НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення бази посилань на робочі ресурси працівників підприємства»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент	_____ (підпис)	**Ганна ВАЙГАНГ**
Керівник бакалаврської кваліфікаційної роботи к.е.н., ст.викладач	_____ (підпис)	**Дмитро НІКОЛАЄНКО**
Виконав	_____ (підпис)	**Ігор ХОДАКІВСЬКИЙ**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Ходаківському Ігорю Євгенійовичу

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення бази посилань на робочі ресурси працівників підприємства»

затверджена наказом від «19» грудня 2025 р. № 3204 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): веб-система управління корпоративними посиланнями з рольовим доступом, інтелектуальним сортуванням ресурсів, модулем IT-заявок та аналітикою. Стек технологій: Node.js, Express.js, MongoDB Atlas, JavaScript, HTML5, CSS3. Розгортання на хмарному хостингу Render.com.
Перелік питань, що підлягають дослідженню:
1. Аналіз предметної області управління корпоративними ресурсами.
2. Проєктування логічної та фізичної моделі бази даних.
3. Розробка серверної частини (REST API) та клієнтського SPA-інтерфейсу.
4. Реалізація алгоритму інтелектуального сортування та адаптивного доступу.
5. Тестування, розгортання та документування системи.
Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Дмитро НІКОЛАЄНКО**

Завдання взяв до виконання _____ (підпис) **Ігор ХОДАКІВСЬКИЙ**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – програмний інтерфейс додатку, набір визначених правил та специфікацій, за допомогою яких програми взаємодіють одна з одною

BSON (Binary JSON) – бінарний формат серіалізації даних, що використовується для зберігання документів у MongoDB

CDN (Content Delivery Network) – мережа доставки контенту, розподілена система серверів для швидкої передачі веб-ресурсів

CORS (Cross-Origin Resource Sharing) – механізм, що дозволяє веб-сторінкам робити запити до інших доменів

CRUD (Create, Read, Update, Delete) – чотири базові операції для роботи з даними у базі даних

CSS (Cascading Style Sheets) – каскадні таблиці стилів, мова опису візуального оформлення веб-сторінок

DOM (Document Object Model) – об'єктна модель документа, інтерфейс для програмного доступу до HTML-документів

HTML (HyperText Markup Language) – мова розмітки гіпертексту, стандартна мова для створення веб-сторінок

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту, основний протокол обміну даними у мережі Інтернет

JSON (JavaScript Object Notation) – текстовий формат обміну даними, заснований на підмножині мови JavaScript

MVC (Model-View-Controller) – архітектурний шаблон проєктування, що розділяє додаток на три компоненти

NoSQL (Not Only SQL) – клас систем управління базами даних, що не використовують реляційну модель

ODM (Object-Document Mapping) – технологія відображення об'єктів програмного коду на документи бази даних

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленої системи

SPA (Single Page Application) – односторінковий додаток, веб-додаток, що завантажує одну HTML-сторінку

SVG (Scalable Vector Graphics) – формат масштабованої векторної графіки для відображення на веб-сторінках

URL (Uniform Resource Locator) – уніфікований покажчик ресурсу, стандартизована адреса ресурсу в Інтернеті

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

ВСТУП

Будь-яка компанія – велика чи маленька – сьогодні використовує купу різних цифрових інструментів: корпоративна пошта, хмарні диски, системи документообігу, CRM, платформи для управління задачами, бухгалтерські програми, месенджери і ще десятки інших сервісів. І з кожним роком цей список тільки росте – відповідно, стає все складніше просто знати, де що знаходиться і хто до чого має доступ.

Бухгалтер і програміст працюють з абсолютно різними інструментами, і це нормально. Але коли у компанії немає єдиного місця, де зберігаються всі ці посилання, виникають типові проблеми: новий працівник тиждень витрачає на те, щоб дізнатися, де що знаходиться; хтось досі використовує застарілу адресу сервісу; в різних відділах для однієї і тієї ж задачі використовуються різні інструменти; а керівництво взагалі не розуміє, якими системами люди реально користуються.

Ось чому виникла ідея цієї роботи – створити систему, яка централізовано зберігає всі робочі посилання, сама розуміє, кому що показувати, і при цьому з часом "навчається" виводити на перший план те, чим ти найчастіше користуєшся.

Об'єктом дослідження є процес організації та управління доступом працівників підприємства до корпоративних інформаційних ресурсів.

Предметом дослідження є методи та програмні засоби побудови веб-системи управління базою посилань на робочі ресурси з адаптивним розмежуванням доступу та інтелектуальним сортуванням.

Метою бакалаврської кваліфікаційної роботи є розробка та реалізація веб-системи управління посиланнями на робочі ресурси працівників підприємства з використанням хмарних технологій, яка забезпечує рольовий доступ, персоналізовані рекомендації та інструменти для різних підрозділів організації.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати предметну область корпоративного управління ресурсами та дослідити існуючі аналогічні рішення;

– розробити модель предметної області та сформулювати вимоги до програмної системи;

– спроєктувати архітектуру системи та структуру бази даних;

– обґрунтувати вибір технологій та інструментарію розробки;

– реалізувати серверну та клієнтську частини програмного продукту;

– впровадити механізм адаптивного доступу на основі ролей працівників;

– реалізувати алгоритм інтелектуального сортування ресурсів;

– провести тестування та розгорнути систему на хмарному хостингу.

Наукова новизна роботи полягає у запропонованому підході до організації корпоративних ресурсів, що поєднує два методи: адаптивне формування доступу на основі посади та відділу працівника, а також інтелектуальне ранжування ресурсів на основі агрегації даних про частоту їх використання як окремим працівником, так і підрозділом в цілому.

Практична цінність роботи полягає у створенні готового до використання програмного продукту, який може бути впроваджений на будь-якому підприємстві для централізованого управління робочими ресурсами. Система розгорнута на хмарній платформі та доступна через мережу Інтернет з будь-якого пристрою.

Апробація результатів. Розроблена система CorpLinks розгорнута на хмарному сервері Render.com та доступна для тестування за адресою, що надається у відповідному розділі роботи. Вихідний код проєкту розміщено у публічному репозиторії GitHub.

Методологічною основою дослідження є системний підхід до аналізу та проєктування програмних систем, а також принципи об'єктно-орієнтованого програмування та проєктування баз даних. У роботі застосовано методи структурного аналізу для декомпозиції предметної області, методи

порівняльного аналізу для оцінки існуючих рішень, а також методи функціонального тестування для верифікації розробленої системи. [1]

Інформаційною базою дослідження є науково-технічна література з питань проєктування програмних систем, документація технологій Node.js, Express.js та MongoDB, а також матеріали конференцій та спеціалізованих видань у галузі веб-розробки. Додатковим джерелом інформації є офіційна документація хмарних платформ Render.com та MongoDB Atlas.

Дипломна робота виконана на кафедрі комп'ютерних наук факультету інформаційних технологій Національного університету біоресурсів і природокористування України. У процесі виконання роботи було використано сучасні інструменти розробки програмного забезпечення: текстовий редактор Visual Studio Code, система контролю версій Git, платформа GitHub для зберігання коду, а також інструменти розробника Google Chrome для тестування та відладки клієнтської частини додатку.

Структура та обсяг роботи. Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Перший розділ присвячено аналізу предметної області та огляду існуючих рішень. Другий розділ описує проєктування інформаційного забезпечення системи. Третій розділ містить опис розробки програмного забезпечення. Четвертий розділ включає рекомендації щодо впровадження та результати тестування системи.

1 УПРАВЛІННЯ КОРПОРАТИВНИМИ РЕСУРСАМИ ЯК ПРЕДМЕТНА ОБЛАСТЬ

1.1 Аналіз процесів управління ресурсами на підприємстві

Важко уявити сучасне підприємство, яке обходиться без інформаційних технологій. Від рядового менеджера до технічного директора – кожен щодня відкриває десятки різних систем: хтось перевіряє пошту, хтось завантажує звіти, хтось залогується в CRM або репозиторій коду. Всі ці інструменти разом і складають те, що можна назвати цифровим робочим середовищем компанії [2]. І від того, наскільки зручно це середовище організоване, напряму залежить ефективність роботи.

Під терміном "робочий ресурс" у цій роботі розуміється будь-який онлайн-інструмент, до якого заходять через браузер за конкретною адресою. Сюди входять і таск-менеджери на кшталт Trello чи Jira, і репозиторії коду типу GitHub, і корпоративні портали, і банківські кабінети, і системи кадрового обліку – словом, все, що має свій URL і що люди регулярно відкривають у роботі.

Якщо поглянути на середню компанію, то стає очевидно: айтівці, бухгалтери і HR-менеджери живуть у зовсім різних цифрових світах. IT-відділ сидить у репозиторіях коду, трекерах задач і системах моніторингу. Бухгалтерія – в банківських кабінетах, податкових порталах і програмах обліку. Продажники щодня відкривають CRM і таблиці з клієнтами. А HR постійно заходить на джоб-сайти, в системи обліку часу і внутрішні документи з регламентами.

Водночас існує набір ресурсів, доступ до яких повинні мати всі працівники незалежно від підрозділу: корпоративна пошта, внутрішній портал новин, система управління задачами тощо. Ця гетерогенність створює складний ландшафт доступу, який потребує систематизації та автоматизації.

Відсутність централізованої системи проявляється у трьох основних проблемах. Перша – втрата часу: люди витрачають хвилини, а то й десятки хвилин щодня тільки на те, щоб знайти потрібне посилання. Особливо боляче це відчувається в перший тиждень роботи нового співробітника – він просто не знає, де що є. Друга проблема – безпека: якщо немає чіткого розмежування доступу, хтось може зайти туди, куди не треба, просто тому що посилання десь зберігається у загальному чаті. Третя – відсутність аналітики: ніхто в компанії не знає, якими інструментами реально користуються, а які просто займають місце у закладках.

Процес управління ресурсами на підприємстві включає кілька ключових етапів. На початковому етапі відбувається реєстрація нового ресурсу у системі із зазначенням його категорії, опису та рівня доступу. Далі адміністратор визначає, які підрозділи або конкретні працівники повинні мати доступ до цього ресурсу. У процесі експлуатації система збирає статистику використання, на основі якої формуються рекомендації та звіти для керівництва. Періодично здійснюється ревізія реєстру ресурсів з метою видалення застарілих та додавання нових.

Окремо слід зазначити роль IT-служби підприємства у контексті управління ресурсами. Саме IT-підрозділ відповідає за технічну підтримку працівників, вирішення проблем із доступом до корпоративних систем та забезпечення працездатності інфраструктури. Тому невід'ємною частиною системи управління ресурсами є модуль для обробки заявок на технічну підтримку, який дозволяє працівникам повідомляти про проблеми, а IT-фахівцям – оперативно на них реагувати.

Організаційна структура типового підприємства передбачає наявність кількох ключових підрозділів, кожен з яких виконує специфічні функції та потребує доступу до відповідних інформаційних ресурсів. Відділ інформаційних технологій забезпечує функціонування корпоративної інфраструктури, підтримку програмного забезпечення та обслуговування апаратних засобів. Фінансовий відділ відповідає за бухгалтерський облік,

формування звітності та роботу з банківськими системами. Відділ кадрів займається управлінням персоналом, веденням кадрового діловодства та забезпеченням відповідності трудовому законодавству.

Аналіз потоків інформації на підприємстві демонструє наявність як вертикальних, так і горизонтальних зв'язків між підрозділами. Вертикальні потоки забезпечують передачу управлінської інформації від керівництва до виконавців та звітної інформації у зворотному напрямку. Горизонтальні потоки реалізують обмін даними між підрозділами одного рівня ієрархії, наприклад, між бухгалтерією та відділом кадрів при нарахуванні заробітної плати. Ці потоки визначають вимоги до системи управління ресурсами.

Класифікація робочих ресурсів підприємства може бути здійснена за кількома критеріями. За функціональним призначенням ресурси поділяються на інструменти розробки, засоби проєктування, управлінські платформи, фінансові системи та кадрові портали. За рівнем конфіденційності ресурси класифікуються як загальнодоступні, обмеженого доступу та конфіденційні. За типом взаємодії ресурси можуть бути інформаційними, операційними та аналітичними.

Особливу увагу слід приділити проблемі онбордингу нових працівників. Новий співробітник витрачає значну частину першого тижня роботи на ознайомлення з корпоративними системами та пошук необхідних ресурсів. Централізована система управління посиланнями з адаптивним інтерфейсом може суттєво скоротити цей період та підвищити продуктивність нового співробітника з перших днів роботи.

Ще одним важливим аспектом предметної області є необхідність забезпечення аудиту дій працівників у контексті використання корпоративних ресурсів. Журнал аудиту фіксує хто, коли та які дії виконував у системі, що є важливим елементом корпоративної безпеки. Аналіз журналу дозволяє виявляти нетипову активність та контролювати дотримання політик безпеки.

Процес взаємодії з IT-службою підприємства є невіддільною частиною управління ресурсами. Працівники регулярно стикаються з технічними

проблемами при роботі з корпоративними системами: забуті паролі, відмова обладнання, помилки програмного забезпечення. Ефективна система обробки таких заявок повинна забезпечувати зручне створення звернень, класифікацію за категоріями та пріоритетами, призначення відповідального виконавця та відстеження статусу виконання.

Окремо слід розглянути питання інтеграції системи управління ресурсами з іншими корпоративними інформаційними системами. Сучасне підприємство може використовувати десятки різних програмних продуктів, які часто працюють ізольовано. Система управління посиланнями може виступати своєрідним хабом, який об'єднує всі ці інструменти через єдиний інтерфейс.

