фактора на час підготовки і аналізу даних, а також їх якість.

У п.1.2 вказано, що одним з практичних рішень вказаних проблем є автоматизація бізнес-процесів у частині моніторингу цінових пропозицій і в п.1.3 наведено перелік вживаних програмних рішень, які дозволяють виконувати вказані завдання.

Згідно з п.1.1 і п.1.3 встановлено, що значний вплив, при прийнятті рішень стосовно автоматизації бізнес-процесу, відіграє кількість інформації якою володіє замовник, розмір компанії, бюджет який компанія може витратити на вказані інструменти та ризики від втрати/розголошення даних під час користування ними. Від так встановлено, що при веденні бізнесу, більш ефективним і безпечним для користувачів є використання локальних (власних) рішень, які реалізують функції SaaS-систем.

Використовуючи дані п.1.1 і п.1.2, отримано дані щодо переліку функцій, які є рекомендованими і надаються існуючими SaaS-рішенням, виконано їх порівняння з метою визначити функції віднесені до базового функціонала SaaS-рішень (табл. 1.1).

Таблиця 1.1 – Порівняння функцій оглянутих програмних рішень

Прикладні завдання	Price Control	Z-Price	PriceCop	Competera
1	2	3	4	5
Пошук Інтернет-майданчиків	+			
Зворотній зв'язок з операторами	+			
Здійснення контрольних закупівель	+			
Збір інформації про товари конкурентів	+	+	+	+
Накопичення інформації про товари конкурентів	+	+	+	
Захист прав інтелектуальної власності магазину	+			
Зняття товарів компанії з продажу	+			
Допомога під час участі в судових справах	+			
Створення звітів		+	+	
Управління номенклатурою			+	
Можливість побудови графіків			+	
API інтеграція		+	+	
Аналітика		+		+
Ціноутворення на основі попиту		+		+
Ціноутворення на основі штучного інтелекту				+
Оповіщення про підозрілих продавців		+		

Згідно з даними табл. 1.1, виходить що до базового функціонала SaaS-рішень можна віднести наступні функції:

- збір інформації про товари і ціни конкурентів;
- накопичення інформації про товари і ціни конкурентів.

Хоч інформація про можливості “зворотного зв'язку з операторами систем” і “створення звітів” не представлена у переліку функцій усіх оглянутих рішень, однак на думку автора вони також будуть важливими для кінцевих користувачів. Тому мінімальною вимогою до розроблюваної системи є реалізація вище вказаних функцій.

1.5 Постановка завдань роботи

Для досягнення мети, поставленої на початку роботи, та згідно з вимогами п.1.4, були сформовані наступні завдання:

- створити вимоги до системи автоматизації пошуку, обробки цінових пропозицій у мережі Інтернет;
- виконати проєктування систему спостереження за ціноутворенням товарів і послуг у мережі Інтернет;
- виконати проєктування бази даних для зберігання інформації;
- розробити систему моніторингу цінових пропозицій на товари та послуги у мережі Інтернеті.

1.6 Висновки за розділом 1

Таким чином, згідно з завданням першого розділу виконано обґрунтування ефективності аналізу цінових пропозицій, переваги та недоліки використання SaaS-рішень, способи й методи його проведення моніторингу цінових пропозицій, огляд онлайн-сервісів автоматизації моніторингу цінових пропозицій, виконано порівняння та аналіз функцій існуючих SaaS-рішень, що дозволило сформулювати перелік завдань на розробку власного проєкту.

2. РОЗРОБКА АРХІТЕКТУРИ ВЕБСАЙТУ

З метою урахування та розробки усіх необхідних елементів вебсайту, на етапі його проєктування необхідно виконати наступні завдання: створити вимоги до функціонала та дизайну сторінок, мов для Frontend (FE) і Backend (BE) реалізацій, структури і серверної частини проєкту, а також обрати інструментарій для розробки проєкту.

2.1 Вимоги до функціоналу

З метою задоволення інтересів користувачів вебсайту, відповідно до п.п.1.4 і п.п.1.5, необхідно розробити функціонал, який дозволяє:

- збирати інформацію про цінові пропозиції конкурентів;
- виконувати обробку, накопичення та виведення на вебсайті;
- керувати процесом збору та відображення даних на вебсайті.

Для збору інформації про цінові пропозиції конкурентів необхідно розробити рішення (парсери), які дозволять виконувати наступні завдання:

- перевіряти доступність сайтів-конкурентів та торгових платформ;
- здійснювати доступ на сайти-конкурентів або торгові платформи, які містять необхідну інформацію;
- вивантажувати дані шляхом виконання API запитів або парсингу документів формату “HyperText Markup Language” (HTML);
- перетворювати дані у вигляд необхідний для зберігання у базі даних;
- забезпечити отримання даних з заданою періодичністю.

Для накопичення, зберігання та подальшого використання необхідно виконати перетворення зібраних даних відповідно до вимог бази даних проєкту, виконавши наступні завдання:

- визначення наявності обов’язкової інформації для зберігання у базі даних;

- додавання нових записів до бази даних;
- оновлення існуючих записів у базі даних;
- архівування застарілих даних;
- вибір активних записів з бази даних для відображення на дашборді (для користувачів);
- вибір усіх записів з бази даних для відображення на дашборді (для адміністратора).

З метою управління процесом збору та відображення даних на вебсайті, необхідно реалізувати наступний функціонал:

- можливість запуску парсерів з консолі;
- можливість запуску парсерів з вебформи сайту (за можливості);
- можливість оновлення налаштувань парсерів у ручному та напіваавтоматичному режимах;
- вебсервер обробки запитів від користувачів та парсерів;
- сервер бази даних для зберігання та доступу до інформації;
- механізми відправки повідомлень електронною поштою (у адміністративних цілях);
- реалізувати механізми розмежування доступу до даних і налаштувань на основі ролей, паролів і ключів доступу.

Перелік вищевказаних функцій є узагальненим і може бути переглянутий у ході розробки проєкту.

2.2 Вимоги до дизайну сторінок

Реалізація функціонала, зазначеному у п.п. 2.1, потребує розробки відповідних вебінтерфейсів для використання кінцевими користувачами. Для цього необхідно розробити сторінки сайту відповідно до макетів на рис. 2.1-2.3.

Wireframe (a) shows a login form with two input fields: "Поле 'Username'" and "Поле 'Password'". Below these fields are two buttons: "Sign up" and "Sign in".

а

 Wireframe (б) shows a registration form with five input fields: "Поле 'Firstname'", "Поле 'Lastname'", "Поле 'Username'", "Поле 'E-mail'", and "Поле 'Password'". Below these fields are two buttons: "Forgot password" and "Sign up".

б

Рисунок 2.1 – Макети сторінок для доступу на сайт:
авторизація (а) і реєстрація (б)

 Wireframe (а) shows a form for requesting a password reset with one input field: "Поле 'Your Email Address'". Below this field are two buttons: "Back to Log in" and "Reset".

а

 Wireframe (б) shows a form for confirming a password reset with two input fields: "Поле 'Password'" and "Поле 'Confirmation Password'". Below these fields are two buttons: "To Log in" and "Reset".

б

Рисунок 2.2 – Макети сторінок для відновлення доступу на сайт:
створення запиту (а) і встановлення нового пароля (б)

 Wireframe (а) shows a dashboard layout. At the top is a navigation bar with buttons: "Логотип", "Частота оновлення", "К-ть пропозицій", "Репорти", "Тип графіку", and "Меню". Below the navigation bar is a large area labeled "Графік / Таблиця".

а

 Wireframe (б) shows a settings panel layout. At the top is a navigation bar with buttons: "Логотип", "Годинник", "Повідомлення", and "Меню". Below the navigation bar is a sidebar with a list of settings: "Application", "Accounts", "Sessions", "Tables Management", "Charts Management", "Spiders Management", and "Feedbacks Management". To the right of the sidebar is a large area labeled "Таблиця".

б

Рисунок 2.3 – Макети основних сторінок проєкту:
дашборд (а) і панель налаштувань (б)

Відповідно до макета (рис. 2.1а), доступ до вебсайту обмежено за допомогою імені користувача (username) та пароля (password). У випадку, якщо користувач не має атрибутів доступу або втратив їх, то йому доступна можливість створення нового облікового запису.

Згідно з макетом (рис. 2.1б), для створення облікового запису користувач повинен надати мінімальний набір даних про себе: реальне ім'я (firstname) та прізвище (lastname), ім'я у системі (username), адреса електронної поштової скрині (email), пароль доступу (password) та підтвердження пароля (confirmation password) для доступу до системи. У випадку, якщо користувач володіє інформацією про свій обліковий запис, то він може ініціювати процедуру відновлення пароля.

Відповідно до макета (рис. 2.2), відновлення пароля складатиметься з 2 етапів: введення електронної адреси користувача (рис. 2.2а), введення, підтвердження та застосування нового пароля (рис. 2.2б).

Макети (рис. 2.1 і 2.2) повинні бути пов'язані між собою, для виключення ситуації коли один з них недоступний з будь-якою частини вебсайту.

Макет (рис. 2.3а) реалізує головну сторінку вебсайту, на якій будуть відображені дані щодо цінових пропозицій компаній-конкурентів у табличному та графічному режимах. Натомість макет (рис. 2.3б) реалізує сторінку з розділами налаштувань вебсайту, зміст і вигляд центральної частини яких буде залежати виключно від обраного користувачем розділу.

В шапці дашборду, головними атрибутами є логотип і кнопка меню користувача. Наявність інших елементів: годинник, повідомлення та випадваючі переліки (тип графіку, репорти, таймфрейми, кількість записів), залежить виключно від налаштувань дашборду і привілеїв доступу користувача.

У підвалі сайту повинен знаходитися блок із посиланням на компанію і дизайнера сторінок.

Враховуючи перелік необхідного функціоналу та вищевказані макети, карта вебсайту буде виглядати наступним чином (рис. 2.4).

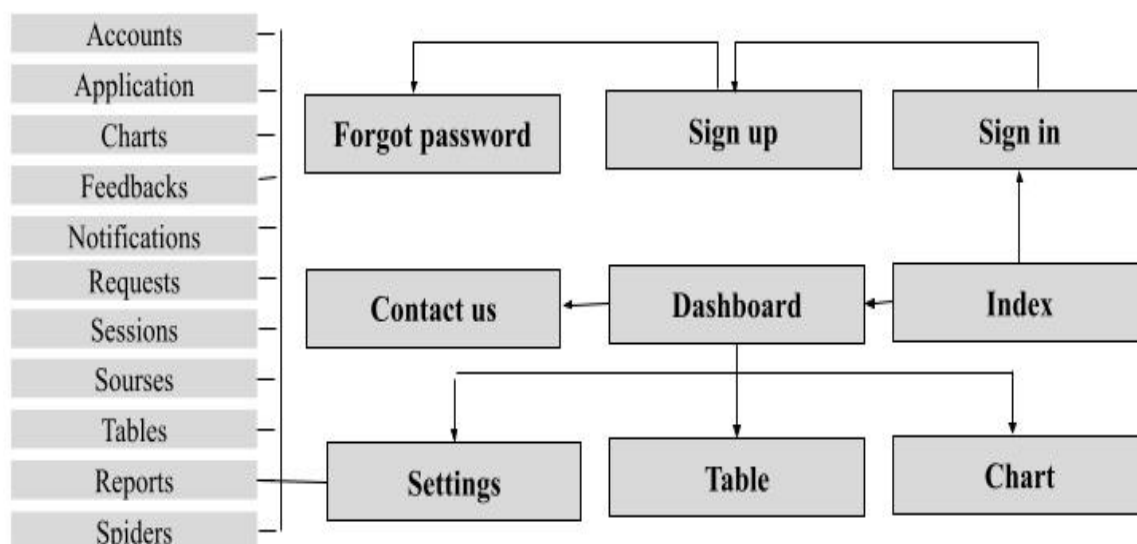


Рисунок 2.4 – Карта розроблюваного вебсайту
(з можливим переліком розділів)

Кожний з розділів має своє функціональне призначення:

- index – основна сторінка проєкту (шаблон);
- sign in – сторінка входу на сайт;
- sign up – сторінка реєстрації на сайті;
- forgot password – сторінка відновлення пароля;
- new password – сторінка встановлення нового пароля;
- dashboard – сторінка дашборду (включає розділи – table, chart, contact us);
- settings – сторінка налаштувань сайту (включає розділи – accounts, application, charts, feedbacks, notifications, requests, sessions, sources, tables, reports, spiders).

2.3 Вибір інструментарію для розробки проєкту

Для швидкого створення проєкту та реалізації усіх заявлених функцій необхідно обрати найбільш зручні інструменти розробки: обрати мову програмування для BE, фреймворку на базі якого буде реалізовано проєкт, інструменти для створення верстки макетів, визначення структури проєкту.

2.3.1 Вибір мови програмування на ВЕ

При підготовці до розробки було виконано огляд мов програмування, які можуть бути використані для створення основи проєкту (табл. 2.1).

Таблиця 2.1 – Порівняння мов програмування

Мова	Python	Java	PHP	Ruby
1	2	3	4	5
Парадигма	Високорівневий, інтерпретований, об'єктно-орієнтований, імперативний, процесуальний, функціональний	На основі класу, інтерпретований, одночасний, об'єктно-орієнтований	Мова сценаріїв, двостороння, інтерпретована	Високорівневий, інтерпретований, об'єктно-орієнтований
Синтаксис	Просто	Комплексний	Складніший, ніж Python (подібний до Java, C і Perl)	Простіший для розуміння та гнучкіший, ніж Java та PHP
Тип	Динамічний	Статичний	Динамічний	Динамічний
Застосування	Веброзробка, штучний інтелект, машинне навчання, Інтернет речей, сценарії, розробка ігор	Розробка веб та мобільних додатків	Веброзробка	Веброзробка

Продовження таблиці 2.1

1	2	3	4	5
Фреймворки	Django, Flask, Pyramid, TurboGears, Web2py	Spring, BladE, JSF, Struts	Laravel, Symiony, Codelgniter, Cake PHP, Zend	Ruby on Rails, Сінатра, Ханамі, Ненсі, Куба
Компанії	Google, Quora, Reddlt, YouTube, Spotiiy	Google, Uber, Lyft, eBay, LinkedIn	Wikipedla, Word Press, Yahoo, Flickr, Tumblr, Udemy	Airbnb. GitHub, Twitter, Apple, Shopify, Groupon, LinkedIn

За результати порівняння прийнято рішення щодо використання мови “Python”, як основи для даного проєкту. Він є багатоцільовою мовою програмування, яка підтримує декілька платформ і дозволяє створювати читабельний програмний код. Відносний лаконізм мови “Python” дозволяє створювати додатки з меншою кількістю коду у порівнянні з іншими мовами програмування. При наступних перевагах: низький поріг для входу, розвинена спільнота програмістів, зручна документація, суворе дотримання стандартів написання коду (наприклад, PEP8), велика кількість готових бібліотек, його єдиним недоліком є відносно низька швидкість виконання коду.

2.3.2 Вибір фреймворку для створення BE

Враховуючи обрану мову програмування на стороні BE, для реалізації даного проєкту прийнято рішення про використання популярного фреймворку для побудови мікросервісів “Flask”, який надає корисні інструменти та функції для полегшення процесу створення вебзастосунків з використанням мови Python. Він досить гнучкий і дозволяє створювати додатки досить швидко [16].

При підготовці до розробки були визначені основні методи і модулі розширення фреймворку, які будуть використані для побудови сайту [8]:

- `flash()` – метод, який дозволяє зберігати та відображати повідомлення про результати роботи тих чи інших процедур, ендпоінтів;
- `render_template()` – метод, який дозволяє завантажити опис HTML сторінки, передати її браузеру для відображення із заданим форматуванням даних і стилями. Його використання тісно пов'язане з використанням шаблонизатора Jinja2;
- `request` – об'єкт, який дозволяє вебзастосункам реагувати на появу даних, здійснювати обробку запитів до BE;
- `session` – об'єкт, який реалізує механізм роботи з сесіями на програмному рівні;
- `redirect()` – метод, який виконує перенаправлення запитів, які надходять до BE;
- `flask_mail` – модуль-розширення фреймворку “Flask” для роботи з електронною поштою без використання сторонніх модулів.

2.3.3 Вибір інструментів для створення верстки макетів

Для реалізації макетів вказаних у п.п.2.1, у даному проєкті використані наступні мови програмування, інструменти та бібліотеки: HTML, CSS, JS, Bootstrap, JQuery, Plotly.

Мова гіпертекстової розмітки (HTML) призначена для опису структури вебсторінок, в якій можуть бути використані інструкції з форматування даних, розбиття даних на розділи, що дозволяє веббраузерам відображати дані на екрані з вказаними стилями [17].

У сучасних вебсайтах, можливість використання HTML у чистому вигляді обмежено. Для компенсації даного недоліку прийнято використовувати Cascading Style Sheets(CSS), які реалізують гнучкий механізм стильового оформлення сторінок [17]. Найчастіше CSS використовується для документів,

котрі розмічені мовою HTML, XHTML та XML [18].

Для гнучкого управління версткою проєкту, прийнято рішення щодо використання фреймворку “Bootstrap”, який надає набір інструментів створення макетів вебсторінок на основі HTML і CSS.

Забезпечення інтерактивності є одним з критеріїв зручного вебсайту [12], тому для надання вебсторінкам більшої динамічності буде використано мову програмування JavaScript, яка дозволяє забезпечити інтерактивність взаємодії між користувачем і веббраузером [19], при відправці та отриманні/оновленні даних, обробки запитів користувачів без перезавантаження вебсторінок.

З метою спрощення окремих сценаріїв створених на JavaScript, у проєкті буде використано фреймворк jQuery, який допомагає спростити написання великих фрагментів коду. До його складу входить велика кількість базових функцій для надання вебсторінкам інтерактивності [21].

Для побудови графіків і їх подальшого відображення на дашборді, прийнято рішення щодо використання бібліотеки “Plotly”, яка дозволяє є досить популярною у галузі Data Science. Її основною перевагою перед подібними проєктами (seaborn і matplotlib) є зручність побудови складних інтерактивних візуалізацій, які роблять результат роботи аналітика доступнішим для кінцевого користувача [22].

2.4 Структура проєкту

За для підтримки проєкту в актуально стані, читабельність коду та можливості його інтеграції із зовнішніми сервісами (наприклад, вебхостингами), в проєкті використано патерн проєктування Model-View-Controller (MVC), який розшифровується як модель-уявлення-контролер.

MVC є таким способом організації коду, який передбачає виділення блоків відповідальних за вирішення різних завдань:

- модель (model) є компонентом, який відповідає за роботу з даними і

визначає структуру додатка;

- подання (view) є компонентом, який відповідає за відображення даних користувачам, визначає зовнішній вигляд додатку та надає доступ до елементів для взаємодії з ним;
- контролер (controller) є компонентом, який відповідає за обробку запитів при взаємодії користувачів з додатком, обміну даними між моделлю (model) та поданням (view).

Враховуючи вимоги патерну MVC, інформацію щодо функціонала, макетів та обраного BE фреймворку, структура проєкту набула наступного вигляду (рис. 2.5).

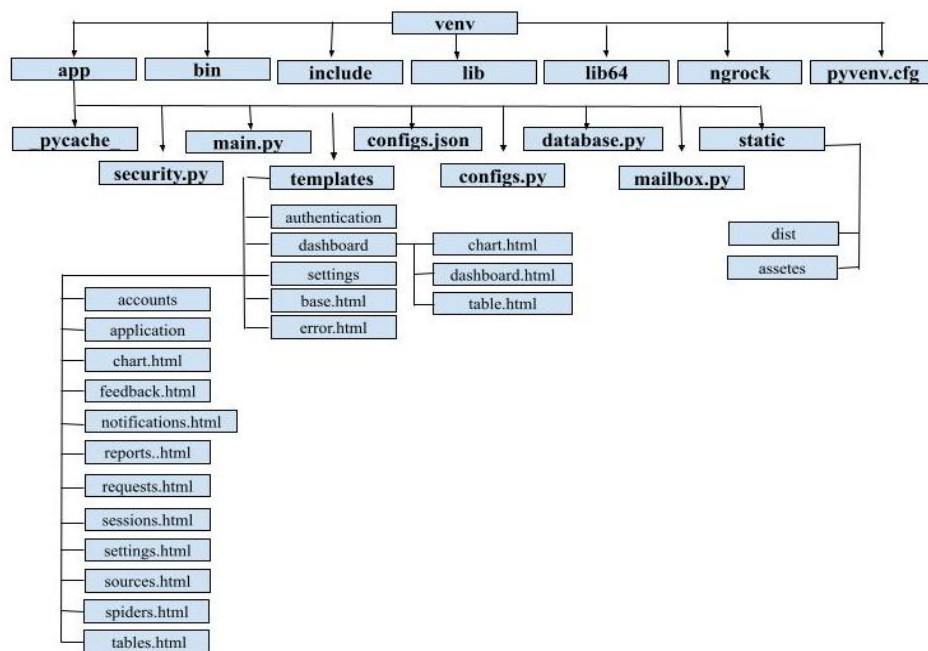


Рисунок 2.5 – Структурована схема проєкту

Згідно з рис. 2.5, проєкт не вийде розділити на окремі модулі, оскільки методи контролеру та подання перетинаються між собою. Таким чином у розрізі патерну MVC, базові модулі включають наступні елементи:

- модель складається з файлів security.py, database.py, mailbox.py, які містять класи і методи перетворення даних;
- подання включає усі файли директорій templates і static, які

впливають на зовнішній вигляд вебсайту, а також файл `main.py` для виклику вказаних файлів;

– контролер включає методи обробки запитів користувачів і фізично знаходиться у файлі `main.py` (поєднаний з методами модуля подання).

2.5 Вибір середовища розробки

Для розробки програмного коду використано редактор коду “Visual Studio Code”, який призначено для створення, редагування та запуску створених додатків, у тому числі й вебдодатків (вебсайтів) та хмарних сервісів. Вказаний редактор коду розповсюджується за безкоштовною ліцензією для платформ Windows, Linux, OS X, має вбудовані інструменти для роботи з Git, надає можливість роботи з мовами програмування/технологіями JavaScript, C++, C#, TypeScript, jade, PHP, Python, XML, Batch, F#, DockerFile, Coffee Script, Java, HandleBars, R, Objective-C, PowerShell, Luna, Visual Basic, Markdown, JSON, HTML, CSS, LESS і SASS, Haxe (рис. 2.6).

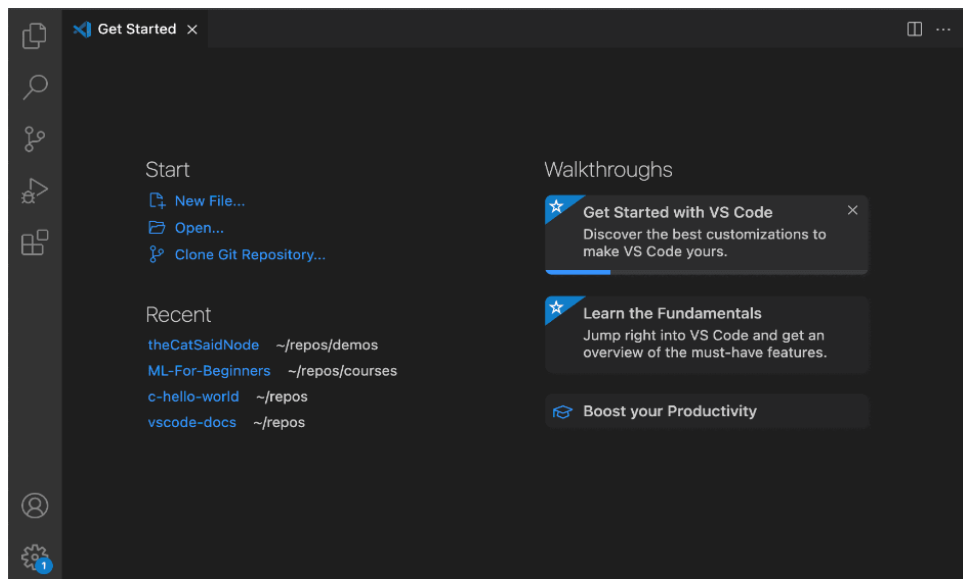


Рисунок 2.6 – Стартове вікно редактор коду VS Code [14]

2.6 Висновки за розділом 2

Відповідно до завдання до розділу, наведено перелік базового функціонала, який буде реалізовано під час розробки сайту, перелік обов’язкових макетів для розробки вебсторінок, карту сайту з урахуванням усіх необхідних сторінок для користування сайтом, обрано мову, фреймворк і відповідні інструменти для реалізації проєкту.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ ВЕБСАЙТУ

Виходячи з вимог до проєкту у розділі 2 і та задач у п.п.1.6, під час роботи над проєктом були розроблені наступні програмні рішення: модулі управління обліковими записами, сесіями, даними, безпеки даних і підтримки користувачів. Окремо розроблено структуру бази даних для зберігання системної, презентаційної та бізнес інформації.

