

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ ОПРАЦЮВАННЯ
ЗАПИСІВ КОРИСТУВАЧІВ
DESIGN OF A SOFTWARE SYSTEM FOR USER
RECORD PROCESSING

Виконав(ла): студент(ка) 4 курсу, групи КНТ-212

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

КОМІСАРЕНКО М.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент БАБЕНКО Н.В.

(ПРИЗВИЩЕ та ініціали)

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

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

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. СУББОТІН
« » 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КОМІСАРЕНКА Микити Вікторовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Створення програмної системи опрацювання записів користувачів. Design of a Software System for User Record Processing

керівник проєкту (роботи) д.т.н., професор ПОЛЯКОВ Михайло Олексійович,

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

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 01 червня 2026 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПОЛЯКОВ М.О., професор		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання «20» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

Микита КОМІСАРЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

Михайло ПОЛЯКОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
99 с., 5 табл., 27 рис., 4 дод., 25 джерел.

ЗАПИС КОРИСТУВАЧА, ОПРАЦЮВАННЯ ЗАПИСІВ, ВЕБ-ДОДАТОК, REST API, БАЗА ДАНИХ, TYPESCRIPT, REACT, РОЗМЕЖУВАННЯ ДОСТУПУ.

Об'єкт дослідження — процес створення програмного забезпечення для автоматизації опрацювання записів користувачів організації.

Мета роботи — дослідження та розробка програмної системи опрацювання записів користувачів, що забезпечує централізований облік, пошук, опрацювання та захист даних.

У роботі проаналізовано предметну область та наявні програмні рішення, обґрунтовано вибір технологій, спроектовано й розроблено програмну систему у вигляді односторінкового веб-додатку з REST API та реляційною базою даних, а також виконано її тестування. Систему реалізовано мовою TypeScript із використанням бібліотеки React, платформи Node.js із каркасом Express та бази даних SQLite.

Розроблена система реалізує реєстрацію та авторизацію користувачів, розмежування доступу за ролями, повний цикл операцій над записами, пошук, фільтрацію, експорт у CSV та формування статистики.

ABSTRACT

Explanatory note to the bachelor's diploma qualification work: 99 pages, 5 tables, 27 figures, 4 appendices, 25 references.

USER RECORD, RECORDS PROCESSING, WEB APPLICATION, REST API, DATABASE, TYPESCRIPT, REACT, ACCESS CONTROL.

The object of research is the process of creating software for automating the processing of user records of an organization.

The aim of the work is the research and development of a software system for processing user records that provides centralized accounting, search, processing and protection of data.

The subject area and existing software solutions were analyzed, the choice of technologies was substantiated, and a software system was designed and developed as a single-page web application with a REST API and a relational database, followed by its testing. The system is implemented in the TypeScript language using the React library, the Node.js platform with the Express framework, and the SQLite database.

The developed system implements user registration and authorization, role-based access control, a full cycle of operations on records, search, filtering, export to CSV, and statistics generation.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	8
Вступ	9
1 Аналіз предметної області	13
1.1 Поняття запису користувача та опрацювання записів	13
1.2 Способи зберігання та опрацювання записів користувачів	15
1.3 Поняття систем для управління записами користувачів	17
1.4 Порівняння наявного програмного забезпечення для управління записами користувачів	19
1.5 Висновки за розділом 1	26
2 Матеріали і методи	27
2.1 Вибір мови програмування	27
2.2 Вибір фреймворку та технологій для розробки	31
2.3 Висновки за розділом 2	35
3 Опис програми	37
3.1 Структура програмного забезпечення	37
3.2 Основний алгоритм роботи програмного забезпечення	40
3.3 Проєктування бази даних	44
3.4 Проєктування інтерфейсу	46
3.5 Висновки за розділом 3	48
4 Експлуатація, тестування та експериментальне дослідження програми	50
4.1 Призначення й умови застосування програми	50
4.2 Умови виконання програми	52
4.3 Виконання програмного забезпечення	52
4.4 Тестування програмного забезпечення	58
4.5 Висновки за розділом 4	60
Висновки	61
Перелік джерел посилання	64

Додаток А Технічне завдання	66
Додаток Б Опис програми	70
Додаток В Текст програми	75
Додаток Г Слайди презентації	93

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface (програмний інтерфейс застосунку);

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення);

CSV – Comma-Separated Values (значення, розділені комами);

JWT – JSON Web Token (веб-токен формату JSON);

RBAC – Role-Based Access Control (розмежування доступу за ролями);

REST – Representational State Transfer (архітектурний стиль вебсервісів);

SPA – Single Page Application (односторінковий застосунок);

SQL – Structured Query Language (мова структурованих запитів);

TS – TypeScript;

БД – база даних;

ОС – операційна система;

ПЗ – програмне забезпечення.

ВСТУП

У сучасних умовах діяльність будь-якої організації супроводжується постійним зростанням обсягів даних про користувачів, співробітників та клієнтів. Традиційні способи ведення обліку, що ґрунтуються на паперових журналах та розрізнених електронних таблицях, дедалі гірше відповідають вимогам до швидкості та якості опрацювання інформації. Такі підходи призводять до численних помилок під час введення даних, дублювання записів та втрати актуальності відомостей. Окремою проблемою є відсутність належного розмежування доступу, унаслідок чого конфіденційна інформація може бути доступною широкому колу осіб без обмежень. Через це постає потреба у переході від ручного обліку до автоматизованих засобів централізованого опрацювання записів користувачів.

Актуальність роботи зумовлена об'єктивною необхідністю впровадження автоматизованих систем опрацювання записів, здатних забезпечити цілісність та достовірність даних. Автоматизація дає змогу пришвидшити типові операції створення, перегляду, редагування та видалення записів, а також організувати ефективний пошук і фільтрацію значних масивів інформації. Сучасні веб-технології уможливають побудову таких систем у вигляді доступних через браузер додатків, що не потребують встановлення спеціалізованого програмного забезпечення на робочих місцях. Запровадження рольового розмежування доступу підвищує рівень захищеності та впорядкованості роботи з даними. Усе це робить розробку відповідної програмної системи своєчасною та практично затребуваною.

Об'єктом дослідження є процес створення програмного забезпечення для автоматизації опрацювання записів користувачів організації. Розглядаються етапи аналізу предметної області, проектування, реалізації та тестування програмної системи. Особливу увагу приділено забезпеченню цілісності даних, зручності взаємодії з користувачем та розмежуванню доступу

за ролями. Дослідження охоплює як серверну, так і клієнтську складові програмного продукту.

Предметом дослідження є програмні засоби, технології та методи, що застосовуються для автоматизації опрацювання записів користувачів. До них належать веб-технології побудови односторінкових застосунків, архітектурний стиль REST для організації обміну даними та реляційні бази даних для надійного зберігання інформації. Окремо проаналізовано методи автентифікації та авторизації, а також підходи до валідації вхідних даних. Сукупність зазначених засобів становить технологічну основу проєктованої системи.

Метою роботи є дослідження та розробка програмної системи опрацювання записів користувачів, яка автоматизує облік і забезпечує централізований доступ до актуальної інформації. Розроблювана система має реалізувати повний цикл операцій над записами, пошук і фільтрацію, експорт даних та формування статистичних показників. Передбачено розмежування повноважень користувачів за ролями адміністратора й оператора. Досягнення поставленої мети має продемонструвати ефективність застосування сучасних веб-технологій для розв'язання прикладних задач обліку.

Для досягнення поставленої мети визначено такі завдання:

- проаналізувати предметну область та особливості опрацювання записів користувачів;
- здійснити огляд наявного програмного забезпечення відповідного призначення;
- обґрунтувати вибір мови програмування та технологій реалізації;
- спроектувати структуру програмної системи та схему бази даних;
- розробити програмне забезпечення серверної та клієнтської частин;
- виконати тестування розробленої системи та перевірити її відповідність вимогам.

Під час виконання роботи використано низку сучасних матеріалів, методів і технологій програмної розробки. Як основну мову програмування для

серверної та клієнтської частин обрано TypeScript [1], що забезпечує статичну типізацію та підвищує надійність коду. Клієнтську частину побудовано засобами бібліотеки React [2], а серверну реалізовано на платформі Node.js [3] із застосуванням каркаса Express [4]. Для зберігання даних застосовано реляційну базу даних SQLite [5], а взаємодію між клієнтом і сервером організовано за принципами REST API [6],[7] з обміном даними у форматі JSON. Захист доступу реалізовано на основі технології JWT [8] та хешування паролів.

Наукова новизна роботи полягає в обґрунтуванні та реалізації комплексного підходу до автоматизації опрацювання записів користувачів, що поєднує рольове розмежування доступу та REST-орієнтовану архітектуру в межах єдиного веб-додатку. Запропоноване рішення інтегрує засоби автентифікації, валідації даних і формування статистики в цілісну програмну систему. Поєднання типобезпечного стека на основі TypeScript із файловою реляційною базою даних спрощує розгортання та супровід продукту. Таким чином, сформовано узгоджену модель побудови систем опрацювання записів для організацій різного масштабу.

Практична цінність роботи визначається можливістю безпосереднього застосування розробленої системи в організаціях для обліку співробітників, клієнтів та інших категорій користувачів. Система дає змогу замінити розрізнені таблиці та паперові журнали єдиним інструментом централізованого опрацювання даних. Завдяки адаптивному інтерфейсу та підтримці пошуку, фільтрації й експорту вона спрощує повсякденну роботу з великими масивами записів. Розмежування доступу за ролями підвищує захищеність інформації та впорядковує дії користувачів.

У результаті виконання роботи розроблено програмну систему опрацювання записів користувачів у вигляді односторінкового веб-додатку з REST API та реляційною базою даних. Система реалізує автентифікацію, рольове розмежування доступу, повний цикл операцій над записами, пошук, фільтрацію, експорт та формування статистики. Сферою застосування продукту

є підрозділи організацій, що потребують надійного та зручного засобу обліку відомостей про користувачів. Отже, поставлену мету досягнуто, а розроблена система підтверджує доцільність використання сучасних веб-технологій для автоматизації опрацювання записів користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття запису користувача та опрацювання записів

Під записом користувача розуміють структуровану сукупність даних, що описують окрему особу в межах певної інформаційної системи або організації. Такий запис об'єднує сукупність атрибутів, кожен з яких характеризує особу з певного боку: ідентифікаційні дані (прізвище, ім'я та по батькові), контактні відомості (адреса електронної пошти, номер телефону), організаційні характеристики (відділ або підрозділ, посада), а також службові ознаки, зокрема статус облікового запису, дату народження, додаткові нотатки та позначки про час створення й останнього оновлення. На відміну від неструктурованого тексту, запис користувача має чітко визначену схему, що дає змогу зберігати, опрацьовувати та аналізувати дані одноманітно й передбачувано.

Сукупність однотипних записів формує реєстр або картотеку користувачів, яка є центральним інформаційним ресурсом багатьох організацій. Кожен запис у такому реєстрі є відносно самостійною одиницею обліку, проте записи можуть пов'язуватися між собою та з іншими сутностями системи, наприклад з обліковими записами осіб, які створили чи змінили відповідні дані. Завдяки структурованому поданню записи користувачів придатні для автоматизованого опрацювання засобами обчислювальної техніки, що принципово відрізняє сучасні інформаційні системи від традиційного паперового діловодства.

Опрацюванням записів користувачів називають сукупність операцій, спрямованих на введення, зберігання, перетворення, пошук та надання відомостей про осіб відповідно до потреб організації. Базовим набором таких операцій є створення, перегляд, редагування та видалення записів, які в англійській літературі позначають аббревіатурою CRUD (Create, Read, Update, Delete). Поряд із цими операціями опрацювання передбачає пошук записів за заданими критеріями, фільтрування та сортування за різними атрибутами,

групування й агрегування даних для отримання узагальнених показників, а також експорт відомостей у зовнішні формати для подальшого використання.

Невід'ємною складовою опрацювання записів є контроль доступу до даних. Оскільки записи користувачів зазвичай містять персональні дані, до них застосовують розмежування повноважень: різні категорії працівників отримують різні права на перегляд, зміну або видалення інформації. Це забезпечує конфіденційність та цілісність відомостей, запобігає несанкціонованим змінам і дає змогу простежувати, хто і коли вносив зміни до записів. Таким чином, опрацювання записів охоплює не лише маніпулювання даними, а й організаційно-технічні заходи щодо їх захисту.

Опрацювання записів користувачів є типовим завданням для широкого кола предметних областей. У сфері управління персоналом (HR) записи описують працівників організації, їхні посади, підрозділи та статус зайнятості. У системах управління взаємовідносинами з клієнтами (CRM) аналогічні записи відображають відомості про клієнтів і контрагентів. Державні реєстри оперують записами громадян, платників податків, суб'єктів господарювання тощо. Навчальні заклади ведуть записи студентів, викладачів і працівників, лікувальні установи — записи пацієнтів. Попри відмінності у предметному наповненні, усі ці системи розв'язують спільне завдання — упорядковане зберігання та опрацювання структурованих даних про осіб.

Узагальнюючи, запис користувача є базовою одиницею обліку особи, а опрацювання записів — комплексом операцій над сукупністю таких одиниць, що включає створення, перегляд, редагування, видалення, пошук, фільтрування, експорт даних та контроль доступу до них. Якість і зручність опрацювання записів безпосередньо впливають на ефективність діяльності організації, що зумовлює потребу в спеціалізованих програмних засобах для автоматизації цих процесів.

1.2 Способи зберігання та опрацювання записів користувачів

Способи зберігання та опрацювання записів користувачів зазнали тривалої еволюції разом із розвитком засобів обчислювальної техніки. Найдавнішою формою обліку були паперові журнали та картотеки, у яких відомості про осіб записувалися від руки або друкувалися на окремих картках і впорядковувалися за певною ознакою, найчастіше за абеткою. Перевагою такого підходу є простота, незалежність від технічних засобів і наочність. Водночас паперовий облік має істотні недоліки: повільний пошук, складність унесення змін, ризик фізичної втрати чи пошкодження документів, неможливість одночасної роботи кількох осіб з одним масивом даних та відсутність будь-якого автоматизованого опрацювання.

Із поширенням персональних комп'ютерів облік записів частково перейшов у середовище електронних таблиць, насамперед табличного процесора Microsoft Excel. Електронні таблиці дають змогу впорядковувати записи в рядках і стовпцях, виконувати елементарні пошук, сортування та фільтрування, а також прості обчислення. Цей спосіб є доступним і не потребує спеціальних навичок програмування. Проте у разі зростання обсягу даних електронні таблиці виявляють низку обмежень: відсутність контролю цілісності та зв'язків між даними, складність розмежування доступу, схильність до випадкових помилок під час ручного введення, а також труднощі організації одночасної багатокористувацької роботи без ризику конфліктів і дублювання.

Наступним щаблем стали файлові, або настільні, бази даних, типовим представником яких є Microsoft Access. Такі системи поєднують реляційну модель даних із засобами побудови форм, запитів і звітів, що дає змогу організувати структуроване зберігання записів та зручне введення даних. Файлові бази підтримують зв'язки між таблицями та контроль типів даних, проте зберігають увесь масив в одному файлі, що погіршує надійність і продуктивність у разі значних обсягів інформації та одночасної роботи багатьох

користувачів. Вони слабо пристосовані до розгортання в розподіленому мережевому середовищі.

Для подолання обмежень файлових баз застосовують клієнт-серверні системи керування базами даних, такі як Microsoft SQL Server, PostgreSQL, MySQL або Oracle Database. У цій моделі дані зберігаються та опрацьовуються на виділеному сервері, а клієнтські застосунки звертаються до нього через мережу. Такий підхід забезпечує надійне багатокористувацьке опрацювання, контроль цілісності та транзакційності, розмежування прав доступу, резервне копіювання та масштабування. Недоліками є потреба в окремому сервері баз даних, складніше адміністрування та вищі вимоги до кваліфікації обслуговуючого персоналу.

Найбільш сучасним способом є спеціалізоване веб-програмне забезпечення, яке надає користувачам доступ до записів через браузер незалежно від операційної системи та розташування. Такі веб-додатки поєднують клієнт-серверну базу даних із зручним графічним інтерфейсом, реалізують увесь набір операцій опрацювання записів, розмежування доступу за ролями, валідацію вхідних даних та засоби захисту персональної інформації. Перевагами є кросплатформність, централізоване оновлення, доступність із будь-якого пристрою та можливість адаптації під конкретну предметну область. Як недолік відзначають потребу в постійному підключенні до мережі та залежність від коректної роботи серверної частини.

Кожен із розглянутих способів має власну сферу доцільного застосування, що визначається обсягом даних, кількістю користувачів, вимогами до безпеки та наявними ресурсами. Узагальнене порівняння переваг і недоліків паперового обліку, електронних таблиць, файлових баз даних, клієнт-серверних СУБД та спеціалізованого веб-програмного забезпечення наведено в таблиці 1.1. Аналіз свідчить, що для організації, якій потрібне багатокористувацьке, безпечне та зручне опрацювання записів користувачів, найдоцільнішим є застосування спеціалізованого веб-додатка на основі клієнт-серверної архітектури.

Таблиця 1.1 – Порівняння способів зберігання та опрацювання записів користувачів

Спосіб	Переваги	Недоліки
Паперові журнали, картотеки	Простота, незалежність від техніки	Повільний пошук, ризик втрати, відсутність багатокористувацької роботи
Електронні таблиці (Excel)	Доступність, прості сортування й фільтри	Немає контролю цілісності, складно розмежувати доступ, помилки введення
Файлові БД (MS Access)	Реляційна модель, форми та звіти	Один файл, низька надійність і продуктивність при масштабі
Клієнт-серверні СУБД	Багатокористувацькість, цілісність, безпека	Потрібен окремий сервер, складніше адміністрування
Спеціалізоване веб-ПЗ	Кросплатформність, зручний інтерфейс, RBAC	Потрібне інтернет-з'єднання, залежність від сервера

1.3 Поняття систем для управління записами користувачів

Системою управління записами користувачів називають програмний комплекс, призначений для централізованого зберігання, опрацювання та надання доступу до структурованих даних про осіб у межах організації. Така система є інструментом автоматизації облікових процесів, що замінює ручне ведення журналів і таблиць та забезпечує цілісність, узгодженість і захищеність відомостей. За своєю природою вона належить до класу інформаційних систем, орієнтованих на роботу з даними, і поєднує засоби введення, зберігання, обробки та подання інформації в єдиному середовищі.

У типовій системі управління записами користувачів вирізняють кілька складових. Підсистема зберігання даних, реалізована на основі бази даних, відповідає за надійне утримання записів та зв'язків між ними. Серверна частина (бекенд) реалізує бізнес-логіку, обробляє запити, виконує перевірку прав доступу та валідацію даних і надає програмний інтерфейс для взаємодії. Клієнтська частина (фронтенд) забезпечує графічний інтерфейс користувача,

через який здійснюються введення та перегляд інформації. Окремо виділяють підсистеми автентифікації та авторизації, що відповідають за вхід користувачів до системи та розмежування їхніх повноважень.

Ключові функції, які має реалізовувати сучасна система управління записами користувачів, можна узагальнити в такому переліку:

- надійне зберігання записів користувачів у структурованій базі даних із підтримкою цілісності та зв'язків між даними;
- повний набір операцій CRUD — створення, перегляд, редагування та видалення записів;
- пошук записів за заданими критеріями, а також фільтрування та сортування за різними атрибутами;
- розмежування доступу за ролями (RBAC) [9] із наданням різним категоріям користувачів відповідних повноважень;
- ведення журналу аудиту, що фіксує факт, час і авторство змін у записах;
- експорт вибірок даних у зовнішні формати, зокрема CSV, для подальшого використання;
- забезпечення безпеки персональних даних: хешування паролів, контроль автентифікації, валідація вхідних відомостей та захист від несанкціонованого доступу;
- формування узагальнених статистичних показників для аналітичних і управлінських потреб.

