ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

КУРСОВИЙ ПРОЄКТ:

ПРИНЦИПИ КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

на тему: Розробка інформаційної системи телефонного довідника

здобувач вищої освіти III курсу 343-А групи
спеціальності 121 «Інженерія програмного
забезпечення»
першого (бакалаврського) рівня вищої освіти
Маргер Б.Є.

Оцінка:
за національною шкалою

_____
(словами)

кількість балів

_____
(цифра)

за шкалою ECTS

_____
(літера)

Чернівці, 2025

ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Навчально-науковий інститут фізико-технічних та комп'ютерних наук

Кафедра програмного забезпечення комп'ютерних систем

ЗАВДАННЯ

на курсовий проєкт

здобувачу вищої освіти

першого (бакалаврського) рівня вищої освіти

Маргер Богдану Євгеновичу

1) Тема проєкту. « Розробка інформаційної системи телефонного довідника».
2) Вхідні дані до проєкту: тематика індивідуального завдання.
3) Вихідні дані до проєкту:
   - виокремленні сутності предметної області з набором їх власних атрибутів Subscriber, Address, Street, PhoneNumber, MobileOperator, Debt, SpecialService, RepairWork, PostOffice, NumberChangeRequest, User, RegistrationRequest
   - спроєктована структура сховища даних у форматі ER-моделі;
   - сформовано набір функціональних та не функціональних вимог до системи, побудовано модель взаємодії користувачів з системою (Use Case діаграма);
   - розроблений програмний додаток для взаємодії із СКБД із механізмом доступу до сховища даних;
   - проведено функціональне тестування, яке перевіряє, як програма виконує свої функції в умовах коректних та некоректних дій користувача;
   - розроблена програмна документація;
   - підготовлений відеоролик демонстрації функціональних можливостей програмної системи.

Керівники проєкту _____ Комісарчук В.В.

(підпис керівника)

_____ Дячук Р.Л.

(підпис керівника)

_____ Нікітін Ю.А.

(підпис керівника)

Завдання взяв до виконання _____ Маргер Б.Є.

(підпис студента)

РЕФЕРАТ

Курсовий проєкт налічує 113 сторінок, 1 рисунків, 22 таблиць та 1 джерел.

Об’єктами проєктування є структура бази даних телефонного довідника, функціональні елементи веб-інтерфейсу та архітектура програмної системи для обробки довідкової інформації про абонентів і спеціальні служби.

Метою роботи є розроблення програмної системи для ведення та обробки даних телефонного довідника, що забезпечує виконання запитів, перегляд, пошук, фільтрацію та відображення даних.

Для забезпечення компактного зберігання даних була обрана вбудована реляційна СКБД SQLite. Розроблено програмний веб-додаток засобами мови програмування Python із використанням фреймворку Flask, який забезпечує користувачам доступ до даних через зручний веб-інтерфейс. Реалізовано функціонал виконання SQL-запитів, відображення результатів, введення параметрів для фільтрації, а також обробку структурованої інформації про абонентів, адреси, ремонтні роботи, спеціальні служби та пов’язані сутності.

Створена програмна система дозволяє підвищити швидкість доступу до довідкової інформації, покращити точність обліку даних та автоматизувати основні операції роботи з телефонним довідником.

Програмний продукт може бути використано в довідкових службах, телекомунікаційних підприємствах, контакт-центрах та інших організаціях, що працюють з телефонними базами даних.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНА СИСТЕМА, ТЕЛЕФОННИЙ ДОВІДНИК, CRUD-ОПЕРАЦІЇ, FLASK, SQLITE, PYTHON, BOOTSTRAP, СЕРВІСИ, WEB-ІНТЕРФЕЙС.

SUMMARY

The course project comprises 113 pages, 1 figures, 22 tables and 1 references.

The objects of design are the structure of the telephone directory database, functional elements of the web interface and the architecture of the software system for processing reference information about subscribers and special services.

The purpose of the work is to develop a software system for maintaining and processing telephone directory data, which provides query execution, viewing, searching, filtering and displaying data.

To ensure compact data storage, the built-in relational SQLite DBMS was selected. A software web application was developed using the Python programming language using the Flask framework, which provides users with access to data through a convenient web interface. The functionality of executing SQL queries, displaying results, entering parameters for filtering, as well as processing structured information about subscribers, addresses, repair work, special services and related entities was implemented.

The created software system allows you to increase the speed of access to reference information, improve the accuracy of data accounting and automate the main operations of working with the telephone directory.

The software product can be used in reference services, telecommunications companies, contact centers and other organizations working with telephone databases.

KEYWORDS: INFORMATION SYSTEM, PHONE INFORMER, CRUD OPERATIONS, FLASK, SQLITE, PYTHON, BOOTSTRAP, SERVICES, WEB GUI.

ЗМІСТ

ВСТУП

У сучасних умовах цифровізації важливо забезпечити зручний та швидкий доступ до довідкової інформації, що використовується у телекомунікаційній сфері. Особливо це стосується телефонних довідників, які містять відомості про абонентів, адреси, спеціальні служби та інші інфраструктурні дані. Ручне ведення такої інформації або її зберігання у неструктурованому вигляді призводить до втрати актуальності, ускладнює пошук та оновлення записів.

Тому виникає потреба у створенні веб-орієнтованої інформаційної системи, яка дозволить централізовано зберігати, редагувати, переглядати та обробляти телефонні дані. Автоматизація цих процесів підвищує точність обліку, зменшує кількість помилок і пришвидшує доступ до необхідної інформації.

У межах даної роботи розроблено веб-застосунок телефонного довідника, створений на основі мови програмування Python та фреймворку Flask, з використанням реляційної СКБД SQLite для зберігання даних. Система підтримує виконання CRUD-операцій, роботу зі структурованими сутностями, а також реалізує дев’ять вбудованих SQL-запитів різних типів, включаючи параметризовані.

Особливу увагу приділено проєктуванню структури бази даних, побудові архітектури веб-додатку та створенню інтерфейсу на основі HTML, CSS, Bootstrap і JavaScript, що забезпечує зручність взаємодії користувача із системою.