Дослідження показують, що ефективність роботи працівників із цифровими інструментами безпосередньо залежить від зручності доступу до них. Працівники витрачають у середньому значну частину робочого часу на пошук інформації та необхідних інструментів. Впровадження централізованої системи управління ресурсами дозволяє скоротити цей час, забезпечуючи працівникам миттєвий доступ до релевантних ресурсів.

Аналіз конкурентного середовища свідчить про зростаючий попит на рішення у сфері Digital Workplace – цифрового робочого простору. Ринок платформ для цифрового робочого простору зростає щорічно, що підтверджує актуальність розробки програмного забезпечення для управління корпоративними ресурсами та вказує на значний потенціал для впровадження подібних систем.

У контексті сучасних тенденцій розвитку інформаційних технологій необхідно враховувати перехід більшості корпоративних систем у хмарне середовище. Це означає, що система управління ресурсами також повинна бути доступна з будь-якого пристрою та з будь-якої локації. Віддалена робота посилює потребу у централізованому доступі до робочих інструментів через веб-інтерфейс.

Питання інформаційної безпеки при управлінні корпоративними ресурсами потребує окремого розгляду. Кожен робочий ресурс потенційно містить конфіденційну інформацію або надає доступ до критично важливих систем підприємства. Тому система управління ресурсами повинна забезпечувати кілька рівнів захисту: автентифікацію, авторизацію на рівні ролей, фільтрацію ресурсів за підрозділами та журналювання всіх дій.

1.2 Моделювання предметної області

Для формалізації предметної області управління корпоративними ресурсами було застосовано методологію об'єктно-орієнтованого аналізу з використанням нотації UML (Unified Modeling Language). Моделювання дозволяє визначити основних акторів системи, їхні взаємодії та бізнес-процеси, які повинна підтримувати розроблювана програмна система.

У системі управління ресурсами підприємства визначено чотири основні ролі користувачів, які відрізняються набором доступних функцій. Адміністратор має повний контроль над системою: управління користувачами, ресурсами, перегляд аналітики та журналу аудиту. Менеджер (керівник підрозділу) може додавати та редагувати ресурси, переглядати статистику свого підрозділу. Співробітник має доступ до ресурсів свого підрозділу та загальних ресурсів, може створювати IT-заявки та додавати ресурси до персонального списку обраних. Стажер має обмежений доступ лише для перегляду ресурсів.

На рис. 1.1 представлено діаграму використання системи CorpLinks, яка демонструє розподіл функцій між чотирма ролями користувачів. Як видно з діаграми, кожна наступна роль успадковує можливості попередньої та додає власні: стажер має лише базовий перегляд, співробітник може створювати заявки та обирати ресурси, менеджер отримує права на редагування, а адміністратор має повний контроль над системою.

Рис. 1.1 Діаграма використання системи CorpLinks

Бізнес-процес роботи з системою починається з авторизації користувача. Після успішного входу система визначає роль та підрозділ працівника і формує адаптивний інтерфейс: у бічній панелі відображаються лише ті робочі простори, до яких працівник має доступ відповідно до свого підрозділу. Реєстр ресурсів фільтрується таким чином, що працівник бачить лише ті записи, поле доступу яких містить його підрозділ або позначку "ALL" (доступно всім).

На рис. 1.2 зображено послідовність кроків процесу авторизації у системі. Після введення облікових даних система послідовно перевіряє наявність email у базі та відповідність пароля. У разі успішної перевірки зберігається сесія користувача, фіксується подія входу в журналі аудиту та формується адаптивний інтерфейс відповідно до ролі та підрозділу працівника.

Рис. 1.2 Діаграма активності процесу авторизації

Для моделювання процесу обробки IT-заявок було розроблено діаграму станів, яка описує життєвий цикл заявки. Кожна заявка проходить три стани: "Нова" (щойно створена), "В роботі" (прийнята IT-фахівцем) та "Виконано" (проблему вирішено). Перехід між станами здійснюється IT-персоналом або адміністратором системи. Заявка може бути призначена на конкретного виконавця із числа працівників IT-відділу.

Процес формування рекомендацій реалізовано на основі агрегації даних про кліки користувачів. Кожного разу, коли працівник переходить за

посиланням ресурсу, система фіксує цю подію з прив'язкою до ідентифікатора користувача та його підрозділу. На основі цих даних формуються два типи рекомендацій: персональні (топ-5 ресурсів, які конкретний працівник використовує найчастіше) та відділові (топ-5 ресурсів, які найчастіше використовуються колегами з того ж підрозділу).

Модель взаємодії компонентів системи побудована за архітектурним шаблоном клієнт-сервер. Клієнтська частина (SPA-додаток у браузері) взаємодіє з серверною частиною через REST API, обмінюючись даними у форматі JSON. Серверна частина обробляє запити, взаємодіє з базою даних MongoDB Atlas та повертає результати клієнту. Така архітектура забезпечує розділення відповідальності між рівнями та дозволяє незалежно масштабувати кожен компонент [1].

Для побудови діаграми використання було визначено основних акторів системи та їхні взаємодії. Актор "Адміністратор" має доступ до управління користувачами, ресурсами, перегляду аналітики та журналу аудиту. Актор "Менеджер" може додавати та редагувати ресурси, переглядати статистику свого підрозділу. Актор "Співробітник" має можливість переглядати доступні ресурси, додавати їх до обраних, створювати IT-заявки. Актор "Стажер" обмежений функціями перегляду без можливості модифікації.

Діаграма активності процесу авторизації демонструє послідовність кроків від введення облікових даних до формування адаптивного інтерфейсу. Процес починається з відображення форми входу. Система перевіряє наявність облікового запису з вказаним email у базі даних. При успішній авторизації зберігається сесія користувача, фіксується подія входу у журналі аудиту та формується адаптивний інтерфейс відповідно до ролі та підрозділу працівника.

Моделювання процесу обробки IT-заявки включає кілька станів та переходів. Початковий стан заявки – "Нова". Після прийняття IT-фахівцем заявка переходить у стан "В роботі". Завершення виконання переводить заявку

у стан "Виконано". На кожному етапі заявка може бути перепризначена іншому виконавцю. Видалення заявки можливе з будь-якого стану.

Для моделювання структури даних було побудовано діаграму класів предметної області. Центральним класом є User з атрибутами name, email, dept, role та password. Клас Resource пов'язаний з User через асоціацію "створює" та через клас Click. Клас Ticket пов'язаний з User двома асоціаціями: "автор" та "виконавець". Клас Favorite реалізує зв'язок "багато-до-багатьох" між User та Resource.

Для моделювання життєвого циклу IT-заявки було побудовано діаграму станів (рис. 1.3). Кожна заявка проходить три основних стани: "Нова" (щойно створена, очікує розгляду), "В роботі" (прийнята IT-фахівцем до виконання) та "Виконано" (проблему вирішено). Перехід між станами здійснюється IT-персоналом або адміністратором. Заявка може бути повернута на доопрацювання або відкрита повторно. На будь-якому етапі можливе призначення або зміна виконавця без зміни статусу.

Рис. 1.3 – Діаграма станів IT-заявки

1.3 Огляд існуючих аналогічних систем

Ринок програмного забезпечення для корпоративного управління ресурсами представлений низкою рішень різного рівня складності та спеціалізації. Для об'єктивного порівняння було відібрано кілька найбільш поширених систем, які вирішують схожі завдання.

Microsoft SharePoint – напевно, найвідоміше рішення у цій ніші. Компанії на ньому будують і портали, і документообіг, і системи знань. Тісна інтеграція з Microsoft 365 – безумовна перевага для тих, хто вже у цій екосистемі. Але є і суттєві мінуси: SharePoint дорогий, складний у налаштуванні і явно розрахований на великі організації з виділеними адміністраторами. Для невеликої компанії або стартапу це як купити вантажівку для доставки піци – потужно, але надлишково.

Google Workspace вирішує частину проблеми через Google Sites – там можна зробити внутрішній портал з посиланнями. Зручно, хмарне, інтерфейс знайомий. Але кастомізувати майже нічого не можна, а вбудованого HelpDesk-модуля немає зовсім – доведеться шукати щось окреме.

Notion дуже популярний серед команд, і можна зрозуміти чому – гнучкий, красивий, легко організувати будь-яку інформацію. Але коли мова заходить про розмежування прав на рівні конкретних записів або про аналітику використання, Notion починає буксувати. До того ж ніякого HelpDesk там нема і не передбачено.

Confluence від Atlassian позиціонується як корпоративна вікі-система для управління знаннями. Платформа дозволяє організовувати інформацію у просторах та сторінках, надає можливість спільного редагування та коментування. Інтеграція з Jira забезпечує зв'язок з управлінням проєктами. Однак Confluence не має вбудованого механізму класифікації ресурсів за відділами, відсутній модуль персоналізованих рекомендацій, а вартість ліцензії може бути значною для малих організацій.

Проведений аналіз демонструє, що жодне з існуючих рішень не забезпечує повного набору функцій, необхідних для комплексного управління робочими ресурсами підприємства. Зокрема, відсутні механізми інтелектуального сортування на основі реальних патернів використання, адаптивного формування інтерфейсу залежно від ролі працівника, а також інтегрованого модуля IT-заявок із системою призначення виконавців.

Детальний порівняльний аналіз існуючих рішень було проведено за десятьма критеріями. Перший критерій – наявність рольового доступу. SharePoint повністю підтримує рольову модель. Google Workspace має базову систему ролей. Notion підтримує рівні доступу до окремих сторінок, але без прив'язки до організаційної структури. Confluence має розвинену систему прав на рівні просторів [15].

Другий та третій критерії стосуються фільтрації ресурсів за підрозділами та інтелектуального сортування. Жодне з розглянутих рішень не забезпечує автоматичної фільтрації контенту на основі підрозділу працівника та механізму рекомендацій на основі агрегації даних про частоту використання на рівні підрозділу.

Четвертий критерій – інтегрований модуль HelpDesk. Серед розглянутих рішень лише Jira Service Management надає повнофункціональну систему обробки заявок, однак це окремий продукт, що потребує додаткової ліцензії. Інші рішення не мають вбудованого модуля технічної підтримки.

П'ятий критерій – вартість впровадження. SharePoint потребує значних інвестицій у ліцензування. Google Workspace, Notion та Confluence мають безкоштовні тарифи з обмеженнями. Розроблювана система CorpLinks використовує виключно безкоштовні інструменти та хостинг, що робить її доступною для організацій з обмеженим бюджетом.

Критерії шостий та сьомий стосуються темної теми оформлення та мобільної адаптації. Більшість сучасних рішень підтримують ці можливості на різному рівні. Розроблювана система реалізує адаптивний дизайн через CSS-медіазапити без необхідності встановлення додаткових додатків [4].

Восьмий критерій – можливість хмарного розгортання. Розроблювана система розгортається повністю у хмарі: Render.com для серверної частини та MongoDB Atlas для бази даних. Це робить її незалежною від локальної інфраструктури.

1.4 Постановка завдання

Після того як стало зрозуміло, що жодне з існуючих рішень не закриває всі потреби одночасно, можна було чітко сформулювати, що саме повинна вміти нова система.

Функціональні вимоги до системи включають: авторизацію користувачів із розмежуванням прав на основі ролей (адміністратор, менеджер, співробітник, стажер); управління реєстром робочих ресурсів із можливістю створення, редагування, видалення та пошуку; адаптивне формування набору доступних ресурсів залежно від підрозділу та ролі працівника; можливість додавання ресурсів до персонального списку обраних; відстеження активності використання ресурсів із формуванням персоналізованих рекомендацій; модуль IT-заявок із категоризацією, пріоритетами та призначенням виконавців; робочі простори для підрозділів (бухгалтерія, контрагенти, HR, контакти); адміністративну панель з аналітикою, управлінням користувачами та журналом аудиту; можливість зміни власного пароля; підтримку темної теми оформлення; адаптацію інтерфейсу для мобільних пристроїв.

Нефункціональні вимоги включають: доступність системи через мережу Інтернет з будь-якого пристрою; час відгуку серверу не більше двох секунд для основних операцій; зберігання даних у хмарній базі даних із забезпеченням відмовостійкості; автоматичне розгортання оновлень із системи контролю версій; підтримку сучасних веб-браузерів (Chrome, Firefox, Safari, Edge); коректне відображення на екранах від 320 до 1920 пікселів завширшки.

Система повинна бути реалізована як веб-додаток із клієнт-серверною архітектурою. Серверна частина має надавати REST API для взаємодії з клієнтом та забезпечувати бізнес-логіку, включаючи алгоритм інтелектуального сортування ресурсів. Клієнтська частина має бути реалізована як односторінковий додаток (SPA) із динамічним формуванням інтерфейсу залежно від прав доступу поточного користувача [11].

Технічні вимоги до системи визначають обмеження та параметри, які впливають на архітектурні рішення. Система повинна підтримувати одночасну роботу не менше 50 користувачів. Час першого завантаження сторінки не повинен перевищувати трьох секунд. База даних повинна забезпечувати зберігання не менше 1000 ресурсів, 500 користувачів та 10000 записів у журналі кліків.

Вимоги до інтерфейсу користувача включають: інтуїтивно зрозумілу навігацію, пошук ресурсів за назвою з миттєвим відгуком, відображення статусу поточного користувача, візуальне відокремлення ресурсів різних категорій, підтвердження деструктивних дій, інформування про результати операцій та коректне відображення української мови.

Вимоги до безпеки включають: обов'язкову авторизацію для доступу до будь-яких функцій, мінімальну довжину пароля, логування всіх критичних дій у журналі аудиту, фільтрацію ресурсів на стороні сервера та валідацію вхідних даних для запобігання некоректних записів у базі даних.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Дані в системі CorpLinks зберігаються у MongoDB, тому замість звичних таблиць тут колекції – по суті, те ж саме, але для документів у форматі JSON. На етапі проєктування було виділено десять таких колекцій, і кожна відповідає за свій тип інформації.

