МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Інститут інженерії та інформаційних технологій

Кафедра комп’ютерної інженерії та електромеханіки

**КВАЛІФІКАЦІЙНА РОБОТА**

на тему:

**Застосунок збору та обробки інформації для малих та середніх підприємств**

Рівень вищої освіти другий (магістерський)

Спеціальності 123 «Комп’ютерна інженерія»

Освітня програма «Комп’ютерна інженерія»

Виконав: студент групи.МгКІ-23

Афанасьєв Дмитро Вячеславович .

 (прізвище та ініціали)

Керівник: к.т.н., доц. Стаценко Д.В.

Рецензент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 (науковий ступінь, вчене звання, прізвище та ініціали)

Київ 2024

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Інститут інженерії та інформаційних технологій .

Кафедра комп’ютерної інженерії та електромеханіки .

Спеціальність 123 «Комп’ютерна Інженерія».

Освітня програма «Комп’ютерна Інженерія»

**ЗАТВЕРДЖУЮ**

Завідувач кафедри КІЕМ

\_\_\_\_\_\_\_\_\_\_\_\_ Дмитро СТАЦЕНКО

“\_\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 року

**ЗАВДАННЯ**

**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**

 Афанасьєв Дмитро Вячеславович .

(прізвище, ім’я, по батькові)

1. Тема кваліфікаційної роботи **Застосунок збору та обробки інформації для малих та середніх підприємств**.

Науковий керівник роботи Стаценко Дмитро Володимирович, к.т.н., доцент . (прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

.

затверджені наказом вищого навчального закладу від 03.09.2024 № 188-уч.

2. Вихідні дані до кваліфікаційної роботи: Сучасні інформаційні потреби малих і середніх підприємств, вимоги до автоматизації процесів обробки даних, технології розробки веб-додатків, стек технологій Python/Django

3. Зміст кваліфікаційної роботи (перелік питань, які потрібно опрацювати): Аналіз потреб малих і середніх підприємств у зборі та обробці інформації.Дослідження сучасних технологій розробки веб-додатків, їх переваг та недоліків.Розробка архітектури застосунку для збору та обробки інформації.

4. Дата видачі завдання \_\_\_\_\_\_\_\_\_\_\_\_2024

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №з/п | Назва етапів дипломної бакалаврської роботи | Терміни виконання етапів | Примітка про виконання |
| 1 | Вступ | 11.09.2024 |  |
| 2 | РОЗДІЛ 1. Теоретична частина | 22.09.2024 |  |
| 3 | РОЗДІЛ 2. Дослідницько-аналітична частина | 11.10. 2024 |  |
| 4 | РОЗДІЛ 3. Проектно-рекомендаційна частина | 25.10.2024 |  |
| 5 | Висновки | 26.10.2024 |  |
| 6 | Оформлення (чистовий варіант) | 30.10.2024 |  |
| 7 | Подача кваліфікаційної роботи науковому керівнику для відгуку | 01.11.2024 |  |
| 8 | Подача кваліфікаційної роботи для рецензування (за 14 днів до захисту) | 05.11.2024 |  |
| 9 | Перевірка кваліфікаційної роботи на наявність ознак плагіату та текстових співпадінь (за 10 днів до захисту) | 09.11.2024 |  |
| 10 | Подання кваліфікаційної роботи на завідувачу кафедри (за 7днів до захисту) | 19.11.2024 |  |

З завданням ознайомлений:

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_** \_\_\_Дмитро АФАНАСЬЄВ

 ( підпис ) (Власне ім’я та ПРІЗВИЩЕ)

Науковий керівник \_\_\_\_\_\_\_\_\_\_\_\_ \_\_Дмитро СТАЦЕНКО.

 ( підпис ) (Власне ім’я та ПРІЗВИЩЕ)

**АНОТАЦІЯ**

**Афанасьєв Д.В. «Застосунок збору та обробки інформації для малих та середніх підприємств»**

Дипломну магістерську роботу присвячено розробці застосунку для збору та обробки інформації, орієнтованого на потреби малих та середніх підприємств. Розроблено систему, що дозволяє автоматизувати процеси збору, аналізу та візуалізації даних для прийняття ефективних управлінських рішень. Досліджено сучасні технології розробки веб-додатків, проаналізовано їх переваги та недоліки. На основі проведеного аналізу обрано стек технологій Python/Django для реалізації проекту. Розроблено архітектуру системи, що забезпечує масштабованість та гнучкість у налаштуванні під потреби конкретного бізнесу.

*Ключові слова: інформаційна система, збір даних, обробка даних, Python, Django, малий та середній бізнес, веб-розробка, автоматизація, аналітика.*

**ABSTRACT**

**Afanasiev D.V. Information collection and processing application for small and medium-sized enterprises. – Manuscript.**

The master's thesis is devoted to developing an application for collecting and processing information oriented towards the needs of small and medium enterprises. A system has been developed that automates the processes of collecting, analyzing, and visualizing data for effective management decisions. Modern web application development technologies have been studied, and their advantages and disadvantages have been analyzed. Based on the analysis, the Python/Django technology stack was chosen for project implementation. The system architecture has been developed to ensure scalability and flexibility in customization for specific business needs.

*Keywords: information system, data collection, data processing, Python, Django, small and medium business, web development, automation, analytics.*

ЗМІСТ

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 3](#_3znysh7)

[ВСТУП 4](#_2et92p0)

[РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА 8](#_3dy6vkm)

[1.1 Важливість систем збору інформації для підприємств 8](#_1t3h5sf)

[1.2 Огляд проблематики 12](#_2s8eyo1)

[РОЗДІЛ 2. ДОСЛІДНИЦЬКО-АНАЛІТИЧНА ЧАСТИНА 35](#_17dp8vu)

[2.1 Аналіз мов програмування 35](#_3rdcrjn)

[2.2 Аналіз бібліотек 42](#_26in1rg)

[РОЗДІЛ 3. ПРОЕКТНО-РЕКОМЕНДАЦІЙНА ЧАСТИНА 60](#_lnxbz9)

[3.1 Розробка проектного рішення 60](#_35nkun2)

[3.2 Розробка рекомендацій 88](#_1ksv4uv)

[ВИСНОВКИ 96](#_44sinio)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 99](#_2jxsxqh)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface – інтерфейс програмування додатків.

SQL – Structured Query Language – мова структурованих запитів для роботи з базами даних.

DBMS – Database Management System – система управління базами даних.

SMEs – Small and Medium Enterprises – малі та середні підприємства.

UX – User Experience – користувацький досвід, досвід взаємодії з інтерфейсом.

UI – User Interface – користувацький інтерфейс.

KPI – Key Performance Indicator – ключовий показник ефективності.

ERP – Enterprise Resource Planning – планування ресурсів підприємства.

JSON – JavaScript Object Notation – формат обміну даними.

SaaS – Software as a Service – програмне забезпечення як послуга.

IoT – Internet of Things – інтернет речей.

AI – Artificial Intelligence – штучний інтелект.

ML – Machine Learning – машинне навчання.

BI – Business Intelligence – бізнес-аналітика.

HTTP – Hypertext Transfer Protocol – протокол передачі гіпертексту.

FTP – File Transfer Protocol – протокол передачі файлів.

GUI – Graphical User Interface – графічний інтерфейс користувача.

SSL – Secure Sockets Layer – рівень захисту сокетів.

REST – Representational State Transfer – архітектурний стиль для побудови API.

KMS – Knowledge Management System – система управління знаннями.

# ВСТУП

Малий та середній бізнес часто стикається з проблемами обмежених ресурсів для збору, обробки та аналізу даних, що суттєво впливає на ефективність ухвалення рішень та стратегічний розвиток. У сучасних умовах конкуренції доступ до якісної інформації може бути вирішальним фактором для зростання і стійкості таких підприємств. Запропонований додаток покликаний вирішити проблему нестачі доступних та зручних засобів збору й аналізу даних, що дозволило б малим і середнім підприємствам адаптуватися до змін у середовищі, гнучко реагувати на потреби ринку та приймати обґрунтовані рішення на основі фактів, а не інтуїції.

**Актуальність теми.** З розвитком цифрових технологій, а також зростанням кількості даних, підприємства різного масштабу отримують безпрецедентні можливості для аналізу ринкових тенденцій, споживчих вподобань та внутрішніх бізнес-процесів. Однак для малих і середніх підприємств доступ до таких інструментів часто є обмеженим через високі витрати на впровадження складних систем, необхідність спеціалізованих знань або брак ресурсів. Тому створення доступного та простого у використанні застосунку для збору й обробки даних має значний потенціал допомогти таким компаніям залишатися конкурентоспроможними, використовуючи дані для обґрунтованих рішень та оптимізації своєї діяльності.

Актуальність цієї теми посилюється потребою підприємств в ефективному використанні ресурсів для досягнення більшої продуктивності та скорочення витрат. Малий та середній бізнес є незамінним елементом економіки багатьох країн, оскільки забезпечує робочі місця, сприяє інноваціям та задовольняє унікальні потреби локальних ринків. Тому забезпечення таких підприємств необхідними цифровими інструментами дозволить підвищити їхню ефективність та сприятиме стабільному економічному зростанню. Інформаційні системи збору та обробки даних дозволять малим та середнім підприємствам зосередитися на стратегічному розвитку, зменшуючи витрати часу на обробку великих обсягів інформації вручну, що є надзвичайно важливим для їхньої стабільності та успіху.

Окрім цього, актуальність теми підкріплюється загальними трендами цифрової трансформації, що охоплює всі сфери бізнесу, а також тенденціями до автоматизації рутинних процесів та обробки великих обсягів даних. Застосунок, який дозволяє малим і середнім підприємствам максимально спростити процес збору, зберігання та аналізу інформації, надасть їм можливість отримувати своєчасну аналітичну підтримку, забезпечуючи конкурентні переваги в умовах швидко змінюваного ринкового середовища. Завдяки цьому такі підприємства можуть більш впевнено адаптуватися до змін та швидше реагувати на потреби своїх клієнтів, що є важливим аспектом для стабільного розвитку в довгостроковій перспективі.

Актуальність розробки застосунку для збору та обробки інформації для малих та середніх підприємств стає ще очевиднішою на тлі викликів, що постають перед сучасним бізнесом, таких як економічна невизначеність, зміни у поведінці споживачів і посилення глобальної конкуренції. У цих умовах доступ до якісних та структурованих даних стає не просто перевагою, а необхідністю для виживання і розвитку бізнесу. Малим і середнім підприємствам особливо важливо мати доступ до інструментів, які допомагають швидко адаптуватися до нових реалій, оцінювати результати своєї діяльності та приймати зважені рішення. Цифровий застосунок, що автоматизує процес збору даних і надає зручні способи їх аналізу, дозволяє зосередитися на стратегії розвитку, не відволікаючись на тривалі процеси опрацювання інформації вручну.

Інструменти для збору й аналізу даних надають малим та середнім підприємствам можливість не тільки вивчати ринок і потреби клієнтів, але й оцінювати ефективність власних бізнес-процесів. Наприклад, компанії можуть аналізувати, які продукти користуються найбільшим попитом, які канали продажів є найефективнішими, або ж відстежувати динаміку витрат. Застосунок, розроблений з урахуванням специфічних потреб малих і середніх підприємств, дозволить автоматизувати ці процеси та забезпечити безперервний потік аналітичних даних, які можуть значно підвищити ефективність управлінських рішень. Таким чином, доступність такої системи дозволяє підприємствам працювати більш продуктивно, швидко виявляти проблеми і відповідно коригувати свою діяльність.

Також важливо зазначити, що технологічна підтримка малого та середнього бізнесу є невід’ємною частиною сталого економічного розвитку. Оскільки ці підприємства часто не мають достатнього фінансування для впровадження складних ERP-систем, розробка спеціалізованого застосунку може стати оптимальним рішенням, що враховує їхні потреби та обмеження. Система збору та обробки інформації, адаптована для використання в малому та середньому бізнесі, може забезпечити підприємства необхідними інструментами для інтеграції даних із різних джерел і візуалізації інформації у зручному форматі. Це дозволить зосередитися на основних завданнях і досягти конкурентних переваг, які раніше були доступні лише великим компаніям з потужними інформаційними системами.

**Об’єкт дослідження -** процеси збору, обробки та аналізу даних в інформаційних системах, що використовуються для підтримки управлінських рішень у малих і середніх підприємствах.

**Предмет дослідження -** методи, засоби та технології розробки застосунку для збору та обробки інформації, орієнтованого на специфічні потреби малих і середніх підприємств з метою підвищення їх ефективності.

**Мета дослідження -** розробка і створення ефективного застосунку для збору та обробки інформації, який би забезпечував малим і середнім підприємствам доступ до релевантних аналітичних даних для оптимізації процесів прийняття управлінських рішень і підвищення їх конкурентоспроможності.

**Завдання дослідження** - Провести аналіз потреб малих і середніх підприємств у системах збору та обробки інформації. Дослідити сучасні технології та підходи до розробки інформаційних систем. Розробити і реалізувати програмне рішення для збору, обробки та аналізу даних. Оцінити ефективність розробленого застосунку через тестування на відповідних бізнес-сценаріях.

**Методи наукових досліджень** - Метод аналізу та синтезу для вивчення сучасних технологій та визначення ключових вимог. Системний підхід для проєктування архітектури застосунку. Емпіричні методи для тестування працездатності розробленого рішення. Метод моделювання для оцінки потенційного впливу застосунку на бізнес-процеси підприємств. Комп'ютерне програмування для реалізації програмного забезпечення.

**Практична цінність** – розроблений застосунок має на меті надавати малим і середнім підприємствам ефективний інструмент для збору, аналізу і зберігання інформації, що дозволить швидше адаптуватися до змін ринку, виявляти точки зростання та оптимізувати ресурси. Це дозволить користувачам поліпшувати планування, знижувати витрати та підвищувати ефективність управління, роблячи аналітику доступною і зрозумілою навіть для компаній з обмеженими ресурсами.

**Елементи наукової новизни** – дослідження включає адаптацію сучасних технологій збору та обробки даних для специфічних вимог малого та середнього бізнесу, що традиційно має обмежені фінансові та кадрові ресурси. У розробці застосунку передбачено застосування методів обробки даних і інтеграцію з аналітичними інструментами, які дозволяють автоматизувати процеси аналізу та створення звітів. Унікальною особливістю є також розробка гнучкої, масштабованої системи, яка може адаптуватися до специфічних потреб користувача та особливостей різних галузей бізнесу.

**Структура роботи.** Робота складається з трьох розділів, шести підрозділів, висновків та списку використаних джерел. Загальний обсяг роботи - 100 сторінок.

# РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА

## 1.1 Важливість систем збору інформації для підприємств

У сучасному бізнес-середовищі, яке швидко змінюється, здатність оперативно приймати обґрунтовані рішення є ключовим фактором успіху. Системи збору інформації для бізнесу, включаючи аналітику даних, дають змогу компаніям отримувати цінні інсайти, які сприяють покращенню стратегії, оптимізації процесів і підвищенню конкурентоспроможності. Розробка таких систем допомагає бізнесам не лише ефективно управляти своїми ресурсами, а й передбачати зміни на ринку, реагувати на потреби споживачів та адаптувати свою діяльність до нових умов.

Покращення прийняття рішень є одним із ключових аспектів, що підвищує конкурентоспроможність і ефективність будь-якого бізнесу. У сучасних умовах ринку ухвалення рішень на основі інтуїції чи попереднього досвіду вже не є достатнім, особливо з огляду на швидкі зміни, що відбуваються у поведінці споживачів, технологічному середовищі та ринкових умовах. Системи збору та аналізу даних дозволяють бізнесам отримувати достовірну інформацію про всі аспекти діяльності: від показників ефективності окремих продуктів до поведінки клієнтів у реальному часі [1].

Завдяки збору й аналізу даних компанії можуть побачити, які стратегії працюють, а які – потребують змін, оскільки дані показують конкретні тенденції, яких інтуїтивне прийняття рішень може не врахувати. Зокрема, вивчення звітів про продажі, аналіз споживчих трендів, а також результати маркетингових кампаній надають цінні інсайти, які дозволяють менеджерам швидше і точніше адаптувати свої стратегії до поточних умов. Система збору даних також допомагає виявляти нові можливості для зростання, наприклад, досліджуючи попит на продукти в нових регіонах чи визначаючи потенційні продукти або послуги на основі інтересів існуючих клієнтів.

Крім того, дані сприяють зниженню ризиків у процесі ухвалення рішень. Наприклад, перед запуском нового продукту чи послуги бізнес може проаналізувати досвід попередніх запусків або зібрати інформацію про ринок, що зменшує ймовірність помилок і фінансових втрат. Прогнозування на основі зібраних даних також дозволяє оцінювати можливі зміни на ринку та адаптувати стратегії відповідно. Усе це забезпечує бізнесу конкурентну перевагу, адже рішення приймаються не лише швидко, але й обґрунтовано, з урахуванням як поточних, так і майбутніх потреб та можливостей.

Оптимізація ресурсів є критично важливою для будь-якої організації, оскільки вона безпосередньо впливає на рентабельність і ефективність роботи. Це означає використання всіх доступних засобів – фінансів, часу, персоналу, технологій – з максимальною віддачею та мінімальними втратами. За допомогою сучасних систем збору та аналізу даних можна досягти суттєвої оптимізації ресурсів, оскільки такі системи дають змогу ретельно відслідковувати та оцінювати продуктивність усіх компонентів бізнесу [2].

Однією з найважливіших переваг оптимізації є можливість раціонального розподілу персоналу та матеріальних ресурсів. Дані допомагають виявити надмірне або недостатнє навантаження в різних відділах компанії, а також визначити, де саме варто збільшити або зменшити ресурси для досягнення оптимальної ефективності. Наприклад, аналіз продуктивності може вказати на процеси, які потребують додаткових витрат, або на ті, що виконуються неефективно, тому варто розглянути можливості їх автоматизації чи делегування. Такі зміни дозволяють перерозподіляти ресурси там, де вони дадуть найкращий результат.

Фінансова оптимізація є ще однією важливою перевагою. Системи управління ресурсами дозволяють стежити за витратами, виявляти зайві чи необґрунтовані витрати та скорочувати їх. Завдяки обґрунтованому розподілу бюджетів і ретельному плануванню, компанії можуть значно зменшити операційні витрати, одночасно зберігаючи або навіть підвищуючи рівень якості продуктів і послуг. Наприклад, дані можуть показати, які постачальники є найбільш економічно вигідними або який час є оптимальним для закупівлі сировини.

Ще одним важливим аспектом є зниження втрат часу. Завдяки збору та аналізу даних у реальному часі, компанії можуть швидше приймати рішення та уникати затримок у процесах, таких як обробка замовлень, логістика або обслуговування клієнтів. Усі ці заходи допомагають оптимізувати час як ресурсу, що дозволяє співробітникам зосередитися на стратегічно важливих задачах, уникаючи повторюваних чи малозначних операцій. Усе це призводить до загального підвищення продуктивності організації, сприяючи сталому зростанню і стабільному розвитку.

Збільшення конкурентоспроможності є одним з ключових аспектів для будь-якого бізнесу, який прагне до сталого розвитку та зростання на ринку. Це передбачає здатність компанії не тільки підтримувати свою позицію на ринку, а й активно конкурувати з іншими гравцями, при цьому забезпечуючи високий рівень якості продуктів чи послуг, інноваційні рішення та ефективне управління ресурсами. Підвищення конкурентоспроможності забезпечує компанії можливість не лише збільшити свою частку на ринку, а й зміцнити репутацію, залучити нових клієнтів і підвищити лояльність існуючих [3].

Система збору інформації для бізнесу, зокрема в сфері аналітики та великих даних, допомагає значно підвищити конкурентоспроможність, оскільки дозволяє глибоко розуміти тенденції на ринку, поведінку клієнтів та конкурентів. Використовуючи аналітику, компанії можуть отримати унікальні інсайти, які дозволяють прогнозувати зміни на ринку, передбачати потреби споживачів та вчасно коригувати свою стратегію. Наприклад, знання про уподобання клієнтів або про популярні товари дозволяє компанії швидко реагувати на зміни попиту та пропонувати саме ті продукти чи послуги, які будуть найбільш актуальними для споживачів.

Використання даних також дозволяє бізнесу розробляти інноваційні стратегії маркетингу та продажу, створювати персоналізовані пропозиції для різних сегментів клієнтів, що, у свою чергу, підвищує ймовірність успіху кампаній і залучення нових клієнтів. Наприклад, аналітика допомагає відстежувати ефективність маркетингових кампаній у реальному часі, а також оцінювати, як зміни в ціноутворенні або в асортименті товарів впливають на покупецьку поведінку. З таких даних можна створювати більш таргетовані рекламні кампанії, що дозволяє досягати кращих результатів при менших витратах [4].

Ще один важливий аспект підвищення конкурентоспроможності — це ефективне управління ресурсами, яке дозволяє компанії знижувати витрати, підвищувати продуктивність і швидкість реакції на зміни на ринку. Аналіз даних дозволяє виявляти неефективні бізнес-процеси та оптимізувати їх, що робить компанію більш гнучкою та здатною швидко адаптуватися до змін. Це включає в себе оптимізацію виробничих ліній, зниження логістичних витрат, автоматизацію внутрішніх процесів і багато інших аспектів, що допомагають значно покращити загальну ефективність роботи бізнесу.