До систем управління записами користувачів висувають низку вимог, які поділяють на функціональні та нефункціональні. Функціональні вимоги визначають перелік операцій, що їх має виконувати система, тоді як нефункціональні стосуються якісних характеристик: надійності, продуктивності, зручності використання, масштабованості та захищеності. Особливе значення має забезпечення коректної роботи в багатокористувацькому режимі, адаптивність інформативного інтерфейсу до

різних пристроїв, а також прозоре інформування користувача про результати операцій та можливі помилки.

Оскільки записи користувачів містять персональні дані, важливою вимогою до відповідних систем є дотримання принципів захисту персональної інформації. Загальні принципи передбачають збирання лише необхідного обсягу даних, обмеження мети їх використання, забезпечення конфіденційності, точності та збереженості відомостей, а також надання особам контролю над власними даними. На рівні законодавства ці принципи закріплено в Загальному регламенті захисту даних Європейського Союзу (GDPR) [10] та в Законі України «Про захист персональних даних» [11]. Практична реалізація цих вимог у програмній системі досягається шляхом розмежування доступу, шифрування або хешування критичних даних, ведення аудиту та застосування захищених протоколів передавання інформації.

Отже, система управління записами користувачів є комплексним рішенням, що поєднує засоби зберігання, опрацювання та захисту структурованих даних про осіб із розмежуванням доступу та зручним інтерфейсом. Сукупність наведених складових, функцій і вимог формує основу для проєктування власної програмної системи та слугує критеріями для порівняльного аналізу наявних на ринку рішень.

1.4 Порівняння наявного програмного забезпечення для управління записами користувачів

Для обґрунтування доцільності розроблення власної системи доцільно розглянути наявні програмні рішення, що тією чи іншою мірою забезпечують управління записами користувачів. До огляду відібрано чотири поширені продукти, які представляють різні класи систем: хмарну службу керування ідентичностями, відкриту платформу управління доступом, відкриту платформу керування даними та веб-інструмент адміністрування баз даних. Для

кожного з рішень наведено короткий опис, ключові особливості, переваги та недоліки.

Microsoft Entra ID [12] (раніше відома як Azure Active Directory) — це хмарна служба керування ідентичностями та доступом, що входить до екосистеми Microsoft. Вона виконує роль централізованого каталогу користувачів організації, забезпечує єдиний вхід до численних застосунків, керування ролями та групами, а також багатофакторну автентифікацію та політики умовного доступу. Інтерфейс адміністрування Microsoft Entra ID наведено на рисунку 1.1.

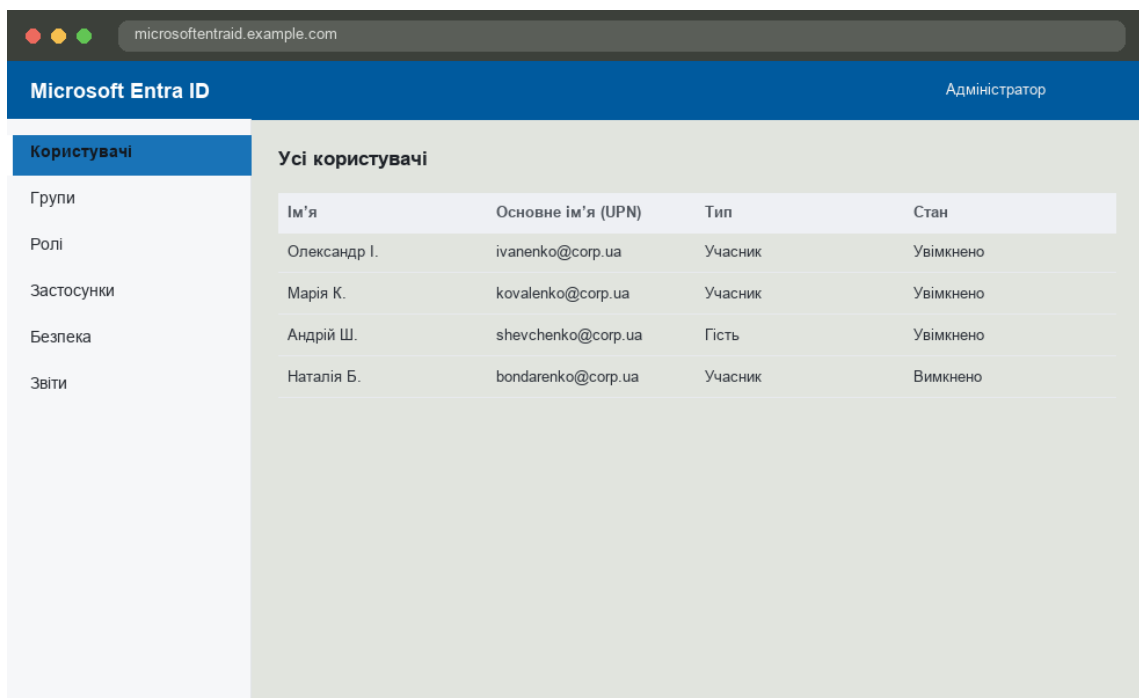


Рисунок 1.1 – Інтерфейс адміністрування Microsoft Entra ID

Ключовими особливостями Microsoft Entra ID є:

- централізований хмарний каталог облікових записів користувачів та їхніх атрибутів;
- керування ролями, групами та правами доступу до ресурсів;
- єдиний вхід (Single Sign-On) і багатофакторна автентифікація;
- інтеграція з хмарними та локальними сервісами Microsoft і сторонніми застосунками.

Перевагами рішення є висока надійність хмарної інфраструктури, розвинені засоби безпеки та глибока інтеграція з продуктами Microsoft. Недоліками є орієнтованість насамперед на керування ідентичностями та доступом, а не на довільні предметні записи, надмірна для невеликих організацій складність, прив'язаність до екосистеми постачальника та потреба у відповідній кваліфікації адміністраторів.

Keycloak [13] — це відкрите програмне рішення класу Identity and Access Management, що розповсюджується з відкритим кодом. Воно призначене для централізованого керування користувачами, ролями та автентифікацією і підтримує сучасні протоколи авторизації OAuth 2.0 та OpenID Connect, а також федерацію ідентичностей із зовнішніми постачальниками. Адміністративну консоль Keycloak показано на рисунку 1.2.

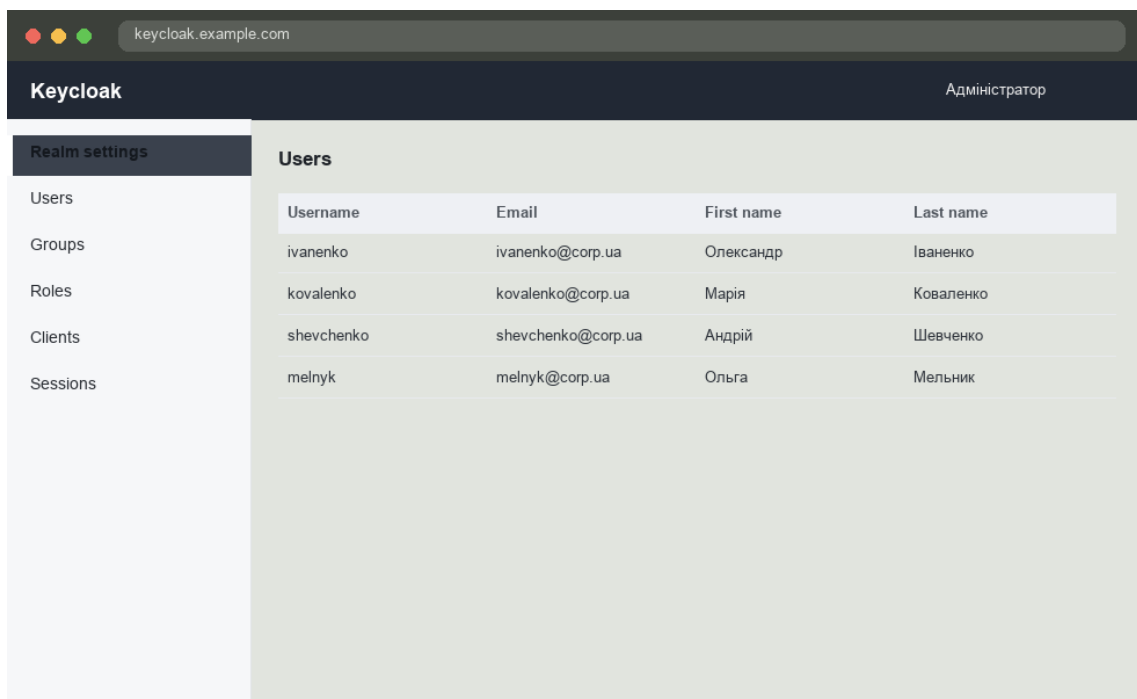


Рисунок 1.2 – Адміністративна консоль Keycloak

Ключовими особливостями Keycloak є:

- керування користувачами, ролями та групами в межах ізольованих областей (realms);
- підтримка протоколів OAuth 2.0, OpenID Connect та SAML;

- єдиний вхід і федерація ідентичностей із зовнішніми каталогами та соціальними мережами;
- відкритий код і можливість самостійного розгортання на власній інфраструктурі.

Перевагами Keycloak є відкритість, безкоштовність базового використання, гнучкість налаштування та незалежність від конкретного хмарного постачальника. До недоліків належать складність розгортання та супроводу, потреба у виділеному сервері та кваліфікованому адмініструванні, а також спрямованість на завдання автентифікації й авторизації, а не на ведення предметних записів користувачів із довільним набором атрибутів.

Directus [14] — це відкрита платформа керування даними, що надбудовується над наявною реляційною базою даних SQL і автоматично формує адміністративний інтерфейс та програмні інтерфейси доступу до даних. Платформа дає змогу керувати довільними таблицями, користувачами й ролями та надає REST і GraphQL API для інтеграції з іншими застосунками. Платформу керування даними Directus наведено на рисунку 1.3.

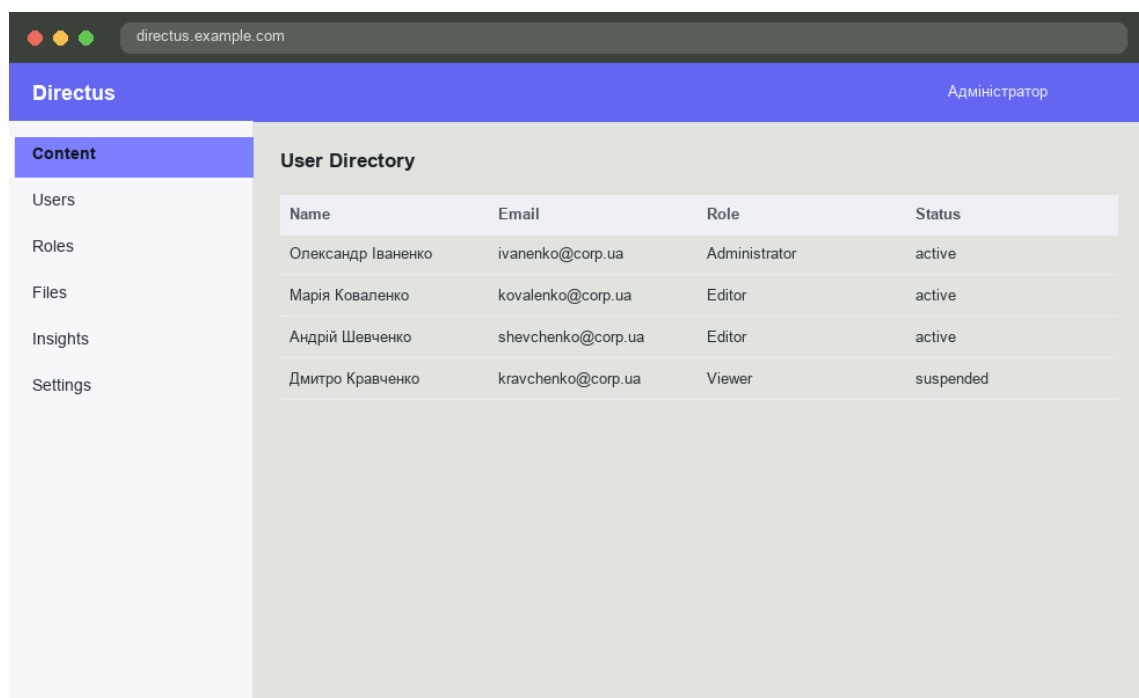


Рисунок 1.3 – Платформа керування даними Directus

Ключовими особливостями Directus є:

- робота поверх наявної реляційної бази даних SQL без зміни її структури;
- автоматичне формування адміністративного інтерфейсу для перегляду та редагування записів;
- керування користувачами, ролями та правами доступу до окремих колекцій даних;
- надання REST та GraphQL API для програмної взаємодії із застосунками.

Перевагами Directus є відкритий код, гнучкість роботи з довільними структурами даних та наявність готових інтерфейсів і програмних інтерфейсів доступу. Недоліками є потреба в розгортанні та супроводі платформи й бази даних, надлишкова для вузьких завдань функціональність, а також узагальнений характер інтерфейсу, що не враховує специфіки конкретної предметної області обліку записів користувачів.

phpMyAdmin [15] — це безкоштовний веб-інструмент адміністрування баз даних MySQL і MariaDB, написаний мовою PHP. Він призначений для виконання адміністративних завдань через браузер, зокрема перегляду структури баз даних, виконання SQL-запитів, а також безпосереднього перегляду та редагування записів у таблицях. Веб-інтерфейс phpMyAdmin наведено на рисунку 1.4.

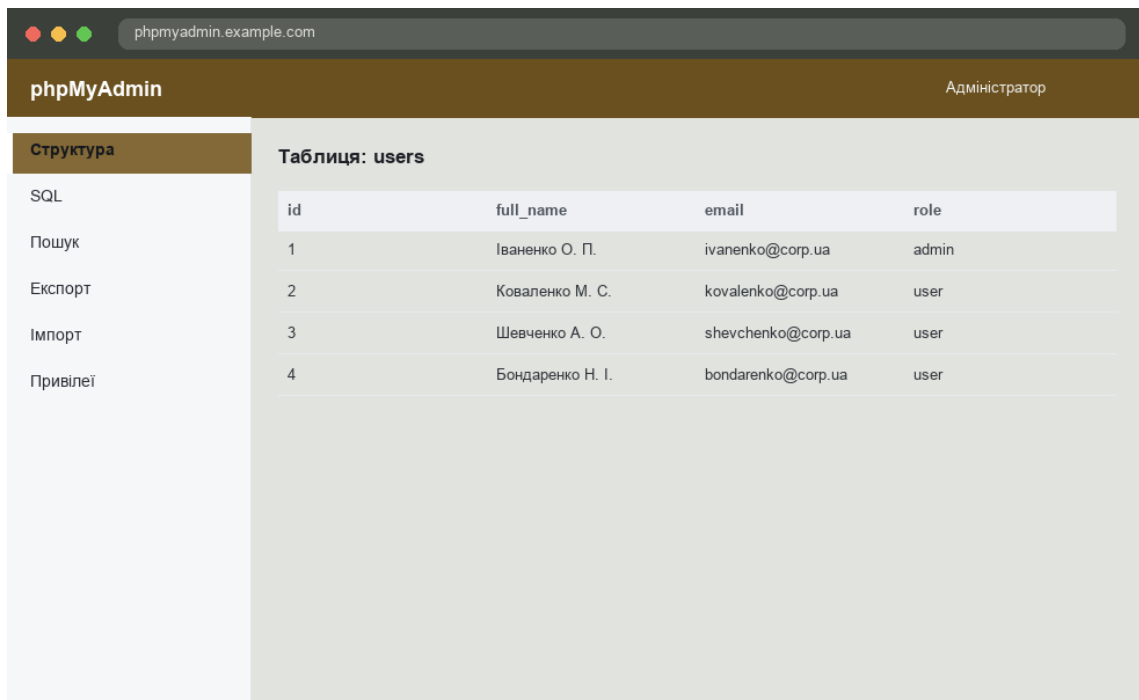


Рисунок 1.4 – Веб-інтерфейс phpMyAdmin

Ключовими особливостями phpMyAdmin є:

- перегляд, створення, редагування та видалення записів у таблицях баз даних через браузер;
- виконання довільних SQL-запитів та керування структурою баз даних;
- імпорт та експорт даних у різних форматах, зокрема SQL і CSV;
- адміністрування облікових записів користувачів і прав доступу на рівні сервера баз даних.

Перевагами phpMyAdmin є простота, безкоштовність, поширеність і пряма робота з базою даних. Водночас інструмент є засобом загального адміністрування баз даних, а не спеціалізованою системою обліку: він не передбачає прикладної бізнес-логіки, валідації предметних даних, зручного для кінцевого користувача інтерфейсу та рольового розмежування доступу на рівні застосунку, що робить його малопридатним для безпосереднього використання непередбаченими операторами.

Зведений порівняльний аналіз розглянутих рішень за такими критеріями, як спеціалізація, складність розгортання, наявність розмежування доступу, зручність інтерфейсу для кінцевого користувача та вартість, наведено

в таблиці 1.2. Проведений огляд свідчить, що кожне з наявних рішень має суттєві недоліки в контексті поставленого завдання: одні системи орієнтовані переважно на керування ідентичностями та доступом, інші є інструментами загального адміністрування баз даних, а ще інші вирізняються складністю розгортання та супроводу або надлишковою для невеликої організації функціональністю. Жодне з них не є простою спеціалізованою системою опрацювання записів користувачів із готовим прикладним інтерфейсом.

Таблиця 1.2 – Порівняльний аналіз програмного забезпечення для управління записами користувачів

Характеристика	Microsoft Entra ID	Keycloak	Directus	phpMyAdmin
Тип	Хмарна IAM	Відкрита IAM	Платформа даних	Адмін. БД
Спеціалізація на обліку записів	Низька	Низька	Середня	Низька
Складність розгортання	Низька	Висока	Середня	Низька
Розмежування доступу	Так	Так	Так	На рівні БД
Зручність для кінцевого користувача	Середня	Низька	Середня	Низька
Вартість	Платна	Безкоштовна	Безкоштовна/платна	Безкоштовна

З огляду на зазначене обґрунтованим є рішення розробити власне просте спеціалізоване програмне забезпечення, яке поєднує необхідний набір функцій опрацювання записів користувачів, рольове розмежування доступу та зручний адаптивний інтерфейс, не обтяжуючи користувача надмірною складністю налаштування й супроводу та не вимагаючи придбання комерційних ліцензій. Таке рішення дасть змогу врахувати специфіку предметної області та забезпечити саме ті можливості, що відповідають реальним потребам організації.

1.5 Висновки за розділом 1

У першому розділі досліджено предметну область, пов'язану з опрацюванням записів користувачів. Визначено поняття запису користувача як структурованої сукупності даних про особу та поняття опрацювання записів як комплексу операцій створення, перегляду, редагування, видалення, пошуку, фільтрування та експорту даних із розмежуванням доступу. Окреслено сфери застосування таких систем, серед яких управління персоналом, управління взаємовідносинами з клієнтами, державні реєстри та навчальні заклади, що підтверджує актуальність і практичну значущість розглядуваного завдання.

Проаналізовано еволюцію способів зберігання та опрацювання записів — від паперових журналів і картотек через електронні таблиці та файлові бази даних до клієнт-серверних систем керування базами даних і спеціалізованого веб-програмного забезпечення. Установлено, що для багатокористувацького, безпечного та зручного опрацювання записів найдоцільнішим є застосування веб-додатка на основі клієнт-серверної архітектури. Розглянуто складові, ключові функції та вимоги до систем управління записами користувачів, зокрема щодо захисту персональних даних відповідно до загальних принципів та вимог GDPR і Закону України «Про захист персональних даних».