Система може застосовуватися у довідкових службах, телекомунікаційних організаціях та інших установах, де потрібна автоматизація роботи з телефонною інформацією.

РОЗДІЛ 1. СПЕЦИФІКАЦІЯ ПРОЄКТУ

1.1 Постановка завдання

Метою даного проєкту є розроблення веб-орієнтованої інформаційної системи телефонного довідника, призначеної для централізованого обліку абонентів, їхніх адрес, телефонних номерів, мобільних операторів та супутніх службових даних. Система має забезпечувати упорядковане зберігання інформації, підтримувати декілька рівнів доступу користувачів та надавати інструменти для виконання службових операцій, пошуку й формування аналітичних вибірок.

Предметна область охоплює процеси ведення абонентської бази, обліку телефонних номерів, обробки заявок на їх зміну, фіксації заборгованостей, реєстрації ремонтних робіт, а також підтримки довідкових даних про спеціальні служби та поштові відділення. Робота системи орієнтована на забезпечення оперативного доступу до даних, підвищення точності обліку та спрощення взаємодії між користувачами з різними ролями.

До основних завдань системи належать:

- зберігання, оновлення та структуроване представлення даних про абонентів і пов’язані з ними сутності;
- підтримка CRUD-операцій для всіх базових об’єктів предметної області;
- здійснення пошуку та фільтрації інформації за різними критеріями;
- формування вбудованих та довільних SQL-запитів;
- забезпечення роботи з даними через веб-інтерфейс, розроблений із використанням Flask та адаптивних клієнтських технологій.

Система повинна підтримувати чотири типи користувачів: гостей, авторизованих користувачів, операторів та адміністраторів, що забезпечує розмежування доступу та виконання службових функцій відповідно до ролі

1.2 Вимоги до програмного забезпечення

1.2.1 Функціональні вимоги

1) Управління користувачами адміністратором

- Перегляд усіх користувачів системи.
- Керування користувачами
- Зміна ролей користувачів.
- Обробка заявок на реєстрацію: прийняття, відхилення.

2) Аутентифікація та авторизація користувачів

- Вхід у систему через логін/пароль.
- Валідація паролю через хешування.
- Надання доступу відповідно до ролі (guest/user/operator/admin).

3) Управління абонентами довідника

- Додавання нового абонента
- Редагування персональних даних.
- Перегляд усіх абонентів з адресою та основним номером.
- Перегляд профілю абонента з телефонами, боргами та поштовим відділенням.

4) Робота з адресами та вулицями

- Створення, оновлення та пошук вулиць.
- Додавання та редагування адрес (будинок, квартира).
- Автоматичне створення або оновлення поштового відділення.

5) Телефонні номери та мобільні оператори

- Додавання номерів різних типів (mobile/home/service).
- Прив’язка номера до абонента та оператора.
- Видалення номерів.
- Перегляд усіх операторів.

6) Управління заборгованостями

- Додавання боргів абоненту.
- Редагування або видалення боргу.
- Перегляд списку боржників.
- Пошук боржників за маскою ПІБ.

7) Управління ремонтними роботами

- Реєстрація нових ремонтів із прив’язкою до адреси.
- Редагування інформації про ремонт.
- Видалення ремонту.
- Пошук ремонтів за адресою.

8) Заявки на зміну номера

- Створення заявки користувачем.
- Схвалення/відхилення заявки оператором або адміністратором.
- Автоматична заміна номера після схвалення.

9) Спеціальні служби

- Перегляд довідника служб.
- Показ телефону служби та графіка роботи.

10) Службові SQL-запити

- Виконання заздалегідь визначених запитів до БД;
- Виконання довільних SQL-запитів в “консолі”.
- Вивід результатів у табличному вигляді.
- Параметризовані запити з формами введення.

11) Пошук і фільтрація

- Пошук абонентів за ПІБ, номером, адресою.
- Пошук боргів за прізвищем.
- Пошук ремонтів за вулицею/будинком/квартирою.
- Масковий пошук (*, ?).

12) Інтерфейс та повідомлення

- графічний веб-інтерфейс українською мовою;
- відображення інформаційних і попереджувальних повідомлень (flash);
- чіткі інструкції при взаємодії з формами.

1.2.2 Нефункціональні вимоги

1) Продуктивність

- час реакції системи на базові операції не повинен перевищувати 1 секунди;
- виконання SQL-запитів повинно здійснюватися без значних затримок.

2) Масштабованість

- архітектура має дозволяти розширення функціоналу;
- можлива заміна SQLite на іншу СУБД у майбутньому.

3) Зручність користування

- інтерфейс українською мовою;
- використання Bootstrap 5 для адаптивного дизайну;
- табличне представлення інформації;
- інтерфейс має бути інтуїтивно зрозумілим.

4) Безпека

- хешування паролів користувачів;
- захист від SQL-ін’єкцій за рахунок параметризованих запитів;
- перевірка коректності введених даних;
- розмежування прав доступу відповідно до ролей.

5) Надійність

- використання механізму винятків для обробки помилок;
- уникнення дублювання даних;
- гарантія незмінності даних при некоректних діях користувача.

6) Вимоги до коду

- дотримання стилю PEP 8;
- логічна структуризація коду (app.py – маршрути, service.py – логіка);
- використання змістовних ідентифікаторів;
- розміщення HTML-шаблонів у окремій директорії.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СХОВИЩА ДАНИХ

2.1 Аналіз предметної області (визначення сутностей та їх атрибутів)

Опис сутностей наведено в табл. 2.1.

Таблиця 2.1 – Опис сутностей

№	Сутність	Короткий опис
1	Subscriber (Абонент)	Представляє фізичну особу, зареєстровану в телефонному довіднику. Містить персональні дані та прив’язку до адреси та поштового відділення
2	Address (Адреса)	Містить інформацію про будинок, квартиру та пов’язану вулицю. Є прив’язкою для абонентів та ремонтних робіт.
3	Street (Вулиця)	Довідник назв вулиць із зазначенням їх типу (вул., просп., пл.). Усуває дублювання даних в адресах.
4	PostOffice (Поштове відділення)	Містить номер поштового відділення, що відповідає конкретній адресі
5	PhoneNumber (Телефонний номер)	Представляє телефонні номери абонента різних типів (мобільний, домашній, службовий).
6	MobileOperator (Мобільний оператор)	Містить назву оператора зв’язку та префікс, який використовують номери.