Збільшення конкурентоспроможності також неможливе без постійного вдосконалення продуктів і послуг. Інформаційні системи, які збирають дані про зворотний зв'язок від клієнтів, дозволяють компанії оперативно виявляти слабкі місця в продуктах або послугах і вдосконалювати їх. Це може стосуватися як технічних характеристик продукту, так і його маркетингової стратегії чи навіть сервісного обслуговування. Постійне вдосконалення дозволяє компанії не лише підтримувати свою актуальність на ринку, а й створювати продукти, які є лідерами у своїй категорії.

У кінцевому підсумку, збільшення конкурентоспроможності через використання систем збору та аналізу даних дозволяє бізнесу бути більш адаптованим до змін на ринку, швидше реагувати на запити клієнтів, знижувати витрати та підвищувати якість продукції чи послуг. Це створює додаткові можливості для зростання і розвитку, що є ключовим фактором для довгострокового успіху в конкурентному середовищі.

Отже, системи збору інформації для бізнесу є необхідним інструментом для забезпечення сталого розвитку в умовах постійних змін. Вони надають бізнесу можливість бути більш гнучким, швидким та адаптованим до змін на ринку, що в свою чергу дозволяє зберігати конкурентоспроможність і збільшувати ефективність діяльності. Розвиток таких систем є інвестицією в майбутнє, що забезпечить компанії не лише успіх на поточному етапі, але й стабільне зростання в довгостроковій перспективі [5].

## 1.2 Огляд проблематики

Важливо розрізняти дані та інформацію. Дані є необробленим фактом і можуть мати форму числа або твердження, наприклад дати чи вимірювання. Підприємствам необхідно запровадити процедури для забезпечення реєстрації даних. Наприклад, щоб гарантувати, що оператор кол-центру включає поштовий індекс кожного клієнта, це можна записати в його сценарій і виконати перевірку підтвердження, щоб перевірити, чи ці дані введено в систему.

Загальне визначення інформації полягає в тому, що це дані, які були оброблені таким чином, щоб вони були значущими. Для цього потрібен процес, який використовується для отримання інформації, який передбачає збір даних і подальше їхнє перетворення для створення інформації. Деякі приклади інформації включають прогноз продажів або фінансовий звіт [6].

Як зазначено, інформація генерується шляхом перетворення даних. Цього можна досягти за допомогою ряду різних процесів перетворення або даних. Деякі приклади обробки даних включають агрегування, яке узагальнює дані за допомогою таких засобів, як отримання середнього значення групи чисел. Класифікація розміщує дані за такими категоріями, як своєчасні та пізні поставки. Сортування впорядковує дані таким чином, щоб елементи розміщувалися в певному порядку, наприклад, перелік замовлень за датою доставки. Розрахунки можна проводити на основі таких даних, як розрахунок оплати праці працівника шляхом множення кількості відпрацьованих годин на погодинну ставку оплати праці. Нарешті, дані можна вибирати на основі набору критеріїв відбору, таких як географічне розташування клієнтів.

Хоча інформація є корисним ресурсом для окремих осіб і організацій, не всю інформацію можна вважати корисною. Відмінності між «хорошою» та «поганою» інформацією можна визначити, враховуючи, чи має вона деякі чи всі атрибути якості інформації [7].

Атрибути можуть бути пов’язані з часом, змістом і формою інформації.

Своєчасність означає, що інформація має бути доступною в разі потреби. Якщо інформація надана занадто рано, вона може бути не актуальною під час використання. Якщо інформація надана занадто пізно, вона не принесе користі. Крім того, інформація повинна охоплювати правильний період часу. Прогноз продажів, наприклад, може включати інформацію про минулу продуктивність, поточну продуктивність і прогнозовану ефективність, щоб одержувач мав уявлення про минулі, теперішні та майбутні обставини.

Зміст інформації стосується таких факторів, як точність інформації та релевантність інформації для конкретної ситуації та користувача.

Форма інформації стосується таких аспектів, як ясність інформації, яка має відповідати призначеному одержувачу. Одержувач повинен мати можливість швидко знаходити конкретні предмети та легко розуміти інформацію. Інформація також повинна містити належний рівень деталізації, щоб задовольнити інформаційні потреби одержувача. Наприклад, в деяких випадках буде потрібна дуже детальна інформація, тоді як в інших буде необхідним лише резюме [8].

Визначення систем

Систему можна визначити як набір компонентів, які працюють разом для досягнення спільної мети. Метою системи є отримання вхідних даних і перетворення їх у вихідні дані. У попередньому розділі «Визначення даних та інформації» використовувався процес перетворення, щоб пояснити, як дані перетворюються на інформацію. Не кожна система має єдину ціль, і часто система містить кілька підсистем із підцілями, кожна з яких сприяє досягненню загальної цілі системи. Наприклад, усі сфери фінансів, операцій і маркетингу організації повинні мати цілі, які разом допомагають досягти загальних корпоративних цілей. Можна побачити, що в системах дані використовуються як вхід для процесу, який створює інформацію як вихід. Щоб контролювати продуктивність системи, потрібен певний механізм зворотного зв’язку. Крім того, слід здійснювати контроль, щоб виправити будь-які проблеми, що виникають, і переконатися, що система працює його призначення. Таким чином, існує п'ять компонентів загальної системи з точки зору входу, процесу, виходу, зворотного зв'язку та контролю [10].

Визначення інформаційних систем

Роль інформаційних систем у наданні інформації керівництву, яка дозволить їм приймати рішення, що забезпечують контроль над організацією. Організація контролюватиме, якщо вона задовольняє потреби середовища. По відношенню до системи управління можна класифікувати на розімкнені та замкнуті.

Система керування з розімкнутим циклом — це система, яка не має можливості забезпечити досягнення цілей процесу. Це означає, що вони непридатні в організаційному контексті через складність середовища, в якому існують організації. Таким чином, системи з відкритим контуром будуть успішними в досягненні цілей системи лише в тих випадках, коли ми точно знаємо події, які відбуватимуться під час процесу системи.

Системи із замкнутим контуром можуть мати два типи механізмів керування, які називаються керуванням зі зворотним зв’язком і керуванням із прямим зв’язком. Системи керування зі зворотним зв’язком зазвичай забезпечують спосіб забезпечення контролю над системою. Негативний зворотний зв'язок - це коли вживаються дії для усунення будь-яких відмінностей між бажаними та фактичними результатами. Слабкість цього підходу полягає в потенційній затримці між невідповідністю та діями, вжитими для її зменшення. Системи прямого керування намагаються подолати затримку часу, пов’язану із системами зворотного зв’язку, шляхом включення елемента передбачення в контур зворотного зв’язку керування. Системи прямого зв’язку не такі поширені, як системи зворотного зв’язку в бізнес-налаштуваннях. Приклади включають плани управління проектами, які створені для досягнення цілей у часі, якості та вартості з часом.

Бізнес-інформаційні системи

Завдяки попереднім визначенням інформації та систем ми тепер можемо визначити бізнес-інформаційну систему як групу взаємопов’язаних компонентів, які працюють разом, щоб виконувати дії введення, обробки, виведення, зберігання та керування з метою перетворення даних в інформаційні продукти, які можна використовувати для підтримки прогнозування, планування, контролю, координації, прийняття рішень та операційної діяльності в організації. З точки зору компонентів, які здійснюють цю діяльність, їх можна класифікувати на п’ять основних ресурсів: люди, апаратне забезпечення, програмне забезпечення, комунікації та дані [11]. До людських ресурсів належать користувачі та розробники інформаційної системи, а також ті, хто допомагає підтримувати та керувати системою, наприклад менеджери ІБ та персонал технічної підтримки. Апаратні ресурси включають комп’ютери та інші елементи, такі як принтери. Ресурси програмного забезпечення стосуються комп’ютерних програм, відомих як програмне забезпечення, і відповідних інструкцій.

Комунікаційні ресурси включають мережі та апаратне та програмне забезпечення, необхідне для їх підтримки. Ресурси даних охоплюють дані, до яких організація має доступ, наприклад комп’ютерні бази даних і паперові файли.

У більшості організацій бізнес-інформаційні системи (BIS) широко використовують інформаційні технології, такі як персональні комп'ютери. Причини, чому комп'ютеризовані BIS набули широкого поширення, очевидні в їх перевагах, таких як швидкість, точність і надійність. Вони також мають високий ступінь гнучкості завдяки своїй здатності програмуватися для виконання широкого спектру завдань [12]. Однак у BIS є деякі недоліки, такі як брак креативності, якою володіють люди, і труднощі з включенням інших факторів у процес прийняття рішень, таких як інновації та інтуїція.

Види бізнес-інформаційної системи

Інформаційні системи можна розділити на дві категорії: системи, які підтримують повсякденну бізнес-діяльність організації, і системи, які підтримують прийняття управлінських рішень. Операційно-інформаційні системи (OIS) зазвичай пов’язані з керуванням процесами, обробкою транзакцій і зв’язком. Інформаційні системи управління (MIS) займаються забезпеченням підтримки прийняття управлінських рішень. Нещодавно цей поділ BIS на операційні та управлінські системи, хоча й корисний для менеджерів, які переглядають типи BIS, що використовуються, зараз не точно відображає реальність систем, що використовуються в організації, особливо з розширеним використанням міжорганізаційної електронної комерції та електронний обмін даними (EDI). Наприклад, системи електронного бізнесу та системи планування ресурсів підприємства поєднують як операційні системи, так і системи управління, щоб забезпечити підприємства більш інтегрованими інформаційними системами.

Обладнання

Апаратне забезпечення описує фізичні компоненти комп’ютерної системи, які можна класифікувати як пристрої введення, центральний процесор, внутрішня та зовнішня пам’ять і пристрої виведення. Пристрої введення використовуються для захоплення або введення даних у комп’ютер. Центральний процесор (CPU) виконує обробку, виконуючи інструкції, подані у формі комп’ютерних програм.

Внутрішня пам'ять використовується як тимчасовий засіб зберігання даних і інструкцій, тоді як зовнішня пам'ять забезпечує засіб зберігання даних і програм поза комп'ютером. Пристрої виведення переводять результати обробки в зрозумілу людині форму. Тепер ці апаратні компоненти будуть описані більш детально.

Пристрої введення

Пристрої введення використовуються для введення даних або інструкцій ззовні в комп’ютер. Прикладами пристроїв введення є миша та клавіатура. Вибір пристрою введення часто залежить від кількості даних, які потрібно ввести. Введення даних у невеликому масштабі зазвичай виконується людьми-операторами за допомогою низки звичних пристроїв введення, таких як миша або клавіатура [14].

Комп’ютерна інформаційна система рідко використовує лише один пристрій введення. Навіть звичайний персональний комп'ютер часто має кілька різних методів введення даних, наприклад клавіатуру, мишу, джойстик і звукову карту.

Центральний процесор (CPU)

Центральний процесор (CPU) або процесор приймає інструкції та дані та виконує їх, зберігаючи результати в пам’яті. Підвищення швидкодії комп’ютерів є насамперед результатом підвищення швидкості процесора. Швидкість процесора буде залежати від ряду різних факторів. наприклад тактова частота та ширина шини. Тактова частота визначає, скільки команд за секунду може виконати процесор. Ширина шини описує, скільки фрагментів даних можна передати одночасно. В обох випадках, чим вище значення, тим потужніший процесор.

Значення тактової частоти та пропускної здатності можуть бути корисними при спробі порівняти процесори, щоб вибрати найбільш відповідний.

Внутрішня та зовнішня пам'ять

Пам’ять комп’ютера класифікується як внутрішня пам’ять (яка також називається основною пам’яттю або основною пам’яттю), яка містить дані на комп’ютері, і зовнішня пам’ять (також звана зовнішньою пам’яттю), яка є даними, які зберігаються на окремому пристрої, де інформація буде зберігатися, навіть якщо машина вимкнено. Пам'ять комп'ютера використовується для зберігання даних, які очікують на обробку, інструкцій, завантажених із програмного забезпечення, яке використовується для обробки даних або керування комп'ютерною системою, а також даних або інформації, яка була оброблена. Прикладами зовнішньої пам’яті є дискети та жорсткі диски [15].

Пристрої виведення

Пристрої виведення відображають результати комп'ютерної обробки. Комп’ютерна інформаційна система використовуватиме низку пристроїв виведення як монітор, принтер і звукова карта.

Основні категорії комп'ютерів

Існує три основні категорії комп’ютерів: мейнфрейм, міні-комп’ютер і мікрокомп’ютер. Ми коротко розглянемо характеристики кожної категорії, щоб краще зрозуміти, як промисловість використовує комп’ютерні технології.

Мейнфрейм

Мейнфрейми традиційно асоціювалися з великими потужними машинами, призначеними для великомасштабної обробки даних. Використання мейнфреймів у промисловості, яке колись відповідало за великі доходи таких компаній, як IBM, неухильно знижувалося протягом останніх двох десятиліть. Поточними постачальниками є IBM, Fujitsu та Unisys. Технологічний прогрес дозволив меншим і менш дорогим системам конкурувати з мейнфреймами за швидкістю та потужністю. Сучасний персональний комп’ютер, наприклад, можна вважати у багато разів потужнішим за одну з найдавніших мейнфреймів. У багатьох організаціях мейнфрейми вважаються застарілими системами, а це означає, що хоча керівники визнають, що існуюча система може бути не зовсім адекватною для задоволення потреб компанії, перехід буде важко реалізувати [16].

Міні-комп’ютер поєднує в собі деякі характеристики мейнфрейму та мікрокомп’ютера. Сьогодні їх часто називають серверами такі компанії, як IBM (наприклад, IBM AS/400) і Hewlett-Packard (наприклад, HP Alpha). Різні типи серверів можуть мати різні функції, такі як керування мережею або розміщення бази даних.

Мікрокомп'ютер використовує більш сучасну технологію, щоб забезпечити відносно потужні обчислювальні засоби за низькою ціною. Зараз мікрокомп’ютери часто називають «клієнтськими» машинами, які отримують послуги та дані від «серверної» машини. Деякі з основних характеристик мікрокомп’ютерів полягають у тому, що вони невеликі, відносно недорогі та можуть використовуватися для різних цілей.

Існує дві основні категорії програмного забезпечення: системне програмне забезпечення та прикладне програмне забезпечення.

Системне програмне забезпечення

Системне програмне забезпечення керує та контролює роботу комп’ютерної системи, коли вона виконує завдання від імені користувача. Системне програмне забезпечення складається з трьох основних категорій: операційні системи, програми розробки програмного забезпечення та службові програми.

Операційні системи (ОС)

Операційна система взаємодіє з апаратним забезпеченням комп’ютера, відстежуючи та надсилаючи інструкції для керування та керування ресурсами комп’ютера. Операційна система функціонує як посередник між функціями, які користувач повинен виконати, наприклад, пошук у базі даних, і тим, як вони транслюються на апаратне забезпечення та з нього у формі відповіді на клацання мишею та відображення інформації на екрані. Основні функції операційної системи включають: розподіл і управління системними ресурсами, планування використання ресурсів і моніторинг діяльності комп'ютерної системи.

Програми розробки програмного забезпечення

Програми розробки програмного забезпечення дозволяють користувачам розробляти власне програмне забезпечення для виконання завдань обробки за допомогою мов програмування. Мови програмування можна описати з точки зору їх історичної позиції в розвитку систем комп'ютерного програмування. Мова програмування першого покоління або машинна мова вимагає від програміста працювати з одиницею та нулями для представлення символів і чисел. Це надзвичайно трудомістке завдання було дещо спрощено за допомогою коротших кодів і названо мовою асемблера. Значний прогрес стався з мовами третього покоління, такими як FORTRAN, COBOL, BASIC, Pascal і C, які значно скорочують час програміста на створення коду. Мови четвертого покоління, такі як SQL, побудовані навколо системи баз даних і роблять створення коду навіть легшим, ніж мови третього покоління [17].

Допоміжні програми

Допоміжні програми надають ряд інструментів, які підтримують роботу та керування комп’ютерною системою. Програми, які відстежують продуктивність системи або забезпечують контроль безпеки, є прикладами службових програм.

Прикладне програмне забезпечення

Прикладне програмне забезпечення можна визначити як набір програм, які дозволяють користувачам виконувати певні дії з обробки інформації. Прикладне програмне забезпечення можна розділити на дві широкі категорії: загального призначення та програмне забезпечення.

Програми загального призначення

Програми загального призначення – це програми, які можна використовувати для виконання широкого спектру типових завдань. Текстовий процесор, наприклад, здатний створювати різноманітні документи, які підходять для багатьох різних цілей. Цей тип програми часто називають програмним забезпеченням для підвищення продуктивності, оскільки воно допомагає підвищити ефективність роботи людини. Програмне забезпечення для обробки текстів передбачає створення різноманітних внутрішніх і зовнішніх документів, включаючи листи, звіти, рахунки-фактури, нотатки та протоколи зустрічей. Програмне забезпечення для роботи з електронними таблицями дозволяє зберігати, систематизувати та аналізувати числові дані. Програмне забезпечення баз даних дозволяє зберігати та шукати інформацію. Мультимедійне програмне забезпечення дозволяє користувачеві працювати з такими засобами масової інформації, як текст, звук, анімація та відео [18].

Спеціальне програмне забезпечення

Спеціальне програмне забезпечення включає програми, призначені для певної мети або виконання чітко визначеного завдання обробки інформації. Програмне забезпечення, призначене для обробки заробітної плати або керування рахунками, є прикладом спеціальної програми.

Системи баз даних

База даних — це сукупність пов’язаних файлів. Бази даних можуть існувати на папері, наприклад, телефонний довідник, але вони неефективні та дорогі в обслуговуванні. Комп’ютерна база даних пропонує перевагу потужних засобів пошуку, які можна використовувати для пошуку та отримання інформації у багато разів швидше, ніж за допомогою ручних методів. Електронна база даних надає можливість користувачам додавати, змінювати або видаляти записи за потреби. Функції індексування означають, що ту саму базову інформацію можна зберігати в кількох різних категоріях. Це забезпечує велику гнучкість і дозволяє користувачам знаходити, отримувати та впорядковувати інформацію за потреби. Бази даних, які використовуються в усій компанії, як правило, доступні багатьом користувачам мережі. Деякі з переваг цього підходу включають мінімізацію непотрібного дублювання інформації, узгодженість підтримується шляхом забезпечення відображення будь-яких змін, внесених до інформації, що зберігається в базі даних, для всіх користувачів, і хоча інформація зберігається в структурованому вигляді, програмне забезпечення бази даних зазвичай забезпечують достатню гнучкість для задоволення різних вимог окремих користувачів і відділів [21].

Oрганізація даних у базі даних

Дані в електронній базі даних організовано за полями та записами. Поле — це окремий елемент інформації, наприклад назва або кількість. Запис — це набір пов’язаних полів, а таблиця — набір пов’язаних записів. Щоб ідентифікувати певний елемент інформації в базі даних, усі записи повинні містити унікальний ідентифікатор, який зазвичай називають ключовим полем або первинним ключем. Ключове поле зазвичай має форму числа або коду та буде різним для кожного запису в базі даних.

Реляційні бази даних дозволяють зберігати дані в кількох різних таблицях і є найпоширенішим типом бази даних. Таблиці в реляційній базі даних можуть бути пов’язані разом за допомогою одного або кількох ключів запису. Це включає первинний ключ, а також інші ключі, які допомагають знайти дані, що зберігаються в іншій таблиці. Ключі запису, що містяться в кожній таблиці, можуть використовуватися для встановлення одного або кількох зв’язків між таблицями. Використовуючи комбінацію ключів запису, можна отримувати дані з кількох таблиць одночасно. Поле, яке використовується для пошуку інформації в іншій пов’язаній таблиці, часто називають зовнішнім ключем.

Програмне забезпечення для баз даних

Більшість програм баз даних підтримують створення реляційних баз даних, що містять кілька пов’язаних таблиць. Багато програм, наприклад Microsoft Access, надають можливість автоматично зв’язувати таблиці між собою для створення будь-яких необхідних зв’язків. Усі основні програми баз даних дозволяють користувачам створювати та змінювати форми введення даних. Форма введення даних забезпечує зручний спосіб перегляду, введення, редагування та видалення записів. Індекс зберігає інформацію про порядок записів у базі даних. Усі сучасні програми баз даних забезпечують ряд складних функцій безпеки. Приклади деяких найпоширеніших доступних функцій, включаючи шифрування та захист паролем [23].

Нарешті, усі основні пакети баз даних дозволяють користувачам створювати різноманітні звіти. Багато програм здатні автоматично створювати прості звіти. Крім того, багато програм дозволяють користувачам виконувати обчислення та інші дії під час створення звіту.

Отримання даних із бази даних

Під час використання програмного забезпечення бази даних дані витягуються з бази даних за допомогою так званого запиту. Запит дозволяє користувачеві знаходити, сортувати, оновлювати або витягувати записи з бази даних. Користувачі створюють запит, вказуючи умови, які мають бути виконані для вибору запису. Існує два типи запитів, які називаються запитами на вибір і запитами на оновлення: запит на вибір можна використовувати для пошуку та відображення будь-яких записів, які відповідають набору заданих умов. Жодні дані, що зберігаються в базі даних, не змінюються, і будь-які записи, які не відповідають встановленим умовам, просто тимчасово приховані від перегляду. Запит на оновлення можна використовувати для зміни записів різними способами, наприклад, відповідно до набору умов, визначених користувачем. Загальні дії, які виконуються запитами на оновлення, включають оновлення значень, що зберігаються в полях, видалення будь-яких записів, які більше не потрібні, додавання нових записів до бази даних і створення нових таблиць, що містять вибрані записи або підсумкову інформацію.