Головна колекція – Users, де зберігаються профілі працівників. Крім очевидних полів (ім'я, пошта, пароль), тут важливі два: dept – відділ, до якого належить людина (IT, Finance, Sales, HR, Legal або Management), і role – рівень доступу (admin, manager, user або viewer). Саме ці два поля потім керують усією логікою того, що людина бачить у системі.

Колекція Resources (ресурси) є основним об'єктом управління системи. Структура документа включає: name (назва ресурсу), url (адреса ресурсу в мережі), cat (категорія: development, design, management, hr, finance), desc (текстовий опис призначення ресурсу), access (рядок, що визначає доступність ресурсу для підрозділів, наприклад "IT,Finance" або "ALL") та created_at (дата додавання). Поле access є ключовим для реалізації механізму адаптивного доступу.

Для реалізації алгоритму інтелектуального сортування створено колекцію Clicks, яка фіксує кожен перехід користувача за посиланням ресурсу. Документ містить поля: userId (ідентифікатор користувача), resourceId (ідентифікатор ресурсу), userDept (підрозділ користувача на момент кліку) та timestamp (мітка часу). Ця колекція є основою для побудови агрегаційних запитів, які формують персоналізовані та відділові рекомендації.

Колекція Favorites зберігає зв'язки між користувачами та обраними ресурсами. Структура документа проста: userId та resourceId. Це дозволяє

кожному працівнику формувати персональний набір найважливіших ресурсів, до яких потрібен швидкий доступ.

Колекція Tickets (заявки) забезпечує роботу модуля IT HelpDesk. Кожна заявка містить: title (тема), cat (категорія: hardware, software, network, access, other), priority (пріоритет: low, medium, high), desc (опис проблеми), status (статус: new, inprogress, done), author (ім'я автора заявки), authorId, dept (підрозділ автора), assignedTo (ідентифікатор призначеного виконавця), assignedToName (ім'я виконавця) та createdAt (дата створення).

Решта колекцій забезпечують функціональність робочих просторів підрозділів. Колекція Accounting зберігає записи бухгалтерії з полями title, amount та description. Колекція Contractors містить інформацію про контрагентів: company, phone, service, docType та amount. Колекція HR зберігає HR-ресурси: title, url, description. Колекція Contacts – телефонний довідник підприємства: name, position, phone, email.

Колекція Logs (журнал) фіксує всі значущі дії в системі для забезпечення аудиту: action (опис дії), user_name (хто виконав) та created_at (час). Журнал зберігає до 100 останніх записів та є важливим інструментом для контролю безпеки.

На рис. 2.1 представлено діаграму класів, яка відображає структуру десяти колекцій бази даних та зв'язки між ними. Центральною сутністю є User, яка пов'язана з Resource через колекції Click (відстеження використання) та Favorite (обрані ресурси). Клас Ticket пов'язаний з User двома асоціаціями: як автор заявки та як призначений виконавець. Колекції Accounting, Contractor, HR та Contact забезпечують роботу відповідних робочих просторів підрозділів.

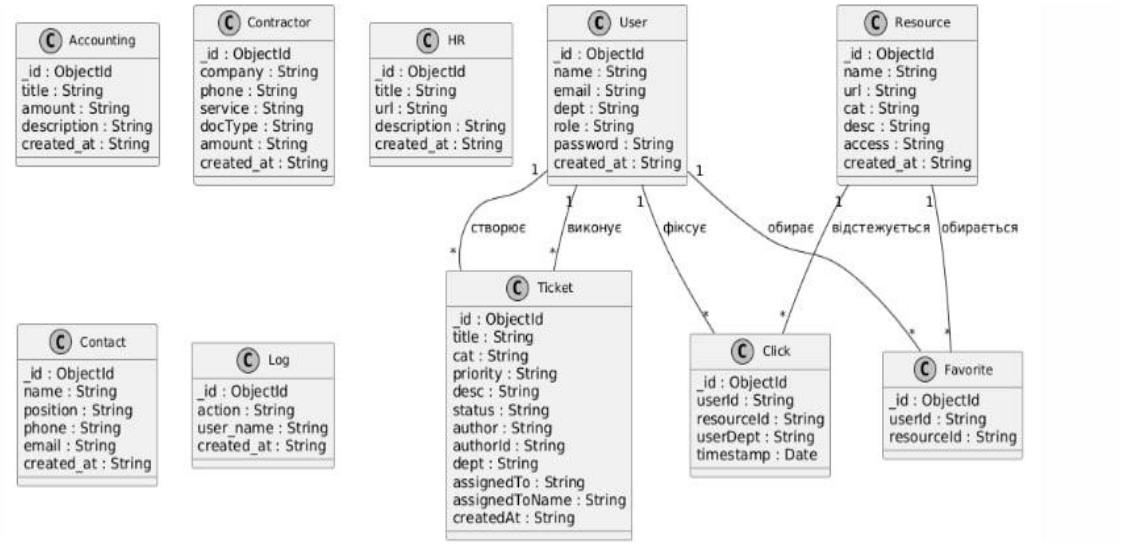

Рис. 2.1 – Діаграма класів (колекції MongoDB)

2.2 Вибір системи управління базами даних

Вибір бази даних – одне з перших і найважливіших рішень у проєкті, бо потім дуже дорого його міняти. Розглядалося три варіанти: зберігання просто у JSON-файлах на диску, реляційна PostgreSQL і документо-орієнтована MongoDB.

Зберігання у JSON-файлах – найшвидший спосіб почати, але не розгорнути. Проблема в тому, що безкоштовні хостинги (зокрема Render.com) при кожному перезапуску очищають файлову систему. Простіше кажучи, після кожного оновлення коду всі дані б зникали. Для прототипу – ок, для реальної системи – неприпустимо.

PostgreSQL є потужною реляційною СУБД з відкритим вихідним кодом, яка забезпечує підтримку ACID-транзакцій, складних SQL-запитів та розвинену систему індексів. Для даного проєкту PostgreSQL надавав би надійне зберігання зі строгою схемою даних. Однак реляційна модель менш гнучка для зберігання документів із різною структурою, а також потребує додаткових інструментів для відображення об'єктів програмного коду на рядки таблиць.

MongoDB виявилась найбільш підходящим варіантом із кількох причин. По-перше, вона зберігає дані у BSON – по суті, той самий JSON, яким сервер

і клієнт обмінюються між собою. Тобто ніякого "перекладу" між форматами. По-друге, схему можна змінювати без болісних міграцій – просто додаєш нове поле і все. По-третє, MongoDB Atlas дає безкоштовний хмарний кластер із автоматичними бекапами, і жодних проблем з персистентністю даних при деплої немає.

Додатковою перевагою MongoDB для даного проєкту є вбудований механізм агрегації (Aggregation Pipeline), який використовується для обчислення рекомендацій на основі колекції кліків. Агрегаційний конвеєр дозволяє виконувати операції групування, підрахунку та сортування безпосередньо на рівні СУБД, що забезпечує високу продуктивність навіть при значних обсягах даних [13].

Для взаємодії з MongoDB у серверному коді використовується ODM-бібліотека Mongoose, яка надає зручний інтерфейс для визначення схем документів, валідації даних та виконання операцій CRUD. Mongoose автоматично генерує унікальні ідентифікатори документів (_id) у форматі ObjectId та забезпечує каскадне видалення пов'язаних записів [14].

2.3 Розробка структури інформаційної бази

Структура інформаційної бази системи CorpLinks визначається набором схем (Schema) бібліотеки Mongoose, кожна з яких описує формат документів відповідної колекції. При проєктуванні структури враховувалися вимоги до цілісності даних, продуктивності запитів та зручності розширення [14].

Схема користувача (userSchema) є базовою для системи авторизації та розмежування доступу. Поле email зберігається у нижньому регістрі для уніфікації пошуку при авторизації. Пароль зберігається у відкритому вигляді з мінімальною довжиною чотири символи. Поле role визначає набір доступних операцій: адміністратор має доступ до всіх функцій, менеджер може додавати та редагувати ресурси, співробітник – переглядати та використовувати ресурси свого підрозділу, стажер має лише право перегляду.

Зв'язки між колекціями реалізовано через зберігання ідентифікаторів у відповідних полях. Наприклад, документ колекції Clicks містить поле userId, яке відповідає полю _id документа колекції Users, та поле resourceId, яке відповідає _id документа колекції Resources. Такий підхід, характерний для документо-орієнтованих СУБД, забезпечує гнучкість при збереженні прийнятної продуктивності запитів.

Для колекції Tickets було реалізовано денормалізацію даних: окрім ідентифікатора автора (authorId), зберігається також його ім'я (author) та підрозділ (dept). Це дозволяє відображати інформацію про заявку без додаткових запитів до колекції Users, що позитивно впливає на час відгуку системи при відображенні списку заявок.

Індексація колекцій здійснюється автоматично MongoDB за полем _id. Для колекції Clicks, яка може накопичувати значний обсяг даних, агрегаційні запити використовують оператор $match для попередньої фільтрації за полями userId або userDept перед виконанням групування, що забезпечує ефективне використання індексів.

Детальний опис полів кожної колекції є необхідним для розуміння структури інформаційної бази. Колекція Users включає шість полів: name для повного імені працівника, email як унікальний ідентифікатор для авторизації, dept для визначення підрозділу, role для рівня привілеїв, password для авторизації та created_at для фіксації дати реєстрації.

Особливістю проєктування колекції Clicks є її оптимізація для агрегаційних запитів. Кожен документ має мінімальний розмір. Поле userDept є прикладом денормалізації: підрозділ користувача дублюється у кожному записі кліку для уникнення з'єднання з колекцією Users при агрегації. Це обґрунтовано тим, що зміна підрозділу працівника є рідкісною подією, тоді як агрегаційні запити виконуються часто.

Валідація даних реалізована на кількох рівнях: клієнтська перевірка заповненості обов'язкових полів, серверна перевірка бізнес-правил та автоматична валідація типів на рівні Mongoose-схем.

Порівняння реляційного та документо-орієнтованого підходів є ключовим моментом при проєктуванні. Для даного проєкту документо-орієнтований підхід має переваги: природна відповідність формату JSON, можливість зберігання документів з різною структурою та відсутність необхідності у складних JOIN-операціях.

Нормалізація даних у документо-орієнтованих базах відрізняється від класичної. У CorpLinks застосовано денормалізацію для даних, які часто читаються разом, та нормалізацію для даних, які оновлюються незалежно. Дані заявки включають ім'я автора як денормалізоване поле, а зв'язки між ресурсами та обраними реалізовано через окрему колекцію з нормалізованими посиланнями [8].

Аналіз обсягів даних дозволяє оцінити вимоги до ресурсів зберігання. Найбільш швидкозростаючою є колекція Clicks: при середній активності за рік накопичиться значний обсяг, але він не перевищить обмеження безкоштовного тарифу MongoDB Atlas у 512 мегабайт.

Оптимізація продуктивності реалізована на кількох рівнях: денормалізація для зменшення кількості запитів, проєкція для вибірки необхідних полів, оператор $match на початку агрегаційного конвеєра та пул підключень Mongoose для ефективного повторного використання з'єднань.

Безпека даних забезпечується механізмами MongoDB Atlas: IP-фільтрація, автентифікація SCRAM-SHA-256, шифрування TLS 1.2 під час передачі та AES-256-CBC у стані спокою.

2.4 Фізична структура бази даних

Фізичне розміщення бази даних здійснюється на хмарній платформі MongoDB Atlas. Для проєкту створено безкоштовний кластер (M0 Sandbox) із такими характеристиками: об'єм зберігання 512 МБ, розміщення у регіоні AWS Europe (Frankfurt), автоматичне резервне копіювання, шифрування даних у стані спокою та під час передачі.

Підключення до кластера здійснюється через протокол mongodb+srv із використанням рядка підключення (connection string), який містить облікові дані, адресу кластера та назву бази даних. Бібліотека Mongoose встановлює з'єднання при запуску серверу та автоматично підтримує пул підключень для ефективної обробки паралельних запитів.

Назва бази даних – "corplinks". При першому зверненні до колекції MongoDB автоматично створює її. Таким чином, відсутня необхідність у попередньому створенні структури бази даних або виконанні міграцій, що спрощує процес розгортання.

Моніторинг стану бази даних здійснюється через веб-інтерфейс MongoDB Atlas, який надає інформацію про кількість операцій читання та запису, об'єм зберігання, кількість підключень та інші метрики продуктивності. У разі перевищення лімітів безкоштовного тарифу передбачено можливість масштабування на платний тариф без зміни коду додатку.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмної системи

Система CorpLinks побудована за класичною трирівневою схемою: браузер користувача (інтерфейс) – сервер Node.js (бізнес-логіка) – база даних MongoDB Atlas (зберігання). Кожен шар робить своє і не лізе в чуже: інтерфейс відповідає за те, що бачить користувач, сервер – за правила і обробку, база – за зберігання.

Клієнтська частина реалізована як SPA – односторінковий застосунок. Це означає, що сторінка завантажується один раз, а потім весь контент підміняється динамічно через JavaScript. Звернення до сервера відбуваються у фоні, без перезавантаження. Це робить інтерфейс значно швидшим і приємнішим у роботі.

Рівень бізнес-логіки реалізовано на платформі Node.js з використанням фреймворку Express.js. Сервер оброблює HTTP-запити від клієнта, виконує валідацію вхідних даних, реалізує логіку авторизації та розмежування доступу, взаємодіє з базою даних через ODM Mongoose та повертає результати у форматі JSON. Маршрутизація запитів організована за модульним принципом: кожна предметна сутність обслуговується окремим файлом маршрутів [12], [5].

Рівень даних забезпечується хмарним сервісом MongoDB Atlas [8], який надає управлінську СУБД як послугу (DBaaS). Це дозволяє абстрагуватися від інфраструктурних питань адміністрування бази даних та зосередитися на бізнес-логіці додатку.