Виконано порівняльний аналіз чотирьох наявних програмних рішень — Microsoft Entra ID, Keycloak, Directus та phpMyAdmin. З'ясовано, що кожне з них має суттєві недоліки в контексті поставленого завдання: складність розгортання та супроводу, надлишкову функціональність, вартість комерційних ліцензій або відсутність спеціалізації на обліку предметних записів користувачів. На підставі проведеного аналізу обґрунтовано доцільність розроблення власної простої спеціалізованої програмної системи опрацювання записів користувачів, вимоги до якої та проєктні рішення розглядаються в наступних розділах пояснювальної записки.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

Вибір мови програмування є одним із визначальних рішень на етапі проєктування програмної системи, оскільки він безпосередньо впливає на швидкість розробки, надійність коду, можливість його супроводу та масштабування. Для реалізації системи опрацювання записів користувачів, яка складається з клієнтської частини (веб-інтерфейсу) та серверної частини (REST API), як основну мову програмування обрано TypeScript . Така мова застосовується одночасно і для фронтенду, і для бекенду, що забезпечує єдиний технологічний простір у межах усього проєкту.

TypeScript є надмножиною мови JavaScript [16], що означає повну сумісність із наявним кодом JavaScript та можливість поступового впровадження статичної типізації. Будь-яка коректна програма мовою JavaScript є водночас коректною програмою мовою TypeScript, проте TypeScript розширює можливості базової мови, додаючи систему типів, інтерфейси, узагальнення (generics), перелічувані типи та інші засоби, властиві мовам зі статичною типізацією. Завдяки цьому розробник дістає переваги строгого контролю типів, не втрачаючи доступу до великої екосистеми бібліотек JavaScript.

Головною перевагою TypeScript над звичайним JavaScript є виявлення значної частини помилок ще на етапі компіляції, а не під час виконання програми. Помилки, пов'язані з невідповідністю типів, звертанням до неіснуючих властивостей об'єктів, передаванням аргументів неправильного типу або хибним використанням значень null та undefined, фіксуються компілятором до запуску застосунку. Це особливо важливо для систем опрацювання даних, де некоректна обробка полів запису користувача (наприклад, дати народження, статусу або ідентифікатора власника) може призвести до порушення цілісності інформації.

Статична типізація суттєво полегшує рефакторинг та супровід коду. У великому проєкті, що складається з багатьох модулів, перейменування поля сутності, зміна сигнатури функції або структури об'єкта автоматично виявляються в усіх місцях використання, де порушено типову узгодженість. Завдяки цьому зменшується ризик внесення прихованих дефектів під час модифікації наявного коду, а самі зміни стають передбачуванішими та безпечнішими. Це є вагомим аргументом для навчального проєкту, який передбачає поетапне нарощування функціональності.

Окремою перевагою є якісна підтримка з боку інтегрованих середовищ розробки. Інформація про типи дає змогу засобам автодоповнення коду, навігації за визначеннями та контекстних підказок працювати точніше й повніше. Середовище розробки здатне підказувати доступні властивості об'єкта, очікувані параметри функцій та можливі значення, що пришвидшує написання коду і знижує когнітивне навантаження на розробника. Порівняння можливостей TypeScript та JavaScript за основними характеристиками наведено в таблиці 2.1, а узагальнені переваги й недоліки застосування TypeScript подано в таблиці 2.2.

Таблиця 2.1 – Порівняння мов TypeScript та JavaScript

Критерій	TypeScript	JavaScript
Типізація	Статична	Динамічна
Виявлення помилок	На етапі компіляції	Під час виконання
Підтримка IDE	Розширена (автодоповнення, підказки типів)	Базова
Масштабованість коду	Висока	Обмежена
Сумісність	Надмножина JS, компілюється у JS	Базова мова
Крива навчання	Потребує вивчення системи типів	Нижчий поріг входження

Таблиця 2.2 – Переваги та недоліки TypeScript

Переваги	Недоліки
Статична типізація та раннє виявлення помилок	Потрібен етап компіляції в JavaScript

Переваги	Недоліки
Кращий рефакторинг та автодоповнення	Додаткові витрати часу на оголошення типів

Кінець таблиці 2.2

Переваги	Недоліки
Підвищена надійність і читабельність коду	Потреба у вивченні синтаксису типів
Повна сумісність із JavaScript	Деякі бібліотеки потребують окремого опису типів
Єдина мова для клієнта і сервера	Незначне збільшення обсягу коду

Важливим чинником вибору стала можливість використання єдиної мови програмування як на клієнті, так і на сервері. Оскільки серверна частина побудована на платформі Node.js, що від початку виконує код мовою JavaScript, застосування TypeScript на бекенді є природним рішенням. Єдина мова дає змогу спільно використовувати типи й описи структур даних між клієнтом і сервером, уніфікувати інструменти збірки та перевірки коду, а також спрощує перехід розробника між частинами системи без потреби змінювати парадигму мислення та синтаксис.

Ключову роль у роботі з TypeScript відіграє компілятор tsc, який виконує перевірку типів та транспіляцію коду TypeScript у код JavaScript, придатний для виконання у браузері або середовищі Node.js. Компілятор аналізує вихідний код, звіряє його з оголошеними типами, повідомляє про виявлені невідповідності та формує підсумкові файли JavaScript відповідно до налаштувань цільової версії стандарту мови. Поведінка компілятора гнучко конфігурується через файл tsconfig.json, де задаються рівень строгості перевірок, цільова версія ECMAScript, модульна система та інші параметри.

Система типів TypeScript охоплює як примітивні типи (number, string, boolean), так і складені й допоміжні конструкції. Інтерфейси (interface) застосовуються для опису форми об'єктів, зокрема для визначення структур сутностей системи, таких як обліковий запис та запис користувача, що робить контракти даних явними й перевірюваними. Класи (class) забезпечують об'єктно-орієнтований підхід із підтримкою інкапсуляції, успадкування та модифікаторів доступу, а узагальнені типи дають змогу створювати повторно використовувані компоненти й функції, що працюють із різними типами даних

без втрати типобезпеки. Сукупність цих засобів робить TypeScript обґрунтованим вибором для розроблюваної системи.

2.2 Вибір фреймворку та технологій для розробки

Для побудови клієнтської частини системи обрано бібліотеку React, яка є однією з найпоширеніших технологій розробки інтерфейсів користувача для вебзастосунків. React реалізує компонентний підхід, за якого інтерфейс розкладається на незалежні, повторно використовувані компоненти, кожен з яких відповідає за певну частину відображення та власну логіку. Такий підхід добре узгоджується зі структурою розроблюваної системи, де чітко виокремлюються екрани входу, реєстрації, інформаційного дашборда, форми редагування запису та панелі деталей.

В основі React лежить механізм віртуального DOM (Virtual DOM), який є полегшеним представленням реального дерева елементів сторінки в пам'яті. Під час зміни стану застосунку React будує нову версію віртуального дерева, порівнює її з попередньою та обчислює мінімальний набір змін, які потрібно внести у справжній DOM. Завдяки цьому оновлення інтерфейсу відбувається ефективно, без надлишкового перемальовування сторінки, що особливо важливо для таблиці записів із пошуком, фільтрацією, сортуванням та пагінацією, яка часто оновлюється у відповідь на дії користувача.

Вагомими перевагами React є велика й активна екосистема, наявність значної кількості готових бібліотек та компонентів, а також широка спільнота розробників, що спрощує пошук рішень типових задач. React добре пристосований для створення односторінкових застосунків (SPA) [17], у яких перехід між екранами відбувається без повного перезавантаження сторінки. Для організації клієнтської маршрутизації застосовано бібліотеку React Router [18], яка зіставляє адреси у браузері з відповідними компонентами та забезпечує навігацію між розділами системи у межах єдиного завантаженого документа.

Керування станом та логікою компонентів реалізовано за допомогою хуків (hooks) — функцій, що дають змогу використовувати локальний стан, побічні ефекти та інші можливості React у функційних компонентах. Зокрема, хук `useState` застосовується для зберігання локального стану компонентів, `useEffect` — для виконання побічних ефектів, як-от завантаження даних із сервера, а механізм `Context` разом із відповідними хуками використано для зберігання інформації про автентифікованого користувача та поширення її між компонентами без потреби передавати дані через увесь ланцюг властивостей.

Під час вибору технології для клієнтської частини розглядався також фреймворк `Angular` як одне з поширених комплексних рішень для побудови вебзастосунків. На відміну від `React`, який є бібліотекою з гнучкою архітектурою та можливістю вибору додаткових інструментів, `Angular` [19] є повноцінним фреймворком із суворішою структурою, власною системою впровадження залежностей та вищим порогом входження. Для навчального проєкту відповідного масштабу обрано `React` як простіше у засвоєнні, гнучкіше та достатнє за можливостями рішення. Детальне порівняння `React` та `Angular` за основними критеріями наведено в таблиці 2.3. Для наочного відображення структури клієнтської частини розроблюваної системи на рисунку 2.1 наведено компонентну архітектуру `React`-застосунку.

Таблиця 2.3 – Порівняння фреймворків `React` та `Angular`

Критерій	React	Angular
Тип	Бібліотека	Повноцінний фреймворк
Мова	JavaScript / TypeScript	TypeScript
Крива навчання	Помірна	Висока
Гнучкість	Висока (вибір бібліотек)	Нижча (рішення вбудовані)
Продуктивність	Легка, віртуальний DOM	Важчий каркас
Спільнота та екосистема	Дуже велика	Велика

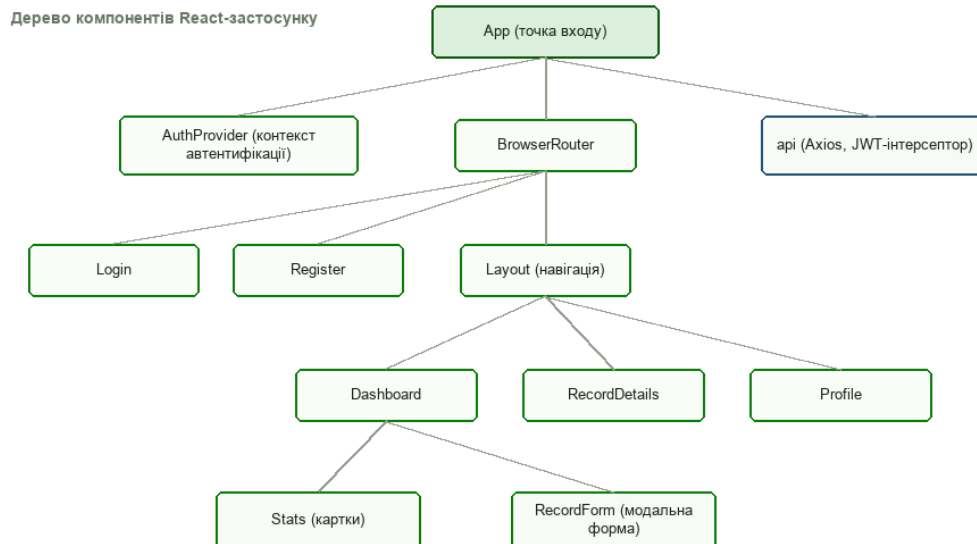


Рисунок 2.1 – Компонентна архітектура React-застосунку

Серверну частину системи побудовано на платформі Node.js із застосуванням фреймворку Express. Node.js є середовищем виконання JavaScript поза браузером, що дає змогу реалізувати серверну логіку тією самою мовою, що й клієнтську частину, та використовувати TypeScript на обох рівнях. Express — це мінімалістичний та гнучкий вебфреймворк, який надає зручні засоби для опрацювання HTTP-запитів, визначення маршрутів та формування відповідей, завдяки чому добре підходить для реалізації архітектури REST API, передбаченої в системі.

Express побудовано навколо концепції проміжного програмного забезпечення (middleware) — послідовності функцій, кожна з яких обробляє запит на певному етапі його проходження. Такий підхід дає змогу зручно реалізувати наскрізну функціональність: розбір тіла запиту у форматі JSON, перевірку JWT-токена автентифікації, контроль прав доступу за ролями користувача, а також централізоване оброблення помилок. Асинхронна неблокувальна модель Node.js забезпечує достатню продуктивність під час одночасного опрацювання запитів, що є важливим для серверної частини, яка обслуговує операції створення, перегляду, редагування та видалення записів.

Для зберігання даних обрано систему керування базами даних SQLite — вбудовану реляційну базу даних, яка зберігає всю інформацію в одному файлі та не потребує розгортання й адміністрування окремого серверного процесу. Завдяки цьому суттєво спрощуються встановлення та розгортання системи, що особливо доречно для застосунків малого та середнього масштабу, до яких належить розроблювана система. SQLite підтримує мову структурованих запитів SQL, забезпечує транзакційність операцій та дотримання вимог цілісності даних, що дає змогу надійно зберігати облікові записи й записи користувачів та оперувати ними засобами реляційної моделі.

Взаємодію серверної частини із базою даних SQLite реалізовано через бібліотеку `better-sqlite3` [20], яка надає простий та продуктивний синхронний інтерфейс для виконання SQL-запитів безпосередньо з коду на Node.js. Синхронна модель роботи спрощує написання серверної логіки, усуваючи потребу в додаткових механізмах опрацювання асинхронних операцій із базою даних, а підтримка підготовлених запитів сприяє безпеці та швидкодії. Реляційна структура бази даних відображає модель предметної області, у якій один обліковий запис пов'язаний із багатьма записами користувачів.

Окрім основних технологій, у проєкті використано низку допоміжних бібліотек, кожна з яких відповідає за окремий аспект функціональності. Для збірки та розробки клієнтської частини застосовано інструмент Vite [21], що забезпечує швидкий запуск середовища розробки, миттєве оновлення модулів під час редагування коду та ефективну підсумкову збірку застосунку. Для взаємодії клієнта із сервером використано HTTP-клієнт Axios [22], який спрощує надсилання запитів до REST API, автоматично серіалізує та десеріалізує дані у форматі JSON і дає змогу зручно додавати до запитів заголовки авторизації з токеном.

Підсистему безпеки реалізовано за допомогою спеціалізованих бібліотек. Бібліотека `jsonwebtoken` застосовується для формування та перевірки токенів JWT, які підтверджують факт успішної автентифікації користувача й передаються у заголовок кожного захищеного запиту, забезпечуючи перевірку

прав доступу без зберігання сесій на сервері. Бібліотека `bcryptjs` використовується для хешування паролів за алгоритмом `bcrypt` [23], завдяки чому в базі даних зберігаються не самі паролі, а їхні криптографічні хеші, що забезпечує облікові дані користувачів навіть у разі компрометації сховища.

Для контролю коректності вхідних даних застосовано бібліотеку `zod`, яка дає змогу декларативно описувати схеми валідації та перевіряти отримані від клієнта дані на відповідність очікуваному формату. За допомогою `Zod` [24] реалізовано перевірку обов'язкових полів, формату адреси електронної пошти, допустимих значень статусу й ролі та обмежень на довжину текстових полів. Важливою перевагою цієї бібліотеки є тісна інтеграція з `TypeScript`, що дає змогу автоматично виводити типи даних зі схем валідації й уникати дублювання описів структур, забезпечуючи узгодженість між перевіркою під час виконання та статичною типізацією.

2.3 Висновки за розділом 2

У цьому розділі обґрунтовано вибір мови програмування, фреймворків та допоміжних технологій для реалізації системи опрацювання записів користувачів. Як основну мову обрано `TypeScript`, що застосовується одночасно на клієнті та сервері й завдяки статичній типізації забезпечує виявлення помилок на етапі компіляції, спрощення рефакторингу, якіснішу підтримку з боку середовищ розробки та узгодженість контрактів даних між частинами системи. Розглянуто роль компілятора `TypeScript`, а також основні засоби системи типів — інтерфейси, класи та узагальнення.

Для клієнтської частини обрано бібліотеку `React` із компонентним підходом, віртуальним `DOM`, маршрутизацією за допомогою `React Router` та керуванням станом через хуки, що відповідає вимогам до односторінкового застосунку. Серверну частину побудовано на платформі `Node.js` із фреймворком `Express`, який забезпечує реалізацію `REST API` та зручну організацію проміжного програмного забезпечення. Для зберігання даних

застосовано вбудовану реляційну базу даних SQLite через бібліотеку better-sqlite3, що спрощує розгортання та відповідає масштабу системи.

Додатково обґрунтовано використання допоміжних бібліотек: Vite для швидкої збірки, Axios як HTTP-клієнта, jsonwebtoken для автентифікації на основі токенів JWT, bcryptjs для хешування паролів та zod для валідації вхідних даних. Сукупність обраних технологій утворює узгоджений та сучасний технологічний стек, який забезпечує надійність, безпеку й супроводжуваність системи та створює основу для подальшого проєктування й програмної реалізації, розглянутих у наступних розділах.

3 ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення

Розроблена програмна система опрацювання записів користувачів побудована за клієнт-серверною архітектурою і реалізована у вигляді односторінкового веб-додатку (SPA, Single Page Application). Під односторінковим додатком розуміють тип веб-застосунку, у якому браузер одноразово завантажує єдину HTML-сторінку та супровідний пакет програмного коду, після чого вся подальша взаємодія з користувачем відбувається без повного перезавантаження сторінки. Зміна вмісту екранів, перехід між розділами та оновлення даних здійснюються динамічно засобами клієнтського коду, який звертається до сервера лише за необхідними порціями інформації. Завдяки такому підходу досягається висока швидкість реакції інтерфейсу, зменшується обсяг переданих через мережу даних і забезпечується відчуття роботи з повноцінним настільним застосунком.

Система складається з трьох логічно відокремлених рівнів. Перший рівень — клієнтська частина (фронтенд), реалізована мовою TypeScript на основі бібліотеки React із застосуванням збирача Vite ; вона відповідає за відображення інтерфейсу та обробку дій користувача. Другий рівень — серверна частина (бекенд), побудована на платформі Node.js із використанням каркаса Express; вона реалізує програмний інтерфейс REST API , бізнес-логіку, автентифікацію та авторизацію. Третій рівень — реляційна база даних SQLite, що забезпечує постійне зберігання облікових записів і записів користувачів. Така трирівнева організація дозволяє чітко розмежувати відповідальність між компонентами, спрощує супровід і надає можливість незалежного розвитку кожного рівня.

Обмін даними між клієнтом і сервером здійснюється за принципами архітектурного стилю REST: клієнт надсилає HTTP-запити до визначених адрес ресурсів, а сервер повертає відповіді у форматі JSON. Для розмежування доступу застосовано механізм автентифікації на основі токенів JSON Web

Token (JWT). Після успішного входу сервер видає клієнту підписаний токен, який зберігається на боці браузера і додається до кожного наступного захищеного запиту в заголовок Authorization за схемою Bearer. Сервер перевіряє підпис та строк дії токена, після чого встановлює особу користувача та його роль, що дає змогу реалізувати рольову модель доступу (адміністратор і оператор/користувач).

Клієнтську частину поділено на сукупність взаємопов'язаних програмних модулів. Кореневий модуль App виконує початкову ініціалізацію застосунку та налаштування маршрутизації між екранами засобами бібліотеки React Router . Окремий модуль контексту автентифікації зберігає поточний стан сеансу (наявність токена, дані та роль користувача) і надає його всім компонентам програми. Логічно завершені екрани винесено в окремі модулі-сторінки: Login (вхід), Register (реєстрація), Dashboard (головний екран зі статистикою і таблицею записів), RecordDetails (деталі окремого запису) та Profile (профіль облікового запису). Багаторазово використовувані елементи інтерфейсу оформлено як компоненти: Layout задає загальний каркас сторінки з навігацією, а RecordForm реалізує модальну форму створення та редагування записів. Доступ до сервера інкапсульовано в окремому модулі арі-клієнта, який централізовано формує HTTP-запити, додає токен авторизації та опрацьовує відповіді.

