Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

«Допущено до захисту»
Завідувач кафедри
комп’ютерних
наук
докт. техн. наук, професор
_____Андрій БОНДАРЧУК
« ___» _____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня «Магістр»
Спеціальність 122 Комп’ютерні науки
Освітня програма 122.00.02 Інформаційно-аналітичні системи

Тема роботи:

Архітектура інформаційної системи та механізми її захисту в електронній комерції: технологічні рішення й регуляторні вимоги

Виконав студент
групи ІАСм-1-24-1.4д
Бондаренко Д.С.

_____
(підпис
) **Науковий керівник**
кандидат технічних
наук,
доцент
Машкіна І. В.

_____
(підпис)

Київ – 2025

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»

Завідувач

кафедри комп'ютерних наук,

к.т.н., доцент

Машкіна І.В.

« ___» _____2024р.

ЗАВДАННЯ НА КВАЛІФІКАЦІНУ РОБОТУ МАГІСТРА

«Архітектура інформаційної системи та механізми її захисту в електронній комерції: технологічні рішення й регуляторні вимоги»

Виконавець – студент групи
спеціальності 122 Комп'ютерні
науки,
освітньої програми 122.00.02 Інформаційно-аналітичні
системи
Бондаренко Дмитро Сергійович

1. **Вихідні дані**: Наукові статті та публікації в галузі безпеки інформаційних систем електронної комерції, архітектурних підходів , регуляторних вимог (GDPR, PCI DSS, українське законодавство) та застосування штучного інтелекту в е-комерції. Аналітичні звіти та документація до використовуваних технологій.
2. **Основні завдання**: Здійснити огляд існуючих аналогів, архітектурних підходів, загроз та регуляторних вимог для систем електронної комерції.Обрати та обґрунтувати методи та засоби дослідження.Обґрунтувати і розробити архітектурну модель

інформаційної системи електронної комерції та практичний веб-застосунок-прототип на базі обраного технологічного стеку.Провести аналіз результатів дослідження, скласти висновки та оформити кваліфікаційну роботу відповідно до затверджених на кафедрі вимог.

3. **Пояснювальна записка**: Обсяг – 60 стор. формату А4 комп’ютерного набору з дотриманням вимог стандарту ДСТУ 3008:2015 та методичних рекомендацій кафедри комп'ютерних наук.
4. **Графічні матеріали:** рисунки
5. **Додатки:** використовувані терміни
6. Строк подання роботи на кафедру: *«_» _____2025 р.*

Науковий керівник
канд. техн. наук, доцент

_____Машкіна І.В.

Виконавець:

_____Бондаренко Д.С.

РЕФЕРАТ магістерської роботи

Кваліфікаційна робота: 57 с., 21 рис., 1 табл., 35 літературних джерел, 1 додаток.

Актуальність даної роботи зумовлена поєднанням трьох критичних факторів: стрімкого зростання ринку електронної комерції, посилення кіберзагроз та впровадження жорстких нормативних вимог

Об'єкт дослідження: процеси побудови та захисту інформаційно-комунікаційних систем в електронній комерції.

Предмет дослідження: архітектурні рішення, методи захисту даних та нормативно-правові аспекти функціонування таких систем.

Мета розробка та практична реалізація безпечної архітектури інформаційної системи для е-комерції, здатної інтегрувати інтелектуальні сервіси.

Завдання :

- Провести аналітичне дослідження сучасного стану та загроз для систем е-комерції, виявити типові вразливості.
- Проаналізувати нормативно-правову базу, що регулює захист даних в е-комерції (GDPR, PCI DSS, українське законодавство).
- Спроектувати архітектурну модель системи на основі мікросервісного підходу з можливістю інтеграції AI-компонентів.
- Реалізувати працюючий веб-застосунок на стекі React, Node.js, TypeScript та PostgreSQL, який демонструє ключові принципи запропонованої архітектури.

У результаті виконання роботи проведено аналітичне дослідження сучасних загроз та архітектурних підходів, проаналізовано вимоги стандартів GDPR та PCI DSS. Практична реалізація включає веб-застосунок з механізмами автентифікації на основі JWT, захисту даних та інтеграцією з OpenAI API для генерації персоналізованих рекомендацій щодо продуктивності.

В результаті розроблено функціонуючу систему, яка ефективно поєднує безпеку та функціональність, враховує нормативно-правові аспекти та є масштабованою для різних бізнес-моделей. Отримані результати можуть бути використані як основа для створення реальних

систем е-комерції, що відповідають сучасним вимогам безпеки та обробки даних.

Практичне значення отриманих результатів полягає в створенні функціонуючої, гнучкої системи з документованими архітектурними рішеннями та рекомендаціями щодо відповідності стандартам, що може бути використано як основа для розробки комерційних рішень в українському та міжнародному сегментах е-комерції. Результати роботи були апробовані на наукових конференціях.

Наукова новизна полягає в розробці гібридної архітектури, яка поєднує технічні та правові аспекти безпеки, а також унікальної системи продуктивнісної аналітики на основі AI.

Ключові слова:

архітектура інформаційної системи, мікросервісна архітектура, електронна комерція , гібридна архітектура, інформаційна безпека

ЗМІСТ

Вступ

Сучасна електронна комерція стала потужним драйвером глобальної економіки. Однак разом із зростанням обсягів онлайн-продажів посилюються й кіберзагрози. За даними досліджень, лише у 2024 році через витоки даних у сфері е-комерції було скомпрометовано понад 22 мільярди записів, що підтверджує критичну потребу в надійних системах захисту. Одночасно з цим, впровадження таких нормативних вимог, як GDPR в ЄС та PCI DSS для платіжних операцій, створює додатковий виклик для розробників, які повинні поєднувати технічну безпеку з юридичною чистотою[1, 2].

Об'єкт дослідження: процеси побудови та захисту інформаційних систем в електронній комерції.

Предмет дослідження: архітектурні рішення, методи захисту даних та нормативно-правові аспекти функціонування таких систем.

Мета роботи полягає в розробці та практичній реалізації архітектури інформаційної системи для е-комерції, яка інтегрує механізми безпеки, відповідає регуляторним вимогам та підтримує інтелектуальні сервіси на основі штучного інтелекту.

Для досягнення мети було поставлено наступні **завдання**:

Провести аналітичний огляд сучасного стану, загроз та існуючих архітектурних підходів в е-комерції.

Дослідити нормативно-правову базу, що регулює захист даних.

Спроектувати гібридну архітектурну модель системи, що поєднує безпеку, масштабованість та можливість інтеграції AI.

Практично реалізувати веб-застосунок, що демонструє ключові принципи запропонованої архітектури.

Новизна роботи полягає в розробці гібридної архітектури, яка поєднує технічні та правові аспекти безпеки, а також унікальної системи продуктивнісної аналітики на основі AI.

Практична цінність полягає в створенні функціонуючої, гнучкої системи з документованими архітектурними рішеннями та рекомендаціями щодо відповідності стандартам, що може бути використано як основа для розробки комерційних рішень в українському та міжнародному сегментах е-комерції. Результати роботи були апробовані на наукових конференціях.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Сучасний стан ринку електронної комерції

Світовий ринок електронної комерції демонструє експоненційне зростання, прогнозоване досягнення обсягу понад $7 трильйонів до 2025-2026 років. Це зумовлено всесвітнім поширенням інтернет-доступу, підвищенням довіри до цифрових платежів та диверсифікацією товарних пропозицій. Лідерами за обсягами залишаються Азійсько-Тихоокеанський регіон та Північна Америка, однак найдинамічніше розширення спостерігається на ринках Латинської Америки та Південно-Східної Азії. Зростання трансформується з кількісного в якісне: збільшується середній чек, частота покупок та проникнення онлайн-торгівлі в нові сегменти, такі як B2B та цифрові послуги.

В Україні до 2022 року ринок показував одне з найвищих у Європі щорічних зростань (30-40%), значно прискорене пандемією COVID-19. Повномасштабне вторгнення завдало удару, але вже з другої половини 2022 року розпочалося швидке відновлення. За даними 2023-2024 років, обсяги не лише досягли довоєнного рівня, але й продовжили зростати[9]. Ключовими драйверами стали зміна споживчих звичок, розвиток логістики, адаптація бізнесу, розширення платіжних інструментів (електронні гаманці, розстрочки) та активізація міжнародної торгівлі.

Сучасну галузь визначають три ключові тенденції. Мобільна комерція (m-commerce) домінує, на яку припадає понад 60-70% онлайн-покупок, що зумовлює перехід до mobile-first дизайну та розвиток прогресивних веб-додатків. Omnichannel-підхід стирає межі між онлайн та офлайн-каналами, вимагаючи від бізнесу забезпечення

єдиного послідовного досвіду для клієнта (наприклад, послуги «click-and-collect»), що неможливо без централізованої CRM-системи. Персоналізація стала стандартом очікувань користувачів, що реалізується через алгоритми штучного інтелекту для рекомендацій, динамічного контенту та цільового маркетингу, безпосередньо впливаючи на конверсію та лояльність.

Пандемія COVID-19 виступила каталізатором глибокої цифровізації, примусово перевівши в онлайн навіть консервативні бізнеси та споживачів. Це прискорило впровадження хмарних технологій, сприяло появі нових бізнес-моделей (наприклад, прямі продажі виробника D2C) та сформувало високі вимоги до швидкості, зручності та безконтактності сервісів, таких як експрес-доставка. Таким чином, сучасна е-комерція є динамічним середовищем, сформованим кризами та технологічним прогресом, де успіх залежить від інтеграції мобільності, багатоканальності та дата-орієнтованої персоналізації.

1.2. Основні загрози безпеці в електронній комерції

Електронна комерція є пріоритетною мішенню для кіберзлочинців через прямий фінансовий мотив та обсяги конфіденційних даних. Статистика свідчить про стійкий ріст загроз. Фінансові втрати від одного витоку даних у роздрібній торгівлі у 2024 році в середньому становили $3.3 мільйони, при цьому загальна кількість скомпрометованих записів перевищила 22 мільярди[9]. Кількість атак на веб-додатки щорічно збільшується на 20-30%.

За структурою інцидентів, витоки даних (Data Breaches) становлять близько 35% усіх випадків, найчастіше через компрометацію серверів БД або незахищені API. Фінансове шахрайство (Fraud) завдає збитків на десятки мільярдів доларів щороку, включаючи платіжне

шахрайство, захоплення облікових записів (ATO, зростання на 40% у 2024) та зловживання поверненнями. Атаки типу DDoS, що націлені на параліч роботи сайту, становлять понад 25% усіх глобальних атак цього типу**, зростаючи у складності.**

Технічною основою цих загроз є класичні веб-вразливості, такі як SQL-ін'єкції, міжсайтовий скриптинг (XSS), підробка міжсайтових запитів (CSRF) та помилки контролю доступу, які посідають провідні місця в рейтингу OWASP Top 10[9, 10].

Таблиця 1.1. Порівняльний аналіз основних веб-загроз для систем е-комерції

Вразливості	**Суть**	**Наслідки**	**Актуальність**
Ін'єкції (Injection), зокрема SQL-ін'єкції	Зловмисник вставляє злий код (найчастіше SQL-запит) через поля вводу форми, параметри URL тощо. При недостатній валідації цей код виконується безпосередньо сервером або базою даних.	Викрадення, модифікація або повне видалення бази даних (дані клієнтів, замовлення, товари). Найкритичніший вплив на бекенд.	**Висока**. Класична, але все ще поширена загроза через застарілий код та недостатнє тестування безпеки додатків (SAST/DAST).
Міжсайтовий скриптинг (Cross-Site Scripting - XSS)	Вставка злого JavaScript-коду на веб-сторінку, який виконується в браузері іншого користувача.	Викрадення сесійних кук (ATO), перенаправлення на фішингові сайти, крадіжка даних платіжних форм на клієнтській стороні.	**Висока**. Часто виникає через недостатню фільтрацію/екранування даних, що виводяться на сторінку. Види: Відображений (Reflected), Збережений (Stored), DOM-базований.
Міжсайтова підробка запиту (Cross-Site Request Forgery - CSRF)	Браузер авторизованого користувача автоматично відправляє підроблений запит на вразливий сайт	Несанкціоновані дії від імені користувача: ATO, зміна критичних	**Помірна/Висока**. Ефективна, поки користувач авторизований. Запобігання потребує використання

	(наприклад, запит на зміну пароля або адреси доставки).	даних облікового запису, несанкціоновані покупки.	токенів (CSRF-токени).
Несанкціонований доступ (Broken Access Control)	Система неправильно перевіряє права доступу користувача. Наприклад, доступ до чужих даних через модифікацію ID в URL (IDOR) або доступ звичайного користувача до адмін-панелі.	Масові витоки конфіденційних даних клієнтів, несанкціоновані транзакції, компрометація всієї системи управління.	**Критично висока.** Найпоширеніша причина витоків даних. Включає широкий спектр вразливостей, таких як Insecure Direct Object Reference (IDOR).

Аналіз реальних кейсів компрометації

- Масштабний виток через API (2023): Великий міжнародний ритейлер втратив понад 100 мільйонів записів клієнтів через незахищений та відкритий API ендпоінт для мобільного додатка, що підкреслює критичність належної автентифікації та сканування API.
- Атака на ланцюг постачання (2024): Компрометація агрегатора платежів призвела до встановлення скриптів-скімерів на сторінки оплати тисяч клієнтських магазинів, демонструючи, як ризик поширюється через партнерські сервіси та необхідність аудиту постачальників.
- SQL-ін'єкція в українському маркетплейсі (2023): Дослідники виявили критичну вразливість у функції пошуку, що потенційно відкривала доступ до всієї бази даних, нагадуючи, що навіть лідери ринку вразливі до базових експлойтів та потребують регулярного тестування безпеки.