Більшість програм баз даних використовують спеціальну структуровану мову запитів (SQL) для створення запитів. Структурована мова запитів (SQL) забезпечує стандартизований метод для отримання інформації з баз даних. Хоча традиційно використовується для керування великими базами даних, які зберігаються на мейнфреймах і міні-комп’ютерах, він став широко використовуваним і популярним інструментом для пакетів баз даних персональних комп’ютерів. Програми SQL створюються шляхом створення серії операторів, що містять спеціальні ключові слова [24].

Перетворення виклику на навчаннякрива.

Просто ще один день в офісі для високого виконавця.

Мережа — це поєднання пристроїв, з’єднаних один з одним за допомогою каналів зв’язку, щоб забезпечити канали для безперервного потоку інформації між людьми. Це дозволяє людям співпрацювати, а також дає змогу економічно ефективніше використовувати обладнання, наприклад принтери та факси.

Мережі важливі для організації, тому що вони допомагають бізнесу зв’язуватися зі своїми клієнтами, постачальниками та співробітниками. Завдяки цьому компанія може швидше й дешевше замовляти нову сировину у своїх постачальників і підтримувати зв’язок із потребами своїх клієнтів. Додаткові переваги мереж включають зниження витрат за рахунок використання таких засобів, як електронна пошта, скорочення часу для потоку інформації, наприклад, порівняння електронної пошти з доставкою поштою, можливість обміну інформацією шляхом доступу до бази даних через мережеву систему, можливість спільного використання апаратних пристроїв, таких як як принтери в мережі, використання інструментів групової роботи для обміну документами та іншою інформацією. Основними недоліками мереж є вартість встановлення мережі та забезпечення безпечної та надійної мережі.

Сервери контролюють потік інформації в мережі та використовують для керування мережею спеціалізоване програмне забезпечення, яке називається мережевою операційною системою (NOS). Сервер і NOS разом забезпечують спільний доступ до інформації, прикладного програмного забезпечення та апаратних пристроїв, таких як принтери. Він також контролює доступ до інформації у файлах. Для мережі, яка складається з 20 або більше людей, описані вище функції можуть бути розподілені між кількома серверами для розподілу навантаження. Може бути окремий файловий сервер, сервер друку, сервер паролів і сервер бази даних. У дуже великих компаніях для зберігання даних буде використовуватися багато серверів. Усі вони будуть об’єднані мережею, щоб забезпечити доступність даних для всіх. Вони також відповідатимуть за те, щоб за допомогою процесу, відомого як реплікація, одна і та сама версія даних існувала на різних серверах. Завдяки використанню багатьох серверів існує можливість розподілити обчислювальне навантаження між цими серверами, а не перевантажувати одну центральну машину, як це сталося за часів мейнфреймів. Спільне використання функцій кількома комп’ютерами називається «розподіленим обчисленням» [26].

Комп’ютери кінцевих користувачів або термінали

Точки доступу для користувачів мережі відомі як клієнти, вузли, робочі станції або, як правило, ПК. Для роботи в мережі кожен клієнт повинен мати встановлене мережеве програмне забезпечення, наприклад Novell Netware. Підключення до мережі також потрібне через мережевий кабель, підключений до карти мережевого інтерфейсу в одному зі слотів ПК, або через систему бездротової мережі.

Телекомунікаційні процесори

Телекомунікаційні процесори — це апаратні засоби, які використовуються для зв’язку між серверами та клієнтами та різними мережами. Зазвичай вони називаються під конкретними назвами, наприклад концентратори, мультиплексори, мости та маршрутизатори. У компанії, яка потребує використання шлюзових пристроїв, необхідний спеціаліст для їх обслуговування. Концентратори використовуються для підключення до 20 комп’ютерів до мережі зручним способом за допомогою патч-кабелів (схожих на телефонні кабелі та розетки), що проходять між задньою частиною кожного комп’ютера та концентратором. Потім концентратор може бути приєднаний до сервера або до магістрального з’єднання, що веде до сервера. Маршрутизатори можуть вибрати найкращий маршрут для передачі пакетів, а також використовуються для цього в магістралях Інтернету та глобальній мережі [27].

Хоча раніше ці пристрої відрізнялися один від одного, тепер вони виробляються як гібриди, які мають спільні функції. Компанії, підключені до Інтернету, зазвичай використовують маршрутизатор як шлюз для підключення своєї внутрішньої мережі до Інтернету. Це часто поєднується з «брандмауером», який призначений для зменшення ризику отримання сторонніми особами несанкціонованого доступу до даних компанії [29].

Проміжне програмне забезпечення

Проміжне програмне забезпечення — це спеціалізований тип програмного забезпечення, яке дозволяє різним програмним програмам обмінюватися даними. Він діє як прошарок між іншим програмним забезпеченням, допомагаючи передавати дані між несумісними системами. Його часто називають «клеєм», який зв’язує програмні додатки з системним програмним забезпеченням. Це важливо в мережевому світі, оскільки надає послуги перекладу між програмним забезпеченням, що працює на різних типах комп’ютерних систем у різних компаніях. Прикладом проміжного програмного забезпечення є програмне забезпечення шлюзу, яке дозволяє внутрішній системі електронної пошти, такій як Lotus cc:Mail, надсилати повідомлення до інших систем електронної пошти через Інтернет. Проміжне програмне забезпечення також необхідне для того, щоб єдине програмне забезпечення, наприклад обробка замовлень на продаж, отримувало доступ до різних типів баз даних, таких як Oracle, Informix або Microsoft SQL Server, якими може користуватися велика компанія. Проміжне програмне забезпечення для підтримки зв’язку можна класифікувати відповідно до семирівневої моделі, відомої як модель OSI.

6. Інтернет і Всесвітня павутина

Інтернет дозволяє спілкуватися між підключеними комп’ютерами по всьому світу. Інформація передається з комп’ютерів, користувачі яких запитують послуги, на комп’ютери, які містять інформацію та розміщені бізнес-додатки, які надають послуги у відповідь на запити. Комп’ютери вдома та на підприємствах підключаються до Інтернету через місцевих Інтернет-провайдерів (ISP), які, у свою чергу, пов’язані з більшими Інтернет-провайдерами з підключенням до основної національної та міжнародної інфраструктури або магістралей. Інтернет можна описати як глобальну мережеву систему, що складається з менших систем. Інтернет був задуманий Агентством перспективних оборонних досліджень (DARPA), американською розвідувальною організацією, у 1969 році. Інтернет почав досягати своєї нинішньої форми в 1987 році, виростаючи з систем, розроблених DARPA та Національним науковим фондом (NSF) [30].

Інтернет-бізнес

Бізнес із підтримкою Інтернету можна класифікувати за тими сторонами, які беруть участь у бізнес-операціях. Найпоширеніші транзакції, визначені як ті, коли організація використовує Інтернет для здійснення операцій із споживачами, що називаються бізнес-споживач (B2C), або коли організація здійснює операції з іншими підприємствами, що називається бізнес-бізнес (B2B). Відносини між компанією та її постачальниками та клієнтами можуть кардинально змінитися завдяки можливостям, які надає Інтернет. Це відбувається тому, що Інтернет пропонує засоби обходу деяких партнерів каналу. Цей процес відомий як деінтермедіація або «виключення посередника». Переваги деінтермедіації полягають у тому, що вона здатна усунути витрати на продаж та інфраструктуру продажу через канал. Частина цих заощаджень може бути передана клієнту у вигляді скорочення витрат. Хоча відмова від посередництва є широко поширеною, також відбулося створення нових посередників між клієнтами та постачальниками, що називається повторним посередництвом. Наприклад, у туристичній галузі такі компанії, як Tripadvisor, надають інформацію про напрямки та готелі, а потім надають посилання на постачальників готелів [31].

Більшість Інтернет-послуг доступні для будь-якого бізнесу або споживача, який має доступ до Інтернету. Однак багато бізнес-додатків, які мають доступ до конфіденційної інформації компанії, вимагають обмеження доступу до привілейованих осіб або третіх осіб. Якщо інформація обмежена тими, хто знаходиться всередині організації, мережа називається інтранет. Якщо доступ поширюється на деяких інших, але не на всіх за межами організації, мережа називається екстранет. Доступ до екстрамережі можуть мати авторизовані люди за межами компанії, наприклад співробітники, постачальники або великі клієнти, але інформація доступна не всім, хто має підключення до Інтернету, але доступ до неї обмежено паролем. Інтрамережі також використовуються для обміну інформацією, такою як довідники телефонів персоналу, процедури персоналу або посібники з якості, інформація для агентів, така як специфікації продукту, поточні списки та знижені ціни, інформація про конкурентів, розклад заводу та рівень запасів – усю цю інформацію зазвичай потрібно оновлювати. часто і може бути дорогим.

Екстранети широко використовуються для підтримки таких дій, як замовлення від постачальників.

Всесвітня павутина надає стандартний спосіб обміну та публікації інформації в Інтернеті. Носій базується на стандартних форматах документів, таких як HTML (мова розмітки гіпертексту), який отримав широке поширення, оскільки він підтримує широкий спектр засобів форматування, що робить документи легкими для читання на різних пристроях доступу. Він також містить графіку та анімацію, які можна інтегрувати у веб-сторінки, а взаємодія можлива через форми на основі HTML, які дозволяють клієнтам надавати свої особисті дані для отримання додаткової інформації про продукт, виконувати пошук, ставити запитання чи залишати коментарі.

Саме поєднання веб-браузерів і HTML виявилося таким успішним у налагодженні широкого ділового використання Інтернету. Використання цих інструментів забезпечує низку переваг, таких як збільшення простоти переходу між документами завдяки використанню гіперпосилань або зображень. Незабаром це стане дуже інтуїтивно зрозумілим способом навігації, подібним до всіх веб-сайтів і програм. Він може забезпечити графічне середовище, що підтримує мультимедіа, яке користується популярністю серед користувачів, і надає візуальне середовище для реклами. Стандартизація інструментів і зростання попиту означає можливість обміну інформацією з багатьма компаніями та споживачами.

Веб-браузери — це програми, які використовуються для доступу до інформації у всесвітній мережі, що зберігається на веб-серверах. Веб-сервери використовуються для зберігання, керування та надання інформації у всесвітній мережі. Основними використовуваними веб-браузерами є Microsoft Internet Explorer і Mozilla Firefox. Браузери відображають текст і графіку, доступ до яких здійснюється з веб-сайтів, і надають інструменти для керування інформацією з веб-сайтів. Веб-браузери взаємодіють із веб-серверами в такий спосіб. Запит від ПК виконується, коли користувач вводить веб-адресу, натискає гіперпосилання або заповнює онлайн-форму, таку як пошук. Потім цей запит надсилається провайдеру та направляється через Інтернет на сервер призначення за допомогою механізму, описаного в розділі про протоколи. Потім сервер повертає запитану веб-сторінку, якщо це статична (фіксована) сторінка або якщо для неї потрібне посилання на базу даних, наприклад запит інформації про продукт, він передасть запит на сервер бази даних, а потім поверне це клієнту як динамічно створювану веб-сторінку. Інформація про всі запити сторінок зберігається у файлі журналу транзакцій, у якому записується запит на сторінку, час її виконання та джерело запиту.

Електронний бізнес включає кілька ключових видів діяльності, включаючи вдосконалення бізнес-процесів, покращення комунікацій і забезпечення засобів для безпечного здійснення бізнес-операцій. Електронний бізнес є частиною ширшої Інтернет-економіки, яка охоплює всі види діяльності, пов’язані з використанням Інтернету для комерції. Інтернет-економіка складається з наступних рівнів:

- Інфраструктура Інтернету. Компанії, які надають апаратне, програмне та інше обладнання для Інтернету. Приклади: Інтернет-провайдери, мережеві компанії та виробники ПК і серверів.

- Інфраструктура Інтернет-додатків.Компанії, які надають програмне забезпечення для полегшення транзакцій в Інтернеті. Також компанії, які надають послуги веб-розробки, дизайну та консалтингу. Приклади: виробники програмного забезпечення для веб-розробки, веб-сумісних баз даних і пошукових систем.

- Інтернет-посередники. Компанії, які зв’язують покупців і продавців, наприклад, надаючи вміст або створюючи ринки, де можна вести бізнес. Приклади: туристичні агентства, постачальники контенту та онлайн-брокерські компанії.

- Інтернет-торгівля. Компанії, які продають товари та послуги споживачам або іншим компаніям. Приклади: роздрібні онлайн-магазини, послуги за передплатою або платні послуги та виробники, які продають безпосередньо населенню.

Загалом переваги електронного бізнесу включають зниження витрат, підвищення ефективності та доступ до більших ринків. Автоматизуючи багато адміністративних завдань, пов’язаних із замовленням, постачанням і доставкою товарів або послуг, можна значно знизити вартість типової бізнес-операції. Електронні закупівлі використовуються для скорочення адміністративних витрат і закупівлі товарів за нижчими цінами. Раніше згадувалося, що впровадження підходу до електронного бізнесу може допомогти покращити три основні сфери бізнесу: виробничі процеси, процеси, орієнтовані на клієнта, і процеси внутрішнього управління. З точки зору процесів, орієнтованих на клієнта, наприклад, ефективність обслуговування клієнтів можна підвищити шляхом впровадження служби підтримки на веб-сайті компанії [35]. Окрім допомоги клієнтам, така установа також може зменшувати витрати, зменшуючи тиск на інші служби підтримки, такі як телефонні лінії довіри. Нарешті, впровадження підходу до електронного бізнесу може допомогти компаніям вийти на більший глобальний ринок. Це часто є однією з переваг реструктуризації відносин між виробником, роздрібними торговцями та клієнтами.

Поширеною діяльністю, пов’язаною з електронним бізнесом, є електронна комерція, яку можна описати як використання технології для проведення ділових операцій, таких як купівля та продаж товарів і послуг.

Однак електронна комерція передбачає більше, ніж просто проведення електронних транзакцій; він також охоплює широкий спектр супутніх видів діяльності, таких як післяпродажна підтримка та навіть логістика.

Діяльність електронної комерції можна розділити на п’ять основних типів:

- Бізнес-бізнес (B2B). Угоди відбуваються між компаніями. Приблизно 80 відсотків усієї електронної комерції відноситься до цього типу.

- Бізнес-споживач (B2C). Компанії продають продукцію безпосередньо споживачам. B2C може включати такі дії, як дослідження продукту (де споживачі збирають інформацію та порівнюють ціни) та електронне постачання (де інформаційні продукти доставляються споживачам електронною поштою чи іншими засобами).

- Бізнес-уряд (B2G). Операції відбуваються між компаніями та організаціями державного сектора.

- Від споживача до споживача (C2C). Операції відбуваються між приватними особами. Можливо, найкращими прикладами комерції C2C є сайти онлайн-аукціонів і однорангові системи.

- Мобільна комерція (m-commerce). M-Commerce є відносно новим розвитком і передбачає продаж товарів або послуг за допомогою бездротових технологій, особливо мобільних телефонів.

Основний вибір під час придбання інформаційних систем можна класифікувати як готові (запаковані), спеціальні програми, розроблені власним ІТ-відділом або фірмою програмного забезпечення, і системи, розроблені кінцевим користувачем.

Індивідуальна розробка означає, що інформаційна система розробляється фахівцем з інформаційних систем відповідно до бізнес-вимог програми. Фахівці з інформаційних систем працюватимуть або на компанію, що називається «власною» розробкою на замовлення, або на третю сторону, таку як фірма програмного забезпечення, що називається «розробкою програмного забезпечення на аутсорсингу». Індивідуальна розробка має переваги у створенні програмного забезпечення, адаптованого до точних вимог бізнесу. До недоліків можна віднести вартість, розробка на замовлення є найдорожчим способом розробки нових інформаційних систем. З точки зору індивідуальної розробки за часом, особливо з використанням формальних структурованих методологій розробки, сумно відомі перевищенням часу, із затримками на місяці чи роки, які нерідкі та якісні. Нарешті, з точки зору якості, створене на замовлення програмне забезпечення зазвичай не позбавлене помилок; помилки програмного забезпечення можуть варіюватися від тривіальних до катастрофічних, останні часто пов’язані з поганим аналізом вимог.

До 2020 року вітер може забезпечувати одну десяту потреб нашої планети в електроенергії. Вже сьогодні інноваційне ноу-хау SKF має вирішальне значення для роботи значної частинисвітові вітрові турбіни.

Сила інженерії знань

Готове програмне забезпечення

Закупівля готового програмного забезпечення — це метод придбання, який передбачає пряме придбання попередньо написаної програми, яка використовується кількома компаніями. Цей тип програмного забезпечення є попередньо написаним і доступним для різноманітних апаратних платформ від ПК до мейнфреймів. Готове програмне забезпечення написано, щоб запропонувати широку функціональність, яка підійде широкому колу різних компаній. Перевагою цього широкого спектру функцій є відповідність вимогам великої кількості підприємств. Він також може пропонувати занадто багато функцій для будь-якого конкретного бізнесу, який потім може вважати, що він платить за те, чим не буде користуватися. У той же час це може вимагати від компаній обробки інформації особливим способом, який суперечить тому, як вони зазвичай ведуть бізнес [36]. Крім того, певний стандартний пакет програмного забезпечення може не мати достатніх функцій. Однак основною перевагою готових пакетів програмного забезпечення є їх низька вартість у порівнянні з придбанням програмного забезпечення на замовлення з таким самим рівнем функціональності. Крім того, оскільки пакетне програмне забезпечення було розроблено для комерційного ринку, воно менше страждає від помилок, які вражають програмне забезпечення на замовлення. У разі спеціального готового придбання попередньо написане програмне забезпечення купується у постачальника, але його можна налаштувати відповідно до конкретної компанії. При придбанні готових компонентів різні модулі можна придбати в різних постачальників і створити разом.

**Висновки до розділу**

У даному розділі було визначено, що системи збору інформації для бізнесу виступають інструментом, який сприяє стабільному розвитку в умовах постійної динаміки змін. Вони дозволяють підприємствам бути гнучкими, оперативно реагувати на ринкові виклики та адаптуватися до нових умов, що забезпечує збереження конкурентоспроможності та підвищення ефективності роботи. Розвиток таких систем є стратегічною інвестицією, яка не лише гарантує успіх на поточному етапі, але й створює підґрунтя для довготривалого зростання та процвітання. Також, було розглянуто проблематику інформаційних систем для бізнесу та проаналізовано їхні осносні особливості.

# РОЗДІЛ 2. ДОСЛІДНИЦЬКО-АНАЛІТИЧНА ЧАСТИНА

## 2.1 Аналіз мов програмування

Мова програмування Python є однією з найбільш популярних та універсальних мов у світі, завдяки своїй простоті та зручності для новачків. Вона була створена Гвідо ван Россумом у 1991 році і з того часу здобула величезну популярність завдяки чистому та зрозумілому синтаксису. Python підтримує кілька парадигм програмування, зокрема об'єктно-орієнтоване, процедурне та функціональне програмування, що дозволяє використовувати її в різноманітних сферах.

Однією з головних переваг Python є його легкість у читанні та написанні коду. Завдяки тому, що синтаксис цієї мови дуже схожий на природну мову, програмісти можуть швидше освоїти її та писати зрозумілий код. Наприклад, Python не вимагає використання фігурних дужок для позначення блоків коду, замість цього використовується відступи. Це дозволяє зменшити ймовірність помилок і покращити структуру програм.

Мова також підтримує динамічну типізацію, що означає, що змінні не потребують явного вказання типу під час оголошення. Python сам визначає тип змінної під час виконання програми. Ця особливість робить мову дуже гнучкою, але також може призвести до деяких проблем під час розробки великих проєктів, якщо не проводити достатню перевірку типів [37].

Python широко використовується в різних галузях програмування. Веб-розробка — одна з найпоширеніших сфер використання цієї мови. Популярні веб-фреймворки, такі як Django та Flask, дозволяють створювати потужні веб-додатки за короткий час. Крім того, Python є розповсюдженим інструментом у науці про дані, машинному навчанні, обробці природної мови, автоматизації задач і навіть розробці ігор.

Ще однією сильною стороною Python є його велика стандартна бібліотека та підтримка величезної кількості зовнішніх бібліотек, що дозволяє програмістам легко розширювати можливості своїх програм. Завдяки цьому Python може бути використаний для вирішення задач в абсолютно різних сферах, від аналізу даних до веб-скрапінгу.

Python також має активне та велике співтовариство, що сприяє розвитку мови. Багато розробників діляться своїми бібліотеками, інструментами та рішеннями, що дозволяє ще швидше розвиватися проектам і вирішувати нові технічні проблеми. Крім того, Python має відмінну документацію, що допомагає новачкам та досвідченим розробникам швидко знаходити відповіді на свої питання.

JavaScript — це мова програмування, яка стала основним інструментом для створення інтерактивних та динамічних веб-додатків. Вперше вона була розроблена в середині 1990-х років Брэндоном Айхом, співробітником компанії Netscape. Спочатку JavaScript використовувався лише для додавання інтерактивних елементів на веб-сторінках, таких як кнопки, форми або анімації, але з часом мова значно розширила свої можливості.