3.1 Модуль управління обліковими записами

Модуль управління обліковими записами є базовим механізмом організації доступу до вебсайту. Його основним завданням є забезпечення можливості доступу до системи всередині компанії і попередження несанкціонованих підключень сторонніх осіб з мережі Інтернет. Для виконання вказаного завдання були розроблені наступні процедури: створення запису, вхід до системи і вихід із системи, відновлення пароля, перегляд облікових записів.

3.1.1 Створення облікових записів

Процедура створення облікового запису призначена для внесення інформації про користувачів до бази даних системи.

На стороні FE, вона потребує виконання наступних кроків.

- Крок 1. Ввести реальне ім'я користувача системи (firstname).
- Крок 2. Ввести реальне прізвище користувача системи (lastname).
- Крок 3. Ввести ім'я користувача у системі (username).
- Крок 4. Ввести електронну адресу користувача (email address).
- Крок 5. Ввести пароль для доступу користувача у систему (password).
- Крок 6. Повторно ввести пароль для підтвердження його валідності (confirmation password).

Крок 7. Натиснути кнопку “Sign Up” для завершення процедури реєстрації.

Після введення обов’язкових даних та їх відправки на сервер (BE), модуль звертається до ендпоінту “POST /authentication/signup/send” для валідації, обробки та збереження вхідних даних користувача. Валідація даних виконується на основі наступних регулярних виразів:

- “^[A-Za-z]{2,25}\$” згідно з яким, при введенні імені (firstname) та прізвищі (lastname) повинні використовуватися символи латинського алфавіту у нижньому або верхньому регістрі, їх мінімальна та максимальна довжина повинні становити від 2 до 25 символів (наприклад, “John Doe”);

- “^[A-Za-z0-9]{5,25}\$” згідно з яким, при створенні “username” важливими умовами є використання символів латинського алфавіту у нижньому або верхньому регістрі, їх мінімальний і максимальний розміри повинні становити бути від 5 до 25 символів;

- “^[A-Za-z0-9]{8,}\$” згідно з яким при введенні пароля (password) та його підтвердження (confirmation password) повинні використовуватися символи латинського алфавіту у нижньому або верхньому регістрі, їх мінімальна та максимальна довжина повинні становити не менше 8 символів (наприклад, “4lt50lvI0A”).

Для валідації електронної адреси користувача (email address) замість регулярного виразу використано програмний модуль “email_validator.py”, в якому реалізовано відповідний метод перевірки електронної адреси на валідність без необхідності створення додаткових сценаріїв валідації.

Реальні ім’я (firstname) та прізвище (lastname) користувачів необхідні для адміністративних завдань і ідентифікації користувачів у межах компанії. Натомість ім’я користувача призначене виключно для ідентифікації користувача всередині системи. Введена під час реєстрації електронна адреса призначена виключно для відновлення доступу користувача до облікового запису. Умови обрані для валідації паролів відповідають мінімальним вимогам безпеки, згідно з загально прийнятими стандартами парольного захисту даних.

Доступ до усіх вказаних даних має виключно адміністратор системи.

У разі виникнення помилок під час створення користувача, в системі повинні бути передбачені кастомні повідомлення щодо причин їх виникнення.

Серед них передбачено обробку наступних випадків:

- не валідне реальне ім'я користувача (firstname);
- не валідне реальне прізвище користувача (lastname);
- не валідне ім'я користувача (username);
- не валідний пароль (password або confirmation password);
- паролі (password та confirmation password) відрізняються один від одного;
- не валідна електронна адреса (email address);
- ім'я (username) та/або електронна адреса (email address) користувача вже використовуються у системі;
- інші помилки субмодуля пов'язані з валідацією даних і їх додавання до бази даних.

3.1.2 Вхід до системи

Процедура входу в обліковий запис призначена для фіксації факту доступу до системи, і організації доступу до розділів вебсайту.

На стороні FE, вона потребує виконання наступних кроків.

Крок 1. Ввести ім'я користувача у системі (username).

Крок 2. Ввести пароль для доступу користувача у систему (password).

Крок 3. Натиснути кнопку “Sign In” для запуску процедури автентифікації користувача.

Після введення даних користувачем і їх подальшої відправки на сервер (BE), модуль звертається до ендпоінту “POST /authentication/login/send” для створення нової сесії. Процедура валідації вхідних даних здійснюється за допомогою виразів з п.п. 3.1.1. У разі якщо користувача вже відвідував сайт, не розірвав сесію та час її життя не вичерпано, користувачу надається токен сесії з

попередньою перевіркою атрибутів доступу за ім'ям і паролем користувача. Після підтвердження привілеїв доступу до системи, користувач повинен бути перенесений сторінку “Dashboard”.

При виникненні помилок під час входу в обліковий запис необхідно передбачити кастомні повідомлення щодо причин виникнення помилок. Серед них передбачено обробку наступних випадків:

- не валідне ім'я користувача (username);
- не валідний пароль (password);
- користувача з вказаним ім'ям (username) не знайдено;
- користувача з вказаними атрибутами доступу (username і password) не знайдено;
- активного користувача з вказаними атрибутами доступу (username і password) не знайдено;
- інші помилки субмодуля пов'язані з валідацією даних і їх додавання до бази даних.

3.1.3 Вихід з системи

Процедура виходу з системи призначена для закриття сесій і видалення даних, які дозволяють користувачам отримати доступ без використання атрибутів доступу системи.

На стороні FE, вона потребує натиснення кнопки “Logout”. Після відправки запиту на сервер (BE), модуль звертається до ендпоінту “GET /authentication/logout” для видалення токена доступу з бази даних і сховища браузера, а також перенесення користувача на сторінку “Sign In”.

3.1.4 Відновлення доступу до облікових записів

Процедура відновлення пароля облікового запису призначена виключно для користувачів, які мають доступ до електронної адреси використаної під час

їх реєстрації. Вона розділена на дві окремі процедури: створення запиту та збереження нового паролю.

На стороні FE, формування запиту потребує виконання наступних кроків.

Крок 1. Натиснути кнопку “Sign Up”.

Крок 2. Натиснути кнопку “Forgot Password”.

Крок 3. Ввести електронну адресу користувача системі (email address).

Крок 4. Натиснути кнопку “Reset”.

Після введення даних користувачем і їх подальшої відправки на сервер (BE), модуль звертається до ендпоінту “POST /authentication/password/reset/send” для створення токена безпеки та посилання на відновлення пароля. Для валідації електронної адреси користувача (email address) здійснюється відповідно до реалізації у п.п.3.1.1.

Посилання у форматі “<server_hostname>authentication/ password/setup /<activation_token>” відправляється на електронну пошту, вказану користувачем. Виникнення помилок у формуванні токена призводить до скасування процедури відновлення доступу і видалення попередньо виконаних змін у базі даних. Однак, якщо токен сформовано та виникли помилки у формуванні посилання або його відправки на електронну адресу користувача, то зміна пароля можлива лише при прямому доступі до бази даних системи з використанням SQL-запитів.

У разі натиснення на посилання у вхідному електронному листі, користувач переходить на сторінку зміни пароля облікового запису “Setup New Password Page”. Для встановлення нового пароля, на стороні FE, необхідно виконати наступні кроки.

Крок 1. Ввести новий пароль для доступу користувача у систему (password).

Крок 2. Повторити введення нового паролю для підтвердження його валідності (confirmation password).

Крок 3. Натиснути кнопку “Send” для завершення процедури відновлення пароля.

Після введення обов’язкових даних та їх відправки на сервер (BE), модуль звертається до ендпоінту “POST /authentication/password/reset/send” для валідації, обробки та збереження нового пароля користувача. Їх валідація здійснюється відповідно до реалізації у п.п.3.1.1. У разі введення валідних даних, новий пароль буде автоматично збережений у базі даних і користувач отримає можливість доступу до системи використовуючи атрибути доступу “<username>:<password>” свого облікового запису.

При виникненні помилок під час входу в обліковий запис потрібно передбачити кастомні повідомлення щодо причин виникнення помилок. Серед них передбачено обробку наступних випадків:

- не валідний пароль (password або confirmation password);
- паролі (password та confirmation password) відрізняються один від одного;
- інші помилки субмодуля пов’язані з валідацією даних і їх додавання до бази даних.

3.1.5 Перегляд облікових записів

Перегляд облікових записів користувачів системи доступний виключно на сторінці “Settings”, у розділі “Accounts Management”, доступ до якої мають лише користувачі ролі “Administrator”. Відповідно до вимоги макета на вказаній сторінці розміщено наступні дані:

- статус користувача (state);
- реальне ім’я користувача (firstname);
- реальне прізвище користувача (lastname);
- ім’я користувача у системі (username);
- електронна пошта користувача (email address).

Для отримання інформації про користувачів, модуль звертається до

ендпоінту “GET /settings/accounts”. Для доступу до інформації авторизованому користувачу необхідно виконати наступні кроки: у разі спроби доступу до сторінки за прямим посиланням і без наявності відповідних привілеїв, користувач буде направлений на сторінку “Dashboard” та отримає попередження про низький рівень привілеїв для доступу до вказаної сторінки.

3.1.6 Оновлення облікових записів

Процедура оновлення інформації користувачів системи дозволяє змінити дані користувача, які він зазначив під час реєстрації у системі.

Користувач має можливість змінювати атрибути: `firstname`, `lastname`, `email`, `role`, шляхом введення нових значень на стороні FE. У разі підтвердження їх зміни, модуль звертається до ендпоінту “GET /settings/update/accounts /<string:field>/<string:value>/id/<string:key>”, якому передаються вхідні дані: назва атрибута (`firstname`, `lastname`, `email`, `role`), нове значення атрибута (відповідно до п.п.3.1.1), ідентифікатор користувача у системі (`id`). Після відправки запиту на сервер (BE), виконується оновлення вказаних атрибутів у базі даних.

Виключенням є атрибути доступу (`username` і `password`). Можливість зміни першого заблокована за для моніторингу дій користувачів системи. Зміна пароля доступна лише користувачеві. При цьому, адміністратор має можливість змінити електронну адресу користувача і виконати відновлення пароля з її допомогою (резервний спосіб відновлення доступу).

Зміна статусу користувача здійснюється шляхом натиснення кнопок “Connect”/“Disconnect” на стороні FE. У разі взаємодії з ними, модуль звертається до ендпоінту “GET /settings/update/ accounts/state/<string:value> /id/<string:key>”, якому передаються вхідні дані: статус користувача (активний – 1, неактивний – 0), ідентифікатор користувача у системі (`id`). Після відправки запиту на сервер (BE), виконується оновлення статусу користувача у базі даних.

Виключенням є користувач з правами супер адміністратора (`id=0`). В

системі заблоковано можливість його деактивації для попередження втрати доступу до сторінки “Accounts Management”. Перегляд і зміна даних користувачів, при цьому, можливі лише, при прямому доступі до бази даних з відповідних використанням SQL-запитів.

3.2 Модуль підтримки користувачів

Даний модуль забезпечує можливість організації зворотного зв'язку між користувачами вебсайту та адміністраторами. Він може стати у нагоді, при виявленні багів під час роботи проекту.

Для відправки запиту для системного адміністратора, користувач повинен перейти на сторінку “Leave Feedback” виконати наступні кроки.

- Крок 1. Ввести тему звернення (title).
- Крок 2. Ввести текст звернення (text).
- Крок 3. Натиснути кнопку “Send”.

Після введення обов'язкових даних та їх відправки на сервер (BE), модуль звертається до ендпоінту “POST /feedback/send” для валідації, обробки та збереження запиту. Валідація теми і тексту запиту виконується на основі наступного регулярного виразу – “`^[w\d\s,\.]{5,255}$`” згідно з яким, при повинні використовуватися символи латинського алфавіту, цифри, пробіли і крапка. Довжина обох полів повинна складати від 5 до 255 символів.

При виникненні помилок під час входу в обліковий запис потрібно передбачити кастомні повідомлення щодо причин виникнення помилок. Серед них передбачено обробку наступних випадків:

- не валідний формат теми запиту (title);
- не валідний формат тексту запиту (text).;
- інші помилки субмодуля пов'язані з валідацією даних і їх додавання до бази даних.

Перегляд усіх відправлених запитів доступний через електронну пошту або у розділі “Feedback Settings” сторінки “Settings” (рис. 3.1).

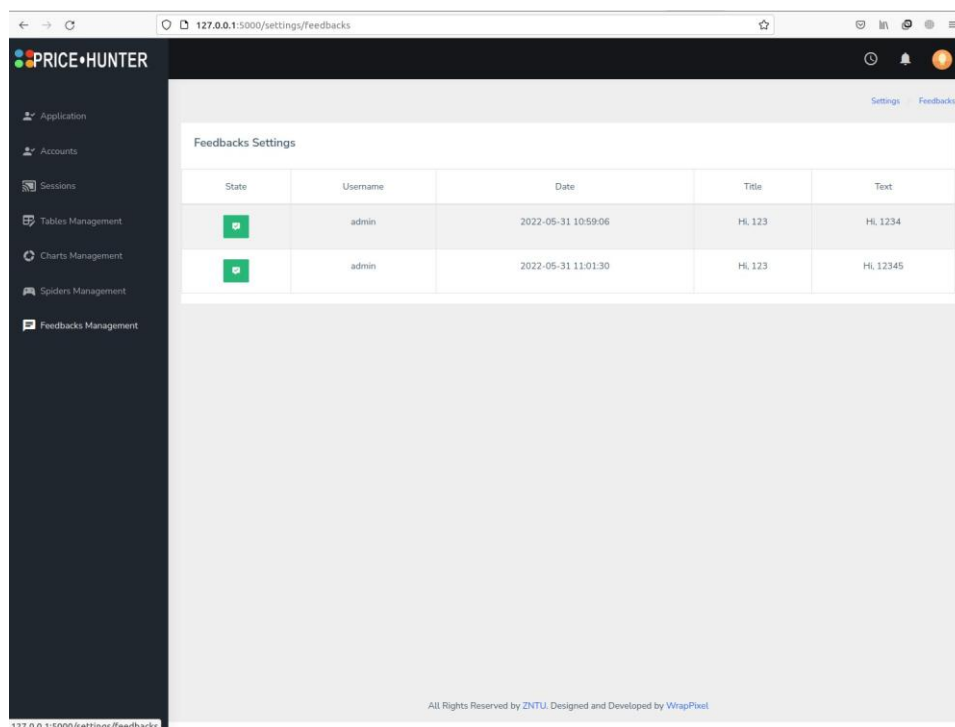


Рисунок 3.1 – Зовнішній вигляд сторінки “Feedback Settings”

3.3 Модуль управління даними

Модуль управління даними призначений для організації робіт щодо введення даних користувачем, збору, накопичення, відображення даних і налаштування зовнішнього виду даних у різних режимах.

3.3.1 Введення даних користувачем

При взаємодії з вебсайтом користувачі здійснюють введення даних за допомогою клавіатури. Для перевірки вказаних даних розроблено клас “Validator”, який дозволяє централізовано перевірити валідність вхідних даних перед їх відправкою до бази даних. Він включає в себе наступні методи:

- `validate_firstname()` – для валідації реального імені користувача;
- `validate_lastname()` – для валідації реального прізвища користувача;
- `validate_username()` – для валідації імені користувача;
- `validate_password()` – для валідації пароля від облікового запису

користувача;

- `validate_password_x2()` – для перевірки паролів на ідентичність;
- `validate_email()` – для валідації електронної адреси;
- `validate_limit()` – для валідації розміру виведення даних у табличному режимі;
- `validate_timeframe()` – для валідації часу оновлення даних на дашборді;
- `validate_mode()` – для валідації режиму відображення даних на дашборді;
- `validate_chart()` – для валідації типу графіку у графічному режимі;
- `validate_report()` – для валідації типу обраного репорту для побудови графіків;
- `validate_feedback_title()` – для валідації теми запиту до адміністратора;
- `validate_feedback_text()` – для валідації тексту запиту до адміністратора.

Інформація для перевірки даних задана у змінних класу.

3.3.2 Введення даних з парсеру

Для збору інформації щодо компаній-конкурентів створено відповідні класи-парсери і відповідні методи у класі “Database”. Отримання даних з сайтів, які належать компаніям-конкурентів здійснюється шляхом вивантаження даних за допомогою API або програмного модуля “bs4” для парсингу документів у форматі HTML. Фрагменти їх програмного коду наведені на рис. 3.2 і 3.3.

```

class Plugin<company_title>:
    def __init__(self, database=None):
        self.app_state = False
        self.service_state = "offline"
        self.service_timeframe = 1
        self.app_timeframe = 1
        self.alert_pause = 1
        self.database = database
        self.table = ""
    def start(self):
        pass
    def get_state(self):
        pass
    def extract_data(self):
        pass
    def save_to_database(self, data):
        pass
# company_title - назва компанії;
# __init__ - метод створення об'єкту;
# стан модуля: активний, неактивний;
# доступність сайту: online, offline;
# інтервал оновлення сайту (хвилини);
# інтервал вивантаження даних (хвилини);
# інтервал перезапуску (хвилини);
# посилання на об'єкт бази даних;
# назва таблиці у базі даних;
# метод запуску парсингу;
# метод перевірки зв'язку з сайтом;
# метод вивантаження даних;
# метод збереження до бази даних;
# код парсеру даних компанії-конкурента;

```

Рисунок 3.2 – Шаблон парсеру на основі API запитів

```

class Plugin<company_title>:
    def __init__(self, database=None,
                  currency=None):
        self.app_state = False
        self.database = database
        self.currency = currency
        self.base_url = ""
        self.service_assets = []
        self.service_state = "offline"
        self.alert_pause = 1
        self.service_timeframe = 1
        self.filter_company = []
        self.filter_asset = []
        self.filter_location = []
        self.filter_place = []
        self.table = ""
    def start(self, currency):
        pass
    def get_state(self):
        pass
    def extract_data(self, asset):
        pass
    def save_to_database(self, data):
        pass
# company_title - назва компанії;
# __init__ - метод створення об'єкту;
# стан модуля: активний, неактивний;
# посилання на об'єкт бази даних;
# вартість валюти для конвертації цін;
# базова адреса сайту;
# фільтр за товаром/послугою (сайт);
# доступність сайту: online, offline;
# інтервал перезапуску (хвилини);
# інтервал вивантаження даних (хвилини);
# фільтр за компанією;
# фільтр за товаром/послугою;
# фільтр за регіоном збуту;
# фільтр за місцем збуту;
# назва таблиці у базі даних;
# метод запуску парсингу;
# метод перевірки зв'язку з сайтом;
# метод вивантаження даних;
# метод збереження до бази даних;
# код парсеру даних компанії-конкурента;

```

Рисунок 3.3 – Шаблон парсеру на основі парсингу HTML документів

3.3.3 Виведення інформації з бази даних

Для відображення інформації про цінові пропозиції збережені у базі даних після парсингу в системі передбачено 2 режими: табличний (рис. 3.4) і графічний (рис. 3.5).

PRICE•HUNTER			
Dashboard / Table Mode			
Updated	Quality	Price	Deviation
2022-05-04 06:43:00	3	9250.0	0.0
2022-05-04 06:43:00	3	9150.0	0.0
2022-05-04 06:43:00	3	9050.0	0.0
2022-05-04 06:43:00	3	9050.0	0.0
2022-05-04 06:43:00	3	9050.0	0.0
2022-05-04 06:43:00	3	8980.0	0.0
2022-05-04 06:43:00	3	8950.0	0.0
2022-05-04 06:43:00	3	8930.0	0.0
2022-05-04 06:43:00	3	8920.0	0.0
2022-05-04 06:43:00	3	8920.0	0.0
2022-05-04 06:43:00	3	8870.0	0.0
2022-05-04 06:43:00	3	8870.0	0.0
2022-05-04 06:43:00	3	8800.0	0.0
2022-05-04 06:43:00	3	8790.0	0.0
2022-05-04 06:43:00	3	8710.0	0.0
2022-05-04 06:43:00	3	8700.0	0.0
2022-05-04 06:43:00	3	8483.921	0.0

Рисунок 3.4 – Зовнішній вигляд сторінки “Dashboard” (табличний режим)



Рисунок 3.5 – Зовнішній вигляд сторінки “Dashboard” (графічний режим)

Табличний вигляд краще підійде для отримання достовірних даних і коригування власних цінових пропозицій. Напроти, графічний режим дозволяє оцінити треди ринку та прийняти обґрунтовані бізнес-рішення.

3.3.4 Встановлення інтервалу оновлення даних дашборду

Функція налаштування часу оновлення даних на дашборді дозволяє

будь-якому користувачу встановити інтервал відображення оновлень з бази даних на сторінці “Dashboard”. При цьому модуль звертається до ендпоінту “GET /settings/update/constants/timeframe/<string:value>/-/-” передавши йому одне із запропонованих значень інтервалу (value).

3.4 Модуль безпеки даних

Модуль безпеки є головою частиною розробленого проєкту. Його основним завданням є розмежування доступу до системи, її налаштувань, даних користувачів і зібраних даних про компанії. Для виконання вказаного завдання були виконані наступні завдання та процедури: створено політики рольового доступу та парольного захисту, процедури активації облікових записів, моніторинг дій користувачів, управління сесіями та блокування доступу до системи.

3.4.1 Політика парольного захисту

Механізм обмеження доступу з використанням пароля є базовим механізмом безпеки будь-якого вебсайту. Його знання дозволить отримати доступ до будь-яких функцій сайту, відповідно до привілеїв доступу закріплених за ним.

Відповідно до [22], були прийняті наступні вимоги до парольного захисту:

- використання символів верхнього та нижнього регістрів (A-Z, a-z, діакритичні та грецькі символи);
- використання базових цифр (0-9);
- мінімальна довжина пароля не менше 8 символів.

Окрім того, висунуто вимоги щодо зберігання паролів у базі даних системи:

- паролі не повинні зберігатися у відкритому вигляді;

– необхідно використовувати односторонню функцію перетворення паролів, стійкість якої не дозволяє виконати їх відновлення (наприклад, хешування за допомогою алгоритму MD5).

Необхідно забезпечити ротацію паролів користувачів кожні 3 місяці.

Використовуючи вищевказані рекомендації створено регулярний вираз для валідації паролів на стороні ВЕ, який наведено у п.п.3.1.1.

3.4.2 Політика рольового доступу

Для обмеження доступу до окремих функцій і розділів сайту, зокрема до дашборду та панелі налаштувань, реалізовано систему розмежування доступу на основі ролей і передбачено лише 2 ролі: administrator та user. Перелік правил доступу для кожної ролі наведено у табл. 3.1.

Таблиця 3.1 – Правила розмежування доступу по API

Ендпоінти	Ролі користувачів		
	Guest	User	Administrator
1	2	3	4
/	—	+	+
/authentication/signup	+	+	+
/authentication/login	+	+	+
/authentication/password/reset	+	+	+
/authentication/password/setup/<string:token>	+	+	+
/authentication/signup/send	+	+	+
/authentication/password/activate/<string:token>	+	+	+
/authentication/login/send	+	+	+
/authentication/logout	—	+	+

Продовження таблиці 3.1

1	2	3	4
/authentication/password/reset/send	+	+	+
/authentication/password/setup/send	+	+	+
/dashboard	—	+	+
/dashboard/chart/select-axis/<string:axis>/<string:title>	—	+	+
/dashboard/select-limit/<int:limit>	—	+	+
/dashboard/select-timeframe/<int:timeframe>	—	+	+
/dashboard/select-mode/<string:mode>	—	+	+
/dashboard/select-chart/<string:chart>	—	+	+
/dashboard/select-report/<string:report>	—	+	+
/dashboard/select-report/<string:report>/<string:type>	—	+	+
/feedback	—	+	+
/feedback/send	—	+	+
/settings	—	—	+
/settings/<string:table>	—	—	+
/settings/update/<string:table>/<string:field>/<string:value>/<string:primary>/<string:key>	—	—	+

З метою перевірки виконання вищевказаних правил доступу створено 2 типи декораторів:

- login_required(), який призначено для перевірки наявності авторизації у системі;
- login_not_required(), який призначено для перевірки відсутності авторизації у системі.

3.4.3 Активація облікових записів

Процедура активації облікових записів призначена для зміни їх статусу в базі даних і підтвердження його привілеїв для використання у системі. Вказана процедура реалізована для наступних сценаріїв:

- створено новий обліковий запис і час життя токenu активації не вичерпано;
- створено новий обліковий запис і час життя токenu активації вичерпано.