Отже, загрози в е-комерції вимагають комплексного захисту, що поєднує технічні (наприклад, запобігання ін'єкціям), організаційні (моніторинг, реагування) та правові (відповідність стандартам) заходи.

1.3. Існуючі архітектурні підходи до побудови систем е-комерції

Вибір архітектури визначає ключові характеристики системи е-комерції: масштабованість, гнучкість, безпеку та швидкість розробки. Історія її еволюції – це пошук оптимального балансу між складністю управління та функціональною повнотою.

Монолітна архітектура, де весь додаток є єдиною нерозривною одиницею, ідеальна для старту (MVP) завдяки простоті розробки, тестування та розгортання[4]. Однак, із зростанням проекту вона проявляє критичні недоліки: складність і витратність масштабування, накопичення технічного боргу, повільні цикли оновлення через високу зв'язаність компонентів та обмеження в технологічному виборі.

На противагу цьому, мікросервісна архітектура розбиває систему на незалежні сервіси, кожен з яких відповідає за конкретну бізнес-можливість. Це забезпечує неперевершені переваги: незалежне масштабування окремих компонентів, високу відмовостійкість, технологічну гнучкість та прискорену розробку силами автономних команд. Проте ці переваги досягаються ціною високої складності: система стає розподіленою, що породжує проблеми узгодженості даних, мережевих затримок та вимагає наявності потужної інфраструктури для оркестрації, моніторингу та логування.

Хмарні платформи (AWS, Azure, GCP) стали стандартом для розгортання, оскільки природно відповідають потребам е-комерції в гнучкому масштабуванні та високій доступності[11]. Від моделі IaaS, що дає повний контроль над віртуальною інфраструктурою, до PaaS, яка

абстрагується від серверного управління, та керованих SaaS-сервісів (бази даних, кешування, CDN, WAF) – хмара пропонує інструменти для реалізації будь-якої архітектури, суттєво прискорюючи time-to-market.

Досвід лідерів ринку підтверджує відсутність універсального рішення. Amazon є взірцем повного переходу на мікросервіси для досягнення неймовірної масштабованості. Shopify демонструє ефективність гібридного підходу, поєднуючи модульний моноліт з винесеними мікросервісами для ресурсомістких задач. Alibaba зосереджується на екстремальній оптимізації власної сервіс-орієнтованої архітектури та інфраструктури для пікових навантажень.

Таким чином, сучасний тренд полягає не у виборі між двома крайнощами, а в стратегічному розділенні системи на чітко визначені, незалежні компоненти (мікросервіси або модулі), розгорнутими в гнучкій хмарній екосистемі. Це дозволяє будувати архітектури, здатні балансувати між продуктивністю, безпекою та швидкістю адаптації до бізнес-вимог.

1.4. Механізми та технології захисту даних

Ефективний захист даних в е-комерції будується на багаторівневому підході, що охоплює всі шари системи — від мережевого транспорту до окремих записів у базі.

Контроль доступу є першим бар'єром. Автентифікація (перевірка "хто ви?") використовує не лише логін/пароль із обов'язковим хешуванням (bcrypt), але й багатофакторну автентифікацію (MFA) для запобігання захопленню акаунтів (ATO). Авторизація (перевірка "що вам дозволено?") реалізується через сесії (з серверним зберіганням стану) або, що частіше для сучасних систем, через безстанні JWT-токени, які ідеально підходять для мікросервісної архітектури. Для зручності

користувача та безпеки делегованого доступу використовується стандарт OAuth 2.0 / OpenID Connect[8].

Шифрування забезпечує конфіденційність даних. На рівні мережі (In Transit) це обов'язкове використання HTTPS/TLS 1.2/1.3. Для даних, що зберігаються (At Rest), застосовуються різні рівні: від базового шифрування диска до більш безпечного прозорого шифрування БД (TDE). Найвищий рівень захисту забезпечує шифрування на рівні застосунку, коли чутливі дані (наприклад, номери карток) шифруються в коді програми перед записом у БД, а ключі зберігаються окремо в спеціалізованих сховищах (Vault, KMS).

Захист на рівні додатку та інфраструктури запобігає конкретним атакам. Веб-брандмауер (WAF) аналізує та блокує шкідливий HTTP-трафік (SQL-ін'єкції, XSS). Валідація та санація вхідних даних, а також налаштування безпекових заголовків (CSP, HSTS) є обов'язковими практиками. На інфраструктурному рівні застосовуються мережева сегментація для ізоляції компонентів, системи виявлення вторгнень (IDS/IPS) та централізоване управління секретами для захисту конфіденційних ключів і паролів.

Інтелектуальний захист на основі AI додає проактивний рівень. На відміну від систем, що працюють за правилами (rule-based), машинне навчання аналізує поведінкові шаблони (патерни транзакцій, активності в акаунті, рухи миші) для побудови "профілю норми". Це дозволяє в реальному часі виявляти аномалії, такі як шахрайські операції, спроби захоплення акаунту (ATO) або активність зловмисних ботів, пропонуючи адаптивний захист проти нових, невідомих загроз.

Таким чином, сучасна система безпеки в е-комерції — це динамічна комбінація фундаментальних криптографічних механізмів, засобів контролю доступу, інфраструктурних заходів та інтелектуальних

AI-рішень, що разом формують живу, здатну до навчання захисну екосистему.

1.5. Нормативно-правове регулювання

Дотримання регуляторних норм є обов'язковою вимогою для функціонування систем е-комерції, що працюють з персональними та платіжними даними, оскільки порушення тягне за собою величезні штрафи, судові позови та втрату довіри клієнтів.

Загальний регламент про захист даних (GDPR) регулює обробку даних громадян ЄС, незалежно від місцезнаходження компанії[1]. Його ключові принципи включають законність, прозорість, мінімізацію та точність даних, обмеження термінів зберігання, забезпечення цілісності та конфіденційності («Security by Design»), а також надання суб'єктам даних прав на доступ, виправлення, видалення («право на забуття») та переносимість. Для системи це означає необхідність отримання інформованої згоди, реалізацію механізмів для виконання запитів користувачів, повідомлення про витоки даних протягом 72 годин та ведення реєстру обробки. Штрафи можуть сягати 20 млн євро або 4% від глобального обороту.

Стандарт безпеки PCI DSS є технічно-операційним обов'язковим стандартом для всіх, хто працює з даними платіжних карток[2]. Його 12 вимог охоплюють мережевий захист, шифрування даних під час передачі та зберігання, контроль доступу, регулярне тестування безпеки та реалізацію політик інформаційної безпеки. На практиці це забороняє зберігати чутливі дані картки (CVV, магнітну доріжку) після авторизації, дозволяючи зберігати лише зашифровані номер, термін дії та ім'я власника, та вимагає щорічного сертифікування.

Невиконання карається платіжними системами штрафами або забороною прийому платежів.

Українське законодавство в цій сфері гармонізоване з європейськими стандартами. Закон «Про захист персональних даних» після змін 2022 року де-факто наближається до GDPR, встановлюючи схожі принципи та права. Також діють закони «Про електронну комерцію» та «Про захист прав споживачів», що регулюють онлайн-продажі та право на повернення[3].

Вплив вимог на архітектуру системи є прямим та суттєвим. Вони формують архітектуру даних, вимагаючи сегментації та ізоляції чутливої інформації (особливо платіжної), підтримки шифрування на рівні застосунку та технічної можливості повного видалення даних («право на забуття»)[1, 2, 3]. Мікросервісна архітектура стає ефективною відповіддю, дозволяючи створити ізольовані, надзвичайно захищені сервіси для платежів (для PCI DSS) або управління згодою (для GDPR). Крім того, вимоги до логування всіх доступів (PCI DSS, GDPR) та регулярного тестування безпеки (PCI DSS) інтегруються в архітектуру через централізовані системи моніторингу та вбудовуються в процеси DevSecOps.

Нормативно-правове поле не є зовнішнім обмеженням, а стає архітектурною рамкою, що безпосередньо визначає вибір технологій, проектування моделей даних та комунікацію між компонентами, спрямовуючи розробку на створення не лише функціональної, але й юридично чистої, безпечної та надійної системи.

1.6. Використання штучного інтелекту в е-комерції

Штучний інтелект перетворився з тренду на ключовий інструмент для підвищення конкурентоспроможності та безпеки в е-комерції[9, 10]. Його роль подвійна: він одночасно є двигуном персоналізації та проактивним захисним механізмом.

AI для покращення користувацького досвіду реалізується через три основні канали. Персоналізація в реальному часі адаптує контент (головні сторінки, банери) під інтереси конкретного користувача на основі аналізу його поведінки. Рекомендаційні системи, побудовані на алгоритмах колаборативної, контентної або гібридної фільтрації, прогнозують релевантні товари, значно підвищуючи конверсію (як це роблять Amazon чи Netflix). Передбачувальна аналітика використовує ML для оптимізації бізнес-процесів: прогнозування попиту, аналізу відтоку клієнтів (Churn Prediction) та динамічного ціноутворення. Додаткові інструменти, такі як чат-боти з NLP та аналіз поведінки (heatmaps, прогнозування наміру покупки), забезпечують оперативну підтримку та глибше розуміння користувача.

AI як інструмент безпеки заповнює прогалини традиційних rule-based систем. На відміну від жорстких правил, моделі машинного навчання аналізують мільйони транзакцій, формуючи профіль нормальної поведінки на основі сотень факторів: поведінкової біометрії, даних пристрою, транзакційних та мережевих характеристик. Це дозволяє в реальному часі виявляти аномалії, такі як шахрайські операції (призначаючи «рахунок ризику»), спроби захоплення облікових записів (ATO) та активність зловмисних ботів для скрейпінгу або DDoS, пропонуючи адаптивний захист.

Підсумовуючи аналітичний огляд, можна виокремити три головні проблеми сучасних систем е-комерції: **дисонанс між безпекою та зручністю, архітектурну негнучкість** застарілих монолітів та фрагментацію підходів до відповідності різним стандартам (GDPR, PCI

DSS). Ця робота покликана заповнити цю прогалину, пропонуючи не набір інструментів, а цілісну архітектурну модель.

Обґрунтуванням цієї моделі є **гібридна архітектура з AI-компонентами**. Вона поєднує переваги мікросервісів для критичних та інтенсивних компонентів (платежі, аутентифікація, AI-сервіси) із модульним підходом для ядра бізнес-логіки, уникаючи надмірної складності повністю розподіленої системи. AI виступає тут як інтегральний «клей», що одночасно забезпечує цінність для користувача (персоналізація) та для бізнесу (проактивна безпека). Таким чином, запропоноване рішення прямо відповідає на виклики сучасної е-комерції, пропонуючи практично орієнтовану, безпечну, масштабовану та регуляторно-відповідну архітектуру.

РОЗДІЛ 2. ВИБІР АРХІТЕКТУРИ ТА ТЕХНОЛОГІЙ РОЗРОБКИ

2.1. Концептуальна архітектура системи е-комерції

Для досягнення цілей сучасної, масштабованої та безпечної системи е-комерції було обрано гібридну архітектуру[4, 5]. Цей підхід інтегрує переваги мікросервісів для критичних компонентів і модульного моноліту для ядра бізнес-логіки, забезпечуючи оптимальний баланс між гнучкістю, продуктивністю та контролем складності розробки.

Концептуальна модель системи ґрунтується на принципах сервіс-орієнтованої архітектури (SOA) і включає наступні ключові компоненти:

- Користувацький інтерфейс (**Frontend**): Односторінковий додаток (SPA) на React.js, що відповідає за візуалізацію та взаємодію.
- **API Gateway**: Єдина точка входу, що забезпечує маршрутизацію, аутентифікацію на рівні шлюзу, обмеження запитів та логування.
- **Сервіс автентифікації та авторизації**: Відокремлений мікросервіс для управління користувачами, генерації JWT та інтеграції OAuth.
- **Сервіс управління завданнями**: Ядро системи, реалізоване як модульний моноліт для обробки CRUD-операцій, що дозволяє зберегти простоту розробки.
- **Платіжний сервіс** (концепт): Виділений мікросервіс для ізоляції та безпечної обробки фінансових транзакцій відповідно до PCI DSS.
- **AI-сервіс аналітики**: Окремий мікросервіс для інтеграції з OpenAI API, що відповідає за генерацію порад (getTipForTodo) та аналіз продуктивності (analyzeLastNTasks).

– **База даних**: PostgreSQL як надійне сховище даних із підтримкою транзакцій та ORM Sequelize для абстракції.

