

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 проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБЛІКУ
СОЦІАЛЬНО ВРАЗЛИВИХ ГРУП НАСЕЛЕННЯ

DEVELOPMENT OF SOFTWARE FOR THE AUTOMATED ACCOUNTING
OF SOCIALLY VULNERABLE GROUPS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-
122

Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ТОЛОЧИН А.Г.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(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 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ТОЛОЧИНА Андрія Георгійовича
 (ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розроблення програмного забезпечення для обліку соціально вразливих груп населення. Development of Software for the Automated Accounting of Socially Vulnerable Groups

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

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

2. Строк подання студентом проєкту (роботи) 10 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 проблеми та постановка завдань дослідження. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

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 архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Андрій ТОЛОЧИН
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ФЕДОРОНЧАК
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 83 с., 7 табл., 19 рис., 2 дод., 20 джерел.

СОЦІАЛЬНА СЛУЖБА, ВЕБЗАСТОСУНОК, ОБЛІК НАСЕЛЕННЯ, ПІЛЬГОВІ КАТЕГОРІЇ, ВРАЗЛИВІ ГРУПИ НАСЕЛЕННЯ, БАЗА ДАНИХ, REACT, SUPABASE.

Об'єкт дослідження – процес розроблення програмного забезпечення для обліку осіб пільгових та соціально вразливих категорій у соціальних службах.

Предмет дослідження – методи та програмні засоби розроблення вебзастосунків для автоматизації роботи соціальних служб.

Мета роботи – підвищення ефективності роботи соціальних працівників та автоматизація процесів обліку соціально вразливих груп населення шляхом розроблення спеціалізованого програмного забезпечення у вигляді вебзастосунку.

Матеріали, методи та технічні засоби: використано методи системного аналізу, об'єктно-орієнтованого та компонентного програмування, а також проєктування реляційних баз даних. Клієнтську частину (Frontend) реалізовано на базі бібліотеки React із використанням строго типізованої мови програмування TypeScript. Серверну частину (Backend), автентифікацію та базу даних побудовано з використанням хмарної платформи Supabase та системи керування базами даних PostgreSQL. Розроблення проводилося в інтегрованому середовищі Visual Studio Code із застосуванням системи контролю версій Git.

Результати. Розроблено повнофункціональне програмне забезпечення у вигляді вебзастосунку для Департаменту соціального захисту населення. Реалізовано електронний кабінет для працівників із системою безпечної автентифікації, зручну електронну таблицю-реєстр для ведення обліку

бенефіціарів із можливістю гнучкої фільтрації (за районами та пільговими категоріями) і пошуку за ПІБ. Створено функціонал додавання нових осіб, редагування їхніх персональних даних, а також систему завантаження та зберігання електронних копій підтверджуючих документів.

Висновки. Досягнуто мети роботи: створено інструмент, який автоматизує рутинні процеси соціальної служби та переводить облік пільговиків з паперового формату в електронний. Програма забезпечує цілісність даних та розмежовує права доступу користувачів. Застосунок має інтуїтивно зрозумілий інтерфейс, що не потребує тривалого навчання персоналу.

Галузь використання – установи соціального захисту населення, центри надання соціальних та адміністративних послуг.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 83 pages, 7 tables, 19 figures, 2 appendixes, 20 sources.

SOCIAL SERVICE, WEB APPLICATION, POPULATION ACCOUNTING, PRIVILEGED CATEGORIES, VULNERABLE POPULATION GROUPS, DATABASE, REACT, SUPABASE.

Object of study is the process of developing software for tracking individuals in disadvantaged and socially vulnerable groups within social services.

Subject of study is the methods and software tools for developing web applications to automate the work of social services.

The purpose of the work is to increase the efficiency of social workers and automate the accounting processes of socially vulnerable population groups by developing specialized software in the form of a web application.

Materials, methods, and tools: methods of system analysis, object-oriented and component programming, as well as relational database design were used. The client side (Frontend) is implemented based on the React library using the strictly typed programming language TypeScript. The server side (Backend), authentication, and database are built using the Supabase cloud platform and the PostgreSQL database management system. The development was carried out in the Visual Studio Code integrated development environment using the Git version control system.

Results. A fully functional software in the form of a web application for the Department of Social Protection of the Population has been developed. An electronic cabinet for employees with a secure authentication system, a convenient electronic spreadsheet-registry for keeping track of beneficiaries with the possibility of flexible filtering (by districts and privileged categories), and search by full name has been implemented. The functionality of adding new individuals, editing their personal data, as well as a system for uploading and storing electronic copies of supporting documents has been created.

Conclusions. The purpose of the work has been achieved: a tool has been created that automates the routine processes of the social service and transfers the accounting of beneficiaries from a paper to an electronic format. The program ensures data integrity and differentiates user access rights. The application has an intuitive interface that does not require lengthy staff training.

The field of use is social protection institutions, centers for the provision of social and administrative services.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	10
Вступ.....	11
1 Аналіз предметної області.....	14
1.1 Опис процесів обліку в установах соціального захисту	14
1.2 Огляд існуючих засобів автоматизації та їх порівняльний аналіз	16
1.3 Обґрунтування функціональних та нефункціональних вимог	19
1.4 Постановка завдання на розроблення програмного забезпечення	21
1.5 Висновки за розділом 1	22
2 Проєктування архітектури та вибір технологічного стеку	24
2.1 Архітектурна побудова вебзастосунку	24
2.2 Вибір та обґрунтування інструментарію розроблення.....	26
2.3 Обґрунтування вибору хмарної платформи Supabase.....	29
2.4 Вимоги до технічних і програмних засобів.....	31
2.5 Висновки за розділом 2	34
3 Реалізація програмного забезпечення	36
3.1 Проєктування та розроблення реляційної бази даних	36
3.2 Реалізація клієнтської частини на базі бібліотеки React.....	39
3.3 Розроблення модулів фільтрації та управління документами.....	41
3.4 Опис взаємодії з API хмарного сервісу	44
3.5 Висновки за розділом 3	45
4 Тестування та експериментальне дослідження програми	47
4.1 Методика проведення функціонального тестування.....	47
4.2 Аналіз результатів тестування продуктивності та безпеки	49
4.3 Інструкція користувача та рекомендації з експлуатації.....	52
4.4 Висновки за розділом 4	54

Висновки	55
Перелік джерел посилання	57
Додаток А Текст програми.....	59
Додаток Б Слайди презентації	75

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД	– база даних;
ВПО	– внутрішньо переміщена особа;
ПІН	– ідентифікаційний податковий номер;
ПІБ	– прізвище, ім'я, по батькові;
СКБД	– система керування базами даних;
API	– Application Programming Interface;
BaaS	– Backend as a Service;
CRUD	– Create, Read, Update, Delete;
DOM	– Document Object Model;
HTTP	– HyperText Transfer Protocol;
HTTPS	– HyperText Transfer Protocol Secure;
JSON	– JavaScript Object Notation;
JWT	– JSON Web Token;
MVC	– Model-View-Controller;
ORM	– Object-Relational Mapping;
RLS	– Row Level Security;
SPA	– Single Page Application;
SQL	– Structured Query Language;
UI	– User Interface;
UX	– User Experience;
URL	– Uniform Resource Locator;
UUID	– Universally Unique Identifier;
WCAG	– Web Content Accessibility Guidelines.

ВСТУП

Стрімка еволюція цифрових технологій диктує нові вимоги до засобів управління державним та соціальним секторами, де ключовим фактором успіху стає швидкість та надійність обробки даних. У сучасних умовах, особливо на тлі збільшення кількості внутрішньо переміщених осіб та інших пільгових категорій, особливої ваги набуває прозорість робочих процесів соціальних служб: від оперативного створення карток бенефіціарів до швидкого пошуку необхідних підтверджуючих документів. Попри велику кількість доступних офісних інструментів, чимало установ досі відчують брак гнучких спеціалізованих рішень, які б забезпечували миттєвий доступ до інформації без перевантаження інтерфейсу зайвими функціями. Актуальність розробки зумовлена необхідністю створення єдиного електронного середовища, де облік, візуалізація та зберігання даних соціально вразливих груп населення відбуваються централізовано та безпечно для всіх працівників установи.

Глобальні тенденції в державному управлінні (e-Government) підтверджують, що цифровізація є фундаментом продуктивності. Аналітичні звіти та практика показують: розрізненість даних, ведення паперових журналів та використання неспеціалізованих табличних процесорів часто стають головними чинниками сповільнення обробки звернень громадян. Хоча автоматизація баз даних не є новою концепцією, соціальна сфера потребує інструментів, що поєднують високу надійність збереження персональних даних із простотою впровадження. Існуючі громіздкі системи часто виявляються занадто складними для швидкого старту у відділах соціального захисту, створюючи додатковий адміністративний тиск на персонал.

Проблема дослідження полягає в необхідності створення легкого та швидкодіючого інструменту, який забезпечує зручне ведення реєстру бенефіціарів, миттєву фільтрацію за районами та категоріями, а також надійне збереження електронних копій документів. Огляд ринку показує, що хоча

питання управління завданнями вирішуються в таких загальних системах, як Jira або Trello, вони мають надлишкову складність, не призначені для специфіки обліку громадян та вимагають адаптації [1], [2]. Провідні дослідження в галузі веброзробки вказують на перспективність використання архітектури на основі хмарних платформ (BaaS) та типізованих мов програмування, як-от TypeScript, для створення надійних систем обробки чутливих даних [3].

Об'єктом дослідження виступають процеси розроблення програмного забезпечення для обліку осіб пільгових та соціально вразливих категорій у соціальних службах.

Предметом дослідження є методи та програмні засоби розроблення вебзастосунків для автоматизації роботи соціальних служб.

Мета роботи полягає у підвищенні ефективності роботи соціальних працівників та автоматизації процесів обліку соціально вразливих груп населення шляхом розроблення спеціалізованого програмного забезпечення у вигляді вебзастосунку.

Для досягнення цієї мети було окреслено та виконано такі етапи дослідження:

- системний аналіз предметної області та процесів соціального захисту для визначення оптимального набору функцій майбутнього застосунку;
- проєктування архітектури системи на базі хмарної інфраструктури для реалізації надійного зберігання персональних даних та файлів;
- розробка та впровадження реляційної бази даних, модулів управління реєстром бенефіціарів зі складними зв'язками, системи фільтрації та електронного кабінету;
- проведення комплексного тестування системи на предмет швидкодії, безпеки та коректності обробки даних при роботі з електронними документами.

Методологічну базу дослідження складають принципи об'єктно-орієнтованого та компонентного проєктування, теорії реляційних баз даних та сучасні стандарти веброзробки. Технічна реалізація базується на використанні бібліотеки React для фронтенд-частини та строго типізованої мови TypeScript [4]. Роботу з даними, автентифікацію та збереження файлів організовано за допомогою хмарної платформи Supabase, що використовує базу PostgreSQL [5], [6].

Розроблений інструментарій має високу практичну цінність для департаментів соціального захисту населення та центрів надання послуг, що прагнуть оптимізувати обробку справ та мінімізувати паперовий документообіг. Результати впровадження свідчать про значне скорочення часу на пошук інформації та підвищення прозорості ведення обліку вразливих груп населення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис процесів обліку в установах соціального захисту

Сучасна система соціального захисту в Україні перебуває на етапі інтенсивної трансформації, що зумовлено як загальнодержавною політикою цифровізації (програма «Держава у смартфоні»), так і критичною необхідністю адаптації до умов воєнного стану. Установи соціального захисту, зокрема профільні департаменти міських рад, виконують роль ключової ланки у забезпеченні підтримки найбільш вразливих верств населення: внутрішньо переміщених осіб (ВПО), пенсіонерів, осіб з інвалідністю, багатодітних родин та малозабезпечених громадян.

Процес обліку бенефіціарів у типовій установі соціального захисту є багаторівневою процедурою, етапи якої представлено далі.

Етап 1. Первинне звернення та ідентифікація. Громадянин надає пакет документів, що підтверджують його особу та право на отримання пільг або допомоги (паспорт, ППН, спеціальні посвідчення).

Етап 2. Перевірка та категоризація. Працівник служби аналізує надані дані та відносить особу до однієї або декількох категорій. Важливою особливістю є те, що одна особа може мати кілька статусів одночасно (наприклад, особа з інвалідністю, яка є внутрішньо переміщеною особою), що вимагає гнучкої моделі обліку.

Етап 3. Формування особової справи. Створення паперової та, за наявності відповідних засобів, електронної картки, де накопичується історія звернень та наданих послуг.

Етап 4. Зберігання документів. Кожна справа повинна містити цифрові або паперові копії документів, доступ до яких має бути оперативним для перевірки актуальності статусів.

Аналіз діяльності Департаменту соціального захисту населення Запорізької міської ради показав, що попри часткове використання загальнодержавних реєстрів, значна частина внутрішнього оперативного

обліку в межах районів міста все ще залишається недостатньо автоматизованою. Часто дані дублюються в розрізнені таблиці (наприклад, Microsoft Excel), що призводить до децентралізації інформації.

Основними проблемами існуючих процесів обліку є:

- фрагментарність даних. Відсутність єдиної бази даних, яка б об'єднувала інформацію за районами міста та дозволяла здійснювати наскрізний пошук;
- складність фільтрації. Ручна обробка великих масивів даних для формування звітів (наприклад, «кількість ВПО в окремому районі за певною пільгою») займає значний обсяг робочого часу працівників;
- ризики втрати інформації. Збереження копій документів у паперовому вигляді або на локальних комп'ютерах працівників підвищує ймовірність їх фізичного знищення або втрати доступу до них;
- бюрократичне навантаження. Велика кількість рутинних операцій з ручного введення даних відволікає спеціалістів від безпосередньої роботи з громадянами.

Таким чином, предметна область характеризується великим обсягом чутливих персональних даних, необхідністю складної фільтрації за декількома ознаками та потребою у централізованому хмарному сховищі для документів. Автоматизація цих процесів шляхом розроблення спеціалізованого вебзастосунку дозволить створити єдине інформаційне поле, підвищити швидкість обслуговування населення та забезпечити надійність зберігання даних.

Для кількісного обґрунтування доцільності розроблення програмного забезпечення було проведено аналіз витрат робочого часу персоналу Департаменту соціального захисту населення Запорізької міської ради. За існуючої децентралізованої моделі ручна обробка однієї справи бенефіціара (ідентифікація, ксерокопіювання документів, внесення даних у локальні табличні процесори) займає в середньому 15 хвилин ($T_{\text{manual}} = 15$ хв). За умови щоденного обслуговування 40 громадян одним спеціалістом, витрати

часу становлять 10 годин на день, що перевищує стандартний робочий день і призводить до накопичення черг.