Серверна частина так само структурована на функціональні модулі. Головний модуль index відповідає за запуск HTTP-сервера, підключення проміжного програмного забезпечення та реєстрацію маршрутів API. Модуль db реалізує підключення до бази даних SQLite , створення таблиць і початкове наповнення довідковими даними. Модуль auth містить логіку реєстрації, входу, формування та перевірки JWT-токенів, а також хешування паролів. Модуль records реалізує операції над записами користувачів — створення, читання, оновлення, видалення, пошук, фільтрацію, сортування, пагінацію, обчислення статистики та експорт у формат CSV. Модуль middleware містить проміжні

обробники, що перевіряють наявність і коректність токена та відповідність ролі користувача необхідним правам доступу.

Взаємодія клієнта із сервером реалізується через визначений набір ендпоінтів програмного інтерфейсу REST API . Кожен ендпоінт відповідає за окрему операцію над ресурсами системи й описується HTTP-методом та адресою. Перелік ендпоінтів наведено нижче:

- POST /api/auth/register — реєстрація нового облікового запису;
- POST /api/auth/login — вхід у систему з поверненням JWT-токена;
- GET /api/auth/me — отримання даних поточного автентифікованого користувача;
- GET /api/records — отримання списку записів із параметрами пошуку, фільтрації, сортування та пагінації (q, status, department, role, sort, page, pageSize);
- GET /api/records/:id — отримання деталей окремого запису за ідентифікатором;
- POST /api/records — створення нового запису;
- PUT /api/records/:id — оновлення наявного запису;
- DELETE /api/records/:id — видалення запису (доступно лише адміністратору);
- GET /api/records/export — експорт вибірки записів у файл формату CSV;
- GET /api/stats — отримання агрегованої статистики для інформаційного дашборда.

Розмежування відповідальності між модулями реалізує загальновідомий принцип розділення зони перегляду й зони бізнес-логіки. Клієнтська частина не містить безпосередніх звернень до бази даних і не зберігає конфіденційних правил доступу — вона лише відображає отримані з сервера дані та формує запити на основі дій користувача. Уся перевірка прав, валідація на боці сервера та операції зі сховищем зосереджені в серверній частині, що унеможливорює обхід обмежень доступу шляхом маніпуляцій із клієнтським кодом. Такий

розподіл підвищує захищеність системи й полегшує її подальше масштабування, оскільки серверний інтерфейс REST API може використовуватися не лише веб-клієнтом, а й іншими можливими клієнтськими застосунками без змін у логіці сервера.

Узагальнену взаємодію між клієнтом, сервером REST API та базою даних, а також рух даних у форматі JSON через протокол HTTP подано на структурній схемі роботи додатку. На схемі відображено три рівні системи й напрямки передавання запитів та відповідей між ними. На рисунку 3.1 наведено структурну схему роботи програмної системи.



Рисунок 3.1 – Загальна структурна схема роботи додатку

3.2 Основний алгоритм роботи програмного забезпечення

Робота програмної системи розпочинається із запуску клієнтського застосунку в браузері користувача. На етапі ініціалізації виконується завантаження програмного коду, налаштування світлої теми оформлення та маршрутизатора, а також створення контексту автентифікації. Одразу після цього застосунок звертається до локального сховища браузера з метою перевірки наявності збереженого JWT-токена попереднього сеансу. Якщо токен

присутній, виконується запит до ендпоінта GET /api/auth/me для підтвердження його чинності та отримання актуальних даних користувача.

За відсутності токена або у випадку його недійсності користувача спрямовують на екран входу. Тут пропонується ввести електронну пошту й пароль або перейти до екрана реєстрації нового облікового запису. Під час входу клієнт надсилає запит POST /api/auth/login, а сервер перевіряє відповідність пароля збереженому хешу та у разі успіху повертає підписаний JWT-токен. Отриманий токен зберігається на боці клієнта й надалі додається до кожного захищеного запиту, після чого користувача автоматично перенаправляють на головний екран — дашборд.

Після успішної автентифікації відбувається завантаження інформаційного дашборда. Спершу клієнт звертається до ендпоінта GET /api/stats для отримання агрегованих показників (загальна кількість записів, кількість активних, неактивних і заблокованих, розподіл за відділами), які відображаються у вигляді карток статистики. Паралельно виконується запит GET /api/records із параметрами за замовчуванням для заповнення таблиці записів. Таким чином користувач одразу отримує цілісну картину стану даних у системі.

Подальша робота визначається діями користувача над таблицею записів. Введення тексту в поле пошуку або зміна параметрів фільтрації за статусом, відділом чи роллю спричиняє повторний запит до сервера з відповідними query-параметрами та оновлення відображеної вибірки; аналогічно опрацьовуються сортування за стовпцями та перехід між сторінками пагінації. Натискання кнопки «Додати» відкриває модальну форму, дані якої після клієнтської валідації надсилаються методом POST для створення нового запису. Редагування наявного запису виконується через ту саму форму, попередньо заповнену поточними значеннями, з подальшим збереженням методом PUT. Видалення запису доступне лише адміністраторові: після підтвердження дії клієнт надсилає запит DELETE, після чого таблиця оновлюється. Окремою дією є експорт вибірки до файлу CSV через ендпоінт GET /api/records/export.

Важливою особливістю алгоритму є те, що після кожної модифікуючої операції (створення, редагування чи видалення запису) клієнт повторно запитує актуальний стан вибірки та оновлені статистичні показники. Завдяки цьому таблиця та картки дашборда завжди відображають узгоджені з базою даних значення, а користувач бачить результат своєї дії без потреби вручну оновлювати сторінку. У разі виникнення помилки на боці сервера — наприклад, через невалідні дані або відсутність прав доступу — клієнт перехоплює відповідну відповідь і відображає інформативне повідомлення, не порушуючи поточного стану інтерфейсу. Це забезпечує стійкість застосунку до помилкових дій користувача та збоїв мережевої взаємодії.

Завершальним кроком алгоритму є вихід із системи (logout): під час нього з боку клієнта видаляється збережений токен, скидається стан контексту автентифікації і користувача повертають на екран входу. Послідовність основних кроків роботи програмного забезпечення з гілками умовних переходів подано на блок-схемі. На ній відображено етапи ініціалізації, перевірки автентифікації, основний цикл опрацювання дій користувача над записами та завершення сеансу. На рисунку 3.2 наведено блок-схему основного алгоритму роботи програмного забезпечення.

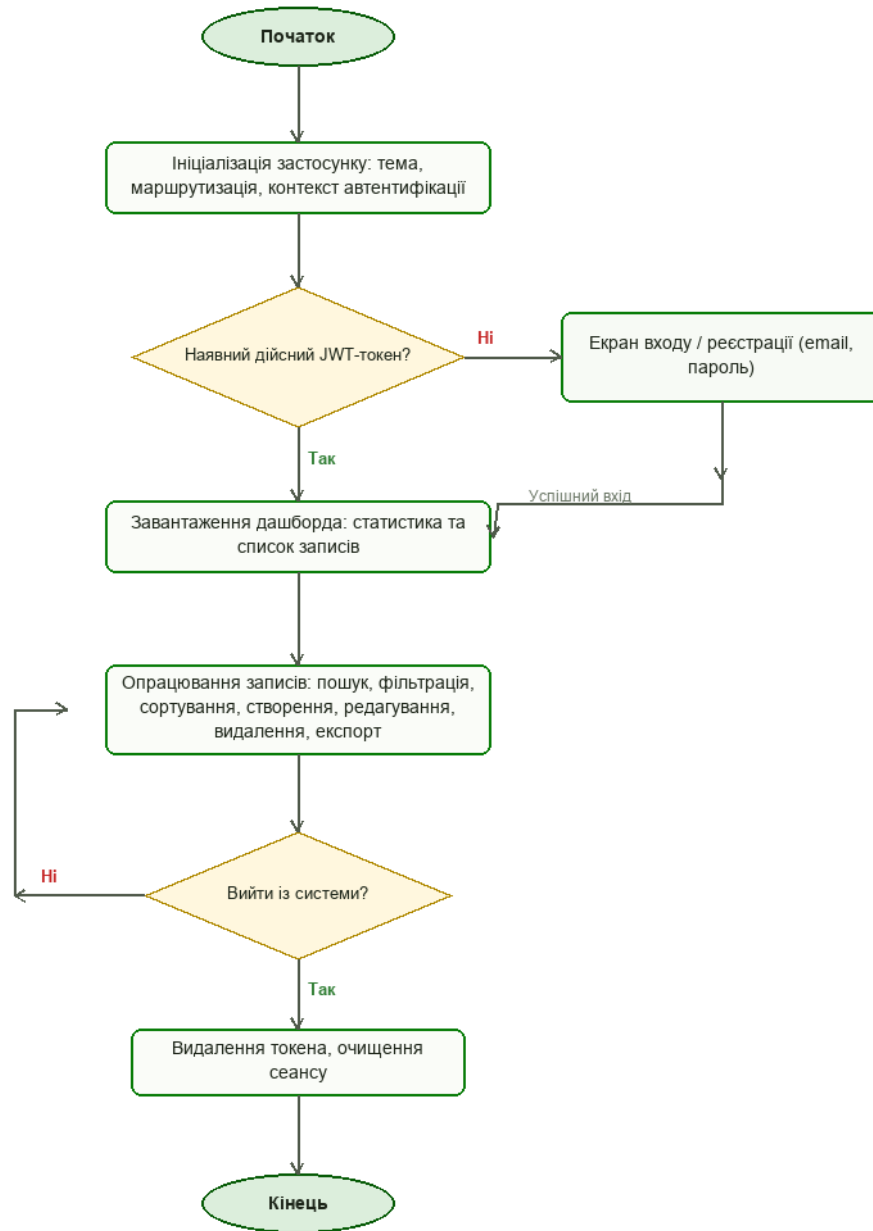


Рисунок 3.2 – Блок-схема основного алгоритму роботи програмного забезпечення

3.3 Прокєтування бази даних

Для постійного зберігання інформації обрано реляційну систему керування базами даних SQLite, що є файловою та не потребує окремого серверного процесу, завдяки чому спрощується розгортання й супровід застосунку. Логічну модель даних побудовано на двох взаємопов'язаних таблицях — accounts (облікові записи для входу) та records (записи користувачів як основна оброблювана сутність). Такий поділ дозволяє відокремити дані автентифікації від змістовних даних про користувачів і забезпечити цілісність та узгодженість інформації.

Таблиця accounts містить поля id (цілочисловий первинний ключ із автоінкрементом), email (текстове поле з обмеженням унікальності, що слугує логіном), passwordHash (текстове поле зі збереженим хешем пароля, обчисленим алгоритмом bcrypt), role (текстове поле ролі зі значеннями admin або user) та createdAt (текстове поле з датою й часом створення у форматі ISO). Зберігання саме хешу пароля, а не його відкритого значення, є обов'язковою вимогою безпеки й унеможливорює відновлення вихідного пароля у разі компрометації бази даних.

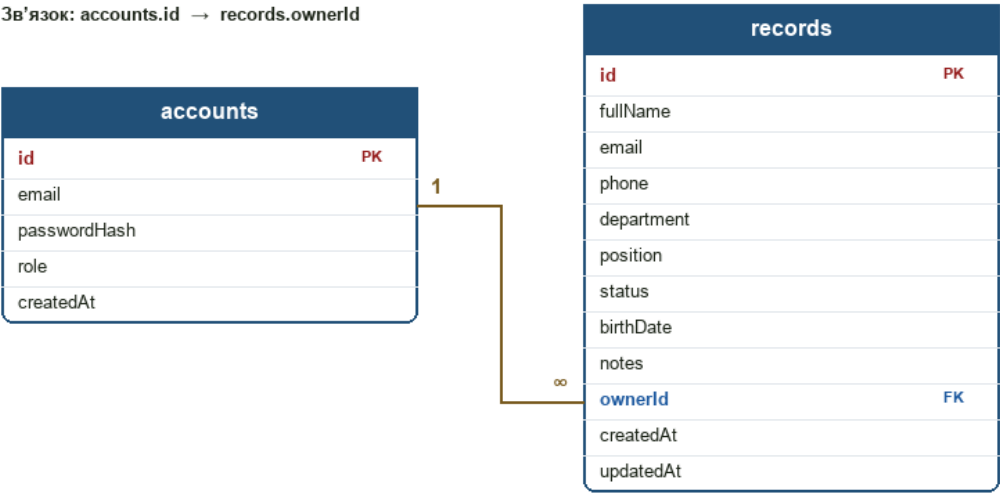
Таблиця records описує запис користувача й містить поля id (первинний ключ), fullName (повне ім'я), email, phone (телефон), department (відділ), position (посада), status (статус зі значеннями active, inactive або blocked), birthDate (дата народження), notes (нотатки), ownerId (зовнішній ключ, що посилається на ідентифікатор облікового запису-автора), а також createdAt і updatedAt (дати створення та останнього оновлення). Поле ownerId реалізує зв'язок між таблицями: один обліковий запис може створити багато записів користувачів, що відповідає типу зв'язку один-до-багатьох (accounts 1—N records).

Оскільки SQLite використовує систему динамічної типізації з обмеженою кількістю класів зберігання, усі текстові поля оголошено типом TEXT, а ідентифікатори та зовнішні ключі — типом INTEGER. Цілочислові первинні ключі оголошено з автоінкрементом, що гарантує унікальність

ідентифікаторів і звільняє застосунок від необхідності самостійно генерувати їх. Дати й позначки часу зберігаються як рядки у форматі ISO 8601, що забезпечує однозначне впорядкування та зручне перетворення на боці застосунку. Для стовпців із обмеженим набором допустимих значень (role у таблиці accounts та status у таблиці records) на рівні застосунку контролюється приналежність значення до переліку допустимих, що підтримує цілісність даних за межами можливостей типізації SQLite.

З метою підвищення продуктивності для найчастіших операцій пошуку та фільтрації створено індекси за стовпцями fullName і status таблиці records. Індекс за fullName прискорює пошук записів за повним іменем, який є основним сценарієм роботи з системою, а індекс за status оптимізує фільтрацію за станом запису й обчислення статистики для дашборда. Застосування індексів суттєво зменшує час виконання відповідних запитів на великих обсягах даних, оскільки уникає повного перегляду таблиці. Під час першого запуску база даних автоматично створюється разом із таблицями та індексами й наповнюється початковими довідковими даними, зокрема обліковим записом адміністратора, що спрощує введення системи в експлуатацію.

Структуру таблиць, склад їхніх полів та зв'язок один-до-багатьох між обліковими записами й записами користувачів наведено на схемі бази даних у вигляді ER-діаграми. На рисунку 3.3 наведено ER-діаграму бази даних програмної системи.



Один обліковий запис створює багато записів (1 : N)

Рисунок 3.3 – Схема бази даних (ER-діаграма)

3.4 Проєктування інтерфейсу

Під час проєктування інтерфейсу користувача дотримувалися низки вимог, спрямованих на зручність та ефективність роботи. Передусім інтерфейс має бути зрозумілим та інтуїтивно передбачуваним, аби користувач без спеціальної підготовки міг швидко опанувати основні операції. Другою вимогою є адаптивність — коректне відображення на екранах різної роздільної здатності, зокрема настільних комп'ютерів і планшетів. Третьою важливою вимогою є наявність зворотного зв'язку: кожна дія користувача супроводжується відповідною реакцією системи — індикаторами завантаження, повідомленнями про успіх та інформативними повідомленнями про помилки валідації.

Візуальне оформлення виконано у світлій темі з опорою на принципи Material-подібного стилю: використано чисті поверхні карток, помірні тіні, виразні акцентні кольори для основних дій та узгоджену систему відступів і

типографіки. Такий підхід забезпечує візуальну охайність, добру читабельність та сучасний вигляд застосунку, не перевантажуючи увагу користувача надмірними декоративними елементами.

Інтерфейс складається з кількох логічно завершених екранів. Екран входу містить поля для введення електронної пошти й пароля та посилання на реєстрацію. Екран реєстрації пропонує ввести електронну пошту, пароль і його повторення з перевіркою збігу. Центральним елементом є дашборд, який поєднує картки зі статистичними показниками та таблицю записів із вбудованими засобами пошуку, фільтрації, сортування й пагінації, а також кнопками додавання запису та експорту до CSV. Форма запису реалізована як модальне вікно з полями всіх атрибутів запису й валідацією введених даних. Екран деталей відображає повну інформацію про окремий запис, а екран профілю надає дані поточного облікового запису та можливість виходу із системи.

Окрему увагу приділено доступності керівних елементів залежно від ролі користувача. Дії, які потребують підвищених прав (зокрема видалення записів), приховуються для оператора й відображаються лише адміністратору, що узгоджує інтерфейс із рольовою моделлю доступу та запобігає спробам виконати недозволені операції. Поля форм супроводжуються підказками щодо очікуваного формату, а у разі введення некоректних значень поряд із відповідним полем з'являється повідомлення про помилку, що дозволяє користувачеві швидко виправити дані без втрати вже введеної інформації. Така узгодженість візуального оформлення, поведінки елементів і повідомлень про стан системи формує цілісний і передбачуваний користувацький досвід.

Загальну композицію головного екрана — розміщення панелі навігації, блоку статистичних карток і таблиці записів з елементами керування — подано на концепті головного вікна додатку. На рисунку 3.4 наведено головне вікно програмної системи опрацювання записів користувачів.

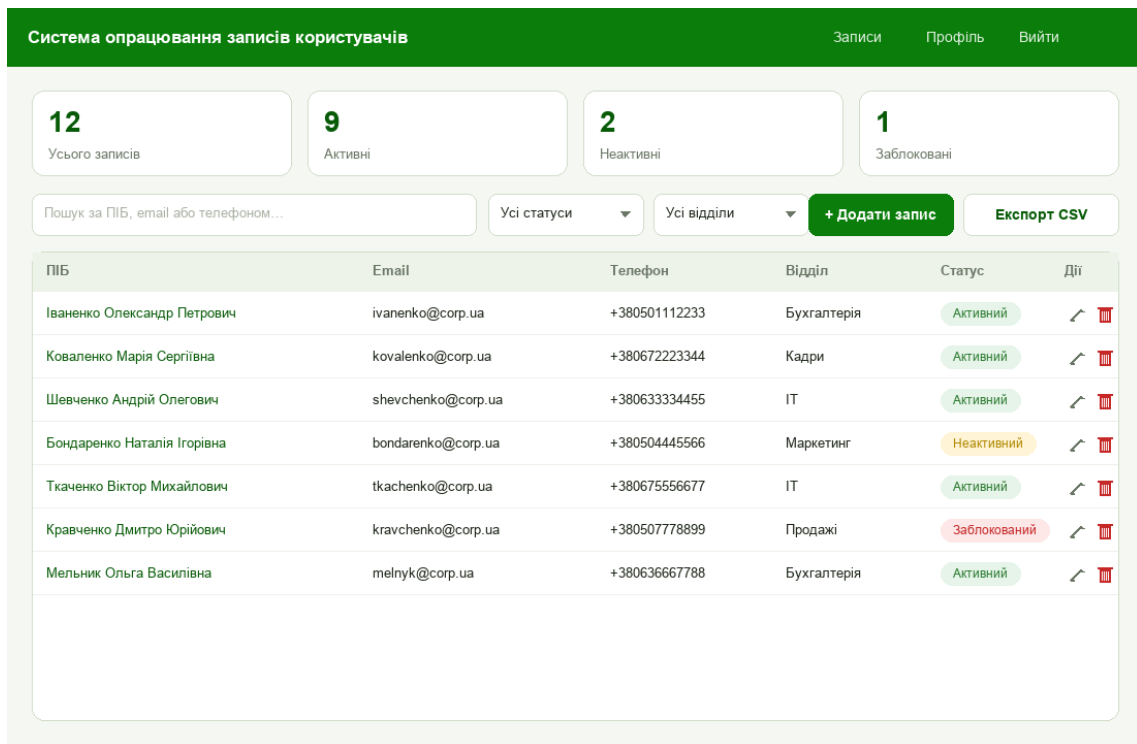


Рисунок 3.4 – Концепт головного вікна додатку

3.5 Висновки за розділом 3

У третьому розділі описано програмну реалізацію системи опрацювання записів користувачів. Обґрунтовано вибір клієнт-серверної архітектури у вигляді односторінкового веб-додатку з трьома рівнями: клієнтом на основі React і TypeScript, серверним програмним інтерфейсом REST API на платформі Node.js та Express, а також реляційною базою даних SQLite. Визначено склад програмних модулів клієнта й сервера, описано принципи обміну даними у форматі JSON через протокол HTTP та механізм автентифікації на основі JWT-токенів, а також наведено повний перелік ендпоінтів REST API.