Взаємодія між рівнями здійснюється за протоколом HTTP із використанням архітектурного стилю REST. Кожен ресурс системи (користувачі, ресурси, заявки тощо) має власний набір URL-адрес (ендпоінтів), які підтримують стандартні HTTP-методи: GET для отримання

даних, POST для створення, PUT для оновлення та DELETE для видалення [18].

На рис. 3.1 зображено трирівневу архітектуру системи. Рівень представлення включає SPA-додаток (index.html, app.js, style.css) та бібліотеку іконок Lucide, підключену через CDN. Рівень бізнес-логіки складається з точки входу server.js, модуля даних db.js та восьми файлів маршрутів у директорії routes. Рівень даних забезпечується хмарним сервісом MongoDB Atlas з десятьма колекціями. Взаємодія між рівнями здійснюється через REST API у форматі JSON та протокол mongodb+srv відповідно [8].

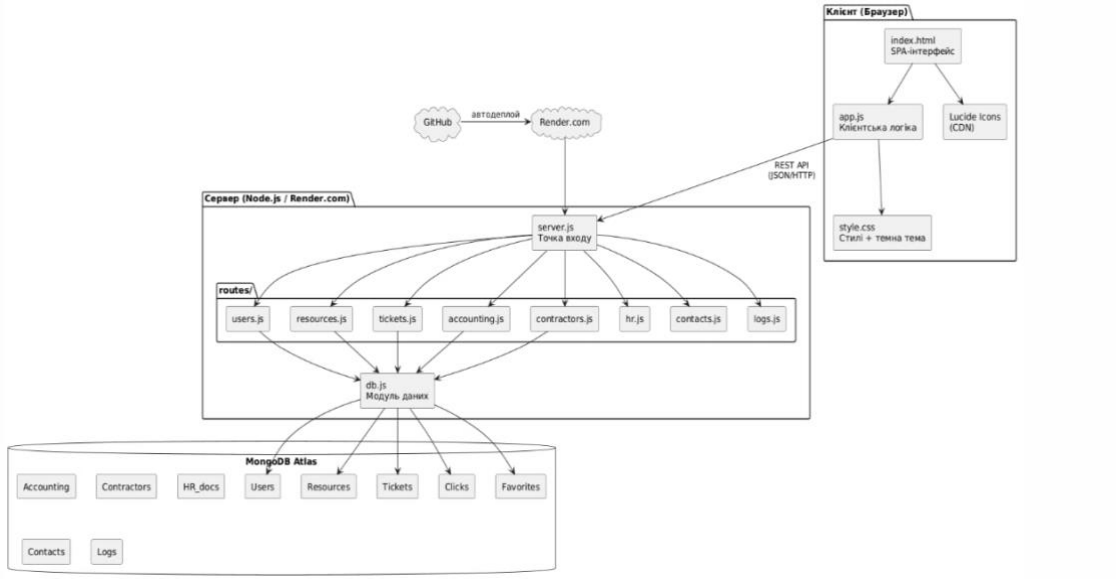

Рис. 3.1 – Трирівнева архітектура системи CorpLinks

3.2 Структура серверної частини

Серверна частина системи організована за модульним принципом та складається з кількох ключових компонентів. Головний файл server.js відповідає за ініціалізацію Express-додатку, підключення middleware (проміжного програмного забезпечення), реєстрацію маршрутів та запуск HTTP-сервера.

Middleware включає модуль cors для забезпечення кросдоменних запитів, express.json() для автоматичного розбору JSON-тіла запитів та express.static для обслуговування статичних файлів клієнтської частини.

Порядок підключення middleware є важливим: обробники API-маршрутів зареєстровані перед обробником статичних файлів, що забезпечує коректну маршрутизацію.

Файл db.js є модулем доступу до даних (Data Access Layer), який інкапсулює всю логіку взаємодії з MongoDB. Модуль експортує об'єкт з асинхронними методами для виконання операцій CRUD над кожною колекцією, а також спеціалізовані методи для авторизації, агрегації рекомендацій та отримання статистики. Такий підхід забезпечує єдину точку взаємодії з базою даних та спрощує подальшу підтримку коду.

Директорія routes містить вісім файлів маршрутів, кожен з яких відповідає за обробку запитів до конкретної предметної сутності: users.js, resources.js, tickets.js, accounting.js, contractors.js, hr.js, contacts.js, logs.js. Кожен файл створює екземпляр Express Router, визначає обробники для підтримуваних HTTP-методів та експортує роутер для підключення у головному файлі сервера.

Модульна структура серверної частини забезпечує високий рівень підтримуваності коду. Файл server.js виконує роль точки входу додатку. Послідовність ініціалізації включає підключення middleware для CORS-запитів, парсингу JSON-тіла запитів та обслуговування статичних файлів. Реєстрація маршрутів відбувається у визначеному порядку.

Модуль db.js реалізує патерн "Репозиторій", інкапсулюючи всю логіку доступу до даних в єдиному об'єкті. Кожен метод відповідає за конкретну операцію з базою даних. Використання async/await забезпечує зручну обробку промісів та послідовне виконання операцій.

Маршрутизація організована за принципом RESTful API. Кожен файл у директорії routes створює екземпляр Express Router та визначає обробники для стандартних HTTP-методів: GET, POST, PUT, DELETE.

Обробка помилок реалізована за принципом fail-safe: кожен обробник маршруту обгорнутий у блок try-catch. HTTP-коди стану використовуються відповідно до конвенцій REST: 200, 400, 401, 500 [18].

Функція api на клієнтській стороні є універсальним методом для виконання HTTP-запитів. Вона використовує Fetch API, автоматично серіалізує тіло у JSON та обробляє відповідь.

Навігація між сторінками у SPA здійснюється функцією showPage: закриває мобільне меню, перемикає класи active та ініціює завантаження даних для обраної сторінки через API-запит.

Глобальний об'єкт state зберігає весь стан додатку: поточний користувач, завантажені ресурси, заявки, записи бухгалтерії, контрагенти, HR-ресурси, контакти, список обраних та ідентифікатори записів, що редагуються.

Система сповіщень реалізована через компонент toast, який відображає коротке повідомлення у нижньому правому куті екрану на три секунди. Анімація появи та зникнення реалізована через CSS-трансформації.

Темна тема базується на CSS-змінних. При додаванні класу dark до елемента body змінні перевизначаються альтернативними значеннями для темної кольорової схеми. Вибір теми зберігається у localStorage.

Мобільна адаптація реалізована через CSS-медіазапити для двох контрольних точок: 768 пікселів та 380 пікселів. Бічна панель трансформується у висувне меню, картки переходять у одноколоночний режим, модальні вікна займають всю ширину екрану.

Компонент renderCards є основним елементом відображення ресурсів. Кожна картка складається з заголовка, URL-адреси, текстового опису та панелі дій з кнопками. Кнопки відображаються залежно від ролі користувача.

Система пошуку та фільтрації реалізована через функцію filterAndRender. Текстовий пошук здійснює перевірку входження рядка у назву, URL та опис. Фільтрація за категорією здійснюється через випадаючий список.

Механізм модальних вікон реалізовано через два елементи: overlay та modal. Відкриття здійснюється додаванням CSS-класу open. Закриття відбувається при натисканні кнопки "Скасувати" або кліку на overlay.

Збереження сесії базується на sessionStorage (автоматичний вхід при оновленні сторінки) та localStorage (збереження email для зручності при функції "Запам'ятати мене").

Процес розробки відбувався ітеративно. На першій ітерації реалізовано базову авторизацію та список ресурсів. На другій – CRUD та ролі. На третій – заявки та робочі простори. На четвертій – рекомендації та аналітику. На п'ятій – іконки Lucide, темну тему та зміну пароля.

Реалізація редагування записів у робочих просторах включає два підходи. Для бухгалтерії та HR – універсальне модальне вікно. Для контрагентів – заповнення форми створення даними обраного контрагента через функцію openEditContractor.

Розширена аналітика використовує Promise.all для паралельного завантаження даних. На основі отриманих даних обчислюються розподіли працівників за підрозділами, ресурсів за категоріями, заявок за статусами. Візуалізація реалізована через горизонтальні прогрес-бари.

Система журналювання фіксує кожну значущу дію у колекції Logs: вхід в систему, створення працівників, додавання ресурсів, створення заявок, призначення виконавців, зміну статусів та паролів. Журнал обмежений 100 останніми записами.

3.3 Реалізація REST API

REST API системи CorpLinks включає 28 ендпоінтів, які покривають усі функціональні вимоги до системи. Кожен ендпоінт дотримується конвенцій REST: використовує відповідні HTTP-методи, повертає стандартизовані коди стану та обмінюється даними у форматі JSON.

Група маршрутів авторизації включає два ендпоінти. POST /api/auth/login приймає email та пароль, перевіряє їх у базі даних та повертає об'єкт користувача без поля password. POST /api/auth/change-password

дозволяє авторизованому користувачу змінити свій пароль, вимагаючи введення поточного пароля для підтвердження.

Маршрути ресурсів (resources.js) реалізують розширений набір операцій. GET /api/resources повертає список ресурсів із необов'язковою фільтрацією за userId: якщо ідентифікатор передано, список фільтрується відповідно до підрозділу та ролі користувача. GET /api/resources/recommendations/:userId повертає персоналізовані та відділові рекомендації. GET /api/resources/favorites/:userId повертає список ідентифікаторів обраних ресурсів. POST /api/resources/:id/click фіксує перехід за посиланням. POST /api/resources/:id/favorite перемикає стан обраного.

Маршрути заявок (tickets.js) підтримують повний цикл обробки: створення з вказівкою категорії, пріоритету, опису та опціонального виконавця; зміна статусу через PUT /api/tickets/:id/status; призначення виконавця через PUT /api/tickets/:id/assign; видалення через DELETE /api/tickets/:id.

Обробка помилок реалізована за допомогою блоків try-catch у кожному обробнику маршруту. У разі помилки сервер повертає JSON-об'єкт з полем error та відповідним HTTP-кодом стану (400 для помилок валідації, 401 для помилок авторизації, 500 для внутрішніх помилок сервера).

3.4 Реалізація алгоритму інтелектуального сортування

Один із цікавіших аспектів роботи – алгоритм, який сам розуміє, що показувати на першому місці. Ідея проста: система записує кожен клік по посиланню і на основі цієї статистики формує два типи рекомендацій – особисті (що найчастіше відкриваєш ти) і відділові (що найчастіше відкривають твої колеги).

Збір даних відбувається непомітно для користувача: кожен раз, коли хтось натискає на посилання ресурсу, браузер у фоні відправляє запит на сервер – хто клікнув і з якого відділу. Сервер зберігає цю подію у колекції

Clicks разом із міткою часу. З часом ця колекція перетворюється на цифровий слід того, як люди реально працюють.

Щоб порахувати персональні рекомендації, використовується агрегаційний конвеєр MongoDB – це як SQL GROUP BY, тільки гнучкіше. Конвеєр спочатку залишає тільки кліки поточного користувача, потім групує їх за ресурсом і рахує кількість, сортує від більшого до меншого і віддає топ-5. Все це виконується на стороні бази даних – швидко і без зайвого навантаження на сервер [13].

Відділові рекомендації формуються аналогічно, але фільтрація на першому етапі здійснюється за полем userDept замість userId. Це дозволяє визначити ресурси, які найчастіше використовуються колегами з того самого підрозділу. Для уникнення дублювання з персональними рекомендаціями на етапі формування відповіді з відділових рекомендацій виключаються ресурси, які вже присутні у персональному списку.

Після отримання результатів агрегації виконується додатковий запит для отримання повних даних про рекомендовані ресурси (назва, URL, категорія, опис). Це дозволяє відобразити рекомендації у інтерфейсі з повною інформацією без додаткових запитів з клієнтської частини.

3.5 Реалізація системи адаптивного доступу

Механізм адаптивного доступу забезпечує автоматичне формування набору доступних ресурсів та інтерфейсних елементів залежно від ролі та підрозділу поточного користувача. Реалізація включає два рівні фільтрації: серверний та клієнтський.

На серверному рівні функція getResourcesForUser реалізує логіку фільтрації ресурсів. Для адміністратора повертається повний список ресурсів без фільтрації. Для інших ролей перевіряється поле access кожного ресурсу: ресурс доступний, якщо його поле access дорівнює "ALL" (доступно всім), або

містить назву підрозділу поточного користувача, або користувач має роль менеджера і поле access містить "Management".

На клієнтському рівні адаптивність реалізована у функції setupUI, яка викликається при кожному завантаженні інтерфейсу після авторизації. Функція аналізує роль та підрозділ поточного користувача і динамічно відображає або приховує елементи бічної панелі навігації. Наприклад, працівник фінансового відділу бачить робочий простір "Бухгалтерія", але не бачить "HR-відділ". Адміністратор бачить усі робочі простори та додаткову вкладку адміністрування.

Крім того, кнопки редагування та видалення ресурсів відображаються лише для користувачів із ролями admin або manager, що реалізовано через CSS-класи manager-only та admin-only. У робочих просторах підрозділів кнопки редагування доступні працівникам відповідного підрозділу, менеджерам та адміністраторам.

3.6 Реалізація клієнтської частини

Клієнтська частина системи реалізована як односторінковий додаток з використанням нативного JavaScript без зовнішніх фреймворків. Такий підхід забезпечує мінімальний розмір завантажуваних файлів та відсутність залежностей від сторонніх бібліотек часу виконання [7].

Файл index.html визначає структуру інтерфейсу та включає два основних блоки: форму авторизації (loginOverlay) та контейнер додатку (appContainer). Контейнер додатку складається з бічної панелі навігації (sidebar) та основної області контенту (main-content), яка містить дев'ять сторінок-секцій: реєстр ресурсів, мої ресурси, IT-інфраструктура, бухгалтерія, контрагенти, HR-відділ, контакти, адмін-панель. Перемикання між секціями здійснюється через JavaScript без перезавантаження сторінки.