Впровадження автоматизованого вебзастосунку з функцією миттєвого пошуку, шаблонами категорій та хмарним завантаженням скан-копій дозволить скоротити час обробки однієї справи до 3–4 хвилин ($T_{\text{auto}} = 4$ хв). Прогнозована економія робочого часу (E) розраховується за формулою:

$$E = \frac{T_{\text{manual}} - T_{\text{auto}}}{T_{\text{manual}}} \times 100\%$$

Відповідно, впровадження системи дозволить зекономити понад 73% робочого часу спеціаліста, який може бути перенаправлений на надання безпосередньої соціальної допомоги та консультацій, що суттєво підвищує загальну пропускну здатність установи

1.2 Огляд існуючих засобів автоматизації та їх порівняльний аналіз

На сьогоднішній день на ринку програмного забезпечення існує велика кількість універсальних систем управління завданнями та проєктами. Через відсутність спеціалізованих локальних рішень для обліку соціально вразливих груп населення, багато установ соціального захисту намагаються адаптувати під свої потреби існуючі системи канбан-дошок та таск-трекери (наприклад, Trello, Jira, Linear), де картка «завдання» використовується як картка «бенефіціара». Для обґрунтування необхідності розроблення нового вебзастосунку було проведено аналіз цих програмних рішень.

1.2.1 Застосунок Trello

Trello — це популярна хмарна програма для управління проєктами, що базується на парадигмі канбан-дошок [2]. Інтерфейс системи дозволяє створювати колонки (наприклад, «Нові звернення», «В обробці», «Затверджено») та переміщувати між ними картки за допомогою механізму drag-and-drop. Перевагою Trello є надзвичайна простота використання та візуальна наочність процесу. Проте для потреб соціальної служби застосунок має суттєві недоліки. Trello не підтримує побудову складних реляційних зв'язків між даними (наприклад, коли одна людина належить до кількох пільгових категорій одночасно). Крім того, система не має вбудованих механізмів гнучкої табличної фільтрації за персональними даними (ПІБ, ПІН, район), що робить пошук у великих базах даних вкрай неефективним.

1.2.2 Застосунок Jira

Jira — це потужна система для відстеження помилок та управління проєктами, розроблена компанією Atlassian [1]. Вона пропонує глибоку кастомізацію робочих процесів (workflows), детальну систему звітності та можливість налаштування складних прав доступу. Незважаючи на високий рівень безпеки та функціональності, використання Jira в установах соціального захисту є недоцільним. По-перше, система орієнтована на процес розробки програмного забезпечення, що робить її інтерфейс перевантаженим і занадто складним для працівників соціальної сфери. По-друге, архітектура Jira є «задаче-орієнтованою» (task-centric), тоді як облік пільговиків вимагає «особо-орієнтованої» (person-centric) архітектури, де головним сутнісним об'єктом є профіль людини з її постійними атрибутами (паспорт, адреса, дата народження), а не життєвий цикл завдання.

1.2.3 Застосунок Linear

Linear — це сучасний інструмент для управління розробкою, який відрізняється високою швидкістю та мінімалістичним дизайном [7]. Система забезпечує миттєву синхронізацію даних та має ефективну систему гарячих клавіш. Як і попередні рішення, Linear оптимізований виключно для інженерних команд. У системі відсутня можливість створення структурованих реєстрів для зберігання специфічних персональних даних громадян. Застосунок не призначений для безпечного зберігання та управління цифровими копіями підтверджуючих документів (довідок ВПО, копій паспортів), що є критичною вимогою для документообігу в державних установах.

1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів

З метою систематизації виявлених недоліків існуючих рішень та виділення переваг системи, що проєктується, було складено порівняльну характеристику. Оцінювання проводилося за критеріями, що є критично важливими для установ соціального захисту населення. Результати порівняння наведено в табл. 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих та розроблюваного програмних продуктів

Критерій порівняння	Trello	Jira	Linear	Розроблюваний застосунок
1	2	3	4	5
Архітектурний підхід	Задаче-орієнтований	Задаче-орієнтований	Задаче-орієнтований	Особо-орієнтований (картка бенефіціара)

Кінець таблиці 1.1

1	2	3	4	5
Підтримка реляційних зв'язків (багато категорій на 1 особу)	Низька	Середня	Низька	Висока (база даних PostgreSQL)
Спеціалізоване сховище для документів бенефіціарів	Відсутнє	Відсутнє	Відсутнє	Наявне (інтеграція з Supabase Storage)
Відповідність Закону України «Про захист персональних даних»	Не відповідає (публічні сервери)	Частково	Не відповідає	Відповідає (ізолювані інстанси, RLS)
Можливість складної фільтрації (за ПІБ, районами)	Відсутня	Обмежена	Обмежена	Наявна (оптимізовані SQL-запити)

Отже, результати порівняльного аналізу свідчать, що використання універсальних таск-трекерів не здатне повною мірою задовольнити потреби соціальних служб. Для забезпечення надійного обліку, швидкого доступу до персональних даних та зручної роботи з документами необхідне розроблення спеціалізованого програмного забезпечення.

1.3 Обґрунтування функціональних та нефункціональних вимог

1.3.1 Визначення функціональних вимог

Формування переліку функціональних вимог є критично важливим етапом, оскільки він визначає межі проєкту та конкретні можливості, які отримає кінцевий користувач — працівник соціальної служби. Насамперед

система повинна забезпечувати повний цикл управління персональними даними громадян. Це передбачає не лише базове створення карток, а й реалізацію механізмів глибокого редагування та валідації введеної інформації, що дозволить уникнути помилок при заповненні таких важливих полів, як ідентифікаційний код чи серія паспорта.

Особлива увага при проєктуванні приділяється реалізації складної моделі зв'язків між об'єктами. Оскільки один бенефіціар може одночасно належати до декількох вразливих груп населення, програмне забезпечення повинно коректно обробляти реляційні зв'язки типу «багато-до-багатьох». Це дозволить персоналу призначати особі статус внутрішньо переміщеної особи одночасно з категорією багатодітної родини чи особи з інвалідністю, що значно розширює можливості точного обліку.

Крім того, функціональна частина системи має включати розвинену підсистему пошуку та фільтрації. В умовах роботи з великими реєстрами, що налічують тисячі записів, звичайний лінійний перегляд стає неможливим. Тому застосунок повинен підтримувати миттєву фільтрацію за районами міста та комбінований пошук за прізвищем, іменем чи по батькові. Важливою функцією є також інтегроване сховище для цифрових документів. Програма має надавати інтерфейс для завантаження сканованих копій документів безпосередньо у профіль бенефіціара, забезпечуючи їх подальший перегляд або завантаження для перевірки актуальності пільгового статусу.

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги встановлюють стандарти якості програмного продукту згідно з міжнародним стандартом ISO/IEC 25010 [8]:

— продуктивність та швидкодія (Performance Efficiency): застосунок має підтримувати одночасну роботу не менше 50 користувачів без деградації продуктивності. Час відгуку сервера на виконання пошукових запитів у базі даних не повинен перевищувати 200 мс. Для оптимізації рендерингу реєстру

обсягом понад 1000 записів обов'язковим є використання алгоритмів серверної пагінації;

— безпека (Security): враховуючи вимоги Закону України «Про захист персональних даних», обмін інформацією між клієнтом та сервером має здійснюватися виключно за зашифрованим протоколом HTTPS із використанням стандарту TLS 1.3. База даних повинна бути захищена від SQL-ін'єкцій на рівні ORM/API, а розмежування прав доступу між працівниками різних районів — забезпечуватися політиками безпеки на рівні рядків (Row Level Security);

— надійність (Reliability) та зберігання: цифрові копії документів мають зберігатися у захищеному хмарному сховищі з регулярним автоматичним резервним копіюванням. Максимальний розмір одного завантажуваного файлу обмежується 5 МБ для оптимізації дискового простору;

— сумісність (Compatibility): застосунок повинен мати кросплатформену клієнтську частину, що коректно функціонує в актуальних версіях веббраузерів (Google Chrome 100+, Mozilla Firefox 90+, Safari 15+) без встановлення додаткових плагінів на робочі станції державних службовців.

1.4 Постановка завдання на розроблення програмного забезпечення

Метою даної дипломної роботи є розроблення спеціалізованого вебзастосунку для автоматизації процесів обліку бенефіціарів у Департаменті соціального захисту населення Запорізької міської ради. Система повинна замінити розрізнені методи зберігання інформації єдиною цифровою платформою, що об'єднує персональні дані, соціальні статуси та підтверджуючі документи в цілісну структуру.

Для реалізації цієї мети необхідно виконати низку послідовних інженерних кроків. По-перше, потрібно провести детальне проєктування

архітектури бази даних, яка б відображала реальні зв'язки між мешканцями міста, адміністративними районами та пільговими категоріями. По-друге, необхідно обрати та обґрунтувати технологічний стек, який дозволить реалізувати клієнтську частину на базі сучасних бібліотек (React) та інтегрувати її з хмарною серверною частиною.

Завдання передбачає створення зручного інтерфейсу для операторів системи, який дозволить не лише вносити нові записи, а й оперативно редагувати існуючі профілі з автоматичною синхронізацією змін у хмарному сховищі. Окремим етапом є реалізація модуля завантаження файлів, який повинен підтримувати роботу з різними форматами зображень та документів. Кінцевим етапом роботи є проведення тестування, яке має підтвердити відповідність розробленого програмного забезпечення всім заявленим функціональним та безпековим вимогам.

1.5 Висновки за розділом 1

У межах першого розділу було проведено всебічне дослідження предметної області та процесів соціального обліку в сучасних реаліях муніципального управління. Аналіз показав, що існуючі підходи до ведення реєстрів у соціальних службах часто страждають від надмірної бюрократизації та децентралізації даних, що суттєво уповільнює надання допомоги вразливим групам населення.

Порівняльне дослідження універсальних систем управління проєктами, таких як Trello та Jira, підтвердило гіпотезу про їхню обмежену ефективність у даній сфері. Було виявлено, що відсутність у цих системах спеціалізованої «особо-орієнтованої» логіки та складність налаштування реляційних зв'язків роблять їх непридатними для повноцінного державного обліку.

На основі отриманих результатів було сформовано та детально описано перелік вимог до майбутньої програми. Формалізація цих вимог дозволила чітко окреслити завдання на розробку, яке зосереджене на створенні

продуктивного, безпечного та інтуїтивно зрозумілого вебзастосунку. Це закладає надійний теоретичний фундамент для переходу до наступного етапу — проєктування архітектури та безпосередньої реалізації програмного продукту.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ

2.1 Архітектурна побудова вебзастосунку

Для забезпечення надійності, масштабованості та зручності супроводу програмного забезпечення було обрано клієнт-серверний підхід до побудови архітектури. Враховуючи сучасні тенденції веброзробки та специфіку цільової аудиторії (працівники соціальної сфери), систему реалізовано у вигляді односторінкового вебзастосунку (Single Page Application, SPA), який взаємодіє з хмарним сервером через прикладний програмний інтерфейс (API) [9].

Архітектура розробленої системи логічно поділяється на два незалежні, але тісно пов'язані рівні: клієнтську частину (Frontend) та серверну інфраструктуру (Backend).

Клієнтська частина відповідає за візуалізацію даних та безпосередню взаємодію з користувачем. Вона генерує користувацький інтерфейс, обробляє події введення (наприклад, заповнення форм бенефіціарів), виконує первинну валідацію даних та ініціює асинхронні мережеві запити. Оскільки це SPA-застосунок, маршрутизація та перемальовування екранів відбуваються на стороні браузера без повного перезавантаження вебсторінки. Це створює ефект роботи у повноцінній десктопній програмі, що позитивно впливає на користувацький досвід та зменшує навантаження на мережу, оскільки між клієнтом та сервером передаються виключно «чисті» дані у форматі JSON, а не готові HTML-документи.

Серверна частина реалізована за сучасною парадигмою Backend-as-a-Service (BaaS), що дозволяє делегувати рутинні завдання з налаштування та підтримки серверів хмарному провайдеру. Серверний рівень виконує низку критично важливих функцій:

Автентифікація та авторизація: перевірка облікових даних користувачів, генерація безпечних токенів доступу та розмежування сесій.

Управління базою даних: виконання операцій читання, запису, оновлення та видалення (CRUD) у реляційній базі даних із дотриманням політик безпеки на рівні окремих записів.

Файлове сховище: надання захищеного простору для завантаження, зберігання та видачі цифрових копій документів громадян.

Процес обміну даними у системі має такий вигляд: клієнтський застосунок формує HTTP-запит (наприклад, на отримання списку осіб певного району), додаючи до нього критерії фільтрації та токен доступу користувача. Запит надсилається до хмарного API. Сервер перевіряє права доступу, звертається до бази даних, формує відповідь у форматі JSON та повертає її клієнту. Після отримання даних клієнтська частина оновлює стан (state) застосунку та миттєво відображає результати у таблиці.

З метою візуалізації логіки роботи системи з головним інформаційним об'єктом — електронною карткою бенефіціара — було спроектовано діаграму станів (State Machine Diagram). Вона відображає життєвий цикл профілю від моменту його створення оператором до переходу в архівний стан у разі втрати пільгового статусу або видалення з системи.

Перехід між станами супроводжується транзакційними запитами до хмарного API та валідацією даних на стороні сервера. Логіку переходів станів наведено на рис. 2.1.