У першому випадку на електрону пошту адміністратора (system email address) надійде повідомлення щодо створення запиту на новий обліковий запис користувача та посилання у форматі “https://<server_address>/authentication/password/activate/<token>”. Після переходу за посиланням модуль звертається до ендпоінту “GET /authentication/password/activate/<string:token>” для зміни статусу облікового запису у базі даних. Якщо на момент переходу за посиланням, обліковий запис існує в базі даних, токен активації не був видалений з бази даних і обліковий запис не був активований раніше, то статус користувача змінюється на активний. Користувач отримає можливість входу на вебсайт.

У другому випадку на електрону пошту адміністратора (system email address) також надійде повідомлення щодо створення запиту на новий обліковий запис користувача та посилання у форматі “https://<server_address>/authentication/password/activate/<token>”.

Через 48 годин строк дії токenu було вичерпано, про що адміністратор буде повідомлений у відповіді ентоінту “GET /authentication/password/activate/<string:token>”. При цьому єдиний спосіб активувати обліковий запис знаходиться на сторінці “Settings”, у розділі “Accounts Management”. На сторони FE, адміністратор повинен знайти необхідний обліковий запис і натиснути на кнопку “Activate”. Після цього модуль звернеться до сервера – ендпоінт “GET /settings/update/accounts/state/<string:value>/id/<string:key>” і активація запису

здійснюватиметься відповідно до кроків у п.п.3.1.6.

3.4.4 Моніторинг дій користувачів

Для виконання функцій управління доступом і розслідування інцидентів інформаційної безпеки, в розробленому вебсайті впроваджено 2 функції для моніторингу дій користувачів:

- логування подій;
- перегляд активних підключень (сесій).

Збір та накопичення логів реалізовані на базі програмного модуля “logging”, який дозволяє збирати інформацію про результати роботи ендпоінтів і записувати їх в окремий текстовий файл. Формат запису логів обраний якомога повніший – “%(asctime)s %(levelname)s %(name)s %(threadName)s : %(message)s”. Він дозволяє збирати наступні дані:

- asctime – час фіксації події у форматі “YYYY-MM-DD HH:MM:SS,MS”;
 - levelname – тип подій для фіксації (CRITICAL, DEBUG, ERROR, FATAL, WARN, INFO);
 - name – назва модуля, фіксація якого здійснюється;
 - threadName – назва потоку обробки запиту.

За допомогою методу logging(string) збирається інформація щодо IP адреси, назви ендпоінту, інформація про помилки (код, назва, опис) і т.д.

Для зручності перегляду активних підключень створено розділ “Sessions Management” на сторінці “Settings”, на якій представлено дані щодо імені користувачів, токенів підключень, дати і часу автоматичного анулювання токенів (рис. 3.6).

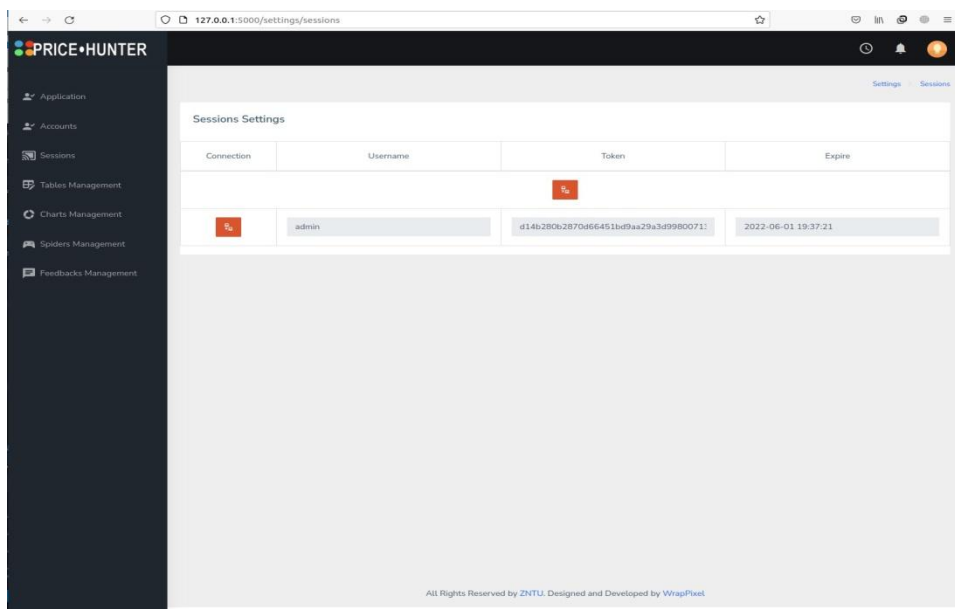


Рисунок 3.6 – Зовнішній вигляд сторінки розділу “Sessions Management”

3.4.5 Блокування доступу користувачам

Процедура блокування доступу облікового запису дозволяє перешкодити можливості авторизації будь-якого користувача у системі. Вказана опція доступна лише користувачам з роллю “Administrator” і може бути використана шляхом натиснення кнопок “Activate” / “Block” у розділі “Accounts Management” сторінки “Settings” (рис. 3.7).

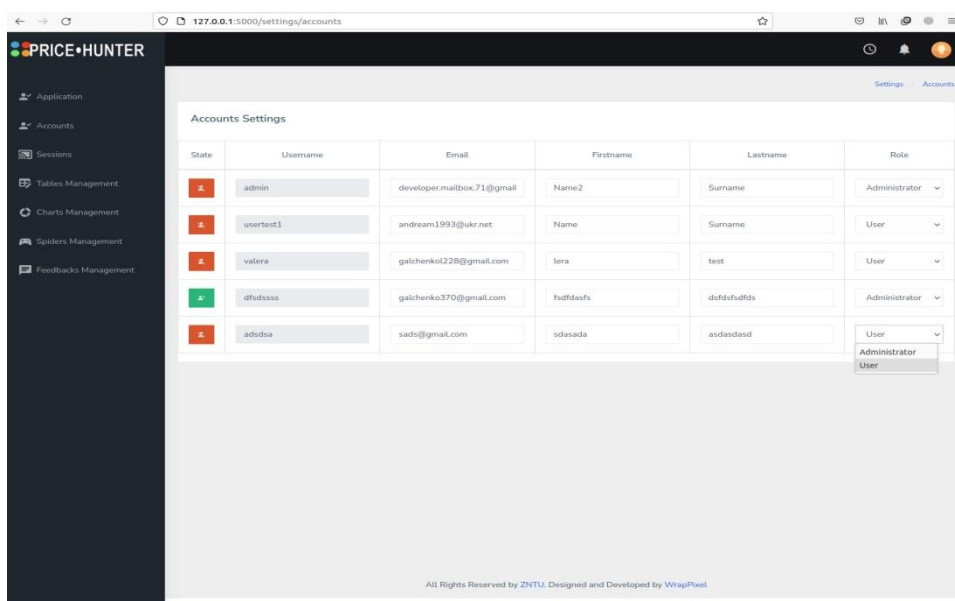


Рисунок 3.7 – Зовнішній вигляд сторінки розділу “Accounts Management”

Натиснення кнопок “Activate” / “Block” на FE дозволяє модулю звернутися до ендпоінту “GET /settings/update/accounts/state/<string:value>/id/<string:key>” і передати у якості параметрів наступні дані: стан облікового запису (value), ідентифікатор облікового запису (id), відповідно до п.п. 3.1.6.

3.4.6 Відключення користувачів від системи

Процедура відключення користувачів призначена для негайного анулювання токенів сесій і переведення користувачів на сторінку “Sign In”. Вказаний інструмент може бути використаний у наступних випадках:

- необхідно одночасно застосувати оновлення вебсайту (адміністрування);
- необхідно відключити сторонніх осіб, які отримали доступ до персонального комп’ютера після входу користувача на сайт на вебсайт.

Анулювання сесій з боку FE можливо виконати лише у розділі “Sessions Managemet” сторінки “Settings” шляхом натискання кнопки “Disconnect” / “Disconnect All” (рис. 3.5).

Натиснення кнопки “Disconnect” на FE дозволяє модулю звернутися до ендпоінту “GET /settings/update/sesstions/-/-/id/<string:key>”, передати у якості параметрів ідентифікатор сесії (id), видалити запис про неї з бази даних, відповідно до п.п. 3.1.6.

Натиснення кнопки “Disconnect All” на FE дозволяє модулю звернутися до ендпоінту “GET /settings/update/sesstions/-/-/-” і видалити усі записи про активні сесії з бази даних, відповідно до п.п. 3.1.6.

3.5 Структура бази даних

Враховуючи вищевказані опції створеного проєкту виникла необхідність у використанні бази даних для зберігання інформації. За функціональним призначенням, їх можливо поділити на 3 групи:

- системні;
- користувацькі;
- для парсингу даних (products_agrotender).

Системні таблиці призначені для зберігання службової інформації вебсайту. До них віднесені таблиці: charts, limits, modes, reports, roles, sorts, sources, timeframes, visibility.

Таблиця “charts” зберігає інформацію про види графіків, які користувач може обрати при виведенні інформації на сторінці “Dashboard” (рис. 3.8).

ph charts	
id	int(11)
title	varchar(50)
variable	varchar(50)
comments	text

Рисунок 3.8 – Структура таблиці “charts”

Вибір типу графіка здійснюється шляхом звернення до ендпоінту “GET /dashboard/select-chart/<string:chart>” і передачі типу графіка (line, bar, pie, donut і т.д).

Таблиця “limits” зберігає інформацію про кількість записів доступних для виведення на сторінці “Dashboard” у табличному режимі (рис. 3.9).

ph limits	
id	int(11)
title	varchar(50)
variable	varchar(50)
comments	text

Рисунок 3.9 – Структура таблиці “limits”

Вибір типу графіка здійснюється шляхом звернення до ендпоінту “GET /dashboard/select-limit/<int:limit>” і передачі кількості записів для

відображення (limit).

Таблиця “modes” зберігає інформацію про види графіків, які користувач може обрати при виведенні інформації на сторінці “Dashboard” (рис. 3.10).



id	title	variable	comments
id : int(11)	title : varchar(50)	variable : varchar(50)	comments : text

Рисунок 3.10 – Структура таблиці “modes”

Вибір типу графіка здійснюється шляхом звернення до ендпоінту “GET /dashboard/select-mode/<string:mode>” і передачі назви режиму (table, chart).

Таблиця “reports” зберігає інформацію про змінні графіків по осях, які користувач може обрати із запропонованих або відповідно до власних завдань при виведенні інформації на сторінці “Dashboard” (рис. 3.11).



id	title	x_axis	y_axis	sources	min	max	avg
id : int(11)	title : varchar(100)	x_axis : varchar(100)	y_axis : varchar(100)	sources : varchar(10)	min : tinyint(4)	max : tinyint(4)	avg : tinyint(4)

Рисунок 3.11 – Структура таблиці “reports”

Вибір типу репорта здійснюється шляхом звернення до ендпоінту “GET /dashboard/select-report/<string:x_axis>_<string:y_axis>” і передачі назв осей (x_axis, y_axis).

Таблиця “roles” зберігає інформацію про доступні ролі, при роботі з вебсайтом (рис. 3.12).

ph roles	
id	int(11)
title	varchar(50)
variable	varchar(50)
comments	text

Рисунок 3.12 – Структура таблиці “roles”

Вибір ролі здійснюється шляхом звернення до ендпоінту “GET /settings/update/accounts/role/<string:role>/id/<int:id>” і передачі наступної інформації: назва ролі (role) та ідентифікатор облікового запису (id).

Таблиця “sorts” зберігає інформацію про режими сортування даних у табличному режимі на сторінці “Dashboard” (рис. 3.13).

ph sorts	
id	tinyint(4)
title	varchar(10)
variable	varchar(5)
comments	text

Рисунок 3.13 – Структура таблиці “sorts”

Вибір напряму сортування здійснюється шляхом звернення до ендпоінту “GET /settings/update/view_table/<string:column>/<string:value>/options /sort” і передачі наступної інформації: назва колонки (column), напрям сортування (value).

Таблиця “sources” зберігає інформацію про таблиці, які містять результати парсингу сайтів-конкурентів (рис. 3.14).

ph sources	
id	int(11)
state	tinyint(4)
title	varchar(50)
table	varchar(50)
comments	text

Рисунок 3.14 – Структура таблиці “sources”

Вибір джерел з даними здійснюється шляхом звернення до ендпоінту “GET /settings/update/sources/state/<int:state>/id/<int:id>” і передачі наступної інформації: статус джерела (state), ідентифікатор таблиці у базі даних (id).

Таблиця “timeframes” зберігає інформацію про кількість записів доступних для виведення на сторінці “Dashboard” у табличному режимі (рис. 3.15).

ph timeframes	
id	int(11)
title	varchar(50)
variable	varchar(50)
comments	text

Рисунок 3.15 – Структура таблиці “timeframes”

Вибір таймфрейма здійснюється шляхом звернення до ендпоінту “GET /dashboard/select-timeframe/<int:timeframe>” і передачі значення інтервалу оновлень (timeframe) у якості вхідного параметра.

Таблиця “visibility” зберігає інформацію щодо видимості окремих стовпців на сторінці “Dashboard” у табличному режимі (рис. 3.16).



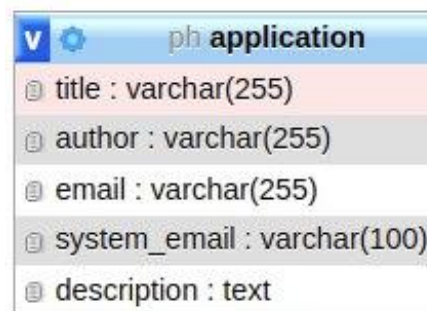
ph visibility	
id	tinyint(4)
title	varchar(5)
variable	varchar(4)
comments	text

Рисунок 3.16 – Структура таблиці “visibility”

Вибір видимості здійснюється шляхом звернення до ендпоінту “GET /settings/update/view_table/<string:column>/<string:value>/options/visibility” і передачі наступної інформації: назва колонки (column), назва стовпчика (value).

Користувачькі таблиці призначені для зберігання робочої інформації вебсайту. До неї віднесені таблиці: application, accounts, constants, feedbacks, notifications, requests, sessions, view_chart, view_table.

Таблиця “application” призначена для зберігання презентаційної інформації вебсайту (рис. 3.17).



ph application	
title	varchar(255)
author	varchar(255)
email	varchar(255)
system_email	varchar(100)
description	text

Рисунок 3.17 – Структура таблиці “application”

Зміна вказаної інформації здійснюється шляхом звернення до ендпоінту “GET /settings/update/application/<string:field>/<string:value>/-/-”, передачі назви поля (field) і його нового значення (value) у якості вхідних даних.

Таблиця “accounts” зберігає інформацію про облікові записи користувачів вебсайту (рис. 3.18).

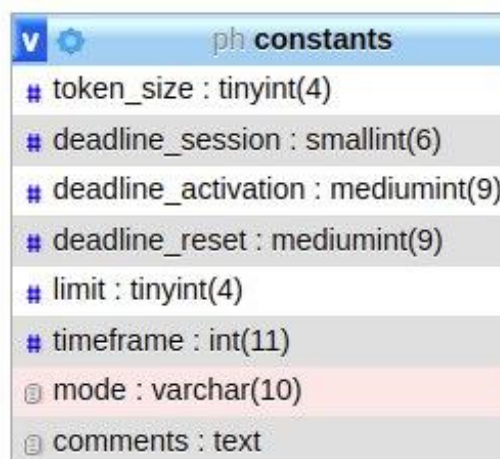


ph accounts	
id	int(11)
state	tinyint(4)
username	varchar(25)
password	varchar(50)
email	varchar(100)
firstname	varchar(50)
lastname	varchar(50)
role	varchar(10)

Рисунок 3.18 – Структура таблиці “accounts”

Зміна вказаної інформації здійснюється шляхом звернення до ендпоінту “GET /settings/update/accounts/<string:field>/<string:value>/id/<int:id>”, передачі назви поля (field), його нового значення (value) і ідентифікатора користувача (id) у якості вхідних даних.

Таблиця “constants” зберігає інформацію про найбільш часто вживані дані вебсайту, які неможливо віднести до будь-якої іншої таблиці (рис. 3.19).

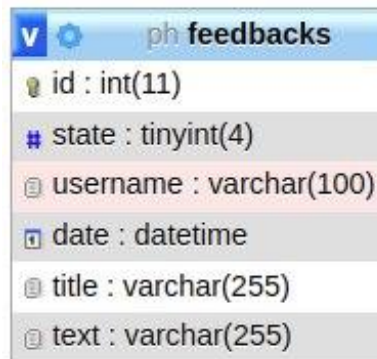


ph constants	
token_size	tinyint(4)
deadline_session	smallint(6)
deadline_activation	mediumint(9)
deadline_reset	mediumint(9)
limit	tinyint(4)
timeframe	int(11)
mode	varchar(10)
comments	text

Рисунок 3.19 – Структура таблиці “constants”

Зміна вказаної інформації здійснюється шляхом звернення до ендпоінту “GET /settings/update/constants/<string:field>/<string:value>/-/-”, передачі назви поля (field), його нового значення (value) у якості вхідних даних.

Таблиця “feedbacks” зберігає інформацію про звернення користувачів до адміністратора системи (рис. 3.20).

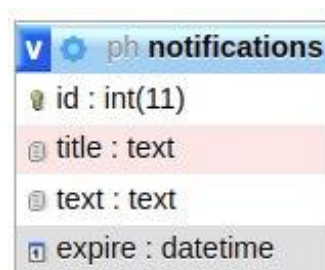


ph feedbacks	
id	int(11)
state	tinyint(4)
username	varchar(100)
date	datetime
title	varchar(255)
text	varchar(255)

Рисунок 3.20 – Структура таблиці “feedbacks”

Можливість перегляду та оновлення статусу запитів можливі шляхом звернення до ендпоінтів “GET /settings/feedbacks” та “GET /settings/update/feedbacks/state/<string:value>/id/<int:id>”, передачі нового статусу (value) і ідентифікатора звернення (id) у якості вхідних даних.

Таблиця “notifications” зберігає інформацію про повідомлення, які адміністратори системи бажають донести до користувачів через сторінку “Dashboard” (рис. 3.21).



ph notifications	
id	int(11)
title	text
text	text
expire	datetime

Рисунок 3.21 – Структура таблиці “notifications”

Таблиця “requests” зберігає інформацію щодо запитів користувача на активацію або відновлення пароля облікового запису системи (рис. 3.22).



ph requests	
id	int(11)
username	varchar(50)
token	varchar(255)
expire	datetime
comments	text

Рисунок 3.22 – Структура таблиці “requests”

Можливість оновлення статусу запитів можлива шляхом звернення до відповідних ендпоінтів для активації (п.п. 3.3.2) і відновлення доступу (п.п. 3.1.4).

Таблиця “sessions” зберігає інформацію про активні підключення до вебсайту (рис. 3.23).



ph sessions	
id	int(11)
username	varchar(50)
token	varchar(255)
expire	datetime
comments	text

Рисунок 3.23 – Структура таблиці “sessions”

Можливість перегляду та оновлення статусу підключень можливі шляхом звернення до ендпоінтів “GET /settings/sessions”, “GET /settings/update/sessions/state/<string:value>/id/<int:id>” та “GET /settings/update/sessions/-/-/-/-”, передачі нового статусу (value) і ідентифікатору підключення (id) у якості вхідних даних (за необхідності).

Таблиця “view_chart” зберігає інформацію про діючі налаштування зовнішнього вигляду даних сторінки “Dashboard” у графічному режимі (рис. 3.24).

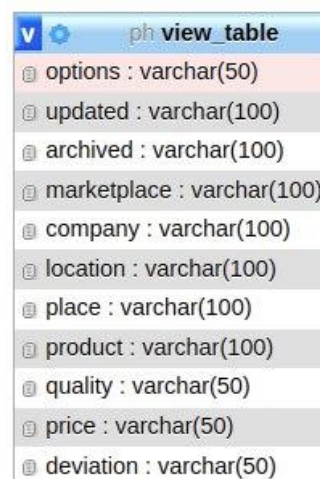


ph view_chart	
#	report_id : tinyint(4)
#	chart_id : tinyint(4)
Ⓜ	x_axis : varchar(100)
Ⓜ	y_axis : varchar(100)
#	videowall_id : tinyint(4)
Ⓜ	source_color_1 : varchar(20)
Ⓜ	source_color_2 : varchar(20)
Ⓜ	source_color_3 : varchar(20)
Ⓜ	comments : text

Рисунок 3.24 – Структура таблиці “view_chart”

Можливість перегляду та оновлення вигляду графіків можливі шляхом звернення до ендпоінтів “GET /settings/view_chart”, “GET /settings/update/view_chart/<string:field>/<string:value>”, передачі назви поля (field) і його нового значення (value) у якості вхідних даних (за необхідності).

Таблиця “view_table” зберігає інформацію про діючі налаштування зовнішнього вигляду даних сторінці “Dashboard” у табличному режимі (рис. 3.25).



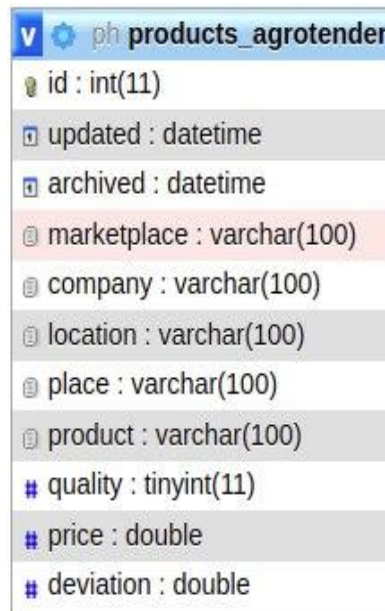
ph view_table	
Ⓜ	options : varchar(50)
Ⓜ	updated : varchar(100)
Ⓜ	archived : varchar(100)
Ⓜ	marketplace : varchar(100)
Ⓜ	company : varchar(100)
Ⓜ	location : varchar(100)
Ⓜ	place : varchar(100)
Ⓜ	product : varchar(100)
Ⓜ	quality : varchar(50)
Ⓜ	price : varchar(50)
Ⓜ	deviation : varchar(50)

Рисунок 3.25 – Структура таблиці “view_table”

Можливість перегляду та оновлення вигляду графіків можливі шляхом звернення до ендпоінтів “GET /settings/view_table”, “GET /settings/update/view_table/<string:field>/<string:value>/options/<string:type>”, передачі назви

поля (field), його нового значення (value) і типу змінюваних даних (type) у якості вхідних даних (за необхідності).

Таблиці для парсингу даних призначені виключно для зберігання актуальних даних про цінові пропозиції, а також іншої інформації конкурентів. На час розробки створено текстову таблицю products_agriotender (рис. 3.26).



Field	Type
id	int(11)
updated	datetime
archived	datetime
marketplace	varchar(100)
company	varchar(100)
location	varchar(100)
place	varchar(100)
product	varchar(100)
quality	tinyint(11)
price	double
deviation	double

Рисунок 3.26 – Структура таблиці “products_agriotender”

Збереження інформації у таблиці здійснюється за допомогою методу парсеру “save_to_database”, вибір даних для відображення у табличному режимі здійснюється шляхом звернення до ендпоінту “GET /settings/update/view_table/<string:field>/<string:value>/options/<string:type>”, перегляд даних у графічному та табличному режимах шляхом звернення до ендпоінту “GET /dashboard”.

Структурно, зв'язок між вищевказаними таблицями наведено на рис.3.27.