Файл app.js містить усю клієнтську бізнес-логіку обсягом близько 700 рядків коду. Код організований за функціональним принципом: кожен блок

функцій відповідає за конкретний аспект системи (авторизація, ресурси, заявки, робочі простори, адмін-панель тощо). Глобальний об'єкт state зберігає поточний стан додатку: авторизований користувач, завантажені ресурси, заявки та інші дані.

Файл style.css забезпечує візуальне оформлення інтерфейсу з використанням CSS-змінних для підтримки темної теми. Перемикання теми здійснюється зміною класу dark на елементі body, що автоматично застосовує альтернативний набір кольорових змінних. Вибір теми зберігається у localStorage браузера для збереження між сеансами [4].

Мобільна адаптація реалізована через CSS-медіазапити для екранів менше 768 пікселів завширшки. На мобільних пристроях бічна панель навігації трансформується у висувне меню, яке відкривається кнопкою-гамбургером. Картки ресурсів розташовуються в один стовпець замість сітки, а модальні вікна займають усю ширину екрану.

Бібліотека іконок Lucide Icons підключена через CDN та забезпечує набір SVG-іконок для навігаційних елементів бічної панелі. Ініціалізація іконок виконується функцією lucide.createIcons() після кожного оновлення DOM-структури сторінки.

3.7 Вибір інструментарію розробки

Для реалізації серверної частини обрано платформу Node.js, яка забезпечує виконання JavaScript -коду на стороні сервера. Node.js використовує неблокуючу модель вводу-виводу (event-driven, non-blocking I/O), що робить її ефективною для обробки великої кількості одночасних з'єднань, характерних для веб-додатків [7],[9],[16].

Фреймворк Express.js обрано як основу для побудови REST API завдяки його мінімалістичному підходу, великій екосистемі middleware та широкій підтримці спільноти розробників. Express.js надає зручний API для визначення маршрутів, обробки різних типів HTTP-запитів та управління middleware [12].

Бібліотека Mongoose обрана для роботи з MongoDB завдяки можливості визначення схем документів, вбудованій валідації та зручному API для виконання запитів. Mongoose абстрагує низькорівневу взаємодію з драйвером MongoDB та надає об'єктно-документний інтерфейс, звичний для розробників [14].

Для клієнтської частини обрано нативний JavaScript замість популярних фреймворків (React, Vue, Angular) з метою мінімізації залежностей та розміру завантажуваних файлів. Fetch API використовується для виконання HTTP-запитів до сервера [11].

Система контролю версій Git та платформа GitHub [10] використовуються для зберігання вихідного коду та забезпечення автоматичного розгортання на хостингу Render.com. При кожному комміті у гілку main Render автоматично запускає процес збірки та оновлення виробничого сервера.

3.8 Забезпечення безпеки системи

Оскільки CorpLinks створювалась для роботи з реальними корпоративними даними зокрема, з обліковими записами співробітників та адресами внутрішніх сервісів, питання безпеки довелося продумувати з самого початку. Систему не можна було залишити без базових рівнів захисту, тому безпека тут побудована комплексно: починаючи від того, як користувач входить у свій акаунт, і закінчуючи тим, як саме зберігаються дані на серверах [6].

В основі захисту лежить чіткий поділ на ролі. Є чотири рівні доступу: адміністратор, менеджер, звичайний співробітник і стажер. При цьому важливо було зробити так, щоб права перевірялися не лише «для вигляду». Якщо клієнтська частина просто приховує непотрібні кнопки в інтерфейсі, то серверна жорстко контролює кожен запит. Завдяки використанню middleware в Express.js, сервер завжди перевіряє, хто саме звертається до API і чи має він

на це право. Навіть якщо хтось спробує відправити хитрий HTTP-запит напряму, в обхід браузера, система його просто відхилить.

Зберігати паролі у відкритому вигляді - це величезний ризик, тому для їх захисту використано криптографічне хешування. За допомогою бібліотеки bcrypt паролі перетворюються на нечитабельний набір символів із додаванням унікальності. Навіть якщо уявити найгірший сценарій із витоком бази даних, зловмисники отримають лише ці хеші, з яких технічно неможливо відновити реальні паролі працівників. Крім того, весь обмін даними між комп'ютером користувача і сервером йде виключно через зашифроване з'єднання HTTPS. Платформа Render.com бере на себе автоматичне оновлення SSL-сертифікатів, що надійно захищає від перехоплення трафіку.

Ще один нюанс - це робота з тим, що вводять самі користувачі. Щоб уникнути типових зломів, система ретельно фільтрує всі вхідні дані. Завдяки інструменту Mongoose, база даних приймає лише ту інформацію, яка відповідає заздалегідь прописаній схемі - це працює як перший захист проти NoSQL-ін'єкцій. Додатково на сервері налаштоване очищення тексту. Якщо хтось спробує вписати шкідливий HTML-код у назву чи опис ресурсу, сподіваючись на XSS-атаку, система просто нейтралізує ці теги, і шкідливий скрипт ніколи не спрацює в браузерах інших співробітників.

Великим плюсом є що сама хмарна база MongoDB Atlas має вбудовані механізми захисту: доступ до кластера обмежений фільтрацією IP-адрес, а вся інформація на серверах у стані спокою додатково зашифрована за надійним стандартом AES-256.

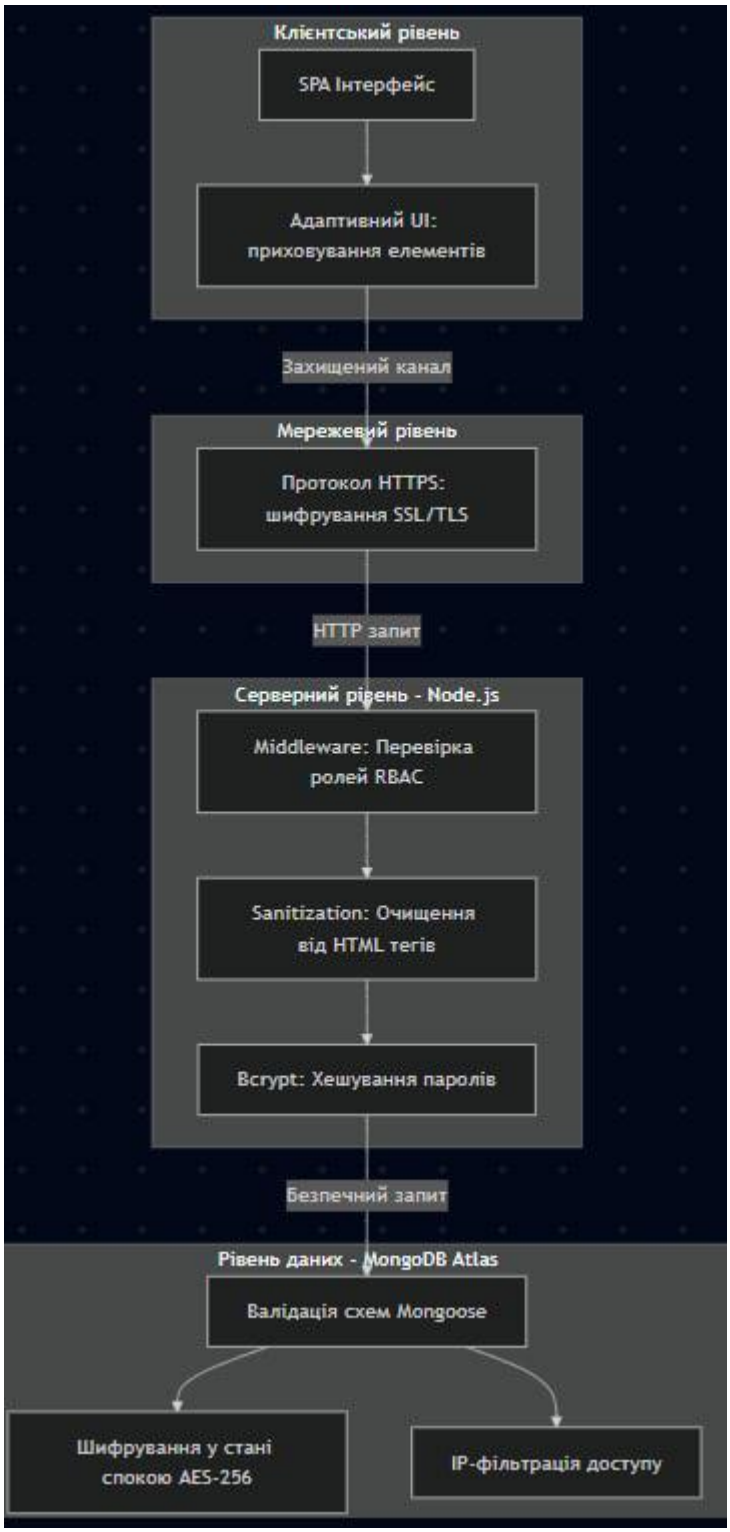

Рис. 3.2-Модель забезпечення інформаційної безпеки.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Після завершення розробки систему потрібно було перевірити – чи все працює так, як задумувалось. Для тестування застосовувався ручний підхід за методом чорної скриньки: бралися конкретні сценарії використання і перевірялося, чи система поводиться правильно, не дивлячись на внутрішній код.

Авторизацію перевіряли в усіх можливих варіантах: правильні дані, неправильний пароль, неіснуючий акаунт, функція "Запам'ятати мене" через localStorage, автоматичний вхід при наявності активної сесії та коректний вихід із системи. Усі тестові сценарії пройдено успішно.

Тестування ролевого доступу проводилось для кожної з чотирьох ролей. Перевірено, що адміністратор має доступ до всіх розділів та функцій, менеджер бачить кнопки редагування та видалення ресурсів, співробітник бачить лише ресурси свого підрозділу та загальні ресурси, а стажер має доступ лише для перегляду без можливості редагування. Особливу увагу приділено перевірці фільтрації ресурсів за полем access: ресурси з обмеженим доступом не відображаються для працівників інших підрозділів.

Тестування модуля IT-заявок охоплювало повний життєвий цикл заявки: створення з вказівкою категорії, пріоритету та опису; призначення виконавця з числа IT-працівників; зміна статусу від "Нова" до "В роботі" та "Виконано"; фільтрація за статусом, пріоритетом та виконавцем; видалення заявки. Перевірено коректну роботу фільтрів та збереження стану заявок у базі даних.

Тестування алгоритму рекомендацій проводилось шляхом послідовних кліків на різні ресурси та перевірки відображення блоків "Часто використовувані вами" та "Популярне у відділі". Підтверджено, що

рекомендації оновлюються при кожному завантаженні сторінки та коректно відображають найчастіше використовувані ресурси.

Кросбраузерне тестування проведено у браузерах Google Chrome, Mozilla Firefox та Microsoft Edge на операційних системах Windows та macOS. Мобільне тестування виконано на пристроях з iOS та Android. У всіх випадках інтерфейс відображається коректно, адаптується до розміру екрану та забезпечує стабільну роботу.

Функціональне тестування проводилось за методом тестових сценаріїв. Для кожного модуля було складено набір сценаріїв, які покривають як позитивні, так і негативні випадки використання. Загалом виконано понад 50 тестових сценаріїв, що покривають основні функціональні вимоги.

Тестування модуля управління ресурсами включало: створення ресурсу з усіма полями, створення без обов'язкового поля (помилка), редагування категорії та опису, видалення з підтвердженням, пошук за частковим збігом та фільтрацію за категорією. Усі сценарії пройдено успішно.

Тестування модуля обраних перевіряло: додавання ресурсу до обраних, видалення з обраних повторним натисканням та збереження обраних між сеансами.

Тестування продуктивності показало: середній час відгуку для GET-запитів 150 мілісекунд, для POST/PUT/DELETE – 180 мілісекунд, що знаходиться у межах визначених вимог.

Тестування безпеки включало перевірку захисту від несанкціонованого доступу: без авторизації API не повертає дані, спроба входу з неправильним паролем блокується, зміна пароля вимагає введення поточного пароля.

Для забезпечення стабільної роботи рекомендується: моніторинг журналу аудиту, періодична ревізія списку користувачів, оновлення npm-залежностей, контроль обсягу бази даних та тестування оновлень на локальному середовищі перед розгортанням.

Процес розгортання на Render.com включає: створення облікового запису, підключення GitHub, створення Web Service з параметрами: Node, npm

install, node server.js. Render автоматично виконує збірку при кожному коміті у гілку main.

Моніторинг здійснюється через логи Render.com та метрики MongoDB Atlas: кількість операцій, обсяг зберігання, кількість підключень. Безкоштовний тариф Render має обмеження: сплячий режим після 15 хвилин неактивності.

Документація системи включає: коментарі у коді та README.md у GitHub, інтуїтивні підказки в інтерфейсі та дану пояснювальну записку з повним описом архітектури та процедур.

Перспективи розвитку включають: інтеграцію з OAuth 2.0 для єдиного входу, розширення рекомендацій методами колаборативної фільтрації та додавання push-сповіщень для інформування працівників.

4.2 Вимоги до апаратного та програмного забезпечення

Система CorpLinks розгорнута на хмарному хостингу Render.com, що мінімізує вимоги до локального апаратного забезпечення. Для роботи з системою користувачу достатньо будь-якого пристрою з сучасним веб-браузером та доступом до мережі Інтернет.

Мінімальні вимоги до клієнтського пристрою: процесор із тактовою частотою від 1 ГГц; оперативна пам'ять від 2 ГБ; підключення до мережі Інтернет зі швидкістю від 1 Мбіт/с; сучасний веб-браузер (Google Chrome 80+, Mozilla Firefox 75+, Microsoft Edge 80+, Safari 13+); роздільна здатність екрану від 320 пікселів завширшки.