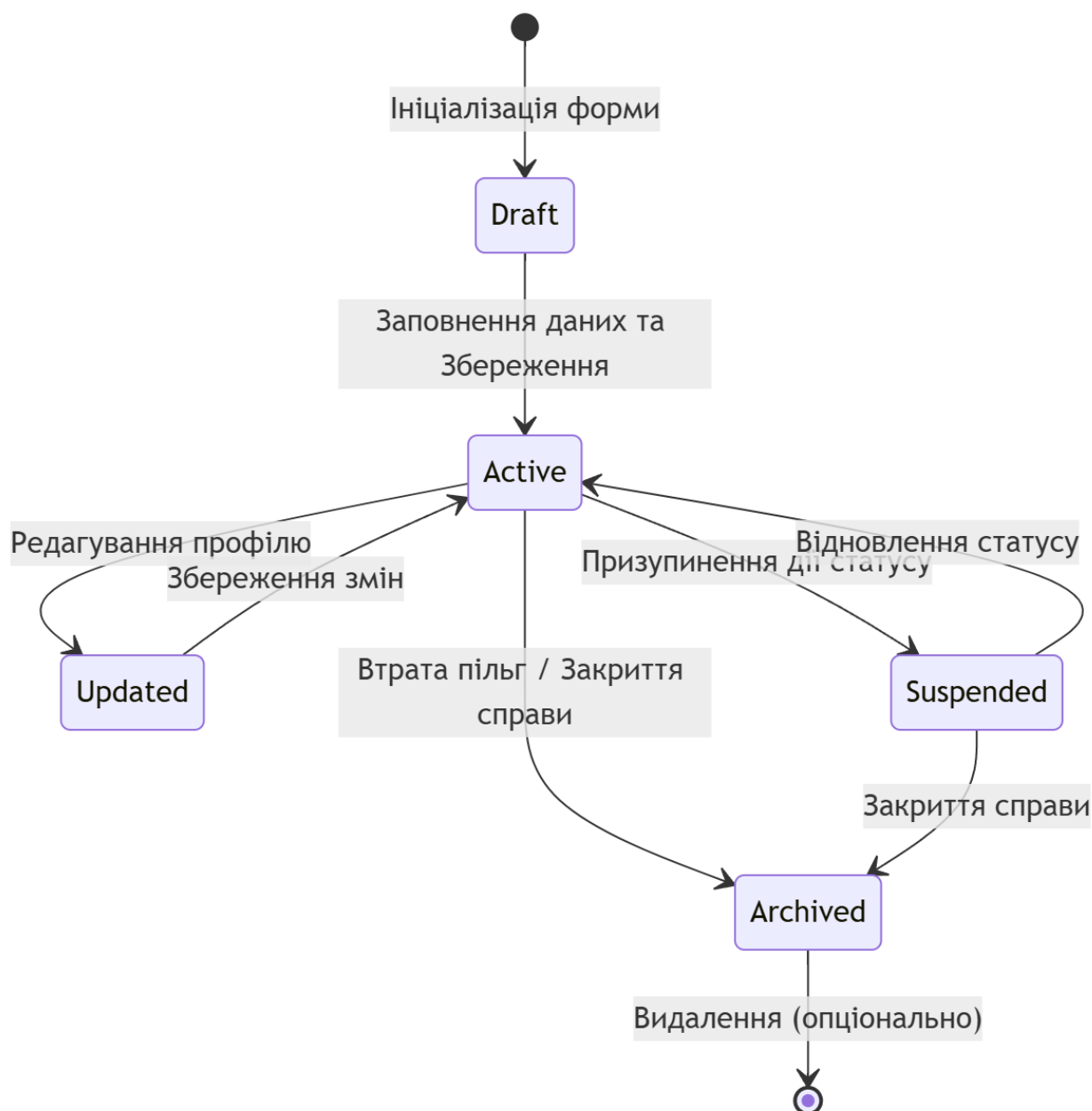


Рисунок 2.1 – UML-діаграма станів електронної картки бенефіціара

2.2 Вибір та обґрунтування інструментарію розроблення

Якість, безпека та легкість подальшого масштабування програмного забезпечення значною мірою залежать від правильно підбраного стеку технологій на етапі проєктування. Для реалізації клієнтської частини застосунку було обрано низку сучасних інструментів, що відповідають

галузевим стандартам розроблення (інструментарій серверної частини буде детально розглянуто в окремому підрозділі).

Мова програмування. Основним інструментом розроблення було обрано мову TypeScript [3]. Це суворо типізована надбудова над JavaScript, яка дозволяє виявляти більшість синтаксичних та логічних помилок ще на етапі написання коду (компіляції), а не під час його виконання. Враховуючи складність предметної області — наявність об'ємних моделей даних (бенефіціари з десятками полів, масиви категорій, вкладені об'єкти районів та документів) — використання TypeScript є критичною необхідністю. Типізація забезпечує автодоповнення коду в середовищі розробки, полегшує рефакторинг та робить архітектуру застосунку більш прозорою і стійкою до збоїв.

Бібліотека для побудови інтерфейсу. Як основу для побудови користувацького інтерфейсу обрано бібліотеку React [4]. Її головна перевага полягає у компонентному підході: весь інтерфейс розбивається на дрібні незалежні частини (кнопки, таблиці, форми введення), які можна багаторазово перевикористовувати в різних частинах програми. Крім того, React використовує алгоритм віртуального дерева (Virtual DOM). Це означає, що при зміні даних (наприклад, при застосуванні нового фільтра до таблиці) бібліотека обчислює мінімально необхідну кількість змін і оновлює лише ті елементи на екрані, які дійсно змінилися. Це забезпечує надзвичайно високу продуктивність інтерфейсу під час роботи з великими реєстрами бенефіціарів.

Бібліотека UI-компонентів. Для забезпечення стандартизованого, сучасного та доступного зовнішнього вигляду було використано систему компонентів Material-UI (MUI) [10]. Використання готових, ретельно протестованих компонентів (таблиць даних з підтримкою пагінації, модальних вікон, селектів, полів введення дат) дозволило значно прискорити процес розроблення та сфокусуватися на реалізації бізнес-логіки соціальної служби, замість рутинного написання базових CSS-стилів. Інтерфейси, побудовані за

принципами Material Design, є інтуїтивно зрозумілими для державних службовців, оскільки наслідують загальноприйняті стандарти ергономіки.

Для остаточного обґрунтування вибору бібліотеки React було проведено порівняльний аналіз із ключовими конкурентами на ринку фронтенд-розроблення: фреймворком Angular (від Google) та прогресивним фреймворком Vue.js [11], [12]. Оцінювання проводилося за параметрами швидкості рендерингу, архітектурної складності та придатності для реалізації SPA-застосунків із великою кількістю реактивних форм. Результати аналізу наведено в табл. 2.2.

Таблиця 2.2 – Порівняльний аналіз сучасних JavaScript-фреймворків/бібліотек

Критерій порівняння	React	Angular	Vue.js
1	2	3	4
Архітектурний підхід	Бібліотека (компонентний підхід)	Повноцінний MVC-фреймворк	Прогресивний фреймворк
Механізм оновлення DOM	Virtual DOM (висока швидкодія)	Real DOM (використовує Zone.js)	Virtual DOM
Крива навчання та гнучкість	Середня (висока гнучкість вибору бібліотек)	Висока (строгі правила та шаблони)	Низька (простий синтаксис)

Кінець таблиці 2.2

1	2	3	4
Розмір базового бандла	~30-40 KB (легкий)	~140-150 KB (важкий)	~20-30 KB (дуже легкий)
Підтримка TypeScript	Нативна (через TSX-шаблони)	Нативна (сувора інтеграція)	Підтримується (через Composition API)

Як свідчать дана таблиця, React пропонує оптимальний баланс між продуктивністю (завдяки алгоритму Virtual DOM), легкою вагою та повною свободою в побудові архітектури. Angular є занадто громіздким для завдань соціальної служби, а екосистема React має ширший вибір готових UI-бібліотек (зокрема Material-UI) порівняно з Vue.js.

Написання вихідного коду здійснювалося у безкоштовному інтегрованому середовищі розроблення Visual Studio Code (VS Code). Воно відрізняється високою швидкістю роботи, гнучкістю налаштувань та має глибоку інтеграцію з мовою TypeScript. Для забезпечення збереження історії змін та можливості відкату до попередніх версій застосовувалася розподілена система контролю версій Git спільно з сервісом хостингу проєктів GitHub [13].

2.3 Обґрунтування вибору хмарної платформи Supabase

Побудова надійної серверної інфраструктури для роботи з конфіденційними даними громадян вимагає виваженого підходу до вибору технологій зберігання та обробки інформації. Традиційний підхід, що передбачає написання власного серверного коду з нуля (наприклад, з використанням Node.js чи Python) та самостійне адміністрування баз даних, вимагає значних часових витрат. Тому для оптимізації процесу розроблення було прийнято рішення використати архітектурну модель «Бекенд як послуга» (Backend as a Service, BaaS). У якості основної платформи було обрано

Supabase — відкриту (open-source) альтернативу популярним закритим екосистемам, яка надає повноцінний набір інструментів для створення сучасних вебзастосунків [6].

Фундаментальною відмінністю та головною перевагою Supabase є те, що її ядром виступає повноцінна об'єктно-реляційна система керування базами даних (СКБД) PostgreSQL [5]. На відміну від нереляційних (NoSQL) рішень, які часто використовуються в інших BaaS-платформах, PostgreSQL забезпечує сувору типізацію даних та повну відповідність принципам ACID (атомарність, узгодженість, ізолюваність, довговічність). Це є критично важливою вимогою для проєктування баз даних в державних органах, де інформація про бенефіціарів має складну структуру. Завдяки реляційній природі PostgreSQL було реалізовано ефективні зв'язки типу «багато-до-багатьох» між громадянами та пільговими категоріями, що гарантує цілісність даних (Data Integrity) на рівні самої бази.

Для забезпечення взаємодії між клієнтським застосунком (React) та базою даних платформа Supabase використовує інструмент PostgREST [14]. Це вебсервер, який автоматично перетворює структуру реляційної бази даних PostgreSQL у повноцінний RESTful API (Application Programming Interface). Завдяки цьому відпадає необхідність ручного написання маршрутів (роутів) для виконання базових операцій (CRUD). PostgREST аналізує структуру таблиць, типи даних та зовнішні ключі (Foreign Keys), забезпечуючи миттєву та безпечну передачу даних у форматі JSON безпосередньо з бази на клієнт.

Питання безпеки в обраній екосистемі вирішується на найнижчому рівні завдяки вбудованому механізму політик безпеки на рівні рядків (Row Level Security, RLS), який є нативною функцією PostgreSQL. Замість того, щоб перевіряти права доступу користувачів у програмному коді серверного застосунку, перевірка здійснюється безпосередньо ядром бази даних. Було налаштовано правила, згідно з якими кожен автентифікований працівник соціальної служби отримує доступ виключно до тих записів (рядків таблиці

бенефіціарів), які належать до його юрисдикції, що повністю відповідає стандартам захисту персональних даних.

Додатково платформа Supabase надає модуль Storage, призначений для роботи з великими бінарними об'єктами. Він використовується для безпечного збереження цифрових копій документів громадян (скан-копій паспортів, довідок ВПО). Інтеграція сховища безпосередньо з базою даних дозволяє прив'язувати файли до конкретних профілів бенефіціарів з дотриманням тих самих суворих правил доступу RLS, що створює єдину захищену екосистему для документообігу установи.

Враховуючи специфіку обробки текстових даних у соціальній сфері (велика кількість запитів на пошук осіб за прізвищем, іменем та по батькові), класичні оператори строгого або часткового збігу (наприклад, LIKE або ILIKE) виявляються недостатньо ефективними при роботі з реєстрами на десятки тисяч записів. Для розв'язання цієї проблеми в архітектуру бази даних PostgreSQL було інтегровано спеціалізоване розширення pg_trgm (Trigram) [15].

Це розширення дозволяє розбивати текстові рядки на триграми (послідовності з трьох символів) та будувати на їхній основі спеціальні індекси типу GIN (Generalized Inverted Index). Застосування такого індексування на колонках last_name, first_name та middle_name таблиці beneficiaries забезпечує можливість миттєвого нечіткого пошуку (fuzzy search) навіть за умови допущення орфографічних помилок оператором під час введення запиту, що критично підвищує зручність експлуатації системи у стресових умовах роботи держслужбовців.

2.4 Вимоги до технічних і програмних засобів

Для забезпечення стабільної роботи та подальшої підтримки програмного продукту необхідно сформулювати чіткі вимоги до апаратного та програмного забезпечення. Оскільки архітектура застосунку передбачає

розподіл на середовище розроблення (Development) та середовище експлуатації (Production), вимоги для них суттєво відрізняються.

2.4.1 Необхідні технічні засоби для розробки та експлуатації

Процес створення сучасних вебзастосунків із використанням інструментів збирання, середовищ програмування та локальних серверів вимагає значних обчислювальних ресурсів комп'ютера розробника. Мінімальними технічними вимогами до робочої станції інженера є наявність багатоядерного процесора архітектури x86-64 (наприклад, Intel Core i5 або AMD Ryzen 5 останніх поколінь). Для забезпечення безперебійної роботи віртуальної машини JavaScript (Node.js), інтегрованого середовища розроблення та інструментів зневадження браузера (DevTools) обсяг оперативної пам'яті (RAM) повинен становити не менше 16 ГБ. Для швидкого встановлення залежностей та збирання проєкту необхідним є використання твердотілого накопичувача (SSD) об'ємом від 256 ГБ.

Вимоги до апаратного забезпечення кінцевого користувача (працівника соціальної служби) є мінімальними, оскільки застосунок функціонує за принципом «тонкого клієнта». Основне обчислювальне навантаження, пов'язане з пошуком даних та обробкою запитів, лягає на хмарні сервери бази даних. Для комфортної роботи із застосунком достатньо стандартного офісного персонального комп'ютера чи ноутбука з процесором базового рівня, 4 ГБ оперативної пам'яті та монітором із роздільною здатністю від 1366x768 пікселів для коректного відображення широких таблиць даних. Обов'язковою технічною вимогою є наявність стабільного підключення до мережі Інтернет із пропускнуою здатністю не менше 5 Мбіт/с для швидкого завантаження електронних копій документів.

2.4.2 Вимоги до програмного забезпечення

Для конфігурації середовища розроблення робоча станція інженера повинна працювати під управлінням операційної системи Windows 10/11, macOS або сучасних дистрибутивів Linux (наприклад, Ubuntu). Основним програмним компонентом є платформа Node.js версії LTS (Long Term Support), яка включає менеджер пакетів npm для встановлення зовнішніх бібліотек (таких як React, Material-UI) [16]. Процес написання вихідного коду відбувається в редакторі Visual Studio Code, а для версіонування застосовується консольна утиліта Git.

Вимоги до програмного забезпечення робочого місця оператора системи зводяться виключно до наявності сучасного веббраузера. Оскільки клієнтська частина базується на передових вебстандартах (ECMAScript 2015+ та HTML5), для повноцінної роботи застосунку рекомендується використовувати актуальні версії браузерів: Google Chrome (починаючи з 90-ї версії), Mozilla Firefox, Microsoft Edge або Apple Safari. Використання застарілих браузерів, таких як Internet Explorer, не підтримується через відсутність у них реалізації сучасних механізмів обробки асинхронних запитів (Fetch API) та модулів JavaScript [17].