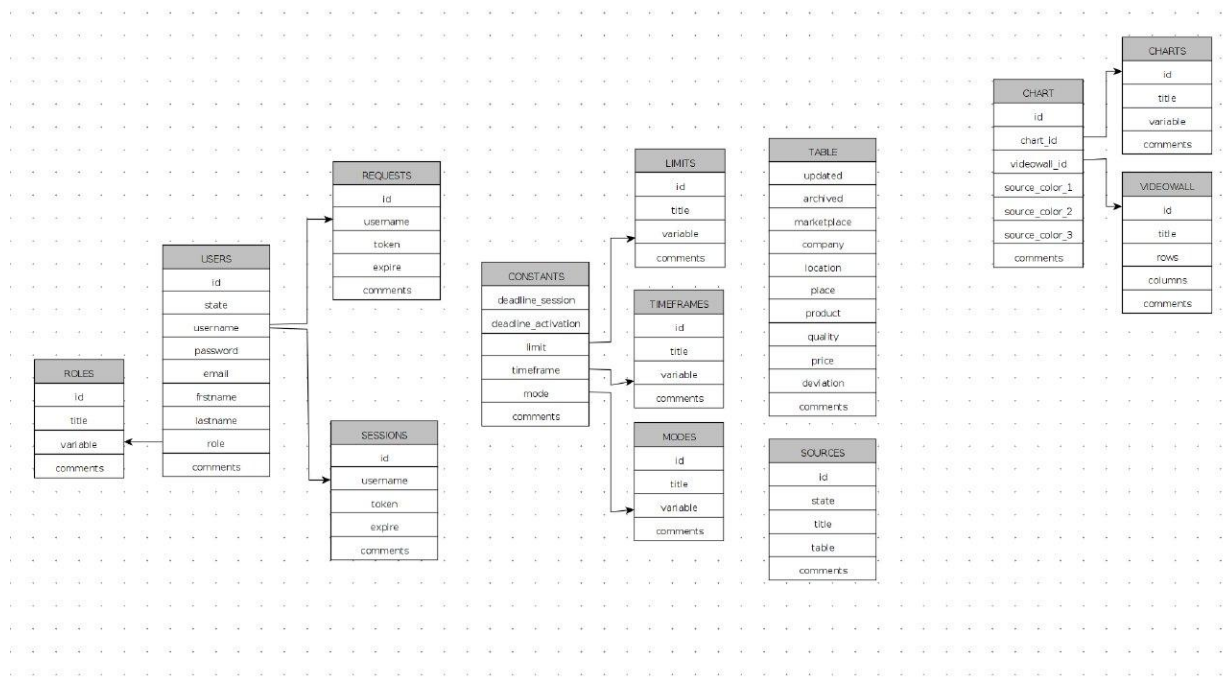


Рисунок 3.27 – Структурна схема відношень між таблицями бази даних

3.6 Висновки за розділом 3

Відповідно до мети роботи, у даному наведено опис реалізацій основних модулів проєкту, створено структуру бази даних та вказано зв'язок її таблиць з розробленими ендпоінтами.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ВЕБСАЙТУ

4.1 Вимоги до клієнтської частини

Для оцінки можливості користувачів ефективно користуватися створеним вебсайтом, виконано оцінку апаратних і програмних характеристик клієнтського робочого місця.

4.1.1 Апаратне забезпечення

Так для користування вебсайтом достатнім є наявність наступних апаратних характеристик:

- ЦП [CPU]: Intel Pentium 4 / Athlon 64 або більш пізньої версії з підтримкою SSE2;
- відеоадаптер [GPU]: 3D адаптер nVidia, Intel, AMD/ATI;
- звукова карта DirectX Compatible;
- відеопам'ять [VRAM]: 64 МБ;
- накопичувач [HDD]: 350 МБ.

4.1.2 Програмне забезпечення

Для використання вебсайту вимоги до програмного забезпечення не суттєві та включають наступні вимоги:

- наявність операційної системи Windows XP/Vista/7/8/10 та більш нові, Linux, Mac OS, Android, IOS;
- підтримка останніх версій браузерів Google Chrome, Opera, Mozilla Firefox, Microsoft Edge (десктопні та мобільні версії);
- стабільний доступ до мережі Інтернет (при необхідності віддаленого доступу) або доступ до локальної мережі (для корпоративного

використання).

4.2 Інструкція по роботі з програмою

Наступна інструкція включає інформацію стосовно розгортання та запуску серверної частини, запуску клієнтської частини та методик роботи з вебсайтом.

4.2.1 Розгортання серверної частини

Для розгортання серверної частини необхідно виконати наступні кроки.

Крок 1. Встановити інтерпретатор мови Python 3.8+ (якщо він не входить до стандартного пакета програм операційної системи сервера).

Крок 2. Встановити сервер баз даних MySQL з панелю керування phpMyAdmin (для зручного управління базою даних). Вказані пакети програм входять до складу програмного комплексу XAMPP або віддаленого хостингу з підтримкою cPanel.

Крок 3. Розпакувати файли серверної частини у будь-яку директорію.

4.2.2 Запуск програми

Для запуску серверної частини необхідно виконати наступні кроки.

Крок 1. Запустити сервер баз даних MySQL за допомогою команди “/opt/lampp/lampp restart” (з директорії встановлення комплексу XAMPP).

Крок 2. Запустити скрипт “wsgi.py”.

Крок 3. Запуск скриптів парсингу даних.

Запуск клієнтської частини передбачає виконання наступних кроків.

Крок 1. Завантаження одного з підтримуваних браузерів.

Крок 2. Перехід за посиланням “http://server:port/” або “https://server:port/”.

4.2.3 Методики роботи з вебсайтом

Методики роботи зі створеним сайтом включають набір інструкцій необхідних користувачеві для використання кожного розділу: реєстрація, авторизація, відновлення доступу, перегляд дашборду у табличному режимі, перегляд дашборду у графічному режимі, зміна кількості значень у табличному режимі, зміна інтервалу оновлення даних, вибір типу графіка, вибір типу даних для відображення на графіку.

Перехід за посиланням “http://server:port/” або “http://server:port/” переносить користувача на сторінку “Dashboard”. Якщо користувач не створював підключень раніше, то він буде перенесений на сторінку “Sign In” (рис. 4.1).

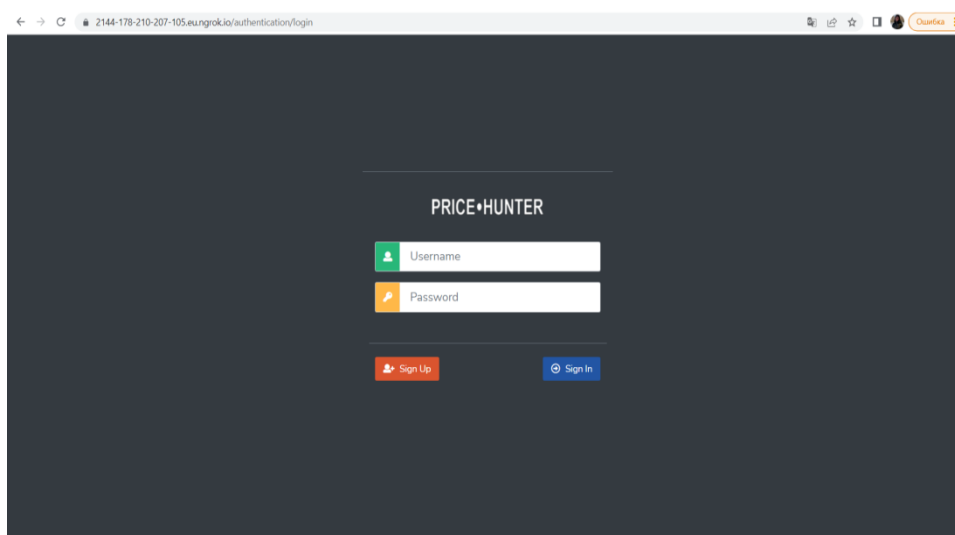


Рисунок 4.1 - Сторінка авторизації користувача

Для авторизації на сайті користувач повинен заповнити поля ім'я користувача (username) і пароль (password) та натиснути на кнопку “Sign In”.

У разі відсутності атрибутів доступу до вебсайту, користувач може створити їх на сторінці “Sign Up” (рис. 4.2).

PRICE•HUNTER

Firstname

Lastname

Username

Email

Password

Confirmation Password

[Forgot Password?](#) [Sign Up](#)

Рисунок 4.2 - Сторінка реєстрації користувача

Для авторизації на сайті користувач повинен заповнити поля реальних імені (Firstname) і прізвища (Lastname), імені користувача (Username), електронної поштової скрині (Email), основного (Password) та перевірного (Confirmation password) паролів, натиснути кнопку “Sign Up”.

Якщо користувач забув пароль, то він може відновити його на сторінці “Forgot Password” (рис. 4.3).

PRICE•HUNTER

Enter your e-mail address below and we will send you instructions how to recover a password

Your Email Address

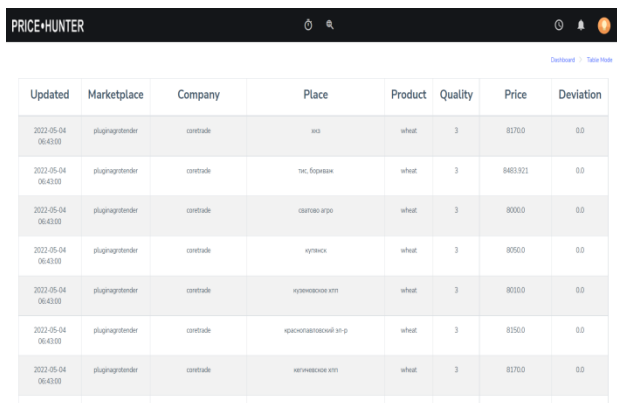
[Back To Login](#) [Reset](#)

Рисунок 4.2 - Сторінка реєстрації користувача

Для відновлення доступу до вебсайту користувач повинен заповнити поле електронної поштової скрині (Email) і натиснути кнопку “Reset”.

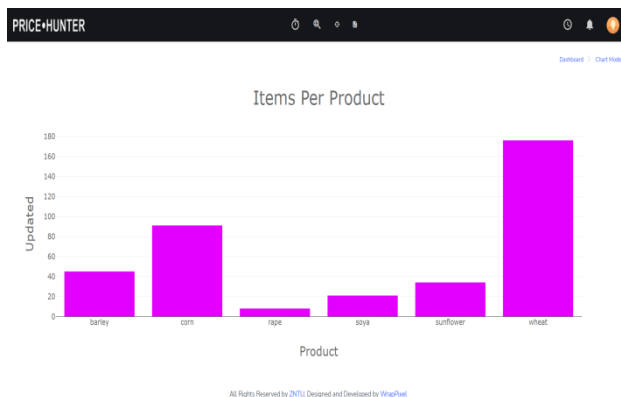
Після успішної авторизації у системі користувач переносить на сторінку

“Dashboard” і може переглянути інформацію щодо цінових пропозицій у табличному (рис. 4.3а) або графічному (рис. 4.3б) режимах, залежно від останніх збережених налаштувань у таблиці “constants”.



Updated	Marketplace	Company	Place	Product	Quality	Price	Deviation
2022-05-04 06:43:00	plugnagrander	contrade	ukr	wheat	3	8170.0	0.0
2022-05-04 06:43:00	plugnagrander	contrade	rus, forssak	wheat	3	8483.921	0.0
2022-05-04 06:43:00	plugnagrander	contrade	canada argo	wheat	3	8000.0	0.0
2022-05-04 06:43:00	plugnagrander	contrade	kyivsk	wheat	3	8050.0	0.0
2022-05-04 06:43:00	plugnagrander	contrade	ukrainian em	wheat	3	8010.0	0.0
2022-05-04 06:43:00	plugnagrander	contrade	ukrainian em p	wheat	3	8150.0	0.0
2022-05-04 06:43:00	plugnagrander	contrade	ukrainian em	wheat	3	8170.0	0.0

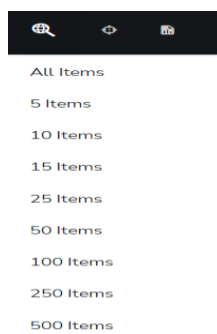
а



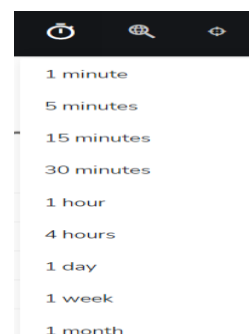
б

Рисунок 4.3 – Сторінка “Dashboard” у режимах: табличний (а) і графічний (б)

На сторінці “Dashboard”, у табличному режимі користувач має можливість налаштувати кількість даних для відображення (рис. 4.4а) і інтервал оновлення даних (рис. 4.4б) у табличному/графічному режимах шляхом вибору відповідних значень у викидному переліку.



а



б

Рисунок 4.4 – Викидні переліки:
кількість результатів (а) і інтервал оновлення результатів (б)

У графічному режимі користувач має можливість обрати налаштування для графіків шляхом вибору відповідних значень у викидному переліку з

типами підтримуваних графіків (рис 4.5а) і стандартними звітами системи (рис 4.5б).

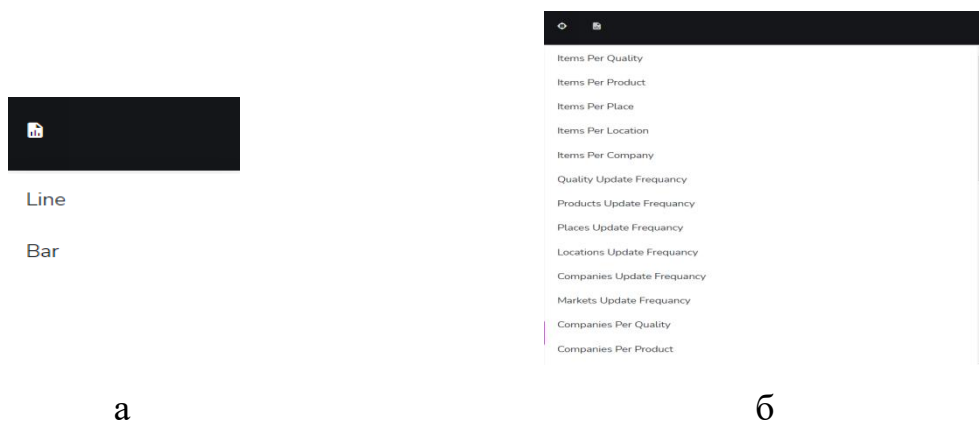


Рисунок 4.5 – Викидні переліки: типи графіків (а) і шаблони звітів (б)

Налаштування інтервалу оновлень даних, поточний час і час останнього оновлення користувач може побачити у викидному переліку (рис. 4.6а), перейти до інших розділів сайту за допомогою меню користувача (рис. 4.6б і 4.6в).

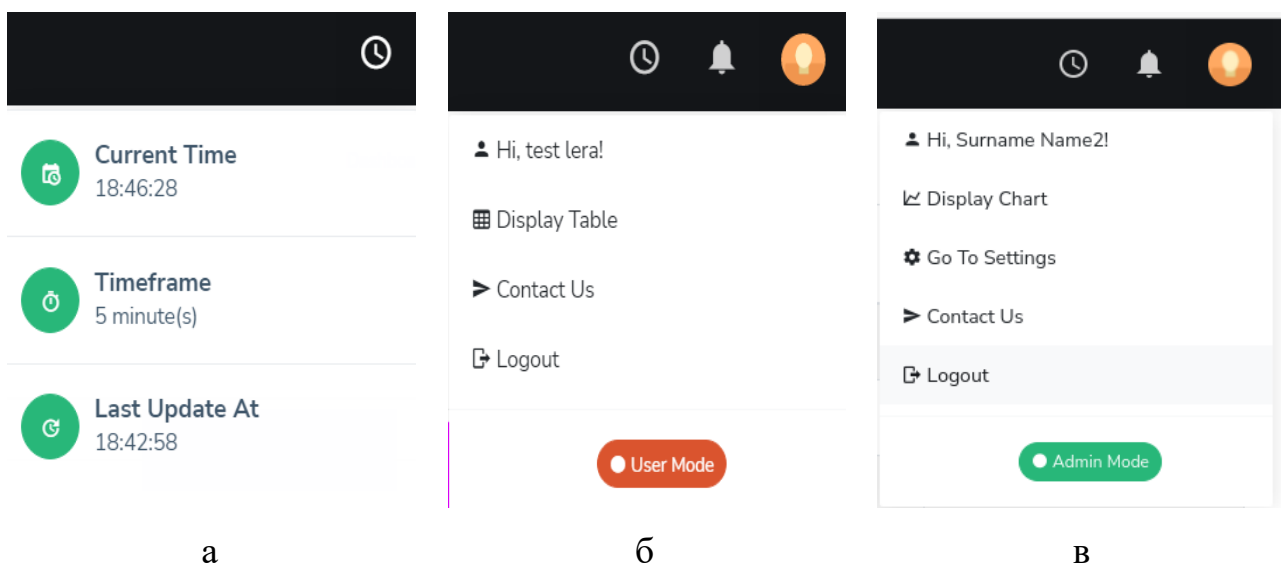


Рисунок 4.6 – Викидні переліки: функція “Clock” (а),
меню користувача в режимі користувача (б) та адміністратора (в)

У разі виникнення питань або проблем у роботі вебсайту, користувач

має можливість звернутися до адміністратора через меню користувача заповнивши форму “Leave Your Feedback” (рис. 4.7).

Рисунок 4.7 – Сторінка для зворотного зв'язку

Для відправки запиту користувач повинен заповнити поля теми (title) і тексту (text), натиснути на кнопку “Send”.

На сторінці “Settings” користувач має можливість змінювати налаштування вебсайту, які приховані від користувачів з низьким рівнем доступу: зміна презентаційних даних, оновлення облікових записів, перегляд та оновлення даних сесій, оновлення налаштувань табличного режиму, оновлення налаштувань графічного режиму, збереження графіків на робочому місці, вибір джерел відображення даних, створення та перегляд запитів отриманих від користувачів, перегляд даних годинника, меню користувача.

Оновлення презентаційних даних доступне у розділі “Application Settings” (рис. 4.8).

Для зміни налаштувань необхідно видалити існуючі та вести нові дані у запропоновані поля. Після зняття фокуса з поля сторінка перезавантажиться і оновлені дані будуть відображені для користувача.

Для оновлення інформації облікових записів користувач повинен перейти до розділу “Accounts Settings” (рис. 4.9).

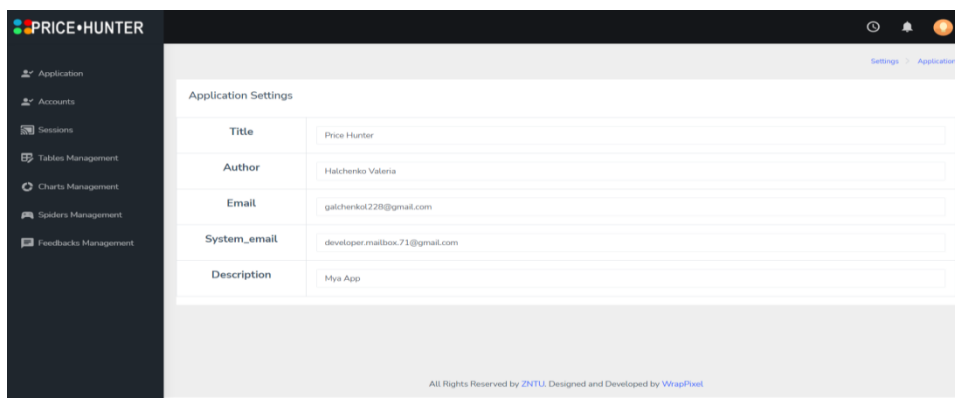


Рисунок 4.8 – Сторінка налаштування презентаційних даних сайту

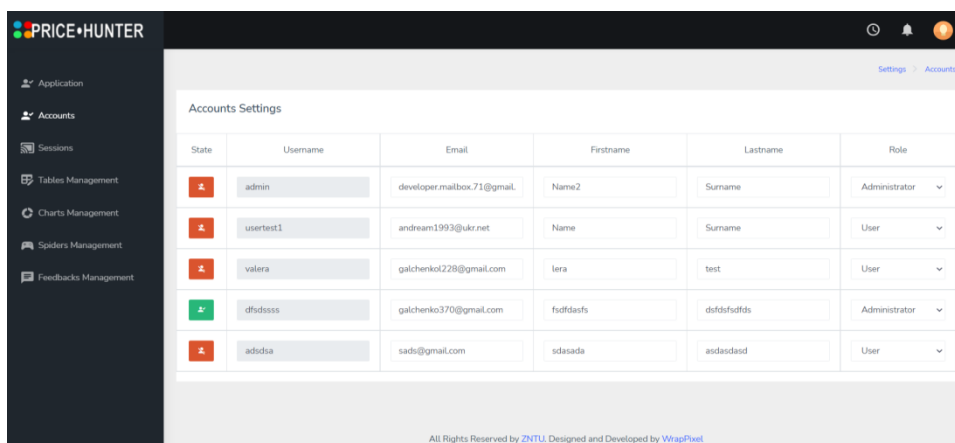


Рисунок 4.9 – Сторінка налаштування презентаційних даних сайту

Для зміни налаштувань необхідно видалити існуючі та вести нові дані у запропоновані поля. Після зняття фокуса з поля сторінка перезавантажиться і оновлені дані будуть відображені для користувача.

Для перегляду даних про активні підключення та їх переривання, користувач повинен відвідати розділ “Sessions Settings” (рис. 4.10).

Для відключення окремих користувачів, необхідно натиснути на кнопку “Disconnect” напроти кожного користувача. Натомість відключення усіх користувачів здійснюється шляхом натиснення кнопки “Disconnect All”.

Для зміни форми представлення даних на дашборді у табличному та графічному режимах (рис. 4.11), користувачеві необхідно перейти до розділів “Tables Management” і “Charts Management” відповідно.

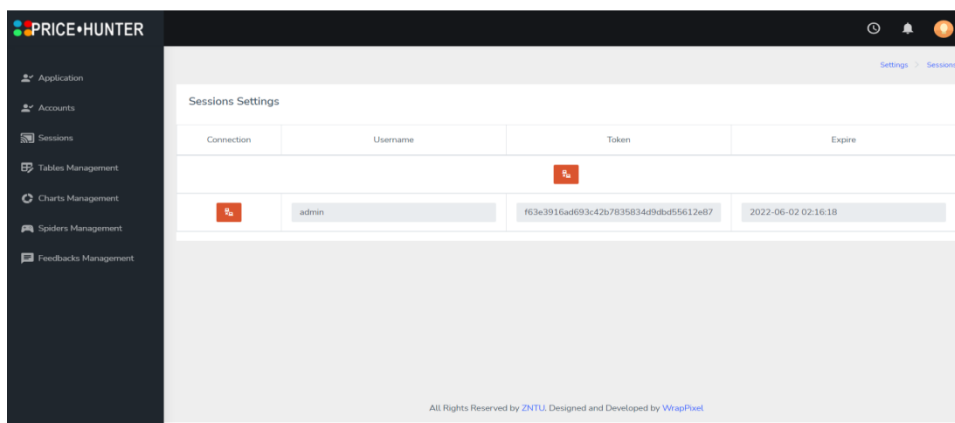
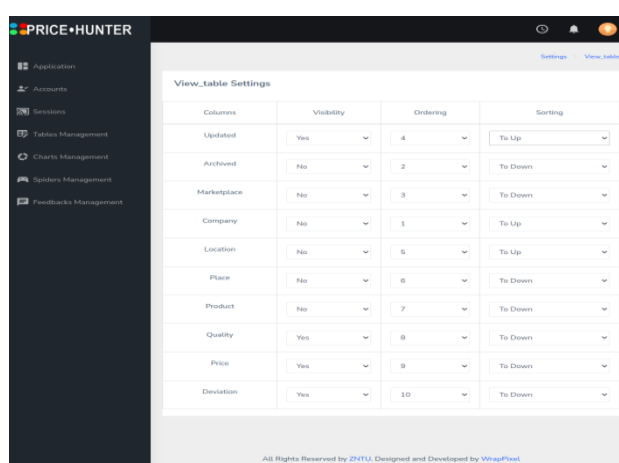
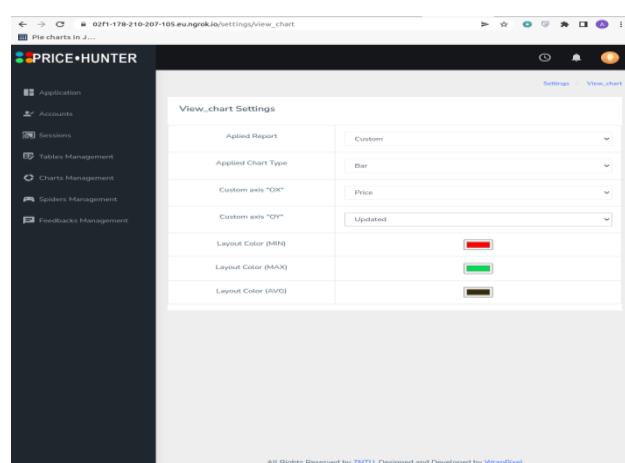


Рисунок 4.10 – Сторінка управління сесіями



а



б

Рисунок 4.11 – Сторінка “Settings”:
розділ “Tables Management” (а) і розділ “Charts Management” (а)

Оновлення налаштувань відбувається одразу після розфокусування поля, з яким взаємодівав користувач. Після автоматичного перезавантаження сторінки, оновлені дані будуть відображені для користувача.

Для управління доступом до даних, які повинні враховуватися при відображенні на дашборді, користувач має можливість налаштувати видимість і порядок відображення стовпців, сортування інформації в них у розділі “Sources Settings” (рис. 4.12).