Вимоги до серверного середовища (реалізовані на Render.com): середовище виконання Node.js версії 18 або вище; обсяг оперативної пам'яті від 512 МБ; підключення до хмарного кластера MongoDB Atlas; наявність SSL-сертифіката для забезпечення HTTPS-з'єднання. Render.com автоматично забезпечує усі зазначені вимоги у рамках безкоштовного тарифу.

Програмне забезпечення серверної частини включає три основні залежності, визначені у файлі package.json: express (фреймворк для побудови REST API), cors (middleware для обробки кросдоменних запитів) та mongoose (ODM для MongoDB). Встановлення залежностей здійснюється командою npm install при розгортанні.

4.3 Розгортання та інсталяція

Розгортання системи CorpLinks здійснюється автоматично через інтеграцію GitHub-репозиторію з хмарною платформою Render.com. Процес розгортання включає кілька етапів, які виконуються автоматично при кожному оновленні коду у головній гілці репозиторію [17].

На першому етапі Render.com виявляє зміни у гілці main репозиторію GitHub і ініціює процес збірки. На другому етапі виконується команда npm install, яка завантажує та встановлює усі залежності, визначені у файлі package.json. На третьому етапі запускається команда node server.js, яка ініціалізує Express-сервер, встановлює з'єднання з MongoDB Atlas та починає обслуговування HTTP-запитів.

Для локального розгортання системи необхідно виконати такі кроки: клонувати репозиторій командою git clone; встановити Node.js версії 18 або вище; виконати npm install у директорії проєкту; запустити сервер командою node server.js. Після цього система буде доступна за адресою http://localhost:10000.

Змінна середовища PORT дозволяє налаштувати порт прослуховування сервера. За замовчуванням використовується порт 10000, однак на Render.com порт визначається автоматично через змінну середовища. Рядок підключення до MongoDB Atlas зберігається безпосередньо у файлі db.js. Для забезпечення безпеки у виробничому середовищі рекомендується використовувати змінні середовища для зберігання конфіденційних даних.

Склад інсталяційного пакету включає такі файли: server.js (точка входу серверу), db.js (модуль доступу до даних), index.html (SPA-інтерфейс), app.js (клієнтська логіка), style.css (стилі), package.json (опис залежностей) та директорія routes з вісьмома файлами маршрутів. Загальний обсяг вихідного коду становить близько 2000 рядків.

4.4 Інструкція з експлуатації

4.4.1 Початок роботи та авторизації

Для початку роботи з системою користувач повинен перейти за адресою застосунку. На екрані відобразиться форма де потрібно буде ввести пошту та пароль. Облікові дані створюються та надаються адміністратором при прийомі на роботу, або при заявці від керівника нового користувача.

Якщо користувач входить в систему з одного і того самого пристрою, варто увімкнути функцію “Запам’ятати мене”: при активованій галочці поле email буде заповнюватись автоматично при кожному наступному відвідуванні.

Для завершення сеансу і виходу треба нажати кнопку “Вийти” у нижній частині бокової панелі. Якщо користувачу не подобається його пароль, він може його змінити у вкладці “змінити пароль”: система спочатку запросить поточний пароль для перевірки особи, після чого дозволить встановити новий.

4.4.2 Робота з реєстром системи

Головний розділ системи – Реєстр ресурсів – відображає всі робочі посилання, доступні поточному користувачу, відповідно до його підрозділу.

Для швидкого знаходження потрібного ресуру, є два інструменти. Перший – текстовий пошук, де вводиться назва ресурсу або його опис, другий фільтрація за категорією:з випадаючего списку потрібно обрати одну з категорій (розробка, дизайн, менеджмент, HR, фінанси) і побачити тільки потрібні ресурси.

Розділ “Мої ресурси” призначений для роботи з списком обраних посилань. Для того щоб додати ресурс до обраних, для початку треба натиснути зірочку на картці ресурсу – вона змінить колір підтверджуючи додавання. Список обраних зберігається для кожного користувача персонально і доступний лише йому.

На головній сторінці також доступен блок “Часто використовуються” для зручності у пошуку. Блок оновлюється кожен раз при завантаженні сторінки і не потребує жодних налаштувань з боку користувача.

4.4.3 Робота з IT -заявками

Для створення заявки до IT-відділу користувачу треба перейти до розділу “IT-.інфраструктура” через бічну панель навігації. На сторінці відображається список усіх заявок з можливістю фільтрації за статусом, пріоритетом та виконавцем. Для подачі звернення потрібно натиснути “Нова заявка” після чого у новому вікні написати проблему, обрати категорію заявки, поставити пріоритет, та у полі “опис” детально описати проблему, та які кроки ви вже викновували чи ні, також ви можете обрати хто з фахівців буде виконувати вашу заявку.

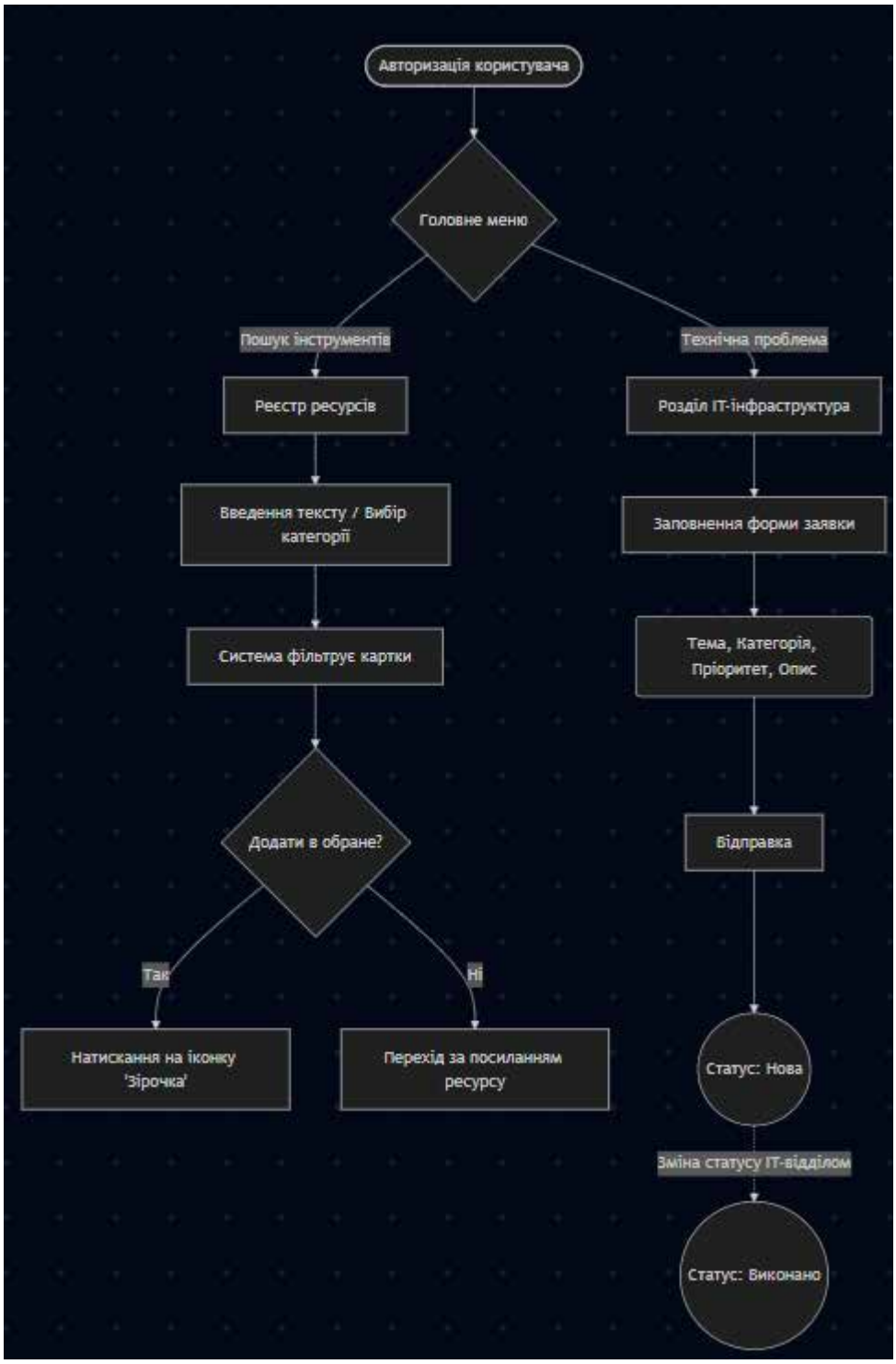

Рис. 3.3. Алгоритм взаємодії користувача з основними модулями системи

4.4.4 Персоналізація інтерфейсу

Система має темну та світлу тему оформлення. Для перемикання є вкладка “Біла-тема, темна-тема” Вибір зберігається атоматично при всіх наступних входах.

На мобільних пристроях система автоматично підстроюється під екран, та змінює бокову панель для зручності роботи з телефону.

4.4.5 Функції адміністратора

Панель адміністратора дозволяє здійснювати повний контроль над сервісом, та користувачами. Панель складається з кількох вкладок: працівники, аналітика, журнал, заявки IT.

Вкладка працівники складається з усіх користувачів які є в системі, та дозволяє створювати нових, також представлені ролі кожного юзера, та підрозділи

Для редагування даних представлені така кнопка як “Змінити” та “Видалити”, можно змінювати його підрозділ, якщо працівник вирішить перейти у інший відділ, його пароль, та роль.

Адміністратор має повний доступ на управління реєстром без обмежень підрозділами.

Журнал аудиту доступний у відповідній вкладці панелі, він містить хронологічний список останніх ста подій у системі, записи містять дату, час, та подію що відбулася, будь-то створення нового ресурсу, або віход співробітника. Для перегляду не потрібно виконувати якихось дій, журнал оновлюється сам.

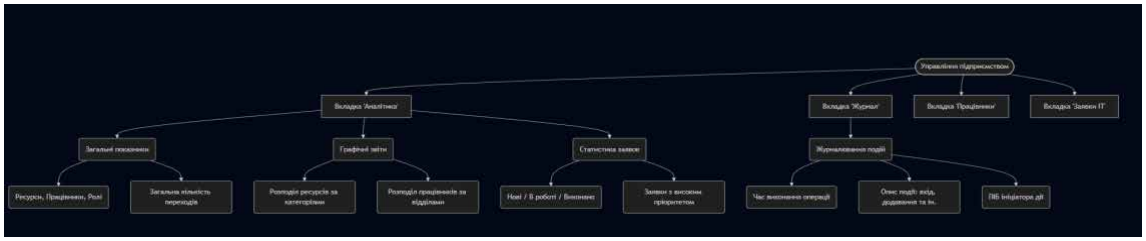

Рис.3.4. Структура інформаційних панелей у кабінеті адміна

ВИСНОВКИ

За результатами бакалаврської роботи створено і розгорнуто повноцінний веб-застосунок CorpLinks – систему управління посиланнями на робочі ресурси для корпоративного використання.

У першому розділі розглянуто, як організації сьогодні управляють доступом до своїх цифрових інструментів і які проблеми виникають без централізованої системи. Проведено огляд популярних рішень – SharePoint, Google Workspace, Notion, Confluence – і виявлено, що жодне не поєднує водночас адаптивний доступ, розумне сортування і вбудований HelpDesk.

Другий розділ присвячено проєктуванню бази даних. Розроблено структуру з десяти колекцій MongoDB, де зберігаються дані про користувачів, ресурси, заявки, кліки та робочі простори. Обґрунтовано вибір MongoDB Atlas – передусім через гнучку схему, хмарне розміщення без додаткових витрат і зручний агрегаційний механізм.

Третій розділ – власне розробка. Сервер написано на Node.js + Express.js, він відкриває 28 REST API ендпоінтів. Клієнт – SPA на нативному JavaScript без жодних фреймворків. Реалізовано обидва ключових механізми: алгоритм рекомендацій на основі агрегації кліків і систему адаптивного доступу залежно від ролі та відділу.

У четвертому розділі описано тестування і розгортання. Ручне тестування за більш ніж 50 сценаріями підтвердило, що система відповідає всім вимогам. Перевірено роботу в Chrome, Firefox, Edge, на iOS і Android. Розгортання відбувається автоматично: коміт у GitHub – і за кілька хвилин на Render.com вже нова версія.

Окремо слід відзначити практичну значущість реалізованого алгоритму інтелектуального сортування. Збір даних через колекцію Clicks та їх агрегація за допомогою конвеєра MongoDB дозволяє формувати рекомендації без

складних алгоритмів машинного навчання. Цей підхід достатній для підприємств малого та середнього розміру.

Реалізована система демонструє ефективність використання хмарних технологій для корпоративних додатків. Використання безкоштовних тарифів MongoDB Atlas та Render.com дозволяє впровадити систему без початкових фінансових інвестицій. Автоматичне розгортання з GitHub мінімізує ризик людських помилок.

Результати тестування підтвердили відповідність системи вимогам. Кросбраузерне та мобільне тестування підтвердило коректну роботу на різних платформах. Час відгуку серверу не перевищує визначених обмежень.

Мету роботи досягнуто: система розроблена, розгорнута і повністю відповідає поставленим вимогам. Всі заплановані завдання виконано – від аналізу предметної області до тестування готового продукту.