Таблиця 2.3 – Мінімальні системні вимоги для роботи із застосунком

Характеристика	Робоча станція розробника (Development)	Робоче місце користувача (Production)
1	2	3
Процесор (CPU)	Intel Core i5 / AMD Ryzen 5 (або аналог)	Двоядерний процесор базового рівня

Кінець таблиці 2.3

1	2	3
Оперативна пам'ять (RAM)	Оперативна пам'ять (RAM)	Не менше 4 ГБ
Не менше 4 ГБ	SSD (від 256 ГБ)	Не регламентується
Операційна система	Windows 10/11, macOS, Ubuntu	Будь-яка (Windows, macOS, Linux)
Мережа Інтернет	Широкосмуговий доступ	Від 5 Мбіт/с
Програмне забезпечення	Node.js, VS Code, Git	Сучасний веббраузер (Chrome, Firefox, Safari)

2.5 Висновки за розділом 2

У другому розділі було обґрунтовано вибір архітектурного підходу та технологічного стеку для реалізації автоматизованої системи обліку бенефіціарів. Встановлено, що використання клієнт-серверної архітектури у форматі Single Page Application на базі бібліотеки React та мови TypeScript забезпечує високу продуктивність інтерфейсу, надійність коду та швидке відображення змін без перезавантаження сторінок.

Вирішення завдання збереження та обробки конфіденційної інформації було здійснено шляхом інтеграції хмарної платформи Supabase. Обґрунтовано переваги використання повноцінної реляційної системи PostgreSQL, що дозволило реалізувати складні взаємозв'язки між даними та гарантувати їх безпеку за допомогою вбудованих політик Row Level Security (RLS).

Також було формалізовано вимоги до апаратної та програмної бази. Розподіл системи на «тонкого клієнта» та хмарний бекенд дозволяє впроваджувати розроблене програмне забезпечення в установах соціального

захисту без необхідності модернізації парку комп'ютерної техніки, що суттєво підвищує економічну ефективність розробки.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проектування та розроблення реляційної бази даних

Основою будь-якої інформаційної системи обліку є надійно спроектована база даних, яка забезпечує цілісність, консистентність та швидкий доступ до інформації. Для розроблюваного вебзастосунку було реалізовано реляційну базу даних під управлінням СКБД PostgreSQL у хмарному середовищі Supabase. Архітектура бази даних базується на принципах нормалізації (доведення до третьої нормальної форми), що дозволило усунути дублювання даних та забезпечити логічну структуру зберігання інформації про бенефіціарів соціальної служби [18].

Головною сутністю в системі є таблиця *beneficiaries*, яка зберігає базову персональну інформацію про особу: ідентифікатор, прізвище, ім'я, по батькові, дату народження, серію та номер паспорта, ідентифікаційний код, адресу проживання та контактні номери телефонів. Для оптимізації роботи з просторовими даними та полегшення фільтрації, райони міста винесено в окрему таблицю-довідник *districts*. Зв'язок між бенефіціаром та районом реалізовано за допомогою зовнішнього ключа (Foreign Key) зі співвідношенням «один-до-багатьох» (один район може містити багатьох бенефіціарів, але бенефіціар належить лише до одного району).

Особливістю предметної області є те, що одна особа може належати до декількох вразливих груп населення одночасно. Для реалізації цього механізму було створено таблицю-довідник *categories* та проміжну (зв'язуючу) таблицю *beneficiary_categories*. Такий підхід дозволив реалізувати складний реляційний зв'язок типу «багато-до-багатьох» (Many-to-Many), гарантуючи коректне відображення кількох статусів в одній особовій справі.

Для зберігання метаданих цифрових копій підтверджуючих документів (наприклад, довідок ВПО) реалізовано таблицю *docs*. Вона пов'язана з головною таблицею бенефіціарів відношенням «один-до-багатьох» та містить посилання (URL) на фізичні файли, що зберігаються у хмарному сховищі

Supabase Storage. Застосування каскадного видалення (ON DELETE CASCADE) забезпечує автоматичне очищення пов'язаних документів при видаленні профілю бенефіціара, що запобігає виникненню «осиротілих» записів у базі даних.

Логічну структуру спроектованої бази даних та типи реляційних зв'язків між сутностями наведено на рис. 3.1.

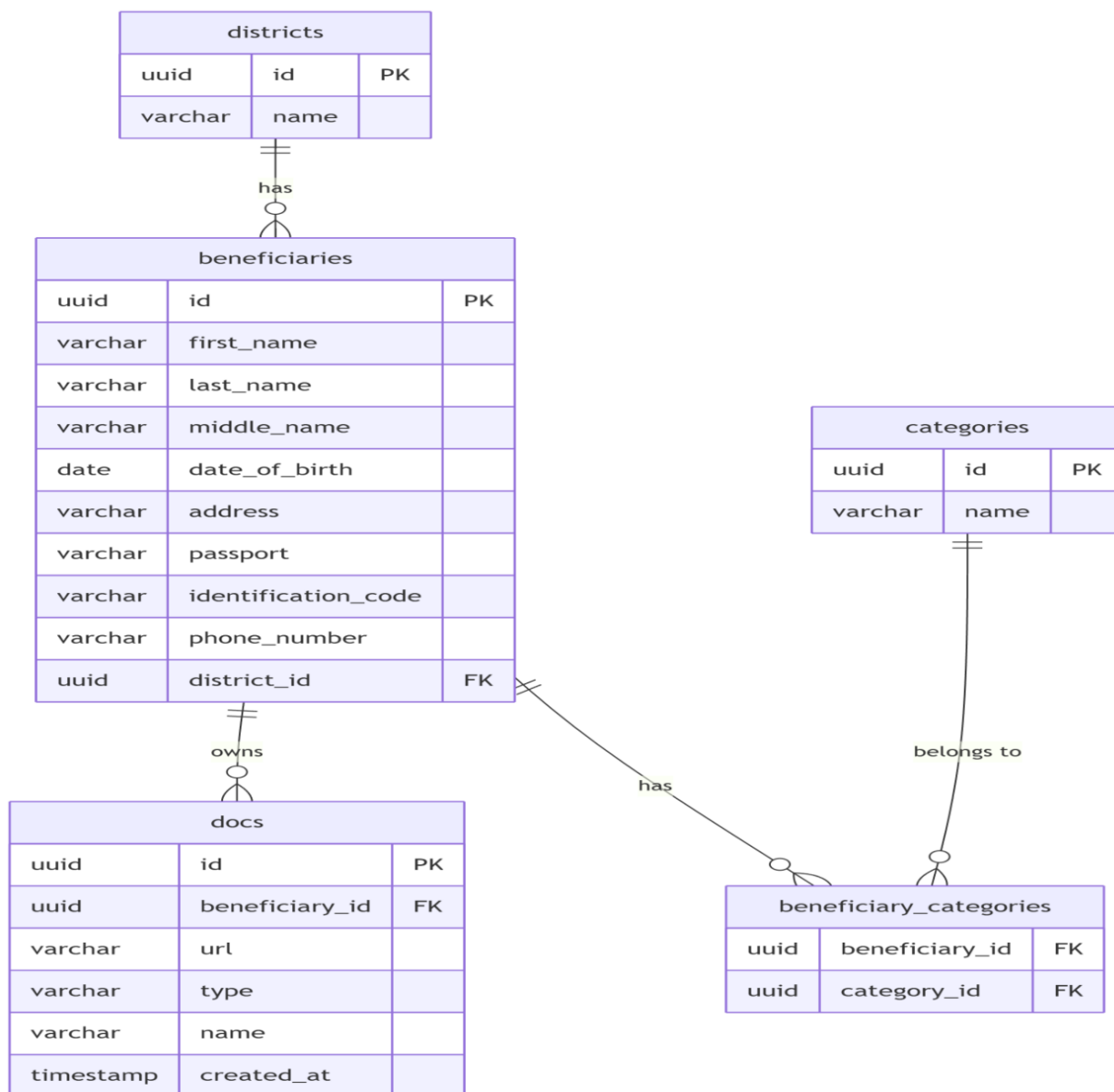


Рисунок 3.1 – ER-діаграма реляційної бази даних застосунку

Для забезпечення консистентності даних на етапі створення бази даних PostgreSQL було розроблено строгу специфікацію типів даних та обмежень (Constraints). Використання правильних типів даних мінімізує обсяг пам'яті на

диску та прискорює роботу індексів. У табл. 3.1 наведено специфікацію головної таблиці beneficiaries.

Таблиця 3.1 – Специфікація атрибутів таблиці beneficiaries

Назва поля (Атрибут)	Тип даних PostgreSQL	Обмеження (Constraints)	Опис призначення
id	UUID	PRIMARY KEY, DEFAULT gen_random_uuid()	Унікальний ідентифікатор запису (захист від передбачува- ності ID)
first_name	VARCHAR(100)	NOT NULL	Ім'я бенефіціара
last_name	VARCHAR(100)	NOT NULL	Прізвище бенефіціара
date_of_birth	DATE	NOT NULL, CHECK (date_of_birth < CURRENT_DATE)	Дата народження (не може бути в майбутньому)
identification_code	VARCHAR(10)	UNIQUE	ІПН (забезпечення унікальності особи в системі)
district_id	UUID	FOREIGN KEY REFERENCES districts(id)	Посилання на ідентифікатор району

3.2 Реалізація клієнтської частини на базі бібліотеки React

Клієнтську частину програмного забезпечення розроблено як сучасний односторінковий застосунок (SPA) з використанням бібліотеки React та суворо типізованої мови програмування TypeScript. Архітектура фронтенду базується на компонентному підході, що дозволяє розбити складний користувацький інтерфейс на незалежні, інкапсульовані модулі, які легко тестувати та повторно використовувати.

Для роботи зі станом (state) у застосунку активно застосовуються хуки React (React Hooks), зокрема `useState` для збереження локальних даних компонента (наприклад, введеного тексту у поле пошуку) та `useEffect` для виконання побічних ефектів (асинхронного завантаження даних з сервера при відкритті сторінки). Маршрутизація між різними розділами системи (реєстр, профіль, сторінка авторизації) реалізована на стороні клієнта, що забезпечує миттєвий перехід між екранами без зайвих звернень до сервера за HTML-документами.

Взаємодія клієнтської частини з хмарною базою даних Supabase здійснюється через вбудований клієнт (Supabase JS Client). Завдяки використанню інтерфейсів TypeScript (наприклад, `BeneficiaryModel`, `TableBeneficiary`, `DocModel`), структура даних, що надходить з сервера, суворо контролюється на етапі написання коду. Це унеможливлює виникнення помилок, пов'язаних зі зверненням до неіснуючих властивостей об'єктів (наприклад, неправильне найменування полів `date_of_birth` чи `address` під час збереження профілю).

Структуру основних React-компонентів, реалізованих у процесі розроблення клієнтської частини вебзастосунку, наведено у табл. 3.2.

Таблиця 3.2 – Основні компоненти клієнтської частини вебзастосунку

Назва компонента	Функціональне призначення
BeneficiaryTable	Головний компонент для відображення реєстру у вигляді таблиці. Відповідає за пагінацію, сортування та рендерінг списку осіб.
BeneficiaryProfile	Компонент-форма для перегляду, створення та редагування детальної інформації про особу (персональні дані, контакти).
FilterSidebar	Бічна панель, яка містить елементи управління (селекти) для фільтрації реєстру за районами міста та пільговими категоріями.
DocumentUploader	Модуль для завантаження сканованих копій документів. Взаємодіє з Supabase Storage та відображає прогрес завантаження.
AuthLogin	Екран автентифікації працівників Департаменту соціального захисту з полями для введення електронної пошти та пароля.

Використання компонентної архітектури у поєднанні з готовими елементами інтерфейсу бібліотеки Material-UI дозволило створити високопродуктивний, стабільний та зручний інструмент для роботи операторів соціальної служби, здатний обробляти великі масиви даних у реальному часі.

Реалізація взаємодії клієнтської частини з базою даних відбувається через асинхронні функції із застосуванням бібліотеки `@supabase/supabase-js`. Завдяки можливостям `PostgREST`, застосунок виконує комплексні запити (з об'єднанням таблиць — `JOIN`) безпосередньо з фронтенду. Це демонструє використання патерну `Builder` для побудови SQL-запиту: застосунок витягує не лише базові поля особи, але й автоматично підтягує назви районів та категорій через зовнішні ключі завдяки синтаксису вкладеного вибору (`districts (name)`). Строга типізація `Promise<BeneficiaryModel[]>` гарантує, що компоненти інтерфейсу отримають дані очікуваної структури.

3.3 Розроблення модулів фільтрації та управління документами

Для забезпечення ефективної роботи з великими масивами даних в умовах Департаменту соціального захисту населення було розроблено оптимізовані модулі багатокритеріальної фільтрації та управління цифровими копіями документів.

Модуль фільтрації реалізовано на клієнтській стороні з використанням реактивного стану (`state`). При зміні параметрів пошуку (наприклад, виборі певного району зі списку або введенні прізвища) застосунок формує динамічний запит до API Supabase. Замість завантаження всієї бази даних на клієнт, що призвело б до критичного падіння продуктивності, фільтрація відбувається безпосередньо на стороні сервера баз даних PostgreSQL. Для цього застосовуються вбудовані методи фільтрації клієнтської бібліотеки Supabase, такі як оператор `eq` (дорівнює) для точного пошуку за ідентифікатором району (`district_id`) та оператор `ilike` для нечутливого до регістру пошуку за текстовими полями (ПІБ). З метою уникнення перевантаження мережі запитами при кожному натисканні клавіші під час текстового пошуку, було впроваджено техніку «`debounce`» (затримка виконання функції), яка ініціює запит лише після того, як користувач завершив введення тексту.

Модуль управління документами вирішує проблему надійного зберігання цифрових копій (сканів паспортів, довідок ВПО тощо), які необхідні для підтвердження пільгового статусу бенефіціара. Логіка роботи модуля поділена на два послідовні етапи. На першому етапі фізичний файл, обраний користувачем, завантажується у захищене хмарне сховище (Supabase Storage) у спеціально виділену директорію (bucket). Хмарне сховище генерує унікальне публічне або підписане посилання (URL) на цей файл. На другому етапі застосунок створює новий запис у реляційній таблиці docs, де зберігає отримане посилання, тип документа (наприклад, «Паспорт» або «Ідентифікаційний код») та прив'язує його до ідентифікатора конкретного бенефіціара за допомогою зовнішнього ключа (beneficiary_id).

Оскільки система дозволяє користувачам завантажувати файли на сервер, критично важливим етапом є попередня валідація вхідних даних на стороні клієнта для запобігання завантаженню шкідливого програмного забезпечення (наприклад, виконуваних файлів .exe) або файлів надмірного обсягу, що можуть переповнити хмарне сховище. Алгоритм роботи модуля завантаження документів ілюструється на рис. 3.2.