JavaScript є мовою сценаріїв, що працює на стороні клієнта. Це означає, що код JavaScript виконується безпосередньо в браузері користувача, на відміну від серверних мов програмування, таких як PHP чи Python. Завдяки цьому JavaScript дозволяє створювати високопродуктивні та швидкі веб-додатки, оскільки користувач не повинен кожного разу звертатися до сервера для оновлення контенту або взаємодії з інтерфейсом.

Однією з основних особливостей JavaScript є його асинхронність. Завдяки механізму обробки асинхронних операцій через колбеки, обіцянки (promises) та async/await, JavaScript дозволяє ефективно працювати з мережевими запитами, таймерами та іншими ресурсами, не блокуючи виконання основного потоку програми. Це важливо для веб-додатків, де необхідно обробляти безліч одночасних запитів без затримок для користувача.

JavaScript підтримує функціональне, об'єктно-орієнтоване та імперативне програмування, що робить його гнучким та універсальним інструментом для різних видів розробки. Кожен об'єкт в JavaScript є динамічним і може бути змінений під час виконання програми. Це дозволяє програмістам легко маніпулювати даними, створювати нові об'єкти та функції під час роботи програми.

У JavaScript також є можливість працювати з потужними веб-API, що дозволяє програмістам взаємодіяти з веб-браузером та зовнішніми сервісами. Наприклад, можна використовувати API для доступу до геолокації, роботи з камерою, мікрофоном, зберігання даних в браузері через LocalStorage або IndexedDB, а також створення анімацій з використанням CSS або WebGL.

JavaScript значною мірою став стандартом для веб-розробки завдяки численним бібліотекам та фреймворкам, таким як React, Angular, Vue.js та Node.js. Ці інструменти значно спрощують створення складних веб-додатків та веб-сайтів, забезпечуючи компонування інтерфейсу, обробку даних та роботу з сервером.

Node.js — це серверне середовище для виконання JavaScript, яке дозволяє використовувати цю мову не тільки для фронтенд-розробки, але й для серверної логіки. Це дозволяє програмістам писати всю програму на одній мові, що спрощує процес розробки та підтримки проектів.

Однією з причин великої популярності JavaScript є його спільнота. Мова має величезну кількість розробників, які створюють нові бібліотеки, інструменти та фреймворки, що дозволяють з легкістю вирішувати складні завдання. Крім того, JavaScript має велику кількість онлайн-ресурсів, документації та навчальних матеріалів, що робить його доступним для новачків.

Однак, як і будь-яка інша мова, JavaScript має свої особливості та недоліки. Наприклад, через динамічну типізацію можуть виникати проблеми з типами даних, а асинхронна модель може бути складною для розуміння новачками. Тим не менш, завдяки потужним інструментам, які були створені для вирішення цих проблем, JavaScript залишається однією з найважливіших мов програмування для сучасних веб-додатків [37].

C# (C-Sharp) — це мова програмування, що була розроблена компанією Microsoft на початку 2000-х років як частина платформи .NET. Її створення було спрямоване на надання потужного інструмента для розробки програмного забезпечення, який поєднує переваги сучасних мов програмування з безпекою типів і підтримкою об'єктно-орієнтованого підходу. C# є мовою з багатим функціоналом, яка має статичну типізацію, що дозволяє забезпечити високу надійність коду та ефективність роботи програм.

Основною особливістю C# є його сильна орієнтація на об'єктно-орієнтоване програмування. Кожен елемент програми в C# є об'єктом або взаємодіє з об'єктами через методи та властивості. Завдяки цьому можна створювати масштабовані та підтримувані програми. C# також підтримує такі концепції, як успадкування, інтерфейси, поліморфізм і абстракцію, що дозволяє ефективно організовувати код і знижувати його складність.

Окрім об'єктно-орієнтованого підходу, C# підтримує й інші парадигми програмування, зокрема функціональне програмування. З новими версіями мови були введені функції, такі як лямбда-вирази, анонімні методи та підтримка делегатів, які дозволяють розробникам використовувати функціональний стиль програмування для певних завдань. C# також має потужні інструменти для роботи з колекціями, LINQ (Language Integrated Query), що дозволяє розробникам зручно працювати з даними, що зберігаються у вигляді колекцій.

Важливою особливістю C# є його інтеграція з платформою .NET, що дає доступ до величезної кількості бібліотек і фреймворків для різноманітних завдань, від роботи з базами даних до створення графічних інтерфейсів користувача. Для розробки веб-додатків C# часто використовується разом з ASP.NET, потужним фреймворком для створення серверних додатків, що забезпечує високу продуктивність і зручність у розробці.

C# також підтримує багатозадачність і асинхронне програмування, що дозволяє ефективно обробляти одночасні запити або задачі без блокування основного потоку. Ключовими інструментами для цього є ключові слова `async` і `await`, які дозволяють писати асинхронний код у зручному синхронному стилі, що значно полегшує розуміння та налагодження програми.

Мова також забезпечує високий рівень безпеки типів, що дозволяє уникати багатьох типових помилок, пов'язаних із неправильним використанням змінних та об'єктів. Завдяки строгій типізації компілятор C# може виявляти помилки на етапі компіляції, що дозволяє знижувати кількість помилок у фінальному коді.

C# активно використовується в розробці програм для Windows, але з появою .NET Core (тепер .NET 5 і новіші версії) мова стала кросплатформенною. Це означає, що з C# можна розробляти програми не тільки для Windows, а й для Linux та macOS. .NET Core також дозволяє використовувати C# для створення серверних додатків, мікросервісів, контейнеризованих додатків та багато іншого.

Велика кількість інструментів для розробки, таких як Visual Studio, допомагає програмістам ефективно писати код, налагоджувати його та проводити тестування. Система керування залежностями через NuGet дозволяє легко додавати сторонні бібліотеки та компоненти, що значно прискорює процес розробки.

Серед популярних застосувань C# можна виділити розробку настільних додатків, мобільних додатків для iOS і Android через Xamarin, а також створення ігор за допомогою Unity. Unity, популярний ігровий движок, активно використовує C# як основну мову програмування, що дозволяє розробляти ігри для різних платформ, включаючи консолі, ПК та мобільні пристрої.

C# має активне співтовариство розробників і великі ресурси для навчання, що робить його популярним вибором серед професіоналів і новачків. Завдяки своїй потужності, багатофункціональності та підтримці сучасних технологій, C# продовжує залишатися однією з найбільш важливих і перспективних мов програмування на сьогоднішній день.

Java — це одна з найпопулярніших мов програмування, розроблена компанією Sun Microsystems у 1995 році, а пізніше придбана Oracle. Мова була створена з метою створення платформонезалежних програм, тобто програм, які можуть працювати на будь-якому пристрої чи операційній системі без необхідності змінювати сам код. Однією з основних переваг Java є її принцип «write once, run anywhere» (напиши один раз — працюй будь-де), що дозволяє програмам, написаним на Java, працювати на різних платформах завдяки використанню віртуальної машини Java (JVM).

Java є мовою з сильною статичною типізацією, що означає, що типи даних змінних і об'єктів визначаються під час компіляції, і компілятор перевіряє ці типи перед виконанням коду. Це забезпечує високу надійність і знижує кількість помилок, оскільки більшість проблем з типами можна виявити на етапі компіляції, а не під час виконання програми.

Однією з основних концепцій Java є об'єктно-орієнтоване програмування (ООП), що дозволяє розробникам створювати програмне забезпечення, орієнтуючись на об'єкти та їх взаємодію. У Java все є об'єктами, за винятком примітивних типів даних. Програмування на Java базується на чотирьох основних принципах ООП: інкапсуляція, успадкування, поліморфізм та абстракція. Це дозволяє створювати добре організовані та масштабовані програми, де кожен об'єкт має свої властивості та методи, а також може взаємодіяти з іншими об'єктами.

Java підтримує багатозадачність, що дозволяє виконувати кілька операцій одночасно. Багатозадачність досягається через потоки, які дозволяють програмам виконувати паралельно декілька задач, що особливо важливо для розробки високопродуктивних додатків, таких як сервери, ігри або програми для обробки великих обсягів даних. Підтримка багатозадачності в Java включає механізми синхронізації, які дозволяють уникати проблем з доступом до спільних ресурсів.

Іншою важливою особливістю Java є її велика екосистема бібліотек і фреймворків. Java пропонує широкий набір бібліотек для роботи з графікою, базами даних, мережею, веб-додатками, а також для розробки корпоративних програм. Одними з найбільш відомих фреймворків є Spring та Hibernate. Spring забезпечує зручне середовище для розробки корпоративних додатків, включаючи підтримку ін'єкції залежностей, транзакцій, безпеки, веб-розробки та багато іншого. Hibernate, у свою чергу, є популярним фреймворком для роботи з базами даних і ORM (Object-Relational Mapping), що дозволяє зручно працювати з реляційними базами даних.

Java має свою віртуальну машину (JVM), яка є основою її платформонезалежності. JVM дозволяє виконувати байт-код Java на будь-якому пристрої, на якому встановлено JVM, незалежно від операційної системи. Коли програма компілюється, Java-код перетворюється на байт-код, який потім виконується JVM. Це робить Java ідеальною мовою для створення кросплатформенних додатків.

Завдяки своїй платформонезалежності, потужним можливостям ООП, багатозадачності та величезній кількості інструментів для розробки, Java часто використовується в розробці корпоративних додатків, веб-додатків, мобільних додатків (особливо для Android), а також для створення серверного програмного забезпечення. Android, одна з найбільш популярних мобільних операційних систем, використовує Java для розробки додатків, хоча з часом на платформу Android почала інтегруватися і Kotlin.

Java також має потужну спільноту розробників і багатий набір інструментів для налагодження, тестування та оптимізації програм, таких як Eclipse, IntelliJ IDEA, NetBeans, а також бібліотеки для тестування, зокрема JUnit.

Крім того, Java активно використовує концепції сучасного програмування, такі як лямбда-вирази, які були введені в Java 8 і забезпечують функціональне програмування, що дозволяє писати більш компактний і читабельний код для обробки колекцій, подій і асинхронних операцій.

Java також відома своєю стабільністю та безпекою. Завдяки використанню автоматичного управління пам'яттю через систему збору сміття (garbage collection) Java мінімізує ризик витоків пам'яті, що робить її однією з найкращих мов для створення надійних та стабільних додатків.

Завдяки своїй багатофункціональності, надійності та великій кількості бібліотек і фреймворків, Java залишається однією з основних мов програмування для розробки програмного забезпечення в різних галузях, від веб-розробки до великих корпоративних систем.

## 2.2 Аналіз бібліотек

Node.js — це середовище виконання JavaScript, яке дозволяє запускати JavaScript-код поза браузером. Воно побудоване на базі движка V8 від Google, який відповідає за виконання JavaScript-коду у браузері Chrome. Однак на відміну від браузера, Node.js надає можливість запускати JavaScript на сервері, що відкриває нові можливості для розробки високопродуктивних серверних додатків. Node.js було створено Райаном Далем у 2009 році, і з того часу воно стало одним з найпопулярніших інструментів для створення серверних додатків, особливо для розробки веб-серверів, реального часу і мікросервісів.

Однією з головних особливостей Node.js є його асинхронний і подієвий (event-driven) підхід до обробки запитів. На відміну від традиційних синхронних моделей серверів, де кожен запит блокує виконання інших, Node.js використовує однопоточний цикл подій (event loop), який дозволяє обробляти численні запити одночасно без блокування основного потоку. Завдяки цьому Node.js здатний обробляти велику кількість запитів одночасно, що робить його ідеальним для створення швидких і масштабованих серверних додатків, таких як веб-сервери, чат-системи або додатки для реального часу.

Ще однією важливою особливістю є використання JavaScript як на клієнтській, так і на серверній стороні. Це дозволяє розробникам використовувати одну мову для створення як фронтенду, так і бекенду, що спрощує процес розробки, дозволяє швидше переходити між різними частинами проекту і знижує складність коду. Завдяки великій кількості доступних бібліотек і модулів через npm (Node Package Manager), Node.js дозволяє ефективно вирішувати широкий спектр завдань, таких як робота з базами даних, обробка файлів, реалізація веб-сервісів, обробка запитів від клієнтів і багато іншого.

Одним з найпопулярніших застосувань Node.js є розробка RESTful API, де його асинхронна природа дозволяє зручно обробляти одночасно велику кількість запитів до сервера. За допомогою таких фреймворків, як Express.js, Node.js надає зручний інтерфейс для створення серверних додатків з обробкою запитів HTTP. Express.js є мінімалістичним, але потужним фреймворком для Node.js, який дозволяє швидко створювати маршрути для обробки запитів, інтегрувати middleware для додаткової обробки запитів і відповідей, а також підтримує зручну роботу з шаблонами та сесіями.

Node.js також активно використовує концепції функціонального програмування, що дозволяє писати ефективний і модульний код для обробки подій. Лямбда-вирази, функції зворотного виклику (callback functions) та проміси (promises) дозволяють легко працювати з асинхронними операціями, наприклад, зчитуванням файлів, запитами до баз даних або викликами HTTP-запитів. Останнім часом в Node.js також активно використовуються async/await конструкції, що дозволяють писати асинхронний код у більш синхронному стилі, що значно підвищує зручність роботи з асинхронними операціями.

Іншою важливою перевагою Node.js є його ефективна робота з великими обсягами даних і реальним часом. Завдяки асинхронному і подієвому механізму обробки запитів, Node.js ідеально підходить для створення додатків, які вимагають високої продуктивності та швидкої обробки даних. Це особливо корисно для чат-систем, ігор, відео- та голосових дзвінків у реальному часі, а також для роботи з великими потоками даних, наприклад, при створенні додатків для обробки логів або стрімінгових сервісів.

Node.js також активно використовується для розробки мікросервісів — архітектурного стилю, який полягає в розподілі великих додатків на менші, незалежні сервіси, що взаємодіють між собою через API. Завдяки своїй здатності швидко обробляти запити і добре працювати з великими обсягами даних, Node.js є чудовим вибором для розробки мікросервісів, що дозволяє розподіляти навантаження між кількома сервісами і масштабувати додаток.

Node.js не обмежується тільки серверними додатками. З ним можна створювати інструменти командного рядка (CLI), скрипти автоматизації, а також працювати з базами даних, що дозволяє використовувати Node.js в якості універсального інструмента для розробки різноманітних типів програм. Наприклад, з допомогою Node.js можна працювати з популярними базами даних, такими як MongoDB або MySQL, а також створювати інтерактивні інтерфейси за допомогою бібліотек, таких як Electron, для розробки десктопних додатків.

Що стосується продуктивності, то Node.js забезпечує дуже високі швидкості завдяки тому, що він працює на основі движка V8, оптимізованого для швидкого виконання JavaScript-коду. Його однопоточна модель дозволяє обробляти запити з мінімальними витратами пам'яті, що робить його дуже ефективним у використанні серверних ресурсів, порівняно з іншими технологіями.

У результаті, Node.js є потужним інструментом для розробки масштабованих, високопродуктивних додатків, які можуть обробляти великі обсяги запитів і працювати в режимі реального часу. Завдяки своїй асинхронній архітектурі, великій екосистемі бібліотек і підтримці сучасних підходів до програмування, Node.js став одним з найбільш використовуваних технологій у сучасній веб-розробці.

React — це популярна бібліотека для створення інтерфейсів користувача, розроблена компанією Facebook і зараз підтримується великою спільнотою розробників. Вона була вперше представлена у 2013 році і відтоді стала однією з основних технологій для розробки сучасних веб-додатків. React дозволяє створювати компоненти, які є незалежними, повторно використовуваними частинами інтерфейсу, що значно спрощує процес розробки, тестування і підтримки великих веб-додатків.

Основною ідеєю React є компоненти. Кожен компонент є частиною інтерфейсу, яка має свою логіку і стан. Компоненти можуть бути як функціональними, так і класовими, хоча останнім часом більшість нових розробок реалізуються саме через функціональні компоненти з використанням хуків (hooks). Хуки дозволяють компоненти працювати з власним станом (state) та ефектами (side effects), що раніше було доступно лише в класових компонентах.

React працює за принципом декларативного програмування, що означає, що замість того, щоб описувати кожен крок, як змінюється інтерфейс користувача, розробник просто визначає, як повинен виглядати інтерфейс на кожному кроці. Наприклад, при зміні стану, React автоматично оновлює тільки ті частини інтерфейсу, які змінилися, замість того, щоб переробляти все вікно або сторінку цілком. Це дозволяє зменшити обсяг необхідних обчислень і покращити ефективність.

Один з основних елементів React — це Virtual DOM (віртуальний DOM). DOM (Document Object Model) — це структура, яка відображає HTML-документ в пам'яті браузера. Замість того щоб відразу змінювати реальний DOM при кожній зміні, React спочатку оновлює Virtual DOM, що дозволяє порівняти поточний стан DOM з новим і тільки потім вносити мінімальні зміни в реальний DOM. Це підвищує продуктивність, оскільки маніпуляції з реальним DOM є дорогими в плані часу виконання.

Іншою важливою особливістю є використання JSX — розширення синтаксису JavaScript, яке дозволяє описувати структуру інтерфейсу безпосередньо в JavaScript-коді. JSX нагадує HTML, але насправді це синтаксичний цукор, який компілюється в JavaScript-код за допомогою транслятора, такого як Babel. Це дозволяє зручно змішувати логіку з розміткою, що робить код більш зрозумілим і компактним.

Для управління станом в React застосовуються різні підходи в залежності від масштабу і складності додатку. У простих компонентах стан можна зберігати локально, використовуючи хук `useState`. Для більш складних додатків часто використовуються більш масштабовані рішення для управління станом, такі як Context API або бібліотеки на кшталт Redux. Redux є однією з найпопулярніших бібліотек для управління глобальним станом, і хоча його використання може бути складнішим, він дає розробникам велику гнучкість при керуванні великими додатками з численними залежностями.

React також дозволяє створювати односторінкові додатки (SPA — Single Page Applications), де інтерфейс не перезавантажується повністю при навігації між сторінками. Це дозволяє забезпечити швидшу та безперебійну взаємодію з користувачем, оскільки завантаження контенту відбувається динамічно через асинхронні запити до сервера. Для маршрутизації в таких додатках використовується бібліотека `react-router`, яка дозволяє керувати URL-адресами і відповідними компонентами без перезавантаження сторінки.

Ще однією важливою особливістю React є підтримка компонування, що дозволяє легко будувати складні інтерфейси з маленьких, самодостатніх компонентів. Це дає змогу зменшити дублювання коду та підвищити його читабельність. Компоненти можуть бути створені з чітко визначеними вхідними і вихідними властивостями (props), що робить їх універсальними і повторно використовуваними в різних частинах додатку.

React має потужну екосистему, що включає безліч сторонніх бібліотек для різних завдань. Наприклад, для роботи з формами можна використовувати бібліотеку `formik`, для анімацій — `react-spring` або `framer-motion`, а для роботи з ефективним завантаженням даних — `react-query`. Завдяки великій спільноті розробників, React продовжує активно розвиватися, і з’являються нові інструменти, що дозволяють спрощувати розробку додатків.

React також є основою для створення мобільних додатків за допомогою фреймворку React Native. React Native дозволяє використовувати ті самі концепції і компоненти, що й у веб-додатках на React, але при цьому створювати нативні мобільні додатки для платформ iOS та Android. Завдяки цьому можна писати додатки для мобільних пристроїв, використовуючи один і той самий код для різних платформ, що значно знижує витрати на розробку і підтримку.

У загальному React є потужною і гнучкою бібліотекою для створення інтерфейсів користувача, яка добре підходить як для малих проектів, так і для великих, складних додатків. Його декларативний підхід, компонувальна архітектура, швидка робота з Virtual DOM і велика спільнота роблять React одним з найкращих виборів для сучасної веб-розробки.

ASP.NET — це потужний фреймворк для розробки веб-додатків, який був створений компанією Microsoft. Він є частиною .NET платформи і призначений для розробки динамічних веб-сайтів, веб-додатків і API. Однією з основних переваг ASP.NET є його висока продуктивність, безпека та інтеграція з іншими технологіями Microsoft, що робить його привабливим для розробників, які працюють у середовищі Windows, а також для корпоративних додатків, де потрібна висока надійність і масштабованість.

ASP.NET був запущений у 2002 році, і з того часу зазнав значних змін і оновлень. Одним з найбільших кроків уперед було введення ASP.NET Core — версії фреймворка, яка підтримує крос-платформеність, що дозволяє запускати додатки не тільки на Windows, але й на Linux та macOS. Це значно розширило аудиторію розробників і дозволило використовувати ASP.NET для більш широкого кола проектів.

ASP.NET працює на основі принципів серверної обробки запитів, що означає, що веб-сторінки генеруються на сервері та відправляються користувачеві у вигляді HTML, CSS та JavaScript. Однак, на відміну від статичних сторінок, які можна створювати просто з HTML і CSS, ASP.NET дозволяє інтегрувати серверну логіку, працювати з базами даних, обробляти форми, автентифікацію користувачів і виконувати багато інших завдань без необхідності вручну оновлювати сторінки.

Одним із основних компонентів фреймворку є модель програмування на основі MVC (Model-View-Controller). Ця архітектура дозволяє розділяти логіку додатка на три основні частини: модель (дані), вид (інтерфейс користувача) та контролер (логіка бізнесу). Такий підхід сприяє кращій організації коду, зменшенню залежностей і покращенню тестованості.