Подано основний алгоритм роботи програмного забезпечення — від запуску й ініціалізації застосунку та перевірки токена до виконання операцій над записами (пошуку, фільтрації, сортування, пагінації, створення, редагування, видалення та експорту) і завершення сеансу. Спроектовано реляційну модель даних із таблицями accounts і records, зв'язком один-до-

багатьох та індексами для прискорення пошуку, а також сформульовано вимоги до інтерфейсу й описано основні екрани застосунку.

Отримані проєктні рішення повністю узгоджуються з функціональними та нефункціональними вимогами до системи й утворюють завершену основу для подальшої практичної реалізації та тестування, які розглянуто в наступному розділі пояснювальної записки.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Розроблена програмна система призначена для автоматизації опрацювання записів користувачів у межах організації. Її основним завданням є заміна ручного або напівавтоматичного обліку, який зазвичай ведеться у розрізнених електронних таблицях, єдиним централізованим веб-орієнтованим інструментом. Система забезпечує цілісність даних, розмежування доступу за ролями, швидкий пошук та актуальність відомостей, а також формування узагальненої статистики й експорт вибірок у зовнішній формат. Завдяки клієнт-серверній архітектурі доступ до інформації можливий одночасно для кількох співробітників із різних робочих місць.

Користувачами системи є дві основні категорії осіб. Адміністратори отримують повний доступ до функціональності: вони керують усіма записами, створюють, редагують та видаляють їх, змінюють ролі облікових записів, переглядають статистику й виконують експорт даних. Оператори (звичайні користувачі) мають обмежений набір повноважень: вони переглядають перелік записів та їхні деталі, користуються пошуком і фільтрами, проте не можуть видаляти записи та керувати ролями. Такий розподіл повноважень реалізує модель рольового розмежування доступу (RBAC) та запобігає ненавмисним або несанкціонованим змінам критичних даних.

Програмна система реалізує такий перелік функцій:

- реєстрація та авторизація користувачів за допомогою електронної пошти й пароля з видачею JWT-токена;
- розмежування доступу за ролями (адміністратор та оператор) на основі моделі RBAC;
- повний цикл опрацювання записів CRUD — створення, перегляд, редагування та видалення;

- пошук записів за прізвищем, ім'ям та по батькові, електронною поштою й телефоном;
- фільтрація записів за статусом, відділом та роллю;
- сортування переліку записів за обраними стовпцями;
- пагінація списку записів для зручної навігації великими обсягами даних;
- експорт вибірки записів у формат CSV;
- формування узагальненої статистики для інформаційного дашборда.

Вхідними даними системи є відомості про записи користувачів (прізвище, ім'я та по батькові, електронна пошта, телефон, відділ, посада, статус, дата народження, нотатки), а також облікові дані для входу — електронна пошта й пароль. Вихідними даними виступають відформатовані переліки записів із результатами пошуку та фільтрації, узагальнена статистика (загальна кількість записів, кількість активних, неактивних і заблокованих, розподіл за відділами), а також сформований файл CSV із вибраними записами. Обмін даними між клієнтом і сервером здійснюється у форматі JSON через протокол HTTP за принципами REST.

Систему розроблено у середовищі Visual Studio Code мовою програмування TypeScript із застосуванням бібліотеки React для побудови клієнтської частини та платформи Node.js разом із фреймворком Express для серверної частини. Для розгортання й запуску системи спершу завантажують її вихідний код із репозиторію командою `git clone`, після чого у кореневому каталозі проєкту встановлюють усі залежності клієнта й сервера командою `npm run install:all`. Далі в окремому терміналі запускають серверну частину командою `npm run server`, яка піднімає REST API за адресою `http://localhost:4000`, а у другому терміналі — клієнтську частину командою `npm run client`, що стартує веб-додаток за адресою `http://localhost:5173`. Після цього система готова до роботи, а доступ до неї здійснюється через веб-браузер.

4.2 Умови виконання програми

Для коректного функціонування серверної частини системи необхідне середовище виконання Node.js версії 18 або вище. Саме ця версія забезпечує підтримку сучасних мовних можливостей та сумісність із використаними бібліотеками, зокрема better-sqlite3, express, jsonwebtoken та bcryptjs. Серверна частина не потребує окремого сервера бази даних, оскільки використовує файлову реляційну базу даних SQLite, що спрощує розгортання та зменшує вимоги до апаратного забезпечення.

Клієнтська частина системи є односторінковим веб-додатком (SPA) і виконується в сучасному веб-браузері. Для повноцінної роботи рекомендовано використовувати браузери Google Chrome, Microsoft Edge або Mozilla Firefox актуальних версій, що підтримують стандарти HTML5, CSS3 та сучасні можливості JavaScript. Інтерфейс є адаптивним і коректно відображається як на персональних комп'ютерах, так і на планшетних пристроях.

База даних системи зберігається у вигляді локального файлу SQLite , у якому розміщуються таблиці облікових записів та записів користувачів. Окремою умовою працездатності є наявність стабільного мережевого з'єднання між клієнтом і сервером, оскільки всі операції з даними виконуються через звернення клієнтського додатку до REST API серверної частини. У разі тимчасової недоступності сервера клієнт повідомляє користувача про помилку та не виконує операцій із даними до відновлення зв'язку.

4.3 Виконання програмного забезпечення

Після запуску серверної та клієнтської частин і відкриття веб-додатку у браузері користувача перенаправляє на сторінку авторизації, оскільки доступ до основної функціональності можливий лише після успішного входу до системи. Сторінка авторизації містить поля для введення електронної пошти й пароля та кнопку входу, а також посилання на сторінку реєстрації для нових

користувачів. Для першого входу передбачено обліковий запис адміністратора з електронною поштою `admin@urs.local` та паролем `admin123`, який створюється автоматично під час першого запуску сервера. Після запуску серверної та клієнтської частин і відкриття веб-додатку у браузері користувача перенаправляє на сторінку авторизації. На рисунку 4.1 наведено сторінку авторизації програмної системи.

Рисунок 4.1 – Сторінка авторизації

Користувачі, які ще не мають облікового запису, переходять на сторінку реєстрації. Вона містить поля для введення електронної пошти, пароля та повторного підтвердження пароля. Перед надсиланням даних на сервер виконується перевірка збігу введених паролів, а також перевірка коректності формату електронної пошти. У разі успішної реєстрації створюється новий обліковий запис із роллю оператора, після чого користувач може увійти до системи. Для створення нового облікового запису користувач переходить на

сторінку реєстрації. На рисунку 4.2 наведено сторінку реєстрації програмної системи.

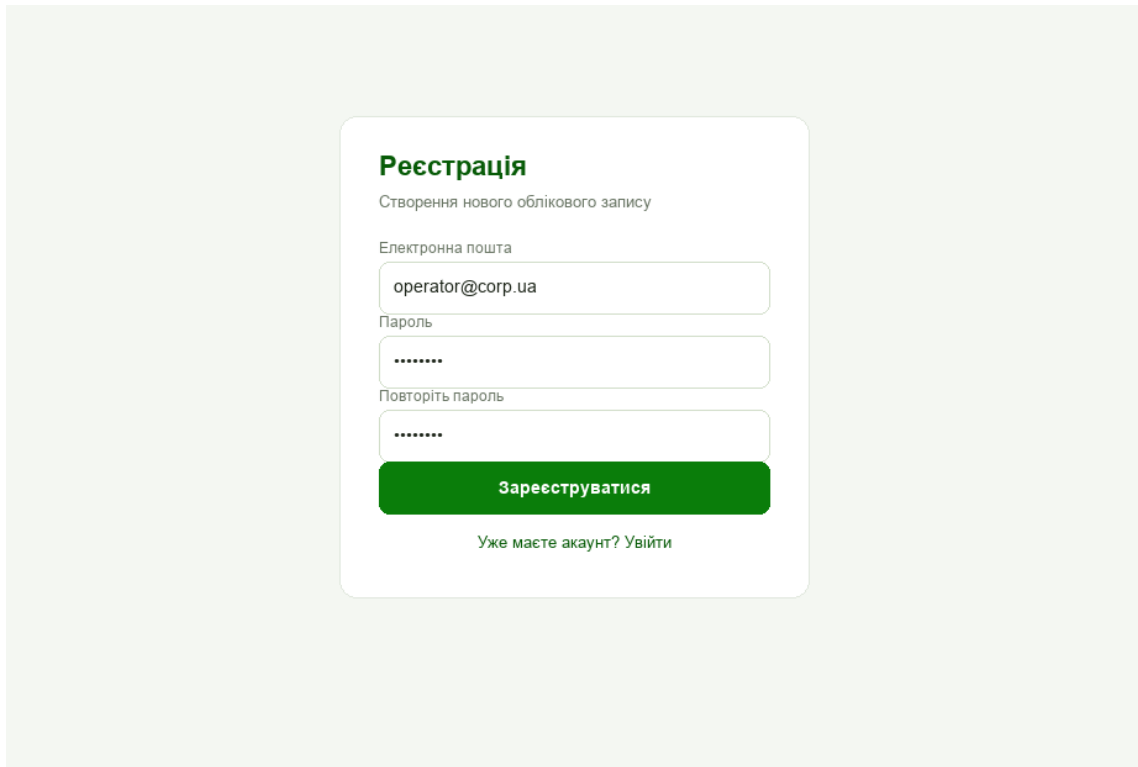


Рисунок 4.2 – Сторінка реєстрації

Після успішної авторизації відкривається головна сторінка системи — інформаційний дашборд. У його верхній частині розташовані картки узагальненої статистики: загальна кількість записів, кількість активних, неактивних та заблокованих записів, а також розподіл за відділами. Нижче розміщено таблицю записів користувачів зі стовпцями прізвища, імені та по батькові, електронної пошти, телефону, відділу, посади й статусу, доповнену елементами керування — полем пошуку, фільтрами, кнопками додавання запису та експорту. Після успішної авторизації користувач отримує доступ до головної сторінки системи. На рисунку 4.3 наведено головну сторінку програмної системи з інформаційним дашбордом та списком записів.

Система опрацювання записів користувачів					
Записи		Профіль		Вийти	
12 Усього записів	9 Активні	2 Неактивні	1 Заблоковані		
Пошук за ПІБ, email або телефоном...		Усі статуси	Усі відділи	+ Додати запис	Експорт CSV
ПІБ	Email	Телефон	Відділ	Статус	Дії
Іваненко Олександр Петрович	ivanenko@corp.ua	+380501112233	Бухгалтерія	Активний	 
Коваленко Марія Сергіївна	kovalenko@corp.ua	+380672223344	Кадри	Активний	 
Шевченко Андрій Олегович	shevchenko@corp.ua	+380633334455	ІТ	Активний	 
Бондаренко Наталія Ігорівна	bondarenko@corp.ua	+380504445566	Маркетинг	Неактивний	 
Ткаченко Віктор Михайлович	tkachenko@corp.ua	+380675556677	ІТ	Активний	 
Кравченко Дмитро Юрійович	kravchenko@corp.ua	+380507778899	Продажі	Заблокований	 
Мельник Ольга Василівна	melnik@corp.ua	+380636667788	Бухгалтерія	Активний	 

Рисунок 4.3 – Головна сторінка (дашборд зі списком записів)

Робота зі списком записів організована за допомогою кількох взаємодоповнювальних механізмів. Поле пошуку дозволяє знаходити записи за підрядком у прізвищі, імені та по батькові, електронній пошті або телефоні без урахування регістру символів. Фільтри надають можливість звужити вибірку за статусом, відділом чи роллю. Натискання на заголовок стовпця таблиці виконує сортування записів за відповідним полем у зростаючому або спадному порядку. Для зручної навігації великими обсягами даних застосовано пагінацію: записи відображаються посторінково, а перемикання між сторінками здійснюється відповідними елементами керування. Кожна зміна параметрів пошуку, фільтрації чи сортування призводить до повторного запиту до сервера й оновлення вмісту таблиці.

Створення нового запису ініціюється кнопкою додавання, після чого відкривається модальна форма з полями для введення відомостей про користувача. Та сама форма використовується для редагування наявного запису, проте в цьому випадку її поля заповнюються поточними значеннями. Після введення даних і підтвердження виконується їх валідація, і за умови

коректності запис зберігається у базі даних, а таблиця оновлюється. Для створення нового запису або редагування наявного використовується спеціальна модальна форма. На рисунку 4.4 наведено форму створення та редагування запису користувача.

Рисунок 4.4 – Форма створення/редагування запису

У разі введення некоректних або неповних даних система не зберігає запис, а виводить повідомлення про помилку валідації безпосередньо у формі. Зокрема, перевіряється правильність формату електронної пошти, заповнення обов'язкового поля прізвища, імені та по батькові, а також допустима довжина введених значень. Таким чином система запобігає потраплянню до бази даних некоректної інформації та підказує користувачеві, які саме поля потребують виправлення. Для контролю коректності введених даних у системі реалізовано механізм валідації. На рисунку 4.5 наведено приклад повідомлень про помилки валідації під час заповнення форми запису.

Система опрацювання записів користувачів

Записи Профіль Вийти

12
Усього записів

Пошук за ПІБ, email або телефоном...

ПІБ

Іваненко Олександр Петрович

Коваленко Марія Сергіївна

Шевченко Андрій Олегович

Бондаренко Наталія Ігорівна

Ткаченко Віктор Михайлович

Кравченко Дмитро Юрійович

Мельник Ольга Василівна

Новий запис

ПІБ *

Вкажіть ПІБ (мінімум 2 символи)

Email

bad-email

Некоректний формат email

Телефон

+380501234567

Посада

Інженер

Статус

Активний

Дата народження

1992-05-14

Нотатки

Скасувати Зберегти

1
Заблоковані

Експорт CSV

Статус	Дії
Активний	
Активний	
Активний	
Неактивний	
Активний	
Заблокований	
Активний	

Рисунок 4.5 – Приклад повідомлення про помилку валідації

Для детального ознайомлення з конкретним записом передбачено сторінку детальної інформації, яка відкривається під час вибору запису у таблиці. На ній відображаються всі відомості про користувача, зокрема прізвище, ім'я та по батькові, контактні дані, відділ, посада, статус, дата народження та нотатки, а також службові поля з датами створення й останнього оновлення запису. Зі сторінки деталей адміністратор може перейти до редагування або видалення запису. Для детального ознайомлення з інформацією про користувача в системі передбачено окрему сторінку перегляду запису. На рисунку 4.6 наведено сторінку детальної інформації про запис користувача.

The screenshot shows a web application interface for user management. At the top, a green header bar contains the title 'Система опрацювання записів користувачів' and navigation links 'Записи', 'Профіль', and 'Вийти'. Below the header, a button labeled '← До списку' is visible. The main content area displays the profile of 'Іваненко Олександр Петрович'. This profile is presented as a table with two columns: labels on the left and values on the right. The labels include 'ПІБ', 'Email', 'Телефон', 'Відділ', 'Посада', 'Статус', 'Дата народження', 'Нотатки', 'Створено', and 'Оновлено'. The corresponding values are: 'Іваненко Олександр Петрович', 'ivanenko@corp.ua', '+380501112233', 'Бухгалтерія', 'Бухгалтер', 'Активний', '1990-01-01', an empty field, '01.06.2026, 21:05', and '01.06.2026, 21:05'.

ПІБ	Іваненко Олександр Петрович
Email	ivanenko@corp.ua
Телефон	+380501112233
Відділ	Бухгалтерія
Посада	Бухгалтер
Статус	Активний
Дата народження	1990-01-01
Нотатки	—
Створено	01.06.2026, 21:05
Оновлено	01.06.2026, 21:05

Рисунок 4.6 – Сторінка детальної інформації про запис

Окрім роботи з окремими записами, система надає можливість експорту поточної вибірки у формат CSV. Під час натискання кнопки експорту сервер формує файл із урахуванням застосованих параметрів пошуку та фільтрації, після чого браузер завантажує його на пристрій користувача для подальшого використання у зовнішніх програмах. Завершення роботи із системою виконується через функцію виходу, яка видаляє збережений токен авторизації та повертає користувача на сторінку входу, унеможливаючи подальший доступ без повторної авторизації.

4.4 Тестування програмного забезпечення

Для підтвердження відповідності розробленої системи висунутим вимогам та забезпечення її надійності було проведено комплексне тестування, що охопило три рівні: модульне, інтеграційне та функціональне тестування. Кожен рівень спрямований на перевірку різних аспектів роботи програмного забезпечення — від коректності окремих функцій до правильності виконання повних користувацьких сценаріїв.

Модульне тестування передбачало перевірку окремих функцій системи у відриві від решти компонентів. Зокрема, було протестовано функції валідації вхідних даних, які перевіряють формат електронної пошти, заповненість обов'язкових полів та допустиму довжину значень, а також функцію формування файлу CSV, що відповідає за коректне екранування спеціальних символів, формування заголовків стовпців та послідовне виведення рядків. Перевірялася поведінка цих функцій як для коректних, так і для свідомо помилкових вхідних даних, що дозволило переконатися у правильності їхньої роботи в усіх передбачених випадках.

Інтеграційне тестування мало на меті перевірити коректність взаємодії клієнтської частини, серверної частини та бази даних. У його межах було протестовано всі ендпоінти REST API : реєстрацію та авторизацію, отримання переліку й деталей записів, створення, оновлення та видалення записів, експорт CSV і отримання статистики. Перевірялося, що дані, надіслані клієнтом, коректно зберігаються у базі даних SQLite та повертаються у незмінному вигляді під час наступних запитів, а сервер формує відповіді з належними кодами стану HTTP.

Функціональне тестування полягало у проходженні повного сценарію роботи з системою: вхід до системи, перегляд переліку записів, виконання пошуку, створення нового запису, його редагування, видалення, експорт вибірки у CSV та вихід із системи. Цей наскрізний сценарій підтвердив узгоджену роботу всіх функціональних модулів у послідовності, що відповідає реальному використанню системи кінцевими користувачами.

Окрему увагу було приділено перевірці валідації вхідних даних та розмежуванню доступу. Під час тестування валідації введення некоректної електронної пошти або порожнього поля прізвища, імені та по батькові очікувано блокувалося із виведенням відповідного повідомлення про помилку, а запис не зберігався у базі даних. Під час перевірки розмежування доступу було підтверджено, що оператор не має змоги видаляти записи, а будь-який запит до захищених ендпоінтів без дійсного токена авторизації відхиляється сервером із

кодом стану 401. Додатково перевірено коректність роботи пошуку за різними полями, фільтрації за статусом, відділом і роллю, сортування та посторінкового виведення записів за допомогою пагінації.

За результатами проведеного тестування всі передбачені тестові сценарії пройдено успішно — система поводитися відповідно до очікувань на всіх рівнях перевірки. Виявлені під час тестування дрібні недоліки, що стосувалися переважно оброблення граничних випадків та відображення повідомлень про помилки, було усунуто, після чого відповідні тести повторено повторно з позитивним результатом.

4.5 Висновки за розділом 4

У цьому розділі описано призначення, умови та порядок застосування розробленої програмної системи опрацювання записів користувачів. Визначено категорії користувачів — адміністраторів та операторів, окреслено перелік функцій системи, склад вхідних і вихідних даних, а також наведено послідовність дій для розгортання й запуску клієнтської та серверної частин.