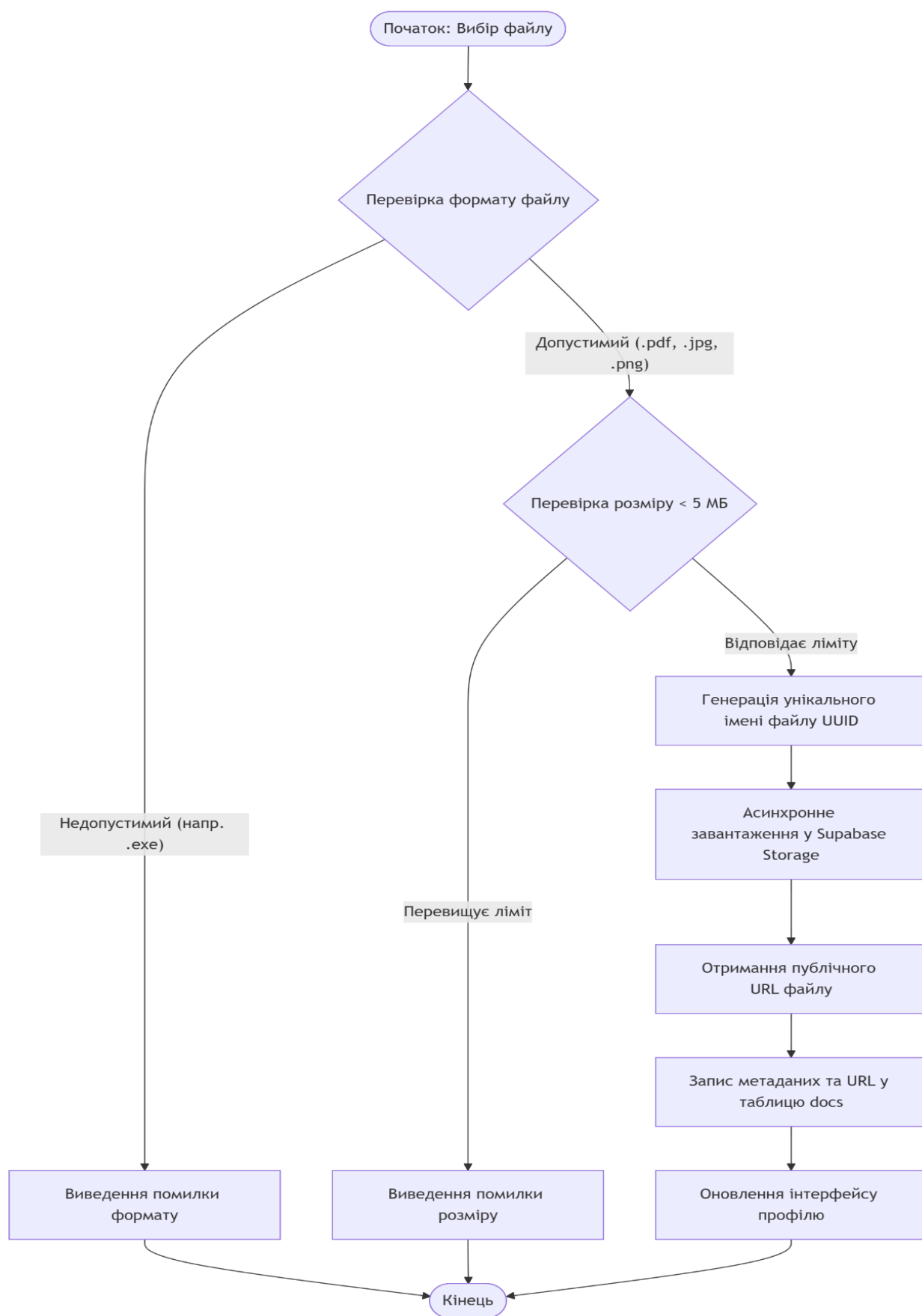


Рисунок 3.2 – Алгоритм завантаження електронної копії документа

3.4 Опис взаємодії з API хмарного сервісу

Взаємодія між клієнтським застосунком, написаним на React, та інфраструктурою Supabase здійснюється через автоматично згенерований RESTful API (завдяки компоненту PostgREST). Цей архітектурний підхід дозволяє виконувати мережеві запити безпосередньо з фронтенду, не створюючи додаткового проміжного серверного шару, що суттєво прискорює процес розроблення.

Під час інтеграції бази даних із користувацьким інтерфейсом було враховано особливості роботи внутрішнього кешування схеми бази даних (schema cache) у PostgREST. Оскільки API автоматично генерується на основі поточної структури таблиць PostgreSQL, будь-які зміни у базі даних (наприклад, додавання нових колонок address, date_of_birth або passport до таблиці beneficiaries) вимагають примусового оновлення кешу. Без виконання команди перезавантаження схеми API відхиляє запити, генеруючи специфічні помилки маршрутизації (наприклад, коди помилок серії PGRST). Цю архітектурну особливість було враховано шляхом виконання SQL-команд оновлення кешу (через виклик NOTIFY pgrst, 'reload schema') після кожної модифікації структури (міграції).

Процес взаємодії компонентів системи під час виконання комплексної операції (наприклад, збереження нового профілю бенефіціара разом із його документами та категоріями) має транзакційний характер. Логіку цієї взаємодії наочно зображено за допомогою UML-діаграми послідовності на рис. 3.3.

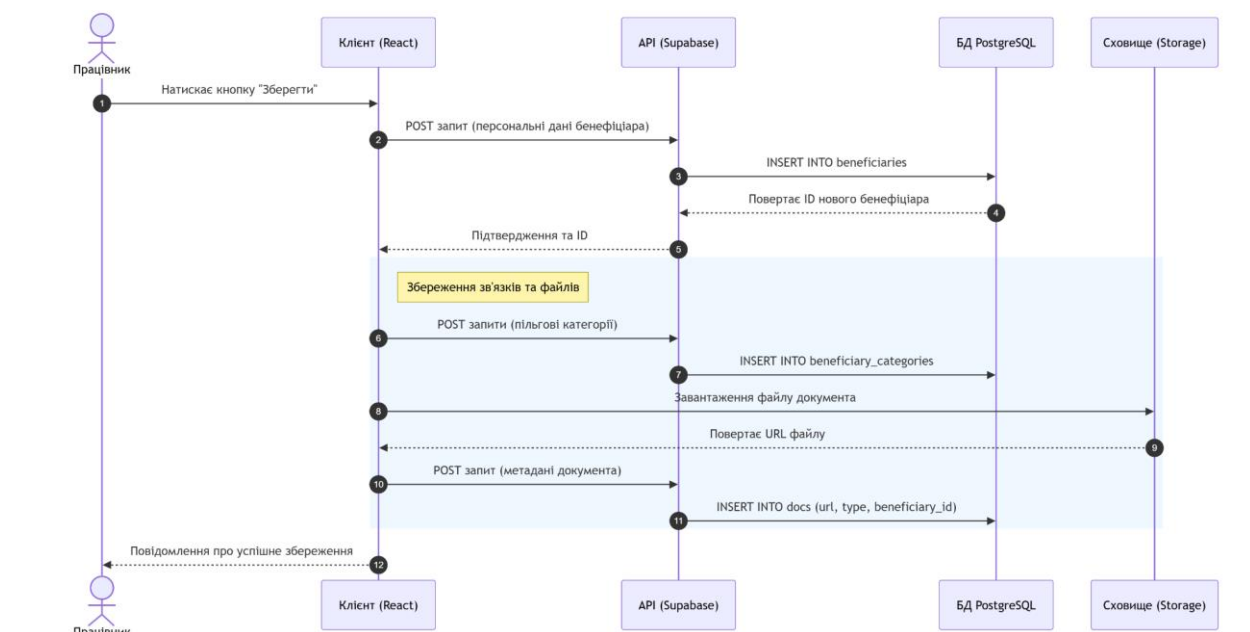


Рисунок 3.3 – UML-діаграма послідовності процесу збереження профілю бенефіціара

Реалізація обміну даними через стандартизований API з використанням суворої типізації TypeScript дозволила мінімізувати кількість помилок під час виконання програми (runtime errors) та забезпечила стабільну синхронізацію даних між клієнтом і сервером.

3.5 Висновки за розділом 3

У третьому розділі було детально розглянуто практичний аспект створення інформаційної системи. Спроековано та реалізовано оптимізовану реляційну базу даних PostgreSQL. Використання нормалізованої структури та механізму зовнішніх ключів дозволило коректно відобразити складну ієрархію зв'язків між громадянами, районами міста та множинними пільговими категоріями, повністю усунувши проблему дублювання даних.

Розроблено клієнтську частину програмного забезпечення за парадигмою Single Page Application з використанням компонентної моделі бібліотеки React. Створено інтерактивні інтерфейси для реєстрації

бенефіціарів, відображення реєстру у вигляді таблиць із вбудованою пагінацією та багатокритеріальною фільтрацією.

Успішно інтегровано сервіси хмарної платформи Supabase, що забезпечило застосунок надійним RESTful API, механізмами автентифікації користувачів та захищеним файловим сховищем для електронних копій документів. Процеси обміну даними були оптимізовані з урахуванням особливостей кешування архітектури PostgREST. У результаті виконання завдань цього розділу було отримано повністю працездатний програмний продукт, готовий до етапу тестування та дослідної експлуатації.

4 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Методика проведення функціонального тестування

Забезпечення високого рівня якості програмного продукту є невід'ємною складовою життєвого циклу розроблення програмного забезпечення. Зважаючи на те, що система призначена для роботи з персональними даними громадян у державних установах, будь-які логічні помилки або збої в роботі інтерфейсу є неприпустимими. Для перевірки коректності роботи всіх модулів системи було застосовано метод функціонального тестування за принципом «чорного ящика» (Black-box testing). Цей підхід передбачає перевірку вихідної поведінки системи на основі заданих вхідних даних без безпосереднього аналізу внутрішньої структури вихідного коду.

Процес тестування було розділено на кілька логічних етапів, що відповідають основним бізнес-процесам установи соціального захисту. На першому етапі тестувалася підсистема автентифікації: перевірялася реакція застосунку на введення невалідних облікових даних, блокування доступу неавторизованим користувачам до захищених маршрутів (роутів) та коректність генерації сесійних токенів.

На другому етапі здійснювалася перевірка основного модуля управління бенефіціарами (CRUD-операції). Досліджувалася робота форм введення даних: валідація обов'язкових полів (система не повинна дозволяти збереження картки без вказання прізвища або ідентифікаційного коду), коректність обробки зв'язків «багато-до-багатьох» при присвоєнні особі одразу кількох пільгових категорій.

Третій етап охоплював тестування підсистеми управління документами. Проводилися спроби завантаження файлів різних форматів (PDF, JPEG, PNG) та розмірів, щоб переконатися, що хмарне сховище Supabase

Storage коректно обробляє вхідні потоки даних і повертає валідні посилання для відображення в інтерфейсі.

Окрему увагу під час тестування було приділено питанням доступності користувацького інтерфейсу (Accessibility) згідно з міжнародними настановами WCAG (Web Content Accessibility Guidelines) [19]. Перевірено можливість повноцінної роботи із застосунком без використання комп'ютерної миші: усі інтерактивні елементи (форми, кнопки збереження, елементи пагінації) коректно обробляють навігацію за допомогою клавіші «Tab» та мають відповідні візуальні індикатори фокусу. Крім того, перевірено реакцію системи на критичні непередбачувані ситуації (Edge Cases), що виникають під час збоїв інфраструктури, результати тестування наведені у табл. 4.1

Таблиця 4.1 – Розширений план функціонального тестування вебзастосунку

№	Опис тестового сценарію (Test Case)	Очікуваний результат	Фактичний результат	Статус
1	2	3	4	5
1	Введення неіснуючого email / пароля на сторінці входу	Виведення повідомлення про помилку автентифікації	Виведено повідомлення «Невірні облікові дані»	Успішно
2	Створення бенефіціара з порожнім полем «ІПН»	Блокування кнопки «Зберегти», підсвічування поля	Кнопку заблоковано, поле підсвічено червоним	Успішно

Кінець таблиці 4.1

1	2	3	4	5
3	Присвоєння особі категорій «ВПО» та «Пенсіонер» одночасно	Створення двох записів у проміжній таблиці	Записи створено, інтерфейс відображає дві мітки	Успішно
4	Імітація обриву з'єднання (Offline) під час збереження картки	Перехоплення помилки (try/catch), збереження введених даних у формі	Виведено сповіщення "Помилка мережі", дані форми не втрачено	Успішно
5	Спроба завантаження шкідливого файлу (script.exe) у Storage	Блокування завантаження через перевірку MIME-типу	Виведено попередження про недопустимий формат файлу	Успішно

Результати функціонального тестування підтвердили, що всі заявлені в постановці завдання функції працюють коректно, а клієнтський інтерфейс адекватно реагує як на правильні, так і на помилкові дії користувача.

4.2 Аналіз результатів тестування продуктивності та безпеки

Окрім функціональної повноти, програмне забезпечення державного сектору повинно відповідати суворим нефункціональним вимогам, зокрема щодо швидкодії та захисту даних. Для дослідження продуктивності

клієнтської частини застосунку використовувався набір інструментів розробника Google Chrome DevTools та інтегрована система аудиту вебзастосунків Google Lighthouse.

Під час тестування продуктивності моделювалася ситуація завантаження реєстру бенефіціарів, що містить понад 1000 записів. Завдяки реалізації механізму серверної пагінації (PostgREST API повертає дані порціями по 20-50 записів) та використанню алгоритмів віртуального дерева (Virtual DOM) бібліотеки React, час первинного відображення сторінки (First Contentful Paint) склав менше 0.8 секунди. Реакція інтерфейсу на перемикання вкладок та відкриття модальних вікон відбувається без затримок (менше 100 мс), що забезпечує плавний користувацький досвід. Результати автоматизованого аудиту продуктивності та доступності інтерфейсу наведено у табл. 4.2.

Таблиця 4.2 – Результати аудиту клієнтської частини за допомогою Google Lighthouse

Метрика тестування	Опис метрики	Отриманий бал (зі 100)
1	2	3
Performance (Продуктивність)	Швидкість завантаження скриптів, рендерингу елементів та загальна реактивність інтерфейсу	94
Accessibility (Доступність)	Зручність використання для людей з порушеннями зору (контрастність, семантика тегів)	98

Кінець таблиці 4.2

1	2	3
Best Practices (Кращі практики)	Відповідність сучасним стандартам веброзробки (використання HTTPS, відсутність застарілих API)	100

Окремим, критично важливим етапом стало тестування безпеки (Security Testing) та механізмів розмежування доступу. Архітектура бази даних Supabase використовує політики безпеки на рівні рядків (Row Level Security). Для перевірки їхньої ефективності було проведено експеримент: у системі було створено два облікові записи працівників, закріплених за різними районами міста (наприклад, Заводським та Комунарським).