ASP.NET Core підтримує також веб-API, що дозволяє створювати RESTful сервіси, які можна використовувати для розробки мобільних додатків або для інтеграції з іншими системами. API, створене за допомогою ASP.NET Core, може бути запитане через HTTP запити, що дозволяє інтегрувати різні системи через стандартні інтерфейси, такі як JSON або XML.

Однією з найбільших переваг ASP.NET є його підтримка роботи з базами даних через Entity Framework. Entity Framework (EF) — це ORM (Object-Relational Mapping) бібліотека, яка дозволяє розробникам працювати з реляційними базами даних через об’єкти .NET. Це значно спрощує процес роботи з даними, оскільки розробник може працювати з об’єктами, а не писати складні SQL запити вручну.

Для роботи з безпекою в ASP.NET є вбудовані можливості автентифікації та авторизації. ASP.NET Identity — це бібліотека, яка дозволяє створювати систему реєстрації та входу користувачів, обробляти ролі та права доступу, а також підтримувати двофакторну автентифікацію. Ця система інтегрується з різними постачальниками автентифікації, такими як Google, Facebook, Twitter або Azure Active Directory.

ASP.NET також підтримує вбудовані засоби для побудови зручних інтерфейсів користувача через використання Razor Pages. Razor — це шаблонізатор, який дозволяє генерувати HTML на сервері, інтерпретуючи C# код безпосередньо в HTML-шаблонах. Це дає можливість ефективно працювати з даними на сервері та зручно відображати їх на веб-сторінках.

Ще однією ключовою можливістю ASP.NET є підтримка мікросервісної архітектури. Це дозволяє створювати додатки, які складаються з окремих, незалежних компонентів (мікросервісів), які можуть бути розгорнуті та оновлені окремо один від одного. Мікросервіси можуть використовувати різні технології та бази даних, що підвищує гнучкість і масштабованість додатків.

Для покращення продуктивності і масштабованості додатків в ASP.NET Core є вбудована підтримка асинхронного програмування. Завдяки використанню async/await можна значно покращити відгук серверів, зменшуючи час блокування під час обробки запитів до бази даних або інших зовнішніх ресурсів. Це дозволяє обробляти більше запитів за менший час, що особливо важливо для веб-додатків з високим навантаженням.

ASP.NET Core також підтримує автоматичне тестування, що дозволяє писати юніт-тести для перевірки правильності роботи різних частин додатку. За допомогою вбудованих інструментів і бібліотек, таких як xUnit або NUnit, розробники можуть створювати тести, які автоматично перевіряють бізнес-логіку, функціональність API та інші компоненти.

Ще однією важливою перевагою ASP.NET є його інтеграція з іншими інструментами і технологіями Microsoft, такими як Azure, Visual Studio і SQL Server. Це дозволяє створювати, тестувати і розгортати додатки з використанням потужних інструментів розробки і хмарних платформ, що значно спрощує роботу і підвищує ефективність розробників.

ASP.NET є дуже потужним і гнучким фреймворком для розробки веб-додатків, який надає розробникам безліч інструментів і можливостей для створення масштабованих, продуктивних і безпечних веб- та мобільних додатків. Завдяки своїй підтримці крос-платформеності, інтеграції з іншими Microsoft технологіями та великим можливостям для роботи з даними і безпекою, ASP.NET є одним з найпопулярніших фреймворків для корпоративних і веб-додатків на ринку.

Spring Boot — це частина популярного фреймворку Spring, яка спрощує розробку і налаштування Java-додатків. Spring Boot дозволяє швидко створювати автономні, продуктивні веб-додатки або мікросервіси без необхідності виконувати складні налаштування і конфігурації. Це знижує витрати часу на початкову налаштування і дозволяє зосередитись на розробці бізнес-логіки.

Spring Boot є одним з найпопулярніших фреймворків для розробки серверних додатків на Java завдяки своїй здатності спрощувати процеси конфігурації та розгортання. Замість того, щоб вручну налаштовувати численні компоненти (як це часто трапляється у великих Java-проектах), Spring Boot автоматизує багато задач, надаючи стандартні налаштування за замовчуванням, що дозволяє розробникам сконцентруватися на логіці додатка.

Однією з ключових особливостей Spring Boot є підтримка принципу "convention over configuration" (конвенція замість конфігурації). Це означає, що Spring Boot надає зручні за замовчуванням конфігурації для більшості сценаріїв, що значно спрощує налаштування додатка. Наприклад, замість того, щоб вручну налаштовувати з’єднання з базою даних, розробник може просто вказати конфігураційні параметри у властивостях або файлі `application.properties`, і Spring Boot самостійно налаштує з'єднання.

Spring Boot дозволяє створювати автономні додатки, що можна запустити як окремі процеси, без необхідності додаткового налаштування серверів (як Tomcat, Jetty або Undertow). Всі необхідні компоненти для запуску веб-сервера включені в сам додаток, що дозволяє просто запустити його за допомогою команд `java -jar` або через IDE. Це значно спрощує процес розгортання, оскільки додаток можна просто запустити на будь-якому сервері або в будь-якому середовищі, яке підтримує Java.

Ще одна значна перевага Spring Boot полягає у підтримці мікросервісної архітектури. Завдяки своїй легкості і зручності налаштування, Spring Boot є ідеальним рішенням для розробки мікросервісів. Кожен мікросервіс можна запускати як автономний додаток, що має свою логіку та базу даних, і все це можна управляти за допомогою таких інструментів, як Spring Cloud. Spring Boot інтегрується з різними бібліотеками для побудови мікросервісів, такими як Spring Cloud Config для централізованого налаштування, Spring Cloud Eureka для реєстрації і відкриття сервісів і Spring Cloud Gateway для маршрутизації запитів між мікросервісами.

Для роботи з базами даних Spring Boot підтримує ряд ORM-бібліотек, зокрема Hibernate, яка інтегрується через Spring Data JPA. Це дозволяє зручно працювати з реляційними базами даних, використовуючи об’єктно-реляційне відображення (ORM), що значно спрощує взаємодію з базами даних і дозволяє писати менш складний код для доступу до даних. Завдяки використанню анотацій, таких як `@Entity`, `@Repository`, `@Service` та інші, програміст може легко визначати моделі, репозиторії та сервіси без необхідності писати багато коду вручну.

Spring Boot також включає в себе зручні інструменти для створення RESTful API. З допомогою анотацій, таких як `@RestController`, `@RequestMapping`, `@GetMapping`, `@PostMapping` та інших, можна легко створювати API для взаємодії з клієнтами чи іншими мікросервісами. Це дозволяє створювати масштабовані та продуктивні додатки для мобільних і веб-клієнтів. Для створення документації API можна інтегрувати бібліотеки, такі як Swagger або Springfox, що автоматично генерують документацію на основі анотацій.

Ще однією важливою характеристикою Spring Boot є підтримка тестування. Spring Boot інтегрується з різними тестовими фреймворками, такими як JUnit, Mockito, та інші. Це дозволяє легко писати юніт-тести для окремих компонентів додатка, а також інтеграційні тести для перевірки взаємодії між різними частинами системи. За допомогою аннотацій, таких як `@SpringBootTest`, можна автоматизувати тестування на різних рівнях.

Spring Boot підтримує різні профілі конфігурації, що дозволяє мати різні налаштування для різних середовищ, наприклад, для розробки, тестування та продакшн-середовища. Це дає можливість зберігати окремі налаштування для кожного середовища без необхідності вручну змінювати конфігураційні файли. Це також дозволяє зручно перемикатися між налаштуваннями для різних оточень під час розробки та тестування додатків.

Оскільки Spring Boot є частиною екосистеми Spring, він добре інтегрується з іншими продуктами та бібліотеками, що надає додаткову гнучкість при розробці додатків. Наприклад, можна використовувати Spring Security для реалізації механізмів автентифікації та авторизації, Spring Batch для обробки великих обсягів даних, Spring WebFlux для реактивного програмування, а також Spring Integration і Spring AMQP для роботи з різними протоколами обміну повідомленнями.

Однією з головних переваг Spring Boot є його велика і активна спільнота розробників, яка постійно підтримує фреймворк і надає різноманітні ресурси для навчання, а також велику кількість бібліотек і плагінів для інтеграції з іншими технологіями.

В підсумку, Spring Boot є потужним, гнучким і зручним інструментом для розробки сучасних веб-додатків, мікросервісів і RESTful API. Завдяки своїй легкості в налаштуванні, широким можливостям інтеграції та підтримці мікросервісної архітектури, Spring Boot є одним з найбільш популярних фреймворків для розробки серверних додатків на Java.

Flask — це мікрофреймворк для розробки веб-додатків на Python, який став популярним завдяки своїй простоті, гнучкості та невеликому розміру. Він є легким і мінімалістичним рішенням, яке дозволяє розробникам швидко створювати веб-додатки з малою кількістю налаштувань та конфігурацій. Flask часто використовують для створення RESTful API, маленьких веб-додатків, а також для більш складних проектів, де потрібна гнучкість у налаштуваннях.

Flask був розроблений Арміном Ронехером в 2010 році і з того часу здобув велику популярність серед Python-розробників. Однією з головних переваг Flask є його мікрофреймворк, що означає, що він надає лише основний функціонал для веб-розробки, залишаючи решту функціональностей на розсуд розробника. Завдяки такому підходу, Flask ідеально підходить для проектів, де потрібно більше контролю над кожним аспектом додатка.

Одна з основних причин популярності Flask — це його простота та лаконічність. Для того, щоб створити простий веб-додаток, достатньо кількох рядків коду. Основні компоненти Flask включають маршрутизацію (обробка запитів), шаблони (для рендерингу HTML) і механізм для обробки форм та баз даних. Завдяки цьому Flask дуже підходить для новачків у веб-розробці, оскільки він не вимагає великої кількості конфігурацій і дозволяє швидко почати роботу.

Однією з ключових особливостей Flask є маршрутизація, яка дозволяє зв'язати URL-запити з функціями. Це здійснюється через декоратори, такі як `@app.route()`, які вказують Flask, яку функцію викликати для певного URL.

Flask дозволяє обробляти різні типи HTTP-запитів, такі як GET, POST, PUT і DELETE, за допомогою параметра `methods`, який вказує, які методи підтримуються для конкретного маршруту. Це дозволяє гнучко обробляти запити від клієнтів і розробляти різні типи API.

Для рендерингу динамічного контенту в Flask використовується шаблонізатор Jinja2. Шаблони дозволяють комбінувати статичний HTML з динамічними даними, що передаються з серверу. Наприклад, можна створювати шаблони для рендерингу веб-сторінок з даними з бази даних або передавати змінні прямо в шаблон, щоб відображати результат на сторінці/

Flask також підтримує обробку форм, що дозволяє створювати веб-сторінки для введення даних користувачем. Це робиться за допомогою спеціальних об'єктів, таких як `request.form`, які дозволяють отримувати дані з форм, що надійшли через POST-запити. Flask також підтримує перевірку даних і валідаторів, що дозволяє забезпечити правильність введення.

Ще однією важливою функціональністю Flask є підтримка роботи з базами даних. Flask не вимагає конкретної ORM (Object-Relational Mapping) системи, але найпопулярнішою бібліотекою для роботи з базами даних у Flask є SQLAlchemy. SQLAlchemy є потужним ORM, який дозволяє легко створювати, змінювати та запитувати дані в реляційних базах даних. З допомогою Flask-SQLAlchemy можна інтегрувати SQLAlchemy у Flask-додаток і використовувати її функціональність для роботи з базами даних, не пишучи SQL-код вручну.

Flask також підтримує розширення, які надають додаткові функціональності для роботи з різними аспектами веб-розробки. Однією з таких розширень є Flask-Login для автентифікації користувачів, Flask-WTF для роботи з веб-формами, Flask-Mail для роботи з електронною поштою і багато інших. Це дозволяє легко розширювати функціональність Flask і інтегрувати додаткові можливості, коли це необхідно.

Для тестування Flask-додатків можна використовувати вбудовані інструменти, такі як `Flask.test\_client()`, які дозволяють створювати тестові клієнти і автоматизувати процес тестування запитів та відповідей. Flask також інтегрується з популярними тестовими фреймворками, такими як `unittest` або `pytest`, що дозволяє писати юніт-тести для додатків.

Оскільки Flask є дуже легким і гнучким фреймворком, він дозволяє розробникам будувати як прості додатки, так і складні системи, залежно від вимог проекту. Flask не має великих залежностей і не нав'язує складних налаштувань, що робить його чудовим вибором для розробки мікросервісів, RESTful API та невеликих веб-додатків.

Завдяки своїй простоті та мінімалізму, Flask також дуже популярний серед стартапів і команд, що працюють над швидким прототипуванням. Його легка інтеграція з іншими бібліотеками та фреймворками дає можливість швидко адаптуватися під будь-які потреби проекту.

В підсумку, Flask є відмінним вибором для тих, хто хоче створювати прості, масштабовані та гнучкі веб-додатки. Його легкість в освоєнні, потужні можливості для роботи з шаблонами, формами та базами даних, а також велика кількість доступних розширень роблять Flask однією з найпопулярніших платформ для розробки веб-додатків на Python.

Django — це високорівневий фреймворк для веб-розробки на Python, який дозволяє швидко створювати складні веб-додатки з високою продуктивністю та зручним адмініструванням. Django був створений у 2005 році командою розробників з сайту журналістів, що вимагали потужного і масштабованого інструменту для створення динамічних веб-ресурсів. Фреймворк надає вбудовані функції для обробки HTTP-запитів, рендерингу шаблонів, роботи з базами даних, автентифікації користувачів і багато іншого, що дозволяє значно прискорити процес розробки.

Однією з головних особливостей Django є його принцип "бatteries-included" — це означає, що в рамках фреймворка вже є все необхідне для створення повноцінного веб-додатка. Django забезпечує багато вбудованих компонентів, таких як система аутентифікації користувачів, система управління сесіями, обробка форм, маршрутизація запитів, ORM (Object-Relational Mapping) для роботи з базами даних, інструменти для тестування та інші важливі функції. Такий підхід дозволяє розробникам зосередитись на створенні бізнес-логіки, а не на розробці базових функціональностей.

Однією з основних особливостей Django є система маршрутизації, яка дозволяє обробляти HTTP-запити за допомогою URL-шляхів. У Django маршрути задаються в файлі `urls.py`, де можна визначити, який шаблон або функція повинна бути викликана для конкретного запиту. Наприклад, маршрут може виглядати так:

Тут при зверненні до кореневого URL (`/`), буде викликана функція `home` з модуля `views`. Функція `home` може рендерити шаблон або повертати інші відповіді.

Ще однією важливою складовою Django є система шаблонів (template engine), яка дозволяє створювати динамічні веб-сторінки. Вона забезпечує можливість вставляти Python-вирази в HTML-шаблони. Наприклад, можна передавати змінні в шаблони для динамічного відображення контенту.

Django також підтримує роботу з базами даних через свою власну ORM. ORM дозволяє працювати з базами даних, використовуючи Python-класи замість SQL-запитів. Наприклад, можна створити модель для користувача та зберігати дані в базі даних, використовуючи прості Python-методи:

Моделі Django дозволяють виконувати операції над даними (додавати, оновлювати, видаляти) через прості методи Python, і при цьому фреймворк автоматично генерує SQL-запити. Django підтримує кілька баз даних, зокрема PostgreSQL, MySQL, SQLite та Oracle.

Ще одна потужна функція Django — це система адміністрування. Вбудоване адміністрування дозволяє розробникам швидко створювати панелі управління для додатків. Django автоматично генерує інтерфейс для роботи з моделями, що дозволяє здійснювати операції з даними без необхідності писати окремі функції для кожної таблиці. Це значно полегшує керування даними під час розробки та тестування:

Ще однією важливою особливістю є система форм. Django надає потужні інструменти для обробки форм, включаючи валідацію даних, генерацію HTML-кодів форм і обробку помилок. Django дозволяє легко створювати форми для створення або оновлення об'єктів моделей, використовуючи форми, пов'язані з моделями (ModelForm).

Django також підтримує створення RESTful API через розширення, наприклад, Django REST Framework (DRF). Це дозволяє розробникам створювати веб-сервіси для обміну даними між клієнтами та сервером. DRF надає безліч зручних функцій для роботи з JSON, а також забезпечує вбудовану аутентифікацію та авторизацію для API.

Однією з ключових переваг Django є масштабованість. Завдяки своїй структурі та розширюваності, Django дозволяє легко адаптуватися до великих проектів з високими вимогами до продуктивності. Для оптимізації роботи додатків в Django є можливості кешування, підтримка різних баз даних, розподілені системи та асинхронне оброблення запитів.

Також важливою частиною Django є безпека. Фреймворк надає вбудовані механізми для захисту від найпоширеніших загроз безпеці, таких як SQL-ін'єкції, CSRF (Cross-Site Request Forgery), XSS (Cross-Site Scripting) та інші. Це досягається завдяки механізмам валідації, автогенерації безпечних SQL-запитів, захисту сесій та збереження паролів в зашифрованому вигляді.

Для тестування в Django передбачені потужні інструменти, зокрема підтримка юніт-тестів за допомогою фреймворків, таких як `unittest` або `pytest`. Django дозволяє легко писати автоматизовані тести для перевірки роботи додатка, що забезпечує стабільність проекту та полегшує процес його розвитку.

Завдяки своїй величезній документації, великій спільноті розробників і безлічі готових пакетів і розширень, Django є одним з найпопулярніших фреймворків для розробки веб-додатків на Python. Його використовують для створення як простих веб-сайтів, так і великих масштабованих додатків, таких як соціальні мережі, платформи для обміну контентом і електронної комерції.

Вибір Python, Django для розробки системи можна обґрунтувати декількома аспектами, що забезпечують ефективність, масштабованість і зручність в процесі розробки.

Python є однією з найпопулярніших мов програмування завдяки своїй простоті та зручності. Вона має зрозумілий синтаксис, що дозволяє швидко розпочати розробку та підтримку проектів. Це важливо для систем, де необхідно швидко реагувати на зміни вимог чи реалізовувати нові функціональності. Python має великий набір бібліотек та фреймворків, що дозволяє реалізувати різноманітні задачі, починаючи від веб-розробки і закінчуючи машинним навчанням або обробкою даних. Вибір Python гарантує гнучкість і можливість швидко масштабувати систему, а також її інтеграцію з іншими технологіями та інструментами.

Django є потужним фреймворком для веб-розробки, побудованим на Python, який дозволяє швидко створювати безпечні, масштабовані веб-додатки. Однією з головних переваг Django є його "підхід із коробки", що включає в себе все необхідне для розробки — від модулів для роботи з базою даних до системи аутентифікації, авторизації та адміністративної панелі. Це дозволяє зосередитися на реалізації бізнес-логіки і функціоналу, а не на написанні базових компонентів. Django забезпечує високу продуктивність, що робить його ідеальним вибором для проектів з великими обсягами даних або високими вимогами до навантаження. Його безпека є важливим фактором для створення систем, де необхідно захищати дані користувачів і запобігати атакам, таким як SQL-ін'єкції або CSRF.

**Висновки до розділу**

Поєднання Python i Django для розробки системи забезпечує надійність, високу продуктивність і зручність в роботі. Це рішення дозволяє ефективно вирішувати широке коло завдань, від простих веб-додатків до складних і масштабованих систем, гарантуючи при цьому високу безпеку, підтримку великих обсягів даних і можливість легкої інтеграції з іншими технологіями.

# РОЗДІЛ 3. ПРОЕКТНО-РЕКОМЕНДАЦІЙНА ЧАСТИНА

## 3.1 Розробка проектного рішення

Під час проектування додатку було розроблено наступні діаграми та схеми:


Рисунок 3.1 - Діаграма використання


Рисунок 3.2 - Діаграма класів



Рисунок 3.3 - Діаграма класів

#!/usr/bin/env python

import os

import sys

def main():

 os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'config.settings')

 try:

 from django.core.management import execute\_from\_command\_line

 except ImportError:

 try:

 import django

 except ImportError:

 raise ImportError(

 "Couldn't import Django. Are you sure it's installed and "

 "available on your PYTHONPATH environment variable?"

 )

 raise

 execute\_from\_command\_line(sys.argv)

if \_\_name\_\_ == '\_\_main\_\_':

 main()

Лістинг 3.1 - Основний скрипт запуску Django проекту

Цей фрагмент Python коду є частиною головного скрипту для запуску Django-проекту. Перш за все, він налаштовує середовище для Django, встановлюючи параметри конфігурації через змінну середовища DJANGO\_SETTINGS\_MODULE, вказуючи на файл налаштувань проекту. Після цього код намагається імпортувати необхідну функцію для виконання команд Django через execute\_from\_command\_line, що дозволяє запускати сервер або виконувати інші адміністративні команди Django, передаючи аргументи командного рядка через sys.argv.

У випадку, якщо модуль django.core.management не знайдено, код спробує імпортувати сам Django. Якщо ж і це не вдасться, піднімається виключення з повідомленням про проблему з інсталяцією Django, яке вказує на те, що необхідний пакет не встановлений або не доступний у середовищі Python. Якщо Django доступний, він ініціалізує його і виконує передані команди.

Що стосується залежностей, зазначених у списку, вони включають необхідні бібліотеки для роботи з Django та його компонентами. Відзначені версії гарантують сумісність проекту з конкретними випусками Django (версії від 4.2 до 5.0) та іншими важливими бібліотеками, такими як psycopg2 для взаємодії з PostgreSQL, djangorestframework для створення API, django-allauth для управління автентифікацією користувачів, django-crispy-forms для покращеного рендерингу форм та python-dotenv для зручного налаштування середовища за допомогою файлів .env. Ці залежності допомагають забезпечити надійність та зручність при розробці Django-проекту.