Детально розглянуто виконання програмного забезпечення з ілюстрацією основних екранів — сторінок авторизації та реєстрації, головного дашборда зі списком записів, форми створення й редагування запису, повідомлення про помилку валідації та сторінки детальної інформації. Продемонстровано роботу механізмів пошуку, фільтрації, сортування, пагінації, експорту даних у формат CSV та виходу із системи.

Проведене комплексне тестування на модульному, інтеграційному та функціональному рівнях підтвердило коректність роботи системи, надійність валідації вхідних даних та правильність розмежування доступу за ролями. Усі тестові сценарії пройдено успішно, а виявлені дрібні недоліки усунуто, що свідчить про готовність розробленої системи до практичного використання.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи здійснено програмну реалізацію системи опрацювання записів користувачів. Розроблений програмний продукт являє собою веб-орієнтований застосунок, призначений для централізованого створення, перегляду, редагування, видалення, пошуку, фільтрування та експортування записів користувачів у межах організації. Створена система забезпечує цілісність даних, розмежування доступу за ролями та оперативний доступ до актуальної інформації, замінюючи ручний та напівавтоматичний облік у табличних формах.

Для досягнення поставленої мети виконано низку завдань:

- розглянуто основні поняття предметної області опрацювання записів користувачів;
- проаналізовано наявні способи зберігання та опрацювання записів;
- розглянуто сучасні системи управління записами користувачів;
- виконано порівняльний аналіз наявного програмного забезпечення відповідного призначення;
- обґрунтовано вибір мови програмування TypeScript та технологій React, Node.js/Express і SQLite;
- спроектовано структуру системи та логічну й фізичну модель бази даних;
- розроблено серверну частину системи у вигляді REST API та клієнтську частину у вигляді односторінкового застосунку;
- описано порядок виконання програми та взаємодію її складових;
- проведено тестування розробленого програмного забезпечення.

Під час реалізації системи використано низку сучасних технологій. Як єдину мову програмування для серверної та клієнтської частин обрано TypeScript, що забезпечило статичну типізацію та підвищило надійність коду. Клієнтську частину побудовано на бібліотеці React, серверну частину реалізовано засобами платформи Node.js та фреймворку Express, а для

зберігання даних застосовано реляційну файлову базу даних SQLite. Обмін даними між клієнтом і сервером організовано за принципами архітектурного стилю REST із передаванням інформації у форматі JSON, а автентифікацію та авторизацію реалізовано на основі механізму JWT. Загалом систему побудовано за архітектурою односторінкового застосунку (SPA), що взаємодіє із серверним REST API, який, своєю чергою, звертається до бази даних, що забезпечило чітке розмежування рівнів представлення, бізнес-логіки та зберігання даних.

Результати проведеного тестування підтвердили стабільність роботи розробленого програмного забезпечення за різних сценаріїв використання та відсутність критичних помилок. Установлено, що поведінка системи відповідає сформульованим функціональним і нефункціональним вимогам, а опрацювання вхідних даних супроводжується належною валідацією та коректним інформуванням користувача про помилки. Це дає підстави стверджувати, що створена система є придатною до практичного застосування у межах організації.

Розроблена система реалізує такі функціональні можливості:

- реєстрація та авторизація користувачів;
- рольове розмежування доступу до функцій системи;
- створення, перегляд, редагування та видалення записів користувачів (CRUD);
- пошук записів за прізвищем, електронною поштою та номером телефону;
- фільтрування та сортування записів за заданими критеріями;
- пагінація списку записів;
- експортування вибірки записів у формат CSV;
- формування статистичних показників для інформаційного дашборда.

Отже, у результаті виконання роботи всі поставлені завдання виконано в повному обсязі, а мету роботи — створення програмної системи опрацювання записів користувачів — досягнуто. Розроблений програмний продукт має практичну цінність і може бути впроваджений для автоматизації обліку записів

користувачів в організації. Подальше вдосконалення системи доцільно спрямувати на впровадження журналу аудиту дій користувачів, інтеграцію із зовнішніми системами та сервісами, а також розширення аналітичних можливостей шляхом додавання поглибленої звітності та засобів візуалізації даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. TypeScript Documentation. Microsoft. URL: <https://www.typescriptlang.org/docs/> (date of access: 12.05.2026).
2. React. A JavaScript library for building user interfaces. Meta. URL: <https://react.dev/> (date of access: 14.05.2026).
3. Node.js Documentation. OpenJS Foundation. URL: <https://nodejs.org/en/docs> (date of access: 15.05.2026).
4. Express — Node.js web application framework. URL: <https://expressjs.com/> (date of access: 15.05.2026).
5. SQLite Documentation. URL: <https://www.sqlite.org/docs.html> (date of access: 16.05.2026).
6. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: PhD dissertation. University of California, Irvine, 2000. URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> (date of access: 19.05.2026).
7. What is a REST API? Red Hat. URL: <https://www.redhat.com/en/topics/api/what-is-a-rest-api> (date of access: 19.05.2026).
8. Introduction to JSON Web Tokens. URL: <https://jwt.io/introduction> (date of access: 18.05.2026).
9. Role-Based Access Control. NIST. URL: <https://csrc.nist.gov/projects/role-based-access-control> (date of access: 23.05.2026).
10. General Data Protection Regulation (GDPR). URL: <https://gdpr-info.eu/> (date of access: 25.05.2026).
11. Про захист персональних даних: Закон України від 01.06.2010 № 2297-VI. URL: <https://zakon.rada.gov.ua/laws/show/2297-17> (date of access: 26.05.2026).
12. Microsoft Entra ID Documentation. URL: <https://learn.microsoft.com/en-us/entra/identity/> (date of access: 25.05.2026).
13. Keycloak Documentation. URL: <https://www.keycloak.org/documentation> (date of access: 26.05.2026).
14. Directus Documentation. URL: <https://docs.directus.io/> (date of access: 26.05.2026).
15. phpMyAdmin Documentation. URL: <https://docs.phpmyadmin.net/> (date of access: 27.05.2026).
16. JavaScript. MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (date of access: 12.05.2026).
17. Single-page application. MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Glossary/SPA> (date of access: 20.05.2026).
18. React Router. URL: <https://reactrouter.com/> (date of access: 21.05.2026).
19. Angular. Google. URL: <https://angular.dev/> (date of access: 14.05.2026).

- 20.better-sqlite3. npm. URL: <https://www.npmjs.com/package/better-sqlite3> (date of access: 16.05.2026).
- 21.Vite. Next Generation Frontend Tooling. URL: <https://vitejs.dev/> (date of access: 20.05.2026).
- 22.Axios. Promise based HTTP client. URL: <https://axios-http.com/> (date of access: 21.05.2026).
- 23.Bcrypt. Wikipedia. URL: <https://en.wikipedia.org/wiki/Bcrypt> (date of access: 18.05.2026).
- 24.Zod. TypeScript-first schema validation. URL: <https://zod.dev/> (date of access: 22.05.2026).
- 25.Common Format and MIME Type for Comma-Separated Values (CSV) Files: RFC 4180. URL: <https://www.rfc-editor.org/rfc/rfc4180> (date of access: 24.05.2026).

ДОДАТОК А
Технічне завдання

Вступ

Цей застосунок присвячено автоматизації опрацювання записів користувачів організації. Технічне завдання визначає підставу для розробки, призначення програмної системи, перелік функціональних та нефункціональних вимог, а також вимоги до інтерфейсу, надійності, безпеки і технічних засобів, потрібних для розгортання та експлуатації веб-орієнтованого програмного продукту.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Створення програмної системи опрацювання записів користувачів», затверджене наказом Національного університету «Запорізька політехніка» № 139 від 07 квітня 2026 р.

A.2 Призначення розробки

Розроблювана система призначена для автоматизації процесів опрацювання записів користувачів організації, а саме централізованого створення, перегляду, редагування, видалення, пошуку, фільтрації та експорту відомостей про осіб. Програмний продукт замінює ручний та напівавтоматичний облік даних у електронних таблицях, забезпечуючи цілісність інформації, розмежування доступу за ролями та оперативний доступ до актуальних відомостей.

Система реалізується як веб-додаток з клієнт-серверною архітектурою, що дає змогу одночасно обслуговувати декількох користувачів через сучасний веб-браузер без потреби встановлення додаткового програмного забезпечення на робочих місцях. Основними користувачами є адміністратори, які здійснюють

повне керування записами, та оператори, які виконують перегляд відомостей і редагування власного профілю.

A.3 Вимоги до функціональних характеристик

Система реалізує такі функціональні вимоги:

- реєстрація та авторизація користувачів;
- рольове розмежування доступу (admin/user);
- створення, перегляд, редагування та видалення записів;
- пошук за ПІБ, email та телефоном;
- фільтрація за статусом і відділом;
- сортування та пагінація;
- експорт записів у формат CSV;
- формування статистики;
- валідація вхідних даних.

A.4 Вимоги до інтерфейсу

Інтерфейс користувача має бути лаконічним та інтуїтивно зрозумілим, забезпечувати швидке опанування системи без спеціального навчання. Передбачається адаптивність відображення для роботи на пристроях з різною роздільною здатністю екрана, а також наявність зворотного зв'язку щодо виконуваних дій. Система повинна надавати зрозумілі повідомлення про помилки, зокрема під час валідації введених даних, щоб користувач міг своєчасно виправити некоректні відомості.

A.5 Вимоги до надійності та безпеки

Для забезпечення надійності та безпеки система повинна зберігати паролі користувачів виключно у вигляді хешів, обчислених за алгоритмом

bcrypt, без зберігання їх у відкритому вигляді. Автентифікація користувачів реалізується на основі токенів JWT, що передаються у заголовок запиту та перевіряються на стороні сервера під час кожного звернення до захищених ресурсів.

Кожна операція супроводжується перевіркою ролі користувача для розмежування доступу та запобігання несанкціонованим діям. Усі вхідні дані підлягають валідації на сервері, що забезпечує цілісність бази даних та захищає від некоректних запитів. Персональні дані користувачів захищаються від несанкціонованого доступу засобами автентифікації, авторизації та контролю прав доступу.

А.6 Вимоги до складу та параметрів технічних засобів

Для функціонування серверної частини системи потрібне середовище виконання Node.js версії 18 або новішої. Клієнтська частина працює у сучасному веб-браузері (Chrome, Edge, Firefox) з підтримкою стандартів HTML5 та JavaScript. Базою даних слугує СУБД SQLite, що зберігає інформацію у файлі та не потребує окремого сервера баз даних. Розробку системи здійснено в інтегрованому середовищі Visual Studio Code.

ДОДАТОК Б
Опис програми

Б.1 Загальні відомості

Програмне забезпечення має назву «Система опрацювання записів користувачів» (англ. "Creating a software system for processing user records"). Воно призначене для централізованого опрацювання записів користувачів у межах організації, а саме для їх створення, перегляду, редагування, видалення, пошуку, фільтрування, сортування та експортування. Розроблена система замінює ручний або напівавтоматичний облік даних у таблицях, забезпечує цілісність інформації, розмежування доступу за ролями та оперативний доступ до актуальних даних.

Програмний продукт реалізовано як веб-орієнтований застосунок з єдиною сторінкою (Single Page Application). Як основну мову програмування для клієнтської та серверної частин обрано TypeScript. Клієнтську частину побудовано на бібліотеці React, серверну — на платформі Node.js із застосуванням каркаса Express, а для зберігання даних використано реляційну базу даних SQLite. Розроблення програмного забезпечення здійснювалося в інтегрованому середовищі розробки Visual Studio Code.

Б.2 Функціональне призначення

Система реалізує такі функції:

- реєстрація та авторизація користувачів за електронною поштою та паролем із застосуванням токена JWT;
- розмежування доступу за ролями (адміністратор та оператор) на основі моделі RBAC;
- створення, перегляд, редагування та видалення записів користувачів (операції CRUD);
- пошук записів за прізвищем, ім'ям, електронною поштою або номером телефону;

- фільтрування записів за статусом, відділом і роллю, а також сортування за стовпцями;
- посторінкове виведення (пагінація) списку записів;
- експортування вибірки записів у формат CSV;
- формування інформаційної статистики для дашборда (загальна кількість записів, кількість активних, неактивних і заблокованих записів, розподіл за відділами);
- валідація вхідних даних (перевірка формату електронної пошти, обов'язкових полів та довжини значень).

Б.3 Опис логічної структури

Програмну систему побудовано за клієнт-серверною архітектурою з застосуванням підходу єдиної сторінки (SPA). Логічно вона поділяється на три основні складові: клієнтську частину, серверну частину та базу даних. Взаємодія між клієнтом і сервером відбувається через інтерфейс REST API, дані передаються у форматі JSON засобами протоколу HTTP, а автентифікація користувачів забезпечується за допомогою токенів JWT, що передаються у заголовку Authorization.

Клієнтська частина реалізована на бібліотеці React і охоплює кореневий модуль App, підсистему маршрутизації (React Router), контекст автентифікації для зберігання стану поточного користувача й токена, набір сторінок (вхід, реєстрація, дашборд, форма запису, деталі запису, профіль), повторно використовувані компоненти інтерфейсу та модуль арі-клієнта на основі бібліотеки Axios, що інкапсулює виклики до серверних ендпоінтів.

Серверна частина побудована на платформі Node.js із каркасом Express і складається з вхідного модуля index, що ініціалізує застосунок та реєструє маршрути, модуля db для роботи з базою даних, модуля auth для реєстрації, входу та формування токенів, модуля records для опрацювання операцій над записами, а також проміжного програмного забезпечення (middleware), що

перевіряє дійсність токена та права доступу відповідно до ролі користувача. База даних SQLite містить дві таблиці: accounts (облікові записи для входу) та records (записи користувачів), між якими встановлено зв'язок один-до-багатьох, оскільки один обліковий запис може створити багато записів користувачів.

Загальну структурну схему роботи додатку, що відображає взаємодію клієнтської частини, серверної частини та бази даних, наведено на рисунку Б.1.

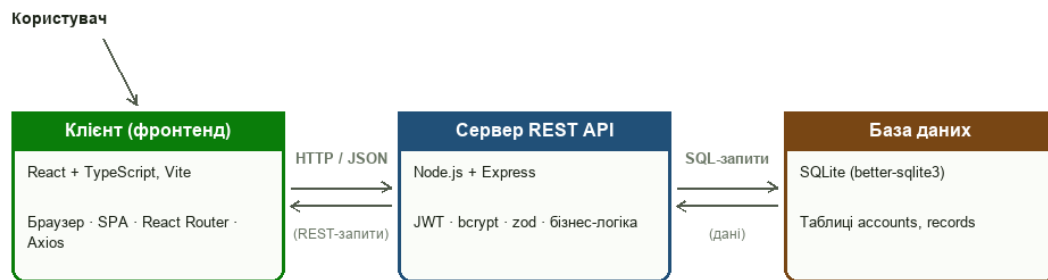


Рисунок Б.1 – Загальна структурна схема роботи додатку

Б.4 Використані технічні засоби

Для функціонування серверної частини програмної системи потрібна платформа Node.js версії 18 або новішої; для роботи з клієнтською частиною достатньо сучасного веб-браузера (Chrome, Edge або Firefox). Зберігання даних здійснюється засобами файлової реляційної бази даних SQLite, що не потребує окремого сервера й спрощує розгортання.

Під час розроблення використано такі бібліотеки та інструменти:

- React, React Router та Axios — для побудови клієнтського інтерфейсу, маршрутизації та виконання HTTP-запитів;
- Vite — для збирання клієнтської частини;
- Express — для реалізації серверного REST API;

- `better-sqlite3` — для доступу до бази даних SQLite;
- `jsonwebtoken` — для формування та перевірки токенів JWT;
- `bcryptjs` — для хешування паролів;
- `zod` — для валідації вхідних даних.

ДОДАТОК В
Текст програми

B.1 db.ts — ініціалізація бази даних SQLite

```
// db.ts — ініціалізація бази даних SQLite та початкове наповнення
import Database from 'better-sqlite3';
import bcrypt from 'bcryptjs';

export const db = new Database('data.sqlite');
db.pragma('journal_mode = WAL');

export function initDb(): void {
  db.exec(`
    CREATE TABLE IF NOT EXISTS accounts (
      id          INTEGER PRIMARY KEY AUTOINCREMENT,
      email       TEXT UNIQUE NOT NULL,
      passwordHash TEXT NOT NULL,
      role        TEXT NOT NULL DEFAULT 'user',
      createdAt   TEXT NOT NULL
    );
    CREATE TABLE IF NOT EXISTS records (
      id          INTEGER PRIMARY KEY AUTOINCREMENT,
      fullName    TEXT NOT NULL,
      email       TEXT,
      phone       TEXT,
      department  TEXT,
      position    TEXT,
      status      TEXT NOT NULL DEFAULT 'active',
      birthDate   TEXT,
      notes       TEXT,
      ownerId     INTEGER,
      createdAt   TEXT NOT NULL,
      updatedAt   TEXT NOT NULL,
      FOREIGN KEY (ownerId) REFERENCES accounts(id)
    );
    CREATE INDEX IF NOT EXISTS idx_records_fullName ON records(fullName);
    CREATE INDEX IF NOT EXISTS idx_records_status   ON records(status);
  `);
  seed();
}

function seed(): void {
  const { c } = db.prepare('SELECT COUNT(*) AS c FROM accounts').get() as { c: number };
  if (c > 0) return;

  const now = new Date().toISOString();
  const adminId = db
    .prepare('INSERT INTO accounts (email, passwordHash, role, createdAt) VALUES (?, ?, ?, ?)')
    .run('admin@urs.local', bcrypt.hashSync('admin123', 10), 'admin', now)
    .lastInsertRowid as number;

  const samples = [
    ['Іваненко Олександр Петрович', 'ivanenko@corp.ua', '+380501112233', 'Бухгалтерія', 'Бухгалтер', 'active'],
    ['Коваленко Марія Сергіївна', 'kovalenko@corp.ua', '+380672223344', 'Кадри', 'HR-менеджер', 'active'],
    ['Шевченко Андрій Олегович', 'shevchenko@corp.ua', '+380633334455', 'IT', 'Розробник', 'active'],
    ['Бондаренко Наталія Ігорівна', 'bondarenko@corp.ua', '+380504445566', 'Маркетинг', 'Маркетолог', 'inactive'],
    ['Ткаченко Віктор Михайлович', 'tkachenko@corp.ua', '+380675556677', 'IT', 'Системний адміністратор', 'active'],
    ['Мельник Ольга Василівна', 'melnyk@corp.ua', '+380636667788', 'Бухгалтерія', 'Головний бухгалтер', 'active'],
  ];
}
```

```

    ['Кравченко Дмитро Юрійович', 'kravchenko@corp.ua', '+380507778899', 'Продажі',
    'Менеджер з продажів', 'blocked'],
    ['Поліщук Ірина Романівна', 'polishchuk@corp.ua', '+380678889900', 'Кадри',
    'Рекрутер', 'active'],
    ['Гриценко Сергій Анатолійович', 'hrytsenko@corp.ua', '+380639990011', 'IT',
    'Тестувальник', 'inactive'],
    ['Лисенко Тетяна Борисівна', 'lysenko@corp.ua', '+380500011122', 'Маркетинг', 'SMM-
    фахівець', 'active'],
    ['Савченко Юрій Володимирович', 'savchenko@corp.ua', '+380671122334', 'Продажі',
    'Керівник відділу', 'active'],
    ['Романенко Аліна Дмитрівна', 'romanenko@corp.ua', '+380632233445', 'IT', 'UX/UI
    дизайнер', 'active'],
    ];

    const ins = db.prepare(`
        INSERT INTO records (fullName, email, phone, department, position, status,
        birthDate, notes, ownerId, createdAt, updatedAt)
        VALUES (@fullName, @email, @phone, @department, @position, @status, @birthDate,
        @notes, @ownerId, @createdAt, @updatedAt)
    `);
    const tx = db.transaction(() => {
        for (const [fullName, email, phone, department, position, status] of samples) {
            ins.run({
                fullName, email, phone, department, position, status,
                birthDate: '1990-01-01', notes: '', ownerId: adminId, createdAt: now,
                updatedAt: now,
            });
        }
    });
    tx();
    console.log('БД ініціалізовано: створено адміністратора (admin@urs.local / admin123)
    і %d записів.', samples.length);
}