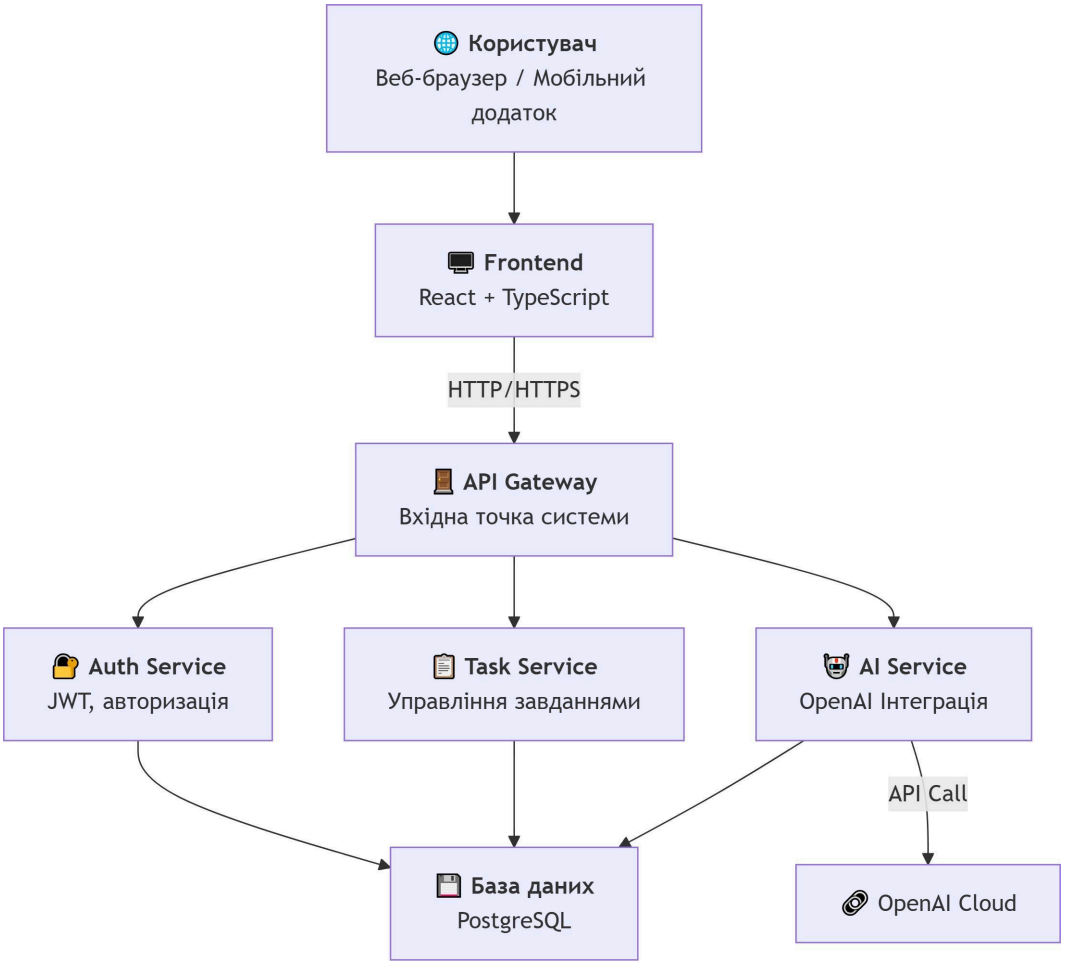


Рис 2.1: Архітектура з технологічним стеком

Обґрунтування гібридного підходу полягає у його перевагах для масштабування та безпеки. З точки зору масштабування, він дозволяє незалежно розширювати ресурсоємні компоненти (AI, платежі), використовувати різні технології для різних сервісів та забезпечує високу доступність за рахунок ізоляції відмов. З точки зору безпеки, архітектура реалізує принцип мінімальних привілеїв, зменшує поверхню атаки через виділення критичних сервісів та запобігає випадковому доступу до даних на структурному рівні.

Таким чином, обрана архітектура забезпечує шлях від відносно простої початкової реалізації до повноцінної мікросервісної системи, оптимально балансуючи витрати на розробку з вимогами безпеки, продуктивності та майбутнього зростання.

2.2. Проектування моделі даних

Модель даних є фундаментом всієї системи, що визначає цілісність, продуктивність та масштабованість. Для забезпечення ефективного зберігання та обробки даних було спроектовано реляційну модель, що базується на сутностях та їх взаємозв'язках, з чітко визначеними атрибутами та обмеженнями.

ER-діаграма (Entity-Relationship): Сутності та зв'язки між ними

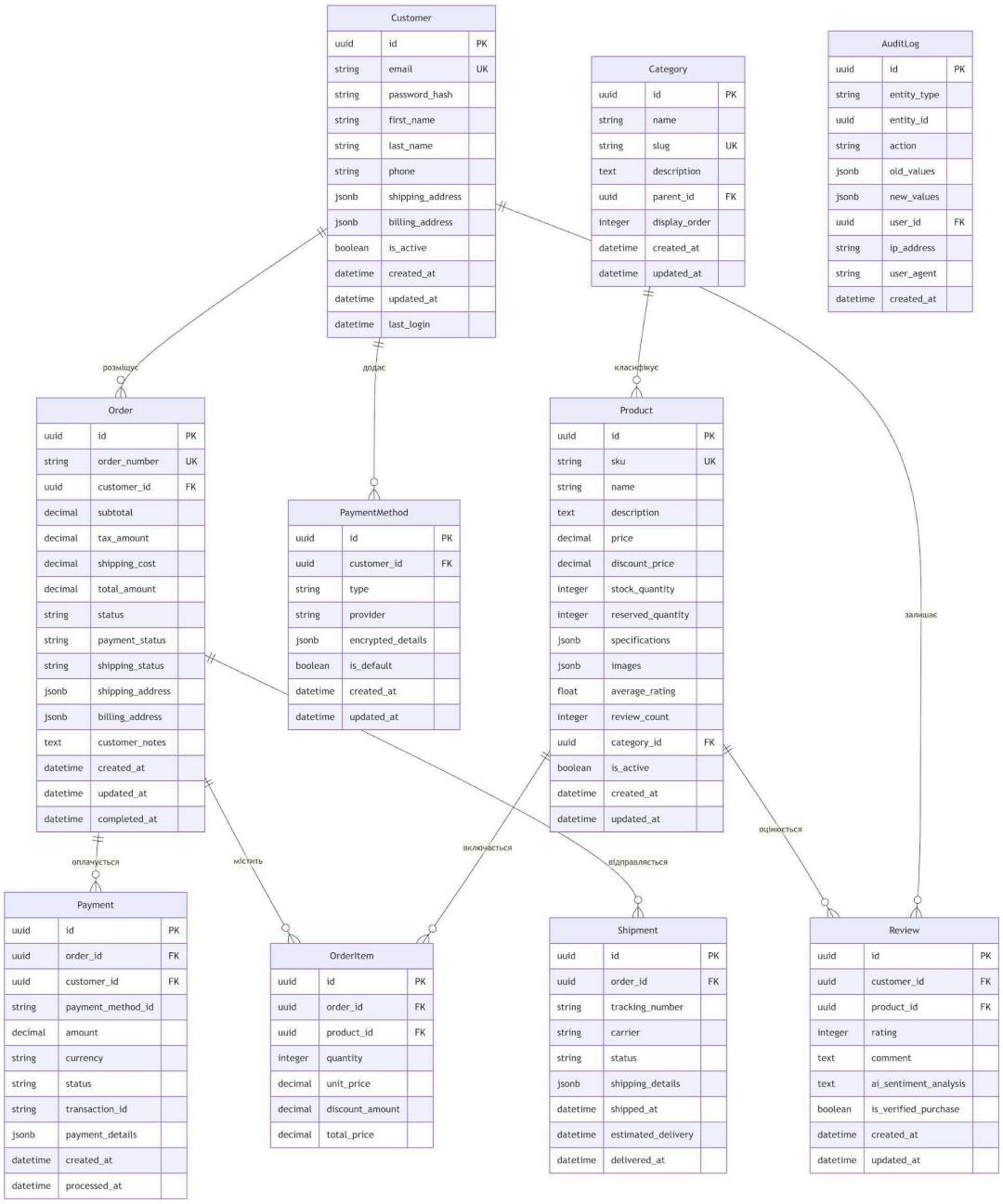


Рис 2.2 Схема взаємозв'язків сутностей (Entity-Relationship Diagram)

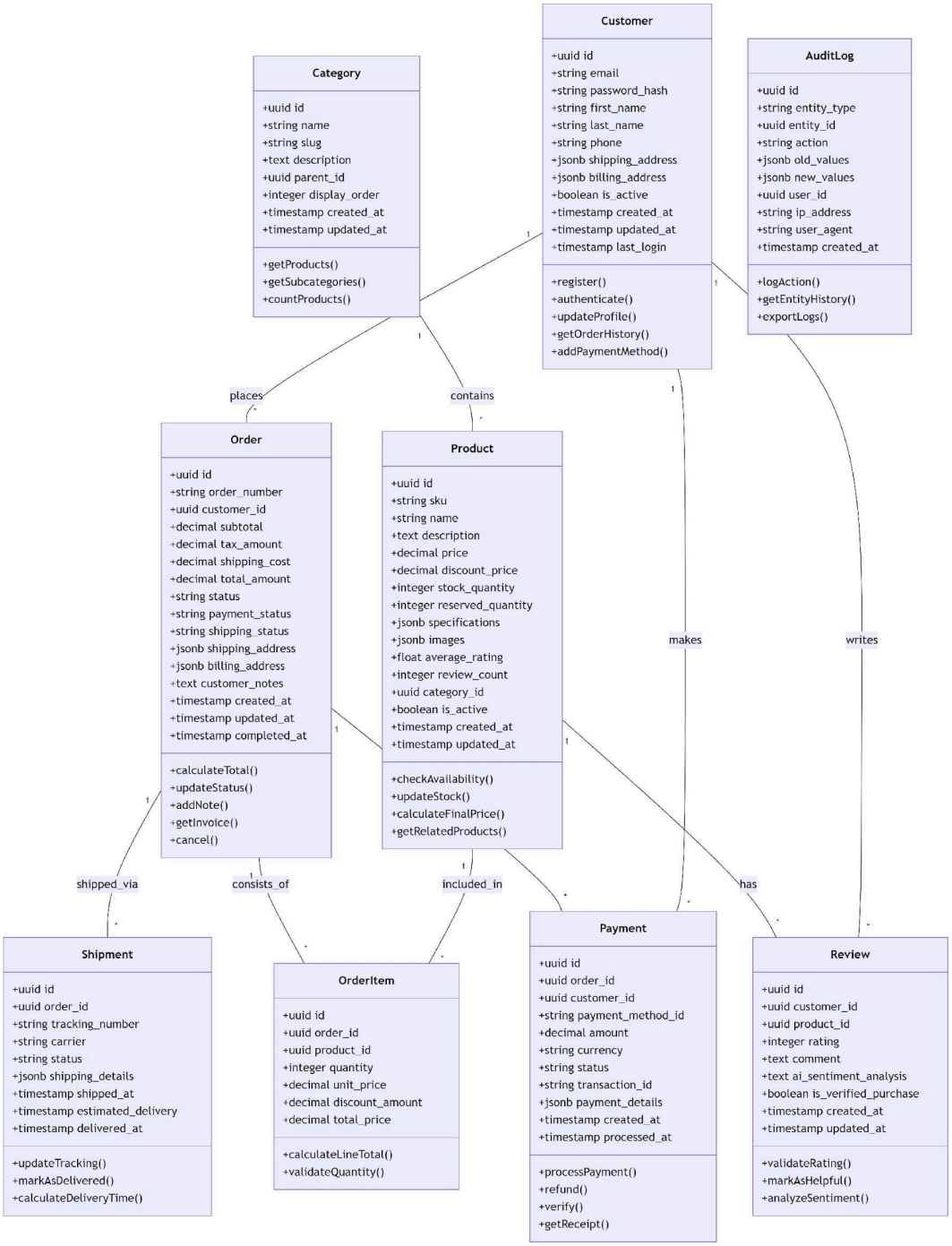


Рис 2. 3 Фізична модель даних (Physical Data Model)

Ключові аспекти проектування:

- Нормалізація: Модель приведена до 3-ї нормальної форми (3NF) для усунення надмірності даних та аномалій вставки, оновлення та видалення.
- Індекси: Для підвищення продуктивності запитів плануються індекси на поля, що використовуються в умовах WHERE та JOIN: Users.email, Todos.user_id, Todos.status, Todos.category_id.
- Цілісність даних: Використання зовнішніх ключів (Foreign Keys) гарантує реляційну цілісність - неможливо створити завдання для неіснуючого користувача.
- Масштабованість: Окремі таблиці для категорій та транзакцій дозволяють легко розширювати функціонал без модифікації основної таблиці Todos.

Запропонована модель даних є гнучкою, нормалізованою та готовою до масштабування. Вона ефективно підтримує поточний функціонал керування завданнями та AI-аналітики, а також забезпечує основу для впровадження додаткових функцій, таких як категорії, фінансові операції та розширена аналітика, що є типовими для систем е-комерції.

2.3. Проектування механізмів безпеки

Усі механізми безпеки проєктуються на основі принципів «Security by Design» та «Defence in Depth» (захист у глибину), формуючи багаторівневу систему захисту, інтегровану в архітектуру[10].

Для контролю доступу обрано безстанну автентифікацію на основі JWT (JSON Web Token)[8]. Після перевірки логіну та хешованого

пароля (bcrypt) сервер генерує токен з обмеженим терміном дії. Кожен наступний запит до захищених ендпоінтів повинен містити цей токен у заголовку Authorization: Bearer <token>, що перевіряється сервером на валідність підпису та термін дії. Система розрізняє два рівні доступу: Користувач (повний доступ до власних даних) та Адміністратор (доступ до всіх системних даних), роль якого також кодується в полі JWT.

Рис 2. 4 Схема автентифікації та авторизації на основі JWT-токенів

Захист цілісності та конфіденційності даних досягається через шифрування. Паролі обов'язково хешуються алгоритмом bcrypt. Конфіденційні дані (наприклад, email) можуть додатково шифруватися на рівні застосунку (AES-256) перед записом у БД, з ключами у спеціалізованих сховищах (HashiCorp Vault, AWS KMS). Весь трафік між клієнтом та сервером захищається протоколом HTTPS/TLS 1.3.

Протидія веб-загрозам включає наступні заходи:

- **CSRF**: Використання атрибута SameSite для кук та перевірка власника токена.

− **XSS**: Автоматичне екранування в React на фронтенді, валідація та санація вхідних даних на бекенді, налаштування заголовка Content-Security-Policy.

− **SQL**-ін'єкції: Виключне використання ORM (Sequelize) для параметризованих запитів.

− **DDoS**: Обмеження швидкості запитів (rate limiting) на рівні API Gateway.

− **Несанкціонований доступ**: Перевірка прав доступу на бекенді для кожного запиту (наприклад, належності завдання користувачеві).