При спробі працівника Заводського району виконати прямий HTTP-запит до API (минаючи клієнтський інтерфейс) з метою отримання даних бенефіціарів Комунарського району, сервер бази даних PostgreSQL миттєво відхилив транзакцію та повернув порожній масив даних. Це експериментально доводить, що система захисту функціонує не лише на рівні візуального приховування елементів у React, а на фундаментальному рівні ядра бази даних. Крім того, перевірено механізм дії JSON Web Tokens (JWT): після закінчення терміну дії токена (expire time) будь-які запити блокуються сервером із помилкою 401 Unauthorized, а клієнтський застосунок автоматично перенаправляє користувача на сторінку повторної авторизації [20].

Реалізацію цього криптографічного та логічного захисту забезпечують написані SQL-політики (Row Level Security). Замість того, щоб покладатися виключно на приховування елементів у React-інтерфейсі, безпека гарантується на рівні ядра бази даних. У коді програмного забезпечення написано політику, яка дозволяє оператору читати лише ті картки бенефіціарів, які належать до його адміністративного району. Цей SQL-запит використовує вбудовану

функцію `auth.uid()`, яка безпечно витягує ідентифікатор поточного користувача з розшифрованого та криптографічно підписаного JWT-токена сесії (JSON Web Token). База даних самостійно порівнює закріплений за оператором район (`assigned_district_id`) із районом конкретного бенефіціара (`district_id`). Як засвідчило тестування, обійти це обмеження неможливо навіть при прямому доступі до REST API за допомогою сторонніх інструментів (наприклад, Postman), оскільки запит фільтрується до моменту формування відповіді сервером.

4.3 Інструкція користувача та рекомендації з експлуатації

Успішне впровадження нового інформаційного продукту в державній установі значною мірою залежить від наявності чіткого та зрозумілого регламенту роботи. Розроблений вебзастосунок орієнтований на працівників соціальних служб, тому його інтерфейс спроектовано за принципами максимальної простоти та наочності.

Робота із системою розпочинається з етапу автентифікації. Працівник повинен відкрити вебзастосунок у браузері та на стартовій сторінці ввести свої унікальні облікові дані (електронну пошту та пароль), які попередньо генеруються адміністратором Департаменту. Після успішної перевірки токена доступу користувач потрапляє до головного робочого простору — електронного реєстру бенефіціарів свого району.

Базовий алгоритм (інструкція) зі створення нової особової справи бенефіціара складається з таких кроків:

На головній панелі інструментів необхідно натиснути кнопку додавання нового запису.

У відкритому модальному вікні (електронній картці) оператор послідовно заповнює текстові поля: ПІБ, дату та місце народження, адресу фактичного проживання, контактні дані, а також реквізити документів (серія паспорта, ідентифікаційний код).

У блоці «Категорії» за допомогою випадуючого списку (селекта) працівник обирає пільгові статуси особи. Завдяки підтримці зв'язків «багато-до-багатьох», системі можна передати одразу кілька статусів (наприклад, «ВПО» та «Багатодітна родина»).

Після перевірки введених даних натискається кнопка «Зберегти». Застосунок автоматично відправляє дані на сервер Supabase, генерує ідентифікатор особи і закриває форму, оновлюючи загальну таблицю реєстру.

Для завантаження цифрових копій документів користувач повинен відкрити вже збережений профіль особи. У спеціально відведеному блоці управління файлами (Dropzone) необхідно натиснути іконку завантаження та обрати відсканований файл на локальному комп'ютері. Система автоматично збереже документ у хмарне сховище і відобразить посилання на нього в картці бенефіціара.

З метою забезпечення стабільної та безпечної роботи програмного забезпечення сформовано такі рекомендації для персоналу:

Для роботи із застосунком необхідно використовувати актуальні версії браузерів на базі рушія Chromium (Google Chrome, Microsoft Edge) для гарантованої підтримки всіх сучасних вебстандартів відображення.

З міркувань безпеки персональних даних забороняється передавати власні облікові дані колегам, оскільки система веде аудит дій (логів) з прив'язкою до конкретного профілю оператора.

Перед завантаженням файлів у хмарне сховище рекомендується здійснювати візуальну перевірку якості сканованих копій (відсутність засвітів, читабельність тексту) та дотримуватися оптимального розміру файлу (до 5 МБ) для економії хмарного простору установи.

4.4 Висновки за розділом 4

У четвертому розділі було проведено комплексне експериментальне дослідження розробленого програмного забезпечення та сформовано нормативну документацію для його експлуатації.

Використання методології функціонального тестування («чорного ящика») дозволило підтвердити повну відповідність фактичної поведінки системи закладеним вимогам. Доведено коректність виконання всіх базових операцій зі створення, читання, оновлення та видалення записів (CRUD), а також успішну обробку складних множинних зв'язків між об'єктами бази даних під час роботи з формами.

Дослідження продуктивності за допомогою інструментів автоматизованого аудиту показало високу швидкодію клієнтської частини. Завдяки архітектурі Single Page Application на базі React та оптимізованих обробці запитів сервером PostgREST, інтерфейс миттєво реагує на дії користувача. Експериментальна перевірка політик безпеки Row Level Security (RLS) підтвердила абсолютну надійність розмежування доступу до персональних даних громадян: спроби несанкціонованого доступу до інформації інших районів успішно блокуються ядром бази даних PostgreSQL.

Розроблена інструкція користувача та рекомендації з експлуатації надають вичерпний алгоритм дій для працівників соціальної служби, що дозволяє швидко інтегрувати застосунок у робочий процес установи з мінімальними витратами часу на адаптацію персоналу. Результати експериментальних досліджень підтверджують, що розроблений інформаційний продукт є стабільним, безпечним і повністю готовим до впровадження та промислової експлуатації.

ВИСНОВКИ

У роботі вирішено актуальне науково-практичне завдання, що полягає у підвищенні ефективності роботи соціальних працівників шляхом розроблення спеціалізованого клієнт-серверного вебзастосунку для автоматизованого обліку соціально вразливих груп населення.

За результатами виконання роботи можна зробити такі висновки:

Проведено системний аналіз предметної області та бізнес-процесів Департаменту соціального захисту населення. Виявлено, що використання паперових архівів та неспеціалізованих систем управління завданнями (таких як Trello чи Jira) не забезпечує належного рівня безпеки, ускладнює наскрізний пошук та не дозволяє вести повноцінний облік бенефіціарів із множинними пільговими статусами. На основі виявлених недоліків сформовано детальний перелік функціональних та нефункціональних вимог до створюваної системи.

Обґрунтовано вибір архітектурного підходу та технологічного стеку для реалізації проєкту. Доведено доцільність використання архітектури Single Page Application (SPA) на базі бібліотеки React та строго типізованої мови TypeScript для забезпечення швидкодії клієнтського інтерфейсу. Для серверної інфраструктури обрано модель Backend-as-a-Service (BaaS) на базі платформи Supabase, що дозволило делегувати завдання управління серверами та фокусуватися на реалізації бізнес-логіки.

Спроектовано та реалізовано реляційну базу даних під управлінням СКБД PostgreSQL. Завдяки нормалізації структури даних та використанню проміжних таблиць, успішно реалізовано зв'язки типу «багато-до-багатьох», що дозволило закріплювати за однією особою одразу кілька пільгових категорій. Налаштовано інтеграцію з модулем Supabase Storage для безпечного збереження та прив'язки електронних копій підтверджуючих документів до профілів громадян.

Розроблено повнофункціональний користувацький інтерфейс, адаптований під потреби операторів соціальної служби. Реалізовано модулі безпечної аутентифікації працівників, інтерактивний реєстр бенефіціарів із функцією серверної пагінації та систему багатокритеріальної фільтрації за територіальними районами міста й категоріями вразливості.

Проведено комплексне експериментальне дослідження розробленого програмного продукту. Функціональне тестування підтвердило безпомилкову роботу всіх модулів обробки даних (CRUD). Результати аудиту продуктивності показали високу швидкість рендерингу інтерфейсу при роботі з великими масивами даних. Перевірка механізмів Row Level Security (RLS) підтвердила надійний криптографічний захист персональних даних громадян та коректне розмежування прав доступу між працівниками різних районів.

Розроблене програмне забезпечення повністю відповідає поставленому технічному завданню та має високу практичну цінність. Впровадження системи дозволяє суттєво скоротити час на обробку звернень, мінімізувати бюрократичне навантаження на персонал, забезпечити надійне збереження цифрових копій документів та вивести процес управління соціальними послугами на якісно новий цифровий рівень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Atlassian Jira Software. Atlassian. URL: <https://www.atlassian.com/software/jira> (дата звернення: 10.05.2026).
2. Atlassian Trello. Atlassian. URL: <https://trello.com/> (дата звернення: 10.05.2026).
3. TypeScript: Typed JavaScript at Any Scale. TypeScript Official Website. URL: <https://www.typescriptlang.org/> (дата звернення: 10.05.2026).
4. React – A JavaScript library for building user interfaces. React Official Website. URL: <https://react.dev/> (дата звернення: 10.05.2026).
5. PostgreSQL: The World's Most Advanced Open Source Relational Database. PostgreSQL Official Website. URL: <https://www.postgresql.org/docs/> (дата звернення: 10.05.2026).
6. Supabase: The Open Source Firebase Alternative. Supabase Documentation. URL: <https://supabase.com/docs> (дата звернення: 10.05.2026).
7. Linear – A better way to build products. Linear. URL: <https://linear.app/> (дата звернення: 10.05.2026).
8. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. ISO. URL: <https://www.iso.org/standard/35733.html> (дата звернення: 10.05.2026).
9. Single Page Application (SPA). MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Glossary/SPA> (дата звернення: 10.05.2026).
10. Material-UI: The React component library you always wanted. MUI. URL: <https://mui.com/> (дата звернення: 10.05.2026).
11. Angular – The modern web developer's platform. Angular Official Website. URL: <https://angular.io/> (дата звернення: 10.05.2026).
12. Vue.js – The Progressive JavaScript Framework. Vue.js Official Website. URL: <https://vuejs.org/> (дата звернення: 10.05.2026).

13. Git Documentation. Git. URL: <https://git-scm.com/doc> (дата звернення: 10.05.2026).
14. PostgREST: REST API for any Postgres database. PostgREST Documentation. URL: <https://postgrest.org/> (дата звернення: 10.05.2026).
15. pg_trgm — Text similarity measurement and index searching based on trigrams. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/current/pgtrgm.html> (дата звернення: 10.05.2026).
16. Node.js Documentation. Node.js. URL: <https://nodejs.org/en/docs/> (дата звернення: 10.05.2026).
17. Fetch API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 10.05.2026).
18. Database Normalization Description. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description> (дата звернення: 10.05.2026).
19. Web Content Accessibility Guidelines (WCAG) 2.1. W3C. URL: <https://www.w3.org/TR/WCAG21/> (дата звернення: 10.05.2026).
20. JSON Web Token Introduction. JWT.io. URL: <https://jwt.io/introduction> (дата звернення: 10.05.2026).

ДОДАТОК А
Текст програми

A.1 Текст файлу types/beneficiaries.ts

```
export interface BaseCatalog {
  id: string;
  name: string;
}
```

```
export enum DocType {
  passport = 'Паспорт',
  identification_code = 'Ідентифікаційний код',
  vpo = 'ВПО',
  ubd = 'Учасники бойових дій',
  invalids = 'Інваліди',
  pensioners = 'Пенсіонери',
  manykids = 'Багатодітні',
  poor = 'Малозабезпечені',
  family = 'Родина'
}
```

```
export interface DocModel {
  id?: string;
  url?: string;
  file?: File;
  type: string;
  name: string;
}
```

```
export interface TableBeneficiary {
  id?: string;
  first_name: string;
  middle_name: string;
  last_name: string;
  districts?: BaseCatalog;
  categories?: BaseCatalog[];
  is_all_docs: boolean;
}
```

```
export interface BeneficiaryModel extends TableBeneficiary {
  date_of_birth: string | Date | "";
}
```

```

address: string;
passport: string;
identification_code: string;
place_of_birth: string;
district_id: string | null;
phone_number: string;
phone_number_extra: string | null;
docs?: DocModel[];
}

```

```

export interface PaginationModel {
  page: number;
  pageSize: number;
}

```

A.2 Текст файла helpers/useBeneficiariesHelper.tsx

```

import { supabase } from '../supabase';
import { BaseCatalog, DocModel } from '../types/beneficiaries';

export const insertCategories = async (
  id: string,
  categories: BaseCatalog[] | undefined
) => {
  const categoriesInsertData = categories?.map((category: { id: string }) => ({
    beneficiary_id: id,
    category_id: category.id,
  }));

  const { error } = await supabase
    .from('beneficiary_categories')
    .insert(categoriesInsertData);

  if (error) {
    throw new Error(error.message);
  }
};

export const deleteDocsById = async (id: string) => {
  const { data: urls_data, error: get_Url_Error } = await supabase

```

```

    .from('docs')
    .select('url')
    .eq('beneficiary_id', id);

if (get_Url_Error) throw new Error(get_Url_Error.message);

if (urls_data && urls_data.length > 0) {
  const { error: del_file_error } = await supabase.storage
    .from('docs')
    .remove(urls_data.map((obj) => obj.url));

  if (del_file_error) throw new Error(del_file_error.message);

  const { error: del_error } = await supabase
    .from('docs')
    .delete()
    .eq('beneficiary_id', id);

  if (del_error) throw new Error(del_error.message);
}
};

export const uploadFile = async (doc: DocModel, beneficiary_id: string) => {
  if (doc.file) {
    const fileExt = doc.file.name.split('.').pop();
    const fileName = `${Math.random()}.${fileExt}`;
    const filePath = `${fileName}`;

    const { error: upload_error } = await supabase.storage
      .from('docs')
      .upload(filePath, doc.file);

    if (upload_error) throw new Error(upload_error.message);

    const docsInsertData = {
      id: doc.id,
      type: doc.type,
      name: doc.name,
      beneficiary_id,
      url: filePath,
    };

    const { error: docs_error } = await supabase
      .from('docs')
      .insert(docsInsertData);
  }
};