7	Debt (Заборгованість)	Містить дані про борги абонентів: суму, дату виникнення та крайній термін погашення.
8	RepairWork (Ремонтні роботи)	Представляє записи про ремонтні роботи, прив’язані до адрес.
9	SpecialService (Спеціальна служба)	Містить назву служби, телефон та графік роботи.
10	NumberChangeRequest (Заявка на зміну номера)	Містить старий і новий номера, дату звернення та абонента, що подав заявку.
11	User (Користувач системи)	Адміністратори, оператори, звичайні користувачі та гості із відповідними ролями доступу.
12	RegistrationRequest (Заявка на реєстрацію)	Запити користувачів на створення облікового запису, які адміністратор може схвалити або відхилити.

Перелік атрибутів сутностей та їх опис наведено у табл. 2.2.

Таблиця 2.2 – Перелік атрибутів сутностей

№	**Об’єкт (сутність)**	**№**	**Атрибут**	**Опис**
1	Subscriber (Абонент)	1	id	Унікальний ідентифікатор абонента,
		2	lastname	Прізвище абонента.
		3	firstname	Ім’я абонента.
		4	middlename	По-батькові абонента.
		5	id_address	Посилання на адресу проживання (FK → Address).
		6	id_post_office	Посилання на поштове відділення (FK → PostOffice),

2	Street (Вулиця)	1	id	Ідентифікатор вулиці
		2	name	Назва вулиці
		3	type	Тип (вул. просп. пл.)
3	Address (Адреса)	1	id	Ідентифікатор адреси
		2	id_street	Посилання на вулицю
		3	building	Номер будинку
		4	apartment	Номер квартири
4	PostOffice (Поштове відділення)	1	id (PK)	Ідентифікатор
		2	office_number	Номер поштового відділення
		3	id_address	Адреса відділення
5	PhoneNumber (Номера телефону)	1	id (PK)	Ідентифікатор
		2	number	Номер телефону
		3	type	Тип номера (mobile/home/service)
		4	id_subscriber	Абонент
		5	id_operator	Мобільний оператор
		6	active	Статус активності
6	MobileOperator (Оператори телефонів)	1	id (PK)	Ідентифікатор
		2	name	Назва оператора
		3	prefix	Код оператора (050, 097)
7	Debt (борг)	1	id (PK)	Ідентифікатор боргу
		2	id_subscriber	Абонент
		3	amount	Сумма
		4	date_start	Дата виникнення
		5	deadline	Термін погашення
		6	status	active / paid
8	RepairWork (Ремонті роботи)	1	id	Ідентифікатор
		2	id_address	Адреса ремонту

		3	date_start	Дата початку
		4	date_end	Дата завершення
		5	description	Опис роботи
9	SpecialService (Спеціальні служби)	1	id	Ідентифікатор
		2	name	Назва служби
		3	weekday	Дні роботи
		4	time_start	Початок роботи
		5	time_end	Кінець роботи
		6	id_number	Телефон служби
10	NumberChangeRequest (Зміна номеру)	1	id	Ідентифікатор
		2	id_subscriber	Абонент
		3	old_number	Поточний номер
		4	new_number	Запропонований номер
		5	data_request	Дата подання
11	User (Користувачі)	1	id	Ідентифікатор юзера
		2	login	Логін
		3	password	Хеш пароля
		4	role	guest/user/operator/admin
12	RegistrationRequest (Запит реєстрації)	1	id	Ідентифікатор
		2	login	Логін
		3	password	Хеш пароля
		4	status	Статус запроса

2.2 Фізична модель сховища даних

У цьому підрозділі наведено конкретне відображення сутностей із розділу 2.1 із зазначенням полів і типів, опис зв’язків між документами та коротку перевірку на нормалізацію (3НФ) для реляційного варіанта.

2.2.1 Перелік таблиць / документів / ключів-значень / вершин (вузлів) / колонок (атрибут / поле даних) з визначенням їх полів/записів і типів даних

Перелік наведено у табл. 2.2.1.

Таблиця 2.2.1 – Перелік атрибутів сутностей

№	Сутність	Поле	Тип	Ключ
1	Subscriber	id	INTEGER	PK
		lastname	TEXT	
		firstname	TEXT	
		middlename	TEXT	
		id_address	INTEGER	FK
		id_post_office	INTEGER	FK
2	Street	id	INTEGER	PK
		name	TEXT	
		type	TEXT	
3	Address	id	INTEGER	PK
		id_street	INTEGER	FK
		building	TEXT	
		apartament	TEXT	
4	User	id	INTEGER	PK
		login	TEXT	
		password	TEXT	

		role	TEXT	
5	RegistrationRequest	id	INTEGER	PK
		login	TEXT	
		password	TEXT	
		status	TEXT	
6	PostOffice	id	INTEGER	PK
		office_number	INTEGER	
		id_address	INTEGER	FK
7	MobileOperator	id	INTEGER	PK
		name	TEXT	
		prefix	TEXT	
8	PhoneNumber	id	INTEGER	PK
		number	TEXT	
		type	TEXT	
		id_subscriber	INTEGER	FK
		id_operator	INTEGER	FK
		active	INTEGER	
9	NumberChangeRequest	id	INTEGER	PK
		id_subscriber	INTEGER	FK
		old_number	TEXT	
		new_number	TEXT	
		data_request	TEXT	
10	Debt	id	INTEGER	PK
		id_subscriber	INTEGER	FK
		amount	REAL	
		date_start	TEXT	
		deadline	TEXT	
		status	TEXT	

11	RepairWork	id	INTEGER	PK
		id_address	INTEGER	FK
		date_start	TEXT	
		date_end	TEXT	
		description	TEXT	
12	SpecialService	id	INTEGER	PK
		name	TEXT	
		weekday	TEXT	
		time_start	TEXT	
		time_end	TEXT	
		id_number	INTEGER	FK

2.2.2 Перелік зв'язків між сутностями

Перелік зв’язків наведено у табл. 2.2.2.