Архітектурна підготовка до відповідності стандартам закладена в системі[1, 2, 3]. Для GDPR це мінімізація даних, реалізація ендпоінтів для виконання прав на доступ та видалення («право на забуття»), отримання згоди та захист даних через шифрування. Мікросервісна архітектура дозволяє виділити спеціалізований сервіс для обробки запитів користувачів. Для майбутньої відповідності PCI DSS архітектура передбачає повну ізоляцію платіжного сервісу, шифрування даних картки на рівні застосунку, детальне логування та планування регулярного тестування безпеки.

Запропонована архітектура формує комплексну систему захисту, яка не лише протистоїть атакам, але й закладає технічну основу для відповідності суворим міжнародним стандартам.

2.4. Проектування AI-компонентів системи

Проєктування AI-компонентів спрямоване на їхню стратегічну інтеграцію як невід'ємної частини системи, що забезпечує додаткову

цінність через механізми персоналізації, аналітики та проактивної підтримки користувача. Запропонована архітектура передбачає реалізацію AI-функціоналу у вигляді виділеного мікросервісу, що забезпечує ізоляцію складних обчислень, незалежне масштабування та ефективне керування ресурсами.

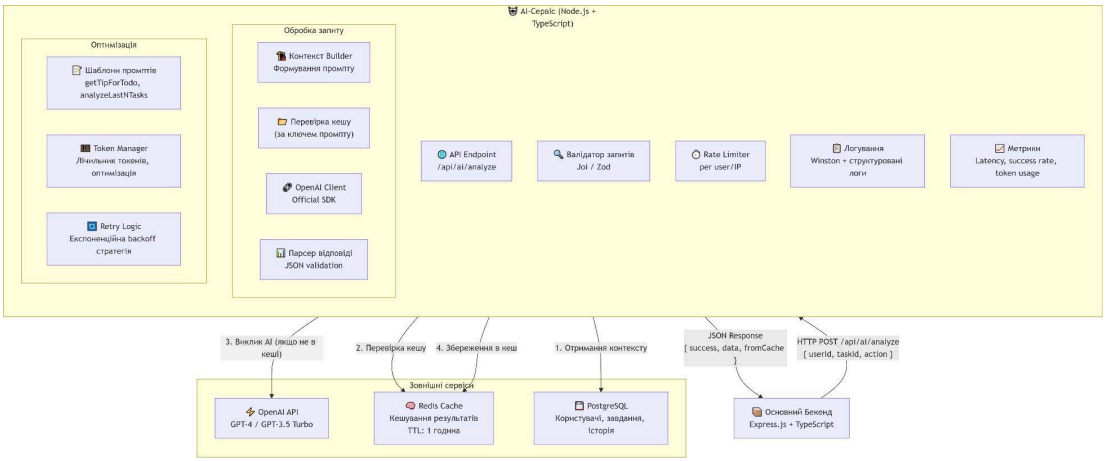


Рис.2.5 Блок-схема: архітектура AI-сервісу

Архітектурно взаємодія організована таким чином: клієнтський запит ініціює виклик основного бекенду, який формує та перенаправляє запит до AI-сервісу. Останній, отримавши необхідні дані з основного сховища, конструює структурований запит (prompt) до зовнішнього OpenAI API, виконує його через безпечний канал з використанням ключа зі змінних середовища, а потім валідує, обробляє та повертає структуровану відповідь основним компонентам системи.

В рамках роботи визначено три ключові сценарії використання штучного інтелекту:

− **Оперативна допомога (getTipForTodo):** Режим надання контекстної поради для поточного завдання на основі аналізу його

метаданих (назва, стан, орієнтовний та фактичний час виконання). Метою є покращення індивідуальної продуктивності.

− **Аналітичний огляд (analyzeLastNTasks):** Режим комплексного аналізу паттернів продуктивності користувача на основі вибірки останніх N завдань. Результатом є структуровані рекомендації щодо оптимізації робочих процесів.

− **Виявлення аномалій (концепт):** Фоновий режим моніторингу агрегованих даних активності для проактивного виявлення відхилень від типової поведінки (наприклад, ознак стресу чи неефективного планування).

Підготовка даних для AI включає їх вибірку з нормалізованих таблиць (Users, Todos, Categories) з подальшою структурною трансформацією та обов'язковою анонімізацією персональних даних (PII) для забезпечення конфіденційності. Критичним етапом є інженерія запитів (prompt engineering), що полягає у формуванні чіткого завдання, вказівки формату відповіді та обмежень на тон і стиль комунікації, що забезпечує передбачуваність та якість результату.

Запропонований підхід до інтеграції AI перетворює систему з інструмента виконання завдань на адаптивну платформу з інтелектуальною підтримкою, де технологія використовується цілеспрямовано для підвищення ефективності користувача та загальної якості сервісу.

2.5. Вибір технологічного стеку

Вибір технологічного стеку визначає ключові атрибути системи: продуктивність, масштабованість та підтримуваність. Для реалізації було

обрано сучасний та взаємодоповнюючий набір інструментів, що відповідає архітектурним вимогам.

Фронтенд: React, TypeScript, Ant Design. Фронтенд реалізовано на бібліотеці React, що забезпечує компонентну архітектуру для створення багаторазових UI-елементів та високої продуктивності за рахунок віртуального DOM[6, 7]. Використання TypeScript додає статичну типізацію, що підвищує надійність коду, покращує підтримку середовищ розробки та забезпечує узгодженість типів даних з бекендом. Бібліотека компонентів Ant Design прискорює розробку за рахунок готових, доступних та візуально консистентних елементів інтерфейсу.

Бекенд: Node.js, Express, TypeScript. На серверній стороні використано середовище Node.js, що забезпечує високу продуктивність за рахунок неблокуючої, подієвої архітектури, ідеальної для обробки I/O-операцій[8]. Фреймворк Express надає мінімалістичну та гнучку основу для побудови REST API з потужною middleware-архітектурою. TypeScript застосовано на бекенді для забезпечення типобезпеки, безпечної роботи з базою даних та спрощення рефакторингу.

База даних: PostgreSQL, Sequelize. В якості системи керування базами даних обрано PostgreSQL через її надійність, підтримку ACID-транзакцій, потужні аналітичні можливості та масштабованість. ORM Sequelize слугує абстракцією для роботи з БД, забезпечуючи безпеку від SQL-ін'єкцій, зручне визначення зв'язків між сутностями та інструменти для керування міграціями схем.

Інфраструктура розгортання. Для забезпечення доступності компонентів системи використано спеціалізовані хостинг-сервіси. Фронтенд розгорнуто на GitHub Pages для безкоштовного хостингу статичних файлів з інтеграцією CI/CD. Бекенд розміщено на Render.com, що пропонує простий процес розгортання з безкоштовним стартовим

планом та автоматичним масштабуванням. База даних PostgreSQL хоститься на Neon.tech — сервісі, що надає масштабоване serverless-рішення з економією ресурсів.

Обраний стек технологій забезпечує сучасний фундамент для розробки, що поєднує ефективність, безпеку та сприяє швидкій ітераційній розробці.

2.6. Проектування інтерфейсу користувача

Проектування інтерфейсу користувача ґрунтується на принципах орієнтованості на користувача (user-centered design), що забезпечує інтуїтивність, ефективність та доступність системи. Метою є створення зручного інструменту для виконання завдань з мінімальними когнітивними навантаженням.

Інформаційна архітектура системи організована навколо двох ключових сценаріїв: операційного (управління завданнями) та аналітичного. Це досягається завдяки логічному розділенню інтерфейсу на відповідні зони в рамках односторінкового додатку (SPA), що усуває необхідність перезавантаження сторінок та забезпечує миттєву відповідь. Пріоритетність надається основним діям (створення, перегляд), які візуально акцентуються, тоді як допоміжні операції (редагування) виконуються в модальних вікнах для збереження контексту.

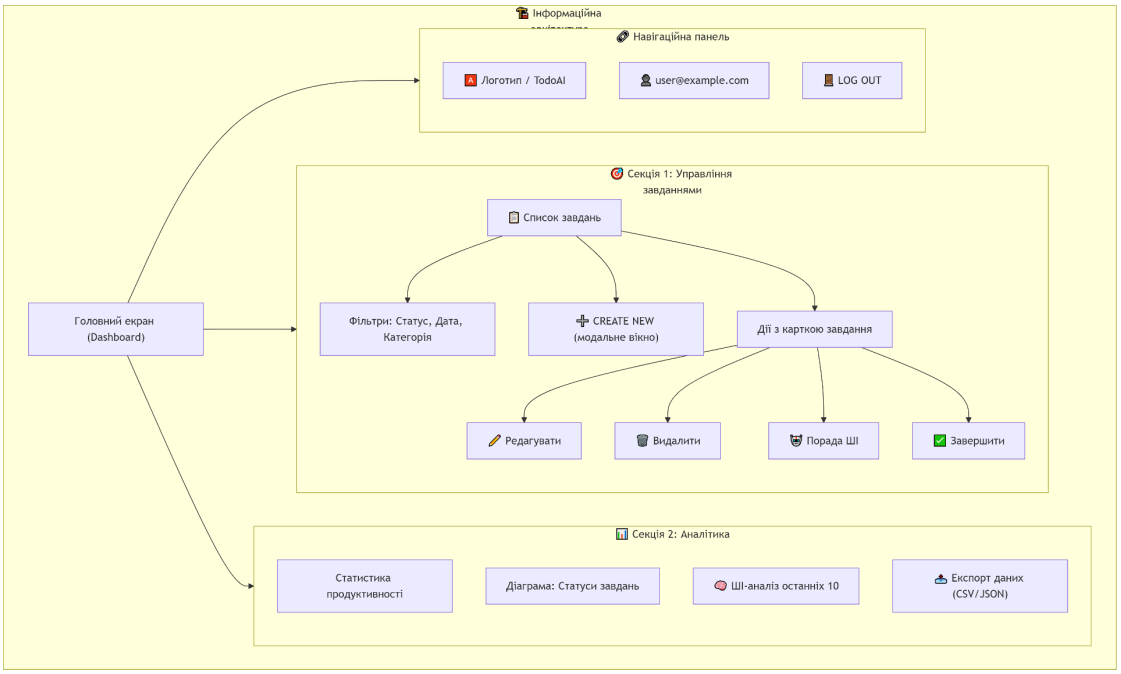


Рис.2.6 Ієрархічна UI-орієнтована структура

Візуальна складова реалізована на основі дизайн-системи Ant Design, що гарантує професійний вигляд та послідовність. Ключовими принципами є чіткість, візуальна ієрархія та консистентність усіх елементів інтерфейсу. Кольорова схема використовує семантичне кодування: синій колір для основних дій та стану «в роботі», зелений – для успішного завершення, помаранчевий – для попереджень, червоний – для помилок. Нейтральна сіра гама застосовується для фонів та другорядних елементів.

Забезпечення юзабіліті досягається через оптимізацію основних робочих процесів. Це включає мінімалістичні форми для швидкого створення завдань, миттєву фільтрацію та можливість швидкої зміни статусів. Для запобігання помилок реалізовано валідацію форм у реальному часі, підтвердження руйнівних дій та зрозумілі повідомлення про помилки.

Доступність інтерфейсу забезпечується підтримкою клавіатурної навігації, наявністю ARIA-атрибутів у компонентах Ant Design,

достатнім рівнем контрастності та адаптивним дизайном для різних пристроїв. Для полегшення навчання використовуються стандартні інтуїтивні іконки, а розширений функціонал представлений прогресивно, без перевантаження новачка.

Запропонований підхід до проектування інтерфейсу інтегрує логічну структуру, сучасну візуальну презентацію та глибоке розуміння ергономіки для створення ефективного та доступного інструменту управління.

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ СИСТЕМИ

3.1 Конструювання і розробка системи

Для створення сучасного веб-застосунку вибір технологічного стеку є фундаментальним, оскільки він безпосередньо впливає на гнучкість, продуктивність та можливості майбутнього масштабування системи. Архітектура заснована на клієнт-серверній моделі, що забезпечує чітке розділення обов'язків між користувацьким інтерфейсом (frontend) та серверною логікою (backend), сприяючи незалежній оптимізації кожного шару та спрощуючи процес обслуговування. У контексті розробленого ToDo-застосунку, який дозволяє управляти завданнями зі зміною станів, планувати час їх виконання та отримувати інтелектуальні рекомендації через інтеграцію з OpenAI API, коректний підбір інструментів дозволив ефективно реалізувати концепцію з мінімальними витратами на супровід.

Фронтенд-частина реалізована з використанням бібліотеки React.js та мови TypeScript. React.js надає компонентний підхід до побудови інтерфейсу, що сприяє створенню модульних, багаторазово використовуваних елементів та знижує загальну складність коду. TypeScript, як надбудова над JavaScript, впроваджує статичну типізацію, що дозволяє виявляти помилки на етапі компіляції, підвищує читабельність та робить код більш передбачуваним, особливо при роботі зі складними структурами даних завдань.

Для прискорення розробки візуальної складової та забезпечення професійного вигляду інтерфейсу була застосована бібліотека готових компонентів Ant Design (antd)[6, 7]. Її використання дозволило сконцентруватися на бізнес-логіці застосунку, мінімізуючи час на

створення базових UI-елементів, таких як форми, кнопки, модальні вікна та таблиці.

Серверна частина побудована на основі середовища виконання Node.js та фреймворку Express.js з підтримкою TypeScript[8]. Node.js, з його асинхронною, неблокуючою архітектурою, ефективно обробляє велику кількість одночасних підключень, що є важливим для продуктивності веб-сервісу. Express.js надає мінімалістичну та гнучку основу для побудови RESTful API, спрощуючи маршрутизацію, обробку запитів та впровадження проміжного програмного забезпечення (middleware), наприклад, для авторизації. Використання TypeScript на бекенді забезпечує узгодженість типів даних по всій системі та підвищує надійність серверної логіки.