DEBUG=True

SECRET\_KEY=your-secret-key-here

DATABASE\_URL=postgres://username:password@localhost:5432/mydatabase

ALLOWED\_HOSTS=127.0.0.1, .localhost

EMAIL\_HOST=smtp.gmail.com

EMAIL\_PORT=587

EMAIL\_HOST\_USER=your-email@gmail.com

EMAIL\_HOST\_PASSWORD=your-email-password

Лістинг 3.2 - Файл налаштувань для середовища розробки

Цей фрагмент конфігураційного файлу .env використовується для налаштування середовища Django-проекту. У ньому містяться важливі змінні, які допомагають налаштувати проект на локальному сервері або на продакшн-середовищі.

Перша змінна, DEBUG, вказує на те, чи повинна працювати режим налагодження Django. Якщо значення встановлене на True, Django надає докладні повідомлення про помилки, що корисно при розробці, але небезпечно в продакшн-середовищі, оскільки це може розкрити конфіденційну інформацію.

SECRET\_KEY містить ключ для забезпечення безпеки веб-додатку, який використовується для шифрування даних сесій та інших механізмів безпеки. Цей ключ потрібно тримати в секреті і не розголошувати.

DATABASE\_URL визначає адресу та налаштування для бази даних PostgreSQL. Тут вказано ім'я користувача, пароль, адресу сервера та порт, а також ім'я бази даних, з якою Django має працювати.

ALLOWED\_HOSTS містить список дозволених хостів, які можуть звертатися до веб-додатку. У цьому випадку це 127.0.0.1 (локальний хост) і всі піддомени localhost.

Налаштування для відправлення електронних листів задаються через змінні EMAIL\_HOST, EMAIL\_PORT, EMAIL\_HOST\_USER та EMAIL\_HOST\_PASSWORD. Вони вказують на використання сервера Gmail для відправки листів. Порт 587 вказує на використання протоколу SMTP для відправлення електронних листів з аутентифікацією.

Цей файл .env зазвичай не додається до системи контролю версій (наприклад, через .gitignore), щоб уникнути витоку конфіденційної інформації, як от секретних ключів чи паролів.

import os

import environ

# Читання середовищних змінних з .env файлу

env = environ.Env()

environ.Env.read\_env()

# Безпека

SECRET\_KEY = env('SECRET\_KEY')

DEBUG = env.bool('DEBUG', default=False)

ALLOWED\_HOSTS = env.list('ALLOWED\_HOSTS', default=['127.0.0.1'])

# Додаткові налаштування

INSTALLED\_APPS = [

 'django.contrib.admin',

 'django.contrib.auth',

 'django.contrib.contenttypes',

 'django.contrib.sessions',

 'django.contrib.messages',

 'django.contrib.staticfiles',

 'users', # Ваш додаток для користувачів

 'crispy\_forms',

 'rest\_framework',

 'django.contrib.sites', # Для django-allauth

 'allauth',

 'allauth.account',

 'allauth.socialaccount',

]

MIDDLEWARE = [

 'django.middleware.security.SecurityMiddleware',

 'django.contrib.sessions.middleware.SessionMiddleware',

 'django.middleware.common.CommonMiddleware',

 'django.middleware.csrf.CsrfViewMiddleware',

 'django.contrib.auth.middleware.AuthenticationMiddleware',

 'django.contrib.messages.middleware.MessageMiddleware',

 'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'config.urls'

TEMPLATES = [

 {

 'BACKEND': 'django.template.backends.django.DjangoTemplates',

 'DIRS': [os.path.join(BASE\_DIR, 'templates')],

 'APP\_DIRS': True,

 'OPTIONS': {

 'context\_processors': [

 'django.template.context\_processors.debug',

 'django.template.context\_processors.request',

 'django.contrib.auth.context\_processors.auth',

 'django.contrib.messages.context\_processors.messages',

 ],

 },

 },

]

# Налаштування бази даних

DATABASES = {

 'default': env.db('DATABASE\_URL', default='postgres://username:password@localhost:5432/mydatabase')

}

# Налаштування для статичних файлів

STATIC\_URL = 'static/'

STATICFILES\_DIRS = [os.path.join(BASE\_DIR, 'static')]

# Налаштування для медіа файлів

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

# Налаштування авторизації

AUTH\_USER\_MODEL = 'users.CustomUser'

# Налаштування для пошти

EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL\_HOST = env('EMAIL\_HOST')

EMAIL\_PORT = env.int('EMAIL\_PORT', default=587)

EMAIL\_USE\_TLS = True

EMAIL\_HOST\_USER = env('EMAIL\_HOST\_USER')

EMAIL\_HOST\_PASSWORD = env('EMAIL\_HOST\_PASSWORD')

# Секрети для аутентифікації

ACCOUNT\_EMAIL\_REQUIRED = True

ACCOUNT\_EMAIL\_VERIFICATION = 'mandatory'

ACCOUNT\_AUTHENTICATED\_REDIRECT\_URL = '/'

Лістинг 3.3 - Конфігурація налаштувань Django за допомогою середовищних змінних

Цей фрагмент Python-коду є конфігурацією для Django-проекту, в якому використовуються середовищні змінні для налаштування параметрів безпеки, бази даних, електронної пошти та інших аспектів роботи веб-додатку. Замість того, щоб жорстко прописувати конфіденційну інформацію безпосередньо в коді, тут застосовується бібліотека environ для зручного завантаження змінних з .env файлу.

У першій частині коду ініціалізується об'єкт env, який відповідає за зчитування змінних середовища з .env файлу за допомогою методу read\_env(). Після цього всі ключові параметри конфігурації, такі як SECRET\_KEY, DEBUG, і ALLOWED\_HOSTS, зчитуються через методи env(), env.bool(), та env.list() відповідно, що дає змогу зберігати налаштування безпосередньо в середовищних змінних. Це дозволяє уникнути попадання чутливої інформації у вихідний код.

Конфігурація для додатків Django задається через список INSTALLED\_APPS. Тут визначені стандартні додатки Django, такі як адміністративна панель (django.contrib.admin), а також встановлені додатки для користувачів (users), форми (crispy\_forms), REST API (rest\_framework), і авторизація через соціальні мережі (allauth).

Механізм безпеки та middleware в Django налаштовується через список MIDDLEWARE, який містить стандартні мідлвари для захисту, сесій та аутентифікації.

Налаштування шаблонів (templates) визначають, що Django має шукати HTML-шаблони в директорії templates в кореневій директорії проекту та підтримує інтерпретацію шаблонів за допомогою контексту з різних джерел.

Налаштування бази даних використовується з параметра DATABASES, де через змінну середовища зчитується URL для підключення до PostgreSQL, що дозволяє вказувати параметри підключення до бази даних в зовнішньому конфігураційному файлі.

Налаштування для статичних та медіа файлів визначаються через STATIC\_URL, STATICFILES\_DIRS для статичних ресурсів, та MEDIA\_URL, MEDIA\_ROOT для медіафайлів, що дозволяє Django коректно працювати з файлами на сервері.

Налаштування поштового сервера дозволяє проекту надсилати електронні листи через SMTP-сервер, зчитуючи відповідні параметри, такі як EMAIL\_HOST, EMAIL\_PORT, EMAIL\_HOST\_USER, і EMAIL\_HOST\_PASSWORD, з середовищних змінних для забезпечення безпеки.

Налаштування для аутентифікації через django-allauth включають обов'язковість підтвердження електронної пошти для нових користувачів та автоматичний редирект на головну сторінку після авторизації.

Всі ці налаштування дозволяють зручно й безпечно налаштувати Django-проект, зберігаючи чутливу інформацію поза кодом і забезпечуючи гнучкість при розгортанні проекту в різних середовищах.

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

 path('admin/', admin.site.urls),

 path('accounts/', include('allauth.urls')), # Для аутентифікації через allauth

 path('', include('users.urls')), # Ваша аплікація для користувачів

Лістинг 3.4 - Конфігурація URL-шляхів для Django з allauth і користувацькими маршрутами

Цей фрагмент коду є частиною основного файлу маршрутів Django-проекту, що налаштовує доступ до різних частин веб-додатку через URL-шляхи.

Перший маршрут вказує на стандартну адміністративну панель Django, яка доступна за адресою /admin/. Це дозволяє адміністраторам мати доступ до інтерфейсу для керування даними користувачів, моделями та іншими аспектами проекту.

Другий маршрут відповідає за інтеграцію з бібліотекою django-allauth, яка використовується для аутентифікації через електронну пошту, соціальні мережі та інші методи. Включення цього маршруту через path('accounts/', include('allauth.urls')) надає доступ до всіх URL-адрес, що визначені в allauth, таких як реєстрація, вхід, підтвердження електронної пошти тощо.

Останній маршрут визначає, що для головної сторінки (шлях '/') буде використовуватися маршрутизація, визначена в додатку користувачів (наприклад, в файлі users/urls.py). Це може бути будь-яка логіка, пов'язана з профілем користувача, налаштуваннями або іншими сторінками додатку для користувачів.

Всі ці маршрути дозволяють зібрати разом основні елементи проекту, такі як аутентифікація, адміністративне управління та функціонал для користувачів, на основі Django.

import os

from django.db import connections

from django.core.exceptions import ImproperlyConfigured

def configure\_database():

 """Конфігурація бази даних PostgreSQL через середовищні змінні"""

 try:

 database\_url = os.getenv('DATABASE\_URL')

 if not database\_url:

 raise ImproperlyConfigured("DATABASE\_URL не знайдено в середовищних змінних")

 connection = connections['default']

 connection.settings\_dict['NAME'] = database\_url.split(':')[-1]

 # Можна додати додаткову обробку для деталей з'єднання, таких як ім'я користувача та пароль

 except ImproperlyConfigured as e:

 print(f"Error in database configuration: {e}")

Лістинг 3.5 - Конфігурація з'єднання з базою даних через середовищні змінні

Цей фрагмент коду відповідає за налаштування з'єднання з базою даних PostgreSQL за допомогою середовищних змінних у Django.

Функція configure\_database() виконує наступні кроки:

Зчитує значення змінної середовища DATABASE\_URL, яка містить URL-адресу з’єднання з базою даних. Якщо ця змінна не знайдена або порожня, викликається виключення ImproperlyConfigured, що вказує на помилку у налаштуванні бази даних.

Перевіряє, чи є в змінних середовища правильне значення для DATABASE\_URL. Якщо його немає, викидається виключення з відповідним повідомленням.

Якщо змінна середовища зчитана успішно, налаштовується підключення до бази даних через Django. За замовчуванням використовує з'єднання, яке визначено в Django (connections['default']).

Після цього в словнику налаштувань підключення бази даних (включаючи NAME) змінюється значення на базу даних, що міститься у DATABASE\_URL. Потрібно додати додаткову обробку для витягнення інших параметрів, таких як ім'я користувача та пароль з DATABASE\_URL, залежно від формату цього URL.

Цей код корисний для динамічної конфігурації підключення до бази даних у різних середовищах (наприклад, для локальної та продакшн версії проекту) з використанням змінних середовища, що є безпечним способом керувати конфіденційними даними, такими як паролі.

from django.contrib import admin

from .models import CustomUser, Profile, UserActivity, UserSettings

class UserAdmin(admin.ModelAdmin):

 """Налаштування адміністратора для моделі користувача."""

 list\_display = ("username", "email", "phone\_number", "is\_verified", "date\_of\_birth")

 search\_fields = ("username", "email", "phone\_number")

 list\_filter = ("is\_verified", "is\_staff")

 ordering = ("username",)

class ProfileAdmin(admin.ModelAdmin):

 """Налаштування адміністратора для профілю користувача."""

 list\_display = ("user", "location", "bio", "joined\_date")

 search\_fields = ("user\_\_username", "location")

 list\_filter = ("joined\_date",)

 ordering = ("user\_\_username",)

class UserActivityAdmin(admin.ModelAdmin):

 """Налаштування адміністратора для активності користувача."""

 list\_display = ("user", "last\_login\_time", "last\_activity\_time", "ip\_address")

 search\_fields = ("user\_\_username", "ip\_address")

 list\_filter = ("last\_activity\_time",)

 ordering = ("user\_\_username",)

class UserSettingsAdmin(admin.ModelAdmin):

 """Налаштування адміністратора для налаштувань користувача."""

 list\_display = ("user", "receive\_newsletter", "dark\_mode", "language\_preference")

 search\_fields = ("user\_\_username",)

 list\_filter = ("dark\_mode", "language\_preference")

 ordering = ("user\_\_username",)

# Реєстрація моделей в адмін-панелі

admin.site.register(CustomUser, UserAdmin)

admin.site.register(Profile, ProfileAdmin)

admin.site.register(UserActivity, UserActivityAdmin)

admin.site.register(UserSettings, UserSettingsAdmin)

Лістинг 3.6 - Налаштування адміністратора для моделей користувача та профілю

У цьому фрагменті коду описуються налаштування адміністратора Django для різних моделей, пов'язаних з користувачами та їх профілями. Для кожної моделі створено власний клас адміністратора, який визначає, які поля повинні бути відображені в адмін-панелі, як можна здійснювати пошук та фільтрацію, а також порядок сортування елементів.

Клас UserAdmin налаштовує відображення користувачів в адмін-панелі, включаючи поля для імені користувача, електронної пошти, номера телефону, статусу перевірки та дати народження. Також визначені поля для пошуку (ім'я користувача, електронна пошта, номер телефону) та фільтрації за статусами перевірки та адміністратора. Крім того, користувачі будуть відсортовані за іменем користувача.

Для профілю користувача створений клас ProfileAdmin, де відображаються поля, що стосуються профілю, такі як користувач, місцезнаходження, біографія та дата приєднання. Окрім цього, доступна фільтрація за датою приєднання та пошук по імені користувача і місцезнаходженню.

Клас UserActivityAdmin відповідає за налаштування відображення активності користувача. У ньому зазначені поля для відображення останнього часу входу, останньої активності та IP-адреси. Це дозволяє адміністраторам відстежувати активність користувачів на сайті.

В останньому класі UserSettingsAdmin налаштовуються відображення налаштувань користувача, таких як отримання розсилки, темна тема та вибір мови. Також є можливість фільтрувати за темною темою та мовою, а також шукати за ім'ям користувача.

Кожна з моделей реєструється в адмін-панелі за допомогою admin.site.register, що дозволяє адміністраторам проекту зручно керувати користувачами, їх профілями, активністю та налаштуваннями через стандартний інтерфейс Django.

from django.apps import AppConfig

class UsersConfig(AppConfig):

 """Конфігурація додатка користувачів."""

 name = 'users'

 verbose\_name = "Користувачі"

Лістинг 3.7 - Конфігурація додатка користувачів

У цьому фрагменті коду створюється конфігурація для Django додатка користувачів. Клас UsersConfig є підкласом AppConfig, що відповідає за налаштування імені та мета-даних додатка. Властивість name вказує на ім'я додатка, що буде використовуватися Django для його ідентифікації, в даному випадку це 'users'. Властивість verbose\_name дозволяє задати людську, читабельну назву для додатка, яка буде відображатися в адміністративному інтерфейсі Django, і в цьому випадку це "Користувачі". Це дає змогу краще організувати структуру проекту та полегшує навігацію в адмін-панелі, коли кілька додатків можуть бути присутні в одному проекті.

from django.shortcuts import render, redirect

from django.contrib.auth import login, authenticate

from django.contrib.auth.forms import AuthenticationForm

from .forms import CustomUserCreationForm, ProfileUpdateForm, UserUpdateForm, UserSettingsForm

from .models import Profile, UserSettings

from django.contrib.auth.decorators import login\_required

def register(request):

 """Реєстрація нового користувача."""

 if request.method == "POST":

 form = CustomUserCreationForm(request.POST)

 if form.is\_valid():

 user = form.save()

 login(request, user)

 return redirect("profile")

 else:

 form = CustomUserCreationForm()

 return render(request, "users/register.html", {"form": form})

def user\_login(request):

 """Вхід користувача в систему."""

 if request.method == "POST":

 form = AuthenticationForm(data=request.POST)

 if form.is\_valid():

 user = form.get\_user()

 login(request, user)

 return redirect("profile")

 else:

 form = AuthenticationForm()

 return render(request, "users/login.html", {"form": form})

@login\_required

def profile(request):

 """Профіль користувача."""

 return render(request, "users/profile.html")

@login\_required

def update\_profile(request):

 """Оновлення профілю користувача."""

 if request.method == "POST":

 profile\_form = ProfileUpdateForm(request.POST, request.FILES, instance=request.user.profile)

 user\_form = UserUpdateForm(request.POST, instance=request.user)

 settings\_form = UserSettingsForm(request.POST, instance=request.user.settings)

 if profile\_form.is\_valid() and user\_form.is\_valid() and settings\_form.is\_valid():

 profile\_form.save()

 user\_form.save()

 settings\_form.save()

 return redirect("profile")

 else:

 profile\_form = ProfileUpdateForm(instance=request.user.profile)

 user\_form = UserUpdateForm(instance=request.user)

 settings\_form = UserSettingsForm(instance=request.user.settings)

 return render(request, "users/update\_profile.html", {

 "profile\_form": profile\_form,

 "user\_form": user\_form,

 "settings\_form": settings\_form

 })

Лістинг 3.8 - Керування профілем користувача

Цей фрагмент коду надає функціональність для реєстрації, входу, перегляду та оновлення профілю користувача в Django додатку.

Функція register обробляє реєстрацію нових користувачів. Якщо запит є POST, то створюється форма реєстрації CustomUserCreationForm і перевіряється її валідність. Якщо форма валідна, користувач зберігається, і його авторизують за допомогою login, після чого відбувається перенаправлення на сторінку профілю.

Функція user\_login обробляє вхід користувача. При отриманні POST-запиту створюється форма AuthenticationForm, яка перевіряється на валідність. Якщо форма валідна, користувач аутентифікується і перенаправляється на профіль.

Функція profile відображає сторінку профілю користувача, доступну тільки після входу (завдяки декоратору login\_required).

Функція update\_profile дозволяє користувачеві оновлювати свої дані профілю, такі як аватар, ім'я користувача, налаштування профілю. Якщо форма заповнена коректно, зміни зберігаються і користувач перенаправляється на свій профіль.

Усі ці функції надають важливу частину користувацької взаємодії, дозволяючи працювати з обліковими записами користувачів, їх профілями та налаштуваннями.

from django.urls import path

from . import views

urlpatterns = [

 path("register/", views.register, name="register"),

 path("login/", views.user\_login, name="login"),

 path("profile/", views.profile, name="profile"),

 path("update\_profile/", views.update\_profile, name="update\_profile"),

]

Лістинг 3.9 - URL конфігурація для користувачів

Цей фрагмент коду описує маршрути для обробки запитів, пов'язаних з користувацькими функціями, такими як реєстрація, вхід, перегляд і оновлення профілю.

Маршрут "register/" призначений для обробки запитів на реєстрацію нових користувачів. Він використовує функцію register з відповідного модуля views, яка відповідає за обробку форми реєстрації.

Маршрут "login/" направляє на сторінку входу, де користувач може ввести свої облікові дані для аутентифікації. Для цього використовується функція user\_login.

Маршрут "profile/" надає доступ до профілю користувача після входу в систему. Цей маршрут використовує функцію profile, яка відображає дані профілю поточного користувача.

Маршрут "update\_profile/" дозволяє користувачеві оновити свій профіль, включаючи такі параметри, як ім'я, аватар та налаштування профілю. Для цього використовується функція update\_profile, яка обробляє форму редагування профілю.

Ці маршрути підтримують базові функції для керування обліковими записами користувачів в Django додатку.

from django.db.models.signals import post\_save

from django.dispatch import receiver

from django.core.mail import send\_mail

from django.conf import settings

from .models import CustomUser, UserActivity

@receiver(post\_save, sender=CustomUser)

def send\_welcome\_email(sender, instance, created, \*\*kwargs):

 """Відправлення вітального листа після реєстрації користувача."""

 if created:

 subject = "Вітаємо на нашій платформі!"

 message = f"Привіт {instance.username}, дякуємо за реєстрацію!"

 from\_email = settings.DEFAULT\_FROM\_EMAIL

 send\_mail(subject, message, from\_email, [instance.email])

@receiver(post\_save, sender=CustomUser)

def create\_user\_activity(sender, instance, created, \*\*kwargs):

 """Автоматичне створення запису активності для нового користувача."""

 if created:

 UserActivity.objects.create(user=instance)

@receiver(post\_save, sender=CustomUser)

def update\_user\_activity(sender, instance, \*\*kwargs):

 """Оновлення активності користувача після кожного збереження."""

 try:

 activity = UserActivity.objects.get(user=instance)

 activity.last\_activity\_time = instance.last\_login

 activity.save()

 except UserActivity.DoesNotExist:

 pass

Лістинг 3.10 - Сигнали для користувачів і активності

Цей фрагмент коду описує використання сигналів Django для автоматичної обробки подій, що відбуваються після збереження користувача в базі даних, зокрема після його реєстрації чи оновлення.