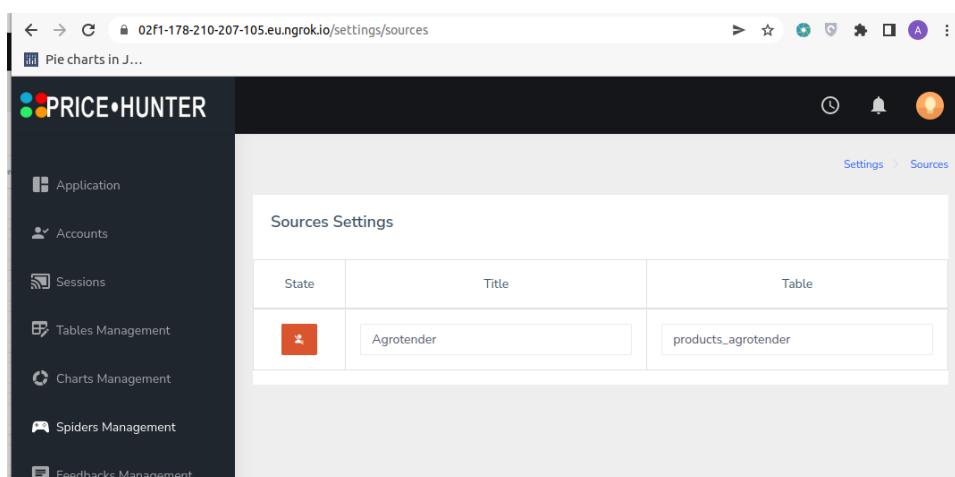


Рисунок 4.12 – Розділ “Sources Settings” сторінки “Settings”

Для ввімкнення/вимкнення доступності окремих таблиць з даними конкурентів, необхідно натиснути на кнопку “Connect” / “Disconnect” напроти кожного джерела даних.

Для перегляду звернень від користувачів вебсайту, адміністратор повинен перейти до розділу “Feedbacks Management” (рис. 4.13).

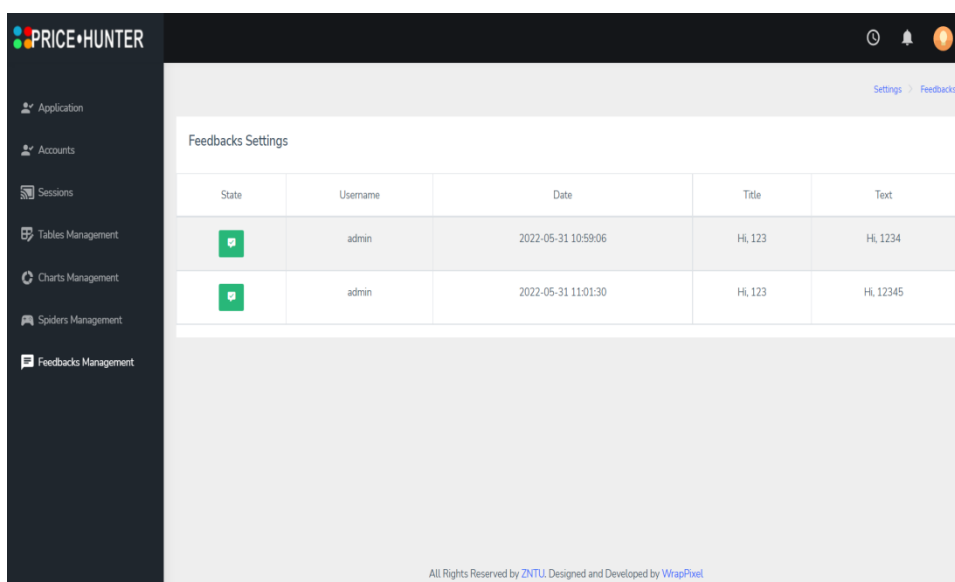


Рисунок 4.13 – Розділ “Sources Settings” сторінки “Settings”

Для зміни стану звернення оброблене/необроблене необхідно натиснути на кнопку “Done” / “New” напроти кожного звернення.

4.3 Висновки за розділом 4

Таким чином у даному розділі виконано огляд функціонала, розробленого у розділі 3, з боку користувача та адміністратора вебсайту, зокрема створено інструкції щодо розгортання серверної частини сайту на новому робочому місці, вимоги до клієнтської частини та методики роботи з вебсайтом.

ВИСНОВКИ

Згідно з завданням, поставленим у кваліфікаційній роботі, створено інформаційну систему “Price Hunter” для отримання інформації щодо цінкових пропозицій на ринку. Для досягнення поставленого завдання виконано наступні кроки:

- виконано огляд аналогів програмного забезпечення, які надають можливість збирати, обробляти і аналізувати інформацію про товари та послуги на будь-якому ринку;
- на основі даних про аналітичні проекти створено технічне завдання з вимогами до інформаційної системи “Price Hunter”;
- обрано найбільш популярні інструменти розробника та методології для реалізації проекту;
- спроектовано та створено базу даних для зберігання даних користувачів, серверу та отриманих з мережі Інтернет даних;
- розроблено макети сторінок вебсайту, алгоритми та процедури обробки даних;
- реалізовано функцію розмежування доступу до дашборду та налаштувань системи;
- створено інструкцію з мінімальним набором інформації необхідним для користування системою.

В результаті створена система дозволяє виконувати наступні дії:

- отримання інформації з мережі Інтернет (тематичні сайти, форуми, маркетплейси та ін.);
- обробка інформації (пошук ключових даних, їх адаптація до формату необхідного для роботи системи);
- накопичення (збереження даних у базі даних);
- графічне представлення оброблених даних.

З метою удосконалення проекту можливо впровадження додаткових функцій, які будуть корисними для клієнтів:

- виведення інформації у файл;
- порівняння даних між різними торговими платформами;
- порівняння даних про товари та послуги з власними;
- створення системи дистанційного сповіщення про зміни цінових пропозицій через месенджер/електронну пошту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Около 40% войн в мире велись за природные ресурсы [Электрон. ресурс] – Режим доступа: aa.com.tr.
2. Пименова Е.М. Критерии оценки эффективности и успешности предпринимательства и малого бизнеса // Креативная экономика. — 2020. — Том 14. — № 10. — С. 2431-2444.
3. Дроздов И. Н. Факторы и критерии успешности предпринимателей малого и среднего бизнеса // Креативная экономика. — 2020. — Том 14. — № 11. — С. 2923-2934.
4. Analysis and design of intelligent systems using soft computing techniques / [eds.: P. Melin, O.R. Castillo, E.G. Ramirez, J. Kacprzyk]. – Heidelberg: Springer, 2007. – 855 p.
5. Intelligent Transformer Monitoring System Utilizing Neuro-Fuzzy Technique Approach : project report (final) : 04-26 / R. Shoureshi, T. Norick, R. Swartzendruber / Colorado School of Mines. – Golden, 2004. – 111 p.
6. Abraham, A. Engineering evolutionary intelligent systems / A. Abraham, C. Grosan, W. Pedrycz. – Berlin: Springer, 2008. – 444 p.
7. Старинский Б. Автоматизация бизнес-процессов [Электрон. ресурс] / Борис Старинский – Режим доступа: <https://finance-abm.com/library/articles/article.php?ID=25566>.
8. Автоматизация бизнес процессов. С чего начать? [Электрон. ресурс]. – 2021. – Режим доступа: <https://www.bizadviser.com.ua/avtomatizacija-biznes-processov-s-chego-nachat/>.
9. Биргер, И. А. Техническая диагностика / И. А. Биргер. – М. : Машиностроение, 1978. – 240 с.
10. Abraham, A. Engineering evolutionary intelligent systems / A. Abraham, C. Grosan, W. Pedrycz. – Berlin: Springer, 2008. – 444 p.
11. Моніторинг цін конкуренті в інтернеті [Електрон. ресурс] – Режим доступу: <https://www.pricecontrol.com.ua/ua/monitoring-czin-konkurentiv-v->

interneti/.

12. Моніторинг Цін Конкуrentів Інтернет Магазинів [Електрон. ресурс] – Режим доступу: <https://www.plerdy.com/ua/blog/monitoring-cen/>.

13. Goldberg K. An Introduction to Python WSGI Servers: Part 1 [Electronic resource] / Kevin Goldberg. – 2016. – Access mode: <https://www.appdynamics.com/blog/engineering/an-introduction-to-python-wsgi-servers-part-1/>.

14. WSGI (pep-333) [Electronic resource]. – Access mode: <https://lectureswww.readthedocs.io/5.web.server/wsgi.html#id8>.

15. Neys X. MySQL/Startup Guide [Electronic resource]/ X. Neys, S. Mani, C. White. – Access mode: https://wiki.gentoo.org/wiki/MySQL/Startup_Guide/ru.

16. Send an E-mail with Python Flask [Electronic resource]. – Access mode: https://pythonbasics-org.translate.goog/flask-mail/?_x_tr_sl=en&_x_tr_tl=uk&_x_tr_hl=uk&_x_tr_pto=sc.

17. Лекція 3. Основні поняття мови HTML та структура документів [Електрон. ресурс] – Режим доступу: <https://sites.google.com/site/vivcaemowebdizajndistancijno/html/lekcia-3-osnovni-ponatta-movi-html-tastruktura-dokumentiv>.

18. Що таке CSS [Електрон. ресурс]. – 2017. – Режим доступу: https://css.in.ua/article/shcho-take-html_10.

19. Петрішенко С. О. Мова JavaScript та її можливості [Електрон. ресурс] / Сергій Олександрович Петрішенко. – 2014. – Режим доступу: <https://sites.google.com/site/webtehnologiietawebdizajn/mova-javascript-ta-ieie-mozlivosti>.

20. jQuery [Електрон. ресурс] – Режим доступу: <https://astwellsoft.com/uk/blog/tehnology/jquery.html>.

21. Komissarenko N. Everything you need to know to build figures in Plotly [Electronic resource] / Nick Komissarenko. – 2020. – Access mode: <https://medium.com/@bigdataschool/все-что-нужно-знать-для-построения-фигур-в-plotly-e3c7eda1bc4c>.

22. Улучшение Visual Studio Code для новичков [Электрон. ресурс] –
Режим доступа: <https://habr.com/ru/post/653465/>.

ДОДАТОК А
Текст програми

config.py

```
class Configs:
    APP_NAME = "Price Hunter"
    # Database Settings;
    DB_NAME = ""
    DB_HOST = "localhost"
    DB_PORT = 3306
    DB_USER = ""
    DB_CODE = ""
    # Mailbox Settings;
    MAIL_DEFAULT_SENDER = ""
    MAIL_PASSWORD = ""
    MAIL_PORT = 587
    MAIL_SERVER = "smtp.gmail.com"
    MAIL_USE_TLS = True
    MAIL_USE_SSL = True
    MAIL_USERNAME = ""

class Development(Configs):
    DEBUG = True
    ENV = "development"
    SECRET_KEY = ""
    TESTING = True

class Production(Configs):
    ENV = "production"
    SECRET_KEY = ""
```

database.py

```
from unittest import result
from pymysql import connect
from pymysql import cursors
from datetime import timedelta
from datetime import datetime
from os import urandom

class Database:
    def __init__(self, configs):
        self.DB_NAME = configs["DB_NAME"]
        self.DB_HOST = configs["DB_HOST"]
        self.DB_PORT = configs["DB_PORT"]
        self.DB_USER = configs["DB_USER"]
        self.DB_CODE = configs["DB_CODE"]
    def connect(self):
        try:
            self.conn = connect(database=self.DB_NAME,
                                host=self.DB_HOST,
                                port=self.DB_PORT,
                                user=self.DB_USER,
                                password=self.DB_CODE)
            self.curs = self.conn.cursor(cursors.DictCursor)
        except:
            self.conn = None
            self.curs = None
    def disconnect(self):
        try:
            if self.conn or self.curs:
                self.curs.close()
                self.conn.close()
        except:
            pass
    def sessions_init(self, username, password):
        try:
            print(username, password)
            sql = "SELECT * FROM `accounts` WHERE `username`=%s AND `password`=%s AND `state`=%s"
            self.curs.execute(sql, (username, password, 1))
            accounts = self.curs.fetchall()
```

```

if not len(accounts):
    return {
        "state": False,
        "info": {
            "title": "Access Restriction",
            "text": "Active account {} has not been found".format(username)
        }
    }

sql = "SELECT * FROM `sessions` WHERE `username`=%s"
self.curs.execute(sql, [(username)])
s_session = self.curs.fetchone()
#print(s_session)
if s_session:
    token = s_session["token"]
    #print(token)
    s_session = self.sessions_validate(token)
    if s_session["state"]:
        return {"state": True, "token": token}
sql = "SELECT * FROM `constants`"
self.curs.execute(sql)
constants = self.curs.fetchone()
if constants is None:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Constants have not been found in database"
        }
    }
#print(constants)
token = urandom(constants["token_size"]).hex()
expire = datetime.now() + timedelta(seconds=constants["deadline_session"])
#print(token, expire)
sql = "INSERT INTO `sessions` (`username`,`token`,`expire`) VALUES (%s,%s,%s)"
self.curs.execute(sql, (username, token, expire.strftime('%Y-%m-%d %H:%M:%S')))
self.conn.commit()
return {"state": True, "token": token}
except:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Unknown Error with New Session Initialization"
        }
    }
}

def sessions_complete(self, token):
    try:
        sql = "DELETE FROM `sessions` WHERE `token`=%s"
        self.curs.execute(sql, [(token)])
        self.conn.commit()
        return {"state": True}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Unknown Error with Session Token"
            }
        }
    }

def sessions_validate(self, token):
    try:
        sql = "SELECT * FROM `sessions` WHERE `token`=%s"
        self.curs.execute(sql, [(token)])
        s_session = self.curs.fetchone()
        if s_session is None:
            return {
                "state": False,
                "info": {
                    "title": "Access Restriction",
                    "text": "Authorize to access that page"
                }
            }

```

```

    }
    }
    if datetime.now() >= s_session["expire"]:
        self.sessions_complete(token)
        return {
            "state": False,
            "info": {
                "title": "Access Restriction",
                "text": "Token has not been expired and removed from database"
            }
        }
    return {"state": True}
except:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Unknown Error with Session Token"
        }
    }
}

def accounts_get(self, username=None, email=None, token=None):
    try:
        account = None
        if username:
            sql = "SELECT * FROM `accounts` WHERE `username`=%s"
            self.curs.execute(sql, [(username)])
            account = self.curs.fetchone()
        elif email:
            sql = "SELECT * FROM `accounts` WHERE `email`=%s"
            self.curs.execute(sql, [(email)])
            account = self.curs.fetchone()
        elif token:
            sql = "SELECT * FROM `sessions` WHERE `token`=%s"
            self.curs.execute(sql, [(token)])
            s_session = self.curs.fetchone()
            if s_session is None:
                pass
            sql = "SELECT * FROM `accounts` WHERE `username`=%s"
            self.curs.execute(sql, [(s_session["username"])])
            account = self.curs.fetchone()
        else:
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "Give any data to search an account"
                }
            }
        if account is None:
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "Account has not been found"
                }
            }
        account.pop("password")
        return {"state": True, "account": account}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Unknown Error with Account Identification"
            }
        }
    }

def accounts_add(self, username, password, email, firstname, lastname):
    try:
        sql = "SELECT * FROM `accounts` WHERE `username`=%s OR `email`=%s"
        self.curs.execute(sql, (username, email))
        accounts = self.curs.fetchall()

```

```

if len(accounts):
    return {
        "state": False,
        "info": {
            "title": "Sign Up Restriction",
            "text": "Give username/email has been registered"
        }
    }
sql = "SELECT * FROM `constants`"
self.curs.execute(sql)
constants = self.curs.fetchone()
if constants is None:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Constants have not been found in database"
        }
    }
token = urandom(constants["token_size"]).hex()
expire = datetime.now() + timedelta(seconds=constants["deadline_activation"])
sql = "SELECT * FROM `accounts` WHERE `role`='admin'"
self.curs.execute(sql)
admins = self.curs.fetchall()
sql = "INSERT INTO `accounts`
(`username`,`password`,`email`,`firstname`,`lastname`,`role`) VALUES (%s,%s,%s,%s,%s,%s)"
if not len(admins):
    self.curs.execute(sql, (username, password, email, firstname, lastname,
"admin"))
else:
    self.curs.execute(sql, (username, password, email, firstname, lastname,
"user"))
sql = "INSERT INTO `requests` (`username`,`token`,`expire`) VALUES (%s,%s,%s)"
self.curs.execute(sql, (username, token, expire.strftime('%Y-%m-%d %H:%M:%S')))
self.conn.commit()
return {"state": True, "token": token}
except:
    return {
        "state": False,
        "info": {
            "title": "Sign Up Rejection",
            "text": "Account has not been created and/or confirmation request has
not been sent"
        }
    }
def accounts_enable(self, token):
    try:
        sql = "SELECT * FROM `requests` WHERE `token`=%s"
        self.curs.execute(sql, [(token)])
        request = self.curs.fetchone()
        if request is None:
            return {
                "state": False,
                "info": {
                    "title": "Activation Error",
                    "text": "Given token has already been activated"
                }
            }
        if datetime.now() >= request["expire"]:
            return {
                "state": False,
                "info": {
                    "title": "Activation Error",
                    "text": "Given token has already been expired. User should reset
password"
                }
            }
        sql = "SELECT * FROM `accounts` WHERE `username`=%s"
        self.curs.execute(sql, [(request["username"])])
        account = self.curs.fetchone()
        if account is None:

```

```

        return {
            "state": False,
            "info": {
                "title": "Activation Error",
                "text": "No any account is related to given token"
            }
        }
    }
    sql = "UPDATE `accounts` SET `state`=%s WHERE `username`=%s"
    self.curs.execute(sql, (1, request["username"]))
    self.conn.commit()
    sql = "SELECT * FROM `accounts` WHERE `username`=%s AND `state`=%s"
    self.curs.execute(sql, (request["username"], 1))
    account = self.curs.fetchone()
    if account is None:
        return {
            "state": False,
            "info": {
                "title": "Activation Error",
                "text": "Account state has not been updated"
            }
        }
    }
    sql = "DELETE FROM `requests` WHERE `token`=%s"
    self.curs.execute(sql, [(token)])
    self.conn.commit()
    return {
        "state": True,
        "info": {
            "title": "Completed Activation",
            "text": "Account has been activated"
        }
    }
except:
    return {
        "state": False,
        "info": {
            "title": "Activation Error",
            "text": "Unknown Error during the account activation"
        }
    }
}

def constants_get(self):
    try:
        sql = "SELECT * FROM `constants`"
        self.curs.execute(sql)
        constants = self.curs.fetchone()
        constants.pop("comments")
        return {"state": True, "constants": constants}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Constants have not been found in database"
            }
        }
    }

def application_get(self):
    try:
        sql = "SELECT * FROM `application`"
        self.curs.execute(sql)
        application = self.curs.fetchone()
        return {"state": True, "application": application}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Application data has not been found in database"
            }
        }
    }

def requests_password_add(self, email):
    try:
        #print(email)

```

```

sql = "SELECT * FROM `accounts` WHERE `email`=%s"
self.curs.execute(sql, [(email)])
account = self.curs.fetchone()
if account is None:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Given email address \"%{}\" has not been found in
database!".format(email)
        }
    }
#print(account)
sql = "SELECT * FROM `constants`"
self.curs.execute(sql)
constants = self.curs.fetchone()
if constants is None:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Constants have not been found in database"
        }
    }
#print(constants)
token = urandom(constants["token_size"]).hex()
expire = datetime.now() + timedelta(seconds=constants["deadline_reset"])
#print(token, expire)
sql = "INSERT INTO `requests` (`username`,`token`,`expire`) VALUES (%s,%s,%s)"
self.curs.execute(sql, (account["username"], token, expire.strftime('%Y-%m-%d
%H:%M:%S'))))
self.conn.commit()
return {"state": True, "firstname": account["firstname"], "token": token}
except:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Request data has not been given from database (for email
address {})".format(email)
        }
    }

def requests_password_delete(self, token, new_password):
    try:
        sql = "SELECT * FROM `requests` WHERE `token`=%s"
        self.curs.execute(sql, [(token)])
        r_request = self.curs.fetchone()
        if r_request is None:
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "Reset token has not been found and request has been
rejected"
                }
            }
        if datetime.now() >= r_request["expire"]:
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "Reset token has been expired and request has been
rejected"
                }
            }
        sql = "SELECT * FROM `accounts` WHERE `username`=%s"
        self.curs.execute(sql, [(r_request["username"])])
        account = self.curs.fetchone()
        if account is None:
            return {
                "state": False,

```

```

        "info": {
            "title": "Database Error",
            "text": "Account related to given token has not been found (for
account <{}>)".format(r_request["username"])
        }
    }
    sql = "UPDATE `accounts` SET `password`=%s WHERE `username`=%s"
    self.curs.execute(sql, (new_password, r_request["username"]))
    self.conn.commit()
    sql = "SELECT * FROM `accounts` WHERE `username`=%s AND `password`=%s"
    self.curs.execute(sql, (r_request["username"], new_password))
    account = self.curs.fetchone()
    if account is None:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "New password has not been saved for account
<{}>".format(r_request["username"])
            }
        }
    sql = "DELETE FROM `requests` WHERE `token`=%s"
    self.curs.execute(sql, [(token)])
    self.conn.commit()
    return {
        "state": True,
        "info": {
            "title": "Successfully Updated",
            "text": "New password has been saved (for account
<{}>)".format(r_request["username"])
        }
    }
except:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Reset password request has not been completed (for account
<{}>)".format(r_request["username"])
        }
    }
}

def notifications_get(self, all=False):
    try:
        if not all:
            sql = "SELECT * FROM `notifications` WHERE `expire`>%s"
            self.curs.execute(sql, [(datetime.now().strftime("%Y-%m-%d %H:%M:%S"))])
        else:
            sql = "SELECT * FROM `notifications`"
            self.curs.execute(sql)
            notifications = self.curs.fetchall()
            #print(notifications)
            if not len(notifications):
                return {"state": True, "notifications": notifications}
            return {"state": True, "notifications": notifications, "fields":
list(notifications[0].keys())}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Notifications have not been found in database"
            }
        }
}

def notifications_update(self):
    pass

def table_get(self, admin_mode = False):
    try:
        sql = "SELECT * FROM `sorts`"
        self.curs.execute(sql)
        sort_modes = self.curs.fetchall()
        sort_modes = { mode["id"]:mode["variable"] for mode in sort_modes }

```

```

# print(sort_modes)
sql = "SELECT * FROM `constants`"
self.curs.execute(sql)
constants = self.curs.fetchone()
if constants is None:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Constants have not been found in database"
        },
        "placeholder": "No items to display"
    }
# print(constants)
sql = "SELECT * FROM `sources` WHERE `state`='1'"
self.curs.execute(sql)
sources = self.curs.fetchall()
if not len(sources):
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "No sources to display data"
        },
        "placeholder": "No items to display"
    }
tables = [source["table"] for source in sources]
# print(tables)
sql = "SELECT * FROM `view_table` WHERE `options`=%s"
self.curs.execute(sql, [("order")])
order = self.curs.fetchone()
order.pop("options")
# order.pop("comments")
order = { int(order[i]):i for i in order }
order = [order[i] for i in sorted(order.keys())]
# print(order)
sql = "SELECT * FROM `view_table` WHERE `options`=%s"
self.curs.execute(sql, [("visibility")])
columns = self.curs.fetchone()
columns.pop("options")
# columns.pop("comments")
columns = [column for column in order if columns[column]=="1"]
# print(columns)
sql = "SELECT * FROM `view_table` WHERE `options`=%s"
self.curs.execute(sql, [("sort")])
sorts = self.curs.fetchone()
sorts = [sort_modes[int(sorts[column])]] for column in columns
# print(sorts)
# print(admin_mode)
if admin_mode:
    sql = "SELECT {} FROM {} {} {}".format(
        "", ".join(["`{}`".format(column) for column in columns]),
        "", ".join(["`{}`".format(table) for table in tables]),
        "ORDER BY {}".format("", ".join(["`{}`".format(columns[i], sorts[i])
        for i in range(len(columns))])),
        "LIMIT {}".format(constants["limit"]) if constants["limit"] else ""
    )
else:
    sql = "SELECT {} FROM {} WHERE {} {} {}".format(
        "", ".join(["`{}`".format(column) for column in columns]),
        "", ".join(["`{}`".format(table) for table in tables]),
        "`archived`='0000-00-00 00:00:00'",
        "ORDER BY {}".format("", ".join(["`{}`".format(columns[i], sorts[i])
        for i in range(len(columns))])),
        "LIMIT {}".format(constants["limit"]) if constants["limit"] else ""
    )
    sql = "SELECT {} FROM {} WHERE {} {} {}".format(
        "", ".join(["`{}`".format(column) for column in columns]),
        "", ".join(["`{}`".format(table) for table in tables]),
        "`archived`='0000-00-00 00:00:00'",
        "ORDER BY {}".format("", ".join(["`{}`".format(columns[i], sorts[i])
        for i in range(len(columns))])),
        "LIMIT {}".format(constants["limit"]) if constants["limit"] else ""
    )