Для зберігання та управління даними обрано реляційну систему керування базами даних PostgreSQL, що характеризується надійністю, підтримкою ACID-транзакцій та потужними аналітичними можливостями. Для взаємодії з БД використано ORM Sequelize, який абстрагує роботу з базою даних, представляючи таблиці та записи у вигляді об'єктів, що спрощує операції CRUD (створення, читання, оновлення, видалення) та забезпечує захист від SQL-ін'єкцій.

Налаштування для інтеграції зі штучним інтелектом реалізована за допомогою офіційної бібліотеки OpenAI для Node.js. Це дозволить організувати безпечне спілкування з API моделей GPT для генерації контекстних порад щодо виконання завдань та проведення аналізу продуктивності користувача на основі історії діяльності, розширюючи функціональність застосунку та підвищуючи його користь.

Розгортання системи. Для забезпечення доступності та надійності застосунку компоненти системи були розгорнуті на спеціалізованих хостинг-платформах. Фронтенд, як статичний односторінковий додаток (SPA), розміщено на GitHub Pages, що

забезпечує безкоштовний хостинг, простий процес розгортання через інтеграцію з Git та автоматичне оновлення. Серверна частина (бекенд) розгорнута на платформі Render.com, яка пропонує зручний інтерфейс для розгортання, безкоштовний стартовий план та можливості автоматичного масштабування. База даних PostgreSQL хоститься на сервісі Neon.tech, що надає сучасне serverless-рішення для PostgreSQL з автоматичним масштабуванням, зручним моніторингом та ефективним використанням ресурсів.

Система контролю версій. Процес розробки супроводжувався системою контролю версій Git, що дозволяє ефективно відстежувати зміни в коді, працювати в паралельних гілках та інтегрувати новий функціонал. Для зберігання репозиторію та спільної роботи використана платформа GitHub. Вона надає веб-інтерфейс для перегляду коду, управління завданнями (issues) та запитами на злиття (pull requests), а також легко інтегрується з інструментами безперервної інтеграції та доставки (CI/CD), сприяючи автоматизації тестування та розгортання[11]. Використання Git та GitHub створило надійну основу для керування життєвим циклом проєкту, забезпечуючи прозорість, контроль версій та можливість швидкого реагування на виявлені проблеми.

3.2 Реалізація клієнтської (фронтенд) частини проєкту

Створення клієнтської частини застосунку розпочато з ініціалізації проекту за допомогою інструменту Create React App, налаштованого на використання TypeScript. Це забезпечило готову конфігурацію з оптимальною структурою каталогів, попередньо налаштованими засобами збірки (Webpack) та транспіляції коду (Babel), що сприяє

швидкому старту розробки та гарантує сумісність з сучасними середовищами виконання.

Після базової настройки було додано та інтегровано ключові бібліотеки. Для побудови інтерфейсу користувача використана бібліотека Ant Design (antd), що надає набір готових, стилістично єдиних UI-компонентів. Це дозволило прискорити розробку візуальної складової, зосередившись на реалізації бізнес-логіки, та забезпечити професійний зовнішній вигляд додатку.

Організація навігації між різними функціональними розділами застосунку (авторизація, список завдань, аналітика) була реалізована за допомогою бібліотеки React Router. Для коректної роботи односторінкового додатку (SPA) на хостингу GitHub Pages було обрано реалізацію маршрутизатора HashRouter. Такий підхід забезпечив стабільну роботу навігації та залишає можливість для майбутнього переходу на інші типи роутерів (наприклад, BrowserRouter) при зміні умов розгортання.

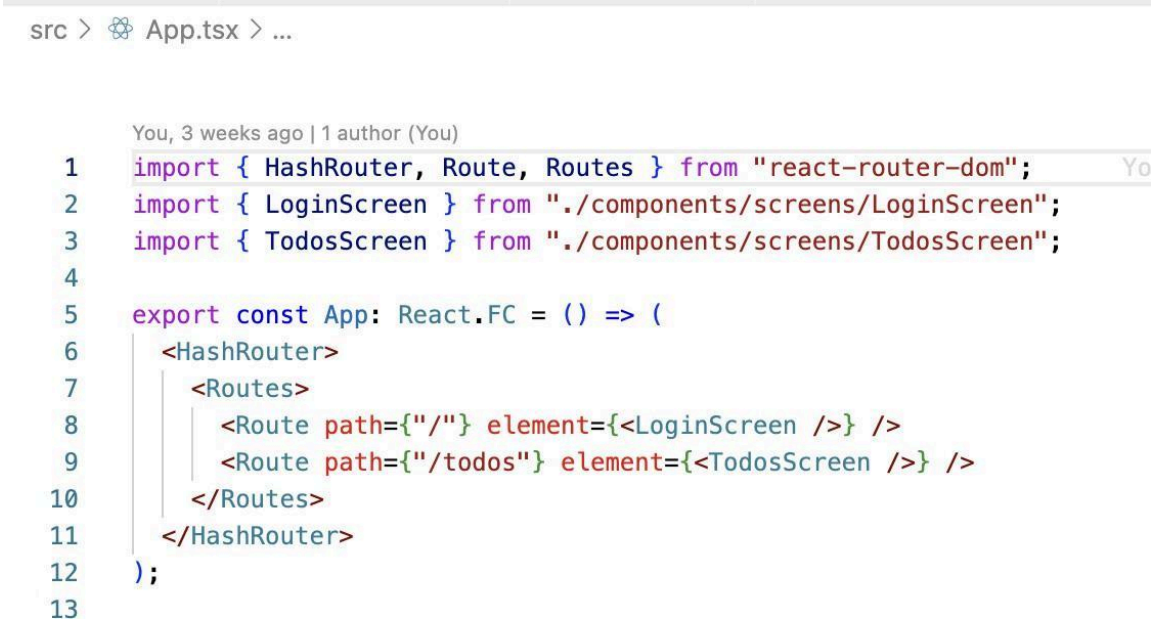


Рисунок 3.1 - лістинг компонента маршрутизації

Для здійснення комунікації між клієнтською частиною та серверним REST API на основі Express.js було розроблено спеціалізований HTTP-клієнт. Рисунку 3.2 є прикладом коду цього клієнта. Його основна функція полягає в автоматичному додаванні до заголовків кожного запиту токена авторизації формату JWT, який користувач отримує під час успішного входу в систему та який зберігається локально (наприклад, у Local Storage). Цей токен містить закодовану інформацію, зокрема унікальний ідентифікатор користувача (user ID), що відповідає первинному ключу в базі даних PostgreSQL. На серверній стороні цей ідентифікатор використовується для точної ідентифікації користувача, що ініціював запит, та для наступної перевірки прав доступу до відповідних даних, формуючи таким чином базовий механізм захисту від несанкціонованих операцій.

src > utils > TS fetchClient.ts > ...

```ts
type RequestMethod = "GET" | "POST" | "PATCH" | "DELETE";

async function request<T>(url: string, method: RequestMethod = "GET", data: any = null): Promise<T> {
  const token = getToken();
  const config: AxiosRequestConfig = {
    method,
    url: BASE_URL + url,
    headers: {},
  };
  if (data) {
    config.data = data;
    if (config?.headers) {
      config.headers["Content-Type"] = "application/json; charset=UTF-8";
    }
  }
  if (token && config.headers) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  try {
    const response = await axios(config);
    return response.data;
  } catch (error: any) {
    if (error.response) {
      if (error.response.status === 401) {
        removeToken();
        window.location.href = "/";
      }
      throw new Error(error.response.data);
    } else {
      throw new Error(error.message);
    }
  }
}
export const client = {
  get: <T>(url: string) => request<T>(url),
  post: <T>(url: string, data: any) => request<T>(url, "POST", data),
  patch: <T>(url: string, data: any) => request<T>(url, "PATCH", data),
  delete: (url: string) => request(url, "DELETE"),
};
```

Рисунок 3.2 - код клієнту роботи з API.

Для забезпечення повноцінного функціонування системи було розроблено набір інтерфейсних компонентів. До них належать компоненти для процесів авторизації та реєстрації користувачів, а також комплекс компонентів для повного життєвого циклу управління завданнями, що включає їх створення, перегляд, модифікацію та видалення. Користувач отримує можливість не лише переглядати список своїх активностей, але й динамічно оновлювати їхній статус та корегувати запланований час виконання.

Додатково, через інтегровані сервіси, система надає функціонал для отримання інтелектуальних порад, спрямованих на оптимізацію роботи з поточними чи запланованими завданнями. Крім того, реалізовано механізм аналізу історії діяльності, який дозволяє на основі останніх 10 завдань оцінити дотримання термінів, виявити тенденції та отримати конкретні рекомендації щодо підвищення особистої ефективності користувача.

src > components > LogInScreen > LogInForm.tsx > ...

```tsx
export const LogInForm: React.FC = () => {
  return (
    <form>
      <input
        type="text"
        className="todoapp__new-todo"
        placeholder="Email"
        value={email}
        onChange={(event) => setEmail(event.target.value)}
      />
      <input
        type="password"
        className="todoapp__new-todo"
        placeholder="Password"
        value={password}
        onChange={(event) => setPassword(event.target.value)}
      />
      <Footer
        signUpLoading={signUpLoading}
        signInLoading={signInLoading}
        handleSubmit={handleFormSubmit}
      />
    </form>
  );
};
```

src > components > LogInScreen > Footer.tsx > ...

You, 51 seconds ago | 1 author (You)

```tsx
import { Spin } from "antd";

type Props = {
  handleSubmit: (type: "login" | "signup") => () => void;
  signUpLoading: boolean;
  signInLoading: boolean;
};

export const Footer: React.FC<Props> = ({
  signUpLoading,
  handleSubmit,
  signInLoading
}) => (
  <footer className="todoapp__footer">
    <button
      type="button"
      className="todoapp__clear-completed"
      disabled={signUpLoading || signInLoading}
      onClick={handleSubmit("signup")}
    >
      {signUpLoading ? <Spin /> : "SIGN UP"}
    </button>
    <button
      type="submit"
      className="todoapp__clear-completed"
      disabled={signUpLoading || signInLoading}
      onClick={handleSubmit("login")}
    >
      {signInLoading ? <Spin /> : "LOG IN"}
    </button>
  </footer>
);

```

Рисунок 3.3 - Код JSX-компоненту авторизації/реєстрації

3.3 Реалізація серверної (бекенд) частини проєкту

Серверна частина створена на основі Node.js з інтеграцією TypeScript для забезпечення типобезпеки та узгодженості коду. Ядро API реалізоване з використанням фреймворку Express.js, який відповідає за обробку HTTP-запитів, автентифікацію та комунікацію з базою даних та зовнішніми сервісами (OpenAI API).

Для роботи з PostgreSQL застосовано ORM Sequelize. Визначено дві основні моделі даних: User (користувачі) та Todo (завдання). Модель Todo включає атрибути статусу, назви, опису та планованого часу виконання. Така структура забезпечує гнучкість системи для майбутнього розширення функціоналу та

```
src > models > TS User.ts > ...
11  @Table({
12    tableName: "users",
13    createdAt: true,
14    updatedAt: true,
15  })
16  export class User extends Model {
17    @PrimaryKey
18    @AutoIncrement
19    @Column({
20      type: DataTypes.INTEGER,
21    })
22    id!: number;
23
24    @AllowNull(false)
25    @Column({
26      type: DataTypes.STRING,
27    })
28    email!: string;
29
30    @AllowNull(false)
31    @Column({
32      type: DataTypes.STRING,
33    })
34    password!: string;
35  }
```

даних.

Рисунок 3.4 - реалізація моделі User

Для інтеграції з OpenAI API необхідно отримати та безпечно зберігати унікальний ключ доступу, який генерується на платформі OpenAI. Як проілюстровано на Рисунку 3.5, цей ключ зберігається у змінних середовища виконання (environment variables), що є стандартною практикою для захисту конфіденційних даних від

потрапляння в публічний вихідний код. Використання API є платним, тарифікація залежить від конкретної моделі штучного інтелекту та обсягу використаних запитів. На етапі розробки на обліковий запис було внесено стартовий баланс у розмірі 5 доларів США, а для моніторингу витрат та аналізу використання застосунком передбачено спеціальну панель управління (dashboard) в інтерфейсі OpenAI.