```

B.2 middleware.ts — автентифікація JWT та перевірка ролей

```

// middleware.ts — автентифікація за JWT та перевірка ролей (RBAC)
import type { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';

export const JWT_SECRET = process.env.JWT_SECRET ?? 'urs-dev-secret-change-me';

export interface AuthPayload {
    id: number;
    email: string;
    role: 'admin' | 'user';
}

// Розширюємо тип Request полем user
declare global {
    // eslint-disable-next-line @typescript-eslint/no-namespace
    namespace Express {
        interface Request {
            user?: AuthPayload;
        }
    }
}

export function authRequired(req: Request, res: Response, next: NextFunction): void {
    const header = req.headers.authorization;
    if (!header?.startsWith('Bearer ')) {
        res.status(401).json({ error: 'Потрібна автентифікація' });
    }
}

```

```

    return;
  }
  try {
    req.user = jwt.verify(header.slice(7), JWT_SECRET) as AuthPayload;
    next();
  } catch {
    res.status(401).json({ error: 'Недійсний або прострочений токен' });
  }
}

export function adminOnly(req: Request, res: Response, next: NextFunction): void {
  if (req.user?.role !== 'admin') {
    res.status(403).json({ error: 'Недостатньо прав (потрібна роль admin)' });
    return;
  }
  next();
}

```

B.3 auth.ts — реєстрація, вхід, поточний користувач

```

// auth.ts — маршрути реєстрації, входу та отримання поточного користувача
import { Router } from 'express';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import { z } from 'zod';
import { db } from './db';
import { JWT_SECRET, authRequired, type AuthPayload } from './middleware';

export const authRouter = Router();

const credentials = z.object({
  email: z.string().email('Некоректний формат email'),
  password: z.string().min(6, 'Пароль має містити щонайменше 6 символів'),
});

authRouter.post('/register', (req, res) => {
  const parsed = credentials.safeParse(req.body);
  if (!parsed.success) {
    return res.status(400).json({ error: parsed.error.issues[0].message });
  }
  const { email, password } = parsed.data;
  const exists = db.prepare('SELECT id FROM accounts WHERE email = ?').get(email);
  if (exists) return res.status(409).json({ error: 'Користувач з таким email уже існує' });

  const now = new Date().toISOString();
  const role = 'user';
  const id = db
    .prepare('INSERT INTO accounts (email, passwordHash, role, createdAt) VALUES (?, ?, ?, ?)')
    .run(email, bcrypt.hashSync(password, 10), role, now)
    .lastInsertRowid as number;

  const payload: AuthPayload = { id, email, role };
  const token = jwt.sign(payload, JWT_SECRET, { expiresIn: '7d' });
  res.status(201).json({ token, user: payload });
});

authRouter.post('/login', (req, res) => {
  const parsed = credentials.safeParse(req.body);
  if (!parsed.success) {

```

```

    return res.status(400).json({ error: parsed.error.issues[0].message });
  }
  const { email, password } = parsed.data;
  const account = db
    .prepare('SELECT id, email, passwordHash, role FROM accounts WHERE email = ?')
    .get(email) as { id: number; email: string; passwordHash: string; role: 'admin' |
'user' } | undefined;

  if (!account || !bcrypt.compareSync(password, account.passwordHash)) {
    return res.status(401).json({ error: 'Невірний email або пароль' });
  }
  const payload: AuthPayload = { id: account.id, email: account.email, role:
account.role };
  const token = jwt.sign(payload, JWT_SECRET, { expiresIn: '7d' });
  res.json({ token, user: payload });
});

authRouter.get('/me', authRequired, (req, res) => {
  res.json({ user: req.user });
});

```

B.4 records.ts — CRUD записів, пошук, експорт

```

// records.ts — CRUD записів користувачів, пошук, фільтрація, експорт, статистика
import { Router } from 'express';
import { z } from 'zod';
import { db } from './db';
import { authRequired, adminOnly } from './middleware';

export const recordsRouter = Router();
recordsRouter.use(authRequired);

const recordSchema = z.object({
  fullName: z.string().min(2, 'ПІБ обов'язкове (мінімум 2 символи)'),
  email: z.string().email('Некоректний email').or(z.literal('')),
  phone: z.string().optional().default(''),
  department: z.string().optional().default(''),
  position: z.string().optional().default(''),
  status: z.enum(['active', 'inactive', 'blocked']).default('active'),
  birthDate: z.string().optional().default(''),
  notes: z.string().optional().default(''),
});

const SORT_COLUMNS = ['fullName', 'email', 'department', 'status', 'createdAt'] as
const;

// GET /api/records — список з пошуком, фільтрами, сортуванням, пагінацією
recordsRouter.get('/', (req, res) => {
  const q = String(req.query.q ?? '').trim();
  const status = String(req.query.status ?? '');
  const department = String(req.query.department ?? '');
  const sort = SORT_COLUMNS.includes(req.query.sort as any) ? String(req.query.sort) :
'createdAt';
  const dir = String(req.query.dir) === 'asc' ? 'ASC' : 'DESC';
  const page = Math.max(1, Number(req.query.page) || 1);
  const pageSize = Math.min(100, Math.max(1, Number(req.query.pageSize) || 10));

  const where: string[] = [];
  const params: Record<string, unknown> = {};
  if (q) {
    where.push('(fullName LIKE @q OR email LIKE @q OR phone LIKE @q)');
  }

```

```

    params.q = `%${q}%`;
  }
  if (status) { where.push('status = @status'); params.status = status; }
  if (department) { where.push('department = @department'); params.department =
department; }
  const whereSql = where.length ? `WHERE ${where.join(' AND ')}` : '';

  const total = (db.prepare(`SELECT COUNT(*) AS c FROM records
${whereSql}`)).get(params) as { c: number }.c;
  const rows = db
    .prepare(`SELECT * FROM records ${whereSql} ORDER BY ${sort} ${dir} LIMIT @limit
OFFSET @offset`)
    .all({ ...params, limit: pageSize, offset: (page - 1) * pageSize });

  res.json({ data: rows, total, page, pageSize, totalPages: Math.ceil(total / pageSize)
});
});

// GET /api/records/export - экспорт в CSV
recordsRouter.get('/export', (req, res) => {
  const rows = db.prepare('SELECT * FROM records ORDER BY fullName ASC').all() as
Record<string, unknown>[];
  const cols = ['id', 'fullName', 'email', 'phone', 'department', 'position', 'status',
'birthDate'];
  const esc = (v: unknown) => `"${String(v ?? '').replace(/"/g, '"')}"`;
  const csv = [cols.join(','), ...rows.map((r) => cols.map((c) =>
esc(r[c])).join(','))].join('\r\n');
  res.setHeader('Content-Type', 'text/csv; charset=utf-8');
  res.setHeader('Content-Disposition', 'attachment; filename="records.csv"');
  res.send('' + csv); // BOM для колектного UTF-8 в Excel
});

// GET /api/records/:id - детали
recordsRouter.get('/:id', (req, res) => {
  const row = db.prepare('SELECT * FROM records WHERE id = ?').get(req.params.id);
  if (!row) return res.status(404).json({ error: 'Запис не знайдено' });
  res.json(row);
});

// POST /api/records - створити
recordsRouter.post('/', (req, res) => {
  const parsed = recordSchema.safeParse(req.body);
  if (!parsed.success) return res.status(400).json({ error:
parsed.error.issues[0].message });
  const now = new Date().toISOString();
  const d = parsed.data;
  const id = db
    .prepare(`INSERT INTO records
(fullName,email,phone,department,position,status,birthDate,notes,ownerId,createdAt,updatedAt)
VALUES
(@fullName,@email,@phone,@department,@position,@status,@birthDate,@notes,@ownerId,@crea
tedAt,@updatedAt)`)
    .run({ ...d, ownerId: req.user!.id, createdAt: now, updatedAt: now
}).lastInsertRowid;
  res.status(201).json(db.prepare('SELECT * FROM records WHERE id = ?').get(id));
});

// PUT /api/records/:id - оновити
recordsRouter.put('/:id', (req, res) => {
  const existing = db.prepare('SELECT * FROM records WHERE id = ?').get(req.params.id);
  if (!existing) return res.status(404).json({ error: 'Запис не знайдено' });
  const parsed = recordSchema.safeParse(req.body);
  if (!parsed.success) return res.status(400).json({ error:
parsed.error.issues[0].message });
  const d = parsed.data;

```

```

    db.prepare(`UPDATE records SET
fullName=@fullName,email=@email,phone=@phone,department=@department,
position=@position,status=@status,birthDate=@birthDate,notes=@notes,updatedAt=@updatedAt
WHERE id=@id`)
    .run({ ...d, updatedAt: new Date().toISOString(), id: req.params.id });
    res.json(db.prepare('SELECT * FROM records WHERE id = ?').get(req.params.id));
  });

// DELETE /api/records/:id — видалити (тільки admin)
recordsRouter.delete('/:id', adminOnly, (req, res) => {
  const info = db.prepare('DELETE FROM records WHERE id = ?').run(req.params.id);
  if (info.changes === 0) return res.status(404).json({ error: 'Запис не знайдено' });
  res.status(204).end();
});

```

B.5 index.ts — точка входу REST API сервера

```

// index.ts — точка входу REST API сервера
import express from 'express';
import cors from 'cors';
import { db, initDb } from './db';
import { authRouter } from './auth';
import { recordsRouter } from './records';
import { authRequired } from './middleware';

initDb();

const app = express();
app.use(cors());
app.use(express.json());

app.use('/api/auth', authRouter);
app.use('/api/records', recordsRouter);

// GET /api/stats — агрегована статистика для дашборда
app.get('/api/stats', authRequired, (_req, res) => {
  const total = (db.prepare('SELECT COUNT(*) AS c FROM records')).get() as { c: number };
  const byStatus = db.prepare('SELECT status, COUNT(*) AS c FROM records GROUP BY status').all() as { status: string; c: number }[];
  const byDepartment = db.prepare('SELECT department, COUNT(*) AS c FROM records GROUP BY department ORDER BY c DESC').all() as { department: string; c: number }[];
  const stat = (s: string) => byStatus.find((x) => x.status === s)?.c ?? 0;
  res.json({
    total,
    active: stat('active'),
    inactive: stat('inactive'),
    blocked: stat('blocked'),
    byDepartment,
  });
});

app.get('/api/health', (_req, res) => res.json({ ok: true }));

const PORT = Number(process.env.PORT) || 4000;
app.listen(PORT, () => console.log(`REST API сервер запущено: http://localhost:${PORT}`));

```

B.6 api.ts — HTTP-клієнт (Axios) з JWT

```
// api.ts — HTTP-клієнт (Axios) з автоматичним додаванням JWT-токена
import axios from 'axios';

export const api = axios.create({ baseURL: '/api' });

api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

export interface UserRecord {
  id: number;
  fullName: string;
  email: string;
  phone: string;
  department: string;
  position: string;
  status: 'active' | 'inactive' | 'blocked';
  birthDate: string;
  notes: string;
  createdAt: string;
  updatedAt: string;
}

export interface RecordsPage {
  data: UserRecord[];
  total: number;
  page: number;
  pageSize: number;
  totalPages: number;
}

export interface Stats {
  total: number;
  active: number;
  inactive: number;
  blocked: number;
  byDepartment: { department: string; c: number }[];
}

export interface Query {
  q?: string;
  status?: string;
  department?: string;
  sort?: string;
  dir?: 'asc' | 'desc';
  page?: number;
  pageSize?: number;
}

export const RecordsApi = {
  list: (params: Query) => api.get<RecordsPage>('/records', { params }).then((r) => r.data),
  get: (id: number) => api.get<UserRecord>(`/records/${id}`).then((r) => r.data),
  create: (body: Partial<UserRecord>) => api.post<UserRecord>('/records', body).then((r) => r.data),
}
```

```

    update: (id: number, body: Partial<UserRecord>) =>
    api.put<UserRecord>(`/records/${id}`, body).then((r) => r.data),
    remove: (id: number) => api.delete(`/records/${id}`),
    stats: () => api.get<Stats>('/stats').then((r) => r.data),
    exportUrl: '/api/records/export',
  };
};

```

B.7 auth.tsx — контекст автентифікації

```

// auth.tsx — контекст автентифікації (зберігання токена, вхід, реєстрація, вихід)
import { createContext, useContext, useEffect, useState, type ReactNode } from 'react';
import { api } from './api';

export interface User {
  id: number;
  email: string;
  role: 'admin' | 'user';
}

interface AuthContextValue {
  user: User | null;
  loading: boolean;
  login: (email: string, password: string) => Promise<void>;
  register: (email: string, password: string) => Promise<void>;
  logout: () => void;
}

const AuthContext = createContext<AuthContextValue>(null!);

export function AuthProvider({ children }: { children: ReactNode }) {
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState(true);

  // Перевірка наявності сесії при старті застосунку
  useEffect(() => {
    const token = localStorage.getItem('token');
    if (!token) { setLoading(false); return; }
    api.get('/auth/me')
      .then((r) => setUser(r.data.user))
      .catch(() => localStorage.removeItem('token'))
      .finally(() => setLoading(false));
  }, []);

  const persist = (token: string, u: User) => {
    localStorage.setItem('token', token);
    setUser(u);
  };

  const login = async (email: string, password: string) => {
    const { data } = await api.post('/auth/login', { email, password });
    persist(data.token, data.user);
  };

  const register = async (email: string, password: string) => {
    const { data } = await api.post('/auth/register', { email, password });
    persist(data.token, data.user);
  };

  const logout = () => {

```

```

    localStorage.removeItem('token');
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, loading, login, register, logout }}>
      {children}
    </AuthContext.Provider>
  );
}

export const useAuth = () => useContext(AuthContext);

```

B.8 App.tsx — маршрутизація застосунку

```

// App.tsx — маршрутизація застосунку та захист приватних сторінок
import { Navigate, Route, Routes } from 'react-router-dom';
import { useAuth } from '../auth';
import Layout from './components/Layout';
import Login from './pages/Login';
import Register from './pages/Register';
import Dashboard from './pages/Dashboard';
import RecordDetails from './pages/RecordDetails';
import Profile from './pages/Profile';
import type { ReactNode } from 'react';

function Private({ children }: { children: ReactNode }) {
  const { user, loading } = useAuth();
  if (loading) return <div className="center">Завантаження...</div>;
  return user ? <>{children}</> : <Navigate to="/login" replace />;
}

export default function App() {
  const { user } = useAuth();
  return (
    <Routes>
      <Route path="/login" element={user ? <Navigate to="/" replace /> : <Login />} />
      <Route path="/register" element={user ? <Navigate to="/" replace /> : <Register
/>} />
      <Route element={<Private><Layout /></Private>}>
        <Route path="/" element={<Dashboard />} />
        <Route path="/records/:id" element={<RecordDetails />} />
        <Route path="/profile" element={<Profile />} />
      </Route>
      <Route path="*" element={<Navigate to="/" replace />} />
    </Routes>
  );
}

```

B.9 Layout.tsx — макет із навігацією

```

// Layout.tsx — спільний макет: верхня панель навігації та область контенту
import { Link, Outlet, useNavigate } from 'react-router-dom';
import { useAuth } from '../auth';

export default function Layout() {

```

```

const { user, logout } = useAuth();
const navigate = useNavigate();
const onLogout = () => { logout(); navigate('/login'); };

return (
  <>
    <header className="appbar">
      <span className="title">Система опрацювання записів користувачів</span>
      <Link to="/">Записи</Link>
      <Link to="/profile">Профіль</Link>
      <span style={{ opacity: 0.85 }}>{user?.email} ({user?.role})</span>
      <button className="link" onClick={onLogout}>Вийти</button>
    </header>
    <main className="container">
      <Outlet />
    </main>
  </>
);
}

```

B.10 RecordForm.tsx — форма запису

```

// RecordForm.tsx – модальна форма створення/редагування запису користувача
import { useState, type FormEvent } from 'react';
import type { UserRecord } from '../api';

const empty: Partial<UserRecord> = {
  fullName: '', email: '', phone: '', department: '', position: '', status: 'active',
  birthDate: '', notes: '',
};

interface Props {
  initial?: UserRecord;
  onCancel: () => void;
  onSave: (data: Partial<UserRecord>) => Promise<void>;
}

export default function RecordForm({ initial, onCancel, onSave }: Props) {
  const [form, setForm] = useState<Partial<UserRecord>>(initial ?? empty);
  const [error, setError] = useState('');
  const [busy, setBusy] = useState(false);

  const set = (k: keyof UserRecord, v: string) => setForm((f) => ({ ...f, [k]: v }));

  const validate = (): string | null => {
    if (!form.fullName || form.fullName.trim().length < 2) return 'Вкажіть ПІБ (мінімум 2 символи)';
    if (form.email && !/^[^@\s]+@[^@\s]+\.[^@\s]+$/.test(form.email)) return 'Некоректний формат email';
    return null;
  };

  const onSubmit = async (e: FormEvent) => {
    e.preventDefault();
    const v = validate();
    if (v) { setError(v); return; }
    setBusy(true);
    try {
      await onSave(form);
    } catch (err: any) {

```

```

        setError(err.response?.data?.error ?? 'Не вдалося зберегти запис');
    } finally {
        setBusy(false);
    }
};

return (
    <div className="modal-overlay" onMouseDown={onCancel}>
        <form className="modal" onSubmit={onSubmit} onMouseDown={e =>
e.stopPropagation()}>
            <h2>{initial ? 'Редагування запису' : 'Новий запис'}</h2>
            <div className="field">
                <label>ПІБ *</label>
                <input value={form.fullName ?? ''} onChange={e => set('fullName',
e.target.value)} />
            </div>
            <div className="grid2">
                <div className="field">
                    <label>Email</label>
                    <input value={form.email ?? ''} onChange={e => set('email',
e.target.value)} />
                </div>
                <div className="field">
                    <label>Телефон</label>
                    <input value={form.phone ?? ''} onChange={e => set('phone',
e.target.value)} />
                </div>
            </div>
            <div className="grid2">
                <div className="field">
                    <label>Відділ</label>
                    <input value={form.department ?? ''} onChange={e => set('department',
e.target.value)} />
                </div>
                <div className="field">
                    <label>Посада</label>
                    <input value={form.position ?? ''} onChange={e => set('position',
e.target.value)} />
                </div>
            </div>
            <div className="grid2">
                <div className="field">
                    <label>Статус</label>
                    <select value={form.status ?? 'active'} onChange={e => set('status',
e.target.value)}>
                        <option value="active">Активний</option>
                        <option value="inactive">Неактивний</option>
                        <option value="blocked">Заблокований</option>
                    </select>
                </div>
                <div className="field">
                    <label>Дата народження</label>
                    <input type="date" value={form.birthDate ?? ''} onChange={e =>
set('birthDate', e.target.value)} />
                </div>
            </div>
            <div className="field">
                <label>Нотатки</label>
                <textarea rows={3} value={form.notes ?? ''} onChange={e => set('notes',
e.target.value)} />
            </div>
            <div className="error">{error}</div>
            <div className="modal-actions">
                <button type="button" className="btn secondary"
onClick={onCancel}>Скасувати</button>
                <button type="submit" className="btn" disabled={busy}>{busy ? 'Збереження...' :
'Зберегти'}</button>
            </div>
        </form>
    </div>