Таблиця 2.2.2 – Перелік зв’язків сутностей

Сутність A	**Сутність B**	**Тип зв’язку**	**Опис**
Street	Address	1:M	Одна вулиця має багато адрес
Address	Subscriber	1:M	За однією адресою можуть проживати кілька абонентів
Address	RepairWork	1:M	На одну адресу може бути кілька ремонтів
Address	PostOffice	1:1	Одне поштове відділення відповідає одній адресі

Subscriber	PhoneNumber	1:M	Абонент може мати кілька номерів
MobileOperator	PhoneNumber	1:M	Оператор може мати багато номерів
Subscriber	Debt	1:M	Абонент може мати кілька боргів

2.2.3 Нормалізація даних до 3НФ

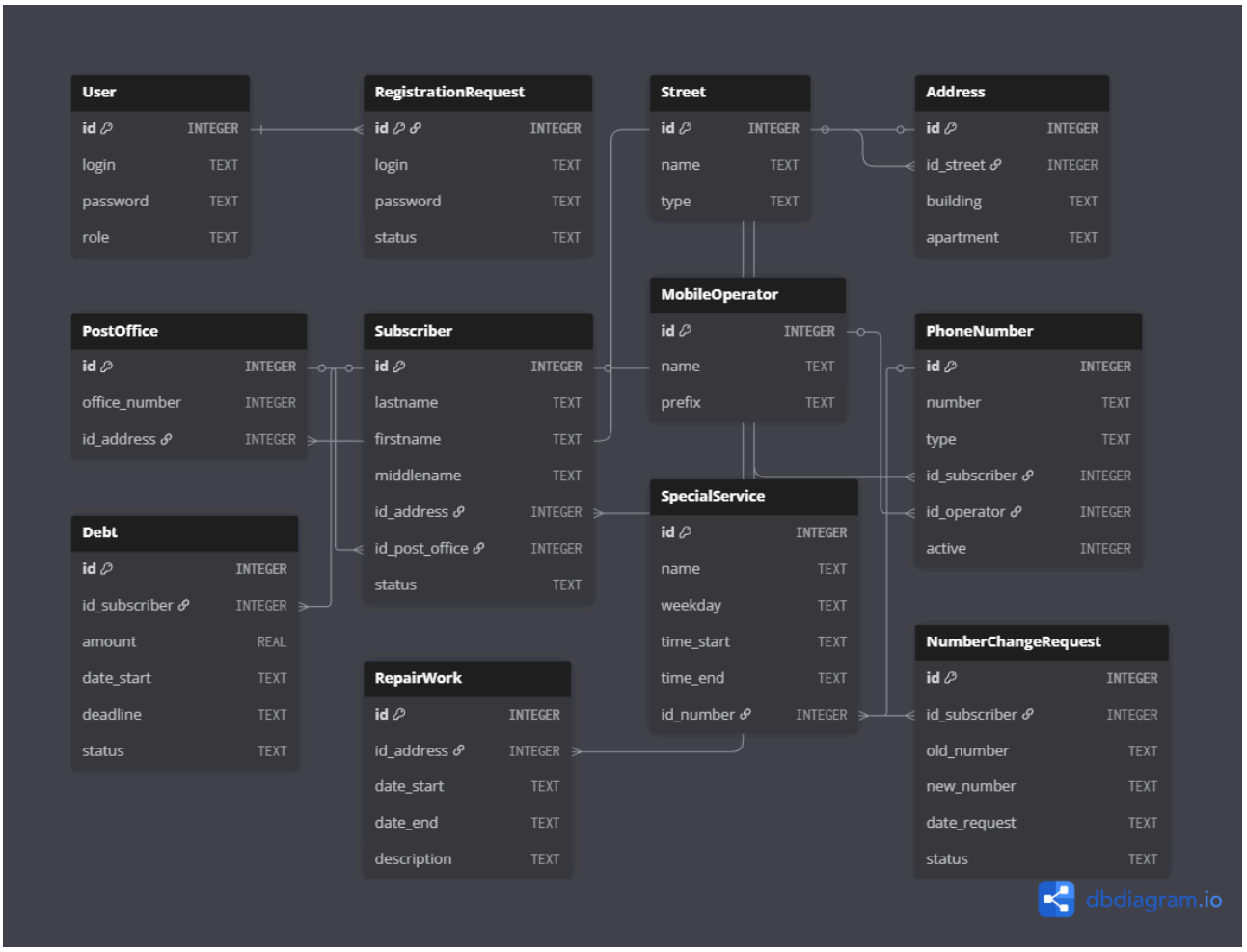
Наші дані вже відповідають третій нормальній формі:

- 1НФ:
  – усі атрибути таблиць атомарні;
  – усі записи унікальні;
  – відсутні повторювані групи.
- 2НФ:
  – кожен неключовий атрибут залежить від усього первинного ключа;
  – складних первинних ключів немає, тому часткові залежності відсутні.
- **3НФ:**
  – відсутні транзитивні залежності між не ключовими атрибутами;
  – усі довідникові дані (вулиці, адреси, оператори, поштові відділення) винесені в окремі таблиці.

2.3. Моделювання структури сховища даних (ER-діаграма)

ER-діаграма наведена на рисунку 1

Рисунок 1 – ER діаграма



РОЗДІЛ 3. ПРОГРАМНА ДОКУМЕНТАЦІЯ

3.1 Документування вимог до програмної системи

У цьому підрозділі наведемо вимоги користувачів у трьох форматах: User Stories, Use Case-діаграми та детальні сценарії Use Case.

3.1.1 Формат User Stories

US1 — Подання заявки на реєстрацію

ЯК Гість,
Я ХОЧУ подати заявку на створення облікового запису,
ЩОБ отримати доступ до закритого функціоналу системи.

Description

Гість може створити заявку на реєстрацію.

Адміністратор її переглядає та підтверджує або відхиляє.

Acceptance Criteria

1. Є форма подання заявки (логін, пароль).
2. Перевірка унікальності логіна.
3. Після подання заявки — повідомлення про відправку.
4. Адміністратор бачить усі заявки.
5. Адміністратор може схвалити або відхилити заявку.