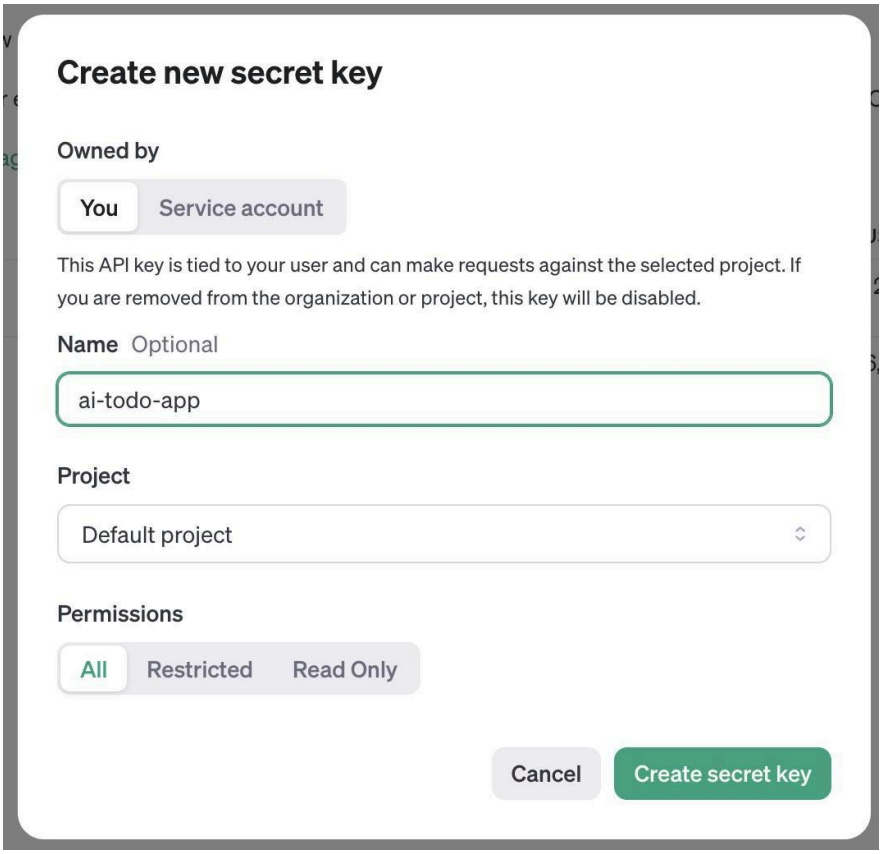


Рисунок 3.5 - модальне вікно створення API-ключа

Як показано на Рисунку 3.6, для отримання інтелектуального аналізу історії діяльності система формує структурований запит до OpenAI API. Сервер агрегує дані про останні N завдань користувача, передає їх як контекст у вибрану модель GPT та отримує у відповідь текстовий аналіз з конкретними порадами. Цей механізм дозволяє генерувати персоналізовані рекомендації щодо підвищення особистої

продуктивності, базуючись на об'єктивних даних про виконану роботу та виявлених патернах поведінки.

src > controllers > TS User.ts > analyzeLastNTasks > completion > messages > content

```
72  export const analyzeLastNTasks = async (req: Request & any, res: Response) => {
99    try {
100     const openai = new OpenAI();
101
102     const completion = await openai.chat.completions.create({
103       messages: [
104         {
105           role: "system",
106           content: baseAiContext,      You, 7 days ago • added ai analysys
107         },
108         {
109           role: "user",
110           content: JSON.stringify(tasksWithTimeSpent),
111         },
112       ],
113       model: "gpt-3.5-turbo-1106",
114       response_format: { type: "text" },
115     });
116     if (completion.choices[0].message.content) {
117       return res.status(200).json(completion.choices[0].message.content);
118     }
```

src > constants > TS index.ts > ...

```
      You, 1 second ago | 1 author (You)
1   export const baseAiContext = `
2     You are a productivity assistant that helps users with their todos.
3     Given the following todo(s), provide a tip to help the user complete it(them).
4     {
5       title: string;
6       description?: string;
7       id: number;
8       status: 'created' | 'inProgress' | 'completed';
9       createdAt: Date;
10      inProgressAt?: Date;
11      completedAt?: Date;
12      estimatedTime?: number; // in hours
13    }
14    If the todo is completed, there will be additional field for completion time (in hours).
15    If user finished it for more than he thought it would take,
16    provide a tip to help him estimate better or best practises for this type of tasks.
17  `;
```

Рисунок 3.6 - приклад запиту для аналізу останніх N завдань користувача

Безпеку серверних ресурсів забезпечує спеціально створене проміжне програмне забезпечення (middleware) для перевірки автентифікації, що проілюстровано на Рисунку 3.7. Даний механізм

обробляє кожен вхідний HTTP-запит, за винятком публічних ендпоінтів для входу та реєстрації. Його основна функція — верифікація цифрового підпису та терміну дії JWT-токена, що передається в заголовках запиту.

Оскільки токен містить інформацію про унікальний ідентифікатор користувача, система може точно визначити автора запиту. Це формує фундаментальний захист на рівні контролю доступу: користувач отримує дозвіл лише на операції з власними даними. Таким чином, цей підхід ефективно запобігає несанкціонованому доступу, зміні або видаленню інформації інших користувачів, забезпечуючи конфіденційність та цілісність даних у системі.

src > middleware > TS auth.ts > ...

You, 1 second ago | 1 author (You)

```ts
import { NextFunction, Request, Response } from "express";
import jwt from 'jsonwebtoken';
import { DecodedToken } from "../types";

export const auth = (req: Request & any, res: Response, next: NextFunction) => {
  const token = req.header('Authorization')?.split(' ')[1];
  if (!token) {
    return res.status(401).send('Access Denied');
  }

  try {
    const decodedData = jwt.verify(token, process.env.JWT_SECRET!) as DecodedToken;
    if ((decodedData.exp || 0) * 1000 < Date.now() || !decodedData.id) {
      return res.status(401).send('Session Expired');
    }
    req.userId = decodedData.id;
  } catch (err) {
    console.log(err);
    return res.status(401).send('Invalid Token');
  }

  return next();
};
```

Рисунок 3.7 - Код middleware перевірки авторизації

Отже, серверна архітектура, побудована на базі Express.js, Sequelize, інтеграції з OpenAI API та міцного механізму авторизації через middleware, формує стабільну основу для роботи застосунку. Ця підсистема комплексно вирішує ключові завдання: ефективну обробку

клієнтських запитів, надійне управління даними в PostgreSQL та безпечну взаємодію з потужними зовнішніми AI-сервісами. Такий підхід забезпечує цілісність, безпеку та функціональну повноту ToDo-системи, роблячи її зручним та продуктивним інструментом для користувача.

3.4. Демонстрація використання створеного програмного продукту

Першим екраном системи є уніфікована форма авторизації та реєстрації (рис. 3.8). Вона містить спільні поля для email та пароля, а також дві кнопки для вибору дії: «LOG IN» для входу або «SIGN UP» для створення нового акаунта. Така структура спрощує початкову взаємодію користувача з застосунком.

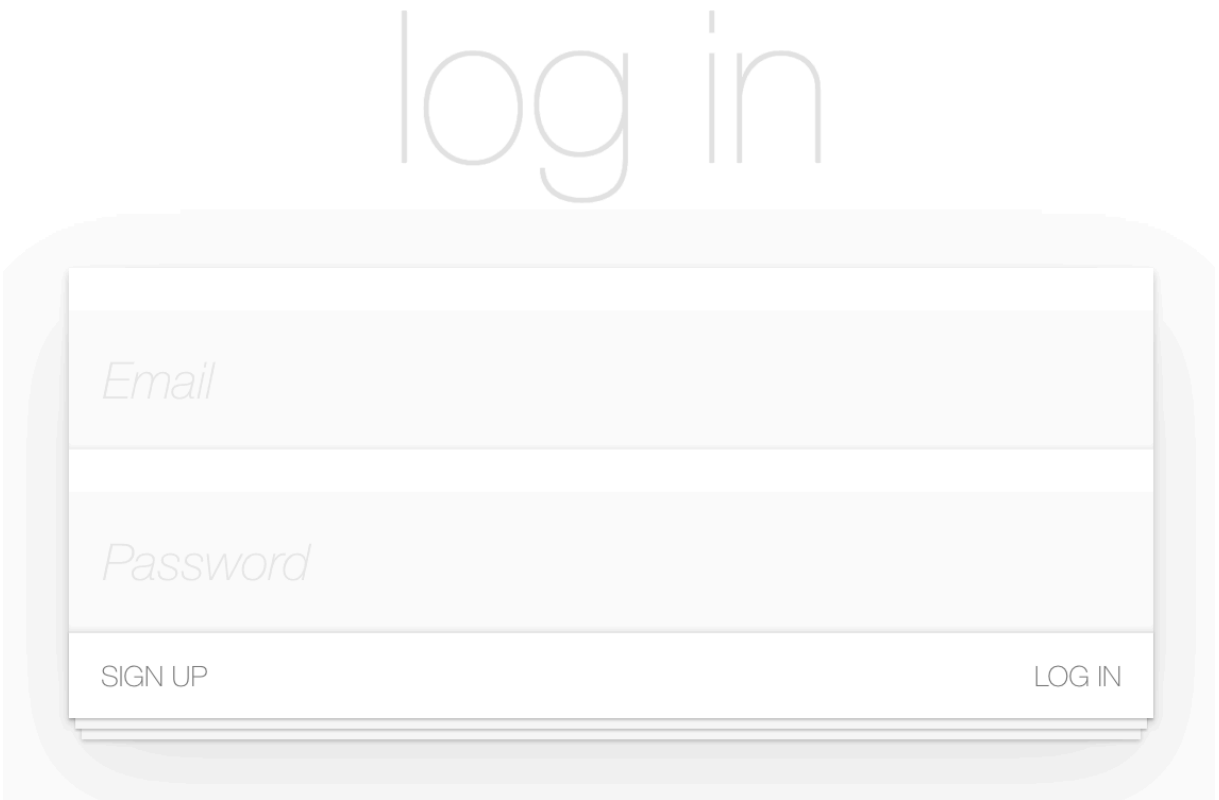


Рисунок 3.8 - Зображення екрану авторизації/реєстрації

Для інформування користувача про стан операцій, зокрема про помилки авторизації, використано компоненти сповіщень (notification) з бібліотеки Ant Design. Як показано на рисунку 3.9, система відображає відповідне повідомлення, наприклад, при спробі реєстрації з уже зайнятою адресою електронної пошти. Ці компоненти надають готові стилізовані шаблони для різних типів сповіщень (помилка, успіх, попередження тощо) та легко інтегруються в код без необхідності ручного маніпулювання DOM-елементами або написання додаткових CSS-правил.

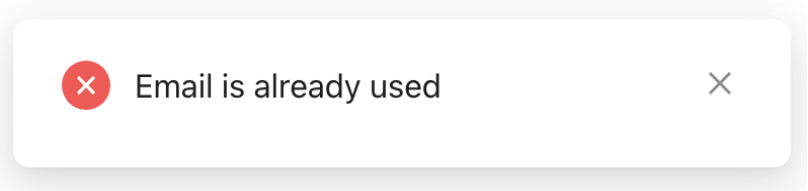
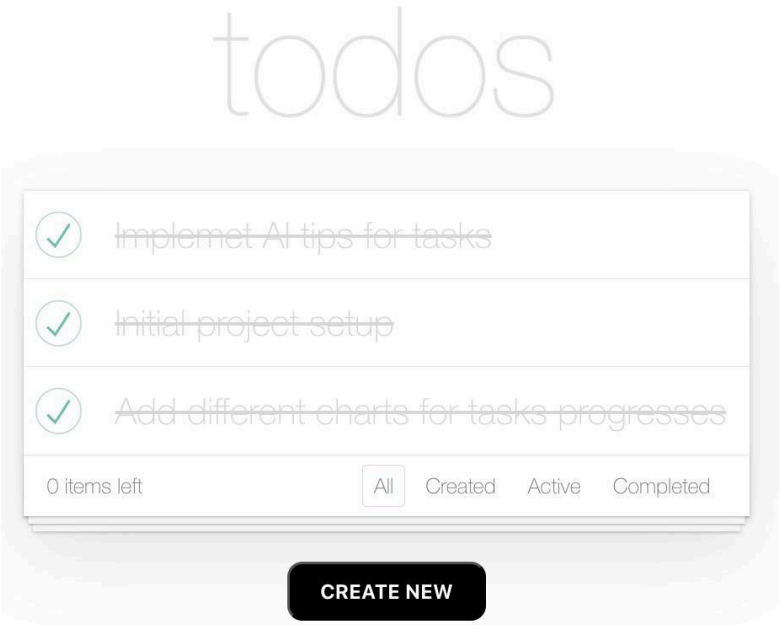


Рисунок 3.9 - Сповіщення Ant Design, використане для повідомлення про помилку при реєстрації

Після успішного завершення процесу реєстрації або авторизації (нового користувача система автоматично автентифікує) відбувається автоматичне перенаправлення на головну робочу сторінку застосунку, що проілюстровано на рисунку 3.10.



а) Перша частина роботи з завданнями

б) Друга, аналітична частина екрану

Рисунок 3.10 - Головний екран роботи з завданнями

Головна сторінка (рис. 3.10) поділена на два логічні блоки. Секція а відповідає за безпосередню роботу з завданнями: їх створення, редагування, видалення та фільтрацію за статусом.

Секція б є аналітичною панеллю. Вона відображає статистику: середню тривалість виконання, стовпчасту діаграму розподілу завдань за статусами (з категоріями: виконані, в процесі, прострочені, вчасно завершені та ті, що перевищили план). Тут також розміщені кнопки для запуску AI-аналізу останніх 10 завдань, експорту даних у JSON та виходу з системи («LOG OUT»).

Інтерактивний елемент «CREATE NEW» відкриває модальне вікно для введення даних нового завдання (рис. 3.11).