```



```

# layouts;
dataset["layouts"]["quantity"] = sum([report["min"], report["max"], report["avg"]])
# list of tables;
sql = "SELECT `table` FROM `sources` WHERE `state`='1'"
self.curs.execute(sql)
sources = self.curs.fetchall()
tables = [source["table"] for source in sources]
if not len(tables):
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "No data to display has been found in database"
        }
    }
print(report)
# X/Y;
if admin_mode:
    sql = "SELECT `{}``,`{}` FROM {} {}".format(
        report["x_axis"], report["y_axis"],
        ",".join(["`{}`".format(table) for table in tables]),
        "LIMIT {}".format(constants["limit"]) if constants["limit"] else ""
    )
else:
    sql = "SELECT `{}``,`{}` FROM {} WHERE {} {}".format(
        report["x_axis"], report["y_axis"],
        ",".join(["`{}`".format(table) for table in tables]),
        "`archived`='0000-00-00 00:00:00'",
        "LIMIT {}".format(constants["limit"]) if constants["limit"] else ""
    )
self.curs.execute(sql)
rows = self.curs.fetchall()
if not len(rows):
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "No data to display has been found in database"
        }
    }
#print(rows, report["x_axis"], report["y_axis"])
dataset["layouts"]["data"]["x"] = sorted(set([str(row[report["x_axis"]]) for row in
rows]))
dataset["layouts"]["data"]["y"] = {x:[] for x in dataset["layouts"]["data"]["x"]}
for row in rows:
    x = str(row[report["x_axis"]])
    y = row[report["y_axis"]]
    if isinstance(y, datetime):
        dataset["layouts"]["data"]["y"][x].append(y.strftime("%Y-%m-%d %H:%M:%S"))
    elif isinstance(y, str) and y.isnumeric():
        dataset["layouts"]["data"]["y"][x].append(int(y))
    else:
        dataset["layouts"]["data"]["y"][x].append(y)
if dataset["layouts"]["quantity"] > 1:
    dataset["layouts"]["data"]["y"] = {
        "min": [min(dataset["layouts"]["data"]["y"][x]) for x in
dataset["layouts"]["data"]["x"]],
        "max": [max(dataset["layouts"]["data"]["y"][x]) for x in
dataset["layouts"]["data"]["x"]],
        "avg":
[sum(dataset["layouts"]["data"]["y"][x])/len(dataset["layouts"]["data"]["y"][x]) for x in
dataset["layouts"]["data"]["x"]]}
    else:
        dataset["layouts"]["data"]["y"] = [len(dataset["layouts"]["data"]["y"][x]) for
x in dataset["layouts"]["data"]["x"]]
# axis MIN/MAX;
dataset["layouts"]["x-axis"]["title"] = report["x_axis"].capitalize()
dataset["layouts"]["y-axis"]["title"] = report["y_axis"].capitalize()
dataset["layouts"]["x-axis"]["min"], dataset["layouts"]["x-axis"]["max"] =
dataset["layouts"]["data"]["x"][0], dataset["layouts"]["data"]["x"][-1]
if dataset["layouts"]["quantity"] > 1:

```

```

        y_min = min([min(dataset["layouts"]["data"]["y"]["min"]),
min(dataset["layouts"]["data"]["y"]["max"]), min(dataset["layouts"]["data"]["y"]["avg"])]))
        y_max = max([max(dataset["layouts"]["data"]["y"]["min"]),
max(dataset["layouts"]["data"]["y"]["max"]), max(dataset["layouts"]["data"]["y"]["avg"])]))
        dataset["layouts"]["y-axis"]["min"] = y_min if dataset["layouts"]["y-
axis"]["min"] > y_min else dataset["layouts"]["y-axis"]["min"]
        dataset["layouts"]["y-axis"]["max"] = y_max if dataset["layouts"]["y-
axis"]["max"] > y_max else dataset["layouts"]["y-axis"]["max"]
        dataset["layouts"]["y-axis"]["colors"] = {
            "min": view_chart["source_color_1"],
            "max": view_chart["source_color_2"],
            "avg": view_chart["source_color_3"]
        }
    else:
        y_min = min(dataset["layouts"]["data"]["y"])
        y_max = max(dataset["layouts"]["data"]["y"])
        dataset["layouts"]["y-axis"]["min"] = y_min if dataset["layouts"]["y-
axis"]["min"] > y_min else dataset["layouts"]["y-axis"]["min"]
        dataset["layouts"]["y-axis"]["max"] = y_max if y_max > dataset["layouts"]["y-
axis"]["max"] else dataset["layouts"]["y-axis"]["max"]
        dataset["layouts"]["y-axis"]["color"] = view_chart["source_color_1"]
    print(dataset)
    sql = "SELECT `x_axis` FROM `reports`"
    self.curs.execute(sql)
    x_axis = self.curs.fetchall()
    ["updated", "archived", "company", "location", "place", "product", "quality", "price", "deviation"] =
    y_axis = self.curs.fetchall()
    ["updated", "archived", "company", "location", "place", "product", "quality", "price", "deviation"] =
    print(dataset["custom"])
    return {"state": True, "dataset": dataset, "custom": {"x_axis": x_axis, "y_axis":
y_axis, "x": view_chart["x_axis"], "y": view_chart["y_axis"]}}
    def limits_get(self):
        try:
            sql = "SELECT * FROM `limits`"
            self.curs.execute(sql)
            limits = self.curs.fetchall()
            if not len(limits):
                return {
                    "state": False,
                    "info": {
                        "title": "Database Error",
                        "text": "List of Limits is empty"
                    }
                }
            return {"state": True, "limits": limits}
        except:
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "Limits have not been found in database"
                }
            }
    def limits_update(self, limit):
        try:
            sql = "UPDATE `constants` SET `limit`=%s"
            self.curs.execute(sql, [(limit)])
            self.conn.commit()
            return {"state": True}
        except:
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "Current Limit has not been updated in database"
                }
            }
    def timeframes_get(self):
        try:
            sql = "SELECT * FROM `timeframes`"
            self.curs.execute(sql)

```

```

timeframes = self.curs.fetchall()
if not len(timeframes):
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "List of timeframes is empty"
        }
    }
    return {"state": True, "timeframes": timeframes}
except:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Timeframes have not been found in database"
        }
    }
def timeframes_update(self, timeframe):
    try:
        sql = "UPDATE `constants` SET `timeframe`=%s"
        self.curs.execute(sql, [(timeframe)])
        self.conn.commit()
        return {"state": True}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Current timeframe has not been updated in database"
            }
        }
def modes_get(self):
    try:
        sql = "SELECT * FROM `modes`"
        self.curs.execute(sql)
        modes = self.curs.fetchall()
        if not len(modes):
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "List of modes is empty"
                }
            }
        return {"state": True, "modes": modes}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Modes have not been found in database"
            }
        }
def modes_update(self, mode):
    try:
        sql = "UPDATE `constants` SET `mode`=%s"
        self.curs.execute(sql, [(mode)])
        self.conn.commit()
        return {"state": True}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Current mode has not been updated in database"
            }
        }
def charts_get(self):
    try:
        sql = "SELECT * FROM `charts`"

```

```

self.curs.execute(sql)
charts = self.curs.fetchall()
if not len(charts):
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "List of chart types is empty"
        }
    }
    return {"state": True, "charts": charts}
except:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Chart types have not been found in database"
        }
    }
def charts_update(self, chart):
    try:
        sql = "SELECT * FROM `charts` WHERE `variable`=%s"
        self.curs.execute(sql, [(chart)])
        chart_id = self.curs.fetchone()
        if chart_id is None:
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "Selected chart type has not been found in database"
                }
            }
        sql = "UPDATE `view_chart` SET `chart_id`=%s"
        self.curs.execute(sql, [(chart_id["id"])])
        self.conn.commit()
        return {"state": True}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Current chart type has not been updated in database"
            }
        }
def reports_get(self):
    try:
        sql = "SELECT * FROM `reports`"
        self.curs.execute(sql)
        reports = self.curs.fetchall()
        if not len(reports):
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "List of reports is empty"
                }
            }
        return {"state": True, "reports": reports}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Reports have not been found in database"
            }
        }
def reports_update(self, report, custom=False):
    try:
        sql = "SELECT * FROM `reports` WHERE `x_axis`=%s AND `y_axis`=%s"
        self.curs.execute(sql, (report[0], report[1]))
        report = self.curs.fetchall()

```

```

        if not len(report):
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "Selected report has not been found in database"
                }
            }
        sql = "UPDATE `view_chart` SET `report_id`=%s"
        if custom:
            self.curs.execute(sql, [(report[0]["id"])]))
        else:
            self.curs.execute(sql, [(report[-1]["id"])]))
        self.conn.commit()
        return {"state": True}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Current chart\'s report has not been updated in database"
            }
        }
def sessions_get(self):
    try:
        sql = "SELECT * FROM `sessions`"
        self.curs.execute(sql)
        sessions = self.curs.fetchall()
        if not len(sessions):
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "List of sessions is empty"
                }
            }
        return {"state": True, "sessions": sessions}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Sessions have not been found in database"
            }
        }
def sources_get(self):
    try:
        sql = "SELECT * FROM `sessions`"
        self.curs.execute(sql)
        sessions = self.curs.fetchall()
        if not len(sessions):
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "List of sessions is empty"
                }
            }
        return {"state": True, "sessions": sessions}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Sessions have not been found in database"
            }
        }
def spiders_get(self):
    try:
        sql = "SELECT * FROM `spiders`"
        self.curs.execute(sql)

```

```

        spiders = self.curs.fetchall()
        if not len(spiders):
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "List of spiders is empty"
                }
            }
        return {"state": True, "spiders": spiders}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Spiders have not been found"
            }
        }
def view_tables_get(self):
    try:
        sql = "SELECT * FROM `view_table`"
        self.curs.execute(sql)
        view_table = self.curs.fetchall()
        if not len(view_table):
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "List of table settings is empty"
                }
            }
        return {"state": True, "view_table": view_table}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Table settings have not been found in database"
            }
        }
def view_charts_get(self):
    try:
        sql = "SELECT * FROM `view_chart`"
        self.curs.execute(sql)
        view_chart = self.curs.fetchone()
        if view_chart is None:
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "List of chart settings is empty"
                }
            }
        return {"state": True, "view_chart": view_chart}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Chart settings have not been found in database"
            }
        }
def requests_get(self):
    try:
        sql = "SELECT * FROM `requests`"
        self.curs.execute(sql)
        requests = self.curs.fetchall()
        if not len(requests):
            return {
                "state": False,
                "info": {

```

```

        "title": "Database Error",
        "text": "List of requests is empty"
    }
}
return {"state": True, "requests": requests}
except:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Requests have not been found in database"
        }
    }
def axis_update(self, axis, title):
    print(axis, title)
    if axis == "x":
        sql = "UPDATE `view_chart` SET `x_axis`=%s"
        self.curs.execute(sql, [(title)])
        sql = "UPDATE `reports` SET `x_axis`=%s WHERE `id`=%s"
        self.curs.execute(sql, (title, 0))
    else:
        sql = "UPDATE `view_chart` SET `y_axis`=%s"
        self.curs.execute(sql, [(title)])
        sql = "UPDATE `reports` SET `y_axis`=%s WHERE `id`=%s"
        self.curs.execute(sql, (title, 0))
    self.conn.commit()
    sql = "UPDATE `view_chart` SET `report_id`=%s"
    self.curs.execute(sql, [(0)])
    self.conn.commit()
    return {"state": True}

    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Axis has not been found in database"
        }
    }
}
def settings_get(self, table):
    support={}
    if table == "application":
        sql = "SELECT * FROM `application`"
        self.curs.execute(sql)
        settings = self.curs.fetchone()
        if settings is None:
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "List of application data is empty"
                }
            }
    elif table == "notifications":
        pass
    elif table == "sessions":
        sql = "SELECT * FROM `sessions`"
        self.curs.execute(sql)
        settings = self.curs.fetchall()
        if not len(settings):
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "List of sessions is empty"
                }
            }
    }
    for i in range(len(settings)):
        settings[i].pop("comments")
    elif table == "accounts":
        sql = "SELECT * FROM `accounts`"

```

```

self.curs.execute(sql)
settings = self.curs.fetchall()
if not len(settings):
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "List of accounts is empty"
        }
    }
}
for i in range(len(settings)):
    settings[i].pop("password")
print(settings)
sql = "SELECT * FROM `roles`"
self.curs.execute(sql)
support["roles"] = self.curs.fetchall()
elif table == "sources":
    sql = "SELECT * FROM `sources`"
    self.curs.execute(sql)
    settings = self.curs.fetchall()
    if not len(settings):
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "List of sources is empty"
            }
        }
    }
for i in range(len(settings)):
    settings[i].pop("comments")
elif table == "view_table":
    sql = "SELECT * FROM `view_table`"
    self.curs.execute(sql)
    settings = self.curs.fetchall()
    if not len(settings):
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "List of view table is empty"
            }
        }
    }
support["order"] = list(range(1, len(settings[0].keys())))
settings = {
    settings[0]["options"]: {i: settings[0][i] for i in settings[0] if i !=
"options"},
    settings[1]["options"]: {i: settings[1][i] for i in settings[1] if i !=
"options"},
    settings[2]["options"]: {i: settings[2][i] for i in settings[2] if i !=
"options"}
}
sql = "SELECT * FROM `visibility`"
self.curs.execute(sql)
support["visibility"] = self.curs.fetchall()
if not len(support["visibility"]):
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "List of view table visibility is empty"
        }
    }
}
sql = "SELECT * FROM `sorts`"
self.curs.execute(sql)
support["sort"] = self.curs.fetchall()
if not len(support["sort"]):
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "List of view table visibility sorts is empty"
        }
    }
}

```

```

    }
}
elif table == "view_chart":
    sql = "SELECT * FROM `view_chart`"
    self.curs.execute(sql)
    settings = self.curs.fetchone()
    if settings is None:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "List of view chart data is empty"
            }
        }
    sql = "SELECT * FROM `reports`"
    self.curs.execute(sql)
    support["reports"] = self.curs.fetchall()
    sql = "SELECT * FROM `charts`"
    self.curs.execute(sql)
    support["charts"] = self.curs.fetchall()
    sql = "SELECT `x_axis` FROM `reports`"
    self.curs.execute(sql)
    support["x_axis"] =
["updated", "archived", "company", "location", "place", "product", "quality", "price", "deviation"]
    support["y_axis"] =
["updated", "archived", "company", "location", "place", "product", "quality", "price", "deviation"]
    elif table == "requests":
        pass
    elif table == "reports":
        pass
    elif table == "feedbacks":
        sql = "SELECT * FROM `feedbacks`"
        self.curs.execute(sql)
        settings = self.curs.fetchall()
        if not len(settings):
            return {
                "state": False,
                "info": {
                    "title": "Database Error",
                    "text": "List of feedbacks is empty"
                }
            }
        for i in range(len(settings)):
            if isinstance(settings[i]["date"], datetime):
                settings[i]["date"] = settings[i]["date"].strftime("%Y-%m-%d
%H:%M:%S")
        sql = "SHOW COLUMNS FROM `{}`".format(table)
        self.curs.execute(sql)
        fields = self.curs.fetchall()
        print(fields)
        return {"state": True, table: settings, "fields": fields, "support": support}

    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Table {} has not been found in
database".format(table.capitalize())
        }
    }
}

def settings_update(self, table=None, field=None, value=None, primary=None, key=None):

    if table == "application":
        sql = "UPDATE `{}` SET `{}`=%s".format(table, field)
        self.curs.execute(sql, [(value)])
    elif table == "notifications":
        pass
    elif table == "sessions":
        if primary == "-" and key == "-":
            sql = "DELETE FROM `{}` WHERE `{}`=%s".format(table, primary)
            self.curs.execute(sql, [(key)])

```

```

        else:
            sql = "DELETE FROM `{}` WHERE 1".format(table)
            self.curs.execute(sql)
        elif table == "accounts":
            sql = "UPDATE `{}` SET `{}`=%s WHERE `{}`=%s".format(table, field, primary)
            self.curs.execute(sql, (value, key))
        elif table == "sources":
            sql = "UPDATE `{}` SET `{}`=%s WHERE `{}`=%s".format(table, field, primary)
            self.curs.execute(sql, (value, key))
        elif table == "spiders":
            pass
        elif table == "view_table":
            if key == "order":
                sql = "SELECT * FROM `{}` WHERE `{}`=%s".format(table, primary)
                self.curs.execute(sql, [(key)])
                result = self.curs.fetchone()
                old_field = field
                old_index = result[field]
                result = {i:result[i] for i in result if result[i] == value}
                new_field = list(result.items())[0][0]
                new_index = list(result.items())[0][1]
                sql = "UPDATE `{}` SET `{}`=%s WHERE `{}`=%s".format(table, old_field,
primary)

                self.curs.execute(sql, (new_index, key))
                sql = "UPDATE `{}` SET `{}`=%s WHERE `{}`=%s".format(table, new_field,
primary)

                self.curs.execute(sql, (old_index, key))
            else:
                sql = "UPDATE `{}` SET `{}`=%s WHERE `{}`=%s".format(table, field,
primary)

                self.curs.execute(sql, (value, key))
        elif table == "view_chart":
            if field == "x_axis":
                sql = "UPDATE `{}` SET `{}`=%s".format(table, field)
                self.curs.execute(sql, [(value)])
                sql = "UPDATE `reports` SET `x_axis`=%s WHERE `id`=%s"
                self.curs.execute(sql, (value, 0))
            if field == "y_axis":
                sql = "UPDATE `{}` SET `{}`=%s".format(table, field)
                self.curs.execute(sql, [(value)])
                sql = "UPDATE `reports` SET `y_axis`=%s WHERE `id`=%s"
                self.curs.execute(sql, (value, 0))
            if field == "report_id":
                sql = "SELECT * FROM `reports` WHERE `id`=%s"
                self.curs.execute(sql, [(value)])
                axis = self.curs.fetchone()
                sql = "UPDATE `{}` SET `{}`=%s, `x_axis`=%s, `y_axis`=%s".format(table,
field)

                self.curs.execute(sql, (value, axis["x_axis"], axis["y_axis"]))
            else:
                sql = "UPDATE `{}` SET `{}`=%s".format(table, field)
                self.curs.execute(sql, [(value)])
        elif table == "requests":
            pass
        elif table == "reports":
            pass
        elif table == "feedbacks":
            sql = "UPDATE `{}` SET `{}`=%s WHERE `{}`=%s".format(table, field, primary)
            self.curs.execute(sql, (value, key))
        self.conn.commit()
        return {"state": True}

    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Table {} has not been updated in
database".format(table.capitalize())
        }
    }

def feedbacks_get(self):

```

```

try:
    sql = "SELECT * FROM `feedbacks`"
    self.curs.execute(sql)
    feedbacks = self.curs.fetchall()
    if not len(feedbacks):
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "List of feedbacks is empty"
            }
        }
    for i in range(len(feedbacks)):
        if isinstance(feedbacks[i]["date"], datetime):
            feedbacks[i]["date"] = feedbacks[i]["date"].strftime("%Y-%m-%d
%H:%M:%S")
    sql = "SHOW COLUMNS FROM `feedbacks`"
    self.curs.execute(sql)
    fields = self.curs.fetchall()
    return {"state": True, "feedbacks": feedbacks, "fields": fields}
except:
    return {
        "state": False,
        "info": {
            "title": "Database Error",
            "text": "Feedbacks have not been found in database"
        }
    }

def feedbacks_add(self, username=None, title=None, text=None):
    try:
        sql = "INSERT INTO `feedbacks` (`username`,`date`,`title`,`text`) VALUES
(%s,%s,%s,%s)"
        self.curs.execute(sql, (username, datetime.now().strftime("%Y-%m-%d
%H:%M:%S"), title, text))
        self.conn.commit()
        return {"state": True}
    except:
        return {
            "state": False,
            "info": {
                "title": "Database Error",
                "text": "Your feedback has not been saved in database"
            }
        }

```

mailbox.py

```

from email_validator import validate_email
from flask_mail import Mail
from flask_mail import Message
import smtplib
import email.message

class Mailbox:
    def __init__(self, app):
        self.settings = app.config
        self.receiver = None
        self.server = smtplib.SMTP(":".join([self.settings["MAIL_SERVER"],
str(self.settings["MAIL_PORT"])]))
        self.message = email.message.Message()
        self.MAIL_RECEIVER = None
    def create_message_new_member(self, appname, receiver, url, deadline):
        self.receiver = receiver
        self.message['Subject'] = "{} - New Member".format(appname)
        self.message['From'] = self.settings["MAIL_DEFAULT_SENDER"]
        self.message['To'] = receiver
        self.message.add_header('Content-Type', 'text/html')
        self.message.set_payload("""
            <h1>New member acception</h1>
            <p>Navigate by URL to activate member: {}.</p>

```

```

                                <p>Otherwise ignore this mail! URL will be expired in {}
hours!</p>
                                """.format(url, deadline))
    def create_message_reset_password(self, appname, receiver, firstname, url, deadline):
        self.receiver = receiver
        self.message['Subject'] = "{} - Reset Password".format(appname)
        self.message['From'] = self.settings["MAIL_DEFAULT_SENDER"]
        self.message['To'] = receiver
        self.message.add_header('Content-Type', 'text/html')
        self.message.set_payload("""
                                <h1>Dear, {}!</h1>
                                <p>Navigate by URL to reset password: {}.</p>
                                <p>Otherwise ignore this mail! URL will be expired in {}
hours!</p>
                                """.format(firstname, url, deadline))

    def send(self):
        try:
            self.mailbox = smtplib.SMTP(":".join([self.settings["MAIL_SERVER"],
str(self.settings["MAIL_PORT"])]))
            self.mailbox.starttls()
            self.mailbox.login(self.settings["MAIL_DEFAULT_SENDER"],
self.settings["MAIL_PASSWORD"])
            self.mailbox.sendmail(self.settings["MAIL_DEFAULT_SENDER"], [self.receiver],
self.message.as_string())
            return {
                "state": True,
                "info": {
                    "title": "Successfully Sent",
                    "text": "Your request will be processed by administrator soon"
                }
            }
        except:
            return {
                "state": False,
                "info": {
                    "title": "Request Rejection",
                    "text": "Your request has not been sent to administrator"
                }
            }

```

security.py

```

from hashlib import md5

def string_safe(string=None):
    try:
        if isinstance(string, str):
            return {"state": True, "hash": md5(string.encode()).hexdigest()}
        return {
            "state": False,
            "title": "Security Error",
            "text": "Invalid Type of Entry Value"
        }
    except:
        return {
            "state": False,
            "title": "Security Error",
            "text": "Unknown Error with Entry Value"
        }

```

validator.py

```

from email_validator import validate_email
from json import load
from re import match

class Validator:
    def __init__(self):
        self.regexp_firstname = r'^[A-Za-z]{2,25}$'

```

```

self.regex_lastname = r'^[A-Za-z]{2,25}$'
self.regex_username = r'^[A-Za-z0-9]{5,25}$'
self.regex_password = r'^[A-Za-z0-9]{8,}$'
self.regex_feedback_title = r'^[\w\d\s,\.]{5,255}$'
self.regex_feedback_text = r'^[\w\d\s,\.]{5,255}$'
def validate_firstname(self, firstname):
    try:
        if match(self.regex_firstname, firstname) is None:
            return {
                "state": False,
                "info": {
                    "title": "Validation Error",
                    "text": "Firstname should meet with expression:
{}".format(self.regex_firstname)
                }
            }
        return {"state": True, "firstname": firstname}
    except:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Unknown Error with Firstname"
            }
        }

def validate_lastname(self, lastname):
    try:
        if match(self.regex_lastname, lastname) is None:
            return {
                "state": False,
                "info": {
                    "title": "Validation Error",
                    "text": "Lastname should meet with expression:
{}".format(self.regex_lastname)
                }
            }
        return {"state": True, "lastname": lastname}
    except:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Unknown Error with Lastname"
            }
        }

def validate_username(self, username):
    try:
        if match(self.regex_username, username) is None:
            return {
                "state": False,
                "info": {
                    "title": "Validation Error",
                    "text": "Username should meet with expression:
{}".format(self.regex_username)
                }
            }
        return {"state": True, "username": username}
    except:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Unknown Error with Username"
            }
        }

def validate_password(self, password):
    try:
        if match(self.regex_password, password) is None:
            return {
                "state": False,
                "info": {