US2 — Авторизація

ЯК Зареєстрований користувач,
Я ХОЧУ увійти до системи,
ЩОБ отримати доступ до функцій згідно з моєю роллю.

Description

Авторизація виконується через логін і пароль, зберігається у сесії.

Acceptance Criteria

1. Є форма авторизації.
2. Пароль перевіряється через хеш.
3. Успіх = перехід на головну сторінку.
4. Невірні дані = помилка.
5. Є можливість вийти з системи.

US3 — Перегляд абонентів

ЯК Користувач (будь-яка роль),
Я ХОЧУ переглянути список абонентів,
ЩОБ отримати довідкову інформацію.

Description

Усі ролі можуть переглядати список абонентів та інформацію про них.

Acceptance Criteria

1. Сторінка списку абонентів доступна всім.

2. Відображаються ПІБ, адреса, пошт.відділ, основний номер.
3. Перехід до детальної інформації за кліком.
4. Якщо абонент не знайдений - вивід повідомлення.

US4 — Пошук із підтримкою масок

ЯК Користувач (авторизований),
Я ХОЧУ виконувати пошук за ПІБ, номером або адресою,
ЩОБ швидко знаходити потрібні записи.

Description

Пошук підтримує символи * (будь-яка кількість) та ? (один символ).

Acceptance Criteria

1. Поле пошуку приймає текстові маски.
2. Пошук виконується.
3. Результати відображаються у табличному вигляді.
4. Порожній запит повертає попередження.

US5 — Додавання абонента

ЯК Оператор/Адміністратор,
Я ХОЧУ додати нового абонента,
ЩОБ підтримувати актуальність довідника.

Description

Створення нового абонента разом із адресою та поштовим відділенням.

Acceptance Criteria

1. Є форма додавання ПІБ.
2. Є форма адреси.
3. Якщо вулиці немає — вона створюється.
4. Адреса додається автоматично.
5. Після успіху — повідомлення.

US6 — Редагування абонента

ЯК Оператор/Адміністратор,
Я ХОЧУ оновити дані абонента,
ЩОБ підтримувати правильність інформації.

Description

Оновлення ПІБ, адреси, поштового відділення.

Acceptance Criteria

1. Є форма редагування.
2. Оновлюються всі поля ПІБ.
3. Можна змінити адресу.
4. Можна створити нове поштове відділення.
5. Після успіху - повідомлення.

US7 — Видалення абонента

ЯК Оператор/Адміністратор,
Я ХОЧУ мати можливість видалити абонента,
ЩОБ очистити систему від неактуальних даних.

Description

Видалення застарілих абонентів

Acceptance Criteria

1. Кнопка видалення видно лише адміну.
2. При видаленні перевіряються залежні дані.
3. Після видалення - повідомлення.
4. Неможливо видалити неіснуючий запис.

US8 — Управління телефонами абонента

ЯК Оператор/Адміністратор,
Я ХОЧУ додавати та видаляти телефонні номери,
ЩОБ актуалізувати контактну інформацію.

Description

Зміна номерів абонента

Acceptance Criteria

1. Можна додати новий номер.
2. Можна обрати тип номера.
3. Можна вибрати мобільного оператора.
4. Видалення номера працює.

US9 — Заявки на зміну телефону

ЯК Користувач,
Я ХОЧУ подати заявку на зміну номера,
ЩОБ оновити свій телефон.

Description

Зміна застарілого номеру телефону

Acceptance Criteria

1. Форма заявки доступна користувачу.
2. Потрібно ввести старий та новий номер.
3. Оператор може схвалити або відхилити.
4. При схваленні старий номер видаляється.
5. Новий номер додається.

US10 — Управління боргами

ЯК Оператор/Адміністратор,
Я ХОЧУ додавати або видаляти борги,
ЩОБ вести коректний облік заборгованостей.

Description

Оператор/Адміністратор може додавати або видаляти борги

Acceptance Criteria

1. Є форма додавання боргу.
2. Вказується сума та дати.
3. Борг можна видалити.
4. Активні борги відображаються у списку.

US11 — Управління ремонтними роботами

ЯК Оператор/Адміністратор,
Я ХОЧУ реєструвати ремонтні роботи на адресах,
ЩОБ відображати планові або аварійні ремонти.

Description

Оператор/Адміністратор може вносити зміну про ремонтні роботи

Acceptance Criteria

1. Є форма створення ремонту
2. Можна редагувати дані ремонту.
3. Є можливість видалити ремонт.
4. Ремонти прив’язуються до адреси.

US12 — Перегляд спеціальних служб

ЯК Будь-який користувач,
Я ХОЧУ переглядати список спецслужб,
ЩОБ отримати довідкові контакти.

Acceptance Criteria

1. Список доступний усім ролям.
2. Є назва служби та телефон.
3. Відображаються години роботи.

US13 — Перегляд SQL-звітів

ЯК Авторизований користувач,
Я ХОЧУ відкривати та формувати готові SQL-звіти,
ЩОБ швидко отримувати потрібну інформацію.

Description

Перегляд результатів 10 завдань з теми

Acceptance Criteria

1. Список звітів доступний user/operator/admin.
2. Є 10 стандартних звітів.
3. Деякі звіти містять вхідні параметри.
4. Результати показуються у таблиці.
5. Помилки обробляються.

US14 — Виконання довільних SQL-запитів

ЯК Оператор/Адміністратор,
Я ХОЧУ виконувати довільні SQL-запити,
ЩОБ мати повний доступ до даних.

Description

Оператор/Адміністратор можуть вручну редагувати нашу БД

Acceptance Criteria

1. Доступ лише для Оператор/Адміністратор.
2. Підтримуються SELECT, UPDATE, DELETE.
3. Результат виводиться у таблиці.
4. Некоректний SQL = повідомлення про помилку.

US15 — Управління користувачами

ЯК Адміністратор,

Я ХОЧУ створювати, редагувати та видаляти користувачів,

ЩОБ контролювати доступ до системи.