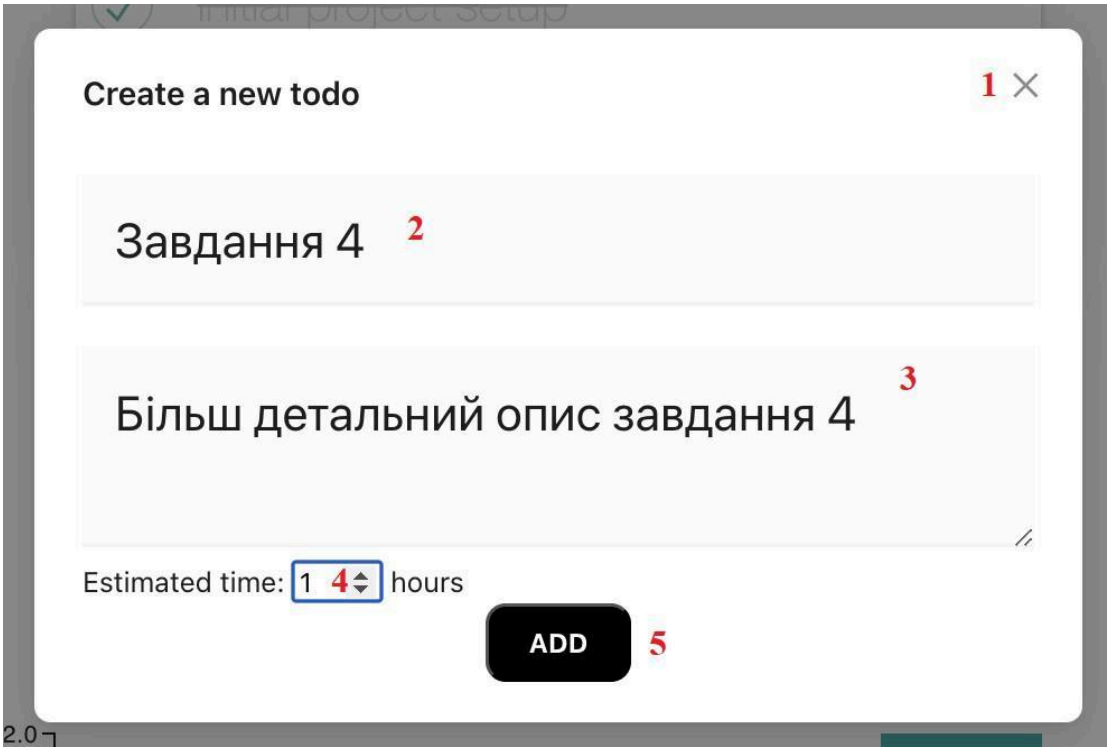


Рисунок 3.11 - Модальне вікно створення нового завдання

1 - кнопка закриття модального вікна

2 - поле введення назви завдання

3 - поле додаткового опису завдання

4 - поле для введення очікуваного часу виконання

5 - кнопка створення завдання

Після створення завдання, воно з’являється в списку зі статусом “створено” (“created”). Це зображено рисунку 3.12

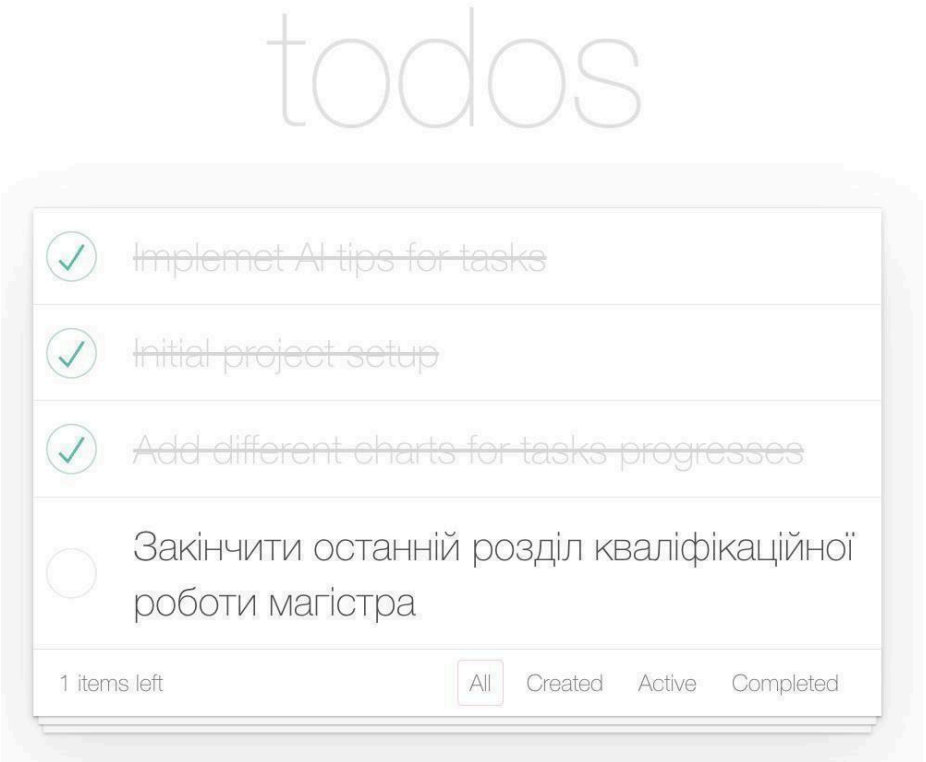


Рисунок 3.12 - Список завдань після створення нового

Для зміни існуючого завдання передбачено два способи: подвійний клік лівою кнопкою миші на відповідному рядку або натискання піктограми олівця (кнопки редагування). Обидві дії відкривають

модальне вікно для редагування запису, зображене на рисунку 3.13.

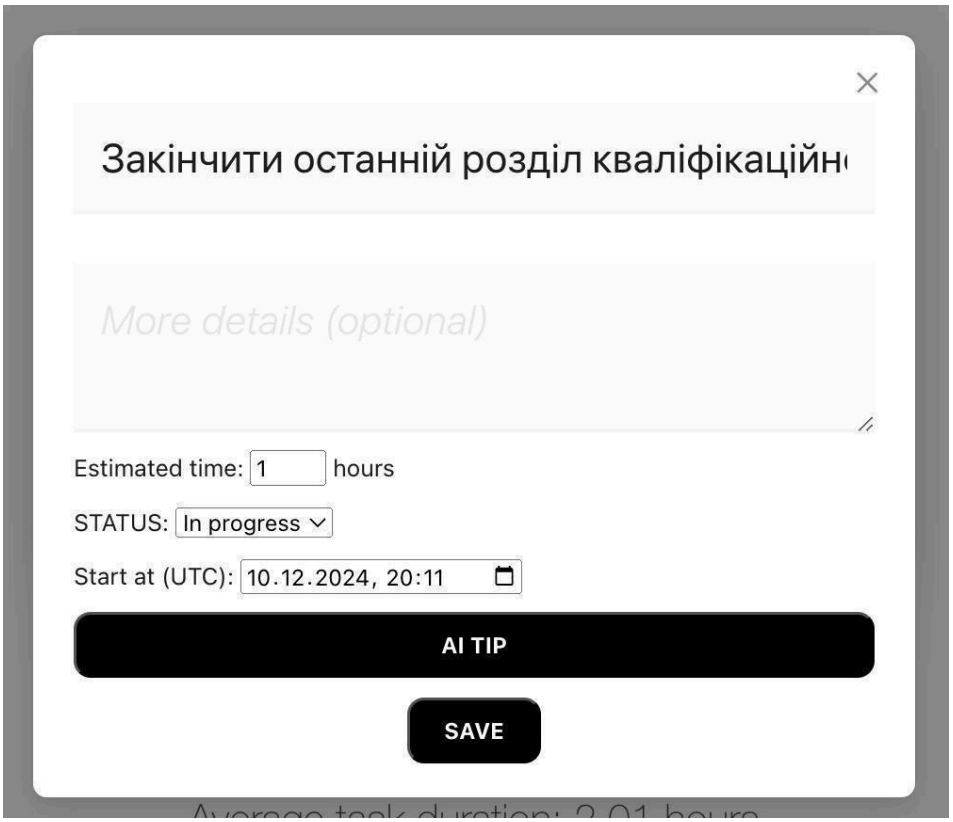


Рисунок 3.13 - Модальне вікно редагування завдання

В рамках демонстрації було оновлено статус обраного завдання на «виконується» (in progress) та встановлено час його початку на 1,5 години раніше від поточного моменту (при заданому орієнтовному часі виконання 1 година). Після збереження змін та активації відповідної функції було отримано інтелектуальну пораду від AI-модуля, результат якої представлено на рисунку 3.14.

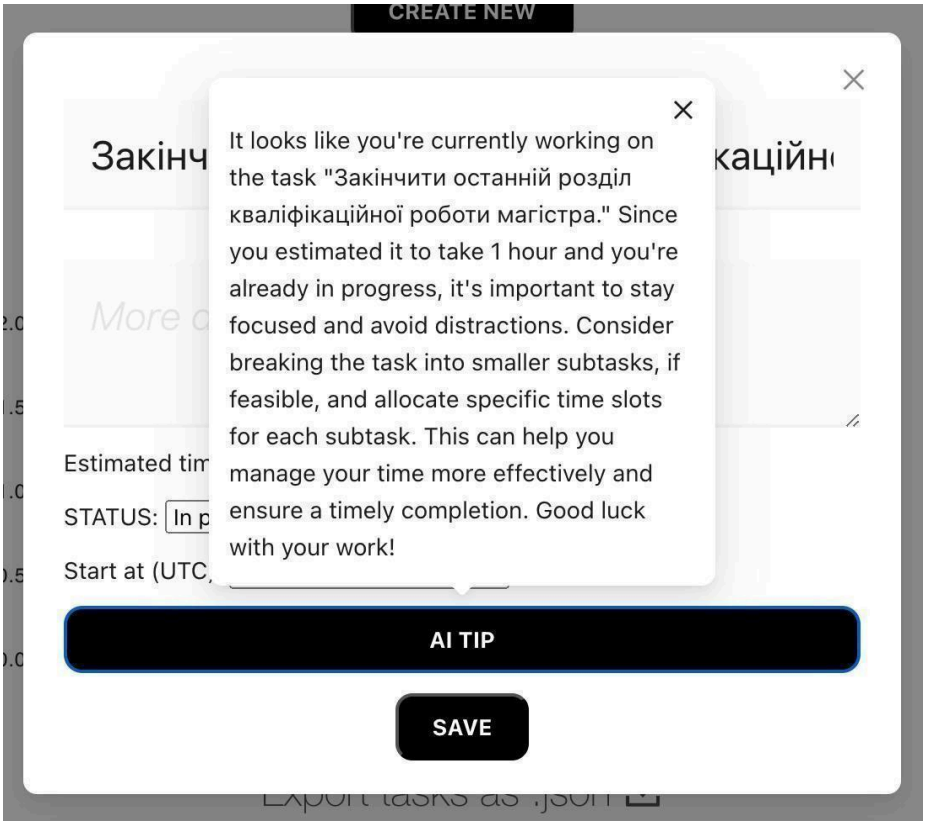


Рисунок 3.14 - Приклад поради до завдання

У даному випадку, оскільки завдання перевищило запланований час та не завершене, AI-порада акцентує на важливості декомпозиції складних цілей на менші підзавдання для точнішого оцінювання тривалості. Ця рекомендація є релевантною, особливо враховуючи відсутність детального опису, і може сприяти покращенню планування та продуктивності користувача.

Окрема функція «ШІ-аналіз» в аналітичному розділі генерує комплексну оцінку продуктивності на основі історії останніх десяти завдань. Візуалізований результат такого аналітичного звіту представлено на рисунку 3.15.

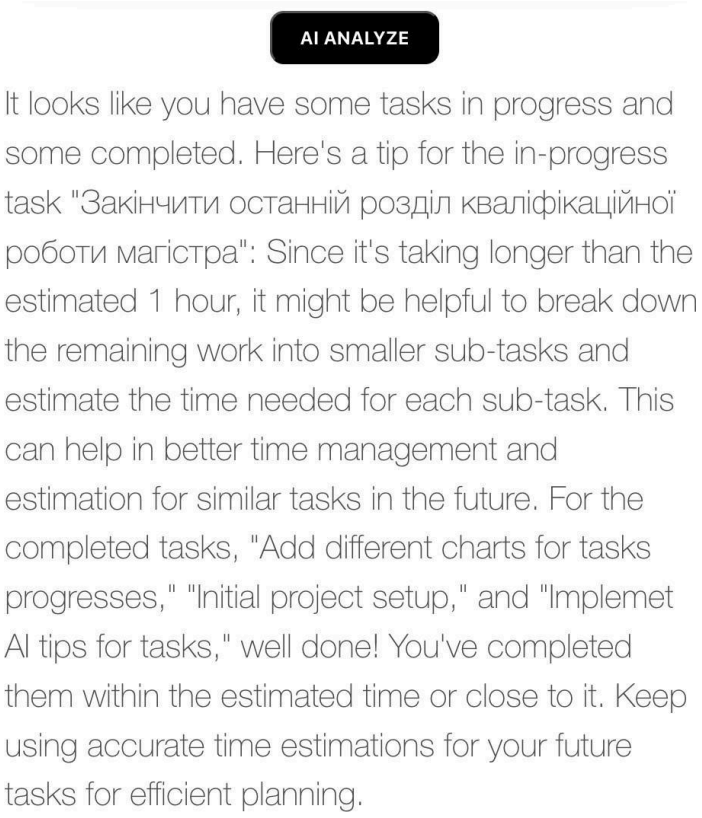


Рисунок 3.15 - Результат аналізу останніх завдань користувача

Аналіз, отриманий від системи підтверджує та деталізує початкову рекомендацію. AI знову акцентує на важливості розбиття завдань для покращення оцінки часу та зазначає проблему з дотриманням дедлайнів: з трьох завершених активностей лише одна була виконана вчасно, хоча інші дві мали незначне відставання. Такий результат демонструє здатність застосунку коректно аналізувати дані користувача та генерувати практично застосовні висновки, що підтверджує його функціональну ефективність та відповідність поставленим цілям.

ВИСНОВКИ

Метою роботи була розробка та практична реалізація архітектури безпечної інформаційної системи для е-комерції з підтримкою AI. Поставлені завдання успішно виконані.

Аналітична частина виявила ключові проблеми галузі: конфлікт безпеки та зручності, архітектурну негнучкість та роздробленість заходів захисту. Це дозволило обґрунтувати вибір гібридної архітектури з AI-компонентами.

У проектному розділі було розроблено гібридну архітектуру (мікросервіси + модульний моноліт), гнучку модель даних PostgreSQL, комплекс механізмів безпеки (JWT-аутентифікація, middleware перевірки прав, хешування паролів) та архітектуру AI-сервісу. Обґрунтовано сучасний технологічний стек.