Перша функція сигналу, send\_welcome\_email, спрацьовує після того, як новий користувач був створений (якщо це перше збереження користувача). Вона генерує і відправляє вітальний лист на електронну пошту користувача, використовуючи стандартний електронний адрес відправника, визначений у налаштуваннях Django.

Друга функція, create\_user\_activity, створює запис активності для нового користувача. Це забезпечує наявність початкового запису в моделі UserActivity для кожного нового користувача, який з'являється в системі.

Остання функція, update\_user\_activity, оновлює час останньої активності користувача після кожного збереження запису про користувача. Якщо у користувача вже є запис активності, то цей запис оновлюється, відображаючи нову дату і час останнього входу. Якщо запису не існує (це може статися, якщо система не згенерувала його раніше), функція просто пропускає оновлення.

Ці сигнали допомагають автоматизувати важливі операції в системі, що знижує потребу в ручному втручанні та підвищує ефективність обробки даних користувачів.

from django.contrib import admin

from .models import Product, Category

class CategoryAdmin(admin.ModelAdmin):

 list\_display = ('name', 'slug')

 prepopulated\_fields = {'slug': ('name',)}

class ProductAdmin(admin.ModelAdmin):

 list\_display = ('name', 'category', 'price', 'is\_available', 'created\_at')

 list\_filter = ('category', 'is\_available')

 search\_fields = ('name', 'description')

 prepopulated\_fields = {'slug': ('name',)}

admin.site.register(Category, CategoryAdmin)

admin.site.register(Product, ProductAdmin)

Лістинг 3.11 - Адміністративна панель для продуктів і категорій

У цьому фрагменті коду визначено адміністративні налаштування для моделей Product та Category в Django.

Клас CategoryAdmin налаштовує відображення категорій у адмін-панелі. Поле list\_display вказує, які поля моделі будуть показані в таблиці адмін-панелі, в даному випадку це name та slug (текстовий ідентифікатор категорії). Опція prepopulated\_fields автоматично генерує значення для поля slug на основі значення поля name, що дозволяє користувачу не вводити це поле вручну, якщо назва категорії вже задана.

Клас ProductAdmin налаштовує відображення продуктів у адмін-панелі. В list\_display визначено, що будуть відображатися поля name (назва продукту), category (категорія продукту), price (ціна), is\_available (статус наявності) та created\_at (дата створення продукту). За допомогою list\_filter можна фільтрувати продукти за категорією та статусом наявності. Поле search\_fields дозволяє здійснювати пошук за полями name і description (опис продукту). Як і в категорії, для поля slug використано prepopulated\_fields, щоб автоматично генерувати його на основі назви продукту.

Обидва класи реєструють відповідні моделі в адмін-панелі за допомогою функції admin.site.register, що дозволяє адміністратору зручно управляти категоріями та продуктами через графічний інтерфейс Django.

from django.apps import AppConfig

class ProductsConfig(AppConfig):

 default\_auto\_field = 'django.db.models.BigAutoField'

 name = 'products'

Лістинг 3.12 - Конфігурація додатка продуктів

Цей фрагмент коду визначає конфігурацію додатку Products для Django. Клас ProductsConfig є підкласом AppConfig і використовується для налаштування та ініціалізації додатку в Django-проекті.

Атрибут default\_auto\_field встановлює тип автоматично генерованих полів для моделей у цьому додатку. В даному випадку використовується значення 'django.db.models.BigAutoField', яке вказує, що всі автоматично генеровані поля (наприклад, поле первинного ключа) будуть мати тип BigAutoField (ціле число, яке може зберігати більші значення, ніж стандартний AutoField).

Атрибут name вказує на повний шлях до пакету додатку в проекті. В даному випадку, додаток має назву 'products', що відповідає директорії або пакету, де зберігаються моделі, представлення, форми та інші компоненти для роботи з продуктами в цьому додатку.

Цей клас дозволяє Django правильно конфігурувати додаток під час запуску, а також дає змогу налаштувати певні параметри, які використовуються для всіх моделей, визначених в цьому додатку.

from django.db import models

class Category(models.Model):

 name = models.CharField(max\_length=100, unique=True)

 slug = models.SlugField(unique=True)

 created\_at = models.DateTimeField(auto\_now\_add=True)

 def \_\_str\_\_(self):

 return self.name

class Product(models.Model):

 name = models.CharField(max\_length=200)

 slug = models.SlugField(unique=True)

 description = models.TextField(blank=True)

 price = models.DecimalField(max\_digits=10, decimal\_places=2)

 category = models.ForeignKey(Category, related\_name='products', on\_delete=models.CASCADE)

 is\_available = models.BooleanField(default=True)

 created\_at = models.DateTimeField(auto\_now\_add=True)

 updated\_at = models.DateTimeField(auto\_now=True)

 def \_\_str\_\_(self):

 return self.name

Лістинг 3.13 - Моделі категорії та продукту

Цей фрагмент коду визначає дві моделі для Django додатку — Category та Product, які будуть використовуватися для збереження інформації про категорії та продукти в базі даних.

Модель Category включає:

Поле name типу CharField з максимальним розміром 100 символів, яке є унікальним (для того, щоб кожна категорія мала свою неповторну назву).

Поле slug типу SlugField, також унікальне, що зазвичай використовується для створення короткої URL-версії назви категорії, наприклад, для SEO.

Поле created\_at типу DateTimeField, яке автоматично заповнюється поточною датою та часом, коли об'єкт категорії був створений.

Модель Product включає:

Поле name типу CharField, яке зберігає назву продукту, з максимальним розміром 200 символів.

Поле slug типу SlugField, яке є унікальним і використовується для створення URL для кожного продукту.

Поле description типу TextField, яке дозволяє додавати опис продукту. Воно може бути порожнім (через параметр blank=True).

Поле price типу DecimalField, яке зберігає ціну продукту з максимальною кількістю цифр — 10, з двома десятковими місцями.

Поле category типу ForeignKey, яке встановлює зв'язок між продуктом та його категорією. Це означає, що кожен продукт належить до певної категорії. Поле має параметр related\_name='products', що дозволяє отримати всі продукти, що належать до певної категорії.

Поле is\_available типу BooleanField, яке вказує, чи доступний продукт для покупки.

Поля created\_at та updated\_at типу DateTimeField, де перше автоматично заповнюється датою та часом створення продукту, а друге оновлюється щоразу при збереженні продукту.

Метод \_\_str\_\_ для обох моделей визначає, як об'єкти цих моделей будуть відображатися у адміністративній панелі Django чи інших контекстах, де потрібно відобразити строкову репрезентацію об'єкта. Для категорії це її назва, а для продукту — його назва.

from django.shortcuts import render, get\_object\_or\_404

from .models import PageView, ProductRating, Report

from django.db.models import Avg

from .forms import ReportForm

def product\_statistics(request, product\_id):

 views = PageView.objects.filter(product\_id=product\_id).count()

 avg\_rating = ProductRating.objects.filter(product\_id=product\_id).aggregate(Avg('rating'))

 context = {

 'views': views,

 'average\_rating': avg\_rating['rating\_\_avg'],

 }

 return render(request, 'analytics/product\_statistics.html', context)

def report\_list(request):

 reports = Report.objects.all().order\_by('-created\_at')

 return render(request, 'analytics/report\_list.html', {'reports': reports})

def create\_report(request):

 if request.method == 'POST':

 form = ReportForm(request.POST)

 if form.is\_valid():

 form.save()

 return redirect('analytics:report\_list')

 else:

 form = ReportForm()

 return render(request, 'analytics/create\_report.html', {'form': form})

Лістинг 3.14 - Види статистики продукту та звітів

У цьому фрагменті коду ми розробляємо кілька представлень для обробки статистики продуктів та звітів у нашій аналітичній частині застосунку.

Перше представлення, product\_statistics, відповідає за надання статистики для конкретного продукту. Воно приймає ідентифікатор продукту, виконує два запити: один для підрахунку кількості переглядів продукту (за допомогою PageView.objects.filter(product\_id=product\_id).count()), а другий для обчислення середнього рейтингу продукту (через агрегацію з Avg('rating')). Ця статистика передається у шаблон, щоб відобразити її користувачам. Для середнього рейтингу результат агрегації зберігається в змінній avg\_rating, і ми передаємо його значення в контекст за допомогою avg\_rating['rating\_\_avg'].

Другий метод, report\_list, обробляє запит на відображення списку всіх звітів. Він отримує всі звіти з бази даних через Report.objects.all().order\_by('-created\_at'), що гарантує сортування звітів за датою створення, починаючи з найновіших. Звіти передаються в контекст для відображення на відповідній сторінці.

Третій метод, create\_report, дозволяє створювати нові звіти. Якщо запит є POST, ми перевіряємо, чи є форма валідною за допомогою form.is\_valid(), і після цього зберігаємо новий звіт у базі даних. Якщо форма була успішно збережена, користувач перенаправляється на сторінку зі списком звітів. Якщо запит є GET (тобто сторінка ще не була відправлена), ми просто передаємо порожню форму для заповнення.

Ці функції забезпечують базовий функціонал для перегляду статистики, роботи зі звітами та їхнього створення, інтегруючи цю інформацію у веб-застосунок через відповідні HTML-шаблони.

from django.urls import path

from . import views

app\_name = 'analytics'

urlpatterns = [

 path('product/<int:product\_id>/statistics/', views.product\_statistics, name='product\_statistics'),

 path('reports/', views.report\_list, name='report\_list'),

 path('reports/create/', views.create\_report, name='create\_report'),

]

Лістинг 3.15 - Шлях до статистики продукту та звітів.

В цьому фрагменті коду ми визначаємо маршрути для аналітичної частини застосунку, використовуючи систему URL-шляхів Django. Кожен шлях відповідає певному представленню в нашому додатку.

Перший шлях, product\_statistics, обробляє запит для перегляду статистики конкретного продукту. Він містить динамічний параметр <int:product\_id>, що означає, що Django передасть ідентифікатор продукту як змінну product\_id у функцію представлення product\_statistics. Цей шлях дозволяє користувачам отримувати статистику для конкретного продукту, включаючи кількість переглядів і середній рейтинг.

Другий шлях, report\_list, обробляє запит на відображення списку всіх звітів. Він не має додаткових параметрів і веде до представлення report\_list, де користувачі можуть побачити всі наявні звіти, сортування яких визначене за датою створення.

Третій шлях, create\_report, відповідає за створення нових звітів. Він веде до представлення create\_report, де користувачі можуть заповнити форму для створення звіту. Цей шлях передбачає використання методу POST для відправки даних форми.

Всі ці маршрути зібрані під простором імен analytics, що дозволяє уникнути потенційних конфліктів з іншими маршрутами в додатку та забезпечує більш організовану структуру URL-ів.

<!DOCTYPE html>

<html lang="uk">

<head>

 <meta charset="UTF-8">

 <meta name="viewport" content="width=device-width, initial-scale=1.0">

 <title>{% block title %}Мій Проект{% endblock %}</title>

 <link rel="stylesheet" href="{% static 'css/main.css' %}">

 {% block extra\_head %}{% endblock %}

</head>

<body>

 <header>

 <nav>

 <ul>

 <li><a href="/">Головна</a></li>

 <li><a href="/products/">Продукти</a></li>

 <li><a href="/accounts/login/">Увійти</a></li>

 <li><a href="/accounts/signup/">Реєстрація</a></li>

 </ul>

 </nav>

 </header>

 <main>

 {% block content %}{% endblock %}

 </main>

 <footer>

 <p>&copy; 2024 Мій Проект</p>

 </footer>

 <script src="{% static 'js/main.js' %}"></script>

</body>

</html>

{% extends 'base.html' %}

{% block title %}Вхід{% endblock %}

{% block content %}

 <h2>Вхід до акаунту</h2>

 <form method="post">

 {% csrf\_token %}

 {{ form.as\_p }}

 <button type="submit">Увійти</button>

 </form>

{% endblock %}

Лістинг 3.16 - Шаблон для сторінки входу.

Цей фрагмент коду описує дві частини веб-сторінки, реалізовані в Django за допомогою шаблонів. Перша частина — це базовий шаблон HTML, а друга — розширення цього шаблону для створення форми входу до акаунту.

У першому шаблоні, який є базовим (base.html), створюється структура сторінки, що включає:

Заголовок сторінки, який дозволяє змінювати назву для кожної сторінки через блок title.

Підключення стилів і JavaScript файлів через тег {% static %}, щоб забезпечити правильне завантаження файлів.

Навігаційне меню з посиланнями на головну сторінку, продукти, сторінки для входу та реєстрації.

Блок для основного контенту, що заповнюється відповідними частинами в дочірніх шаблонах через блок content.

Підключення футера, де зазначено авторське право.

У другій частині, яка є шаблоном для сторінки входу (login.html), використовується конструкція {% extends 'base.html' %}, що дозволяє цей шаблон розширювати основний шаблон і заповнювати конкретні блоки:

Блок title змінює заголовок сторінки на "Вхід".

У блоці content виводиться форма для входу, що містить CSRF токен для захисту від атак типу CSRF і форму, яку передає Django у вигляді полів через {{ form.as\_p }}. Кнопка "Увійти" дозволяє надіслати форму.

Цей підхід дозволяє легко структурувати веб-додаток, розділяючи загальні елементи сторінки (як меню та футер) від змінюваного контенту, що відповідає за конкретну функціональність сторінки, як у випадку з формою входу.

{% extends 'base.html' %}

{% block title %}Реєстрація{% endblock %}

{% block content %}

 <h2>Створити акаунт</h2>

 <form method="post">

 {% csrf\_token %}

 {{ form.as\_p }}

 <button type="submit">Зареєструватися</button>

 </form>

{% endblock %}

Лістинг 3.16 - Шаблон для сторінки реєстрації.

Цей фрагмент коду описує шаблон для сторінки реєстрації користувача, який розширює базовий шаблон base.html. Ось деталі:

Шаблон використовує конструкцію {% extends 'base.html' %}, що дозволяє йому успадковувати структуру базової сторінки. Таким чином, він використовує всі спільні елементи з іншими сторінками, такі як меню навігації і футер.

У блоці {% block title %} задається назва сторінки, яка на цьому шаблоні буде "Реєстрація". Це дозволяє змінювати заголовок сторінки залежно від її контексту.

Основний контент сторінки розміщується у блоці {% block content %}. У цьому блоці:

Заголовок <h2> з текстом "Створити акаунт" пояснює користувачеві мету цієї сторінки.

Форми обробляються через метод POST, і для безпеки додається {% csrf\_token %}, що дозволяє захистити сайт від атак CSRF.

Форма для введення даних користувача виводиться за допомогою {{ form.as\_p }}, що автоматично перетворює форму в HTML, обгорнуту в параграфи.

Кнопка "Зареєструватися" дозволяє надіслати форму.

Цей шаблон дає змогу користувачеві заповнити форму реєстрації, а сервер зможе обробити введені дані для створення нового акаунта.

{% extends 'base.html' %}

{% block title %}Продукти{% endblock %}

{% block content %}

 <h2>Список продуктів</h2>

 <ul>

 {% for product in products %}

 <li>

 <a href="{% url 'product\_detail' product.id %}">{{ product.name }}</a>

 </li>

 {% empty %}

 <li>Продукти не знайдено.</li>

 {% endfor %}

 </ul>

{% endblock %}

Лістинг 3.17 - Шаблон для відображення списку продуктів.

Цей шаблон використовується для відображення списку продуктів на сторінці. Ось деталі:

Наслідування базового шаблону:

Шаблон розширює base.html, що дозволяє використовувати загальні елементи сторінки, такі як навігаційне меню і футер.

Заголовок сторінки:

У блоці {% block title %} вказано заголовок сторінки, що буде відображатися на вкладці браузера — "Продукти".

Основний вміст сторінки:

Блок {% block content %} містить основний контент сторінки. У ньому:

Заголовок <h2> "Список продуктів" інформує користувача про зміст сторінки.

Використовується цикл {% for product in products %}, щоб відобразити список усіх продуктів. Для кожного продукту створюється елемент списку <li>, в якому є посилання на сторінку деталів продукту. Посилання використовує тег {% url 'product\_detail' product.id %}, що генерує URL на основі ідентифікатора продукту.

Якщо продукти не знайдено, використовується тег {% empty %} для виведення повідомлення "Продукти не знайдено."

Секція порожнього списку:

Якщо список продуктів порожній, замість циклу виводиться елемент списку <li>, що повідомляє користувача про відсутність продуктів.

Цей шаблон дозволяє динамічно відображати продукти на веб-сторінці, а також повідомляти користувача про відсутність продуктів, якщо їх немає в базі даних.

body {

 font-family: Arial, sans-serif;

 margin: 0;

 padding: 0;

 background-color: #f4f4f4;

}

header {

 background-color: #333;

 color: white;

 padding: 10px 0;

}

header nav ul {

 list-style-type: none;

 margin: 0;

 padding: 0;

 display: flex;

 justify-content: space-around;

}

header nav ul li {

 display: inline;

}

header nav ul li a {

 color: white;

 text-decoration: none;

 padding: 8px 16px;

}

footer {

 text-align: center;

 padding: 10px;

 background-color: #333;

 color: white;

 position: absolute;

 width: 100%;

 bottom: 0;

}

main {

 padding: 20px;

}

Лістинг 3.18 - CSS для основного стилю сайту.

Цей CSS код забезпечує стилізацію основних елементів веб-сторінки, таких як тіло, заголовок, навігаційне меню, футер та основний вміст:

Основне оформлення (body):

Встановлено шрифт "Arial" для всього тексту на сторінці, з використанням безсерифного шрифту (sans-serif) як резервного.

Відсутні зовнішні та внутрішні відступи для всього тіла сторінки.

Встановлено світло-сірий фон для всієї сторінки.

Заголовок (header):

Фон заголовка має темно-сірий колір (#333), а текст — білий.

Встановлені внутрішні відступи по вертикалі (10px).

Навігаційне меню в заголовку (як список) відображається горизонтально з рівномірним розподілом елементів завдяки flex контейнеру та властивості justify-content: space-around.

Навігаційне меню (header nav ul):

У навігаційному меню немає маркерів або відступів для списку.

Кожен елемент списку відображається як інлайн-елемент.

Посилання в меню мають білий колір, без підкреслення та з додатковими відступами для кращого вигляду.

Футер (footer):

Футер має такий самий темно-сірий фон і білий текст, як і заголовок.

Текст у футері вирівняний по центру.

Встановлено абсолютне позиціонування футера внизу сторінки, що забезпечує його фіксоване положення при прокручуванні контенту.

Основний вміст (main):

В основному блоці контенту є внутрішні відступи (20px), щоб текст не прилипав до країв екрану.

Ці стилі задають простий і чистий дизайн для базової сторінки, з чітко структурованим та легким для сприйняття інтерфейсом.

## 3.2 Розробка рекомендацій

Подальший розвиток системи є критично важливим для підтримки її актуальності, конкурентоспроможності та задоволення потреб користувачів у довгостроковій перспективі. Сучасний ринок технологій динамічно змінюється, а очікування користувачів постійно зростають, що робить оновлення системи ключовим фактором для її успішного функціонування. Без постійного розвитку система ризикує втратити конкурентні переваги та відставати від нових технологічних стандартів і трендів, що з часом може призвести до зниження рівня зацікавленості користувачів і зменшення їхньої активності. Інвестиції в нові функції, технології та оптимізацію роботи платформи забезпечують її стійкість та релевантність у швидкоплинному цифровому світі.

Розширення аналітики в системі передбачає створення глибшої та багатограннішої платформи для аналізу поведінки користувачів та оцінки ключових показників ефективності. Основна мета розширеної аналітики — надання детальної інформації про активність користувачів, яка дозволить адміністраторам системи та бізнес-користувачам отримати точнішу картину використання платформи, а також розуміти, які продукти чи функції користувачі віддають перевагу. Розширена аналітика має базуватись на збиранні та обробці значного обсягу даних про взаємодії користувачів із системою.

Один з ключових напрямків розширеної аналітики — відстеження відвідуваності сторінок продуктів, що включає детальні дані про кількість переглядів кожного продукту, частоту повернень на сторінку, середній час, який користувачі проводять на сторінці продукту, та джерела трафіку. Ця інформація дозволить власникам платформи визначати популярні та непопулярні продукти, а також виявляти потенційні можливості для покращення. Наприклад, якщо на сторінці популярного продукту користувачі проводять значний час, але не здійснюють наступних кроків, це може вказувати на потребу в оптимізації опису продукту чи покращенні процесу навігації.

Ще одним важливим аспектом є впровадження інструментів візуалізації даних, таких як інтерактивні графіки, діаграми та звіти, які дозволять адміністраторам легко інтерпретувати складну інформацію. Використання різних типів графічних елементів дозволить наочно побачити динаміку змін показників у різні періоди часу, зокрема щодо зростання чи зниження кількості користувачів, сезонних коливань та змін у популярності окремих продуктів. Аналітичний дашборд може бути інтегрований у систему таким чином, щоб надавати оновлену інформацію в режимі реального часу, що стане незамінним інструментом для швидкого ухвалення рішень.

Також важливим є додавання можливості сегментації користувачів за різними критеріями, такими як вік, локація, тип пристрою, з якого заходять на платформу, або активність за період. Це допоможе визначити конкретні групи користувачів, які виявляють особливий інтерес до певних продуктів, або навпаки — ігнорують окремі розділи платформи. Таким чином, на основі цих даних можна створювати персоналізовані пропозиції, спрямовані на підвищення зацікавленості конкретних сегментів аудиторії. Розширення аналітики також дозволить краще налаштовувати маркетингові кампанії та таргетувати їх на певні групи користувачів, що підвищить їх ефективність і зменшить витрати.