```

```

    if (docs_error) throw new Error(docs_error.message);
  } else {
    throw new Error('Файл не надано для завантаження');
  }
};

export const deleteDocs = async (docs: DocModel[]) => {
  if (docs.length) {
    const { error: del_file_error } = await supabase.storage
      .from('docs')
      .remove(docs.map((obj) => obj.url!));

    if (del_file_error) throw new Error(del_file_error.message);

    const { error: del_error } = await supabase
      .from('docs')
      .delete()
      .in('url', docs.map((obj) => obj.url!));

    if (del_error) throw new Error(del_error.message);
  }
};

export function removeDuplicates(docs: DocModel[]): DocModel[] {
  return docs.filter(
    (doc) =>
      docs.filter((d) => JSON.stringify(d) === JSON.stringify(doc)).length === 1
  );
}

```

A.3 Текст файлу `hooks/useBeneficiaries.tsx`

```

import {
  deleteDocs,
  deleteDocsById,
  insertCategories,
  removeDuplicates,
  uploadFile,
} from '../helpers/useBeneficiariesHelper';
import { supabase } from '../supabase';
import {
  BaseCatalog,

```

```

BeneficiaryModel,
DocModel,
PaginationModel,
TableBeneficiary,
} from '../types/beneficiaries';

export const useBeneficiaries = () => {
  const fetchCatalog = async (catalog: string) => {
    const { data, error } = await supabase
      .from(catalog)
      .select<string, BaseCatalog>('*');

    if (error) throw new Error(error.message);
    return data;
  };

  const fetchBeneficiaries = async (
    pagination: PaginationModel,
    searchTerms?: string,
    categoryIds?: string[],
    districtIds?: string[]
  ) => {
    const { page, pageSize } = pagination;
    const offset = page * pageSize;
    let beneficiaryIds: string[] = [];

    if (categoryIds && categoryIds.length > 0) {
      const { data: beneficiaries, error: beneficiaryError } = await supabase
        .from('beneficiary_categories')
        .select('beneficiary_id')
        .in('category_id', categoryIds);

      if (beneficiaryError) throw new Error(beneficiaryError.message);
      beneficiaryIds = beneficiaries.map((b) => b.beneficiary_id);

      if (beneficiaryIds.length === 0) return { data: [], count: null };
    }

    let query = supabase
      .from('beneficiaries')
      .select<string, TableBeneficiary>(
        'id, first_name, middle_name, last_name, is_all_docs, ...districts (name),
        categories (name)',
        { count: 'exact' }
      )

```

```

    )
    .range(offset, offset + pageSize - 1);

    if (searchTerms) {
      const subSearchTerms = searchTerms.split(' ');
      subSearchTerms.map((element) => {
        query = query.or(
          `first_name.ilike.%${element}%,last_name.ilike.%${element}%,middle_name.ilike.
          %${element}%`
        );
      });
    }

    if (beneficiaryIds.length > 0) query = query.in('id', beneficiaryIds);
    if (districtIds && districtIds.length > 0) query = query.in('district_id', districtIds);

    const { data, error, count } = await query;
    if (error) throw new Error(error.message);

    return { data, count };
  };

  const fetchBeneficiaryById = async (id: string) => {
    const { data, error } = await supabase
      .from('beneficiaries')
      .select('*', districts (id,name), categories (id,name), docs(id, url, type, name))
      .eq('id', id)
      .single();

    if (error) throw new Error(error.message);

    return {
      ...data,
      categories: data.categories ? data.categories.map((c: any) => c.id) : [],
      district_id: data.districts?.id || data.district_id || "",
      docs: data.docs || []
    } as any;
  };

  const createBeneficiary = async (newBeneficiary: BeneficiaryModel) => {
    const { categories, docs, ...beneficiaryData } = newBeneficiary;

    const { data, error } = await supabase
      .from('beneficiaries')

```

```

    .insert(beneficiaryData)
    .select('id');

if (error) throw new Error(error.message);
const beneficiaryId = data[0]?.id;

if (beneficiaryId) {
  await insertCategories(beneficiaryId, categories);
  if (docs) {
    const uniqueDocs = removeDuplicates(docs);
    for (const doc of uniqueDocs) {
      await uploadFile(doc, beneficiaryId);
    }
  }
}

const { error: isDocError } = await supabase
  .from('beneficiaries')
  .update({ is_all_docs: false })
  .eq('id', beneficiaryId);

if (isDocError) throw new Error(isDocError.message);
};

const updateBeneficiary = async (id: string, updatedBeneficiary: BeneficiaryModel)
=> {
  const { categories, docs, districts, id: bId, ...beneficiaryData } = updatedBeneficiary;

  if (docs && docs?.length > 0) {
    const uniqueDocs = removeDuplicates(docs);
    let docsToDelete: DocModel[] = [];
    for (const doc of uniqueDocs) {
      if (doc.file) {
        await uploadFile(doc, id);
      } else {
        docsToDelete = [...docsToDelete, doc];
      }
    }
    await deleteDocs(docsToDelete);
  }

  if (categories) {
    const { error: delerror } = await supabase
      .from('beneficiary_categories')
      .delete()

```

```

        .eq('beneficiary_id', id);

    if (delerror) throw new Error(delerror.message);

    if (categories.length > 0) {
        await insertCategories(id, categories);
    }
}

const { error } = await supabase
    .from('beneficiaries')
    .update(beneficiaryData)
    .eq('id', id);

    if (error) throw new Error(error.message);
};

const deleteBeneficiary = async (id: string) => {
    await deleteDocsById(id);
    const { error } = await supabase
        .from('beneficiaries')
        .delete()
        .eq('id', id);

    if (error) throw new Error(error.message);
};

async function downloadFile(path: string) {
    const { data, error } = await supabase.storage.from('docs').download(path);
    if (error) throw new Error(error.message);

    return URL.createObjectURL(data);
}

return {
    fetchCatalog,
    fetchBeneficiaries,
    fetchBeneficiaryById,
    createBeneficiary,
    updateBeneficiary,
    deleteBeneficiary,
    downloadFile,
};
};

```

A.4 Текст файла routes/ProtectRoute.tsx

```
import { useSupabaseSession } from '../providers/supabaseProvider';
import { ReactNode } from 'react';
import { Navigate } from 'react-router-dom';

interface ProtectedRouteProps {
  children: ReactNode;
}

const ProtectedRoute = ({ children }: ProtectedRouteProps) => {
  const { session } = useSupabaseSession();

  if (!session) {
    return <Navigate to="/login" replace />;
  }

  return children;
};

export default ProtectedRoute;
```

A.5 Текст файла providers/supabaseProvider.tsx

```
import { supabase } from '../supabase/index';
import { Session } from '@supabase/supabase-js';
import { ReactNode, createContext, useContext, useEffect, useState } from 'react';

interface SupabaseSessionContextType {
  session: Session | null;
  loading: boolean;
  isAdmin: boolean;
  signOut: () => Promise<void>;
}

const SupabaseSessionContext = createContext<SupabaseSessionContextType>({
  session: null,
  loading: true,
  isAdmin: false,
```

```

    signOut: async () => {},
  });

export const SupabaseSessionProvider = ({ children }: { children: ReactNode }) => {
  const [session, setSession] = useState<Session | null>(null);
  const [loading, setLoading] = useState(true);
  const [isAdmin, setIsAdmin] = useState(false);

  useEffect(() => {
    const fetchIsAdmin = async (session: Session) => {
      try {
        const { data, error } = await supabase
          .from('profiles')
          .select<string, boolean>('is_admin')
          .eq('id', session?.user.id)
          .single();

        if (error) throw new Error(error.message);
        setIsAdmin(data);
      } catch (error) {
        console.error(error);
      }
    };

    const fetchSession = async () => {
      try {
        const { data: { session }, error } = await supabase.auth.getSession();
        if (error) throw new Error(error.message);
        setSession(session);
      } catch (error) {
        console.error(error);
      } finally {
        setLoading(false);
      }
    };

    fetchSession();

    const { data: authListener } = supabase.auth.onAuthStateChange(
      (_event, session) => {
        setSession(session);
        if (session) {
          fetchIsAdmin(session);
        }
      }
    );
  });

```

```

);

return () => {
  authListener.subscription.unsubscribe();
};
}, []);

const signOut = async () => {
  await supabase.auth.signOut();
  setSession(null);
};

return (
  <SupabaseSessionContext.Provider value={{ session, loading, isAdmin, signOut }}>
    {loading ? null : children}
  </SupabaseSessionContext.Provider>
);
};

export const useSupabaseSession = () => {
  const context = useContext(SupabaseSessionContext);
  if (context === undefined) {
    throw new Error('useSupabaseSession must be used within a SupabaseSessionProvider');
  }
  return context;
};

```

A.6 Текст файла Store/useTableStore.tsx

```

import { BaseCatalog, PaginationModel } from '../types/beneficiaries';
import { create } from 'zustand';

interface StoreState {
  pagination: PaginationModel;
  setPagination: (page: number, pageSize: number) => void;
  categories: BaseCatalog[];
  setCategories: (data: BaseCatalog[]) => void;
  districts: BaseCatalog[];
  setDistricts: (data: BaseCatalog[]) => void;
  searchTerms: string;
  setSearchTerms: (terms: string) => void;
}

```

```

}

const useTableStore = create<StoreState>((set) => ({
  pagination: { page: 0, pageSize: 10 },
  setPagination: (page, pageSize) => set(() => ({ pagination: { page, pageSize } })),

  categories: [],
  setCategories: (data) => set(() => ({ categories: data })),

  districts: [],
  setDistricts: (data) => set(() => ({ districts: data })),

  searchTerms: "",
  setSearchTerms: (terms) => set(() => ({ searchTerms: terms })),
}));

export default useTableStore;

```

A.7 Текст файлу pages/Home/index.tsx

```

import useTableStore from '../Store/useTableStore';
import CustomDropdown from '../components/Form/CustomDropdown';
import { BENEFICIARY } from '../constants/routes';
import { useBeneficiaries } from '../hooks/useBeneficiaries';
import Topbar from '../layout/Topbar/topbar';
import { BaseCatalog, BeneficiaryModel, PaginationModel, TableBeneficiary } from
'../types/beneficiaries';
import { localizedTextMap } from './localization';
import { Description } from '@mui/icons-material';
import SearchIcon from '@mui/icons-material/Search';
import ClearIcon from '@mui/icons-material/Clear';
import { Alert, Box, Button, InputAdornment, Snackbar, TextField, debounce } from
'@mui/material';
import Paper from '@mui/material/Paper';
import { DataGrid, GridActionsCellItem, GridColDef, GridRowParams } from
'@mui/x-data-grid';
import { useCallback, useEffect, useState } from 'react';
import { useLocation, useNavigate } from 'react-router-dom';

const columns: GridColDef[] = [
  { field: 'last_name', headerName: 'Прізвище', flex: 1 },
  { field: 'first_name', headerName: 'Ім'я', flex: 1 },
  { field: 'middle_name', headerName: 'По батькові', flex: 1 },

```

```

{ field: 'name', headerName: 'Район', type: 'string', flex: 1 },
{ field: 'category_name', headerName: 'Пільгова категорія', type: 'string', flex: 2 },
{
  field: 'deletionMark',
  headerName: "",
  width: 50,
  filterable: false,
  type: 'actions',
  getActions: (params) => [
    <GridActionsCellItem
      icon={<Description color={params.row.is_all_docs ? 'inherit' : 'error'} />}
      label="Наявність документів"
    />,
  ],
},
];

const Home = () => {
  const { fetchBeneficiaries } = useBeneficiaries();
  const [rows, setRows] = useState<TableBeneficiary[]>([]);
  const [rowCount, setRowCount] = useState(0);
  const [loading, setLoading] = useState(false);

  const {
    pagination, setPagination,
    categories, setCategories,
    districts, setDistricts,
    searchTerms, setSearchTerms,
  } = useTableStore();

  const navigate = useNavigate();

  const [inputValue, setInputValue] = useState(searchTerms);
  const debouncedSearchTerms = useCallback(
    debounce((value: string) => {
      setSearchTerms(value);
      setPagination(0, pagination.pageSize);
    }, 500),
    [setSearchTerms, setPagination, pagination.pageSize]
  );

  const handleInputChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    const value = event.target.value;
    setInputValue(value);
    debouncedSearchTerms(value);
  };

```

```

};

useEffect(() => {
  const fetchPageData = async () => {
    try {
      setLoading(true);
      const categoryIds = categories.map((cat) => cat.id);
      const districtIds = districts.map((cat) => cat.id);

      const data = await fetchBeneficiaries(
        pagination,
        searchTerms,
        categoryIds,
        districtIds
      );

      if (data) {
        const transformedData = data.data.map((item) => ({
          ...item,
          category_name: item.categories?.map((cat) => cat.name).join(', '),
        }));
        setRows(transformedData);
        setRowCount(data.count || 0);
      }
    } catch (error) {
      console.error('Помилка завантаження даних:', error);
    } finally {
      setLoading(false);
    }
  };

  fetchPageData();
}, [pagination, searchTerms, categories, districts]);

const handlePaginationChange = (newPaginationModel: PaginationModel) => {
  setPagination(newPaginationModel.page, newPaginationModel.pageSize);
};

const handleRowDoubleClick = (params: GridRowParams<BeneficiaryModel>) => {
  {
    navigate(BENEFICIARY, { state: { id: params.row.id } });
  }
};

return (
  <>

```

```

<Topbar />
<Box className="create-button">
  <Paper sx={{ width: { xs: '100%', md: '80%' }, display: 'flex', py: 1 }}>
    <Button variant="contained" onClick={() => navigate(BENEFICIARY)}>
      СТВОРИТИ
    </Button>
    <TextField
      value={inputValue}
      onChange={handleInputChange}
      placeholder="Пошук за ПІБ"
      size="small"
      slotProps={{
        input: {
          startAdornment: <InputAdornment position="start"><SearchIcon
/></InputAdornment>,
        },
      }}
    />
  </Paper>
</Box>

<Box className="homepage-container">
  <Paper sx={{ width: { xs: '100%', md: '80%' }, height: 629 }}>
    <DataGrid
      rows={rows}
      columns={columns}
      rowCount={rowCount}
      paginationMode="server"
      paginationModel={pagination}
      onPaginationModelChange={handlePaginationChange}
      pageSizeOptions={[10, 25, 50, 100]}
      loading={loading}
      onRowDoubleClick={handleRowDoubleClick}
      localeText={localizedTextMap}
    />
  </Paper>
</Box>
</>
);
};

export default Home;

```

ДОДАТОК Б
Слайди презентації



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
Кафедра програмних засобів

РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБЛІКУ СОЦІАЛЬНО ВРАЗЛИВИХ ГРУП НАСЕЛЕННЯ

Студент гр. КНТ-122:
Толочин А.Г.