Практична реалізація підтвердила життєздатність концепції: створено працюючий веб-застосунок з повним набором компонентів (фронтенд, бекенд, БД, AI-сервіс) та ефективними механізмами безпеки. Систему успішно розгорнуто на хостингових платформах (GitHub Pages, Render.com, Neon.tech). Інтерфейс на основі Ant Design забезпечує зручність використання.

Наукова новизна полягає в комплексному архітектурному підході, що інтегрує технічну безпеку, правову відповідність (GDPR) та інтелектуальні AI-сервіси в єдину систему[1]. Практична цінність підтверджується робочим програмним прототипом, архітектурною документацією та моделлю даних, які можуть слугувати основою для створення комерційних систем е-комерції або управління завданнями.

Таким чином, мета роботи досягнута, система є функціональною та демонструє ефективність запропонованих рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Європейський парламент, Рада Європейського Союзу. Загальний регламент про захист даних (GDPR): Регламент (ЄС) № 2016/679 від 27 квітня 2016 року про захист фізичних осіб стосовно обробки персональних даних та про вільний рух таких даних, що скасовує Директиву 95/46/ЄС. *Офіц. вісн. ЄС*. 2016. L 119. С. 1–88. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj (дата звернення: 18.12.2024).
2. Рада зі стандартів безпеки PCI (PCI SSC). *Стандарт безпеки даних індустрії платіжних карток (PCI DSS) версії 4.0* [Payment Card Industry Data Security Standard v4.0]. 2022. URL: https://docs-prv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI-DSS-v4_0.pdf (дата звернення: 18.12.2024).
3. Закон України «Про захист персональних даних» : станом на 01.01.2024 : від 01.06.2010 № 2297-VI. *Відомості Верховної Ради України (ВВР)*. 2010. № 34. Ст. 481. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 18.12.2024).
4. Ньюман С. Створення мікросервісів: проектування дрібнозернистих систем = Newman S. Building Microservices: Designing Fine-Grained Systems / пер. з англ. 2-ге вид. Київ : ООО «Віва-Стар», 2021. 650 с.
5. Річардс М. Шаблони архітектури програмного забезпечення: розуміння поширених шаблонів архітектури та коли їх використовувати = Richards M. Software Architecture Patterns: Understanding Common Architecture Patterns and When to Use Them / пер. з англ. Київ : ООО «Віва-Стар», 2015. 54 с.
6. Фрімен А. Pro React 16 = Freeman A. Pro React 16. Берлін ; Нью-Йорк : Apress, 2019. 748 с. ISBN 978-1-4842-4450-0.
7. Стефанов С. React: запуск та робота: створення вебзастосунків = Stefanov S. React: Up & Running: Building Web Applications / пер. з англ. 2-ге вид. Кембридж : O'Reilly Media, 2020. 304 с. ISBN 978-1-492-05405-5.
8. Вілсон Дж., Етерінгтон Т. Веб-розробка на Node.js: серверна веб-розробка стала простою за допомогою Node 14 з використанням практичних прикладів = Wilson J., Etherington T. Node.js Web Development: Server-side web development made easy with Node 14 using practical examples / пер. з англ. 5-те вид. Бірмінгем : Packt Publishing, 2020. 664 с. ISBN 978-1-838-82957-8.

9. Бондаренко Д.(2025) Інформаційно-комунікаційні системи та їх захист в електронній комерції: архітектурні рішення та нормативно-правові аспекти. "Інформаційні технології" (ISSN: 2664-2638), 9(9), 39–40. https://zcit.kubg.edu.ua/index.php/journal/article/view/27
10. Алексіна, Л. Т., & Бондарчук, А. П. (2024). Оптимізація гіперпараметрів для машинного навчання. Зв’язок, (2), 18-22.
11. Bondarchuk, A., Dibrivniy, O., Grebenyk, V., & Onyshchenko, V. (2021, October). Motion Vector Search Algorithm for Motion Compensation in Video Encoding. In 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T) (pp. 345-348). IEEE.
12. Сторчак, К. П., Тушич, А. М., & Бондарчук, А. П. (2018). Кластерний аналіз даних із використанням штучних нейронних мереж. Зв'язок, (6), 36-38.
13. Shantyr, A., Zinchenko, O., Storchak, K., Bondarchuk, A., & Pepa, Y. (2025). Prediction of quality software quality indicators with applied modifications of integrated gradiates methods. Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska, 15(2), 139-146.
14. Вембер, В. П., Машкіна, І. В., Носенко, Т. І., & Яскевич, В. О. (2025). Можливості та виклики використання штучного інтелекту у навчанні фахових дисциплін студентів спеціальностей «Комп’ютерні науки» та «Інженерія програмного забезпечення». Open educational e-environment of modern University, (19), 1-16.
15. Тушич, А. М., Сторчак, К. П., & Бондарчук, А. П. (2019). Вимоги до інтелектуальних систем аналізу даних та їх класифікацій. Телекомунікаційні та інформаційні технології, (1), 31-36.
16. Бондарчук, А., Жебка, В., Корецька, В., & Шилкіна, А. (2024). Порівняльна характеристика web-орієнтованих інструментів автоматизації освітнього процесу в умовах цифрової трансформації. Публічно-управлінські та цифрові практики, (1), 13-21.
17. Бондарчук, А. П., Корнага, Я. І., Базалій, М. Ю., Сергієнко, П. А., & Ільїн, О. Ю. (2020). Метод захисту програмного коду від аналізу засобами обфускації. Телекомунікаційні та інформаційні технології, (4), 140-148.
18. Співак, С., Бондарчук, А., & Черевик, О. (2025). AI-система для професійної орієнтації у сфері 3D-графіки. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 3(31), 698-709.
19. Співак, С. М., Білоус, В. В., Горбатовський, Д. В., & Бондарчук, А. П. (2025). Adapting education to the 3D graphics market using

AI. Телекомунікаційні та інформаційні технології, 89(4), 215-221.
20. Бондарчук, А. П., & Глушак, О. М. (2025). Предиктивне управління оновленнями програмного забезпечення в інтернеті речей. Зв'язок, (5), 13-17.
21. Рзаєва, Світлана Леонідівна та Складанний, Павло Миколайович та Машкіна, Ірина Вікторівна та Костюк, Юлія Володимирівна (2025) Модель реалізації керування доступом на основі ролей (RBAC) у багаторівневій архітектурі сховища даних Сучасний захист інформації, 63 (3). с. 137-149. ISSN 2409-7292 https://doi.org/10.31673/2409-7292.2025.031671
22. Співак, Світлана Михайлівна та Машкіна, Ірина Вікторівна та Носенко, Тетяна Іванівна та Білоус, Владислав Володимирович та Глушак, Оксана Михайлівна (2025) Оптимізація customer support за допомогою аі чат-ботів: практичний кейс Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 4 (28). с. 727-739. ISSN 2663-4023 https://doi.org/10.28925/2663-4023.2025.28.838
23. Skladannyi, Pavlo та Mashkina, Iryna та Rzaeva, Svitlana та Kostiuk, Yuliia (2025) Methods of GDPR for Ensuring Data Storage Security Against Leaks and Threats Телекомунікаційні та інформаційні технології, 87 (2). с. 59-76. ISSN 2412-4338 https://doi.org/10.31673/2412-4338.2025.027860
24. Marisyk, S. та Matselyuk, Y. та Charny, D. та Zabulonov, Y. та Nosenko, Tetiana та Pugach, O. та Rudoman, M. (2024) Application of the Latest Design of Combined Adsorber-Settler Structure in the Purification (Deactivation) of Liquid Radioactive Wastes (LRW) Книжкова серія. с. 137-145. ISSN 2366-2557
25. Тетяна Носенко, Ірина Машкіна (2025) Моделювання процесів обробки екологічних даних для систем мобільного моніторингу на основі БПЛА та методу IDW Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», No 2 (30), 2025 с.110-122. Спеціальний випуск ISSN 2663 – 4023 https://doi.org/10.28925/2663-4023.2025.30.955
26. Замрій, І.В. та Шахматов, І.О. та Яскевич, Владислав Олександрович (2024) BlockchainSQLSecure: інтеграція блокчейн-технології для зміцнення захисту від SQL ін'єкцій Вісник Київського національного університету імені Тараса Шевченка. Фізико-математичні науки (1(78)). с. 160-168. ISSN 2218-2055 https://doi.org/10.17721/1812-5409.2024/1.29
27. Чемерис, О., Бушма, О., Литвин, О., Білоцерковський, А., Лукашевич, П. (2021). Мережа автономних пристроїв для надійного моніторингу складних технологічних об'єктів. У: ван Гулійк, К., Зайцева, Е. (ред.) Інженерія надійності та обчислювальний інтелект. Дослідження з

обчислювального інтелекту, т. 976. Springer, Cham. https://doi.org/10.1007/978-3-030-74556-1_16.
28. Бушма, Олександр Володимирович та Турукало, Андрій Валерійович (2022) Оцінка параметрів програмної реалізації шкального відображення даних Cybersecurity: Education, Science, Technique, 4 (16). с. 142-158. ISSN 2663-4023 https://doi.org/10.28925/2663-4023
29. Бушма, Олександр Володимирович та Турукало, Андрій Валерійович (2021) Багатоелементні шкальні індикаторні пристрої у вбудованих системах Кібербезпека: освіта, наука, техніка, 3 (11). с. 43-60. ISSN 2663-4023 https://doi.org/10.28925/2663-4023
30. Бушма, Олександр Володимирович та Абрамов, Вадим Олексійович (2022) Підвищення достовірності визначення концентрації газів в середовищі моніторингу In: III Міжнародна науково-практична конференція: "Інформаційні технології та цифрова економіка" III International scientific and practical conference: "Information technologies and digital economy", 19-20 april 2022, Kyiv. https://elibrary.kubg.edu.ua/id/eprint/41648/
31. Білоус, Владислав та Бодненко, Дмитро та Локазюк, Олександра та Складаний, Павло та Абрамов, Вадим (2025) Програмне забезпечення для кібердоказів як інструмент цифрової криміналістики у розслідуванні кіберзлочинів. Забезпечення кібербезпеки в інформаційно-телекомунікаційних системах. 2025 (3991). С. 26-37. ISSN 1613-0073 https://ceur-ws.org/Vol-3991/
32. Білоус, Владислав Володимирович та Бодненко, Дмитро Миколайович та Хохлов, Олексій Костянтинович та Локазюк, Олександра Вікторівна та Стаднік, Ірина Петрівна (2024) Open Source Intelligence for War Crime Documentation Workshop Cybersecurity Providing in Information and Telecommunication Systems (CPITS 2024) (3654). с. 368-375. ISSN 1613-0073 : https://ceur-ws.org/Vol-3654/short3.pdf
33. Андрій Петрович Бондарчук, Ірина Юріївна Мельник, Євгеній Іванович Суханевич, Вадим Олексійович Абрамов. Підвищення ефективності систем відеоспостереження за рахунок гібридного методу відбору ключових кадрів і інтерпретації рішень. Телекомунікаційні та інформаційні технології. 2025 , №3 с101-105 https://doi.org/10.31673/2412-4338.2025.038710
34. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., Hlushak, O., Zhyltsov, O., Ilich, L., Kobets, N., Kovaliuk, T., Kuchakovska, H., Lytvyn, O., Lytvyn, P., Mashkina, I., Morze, N., Nosenko, T., Proshkin, V., Radchenko, S., … Yaskevych, V. (2021). Theoretical and

practical aspects of the use of mathematical methods and information technology in education and science. https://doi.org/10.28925/9720213284km.

35. Бушма, Олександр Володимирович та Машкіна, Ірина Вікторівна та Носенко, Тетяна Іванівна та Яскевич, Владислав Олександрович (2024) Кваліфікаційна робота магістра: Навчально-методичний посібник для спеціальності «Комп'ютерні науки» Київський столичний університет імені Бориса Грінченка, Україна. https://elibrary.kubg.edu.ua/id/eprint/50205/

ДОДАТОК А

Перелік скорочень, символів і спеціальних термінів:

- **AI (Artificial Intelligence)** – Штучний інтелект. Галузь комп'ютерних наук, що займається створенням машин, здатних виконувати завдання, які зазвичай вимагають людського інтелекту.
- **API (Application Programming Interface)** – Програмний інтерфейс додатку. Набір визначень та протоколів для взаємодії різних програмних компонентів.
- **CRUD (Create, Read, Update, Delete)** – Базові операції з даними: створення, читання, оновлення, видалення.
- **DDoS (Distributed Denial of Service)** – Розподілена атака типу "відмова в обслуговуванні". Спроба зробити ресурс недоступним через надмірне навантаження.
- **GDPR (General Data Protection Regulation)** – Загальний регламент щодо захисту даних. Основне законодавство ЄС про захист персональних даних.
- **JWT (JSON Web Token)** – Веб-токен у форматі JSON. Стандарт для безпечної передачі інформації між сторонами у вигляді JSON-об'єкта.
- **ORM (Object-Relational Mapping)** – Об'єктно-реляційне відображення. Технологія програмування, яка дозволяє конвертувати дані між несумісними типами систем в об'єктно-орієнтованих мовах програмування.
- **PCI DSS (Payment Card Industry Data Security Standard)** – Стандарт безпеки даних індустрії платіжних карток. Набір вимог для забезпечення безпеки операцій з платіжними картками.
- **REST (Representational State Transfer)** – Стиль архітектури програмного забезпечення для розподілених систем, що використовує HTTP протокол.
- **SPA (Single Page Application)** – Односторінковий додаток. Веб-додаток, який завантажує одну сторінку HTML і динамічно оновлює її в міру взаємодії користувача.
- **ATO – Account Takeover** (захоплення облікового запису), коли зловмисник отримує несанкціонований доступ до вашого онлайн-акаунту, використовуючи ваші облікові дані. Після злому зловмисники можуть змінювати ваші дані, здійснювати шахрайські

покупки, викрадати фінансову інформацію або використовувати вашу особу для подальших злочинів.