Розширення аналітичних можливостей відкриває можливість для інтеграції інструментів передбачувальної аналітики. Використовуючи машинне навчання, можна буде прогнозувати попит на окремі продукти та аналізувати фактори, що впливають на активність користувачів. Це дозволить створювати моделі, які прогнозуватимуть поведінку користувачів та допомагатимуть оптимізувати процеси продажів та взаємодії з клієнтами. Такий підхід допоможе не тільки збільшити комерційні показники, але й розвивати систему з урахуванням майбутніх потреб користувачів.

Покращення користувацького досвіду є ключовим напрямком розвитку системи, що передбачає створення зручного, інтуїтивно зрозумілого та приємного середовища для користувачів. Це передбачає аналіз усіх аспектів взаємодії користувача з платформою — від швидкості завантаження сторінок і легкості навігації до візуального оформлення та зручності у доступі до основних функцій. Зокрема, покращення користувацького досвіду починається з оптимізації дизайну, який має бути не лише естетично привабливим, але й практичним. Добре структуровані розділи, логічне розміщення кнопок і доступність важливих елементів навігації сприяють зниженню рівня розчарування користувачів та зменшують кількість помилок під час використання платформи.

Швидкість завантаження є також важливим аспектом користувацького досвіду. Навіть декілька додаткових секунд на завантаження сторінки можуть негативно вплинути на утримання користувачів. Тому слід звернути увагу на оптимізацію завантаження ресурсів, таких як зображення, скрипти та стилі. Використання сучасних технологій, як-от кешування даних, lazy-loading зображень та оптимізація графіки, дозволить значно пришвидшити роботу системи, особливо для мобільних пристроїв, де обмежена швидкість інтернету може бути фактором затримки.

Інтерактивні та персоналізовані елементи допомагають зробити платформу приємнішою для користувача. Зокрема, впровадження адаптивних елементів, які налаштовуються відповідно до інтересів та дій користувача, може підвищити залучення. Наприклад, персоналізовані рекомендації продуктів, сповіщення про оновлення чи нагадування про незавершені покупки створюють відчуття турботи про користувача та покращують досвід використання системи. Також зручною є функція збереження останніх переглядів, що дозволяє користувачам швидко повернутися до того, що їх цікавило раніше.

Окрім того, для створення повноцінного користувацького досвіду важливим є впровадження ефективної системи підтримки. Це може включати інтеграцію функціоналу онлайн-чату, де користувачі можуть швидко отримати відповіді на свої питання, або розділ FAQ із найпоширенішими запитаннями та відповідями. Залучення системи збору відгуків та оцінок допомагає розробникам краще розуміти потреби користувачів та швидко реагувати на їхні побажання. Зворотний зв'язок може бути використаний для подальшого вдосконалення платформи, що, у свою чергу, зміцнює довіру до неї.

Додатково важливо забезпечити зручність роботи з системою на різних пристроях та екранах, що включає адаптивний дизайн, який автоматично підлаштовується під різні розміри екранів та орієнтацію. Це забезпечує, щоб усі елементи інтерфейсу були доступними і добре виглядали як на настільних комп’ютерах, так і на мобільних пристроях. Такий підхід значно підвищує доступність системи та робить її зручною для користувачів, незалежно від пристрою, який вони використовують, що є важливим фактором для сучасної платформи.

Інтеграція e-commerce функцій є важливим кроком для розширення можливостей системи та надання користувачам додаткових переваг, що може значно збільшити цінність платформи для бізнесу та споживачів. Додавання e-commerce модулів дозволяє створити зручні умови для продажу товарів або послуг безпосередньо на платформі, забезпечуючи простий та інтуїтивний процес покупки для користувачів. Це може включати такі функції, як онлайн-каталог товарів, кошик для покупок, безпечну оплату, управління замовленнями та особистий кабінет користувача. Такі можливості не лише покращують досвід користувача, а й можуть сприяти зростанню прибутків та залученню нових клієнтів.

Інтеграція e-commerce функцій на платформі надає можливість продавати не лише фізичні, але й цифрові товари та послуги, що значно розширює сферу діяльності бізнесу. Наприклад, компанії можуть пропонувати віртуальні консультації, онлайн-курси, програмне забезпечення або інші цифрові продукти, які користувачі можуть завантажити після покупки. Це створює нові можливості для монетизації контенту, що є особливо актуальним для платформ, орієнтованих на знання, розваги або освітні проєкти. Завдяки цьому інтеграція e-commerce функцій дозволяє створювати більш гнучкі бізнес-моделі, які адаптуються до змін потреб користувачів та ринку.

Також важливо зазначити, що e-commerce функції, такі як управління інвентарем, автоматичні сповіщення про залишки товару або відстеження доставок, значно полегшують роботу з логістикою та обліком. Автоматизація цих процесів знижує витрати на обслуговування та мінімізує ризик помилок, що може призвести до кращого управління ресурсами та зниження загальних витрат. Більш того, впровадження аналітичних інструментів e-commerce дозволяє отримувати детальну статистику продажів, розуміти переваги клієнтів, аналізувати конверсії та використовувати ці дані для покращення маркетингових стратегій.

Одним із ключових аспектів успішної інтеграції e-commerce є забезпечення високого рівня безпеки для захисту особистих даних та платіжної інформації клієнтів. Інтеграція сертифікованих платіжних шлюзів і застосування шифрування даних є необхідними кроками для гарантування безпечного середовища для покупок. Це особливо важливо в умовах сучасних кіберзагроз, коли будь-яка слабкість у безпеці може призвести до втрати довіри з боку користувачів. Безпека є основою довгострокових відносин з клієнтами, тому забезпечення надійності e-commerce функцій має бути пріоритетом під час їх інтеграції.

Інтеграція e-commerce функцій також відкриває можливість створення персоналізованих пропозицій для користувачів, що значно підвищує ефективність продажів. Наприклад, рекомендаційні системи можуть пропонувати користувачам товари на основі їхніх минулих покупок або інтересів, створюючи більш релевантний досвід і підвищуючи рівень задоволення. Така персоналізація допомагає не тільки покращити обслуговування, а й збільшити середній чек та частоту повторних покупок, що є важливими показниками для зростання доходів.

Загалом, інтеграція e-commerce функцій дозволяє платформі стати більш універсальною, відповідаючи на різноманітні потреби користувачів і забезпечуючи додаткові можливості для бізнесу. Це робить платформу конкурентоспроможною, привабливою для нових клієнтів і здатною адаптуватися до змінних умов ринку.

Розвиток системи дозволяє відповідати мінливим потребам і очікуванням користувачів. Коли користувачі відчувають, що їхні відгуки беруться до уваги і що система покращується відповідно до їхніх побажань, рівень їхньої лояльності зростає. Розширення функціоналу, поліпшення дизайну, швидкості та зручності користування допомагають утримувати існуючих користувачів і залучати нових. Це особливо важливо для продуктів і сервісів, які залежать від високого рівня взаємодії та регулярної активності користувачів.

Ще одним аспектом важливості розвитку системи є підвищення її ефективності та безпеки. Технології швидко розвиваються, і старі методи захисту даних або інструменти оптимізації можуть втратити ефективність. Регулярні оновлення та впровадження нових безпекових протоколів дозволяють знизити ризики, пов'язані з можливими загрозами, та захистити дані користувачів від витоків або несанкціонованого доступу. Це особливо актуально в умовах зростаючих кіберзагроз, коли навіть невелика уразливість може призвести до значних збитків або втрати довіри з боку користувачів. Підтримка безпеки на високому рівні є фундаментом для будь-якої сучасної системи, яка працює з чутливими або конфіденційними даними.

Крім того, розвиток системи дозволяє оптимізувати її роботу та зменшити витрати на обслуговування. Застарілий код або технології можуть бути менш ефективними та вимагати більше ресурсів, що збільшує витрати на підтримку та роботу системи. Оновлення коду, оптимізація баз даних, поліпшення серверної архітектури та використання сучасних підходів дозволяють значно покращити продуктивність та зменшити навантаження на інфраструктуру. У результаті це не лише підвищує ефективність роботи системи, але й знижує витрати на її підтримку в довгостроковій перспективі, що позитивно впливає на загальну рентабельність проєкту.

Загалом, розвиток системи є необхідним не тільки для збереження її актуальності, але й для забезпечення надійного та зручного досвіду для користувачів. Інновації, інтеграція сучасних рішень, розширення функціоналу та підвищення рівня безпеки допомагають системі залишатися привабливою та ефективною, що забезпечує її тривале і стабільне використання на ринку.

**Висновки до розділу**

Таким чином було розроблено систему для збору інформації для підприємства, яка дозволяє зручно і ефективно керувати користувачами, аналітикою та продукцією, а також надано рекомендації з подальшого розвитку системи, зокрема перспективність розширення аналітичної складової, інтеграцію функцій електронної комерції та покращення користувацького досвіду.

# ВИСНОВКИ

Успішна діяльність малого і середнього бізнесу багато в чому залежить від здатності вчасно реагувати на зміни ринку, розуміти потреби клієнтів, керувати витратами та ефективно використовувати наявні ресурси. Застосунок для збору та обробки інформації надає цим підприємствам можливість оперувати аналітичними даними, які раніше були доступні лише крупним корпораціям, і допомагає оптимізувати процеси прийняття рішень.

Крім того, в умовах високої конкуренції малий і середній бізнес потребує інструментів, які дозволяють вчасно отримувати достовірну інформацію та формувати звіти, необхідні для оцінки стану компанії. Саме для таких потреб і потрібен застосунок збору та обробки інформації, адже він надає можливість зібрати дані з різних джерел, структурувати їх, провести глибокий аналіз та отримати рекомендації для подальших дій. Завдяки цьому, підприємства можуть ефективніше використовувати свої ресурси, швидше адаптуватися до умов ринку та знаходити нові можливості для розвитку, що є ключовим для їхньої життєздатності та зростання.

Ще один важливий аспект – доступність. Багато малих і середніх підприємств не можуть собі дозволити дорогі інформаційні системи або наймання професійних аналітиків. Розроблений застосунок, орієнтований на їхні потреби, заповнює цю прогалину, роблячи процес збору та аналізу інформації доступним, простим та зручним. Крім того, такий застосунок має бути адаптивним і зручним у використанні для працівників без спеціалізованих технічних знань. Це допомагає підприємствам з обмеженими ресурсами підвищувати свою конкурентоспроможність, використовуючи сучасні інформаційні технології.

Таким чином, розробка та впровадження застосунку збору та обробки інформації для малих і середніх підприємств є необхідним кроком для їхньої ефективної роботи та конкурентної переваги. Він надає доступ до важливих аналітичних інструментів, сприяє прийняттю обґрунтованих рішень, покращує внутрішні процеси і відкриває нові можливості для стратегічного розвитку компаній у сучасних умовах.

Дослідження показало, що процеси збору, обробки та аналізу даних мають особливе значення для підтримки ефективних управлінських рішень, особливо в умовах динамічного бізнес-середовища. У результаті аналізу було встановлено, що існуючі інформаційні системи часто не враховують потреби малого та середнього бізнесу, що обмежує їхні можливості для швидкої адаптації та прийняття обґрунтованих рішень.

Методи та технології, вивчені у межах даного дослідження, підтвердили необхідність створення доступного інструменту, який би відповідав специфічним вимогам підприємств із обмеженими ресурсами. Особлива увага приділялась адаптації сучасних технологій, що дозволяють мінімізувати витрати на впровадження і підтримку застосунку, зберігаючи при цьому високий рівень функціональності та гнучкості системи. Завдяки використанню сучасних підходів до обробки даних і автоматизації процесів аналізу, розроблений застосунок здатен значно підвищити ефективність управління і підтримати стратегічні рішення в бізнесі.

Практична цінність дослідження полягає в розробці інструменту, що забезпечує малий і середній бізнес ефективними засобами для роботи з інформацією, що сприяє оптимізації ресурсів, зменшенню витрат і підвищенню конкурентоспроможності. Застосунок створює умови для спрощеного аналізу ринкових трендів та адаптації до змін, що дозволяє підприємствам ефективніше використовувати дані для розвитку і стабілізації на ринку. Враховуючи обмеженість ресурсів, доступність такого інструменту має значний вплив на можливість малого та середнього бізнесу зростати, розвиватися та займати більш стійке положення у своїй галузі.

Наукова новизна цього дослідження полягає в адаптації інноваційних технологій обробки інформації до вимог малого і середнього бізнесу. Було створено інструмент, який не тільки враховує обмежені фінансові та кадрові можливості таких компаній, але й надає можливість масштабувати систему відповідно до потреб користувачів і особливостей різних галузей. Дослідження також продемонструвало можливості інтеграції з аналітичними системами, що автоматизує процес створення звітів і дозволяє компаніям проводити глибокий аналіз даних, підтримуючи їхню конкурентоспроможність та здатність реагувати на ринкові виклики.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ackoff R. L. From Data to Wisdom. Journal of Applied Systems Analysis, 1989, 16, 3-9.
2. Watson R. T. Information Systems. Global Text Project, University of Georgia, Collection open source textbooks, 2007, 1-33.
3. Stair R. M., Reynolds G. Fundamentals of business information systems. Thomson Learning, 2008, 118-129.
4. Laudon K. C., Laudon J. P. Management information systems: managing the digital firm. 12th ed., Prentice Hall, 2012.
5. Van Belle J. P., Nash J., Eccles M. Discovering Information Systems: an exploratory approach. University of Cape Town, 2010.
6. Davenport T. H., Harris J. G. Automated Decision Making Comes of Age. Sloan Management Review, 2005, 46(4), 83-89.
7. Kresse W., Danko D. M. Springer Handbook of Geographic Information. Springer Science & Business Media, 2012.
8. Bourgeois D. T. Information Systems for Business and Beyond. Washington: The Saylor Academy, 2014.
9. Nelson R. R. IT project management: infamous failures, classic mistakes, and best practices. MIS Quarterly Executive, 2007, 6(2), 67–78.
10. DeLone W. H., McLean E. R. Information systems success: The quest for the dependent variable. Information Systems Research, 1992, 3(1), 60–95.
11. Petter, Stacie, DeLone, William, McLean, Ephraim R. Information systems success: The quest for the independent variables. Journal of Management Information Systems, 2013, 29(4), 7-62.
12. DeLone, William H., McLean, Ephraim R. The DeLone and McLean model of information systems success: a ten-year update. Journal of Management Information Systems, 2003, 19(4), 9-30.
13. Ågerfalk, P. J., Conboy, K., Myers, M. D. Information systems in the age of pandemics: COVID-19 and beyond. 2020.
14. O’Leary, D. E. Evolving information systems and technology research issues for COVID-19 and other pandemics. Journal of Organizational Computing and Electronic Commerce, 2020, 30(1), 1-8.
15. Рішення для торгових точок для харчової промисловості та виробництва напоїв. URL: <https://www.truemerchant.com/food-and-beverage-pos/>.
16. Блейкборо Фоллоу Фокус, Р., Блейкборо Фокус, Р. 7 найкращих способів збільшити ваші доходи від їжі та напоїв. URL: <https://www.linkedin.com/pulse/top-7-ways-increase-your-food-beverage-revenues-russell-blakeborough>.
17. Oracle MICROS Simphony POS-Hotel Restaurant POS. URL: <https://www.oracle.com/th/industries/hospitality/products/simphony-point-of-sale-hotel.html>.
18. Команда TE. 6 Переваги POS-системи продуктів харчування та напоїв. URL: <https://www.embedcard.com/resources/blog/food-beverage-pos-system-benefits>.
19. Їжа та напої в гостинності: Тенденції в... - Плакат POS. URL: [https://joinposter.com/en/post/hotel-food-і-напій](https://joinposter.com/en/post/hotel-food-%D1%96-%D0%BD%D0%B0%D0%BF%D1%96%D0%B9).
20. Баррі П. Вступ до Python, 2-ге видання. O'Reilly Media, 2020. 624 с.
21. Грінберг М. Розробка веб-додатків з використанням Flask: Розробка веб-додатків з Python. O'Reilly Media, 2020. 258 с.
22. Свейгарт А. Автоматизуй нудну роботу за допомогою Python, 2-ге видання: Практичне програмування для абсолютних початківців. No Starch Press, 2020. 592 с.
23. Шоу З. Вивчаємо Python 3 складним шляхом: Дуже просте введення в жахливо красивий світ комп'ютерів та коду. Addison-Wesley Professional, 2021. 320 с.
24. МакКінні В. Python для аналізу даних: Обробка даних за допомогою Pandas, NumPy та IPython, 3-тє видання. O'Reilly Media, 2021. 550 с.
25. Вандер Плас Дж. Посібник з науки про дані Python: Необхідні інструменти для роботи з даними. O'Reilly Media, 2020. 612 с.
26. Алчін М. Про Python, 3-тє видання. Apress, 2020. 352 с.
27. Хант Дж. Просунуте керівництво з програмування на Python 3. Springer, 2020. 539 с.
28. Хетленд М. Початок роботи з Python: Від новачка до професіонала. Apress, 2020. 608 с.
29. Берд С., Кляйн Е., Лопер Е. Обробка природної мови з Python: Аналіз тексту за допомогою інструментального комплекту Natural Language Toolkit. O'Reilly Media, 2020. 504 с.
30. Мороні Л. ШІ та машинне навчання для програмістів: Посібник з штучного інтелекту. O'Reilly Media, 2020. 350 с.
31. Тартаро А. Програмування на Python для початківців: Введення в мову програмування Python і комп'ютерне програмування. Independently Published, 2021. 178 с.
32. Рамальйо Л. Вивчення Python: Чітко, лаконічно і ефективно. O'Reilly Media, 2020. 792 с.
33. Северенс К. Python для всіх: Дослідження даних у Python 3. CreateSpace Independent Publishing Platform, 2020. 252 с.
34. Лутц М. Програмування на Python: Потужне об'єктно-орієнтоване програмування, 4-те видання. O'Reilly Media, 2020. 1648 с.
35. Філліпс Д. Об'єктно-орієнтоване програмування на Python 3: Побудова міцних та утримуваних об'єктно-орієнтованих додатків Python. Packt Publishing, 2020. 770 с.
36. Лакшманан В., Шанкер С. Аналіз даних Python: Повне керівництво з аналізу даних та візуалізації. Packt Publishing, 2020. 368 с.
37. Харріс К. Програмування на Python і чисельні методи: Посібник для інженерів та вчених. Academic Press, 2021. 400 с.
38. Бізлі Д., Джонс Б. Книга рецептів Python: Рецепти для освоєння Python 3, 3-тє видання. O'Reilly Media, 2021. 706 с.
39. Bishop, C. M. Нейронні мережі для розпізнавання образів. Оксфорд, Англія: Oxford University Press.
40. Бішоп, C. M. Розпізнавання образів і машинне навчання (інформатика та статистика). Нью-Йорк, Нью-Йорк: Springer Science and Business Media.
41. Блок Х, Д. Перцептрон: модель функціонування Брайана, с. 123-135.
42. Карлінг, А. Знайомство з нейронними мережами. Вілмслоу, Великобританія: Sigma Press.
43. Д. Мічі, Д. Дж. Машинне навчання, нейронна та статистична класифікація. Prentice Hall Inc.
44. Fausett, L. Основи нейронних мереж. Нью-Йорк: Prentice Hall.
45. Форсайт, Р. С. Дивна історія персептрона. Огляд штучного інтелекту, с. 147-155.
46. Фрідберг, Р. М. Навчальна машина: частина 1. Журнал IBM, с. 2-13.
47. Гахрамані, З. Навчання без контролю: алгоритми призначені для вилучення структури з даних, с. 1-8. IOS Press.
48. Гілліс, Д. Штучний інтелект і науковий метод. OUP Оксфорд.
49. Хайкін, С. Нейронні мережі: комплексна основа. Нью-Йорк: Macmillan Publishing.
50. Ходж, В. Огляд методологій виявлення викидів. Огляд штучного інтелекту, с. 85-126.
51. Голланд, Дж. Адаптивні алгоритми для виявлення та використання загальних закономірностей у зростаючих базах знань, аналізі політики та інформаційних системах.
52. LambdaTest. Топ 12 веб-фреймворків Python: дізнайтеся про них, як про друзів URL: https://www.lambdatest.com/blog/top-12-python-web-development-frameworks-2024.
53. BrowserStack. Топ 10 веб-фреймворків Python у 2024 році URL: https://www.browserstack.com/blog/top-10-python-web-development-frameworks-2024.
54. KnowledgeHut. Топ 18 проєктів веб-розробки на Python у 2024 році з вихідним кодом URL: https://www.knowledgehut.com/blog/web-development/top-python-web-development-projects-2024.
55. Simplilearn. Тренди Python у 2024 році: остаточний гід по трендам та інноваціям URL: https://www.simplilearn.com/python-trends-2024-guide.
56. ThinkPalm. Які є топ фреймворки Python для веб-розробки у 2024 році? URL: https://thinkpalm.com/blog/python-web-development-frameworks-2024.
57. Digitalogy. Топ 10 фреймворків Python для веб-розробки у 2024 році URL: https://www.digitalogy.co/blog/top-python-web-development-frameworks-2024.
58. Roadmap.sh. Як стати сучасним розробником Python URL: https://roadmap.sh/python.