Керівник проєкту:
доц., к.т.н. Федорончак Т.В.

Запоріжжя – 2026

Рисунок Б.1 - Слайд №1

ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

ОБ'ЄКТ ДОСЛІДЖЕННЯ:

Процеси розроблення програмного забезпечення для обліку осіб пільгових та соціально вразливих категорій у соціальних службах.

ПРЕДМЕТ ДОСЛІДЖЕННЯ:

Методи побудови клієнт-серверних вебзастосунків на основі BaaS-архітектури та інструменти реляційного моделювання для автоматизації соціальних послуг.

Рисунок Б.2 - Слайд №2

МЕТА ТА ЗАДАЧІ РОБОТИ

МЕТА: підвищення ефективності соціального захисту населення шляхом цифровізації процесів обліку та впровадження системи з автоматизованою верифікацією даних.

ОСНОВНІ ЗАДАЧІ:

- Аналіз існуючих засобів автоматизації та виявлення функціональних недоліків.
- Проектування реляційної моделі даних для підтримки множинних пільгових статусів.
- Розроблення підсистеми розмежування прав доступу на рівні ядра бази даних.
- Експериментальне дослідження швидкодії та надійності розробленого рішення.

Рисунок Б.3 - Слайд №3

ПОРІВНЯЛЬНИЙ АНАЛІЗ ТЕХНІЧНИХ РІШЕНЬ

Критерій аналізу	Trello	Jira Software	Google Sheets	Власна розробка
Архітектура даних	Картки (NoSQL)	Об'єктна	Таблична	Реляційна (3NF)
Сховище документів	Атачменти	Атачменти	Посилання	Storage API
Безпека рядків (RLS)	Відсутня	Часткова	Відсутня	Нативна (SQL)
Тип інтерфейсу	Канбан	Складний MVC	Сітка	Особова картка
Пошук за ПІБ	Повнотекстовий	JQL	Стандартний	Trigram Index

Таблиця 1. Порівняння засобів автоматизації за інженерними критеріями

Рисунок Б.4 - Слайд №4

МАТЕМАТИЧНА МОДЕЛЬ ЕФЕКТИВНОСТІ

$$E = \frac{T_{\text{manual}} - T_{\text{auto}}}{T_{\text{manual}}} \times 100 \%$$

Вхідні дані:

$T_{\text{manual}} = 15 \text{ хв}; T_{\text{auto}} = 4 \text{ хв}$

Результат:

$E = 73.3\%$

Формула 1. Розрахунок показника економії людино-годин

Рисунок Б.5 - Слайд №5

АРХІТЕКТУРНА ПОБУДОВА СИСТЕМИ (BAAS)

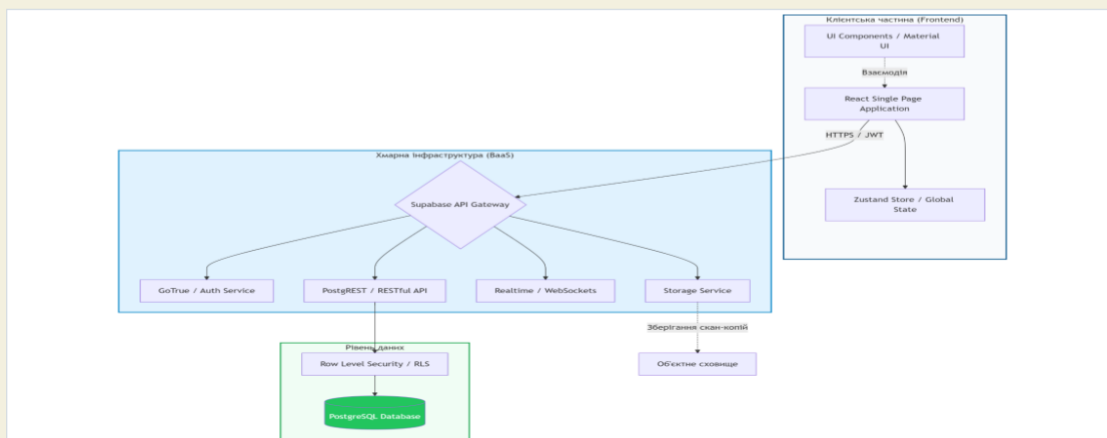


Рисунок 1. Структурна схема взаємодії React SPA та хмарної інфраструктури Supabase

Рисунок Б.6 - Слайд №6

РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ (ERD)

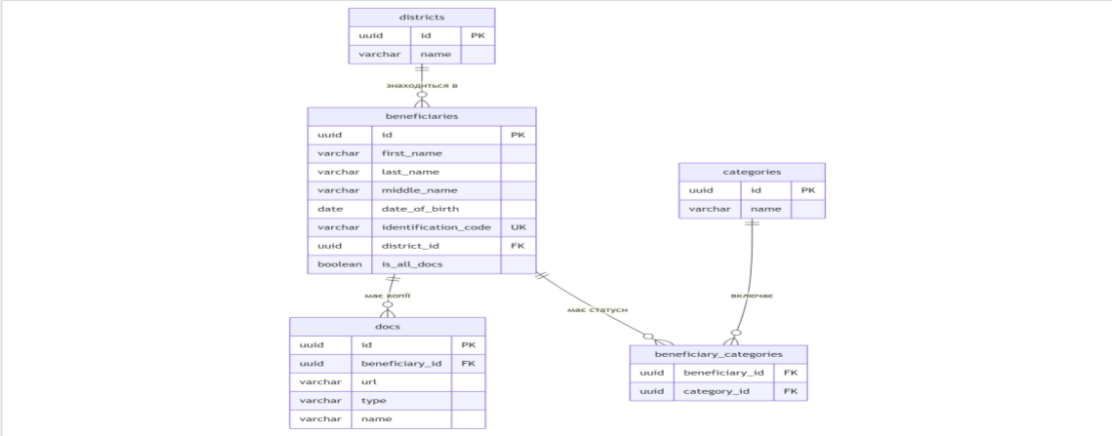


Рисунок 2. Логічна схема бази даних: нормалізація 3NF та зв'язки Many-to-Many

Рисунок Б.7 - Слайд №7

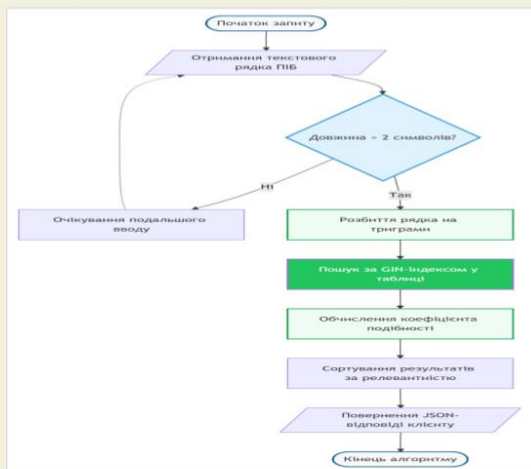
СПЕЦИФІКАЦІЯ АТРИБУТІВ БАЗИ ДАНИХ

Атрибут	Тип (PostgreSQL)	Обмеження (Constraints)	Призначення
id	UUID	PK, DEFAULT uuid_generate_v4()	Унікальний ID
first_name	VARCHAR(100)	NOT NULL	Ім'я особи
last_name	VARCHAR(100)	NOT NULL	Прізвище особи
id_code	VARCHAR(10)	UNIQUE, NOT NULL	ПІН бенефіціара
district_id	UUID	FK (districts.id)	Район проживання
is_all_docs	BOOLEAN	DEFAULT false	Статус документів

Таблиця 2. Технічна специфікація атрибутів таблиці beneficiaries

Рисунок Б.8 - Слайд №8

АЛГОРИТМ НЕЧІТКОГО ПОШУКУ (FUZZY SEARCH)



```

-- 1. Створення інвертованого індексу GIN
CREATE INDEX idx_trgm ON beneficiaries
USING gin (last_name gin_trgm_ops);

-- 2. Запит нечіткого пошуку (поріг подібності > 0.3)
SELECT id, last_name
FROM beneficiaries
WHERE last_name % 'Зубенко'
ORDER BY similarity(last_name, 'Зубенко') DESC;
  
```

Рисунок 3. Блок-схема та програмна реалізація триграмного пошуку

Рисунок Б.9 - Слайд №9

ДІАГРАМА ПОСЛІДОВНОСТІ ОПЕРАЦІЙ

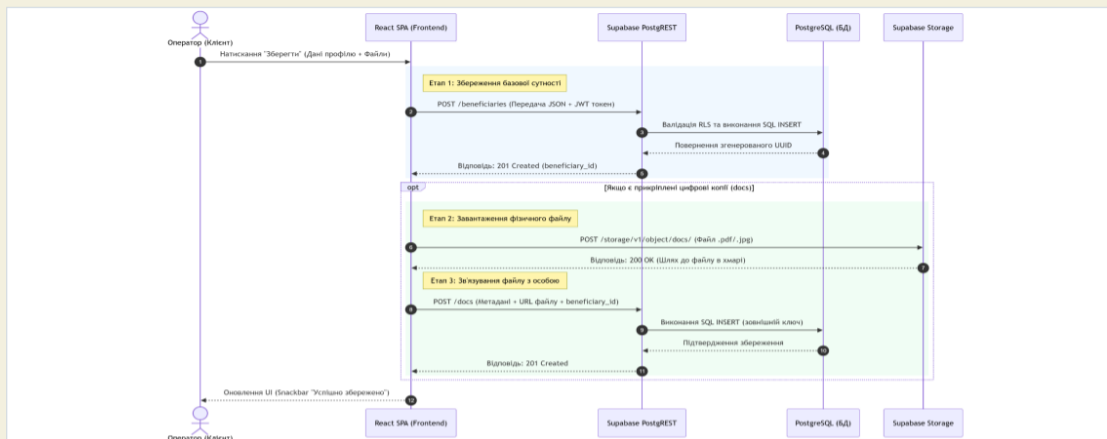


Рисунок 4. UML-діаграма: процес транзакційного збереження профілю та документів

Рисунок Б.10 - Слайд №10

РЕАЛІЗАЦІЯ ПОЛІТИК БЕЗПЕКИ RLS

```
-- Розмежування доступу за районом на рівні СКБД:  
  
ALTER TABLE beneficiaries ENABLE ROW LEVEL SECURITY;  
  
CREATE POLICY "District access only" ON beneficiaries  
FOR SELECT USING (  
  district_id = (  
    SELECT assigned_district  
    FROM auth_profiles  
    WHERE id = auth.uid()  
  )  
);
```

Auth.uid()
Ідентифікатор сесії

JWT
Криптографічний захист

Kernel
Валідація ядром БД

Рисунок 5. Схема логічного захисту персональних даних

Рисунок Б.11 - Слайд №11

ДІАГРАМА СТАНІВ ЕЛЕКТРОННОЇ КАРТКИ

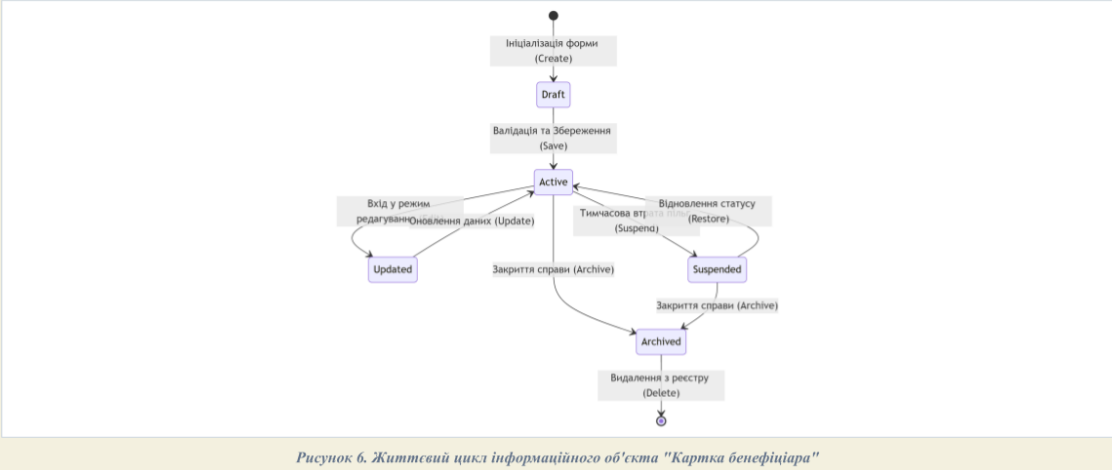


Рисунок 6. Життєвий цикл інформаційного об'єкта "Картка бенефіціара"

Рисунок Б.12 - Слайд №12

ВИМОГИ ДО ТЕХНІЧНИХ ТА ПРОГРАМНИХ ЗАСОБІВ

Характеристика	Робоча станція розробника	Мінімальні вимоги (Production)
Процесор (CPU)	Intel i5+ / Ryzen 5+ (6+ cores)	2 cores, 1.8+ GHz
Пам'ять (RAM)	16 GB DDR4	4 GB DDR4
Накопичувач (Disk)	SSD (NVMe)	SSD / HDD (1 GB free)
Браузер	Chrome 120+, VS Code	Chrome, Firefox, Safari (HTML5 support)
Мережа	100 Mbps (Ethernet)	5 Mbps (Stable connection)

Таблиця 3. Специфікація апаратно-програмного забезпечення

Рисунок Б.13 - Слайд №13

АНАЛІЗ РЕЗУЛЬТАТІВ КОМПЛЕКСНОГО ТЕСТУВАННЯ СИСТЕМИ



Графік 1. Кількісні показники якості та швидкодії за результатами аудиту

Рисунок Б.14 - Слайд №14

ВИСНОВКИ ЗА РЕЗУЛЬТАТАМИ РОБОТИ

- **Розроблено** автоматизовану систему обліку бенефіціарів на основі архітектури SPA та BaaS, що дозволило зменшити навантаження на серверну частину.
- **Побудовано** реляційну модель даних третьої нормальної форми (3NF), що гарантує відсутність аномалій при збереженні множинних статусів.
- **Інтегровано** механізм Row Level Security, який на рівні ядра БД забезпечує розмежування прав доступу між працівниками територіальних підрозділів.
- **Доведено** економічну ефективність рішення: час реєстрації картки скорочено з 15 до 4 хвилин (економія робочого часу спеціаліста складає 73.3%).
- **Система відповідає** стандартам безпеки і доступності та повністю готова до впровадження у соцслужбах.

Рисунок Б.15 - Слайд №15