CorpLinks – це не навчальний прототип, а реально працюючий продукт, який вже зараз можна розгорнути на будь-якому підприємстві. Подальші напрямки розвитку: підключення до LDAP або OAuth для єдиного входу, удосконалення рекомендацій із застосуванням колаборативної фільтрації і додавання push-сповіщень для актуальних оновлень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гавриленко О. В., Петренко І. М. Архітектура та проєктування сучасних вебсистем. Київ : КПІ ім. Ігоря Сікорського, 2024. 312 с.
2. Мельник А. О. Проєктування корпоративних інформаційних систем [2]: підхід на базі мікросервісів. Львів : Видавництво Львівської політехніки, 2025. 280 с.
3. Вайганг Г. О., Голуб Б. Л. Методичні рекомендації щодо підготовки та оформлення бакалаврської кваліфікаційної роботи. Київ : НУБіП, 2026. 31 с.
4. Коваленко В. М. Сучасні фронтенд-технології: екосистема React. Харків : ХНУРЕ, 2024. 256 с.
5. Сидоренко О. І. Розробка високонавантажених бекенд-сервісів на Node.js та Express. Київ : НАУ, 2025. 210 с.
6. Лисенко С. М., Грицюк Ю. І. Кібербезпека та управління доступом у вебзастосунках. Львів : Новий світ-2000, 2025. 198 с.
7. Ткачук Р. В. Глибоке вивчення JavaScript: від основ до асинхронного програмування. Дніпро : ДНУ, 2024. 305 с.
8. Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide. 4th ed. O'Reilly Media, 2024. 530 p.
9. Cantelon M., Harter M. Node.js in Action. 3rd ed. Manning Publications, 2024. 450 p.
10. Vohra C. MERN [10] Stack Web Development: Build and deploy full-stack applications. 2nd ed. Packt Publishing, 2023. 412 p.
11. Single-page application (SPA). *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 10.04.2026).
12. Express routing & middleware. *Express Node.js Framework*. URL: https://expressjs.com/en/guide/routing.html (дата звернення: 12.04.2026).
13. MongoDB Aggregation Pipeline. *MongoDB Documentation*. URL: https://www.mongodb.com/docs/manual/core/aggregation-pipeline/ (дата звернення: 14.04.2026).
14. Mongoose: Object Data Modeling (ODM) for Node.js. *Mongoose Documentation*. URL: https://mongoosejs.com/docs/guide.html (дата звернення: 16.04.2026).
15. Role-Based Access Control (RBAC). *Auth0 Documentation*. URL: https://auth0.com/docs/manage-users/access-control/rbac (дата звернення: 18.04.2026).
16. Node.js Asynchronous, Non-blocking I/O. *Node.js Official Docs*. URL: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/ (дата звернення: 22.04.2026).
17. Deploy a Node.js Web Service. *Render Documentation*. URL: https://docs.render.com/deploy-node (дата звернення: 25.04.2026).

18.15. What is REST? *REST API Tutorial*. URL: https://restfulapi.net/ (дата звернення: 27.04.2026).

Додаток А

Лістинг серверного модуля

Серверний модуль server.js є точкою входу веб-застосунку CorpLinks. Він ініціалізує Express-сервер, підключає middleware для обробки CORS-запитів та JSON-даних, реєструє маршрути для всіх предметних сутностей системи та забезпечує обслуговування статичних файлів клієнтської частини. Окремо визначено ендпоінти авторизації та зміни пароля.

Лістинг А.1 – Серверний модуль (server.js)

```
const express = require('express');
const cors = require('cors');
const path = require('path');
const db = require('./db');
const app = express();
const PORT = process.env.PORT || 10000;

app.use(cors());
app.use(express.json());
app.use(express.static(path.join(__dirname)));

// API МАРШРУТИ (ОБОВ'ЯЗКОВО ПЕРЕД app.get('*'))
app.use('/api/users',        require('./routes/users'));
app.use('/api/tickets',      require('./routes/tickets'));
app.use('/api/resources',    require('./routes/resources'));
app.use('/api/logs',         require('./routes/logs'));
app.use('/api/accounting',   require('./routes/accounting'));
app.use('/api/contractors',  require('./routes/contractors'));
app.use('/api/hr',           require('./routes/hr'));       // Додано для HR
app.use('/api/contacts',     require('./routes/contacts')); // Додано для Контактів

app.post('/api/auth/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await db.loginUser(email, password);
    await db.addLog('Вхід в систему', user.name);
    res.json({ success: true, user });
  } catch (err) {
    res.status(401).json({ error: err.message });
  }
});
app.post('/api/auth/change-password', async (req, res) => {
  try {
    var { userId, oldPassword, newPassword } = req.body;
```

```
    await db.changePassword(userId, oldPassword, newPassword);
    await db.addLog('Змінено пароль', 'Система');
    res.json({ success: true });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});
app.get('/api/stats', async (req, res) => res.json(await db.getStats()));

// Кінцева точка для SPA
app.get('*', (req, res) => res.sendFile(path.join(__dirname, 'index.html')));

app.listen(PORT, () => console.log(`🚀 Cloud Server running on port ${PORT}`));
```

Додаток Б

Лістинг модуля доступу до даних

Модуль db.js реалізує патерн "Репозиторій" та інкапсулює всю логіку взаємодії з базою даних MongoDB Atlas. Він визначає десять Mongoose-схем для колекцій бази даних та експортує об'єкт з асинхронними методами для виконання операцій CRUD, авторизації, агрегації рекомендацій на основі кліків та формування статистики для адміністративної панелі.

Лістинг Б.1 – Модуль доступу до даних (db.js)

```
const mongoose = require('mongoose');

const MONGO_URI =
"mongodb+srv://kott:24861980qwerty@cluster0.dowxxbi.mongodb.net/corplinks?retryWrites=true&w=majority&a
ppName=Cluster0";

mongoose.connect(MONGO_URI)
  .then(() => console.log('Connected to MongoDB Atlas'))
  .catch(err => console.error('Connection Error:', err));

var userSchema = new mongoose.Schema({
  name: String, email: String, dept: String, role: String,
  password: String, created_at: String
});
var ticketSchema = new mongoose.Schema({
  title: String, cat: String, priority: String, desc: String,
  status: String, author: String, authorId: String, dept: String,
  assignedTo: String, assignedToName: String, createdAt: String
});
var logSchema = new mongoose.Schema({ action: String, user_name: String, created_at: String });
var resourceSchema = new mongoose.Schema({ name: String, url: String, cat: String, desc: String, access: String,
created_at: String });
var accountingSchema = new mongoose.Schema({ title: String, amount: String, description: String, created_at:
String });
var contractorSchema = new mongoose.Schema({ company: String, phone: String, service: String, docType: String,
amount: String, created_at: String });
var hrSchema = new mongoose.Schema({ title: String, url: String, description: String, created_at: String });
var contactSchema = new mongoose.Schema({ name: String, position: String, phone: String, email: String,
created_at: String });
var clickSchema = new mongoose.Schema({ userId: String, resourceId: String, userDept: String, timestamp: { type:
Date, default: Date.now } });
var favoriteSchema = new mongoose.Schema({ userId: String, resourceId: String });

var User = mongoose.model('User', userSchema);
```

```
var Ticket = mongoose.model('Ticket', ticketSchema);
var Log = mongoose.model('Log', logSchema);
var Resource = mongoose.model('Resource', resourceSchema);
var Accounting = mongoose.model('Accounting', accountingSchema);
var Contractor = mongoose.model('Contractor', contractorSchema);
var HR = mongoose.model('HR', hrSchema);
var Contact = mongoose.model('Contact', contactSchema);
var Click = mongoose.model('Click', clickSchema);
var Favorite = mongoose.model('Favorite', favoriteSchema);


var db = {
  async loginUser(email, password) {
    var user = await User.findOne({ email: email.toLowerCase().trim(), password: password });
    if (!user) throw new Error('Невірний email або пароль');
    return user;
  },

  async getUsers() { return await User.find().sort({ name: 1 }); },

  async createUser(data) {
    if (!data.password || data.password.length < 4) throw new Error('Пароль мінімум 4 символи');
    var exists = await User.findOne({ email: data.email.toLowerCase().trim() });
    if (exists) throw new Error('Цей email вже зайнятий');
    return await new User({ name: data.name, email: data.email.toLowerCase().trim(), dept: data.dept, role: data.role,
password: data.password, created_at: new Date().toLocaleString('uk-UA') }).save();
  },

  async updateUser(mongoId, data) {
    var update = { name: data.name, email: data.email, dept: data.dept, role: data.role };
    if (data.password && data.password.length >= 4) update.password = data.password;
    var user = await User.findByIdAndUpdate(mongoId, update, { new: true });
    if (!user) throw new Error('Користувача не знайдено');
    return user;
  },

  async deleteUser(mongoId) {
    var user = await User.findByIdAndDelete(mongoId);
    if (!user) throw new Error('Не знайдено');
    await Click.deleteMany({ userId: mongoId });
    await Favorite.deleteMany({ userId: mongoId });
    return user;
  },

  async changePassword(mongoId, oldPassword, newPassword) {
    var user = await User.findById(mongoId);
    if (!user) throw new Error('Користувача не знайдено');
    if (user.password !== oldPassword) throw new Error('Старий пароль невірний');
    if (!newPassword || newPassword.length < 4) throw new Error('Новий пароль мінімум 4 символи');
    user.password = newPassword;
    await user.save();
    return user;
  },

  async getResources() { return await Resource.find().sort({ _id: -1 }); },

  async getResourcesForUser(userId) {
    var user = await User.findById(userId);
    var allResources = await Resource.find().sort({ _id: -1 });
    if (!user) return allResources;
```

```
    if (user.role === 'admin') return allResources;
    return allResources.filter(function(r) {
      var access = (r.access || 'ALL').toString();
      if (access === 'ALL') return true;
      if (access.indexOf(user.dept) !== -1) return true;
      if (user.role === 'manager' && access.indexOf('Management') !== -1) return true;
      return false;
    });
  },

  async createResource(data) {
    return await new Resource({ name: data.name, url: data.url, cat: data.cat, desc: data.desc, access: data.access ||
'ALL', created_at: new Date().toLocaleString('uk-UA') }).save();
  },

  async updateResource(mongoId, data) {
    var res = await Resource.findByIdAndUpdate(mongoId, { name: data.name, url: data.url, cat: data.cat, desc:
data.desc, access: data.access }, { new: true });
    if (!res) throw new Error('Ресурс не знайдено');
    return res;
  },

  async deleteResource(mongoId) {
    await Click.deleteMany({ resourceId: mongoId });
    await Favorite.deleteMany({ resourceId: mongoId });
    return await Resource.findByIdAndDelete(mongoId);
  },

  async getFavorites(userId) { var favs = await Favorite.find({ userId: userId }); return favs.map(function(f) { return
f.resourceId; }); },

  async toggleFavorite(userId, resourceId) {
    var existing = await Favorite.findOne({ userId: userId, resourceId: resourceId });
    if (existing) { await Favorite.findByIdAndDelete(existing._id); return false; }
    else { await new Favorite({ userId: userId, resourceId: resourceId }).save(); return true; }
  },

  async trackClick(userId, resourceId, userDept) { await new Click({ userId: userId, resourceId: resourceId,
userDept: userDept }).save(); },

  async getFrequentForUser(userId) {
    return await Click.aggregate([{ $match: { userId: userId } }, { $group: { _id: '$resourceId', count: { $sum: 1 } } },
{ $sort: { count: -1 } }, { $limit: 5 }]);
  },

  async getPopularInDept(dept) {
    return await Click.aggregate([{ $match: { userDept: dept } }, { $group: { _id: '$resourceId', count: { $sum: 1 } }
}, { $sort: { count: -1 } }, { $limit: 5 }]);
  },

  async getRecommendations(userId) {
    var user = await User.findById(userId);
    if (!user) return { personal: [], department: [], userDept: '' };
    var personal = await this.getFrequentForUser(userId);
    var department = await this.getPopularInDept(user.dept);
    var allIds = [];
    personal.forEach(function(p) { allIds.push(p._id); });
    department.forEach(function(d) { allIds.push(d._id); });
    var resources = await Resource.find({ _id: { $in: allIds } });
```

```
    var resMap = {};
    resources.forEach(function(r) { resMap[r._id.toString()] = r; });
    var personalIds = personal.map(function(p) { return p._id; });
    var personalFull = personal.map(function(p) { var r = resMap[p._id]; return r ? { resource: r, clicks: p.count } :
null; }).filter(Boolean);
    var departmentFull = department.filter(function(d) { return personalIds.indexOf(d._id) === -1;
}).map(function(d) { var r = resMap[d._id]; return r ? { resource: r, clicks: d.count } : null; }).filter(Boolean);
    return { personal: personalFull, department: departmentFull, userDept: user.dept };
  },

  async getTickets() { return await Ticket.find().sort({ _id: -1 }); },

  async createTicket(data) {
    return await new Ticket({ title: data.title, cat: data.cat, priority: data.priority, desc: data.desc, status: 'new', author:
data.author, authorId: data.authorId, dept: data.dept, assignedTo: data.assignedTo || '', assignedToName:
data.assignedToName || 'Не призначено', createdAt: new Date().toLocaleString('uk-UA') }).save();
  },

  async updateTicketStatus(mongoId, status) { return await Ticket.findByIdAndUpdate(mongoId, { status: status },
{ new: true }); },
  async updateTicketAssignee(mongoId, assignedTo, assignedToName) { return await
Ticket.findByIdAndUpdate(mongoId, { assignedTo: assignedTo, assignedToName: assignedToName }, { new: true
}); },
  async deleteTicket(mongoId) { return await Ticket.findByIdAndDelete(mongoId); },

  async getAccounting() { return await Accounting.find().sort({ _id: -1 }); },
  async createAccounting(data) { return await new Accounting({ title: data.title, amount: data.amount, description:
data.description || '', created_at: new Date().toLocaleString('uk-UA') }).save(); },
  async updateAccounting(mongoId, data) { return await Accounting.findByIdAndUpdate(mongoId, { title:
data.title, amount: data.amount, description: data.description }, { new: true }); },
  async deleteAccounting(mongoId) { return await Accounting.findByIdAndDelete(mongoId); },

  async getContractors() { return await Contractor.find().sort({ company: 1 }); },
  async createContractor(data) { return await new Contractor({ company: data.company, phone: data.phone, service:
data.service || '', docType: data.docType || 'general', amount: data.amount || '', created_at: new
Date().toLocaleString('uk-UA') }).save(); },
  async updateContractor(mongoId, data) { return await Contractor.findByIdAndUpdate(mongoId, { company:
data.company, phone: data.phone, service: data.service, docType: data.docType || 'general', amount: data.amount || ''
}, { new: true }); },
  async deleteContractor(mongoId) { return await Contractor.findByIdAndDelete(mongoId); },

  async getHR() { return await HR.find().sort({ _id: -1 }); },
  async createHR(data) { return await new HR({ title: data.title, url: data.url, description: data.description || '',
created_at: new Date().toLocaleString('uk-UA') }).save(); },
  async updateHR(mongoId, data) { return await HR.findByIdAndUpdate(mongoId, { title: data.title, url: data.url,
description: data.description }, { new: true }); },
  async deleteHR(mongoId) { return await HR.findByIdAndDelete(mongoId); },

  async getContacts() { return await Contact.find().sort({ name: 1 }); },
  async createContact(data) { return await new Contact({ name: data.name, position: data.position, phone:
data.phone, email: data.email, created_at: new Date().toLocaleString('uk-UA') }).save(); },
  async deleteContact(mongoId) { return await Contact.findByIdAndDelete(mongoId); },

  async getLogs() { return await Log.find().sort({ _id: -1 }).limit(100); },
  async addLog(action, userName) { await new Log({ action: action, user_name: userName || 'Система', created_at:
new Date().toLocaleString('uk-UA') }).save(); },

  async getStats() {
```

```
    var results = await Promise.all([User.countDocuments(), Resource.countDocuments(), Ticket.countDocuments(),
User.countDocuments({ role: 'admin' }), User.countDocuments({ role: 'manager' }), Click.countDocuments()]);
    return { totalUsers: results[0], totalResources: results[1], totalTickets: results[2], adminCount: results[3],
managerCount: results[4], totalClicks: results[5] };
  }
};

module.exports = db;
```