```

```

        "title": "Validation Error",
        "text": "Password should meet with expression:
{}".format(self.regex_password)
    }
    }
    return {"state": True, "password": password}
except:
    return {
        "state": False,
        "info": {
            "title": "Validation Error",
            "text": "Unknown Error with Password"
        }
    }
}

def validate_password_x2(self, password1, password2):
    try:
        if password1 != password2:
            return {
                "state": False,
                "info": {
                    "title": "Validation Error",
                    "text": "Password and Confirmation Password should be same"
                }
            }
        return {"state": True}
    except:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Unknown Error with Password/confirmation Password"
            }
        }
    }

def validate_email(self, email):
    try:
        validate_email(email)
        return {"state": True, "email": email}
    except:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Email Address is invalid"
            }
        }
    }

def validate_limit(self, limit):
    try:
        if isinstance(limit, int) and limit >= 0:
            return {"state": True, "limit": limit}
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Liimit value is invalid"
            }
        }
    except:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Email Address is invalid"
            }
        }
    }

def validate_timeframe(self, timeframe):
    try:
        if isinstance(timeframe, int) and 0 <= timeframe <= 525600:
            return {"state": True, "timeframe": timeframe}
        return {
            "state": False,
            "info": {

```

```

        "title": "Validation Error",
        "text": "Timeframe value is invalid"
    }
}
except:
    return {
        "state": False,
        "info": {
            "title": "Validation Error",
            "text": "Timeframe is invalid"
        }
    }
}
def validate_mode(self, mode):
    try:
        if mode in ("table", "chart"):
            return {"state": True, "mode": mode}
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Mode value is invalid"
            }
        }
    except:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Mode is invalid"
            }
        }
    }
def validate_chart(self, chart):
    try:
        if chart in ("line", "bar", "pie"):
            return {"state": True, "chart": chart}
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Chart type is invalid"
            }
        }
    except:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Mode is invalid"
            }
        }
    }
def validate_report(self, report):
    try:
        report = [axis.strip() for axis in report.split("_") if len(axis.strip())]
        if len(report) == 2:
            return {"state": True, "report": report}
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Selected report ID is invalid"
            }
        }
    except:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Mode is invalid"
            }
        }
    }
def validate_feedback_title(self, title):

```

```

try:
    if match(self.regex_feedback_title, title) is None:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Feedback title should meet with expression:
{}".format(self.regex_feedback_title)
            }
        }
    return {"state": True, "title": title}
except:
    return {
        "state": False,
        "info": {
            "title": "Validation Error",
            "text": "Unknown Error with feedback title"
        }
    }

def validate_feedback_text(self, text):
    try:
        if match(self.regex_feedback_text, text) is None:
            return {
                "state": False,
                "info": {
                    "title": "Validation Error",
                    "text": "Feedback text should meet with expression:
{}".format(self.regex_feedback_text)
                }
            }
        return {"state": True, "text": text}
    except:
        return {
            "state": False,
            "info": {
                "title": "Validation Error",
                "text": "Unknown Error with feedback text"
            }
        }

```

main.py

```

from flask import Flask
from flask import flash
from flask import render_template
from flask import request
from flask import session
from flask import redirect
from flask import abort
from datetime import datetime
from functools import wraps
from configs import *
from database import *
from mailbox2 import *
from security import *
from validator import *
import logging, os

```

application

```

app = Flask(__name__)
app.config.from_object(Development)
logging.basicConfig(filename="{}/log".format(datetime.now().strftime("%d-%m-%Y")),
                    level=logging.DEBUG, format=f'%(asctime)s %(levelname)s %(name)s %(threadName)s :
%(message)s')

```

modules

```

validator = Validator()
db = Database(app.config)
mailbox = Mailbox(app)

```

errors

```
@app.errorhandler(403)
@app.errorhandler(404)
@app.errorhandler(500)
def view_errors(error):
    if request.url_rule:
        app.logger.info("; ".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip(), str(error.code)]))
    else:
        app.logger.info("; ".join([request.remote_addr.strip(), str(error.code)]))
    return render_template("error.html", e=error)
```

access control

```
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        db.connect()
        state = db.sessions_validate(session.get("token", None))
        print(state)
        if not state["state"]:
            db.disconnect()
            flash(state["info"], "danger")
            app.logger.info("; ".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip(), state["info"]["title"],
state["info"]["text"]]))
            return redirect("/authentication/login")
        return f(*args, **kwargs)
    return decorated_function

def login_not_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        db.connect()
        state = db.sessions_validate(session.get("token", None))
        print(state)
        if state["state"]:
            db.disconnect()
            app.logger.info("; ".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip()]))
            return redirect("/")
        return f(*args, **kwargs)
    return decorated_function
```

authentication

```
@app.route("/authentication/signup")
@login_not_required
def view_signup():
    try:
        return render_template("authentication/signup.html")
    except:
        abort(404)

@app.route("/authentication/login")
@login_not_required
def view_login():
    try:
        return render_template("authentication/login.html")
    except:
        abort(404)

@app.route("/authentication/password/reset")
@login_not_required
def view_password_reset():
    try:
        return render_template("authentication/password/reset.html")
    except:
        abort(404)

@app.route("/authentication/password/setup/<string:token>")
@login_not_required
```

```

def view_password_setup(token):
    try:
        return render_template("authentication/password/setup.html", token=token)
    except:
        abort(404)
@app.route("/authentication/signup/send", methods=["POST"])
@login_not_required
def controller_signup():
    try:
        firstname = validator.validate_firstname(request.form.get("firstname", ""))
        if not firstname["state"]:
            flash(firstname["info"], "danger")
            app.logger.info(";".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip(),
firstname["info"]["title"], firstname["info"]["text"]]))
            return redirect(request.referrer)
        lastname = validator.validate_lastname(request.form.get("lastname", ""))
        if not lastname["state"]:
            flash(lastname["info"], "danger")
            app.logger.info(";".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip(),
lastname["info"]["title"], lastname["info"]["text"]]))
            return redirect(request.referrer)
        username = validator.validate_username(request.form.get("username", ""))
        if not username["state"]:
            flash(username["info"], "danger")
            app.logger.info(";".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip(),
username["info"]["title"], username["info"]["text"]]))
            return redirect(request.referrer)
        email = validator.validate_email(request.form.get("email", ""))
        if not email["state"]:
            flash(email["info"], "danger")
            app.logger.info(";".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip(), email["info"]["title"],
email["info"]["text"]]))
            return redirect(request.referrer)
        password = validator.validate_password(request.form.get("password", ""))
        if not password["state"]:
            flash(password["info"], "danger")
            app.logger.info(";".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip(),
password["info"]["title"], password["info"]["text"]]))
            return redirect(request.referrer)
        confirmation_password = validator.validate_password_x2(password["password"],
request.form.get("confirmation_password", ""))
        if not confirmation_password["state"]:
            flash(confirmation_password["info"], "danger")
            app.logger.info(";".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip(),
confirmation_password["info"]["title"], confirmation_password["info"]["text"]]))
            return redirect(request.referrer)
        password = string_safe(password["password"])
        db.connect()
        constants = db.constants_get()
        if not constants["state"]:
            flash(constants["info"], "danger")
            app.logger.info(";".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip(),
constants["info"]["title"], constants["info"]["text"]]))
            db.disconnect()
            return redirect(request.referrer)
        application = db.application_get()
        if not application["state"]:
            flash(application["info"], "danger")
            app.logger.info(";".join([request.remote_addr.strip(),
request.url_rule.rule.strip(), request.url_rule.endpoint.strip(),
application["info"]["title"], application["info"]["text"]]))
            db.disconnect()
            return redirect(request.referrer)

```

```

        #print(username["username"], password["hash"], email["email"],
        firstname["firstname"], lastname["lastname"])
        token = db.accounts_add(username["username"], password["hash"], email["email"],
        firstname["firstname"], lastname["lastname"])
        if not token["state"]:
            flash(token["info"], "danger")
            app.logger.info("; ".join([request.remote_addr.strip(),
            request.url_rule.rule.strip(), request.url_rule.endpoint.strip(), token["info"]["title"],
            token["info"]["text"]]))
            return redirect(request.referrer)
            db.disconnect()
            mailbox.create_message_new_member(application["application"]["title"],
            app.config["MAIL_USERNAME"],

"{}/authentication/password/activate/{}".format(request.host_url, token["token"]),
            constants["constants"]["deadline_activation"] //
3600)
        #print("{}/authentication/password/setup/{}".format(request.host_url,
        token["token"]))
        message = mailbox.send()
        if not message["state"]:
            flash(message["info"], "danger")
            app.logger.info("; ".join([request.remote_addr.strip(),
            request.url_rule.rule.strip(), request.url_rule.endpoint.strip(), message["info"]["title"],
            message["info"]["text"]]))
            return redirect(request.referrer)
            flash(message["info"], "success")
            app.logger.info("; ".join([request.remote_addr.strip(),
            request.url_rule.rule.strip(), request.url_rule.endpoint.strip(), message["info"]["title"],
            message["info"]["text"]]))
            return redirect("/authentication/login")
        except:
            abort(500)
@app.route("/authentication/password/activate/<string:token>", methods=["GET"])
def controller_authentication_activate(token):
    try:
        db.connect()
        activation = db.accounts_enable(token)
        db.disconnect()
        if not activation["state"]:
            flash(activation["info"], "danger")
        else:
            flash(activation["info"], "success")
            return redirect("/authentication/login")
    except:
        abort(500)
@app.route("/authentication/login/send", methods=["POST"])
@login_not_required
def controller_login():

    username = validator.validate_username(request.form.get("username", ""))
    if not username["state"]:
        flash(username["info"], "danger")
        return redirect(request.referrer)
    password = validator.validate_password(request.form.get("password", ""))
    if not password["state"]:
        flash(password["info"], "danger")
        return redirect(request.referrer)
    password = string_safe(password["password"])
    db.connect()
    token = db.sessions_init(username["username"], password["hash"])
    #print(token)
    db.disconnect()
    if not token["state"]:
        flash(token["info"], "danger")
        return redirect(request.referrer)
    session["token"] = token["token"]
    return redirect("/")

    abort(500)
@app.route("/authentication/logout")

```

```

@login_required
def controller_logout():
    try:
        db.connect()
        s_session = db.sessions_complete(session.get("token", None))
        db.disconnect()
        if not s_session["state"]:
            flash(s_session["info"], "danger")
            return redirect(request.referrer)
        session.clear()
        return redirect("/authentication/login")
    except:
        abort(500)
@app.route("/authentication/password/reset/send", methods=["POST"])
@login_not_required
def controller_password_reset():
    try:
        email = validator.validate_email(request.form.get("email", ""))
        if not email["state"]:
            flash(email["info"], "danger")
            return redirect(request.referrer)
        #print(email)
        db.connect()
        constants = db.constants_get()
        if not constants["state"]:
            flash(constants["info"], "danger")
            return redirect(request.referrer)
        application = db.application_get()
        if not application["state"]:
            flash(application["info"], "danger")
            return redirect(request.referrer)
        r_request = db.requests_password_add(email["email"])
        db.disconnect()
        if not r_request["state"]:
            flash(r_request["info"], "danger")
            return redirect(request.referrer)
        mailbox.create_message_reset_password(application["application"]["title"],
                                              email["email"],
                                              r_request["firstname"],

        "{}authentication/password/setup/{}".format(request.host_url, r_request["token"]),
                                              constants["constants"]["deadline_reset"] //
3600)
        #print("{}authentication/password/setup/{}".format(request.host_url,
r_request["token"]))
        message = mailbox.send()
        print(mailbox.message, message)
        if not message["state"]:
            flash(message["info"], "danger")
            return redirect(request.referrer)
        flash(message["info"], "success")
        return redirect("/authentication/login")
    except:
        abort(500)
@app.route("/authentication/password/setup/send", methods=["POST"])
@login_not_required
def controller_password_setup():
    try:
        password = validator.validate_password(request.form.get("new-password", ""))
        print(password)
        if not password["state"]:
            flash(password["info"], "danger")
            return redirect(request.referrer)
        confirmation_password = validator.validate_password_x2(password["password"],
request.form.get("confirmation-password", ""))
        if not confirmation_password["state"]:
            flash(confirmation_password["info"], "danger")
            return redirect(request.referrer)
        password = string_safe(password["password"])
        db.connect()

```

```

application = db.application_get()
if not application["state"]:
    db.disconnect()
    flash(application["info"], "danger")
    return redirect(request.referrer)
new_password = db.requests_password_delete(request.form.get("token",
password["hash"])
db.disconnect()
if not new_password["state"]:
    flash(new_password["info"], "danger")
    return redirect(request.referrer)
flash(new_password["info"], "success")
return redirect("/authentication/login")
except:
    abort(500)

```

dashboard

```

@app.route("/")
@app.route("/dashboard")
@login_required
def view_dashboard():
    constants = db.constants_get()
    if not constants["state"]:
        db.disconnect()
        flash(constants["info"], "danger")
        return redirect("/authentication/logout")
    account = db.accounts_get(token=session.get("token", None))

    if not account["state"]:
        db.disconnect()
        flash(account["info"], "danger")
        return redirect("/authentication/logout")
    notifications = db.notifications_get()
    #print(notifications)
    if not notifications["state"]:
        db.disconnect()
    limits = db.limits_get()
    #print(limits)
    if not limits["state"]:
        db.disconnect()
        flash(limits["info"], "danger")
        return redirect("/authentication/logout")
    timeframes = db.timeframes_get()
    #print(timeframes)
    if not timeframes["state"]:
        db.disconnect()
        flash(limits["info"], "danger")
        return redirect("/authentication/logout")
    if constants["constants"]["mode"] == "table":
        table = db.table_get(account["account"]["role"] == "admin")
        print(table)
        db.disconnect()
        return render_template("dashboard/table.html",
updatedAt=datetime.now().strftime("%H:%M:%S"), limits=limits, timeframes=timeframes,
table=table, constants=constants, account=account, notifications=notifications)
    if constants["constants"]["mode"] == "chart":
        charts = db.charts_get()
        print(charts)
        reports = db.reports_get()
        if not reports["state"]:
            db.disconnect()
            flash(reports["info"], "danger")
            return redirect("/authentication/logout")
        chart = db.chart_get(account["account"]["role"] == "admin")
        print(chart)
        db.disconnect()
        return render_template("dashboard/chart.html",
updatedAt=datetime.now().strftime("%H:%M:%S"), charts=charts, reports=reports,
limits=limits, timeframes=timeframes, chart=chart, constants=constants, account=account,
notifications=notifications)

```

```

@app.route("/feedback")
@login_not_required
def view_contact_us():
    try:
        db.connect()
        constants = db.constants_get()
        if not constants["state"]:
            db.disconnect()
            flash(constants["info"], "danger")
            return redirect("/authentication/logout")
        input(constants)
        account = db.accounts_get(token=session.get("token", None))
        if not account["state"]:
            db.disconnect()
            flash(account["info"], "danger")
            return redirect("/authentication/logout")
        input(account)
        return render_template("dashboard/feedback.html",
                                constants=constants, account=account,
                                page="feedback",
                                )
    except:
        abort(404)

@app.route("/dashboard/select-limit/<int:limit>", methods=["GET"])
@login_required
def controller_limits(limit):
    try:
        limit = validator.validate_limit(limit)
        if not limit["state"]:
            flash(limit["info"], "danger")
            return redirect(request.referrer)
        db.connect()
        new_limit = db.limits_update(limit["limit"])
        print(new_limit)
        if not new_limit["state"]:
            db.disconnect()
            flash(new_limit["info"], "danger")
        db.disconnect()
        return redirect(request.referrer)
    except:
        abort(500)

@app.route("/dashboard/select-timeframe/<int:timeframe>", methods=["GET"])
@login_required
def controller_timeframes(timeframe):
    try:
        timeframe = validator.validate_timeframe(timeframe)
        if not timeframe["state"]:
            flash(timeframe["info"], "danger")
            return redirect(request.referrer)
        db.connect()
        new_timeframe = db.timeframes_update(timeframe["timeframe"])
        print(new_timeframe)
        if not new_timeframe["state"]:
            db.disconnect()
            flash(new_timeframe["info"], "danger")
        db.disconnect()
        return redirect(request.referrer)
    except:
        abort(500)

@app.route("/dashboard/select-mode/<string:mode>", methods=["GET"])
@login_required
def controller_modes(mode):
    try:
        mode = validator.validate_mode(mode)
        print(mode)
        if not mode["state"]:
            flash(mode["info"], "danger")
            return redirect(request.referrer)
        db.connect()
        new_mode = db.modes_update(mode["mode"])
        print(new_mode)

```

```

        if not new_mode["state"]:
            db.disconnect()
            flash(new_mode["info"], "danger")
        db.disconnect()
        return redirect(request.referrer)
    except:
        abort(500)
@app.route("/dashboard/select-chart/<string:chart>", methods=["GET"])
@login_required
def controller_chart(chart):
    try:
        chart = validator.validate_chart(chart)
        print(chart)
        if not chart["state"]:
            flash(chart["info"], "danger")
            return redirect(request.referrer)
        db.connect()
        new_chart = db.charts_update(chart["chart"])
        print(new_chart)
        if not new_chart["state"]:
            db.disconnect()
            flash(new_chart["info"], "danger")
        db.disconnect()
        return redirect(request.referrer)
    except:
        abort(500)
@app.route("/dashboard/select-report/<string:report>", methods=["GET"])
@app.route("/dashboard/select-report/<string:report>/<string:type>", methods=["GET"])
@login_required
def controller_report(report, type=None):
    try:
        report = validator.validate_report(report)
        print(report)
        if not report["state"]:
            flash(report["info"], "danger")
            return redirect(request.referrer)
        db.connect()
        new_report = db.reports_update(report["report"], type)
        print(new_report)
        if not new_report["state"]:
            db.disconnect()
            flash(new_report["info"], "danger")
        db.disconnect()
        return redirect(request.referrer)
    except:
        abort(500)
@app.route("/feedback/send", methods=["POST"])
@login_not_required
def controller_contact_us():
    try:
        username = validator.validate_username(request.form.get("username", ""))
        if not username["state"]:
            flash(username["info"], "danger")
            return redirect(request.referrer)
        title = validator.validate_feedback_title(request.form.get("title", ""))
        if not title["state"]:
            flash(title["info"], "danger")
            return redirect(request.referrer)
        text = validator.validate_feedback_text(request.form.get("text", ""))
        if not text["state"]:
            flash(text["info"], "danger")
            return redirect(request.referrer)
        db.connect()
        new_feedback = db.feedbacks_add(username["username"], title["title"], text["text"])
        print(new_feedback)
        if not new_feedback["state"]:
            db.disconnect()
            flash(new_feedback["info"], "danger")
        db.disconnect()
        flash({"title": "Successfully Sent", "text": "Your feedback has been received.
        Admisitrator will contact with you soon!"}, "success")

```

```

        return redirect(request.referrer)
    except:
        abort(500)

@app.route("/dashboard/chart/select-axis/<string:axis>/<string:title>")
@login_required
def controller_axis(axis, title):
    try:
        print(axis, title)
        db.connect()
        new_axis = db.axis_update(axis, title)
        print(new_axis)
        if not new_axis["state"]:
            db.disconnect()
            flash(new_axis["info"], "danger")
            db.disconnect()
        return redirect(request.referrer)
    except:
        abort(500)

```

settings

```

@app.route("/settings")
@login_required
def view_settings():
    try:
        return redirect("/settings/accounts")
    except:
        abort(404)

@app.route("/settings/<string:table>")
@login_required
def view_settings_table(table):

    db.connect()
    state = db.sessions_validate(session.get("token", None))
    if not state["state"]:
        db.disconnect()
        flash(state["info"], "danger")
        return redirect("/authentication/login")
    account = db.accounts_get(token=session.get("token", None))
    if not account["state"]:
        db.disconnect()
        flash(account["info"], "danger")
        return redirect("/authentication/logout")
    if account["account"]["role"] != "admin":
        db.disconnect()
        flash({
            "title": "Access Restriction",
            "text": "Settings are available for administrators (only)!"
        }, "danger")
        return redirect("/")
    #print(account)
    notifications = db.notifications_get(True)
    if not notifications["state"]:
        db.disconnect()
        flash(notifications["info"], "danger")
        return redirect("/")
    #print(notifications)
    constants = db.constants_get()
    if not constants["state"]:
        db.disconnect()
        flash(constants["info"], "danger")
        return redirect("/")
    #print(notifications)
    settings = db.settings_get(table)
    db.disconnect()
    #print(settings)
    if not settings["state"]:
        db.disconnect()
        flash(settings["info"], "danger")
        return redirect("/")

```

```

        return render_template("settings/{}.html".format(table), table=table,
                                settings=settings, account=account, notifications=notifications, constants=constants)

@app.route("/settings/update/<string:table>/<string:field>/<string:value>/<string:primary>/<string:key>", methods=["GET"])
@login_required
def controller_settings(table, field, value, primary, key):
    try:
        print("xxxxxxxxxxxxxxxxxxxx", table, field, value, primary, key)
        db.connect()
        update = db.settings_update(table, field, value, primary, key)
        print(update)
        if not update["state"]:
            db.disconnect()
            flash(update["info"], "danger")
            db.disconnect()
        return redirect(request.referrer)
    except:
        abort(500)

@app.route("/settings/spider/<string:table>/<string:state>", methods=["GET"])
@login_required
def controller_spiders(table, state):
    pass

#####
# WSGI.PY
#####

import sys, os
sys.path.append("{} /venv/app/".format(os.getcwd()))
sys.path.append("{} /venv/lib/python3.8/site-packages".format(os.getcwd()))
from app.main import app

```

entry point

```

if __name__ == "__main__":
    app.run()

```

ДОДАТОК Б
Слайди презентації

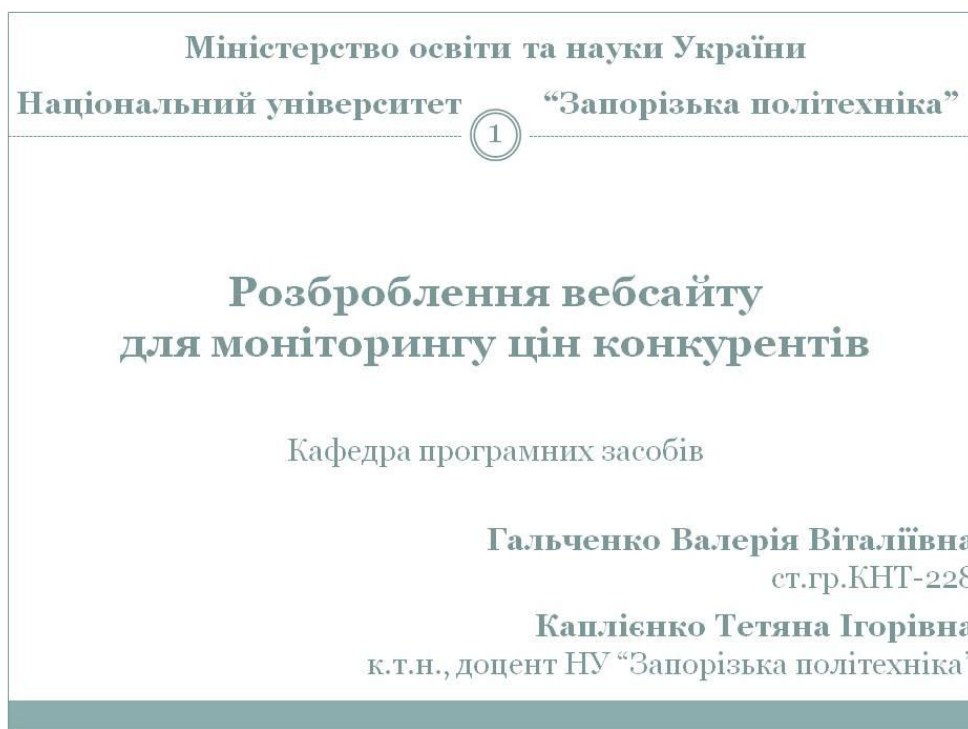


Рисунок Б.1 – Титульний слайд роботи

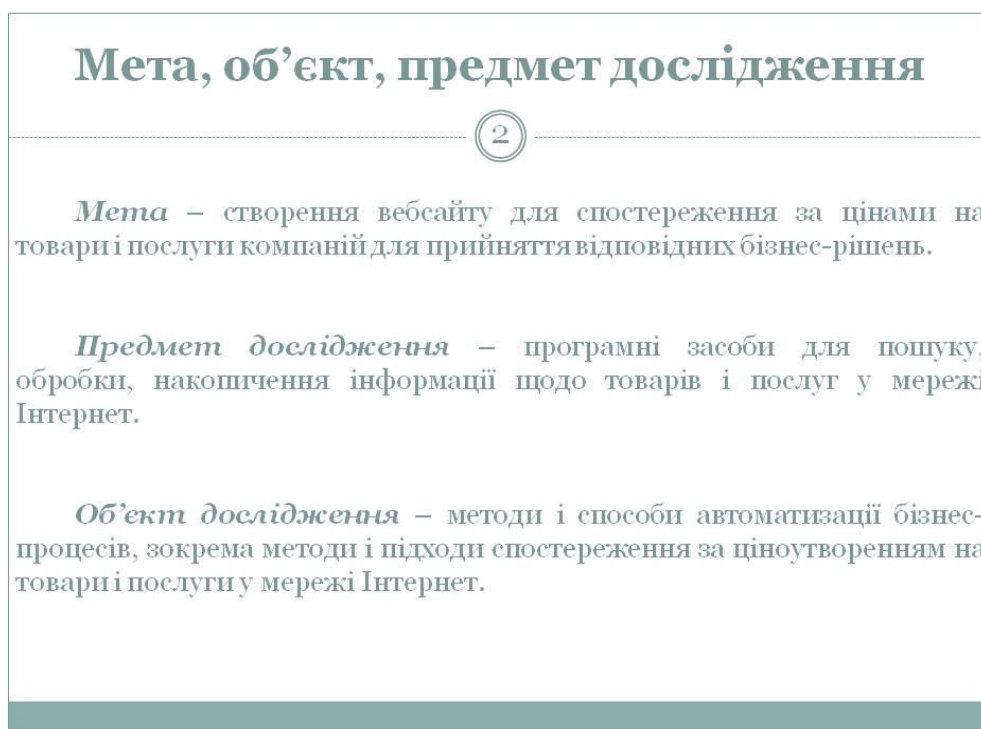


Рисунок Б.2 – Фокус роботи

Огляд і порівняння аналогів				
3				
Прикладні завдання	Price Control	Z-Price	PriceCop	Competera
Пошук Інтернет-майданчиків	+			
Зворотній зв'язок з операторами	+			
Здійснення контрольних закупівель	+			
Збір інформації про товари конкурентів	+	+	+	+
Накопичення інформації про товари конкурентів	+	+	+	
Захист прав інтелектуальної власності магазину	+			
Зняття товарів компанії з продажу	+			
Допомога під час участі в судових справах	+			
Створення звітів		+	+	
Управління номенклатурою			+	
Можливість побудови графіків			+	
API інтеграція		+	+	
Аналітика		+		+
Ціноутворення на основі попиту		+		+
Ціноутворення на основі штучного інтелекту				+
Оповіщення про підозрілих продавців		+		

Рисунок Б.3 – Табличне порівняння аналогів

Постановка завдання
4
- створити модуль управління обліковими записами; - створити модуль управління сесіями; - створити модуль управління даними; - створити модуль управління безпекою даних; - створити модуль управління підтримки користувачів; - розробити додатки для збору даних у мережі Інтернет; - розробити базу даних.