Description

Адмін має повний доступ до всього

Acceptance Criteria

1. Є список усіх користувачів.
2. Є функція створення нового користувача.
3. Є функція зміни ролі.
4. Можна видалити користувача.
5. Будь-які зміни супроводжуються повідомленням.

3.1.2 Формат Use Case-діаграми

Use Case-діаграма відображає акторів і прецеденти (Use Cases).

Для нашої системи це:

1. Актори

    1) Гість (guest)

    2) Авторизований користувач (user)

    3) Оператор (operator)

    4) Адміністратор (admin)

2. Прецеденти:

    1) Гість:

        1. Перегляд абонентів
        2. Перегляд боржників
        3. Перегляд спеціальних служб
        4. Перегляд ремонтних робіт
        5. Увійти в систему
        6. Зареєструватися в системі

    2) Авторизований користувач

        7. Все, що вище
        8. Вийти з системи
        9. Пошук абонентів/ремонтів/боржників
        10. Подати заявку на зміну номера
        11. Переглядати SQL-звіти (10 завдань з теми)

    3) Оператор бази даних

        12. Все, що вище

13. Редагувати абонента
14. Створити абонента
15. Додавати телефонні номери
16. Видаляти телефонні номери
17. Додавати борги
18. Видаляти борги
19. Переглядати борги
20. Додавати ремонтні роботи
21. Редагувати ремонтні роботи
22. Видаляти ремонтні роботи
23. Розглядати заявки на зміну номера (схвалення / відхилення)
24. Виконувати довільні SQL-запити

4) Адміністратор

25. Все, що вище
26. Переглядати заявки на реєстрацію
27. Схвалювати заявки на реєстрацію
28. Відхиляти заявки на реєстрацію
29. Створювати користувача
30. Видаляти користувача
31. Змінювати роль користувача
32. Повний доступ до всіх CRUD-операцій

3.1.3. Формат Use Case сценарії

Use Case сценарії наведені в таблицях нижче

Таблиця 3.1 – UC1

Елемент UC	Опис
Ім’я	User Login (Авторизація користувача)
Пріоритет	Must
Область дії	Telephone Directory IS
Контекст	Анонімний користувач вводить логін та пароль для доступу до системи
Діюча особа (Actor)	Гість (guest)
Мета	Отримати доступ до функціонал.
Передумови	Існує обліковий запис користувача
Тригер	Натискання кнопки «Увійти»
Результат (післяумова)	Створена авторизована сесія
Основний потік	1) Відкрити /login 2) Ввести логін і пароль 3) Натиснути «Увійти» 4) Система перевіряє логін 5) Система перевіряє пароль 6) Створює сесію та перенаправляє на головну
Альтернативні потоки / Розширення	4a) Логін не знайдено 5a) Пароль невірний
Flows Error (потоки помилок)	E1) Немає доступу до БД, повідомлення про помилку сервера.
Бізнес-правила	BR1) Паролі захешовані BR2) Guest не має доступу до закритих функцій.

Таблиця 3.2 – UC2

Елемент UC	Опис
Ім’я	Manage Subscriber (Керувати абонентами)
Пріоритет	Must
Область дії	Telephone Directory IS
Контекст	Робота з реєстром абонентів: створення, редагування
Діюча особа (Actor)	Operator, Admin
Мета	Підтримувати актуальні відомості про абонентів
Передумови	Авторизація оператора/адміна
Тригер	Натискання «Додати» або «Редагувати»
Результат (післяумова)	Зміни збережені в БД
Основний потік	1) Відкрити /subscribers 2) Обрати дію 3) Заповнити форму ПІБ та адресу 4) Натиснути «Зберегти» 5) Система зберігає Street, Address, Subscriber
Альтернативні потоки / Розширення	3a) Адреса не вказана, створення без адреси 4a) Порожні обов’язкові поля
Flows Error (потоки помилок)	E1) SQL-помилка при збереженні.
Бізнес-правила	BR1) Вулиці не дублюються BR2) Неможливо редагувати видаленого абонента.

Таблиця 3.3 – UC2

Елемент UC	Опис
Ім’я	Process Number Change Request
Пріоритет	Should
Область дії	Telephone Directory IS
Контекст	Користувач подає заявку, оператор/адмін її розглядає
Діюча особа (Actor)	User, Operator, Admin
Мета	Оновити номер телефона абонента
Передумови	Авторизований User
Тригер	Натискання «Подати заявку»
Результат (післяумова)	Старий номер видалено, новий додано
Основний потік	1) User обирає абонента 2) Вводить новий номер 3) Система створює заявку 4) Operator переглядає заявки 5) Схвалює 6) Система змінює номер
Альтернативні потоки / Розширення	2a) Новий номер порожній 5a) Відхилення заявки
Flows Error (потоки помилок)	E1) Номер уже використовується
Бізнес-правила	BR1) Лише одна активна заявка на користувача

Таблиця 3.4 – UC4

Елемент UC	Опис
Ім’я	View Subscriber Details
Пріоритет	Must
Область дії	Telephone Directory IS
Контекст	Користувач переглядає повну інформацію про абонента.
Діюча особа (Actor)	Operator, Admin
Мета	Отримати детальну інформацію про абонента
Передумови	Абонент існує в системі
Тригер	Натискання на кнопку в списку абонентів
Результат (післяумова)	Відображено номер(-и), адресу, борги, заявки
Основний потік	1) Обрати абонента 2) Система завантажує Address, PhoneNumber, Debt 3) Виводить сторінку з повною інформацією.
Альтернативні потоки / Розширення	1a) Абонент не знайдений
Flows Error (потоки помилок)	E1) Немає зв’язку з БД
Бізнес-правила	BR1) Guest/User бачать лише загальну інформацію, Operator/Admin повну

Таблиця 3.5 – UC5

Елемент UC	Опис
Ім’я	Manage Phone Numbers
Пріоритет	Must
Область дії	Telephone Directory IS
Контекст	Оператор додає або видаляє номери абонентів
Діюча особа (Actor)	Operator, Admin
Мета	Підтримувати актуальні списки номерів
Передумови	Абонент існує
Тригер	Натискання «Додати номер» / «Видалити».
Результат (післяумова)	Оновлений список номерів
Основний потік	1) Обрати абонента 2) Додати або видалити номер 3) Система оновлює PhoneNumber.
Альтернативні потоки / Розширення	2a) Номер порожній 2b) Номер дублюється
Flows Error (потоки помилок)	E1) Помилка збереження у БД
Бізнес-правила	BR1) Може бути кілька активних номерів