Додаток В

Лістинг маршрутів ресурсів та IT-заявок

Маршрути ресурсів забезпечують повний цикл управління корпоративними посиланнями: отримання списку з фільтрацією за роллю, створення, редагування та видалення ресурсів, а також роботу з обраними, трекінг кліків та формування персоналізованих рекомендацій. Маршрути IT-заявок реалізують створення заявок, зміну статусу, призначення виконавця з автоматичною відправкою email-сповіщень та видалення.

Лістинг В.1 – Маршрути ресурсів (routes/resources.js)

```
const express = require('express');
const router = express.Router();
const db = require('../db');

router.get('/', async (req, res) => {
  try {
    if (req.query.userId) {
      res.json(await db.getResourcesForUser(req.query.userId));
    } else {
      res.json(await db.getResources());
    }
  } catch (e) { res.status(500).json({ error: e.message }); }
});

router.get('/recommendations/:userId', async (req, res) => {
  try {
    res.json(await db.getRecommendations(req.params.userId));
  } catch (e) { res.status(500).json({ error: e.message }); }
});

router.get('/favorites/:userId', async (req, res) => {
  try {
    res.json(await db.getFavorites(req.params.userId));
  } catch (e) { res.status(500).json({ error: e.message }); }
});

router.post('/:id/favorite', async (req, res) => {
  try {
    var result = await db.toggleFavorite(req.body.userId, req.params.id);
    res.json({ isFavorite: result });
  } catch (e) { res.status(400).json({ error: e.message }); }
});
```

```
router.post('/:id/click', async (req, res) => {
  try {
    await db.trackClick(req.body.userId, req.params.id, req.body.userDept);
    res.json({ success: true });
  } catch (e) { res.status(400).json({ error: e.message }); }
});

router.post('/', async (req, res) => {
  try {
    var r = await db.createResource(req.body);
    await db.addLog('Додано ресурс: ' + r.name, req.body.userName || 'Система');
    res.json(r);
  } catch (e) { res.status(400).json({ error: e.message }); }
});

router.put('/:id', async (req, res) => {
  try {
    var r = await db.updateResource(req.params.id, req.body);
    await db.addLog('Оновлено ресурс: ' + r.name, req.body.userName || 'Система');
    res.json(r);
  } catch (e) { res.status(400).json({ error: e.message }); }
});

router.delete('/:id', async (req, res) => {
  try {
    await db.deleteResource(req.params.id);
    res.json({ success: true });
  } catch (e) { res.status(400).json({ error: e.message }); }
});

module.exports = router;
```

Лістинг В.2 – Маршрути IT-заявок (routes/tickets.js)

```
const express = require('express');
const router = express.Router();
const db = require('../db');
const { sendMail } = require('../mailer');

router.get('/', async (req, res) => {
  try { res.json(await db.getTickets()); }
  catch (e) { res.status(500).json({ error: e.message }); }
});

router.post('/', async (req, res) => {
  try {
    var ticket = await db.createTicket(req.body);
    await db.addLog('Нова заявка: ' + ticket.title, req.body.author || 'Система');

    if (ticket.assignedTo) {
      var users = await db.getUsers();
      var assignee = users.find(u => u._id.toString() === ticket.assignedTo);
      if (assignee) {
        sendMail(assignee.email,
          'Нова заявка: ' + ticket.title,
```

```
        '<div style="font-family:Arial,sans-serif;max-width:600px;padding:20px">' +
        '<h2 style="color:#2563eb;margin-bottom:20px">Вам призначено нову заявку</h2>' +
        '<table style="border-collapse:collapse;width:100%">' +
        '<tr><td style="padding:10px 12px;border-bottom:1px solid
#e5e7eb;color:#6b7280;width:120px">Тема</td><td style="padding:10px 12px;border-bottom:1px solid
#e5e7eb;font-weight:bold">' + ticket.title + '</td></tr>' +
        '<tr><td style="padding:10px 12px;border-bottom:1px solid #e5e7eb;color:#6b7280">Категорія</td><td
style="padding:10px 12px;border-bottom:1px solid #e5e7eb">' + ticket.cat + '</td></tr>' +
        '<tr><td style="padding:10px 12px;border-bottom:1px solid #e5e7eb;color:#6b7280">Пріоритет</td><td
style="padding:10px 12px;border-bottom:1px solid #e5e7eb;font-weight:bold;color:' + (ticket.priority === 'high' ?
'#dc2626' : ticket.priority === 'medium' ? '#ca8a04' : '#16a34a') + '">' + ticket.priority + '</td></tr>' +
        '<tr><td style="padding:10px 12px;border-bottom:1px solid #e5e7eb;color:#6b7280">Від кого</td><td
style="padding:10px 12px;border-bottom:1px solid #e5e7eb">' + ticket.author + ' (' + ticket.dept + ')</td></tr>' +
        '<tr><td style="padding:10px 12px;color:#6b7280">Опис</td><td style="padding:10px 12px">' +
(ticket.desc || 'Без опису') + '</td></tr>' +
        '</table>' +
        '<p style="margin-top:24px;padding-top:16px;border-top:1px solid #e5e7eb;color:#9ca3af;font-
size:12px">Цей лист надіслано автоматично системою CorpLinks</p></div>'
       );
     }
   }


   res.json(ticket);
  } catch (e) { res.status(400).json({ error: e.message }); }
});

router.put('/:id/status', async (req, res) => {
  try {
    var ticket = await db.updateTicketStatus(req.params.id, req.body.status);
    await db.addLog('Статус заявки "' + ticket.title + '": ' + req.body.status, req.body.adminName || 'IT');


    if (ticket.authorId) {
      var users = await db.getUsers();
      var author = users.find(u => u._id.toString() === ticket.authorId);
      var statusLabels = { new: 'Нова', inprogress: 'В роботі', done: 'Виконано' };
      var statusColors = { new: '#2563eb', inprogress: '#ca8a04', done: '#16a34a' };
      if (author) {
        sendMail(author.email,
          'Заявка "' + ticket.title + '" — ' + (statusLabels[req.body.status] || req.body.status),
          '<div style="font-family:Arial,sans-serif;max-width:600px;padding:20px">' +
          '<h2 style="color:#059669;margin-bottom:20px">Статус вашої заявки змінено</h2>' +
          '<p style="font-size:16px;margin-bottom:16px"><strong>' + ticket.title + '</strong></p>' +
          '<p>Новий статус: <span style="display:inline-block;padding:6px 16px;border-radius:6px;background:' +
(statusColors[req.body.status] || '#666') + '18;color:' + (statusColors[req.body.status] || '#666') + ';font-
weight:bold;border:1px solid ' + (statusColors[req.body.status] || '#666') + '33">' + (statusLabels[req.body.status] ||
req.body.status) + '</span></p>' +
          (ticket.assignedToName && ticket.assignedToName !== 'Не призначено' ? '<p style="margin-
top:12px;color:#6b7280">Виконавець: ' + ticket.assignedToName + '</p>' : '') +
          '<p style="margin-top:24px;padding-top:16px;border-top:1px solid #e5e7eb;color:#9ca3af;font-
size:12px">Цей лист надіслано автоматично системою CorpLinks</p></div>'
        );
      }
    }


    res.json(ticket);
  } catch (e) { res.status(400).json({ error: e.message }); }
});

router.put('/:id/assign', async (req, res) => {
  try {
```

```
    var ticket = await db.updateTicketAssignee(req.params.id, req.body.assignedTo, req.body.assignedToName);
    await db.addLog('Заявку "' + ticket.title + '" призначено: ' + req.body.assignedToName, req.body.adminName ||
'Адмін');

    var users = await db.getUsers();
    var assignee = users.find(u => u._id.toString() === req.body.assignedTo);
    if (assignee) {
      sendMail(assignee.email,
        'Вам призначено заявку: ' + ticket.title,
        '<div style="font-family:Arial,sans-serif;max-width:600px;padding:20px">' +
        '<h2 style="color:#7c3aed;margin-bottom:20px">Вам призначено заявку</h2>' +
        '<p style="font-size:16px;margin-bottom:8px"><strong>' + ticket.title + '</strong></p>' +
        '<p style="color:#6b7280">Пріоритет: ' + ticket.priority + '</p>' +
        '<p style="color:#6b7280">Від: ' + ticket.author + ' (' + ticket.dept + ')</p>' +
        '<p style="color:#6b7280">Опис: ' + (ticket.desc || 'Без опису') + '</p>' +
        '<p style="margin-top:24px;padding-top:16px;border-top:1px solid #e5e7eb;color:#9ca3af;font-
size:12px">Цей лист надіслано автоматично системою CorpLinks</p></div>'
      );
    }

    res.json(ticket);
  } catch (e) { res.status(400).json({ error: e.message }); }
});

router.delete('/:id', async (req, res) => {
  try { await db.deleteTicket(req.params.id); res.json({ success: true }); }
  catch (e) { res.status(400).json({ error: e.message }); }
});

module.exports = router;
```

Додаток Д

Лістинг модуля email-сповіщень

Модуль mailer.js забезпечує автоматичну відправку email-сповіщень при створенні IT-заявок, зміні їх статусу та призначенні виконавців. Для відправки використовується Resend API через HTTPS-запит. Корпоративні адреси працівників автоматично перенаправляються на реальні поштові скриньки через маппінг EMAIL_MAP.

Лістинг Д.1 – Модуль email-сповіщень (mailer.js)

```
async function sendMail(to, subject, html) {
  try {
    var EMAIL_MAP = {
      'khodakivskyi@enterprise.com': 'igorhodaskovskiy@gmail.com',
      'petrenko@enterprise.com':     'igorhodaskovskiy@gmail.com',
      'koval@enterprise.com':        'igorhodaskovskiy@gmail.com',
      'bondarenko@enterprise.com':   'igorhodaskovskiy@gmail.com',
      'shevchenko@enterprise.com':   'igorhodaskovskiy@gmail.com',
      'tarasenko@enterprise.com':    'igorhodaskovskiy@gmail.com',
      'sydorenko@enterprise.com':    'igorhodaskovskiy@gmail.com',
      'yanenko@enterprise.com':      'igorhodaskovskiy@gmail.com',
      'lytvynenko@enterprise.com':   'igorhodaskovskiy@gmail.com',
      'moroz@enterprise.com':        'igorhodaskovskiy@gmail.com',
    };

    var realEmail = EMAIL_MAP[to] || to;
    if (!realEmail || realEmail.indexOf('@') === -1) return;

    var res = await fetch('https://api.resend.com/emails', {
      method: 'POST',
      headers: {
        'Authorization': 'Bearer re_jAExJ7Hr_kA1YUwJ2gqKDq9LU2P9wAWcm',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        from: 'CorpLinks <onboarding@resend.dev>',
        to: realEmail,
        subject: subject,
        html: html
      })
    });

    var data = await res.json();
    console.log('Email sent: ' + to + ' -> ' + realEmail, data);
  } catch (err) {
    console.error('Email error:', err.message);
  }
}
```

```
module.exports = { sendMail };
```

Додаток Е

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій
Декларація про академічну доброчесність та використання ШІ

Я, Ходаківський Ігор Євгенійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення бази посилань на робочі ресурси працівників підприємства» є результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

3. Щодо використання інструментів ШІ зазначаю:

Інструменти ШІ (Claude, Gemini) були використані для допомоги у відлагодженні деяких частин програмного коду, перевірки граматики, та пошуку деяких джерел.

Усі тексти та рішення були ретельно перевірені, проаналізовані та відредаговані автором роботи.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____ 2026 р. _____ Ігор ХОДАКІВСЬКИЙ