Рисунок Б.4 – Постановка задачі

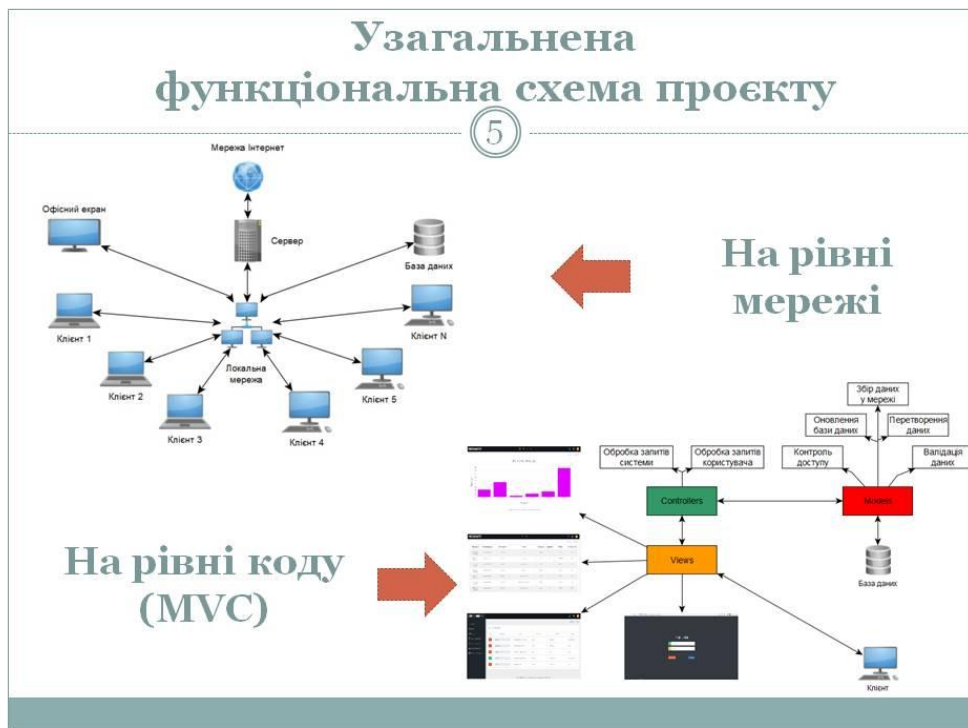


Рисунок Б.5 – Узагальнена функціональна схема проєкту



Рисунок Б.6 – Структурна схема проєкту

⑦

Шаблон парсеру

```
def start(self):
    self.app.state = True
    while self.app.state:
        while True:
            if not self.app.state:
                break
            self.get_state()
            if self.service.state == "online":
                break
            self.get_currencies()
            print("RECEIVED: {} (restart in {} minutes)".format(datetime.now().strftime("%H:%M:%S"),
                                                             self.service.timeframe))
            sleep(self.service.timeframe * 60)
        if self.state:
            print("ALERT: {} (restart in {} minutes)".format(datetime.now().strftime("%H:%M:%S"),
                                                            self.alert.pause))
            sleep(self.alert.pause * 60)
            continue
        print("{} is stopped)".format(PluginDBU._name_))

def save_aggregator(self, table, data):
    try:
        for item in data:
            print(item)
            sql = "SELECT * FROM {} WHERE marketplace={} AND company={} AND location={} AND place={} AND product={} AND quality={} AND archived={} ".format(table,
                                                                                                                                           item["marketplace"],
                                                                                                                                           item["company"],
                                                                                                                                           item["location"],
                                                                                                                                           item["place"],
                                                                                                                                           item["product"],
                                                                                                                                           item["quality"],
                                                                                                                                           "enum=0 or 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9 or 10 or 11 or 12 or 13 or 14 or 15 or 16 or 17 or 18 or 19 or 20 or 21 or 22 or 23 or 24 or 25 or 26 or 27 or 28 or 29 or 30 or 31 or 32 or 33 or 34 or 35 or 36 or 37 or 38 or 39 or 40 or 41 or 42 or 43 or 44 or 45 or 46 or 47 or 48 or 49 or 50 or 51 or 52 or 53 or 54 or 55 or 56 or 57 or 58 or 59 or 60 or 61 or 62 or 63 or 64 or 65 or 66 or 67 or 68 or 69 or 70 or 71 or 72 or 73 or 74 or 75 or 76 or 77 or 78 or 79 or 80 or 81 or 82 or 83 or 84 or 85 or 86 or 87 or 88 or 89 or 90 or 91 or 92 or 93 or 94 or 95 or 96 or 97 or 98 or 99 or 100 or 101 or 102 or 103 or 104 or 105 or 106 or 107 or 108 or 109 or 110 or 111 or 112 or 113 or 114 or 115 or 116 or 117 or 118 or 119 or 120 or 121 or 122 or 123 or 124 or 125 or 126 or 127 or 128 or 129 or 130 or 131 or 132 or 133 or 134 or 135 or 136 or 137 or 138 or 139 or 140 or 141 or 142 or 143 or 144 or 145 or 146 or 147 or 148 or 149 or 150 or 151 or 152 or 153 or 154 or 155 or 156 or 157 or 158 or 159 or 160 or 161 or 162 or 163 or 164 or 165 or 166 or 167 or 168 or 169 or 170 or 171 or 172 or 173 or 174 or 175 or 176 or 177 or 178 or 179 or 180 or 181 or 182 or 183 or 184 or 185 or 186 or 187 or 188 or 189 or 190 or 191 or 192 or 193 or 194 or 195 or 196 or 197 or 198 or 199 or 200 or 201 or 202 or 203 or 204 or 205 or 206 or 207 or 208 or 209 or 210 or 211 or 212 or 213 or 214 or 215 or 216 or 217 or 218 or 219 or 220 or 221 or 222 or 223 or 224 or 225 or 226 or 227 or 228 or 229 or 230 or 231 or 232 or 233 or 234 or 235 or 236 or 237 or 238 or 239 or 240 or 241 or 242 or 243 or 244 or 245 or 246 or 247 or 248 or 249 or 250 or 251 or 252 or 253 or 254 or 255 or 256 or 257 or 258 or 259 or 260 or 261 or 262 or 263 or 264 or 265 or 266 or 267 or 268 or 269 or 270 or 271 or 272 or 273 or 274 or 275 or 276 or 277 or 278 or 279 or 280 or 281 or 282 or 283 or 284 or 285 or 286 or 287 or 288 or 289 or 290 or 291 or 292 or 293 or 294 or 295 or 296 or 297 or 298 or 299 or 300 or 301 or 302 or 303 or 304 or 305 or 306 or 307 or 308 or 309 or 310 or 311 or 312 or 313 or 314 or 315 or 316 or 317 or 318 or 319 or 320 or 321 or 322 or 323 or 324 or 325 or 326 or 327 or 328 or 329 or 330 or 331 or 332 or 333 or 334 or 335 or 336 or 337 or 338 or 339 or 340 or 341 or 342 or 343 or 344 or 345 or 346 or 347 or 348 or 349 or 350 or 351 or 352 or 353 or 354 or 355 or 356 or 357 or 358 or 359 or 360 or 361 or 362 or 363 or 364 or 365 or 366 or 367 or 368 or 369 or 370 or 371 or 372 or 373 or 374 or 375 or 376 or 377 or 378 or 379 or 380 or 381 or 382 or 383 or 384 or 385 or 386 or 387 or 388 or 389 or 390 or 391 or 392 or 393 or 394 or 395 or 396 or 397 or 398 or 399 or 400 or 401 or 402 or 403 or 404 or 405 or 406 or 407 or 408 or 409 or 410 or 411 or 412 or 413 or 414 or 415 or 416 or 417 or 418 or 419 or 420 or 421 or 422 or 423 or 424 or 425 or 426 or 427 or 428 or 429 or 430 or 431 or 432 or 433 or 434 or 435 or 436 or 437 or 438 or 439 or 440 or 441 or 442 or 443 or 444 or 445 or 446 or 447 or 448 or 449 or 450 or 451 or 452 or 453 or 454 or 455 or 456 or 457 or 458 or 459 or 460 or 461 or 462 or 463 or 464 or 465 or 466 or 467 or 468 or 469 or 470 or 471 or 472 or 473 or 474 or 475 or 476 or 477 or 478 or 479 or 480 or 481 or 482 or 483 or 484 or 485 or 486 or 487 or 488 or 489 or 490 or 491 or 492 or 493 or 494 or 495 or 496 or 497 or 498 or 499 or 500 or 501 or 502 or 503 or 504 or 505 or 506 or 507 or 508 or 509 or 510 or 511 or 512 or 513 or 514 or 515 or 516 or 517 or 518 or 519 or 520 or 521 or 522 or 523 or 524 or 525 or 526 or 527 or 528 or 529 or 530 or 531 or 532 or 533 or 534 or 535 or 536 or 537 or 538 or 539 or 540 or 541 or 542 or 543 or 544 or 545 or 546 or 547 or 548 or 549 or 550 or 551 or 552 or 553 or 554 or 555 or 556 or 557 or 558 or 559 or 560 or 561 or 562 or 563 or 564 or 565 or 566 or 567 or 568 or 569 or 570 or 571 or 572 or 573 or 574 or 575 or 576 or 577 or 578 or 579 or 580 or 581 or 582 or 583 or 584 or 585 or 586 or 587 or 588 or 589 or 590 or 591 or 592 or 593 or 594 or 595 or 596 or 597 or 598 or 599 or 600 or 601 or 602 or 603 or 604 or 605 or 606 or 607 or 608 or 609 or 610 or 611 or 612 or 613 or 614 or 615 or 616 or 617 or 618 or 619 or 620 or 621 or 622 or 623 or 624 or 625 or 626 or 627 or 628 or 629 or 630 or 631 or 632 or 633 or 634 or 635 or 636 or 637 or 638 or 639 or 640 or 641 or 642 or 643 or 644 or 645 or 646 or 647 or 648 or 649 or 650 or 651 or 652 or 653 or 654 or 655 or 656 or 657 or 658 or 659 or 660 or 661 or 662 or 663 or 664 or 665 or 666 or 667 or 668 or 669 or 670 or 671 or 672 or 673 or 674 or 675 or 676 or 677 or 678 or 679 or 680 or 681 or 682 or 683 or 684 or 685 or 686 or 687 or 688 or 689 or 690 or 691 or 692 or 693 or 694 or 695 or 696 or 697 or 698 or 699 or 700 or 701 or 702 or 703 or 704 or 705 or 706 or 707 or 708 or 709 or 710 or 711 or 712 or 713 or 714 or 715 or 716 or 717 or 718 or 719 or 720 or 721 or 722 or 723 or 724 or 725 or 726 or 727 or 728 or 729 or 730 or 731 or 732 or 733 or 734 or 735 or 736 or 737 or 738 or 739 or 740 or 741 or 742 or 743 or 744 or 745 or 746 or 747 or 748 or 749 or 750 or 751 or 752 or 753 or 754 or 755 or 756 or 757 or 758 or 759 or 760 or 761 or 762 or 763 or 764 or 765 or 766 or 767 or 768 or 769 or 770 or 771 or 772 or 773 or 774 or 775 or 776 or 777 or 778 or 779 or 780 or 781 or 782 or 783 or 
```

Метод збереження даних у БД

Рисунок Б.7 – Структура парсеру даних

Моделі бізнес даних системи

8

Внутрішні дані (БД)

Зовнішні дані (WEB)

```
item = {
    "company": columns[0].text.strip().lower(),
    "location": "",
    "place": "",
    "product": asset["title"].lower(),
    "quality": asset["cells"],
    "price": 0
}
```

```
item = {
    "updated": current_datetime,
    "archived": "0000-00-00 00:00:00",
    "marketplace": PluginAgrotender._name_.lower(),
    "company": columns[0].text.strip().lower(),
    "location": "",
    "place": "",
    "product": asset["title"].lower(),
    "quality": asset["cells"],
    "price": 0,
    "variation": 0
}
```

Формат даних дашборду (табличний режим)

```
return {"state": True, "limit": constants["limit"], "rows": rows, "columns": columns}
```

Формат даних дашборду (графічний режим)

```
dataset = {
    "caption": "",
    "layouts": {
        "quantity": 0,
        "data": {},
        "x-axis": {
            "title": "",
            "min": 0,
            "max": 0
        },
        "y-axis": {
            "title": "",
            "min": 0,
            "max": 0
        }
    },
    "type": "",
    "custom": None
}
```

Рисунок Б.8 – Моделі бізнес інформації

Діаграма зв'язків у базі даних

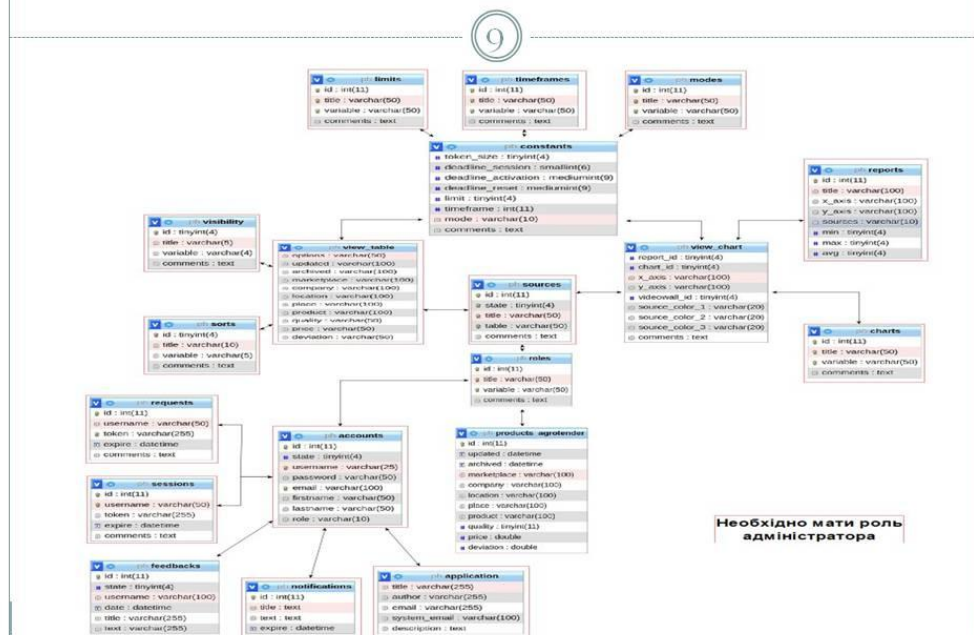


Рисунок Б.9 – Зв'язки у базі даних

Діаграма прецедентів “Робота з вебсайтом”

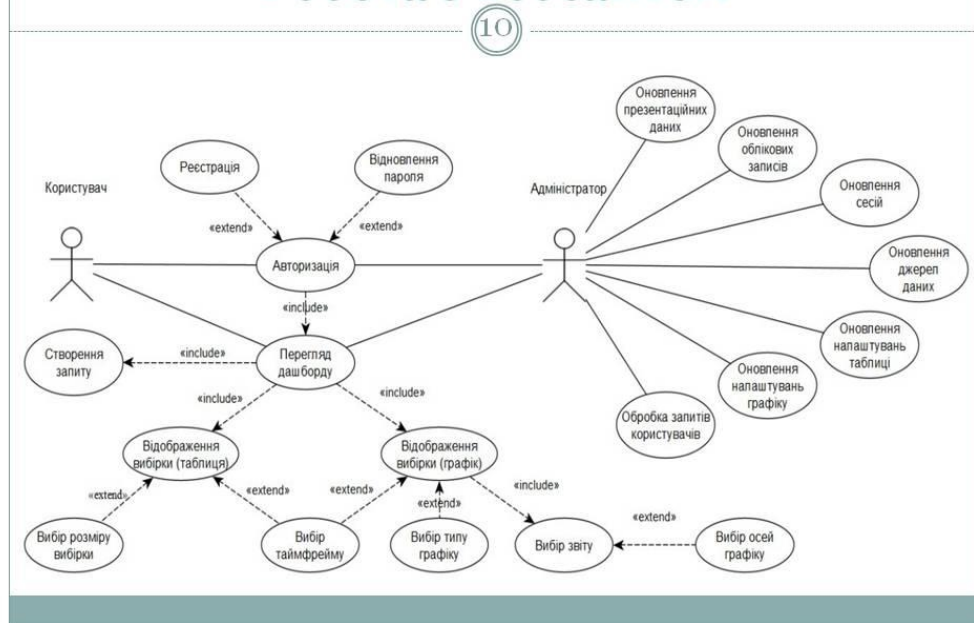


Рисунок Б.10 – Робота з вебсайтом

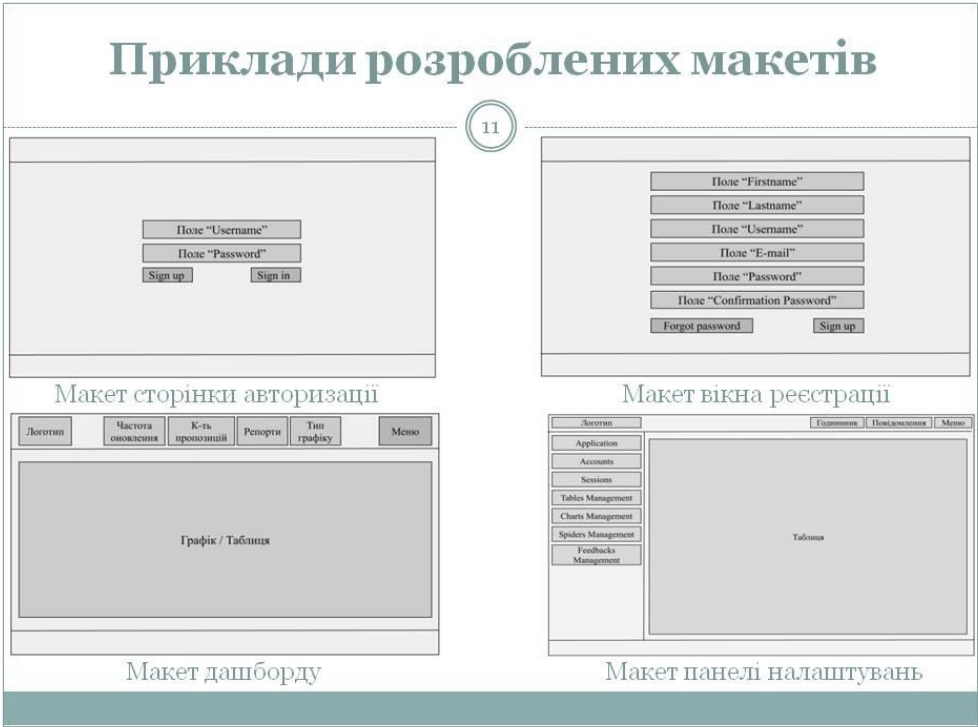


Рисунок Б.11 – Макети вебсайту

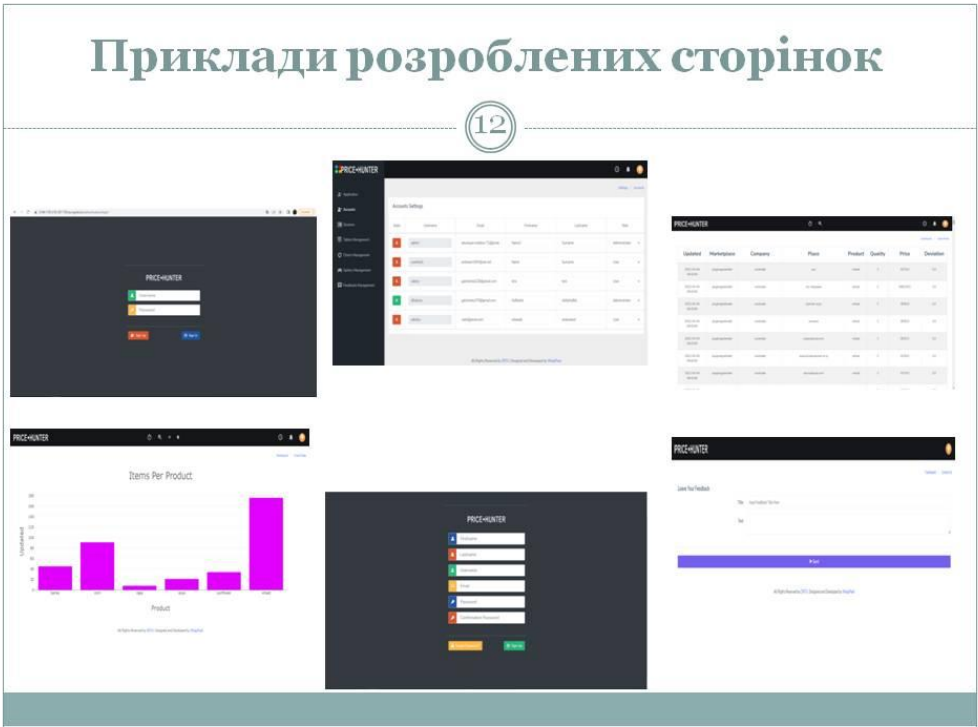


Рисунок Б.12 – Сторінки вебсайту

Висновки

13

- 1) Оглянуто аналоги систем для моніторингу цінових пропозицій
- 2) Створено технічне завдання з вимогами до вебсайту
- 3) Обрано інструменти розробника та методології для реалізації проекту
- 4) Спроектовано та створено базу даних для зберігання системних і бізнес даних
- 5) Розроблено макети сторінок вебсайту, алгоритми та процедури обробки даних
- 6) Розроблено систему «Price Hunter»

Рисунок Б.13 – Висновок по роботі