Таблиця 3.6 – UC6

Елемент UC	Опис
Ім’я	Manage Debts
Пріоритет	Should
Область дії	Telephone Directory IS
Контекст	Оператор керує боргами абонентів
Діюча особа (Actor)	Operator, Admin
Мета	Підтримувати актуальну інформацію про заборгованості
Передумови	Абонент існує
Тригер	Натискання картки абонента
Результат (післяумова)	Інформація про борг оновлена
Основний потік	1) Відкрити картку абонента 2) Додати борг 3) Вказати суму/дати 4) Зберегти.
Альтернативні потоки / Розширення	3a) Сума = 0
Flows Error (потоки помилок)	E1) Помилка SQL
Бізнес-правила	BR1) Тільки оператор або адмін можуть редагувати борги

Таблиця 3.7 – UC7

Елемент UC	Опис
Ім’я	Manage Repairs
Пріоритет	Should
Область дії	Telephone Directory IS
Контекст	Оператор керує списком ремонтів
Діюча особа (Actor)	Operator, Admin
Мета	Вести список ремонтів у місті
Передумови	Якщо редагування, то має існувати
Тригер	Відкриття вкладки ремонтів
Результат (післяумова)	Збереження даних
Основний потік	1) Відкрити /repairs 2) Додати/редагувати/видалити ремонт.
Альтернативні потоки / Розширення	2a) Відсутня адреса
Flows Error (потоки помилок)	E1) Невірна дата
Бізнес-правила	BR1) Ремонт має містити опис

Таблиця 3.8 – UC8

Елемент UC	Опис
Ім’я	View Special Services
Пріоритет	Must
Область дії	Telephone Directory IS
Контекст	Користувач відкриває сторінку спеціальних служб, щоб переглянути їхній перелік та контактні номери
Діюча особа (Actor)	Guest, User, Operator, Admin
Мета	Переглянути список служб та їх номери
Передумови	В БД є інфа спецслужби
Тригер	Натискання пункту меню «Спецслужби» або відкриття сторінки /services
Результат (післяумова)	Відображається список з усіма спеціальними службами.
Основний потік	1) Відкрити /services 2) Система показує перелік служб
Альтернативні потоки / Розширення	1a) Немає даних то відображається пустий список
Flows Error (потоки помилок)	E1) База недоступна
Бізнес-правила	Всі ролі можуть переглядати список спецслужб. CRUD доступний лише для Admin/Operator

Таблиця 3.9 – UC9

Елемент UC	Опис
Ім’я	Process Registration Requests
Пріоритет	Must
Область дії	Telephone Directory IS
Контекст	Гість подає заявку на реєстрацію; адміністратор розглядає її
Діюча особа (Actor)	Admin
Мета	Дозволити адміністратору створювати нових користувачів
Передумови	У таблиці RegistrationRequest є хоча б одна нова заявка
Тригер	Адміністратор переходить до сторінки /admin/requests
Результат (післяумова)	Додавання нового користувача
Основний потік	1) Admin відкриває /admin/requests 2) Обирає потрібну заявку 3) Натискає «Схвалити» 4) Система створює нового користувача 5) Оновлює статус заявки на “approved”
Альтернативні потоки / Розширення	3a) Адмін натискає «Відхилити» - статус “rejected”, користувача не створено. 2a) Заявку видалено раніше - повідомлення «Заявку не знайдено».
Flows Error (потоки помилок)	E1) Неможливо створити користувача через дубль логіну
Бізнес-правила	BR1) Новий користувач завжди отримує роль “user”

Таблиця 3.10 – UC10

Елемент UC	Опис
Ім’я	Manage Users
Пріоритет	Must
Область дії	Telephone Directory IS
Контекст	Адміністратор працює зі списком користувачів: зміна ролей, видалення.
Діюча особа (Actor)	Admin
Мета	Повний контроль над обліковими записами
Передумови	Admin авторизований, та у БД існує табличка User
Тригер	Перехід на сторінку /admin/users
Результат (післяумова)	Збереження даних таблиці User
Основний потік	1) Admin відкриває /admin/users 2) Обирає дію: «Видалити», «Змінити роль» 3) Система валідує введені дані 4) Оновлює таблицю User відповідно до дії.
Альтернативні потоки / Розширення	2c) Видалення себе самого - «Не можна видалити власний акаунт»
Flows Error (потоки помилок)	E1) SQLite помилка → запис не збережено.
Бізнес-правила	BR1) Доступ до управління користувачами має лише Admin BR2) Роль користувача може бути лише guest, user, operator або admin.

3.2 Архітектура програмної системи

Нижче наведено опис основних класів і модулів, які реалізують бізнес-логіку “Інформаційної системи телефонного довідника”

3.2.1 Документування класів

Таблиця 3.11

Характеристика	Опис
Назва класу/модуля	db.utils
Опис класу/модуля	Модуль відповідає за встановлення з’єднання з базою даних SQLite та забезпечує єдиний доступ до неї за допомогою патерну «контекстний менеджер»
Опис атрибутів (полів)	Модуль не містить статичних полів; конфігурація шляху до БД виконується у функції `get_db()`
Опис методів	get_db() — відкриває з’єднання з SQLite, встановлює row_factory та повертає курсор; використовується в усіх сервісних методах.
Залежності від інших класів	Використовується у всіх бізнес-модулях (`service.py`, `app.py`) для доступу до БД. Модуль виступає фундаментальною залежністю системи

Таблиця 3.12

Характеристика	Опис
Назва класу/модуля	db.init
Опис класу/модуля	Модуль відповідає за створення фізичної структури БД: таблиць користувачів, абонентів, адрес, вулиць, телефонів, операторів, боргів, ремонтів та служб Також при першому запуску, для демонстрації можливостей, генерує випадкові записи різних типів
Опис атрибутів (полів)	Немає полів - структура визначена SQL-скриптами всередині методу