```

```
);
}
```

B.11 Login.tsx — сторінка авторизації

```
// Login.tsx — сторінка авторизації
import { useState, type FormEvent } from 'react';
import { Link } from 'react-router-dom';
import { useAuth } from '../auth';

export default function Login() {
  const { login } = useAuth();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const onSubmit = async (e: FormEvent) => {
    e.preventDefault();
    setError('');
    try {
      await login(email, password);
    } catch (err: any) {
      setError(err.response?.data?.error ?? 'Помилка входу');
    }
  };

  return (
    <div className="auth-wrap">
      <form className="auth-card" onSubmit={onSubmit}>
        <h1>Вхід до системи</h1>
        <p className="sub">Опрацювання записів користувачів</p>
        <div className="field">
          <label>Електронна пошта</label>
          <input type="email" value={email} onChange={(e) => setEmail(e.target.value)}
placeholder="admin@urs.local" />
        </div>
        <div className="field">
          <label>Пароль</label>
          <input type="password" value={password} onChange={(e) =>
setPassword(e.target.value)} placeholder="....." />
        </div>
        <div className="error">{error}</div>
        <button className="btn block" type="submit">Увійти</button>
        <p style={{ textAlign: 'center', marginTop: 16, fontSize: 14 }}>
Немає облікового запису? <Link to="/register">Зареєструватися</Link>
        </p>
      </form>
    </div>
  );
}
```

B.12 Register.tsx — сторінка реєстрації

```
// Register.tsx — сторінка реєстрації нового облікового запису
import { useState, type FormEvent } from 'react';
import { Link } from 'react-router-dom';
import { useAuth } from '../auth';
```

```

export default function Register() {
  const { register } = useAuth();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [repeat, setRepeat] = useState('');
  const [error, setError] = useState('');

  const onSubmit = async (e: FormEvent) => {
    e.preventDefault();
    setError('');
    if (password !== repeat) { setError('Паролі не збігаються'); return; }
    if (password.length < 6) { setError('Пароль має містити щонайменше 6 символів');
return; }
    try {
      await register(email, password);
    } catch (err: any) {
      setError(err.response?.data?.error ?? 'Помилка реєстрації');
    }
  };

  return (
    <div className="auth-wrap">
      <form className="auth-card" onSubmit={onSubmit}>
        <h1>Реєстрація</h1>
        <p className="sub">Створення нового облікового запису</p>
        <div className="field">
          <label>Електронна пошта</label>
          <input type="email" value={email} onChange={(e) => setEmail(e.target.value)} />
        </div>
        <div className="field">
          <label>Пароль</label>
          <input type="password" value={password} onChange={(e) =>
setPassword(e.target.value)} />
        </div>
        <div className="field">
          <label>Повторіть пароль</label>
          <input type="password" value={repeat} onChange={(e) =>
setRepeat(e.target.value)} />
        </div>
        <div className="error">{error}</div>
        <button className="btn block" type="submit">Зареєструватися</button>
        <p style={{ textAlign: 'center', marginTop: 16, fontSize: 14 }}>
          Уже маєте акаунт? <Link to="/login">Увійти</Link>
        </p>
      </form>
    </div>
  );
}

```

B.13 Dashboard.tsx — головна сторінка зі списком записів

```

// Dashboard.tsx - головна сторінка: статистика + таблиця записів із пошуком,
// фільтрами, сортуванням, пагінацією, створенням/редагуванням/видаленням, експортом
import { useCallback, useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { RecordsApi, type RecordsPage, type Stats, type UserRecord, type Query } from
'../api';
import { useAuth } from '../auth';
import RecordForm from '../components/RecordForm';

```

```

const STATUS_LABEL: Record<string, string> = { active: 'Активний', inactive:
'Неактивний', blocked: 'Заблокований' };

export default function Dashboard() {
  const { user } = useAuth();
  const navigate = useNavigate();
  const [page, setPage] = useState<RecordsPage | null>(null);
  const [stats, setStats] = useState<Stats | null>(null);
  const [query, setQuery] = useState<Query>({ q: '', status: '', department: '', sort:
'createdAt', dir: 'desc', page: 1, pageSize: 10 });
  const [editing, setEditing] = useState<UserRecord | null>(null);
  const [creating, setCreating] = useState(false);

  const reload = useCallback(async () => {
    setPage(await RecordsApi.list(query));
    setStats(await RecordsApi.stats());
  }, [query]);

  useEffect(() => { void reload(); }, [reload]);

  const setQ = (patch: Partial<Query>) => setQuery((q) => ({ ...q, ...patch, page:
patch.page ?? 1 }));
  const toggleSort = (col: string) =>
    setQuery((q) => ({ ...q, sort: col, dir: q.sort === col && q.dir === 'asc' ? 'desc'
: 'asc', page: 1 }));

  const onDelete = async (r: UserRecord) => {
    if (!confirm(`Видалити запис «${r.fullName}»?`)) return;
    await RecordsApi.remove(r.id);
    await reload();
  };

  const onSave = async (data: Partial<UserRecord>) => {
    if (editing) await RecordsApi.update(editing.id, data);
    else await RecordsApi.create(data);
    setEditing(null); setCreating(false);
    await reload();
  };

  const departments = stats?.byDepartment.map((d) => d.department).filter(Boolean) ??
[];

  return (
    <>
      <div className="stats">
        <div className="stat-card"><div className="num">{stats?.total ?? '-'}</div><div
className="label">Усього записів</div></div>
        <div className="stat-card"><div className="num">{stats?.active ?? '-'}</div><div
className="label">Активні</div></div>
        <div className="stat-card"><div className="num">{stats?.inactive ?? '-'}</div><div
className="label">Неактивні</div></div>
        <div className="stat-card"><div className="num">{stats?.blocked ?? '-'}</div><div
className="label">Заблоковані</div></div>
      </div>

      <div className="toolbar">
        <input className="grow" placeholder="Пошук за ПІБ, email або телефоном..."
          value={query.q} onChange={(e) => setQ({ q: e.target.value })} />
        <select value={query.status} onChange={(e) => setQ({ status: e.target.value
    })}>
          <option value="">Усі статуси</option>
          <option value="active">Активні</option>
          <option value="inactive">Неактивні</option>
          <option value="blocked">Заблоковані</option>
        </select>
      </div>
    </>
  );
}

```

```

    <select value={query.department} onChange={(e) => setQ({ department:
e.target.value })}>
      <option value="">Усі відділи</option>
      {departments.map((d) => <option key={d} value={d}>{d}</option>)}
    </select>
    <button className="btn" onClick={() => setCreating(true)}>+ Додати
запис</button>
    <a className="btn secondary" href={RecordsApi.exportUrl}>Експорт CSV</a>
  </div>

  <div className="table-wrap">
    <table>
      <thead>
        <tr>
          <th onClick={() => toggleSort('fullName')}>ПІБ</th>
          <th onClick={() => toggleSort('email')}>Email</th>
          <th>Телефон</th>
          <th onClick={() => toggleSort('department')}>Відділ</th>
          <th onClick={() => toggleSort('status')}>Статус</th>
          <th style={{ width: 180 }}>Дії</th>
        </tr>
      </thead>
      <tbody>
        {page?.data.map((r) => (
          <tr key={r.id}>
            <td><a className="link" onClick={() =>
navigate(`/records/${r.id}`)}>{r.fullName}</a></td>
            <td>{r.email}</td>
            <td>{r.phone}</td>
            <td>{r.department}</td>
            <td><span className={`badge
${r.status}`}>{STATUS_LABEL[r.status]}</span></td>
            <td>
              <div className="row-actions">
                <button className="btn small secondary" onClick={() =>
setEditing(r)}>Змінити</button>
                {user?.role === 'admin' && (
                  <button className="btn small danger" onClick={() =>
onDelete(r)}>Видалити</button>
                )}
              </div>
            </td>
          </tr>
        ))}
        {page && page.data.length === 0 && (
          <tr><td colspan={6} style={{ textAlign: 'center', color: 'var(--muted)'
}}>Записів не знайдено</td></tr>
        )}
      </tbody>
    </table>
  </div>

  <div className="pagination">
    <span style={{ color: 'var(--muted)', fontSize: 13 }}>
      Сторінка {page?.page ?? 1} з {page?.totalPages ?? 1} (усього {page?.total ??
0})
    </span>
    <button className="btn small secondary" disabled={page?.page ?? 1} <= 1}
      onClick={() => setQuery((q) => ({ ...q, page: (q.page ?? 1) - 1 })))>←
Назад</button>
    <button className="btn small secondary" disabled={page?.page ?? 1} >=
      (page?.totalPages ?? 1)}
      onClick={() => setQuery((q) => ({ ...q, page: (q.page ?? 1) + 1
}))}>Далі →</button>
  </div>

  {(creating || editing) && (
    <RecordForm initial={editing ?? undefined} onCancel={() => {
setCreating(false); setEditing(null); }} onSave={onSave} />
  )}

```

```

    })
  </>
);
}

```

B.14 RecordDetails.tsx — деталі запису

```

// RecordDetails.tsx — перегляд детальної інформації про окремий запис
import { useEffect, useState } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import { RecordsApi, type UserRecord } from '../api';

const STATUS_LABEL: Record<string, string> = { active: 'Активний', inactive:
'Неактивний', blocked: 'Заблокований' };

export default function RecordDetails() {
  const { id } = useParams();
  const navigate = useNavigate();
  const [record, setRecord] = useState<UserRecord | null>(null);

  useEffect(() => { RecordsApi.get(Number(id)).then(setRecord).catch(() =>
navigate('/')); }, [id, navigate]);

  if (!record) return <div className="center">Завантаження...</div>;

  const rows: [string, string][] = [
    ['ПІБ', record.fullName],
    ['Email', record.email],
    ['Телефон', record.phone],
    ['Відділ', record.department],
    ['Посада', record.position],
    ['Статус', STATUS_LABEL[record.status]],
    ['Дата народження', record.birthDate],
    ['Нотатки', record.notes],
    ['Створено', new Date(record.createdAt).toLocaleString('uk-UA')],
    ['Оновлено', new Date(record.updatedAt).toLocaleString('uk-UA')],
  ];

  return (
    <div className="table-wrap" style={{ padding: 24 }}>
      <button className="btn secondary small" onClick={() => navigate('/')}>← До
списку</button>
      <h2 style={{ color: 'var(--primary-dark)' }}>{record.fullName}</h2>
      <table>
        <tbody>
          {rows.map(([k, v]) => (
            <tr key={k}><th style={{ width: 220 }}>{k}</th><td>{v || '-'}</td></tr>
          ))}
        </tbody>
      </table>
    </div>
  );
}

```

B.15 Profile.tsx — профіль користувача

```
// Profile.tsx — профіль поточного облікового запису, експорт даних, вихід
import { useNavigate } from 'react-router-dom';
import { RecordsApi } from '../api';
import { useAuth } from '../auth';

export default function Profile() {
  const { user, logout } = useAuth();
  const navigate = useNavigate();

  return (
    <div className="table-wrap" style={{ padding: 24, maxWidth: 560 }}>
      <h2 style={{ color: 'var(--primary-dark)' }}>Профіль користувача</h2>
      <table>
        <tbody>
          <tr><th style={{ width: 200 }}>Email</th><td>{user?.email}</td></tr>
          <tr><th>Роль</th><td>{user?.role === 'admin' ? 'Адміністратор' :
'Користувач'}</td></tr>
          <tr><th>Ідентифікатор</th><td>{user?.id}</td></tr>
        </tbody>
      </table>
      <div style={{ display: 'flex', gap: 12, marginTop: 20 }}>
        <a className="btn secondary" href={RecordsApi.exportUrl}>Експортувати записи
(CSV)</a>
        <button className="btn danger" onClick={() => { logout(); navigate('/login');
}}>Вийти із системи</button>
      </div>
    </div>
  );
}
```

ДОДАТОК Г
Слайди презентації

Створення програмної системи опрацювання записів користувачів

Дипломна кваліфікаційна робота бакалавра
Спеціальність 122 «Комп'ютерні науки»
Кафедра програмних засобів
Національний університет «Запорізька політехніка»
Виконав: Микита КОМІСАРЕНКО
Керівник: Михайло ПОЛЯКОВ
Запоріжжя — 2026

Рисунок Г.1 – Слайд 1

Актуальність і проблема

- Ручний облік записів у таблицях — повільно й незручно
- Помилки введення, втрата та неактуальність даних
- Дублювання записів, відсутність цілісності
- Немає розмежування доступу за ролями
- Складно шукати, фільтрувати та експортувати дані
- Потрібна централізована веб-система обліку



Рисунок Г.2 – Слайд 2

Мета і завдання роботи

- Мета — створити веб-систему опрацювання записів користувачів
- Проаналізувати існуючі рішення та обрати технології
- Спроектувати архітектуру SPA, REST API та модель даних
- Реалізувати авторизацію (JWT) і розмежування ролей (RBAC)
- Реалізувати CRUD, пошук, фільтрацію, пагінацію, експорт CSV
- Розробити адаптивний інтерфейс та інформаційний дашборд

Рисунок Г.3 – Слайд 3

Об'єкт і предмет дослідження

- Об'єкт — процес опрацювання записів користувачів в організації
- Предмет — методи та засоби побудови веб-системи обліку записів
- Засоби: SPA на React+TypeScript, REST API на Node.js/Express
- Зберігання даних — реляційна БД SQLite
- Безпека — автентифікація JWT та контроль доступу за ролями

Рисунок Г.4 – Слайд 4

Аналіз існуючих рішень

- **Microsoft Entra ID** — хмарний, платний, надлишковий для задачі
- **Keycloak** — потужний, але складний у налаштуванні та розгортанні
- **Directus** — універсальний, важкий, надмірний функціонал
- **phpMyAdmin** — лише адміністрування БД, не бізнес-облік
- **Спільні недоліки:** складність, вартість, відсутність гнучкого CRUD
- **Висновок** — потрібне власне легке спеціалізоване рішення

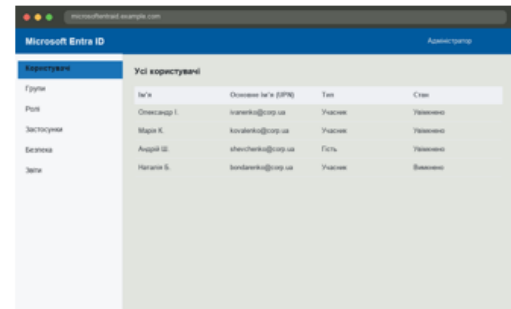


Рисунок Г.5 – Слайд 5

Обрані технології

- **TypeScript** — типізація на фронтенді та бекенді
- **React (Vite, React Router, Axios)** — клієнтський інтерфейс SPA
- **Node.js + Express** — серверна частина та REST API
- **SQLite (better-sqlite3)** — файлова реляційна БД без окремого сервера
- **REST + JSON** — обмін даними клієнт-сервер
- **JWT + bcrypt** — автентифікація та хешування паролів



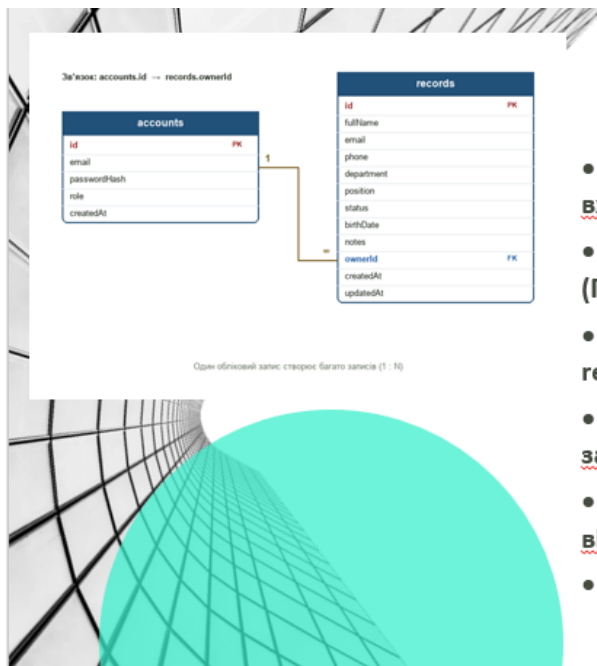
Рисунок Г.6 – Слайд 6

Архітектура системи

- Тришарова архітектура: клієнт ↔ REST API ↔ БД
- Клієнт — SPA на React+TypeScript (Vite)
- Сервер — Node.js/Express, REST-ендпоінти, JWT-автентифікація
- Middleware авторизації за ролями (admin/user)
- База даних — SQLite, доступ через better-sqlite3
- Обмін — JSON через HTTP, токен у заголовку Authorization



Рисунок Г.7 – Слайд 7



Модель даних

- Таблиця accounts — облікові записи для входу (email, passwordHash, role)
- Таблиця records — записи користувачів (ПІБ, email, телефон, відділ, посада, статус)
- Зв'язок один-до-багатьох: account 1—N records
- Зовнішній ключ ownerId — хто створив запис
- Поля createdAt/updatedAt для відстеження змін
- Унікальність email, індекси для пошуку

Рисунок Г.8 – Слайд 8

Основний алгоритм роботи

- Старт — ініціалізація застосунку, перевірка JWT-токена
- Немає токена — екран входу/реєстрації
- Успішний вхід — перехід на дашборд
- Завантаження записів, пошук, фільтрація, сортування, пагінація
- Дії CRUD: додати, редагувати, видалити (admin) з валідацією
- Експорт вибірки у CSV; вихід — видалення токена

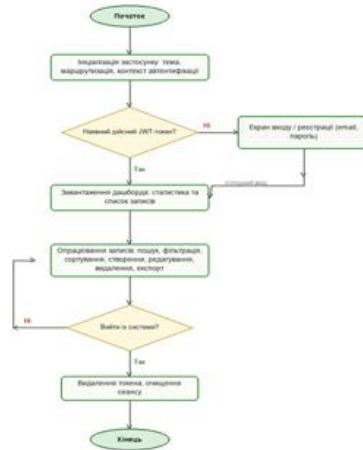


Рисунок Г.9 – Слайд 9

Інтерфейс системи

- Екран входу — авторизація за email і паролем
- Дашборд — картки статистики та таблиця записів
- Форма запису — створення/редагування з валідацією
- Деталі запису — повна інформація про користувача
- Адаптивний дизайн, повідомлення про помилки
- Скріншоти основних екранів системи

Система управління записами користувачів

Записи Профіль Вихід

12 Усього записів 9 Активні 2 Неактивні 1 Зайнятий

Пошук за ID, email або телефонієм: Усі статуси Усі мови + Додати запис Експорт CSV

ID	Ім'я	Email	Телефон	Роль	Статус	Дії
1	Іванов Олександр Петрович	ivanov@corp.ua	+380501112233	Бухгалтер	Активний	✓ ✖
2	Коваленко Марія Сергіївна	kovalevko@corp.ua	+380672223344	Кадр	Активний	✓ ✖
3	Сидоренко Андрій Олександр	sidorenko@corp.ua	+380633334455	ІТ	Активний	✓ ✖
4	Євдокимов Петро Іванович	evdokimov@corp.ua	+380554445566	Маркетинг	Неактивний	✓ ✖
5	Толкачев Віктор Михайлович	tolkachev@corp.ua	+380675556677	ІТ	Активний	✓ ✖
6	Кравченко Дмитро Юрійович	kravchenko@corp.ua	+380507778899	Продажі	Зайнятий	✓ ✖
7	Мельник Ольга Вікторівна	melnyk@corp.ua	+380636667788	Бухгалтер	Активний	✓ ✖

Рисунок Г.10 – Слайд 10

Висновки



- Реалізовано веб-систему опрацювання записів користувачів
- Виконано авторизацію JWT та розмежування доступу за ролями
- Реалізовано CRUD, пошук, фільтрацію, пагінацію, експорт CSV, дашборд
- Мету роботи досягнуто, усі завдання виконано
- Напрями розвитку: ролі-додатки, аудит дій, інтеграції, хмарне розгортання
- Перехід на повноцінну СУБД при зростанні навантаження

Рисунок Г.11 – Слайд 11