Опис методів	init_db() — створює всі таблиці у SQLite, викликається при першому старті системи; забезпечує відповідність БД фізичній моделі
Залежності від інших класів	Використовується в `app.py` для ініціалізації БД при запуску. Не залежить від інших модулів

Таблиця 3.13

Характеристика	**Опис**
Назва класу/модуля	service.py
Опис класу/модуля	Центральний модуль бізнес-логіки. Реалізує всі операції з БД: управління користувачами, абонентами, адресами, номерами, боргами, операторами, ремонтами, службами та SQL-звіти.
Опис атрибутів (полів)	Немає, всі дані обробляються в межах функцій із використанням з'єднання з БД
Опис методів	Управління користувачами: get_user_by_login(), create_user(), delete_user(), update_user_role(), verify_password(). Заявки на реєстрацію: create_registration_request(), get_registration_requests(), approve_request(), reject_request(). Абоненти: create_subscriber(), update_subscriber(), delete_subscriber(), get_all_subscribers(), get_subscriber(). Адреси та вулиці: create_address(), update_address(), update_or_create_address(). Поштові відділення: update_or_create_post_office(). Телефонні номери: create_phone(), delete_phone(), get_phones_by_subscriber(). Борги: create_debt(), delete_debt(), get_debts_by_subscriber(), get_subscribers_with_debts(). Заявки на зміну номера: create_number_change_request(), apply_number_change(), delete_request(). Ремонтні роботи: create_repair(), update_repair(), delete_repair(), get_all_repairs(). Спецслужби: get_all_special_services(). SQL-звіти: run_builtin_query(), run_custom_sql()

Залежності від інших класів	Залежить від db.utils.get_db() для виконання SQL-операцій. Викликається з app.py. Є найбільш залежним і найбільш використовуваним компонентом системи

Таблиця 3.14

Характеристика	**Опис**
Назва класу/модуля	app
Опис класу/модуля	Модуль реалізує веб-інтерфейс: маршрути Flask, авторизацію, контролери для всіх CRUD-операцій та відображення UI через Jinja2 шаблони
Опис атрибутів (полів)	app - об’єкт Flask. secret_key - ключ сесії
Опис методів	Містить набір route-функцій: • /login, /logout - авторизація; • /subscribers, /subscriber/add, /subscriber/edit/... - CRUD абонентів; • /repairs, /repair/add, /repair/edit/... - CRUD ремонтів; • /services - перегляд служб; • /requests - заявки на зміну номерів; • /admin/... - керування користувачами та заявками; • /sql/reports, /sql/custom - робота з SQL.
Залежності від інших класів	Використовує модуль service як шар бізнес-логіки. Використовує Flask для маршрутизації, Jinja2 — для шаблонів, db.init — для ініціалізації бази.

3.2.2 Документування модулів

Наведено в таблиці 3.15

Таблиця 3.15

Назва модуля	Функціональність	Структура / API	Використання
app.py	Точка входу в систему; створює Flask-додаток, реєструє маршрут / для дашборда.	Маршрути: /, /login, /logout, /subscribers, /repairs, /services, /admin/users, /admin/requests, /sql/reports	Запуск системи: python app.py Робота через браузер за адресою http://localhost:5000
service.py	Всі CRUD-операції, робота з БД, бізнес-логіка, 10 SQL-запитів, обслуговування абонентів, адрес, телефонів, боргів, заявок, ремонтів, служб.	Функції групами: користувачі, заявки, абоненти, телефони, адреси, оператори, борги, ремонти, спецслужби	Викликається з app.py. Єдиний шар бізнес-логіки
db/utils.py	Повертає підключення до SQLite.	get_db() відкриває БД і повертає connection.	Використовується у всіх функціях service.py.
db/init.py	Створює структуру БД (CREATE TABLE).	init_db() створення всіх таблиць.	Викликається при першому запуску програми
templates/	HTML-шаблони інтерфейсу системи.	Використовує Bootstrap 5 Шаблони всіх сторінок: index, login, subscribers, repairs, services, admin, sql_reports.	Рендеряться через render_template().
static/	Стилі (CSS) та скрипт (JS).	css/style.css - базові стилі; js/script.js - додаткова логіка.	Підключаються в шаблонах через {{ url_for('static', ...) }}.

3.3 Огляд GUI програмної системи

Графічний інтерфейс системи реалізовано у вигляді веб-додатка на Flask із використанням шаблонів Jinja2 та фреймворку Bootstrap 5.

Дизайн побудований у мінімалістичному стилі: світла кольорова схема, адаптивні таблиці, чіткі форми введення та акцентовані кнопки дій.

Навігація організована у вигляді верхнього меню, яке автоматично змінює доступні пункти залежно від ролі користувача.

Усі сторінки витримані в єдиному стилі, використовують стандартні Bootstrap-компоненти (alerts, forms, tables, modals)

3.5 Програмні засоби розробки

Таблиця 3.16 – Програмні засоби розробки

Тип інструменту	Використані засоби
IDE	PyCharm Community Edition
Пакетний менеджер	pip
СКБД	SQLite
Управління проєктом	Git
Фреймворк UI	Bootstrap 5
JS-бібліотека	jQuery

3.6 Розгортання програмної системи

Таблиця 3.17 – Апаратні та програмні вимоги

Категорія	Вимоги
CPU	≥ 2 ядра
ОЗП	≥ 2 ГБ
Диск	≥ 1 ГБ вільного місця
ОС	Windows 10/11, Linux (Ubuntu 20.04+)
Python	3.10+
База даних	SQLite (вбудована)
Веб-сервер	WSGI (наприклад gunicorn)
Додатково	Git, virtualenv, nginx

Таблиця 3.18 – Процедура розгортання

Крок	Опис
1. Клонування репозиторію	git clone https://…/phone_informer.git і перехід у папку
2. Віртуальне оточення	python3 -m venv venv source venv/bin/activate
3. Встановлення залежностей	pip install -r requirements.txt
4. Ініціалізація бази даних	При першому запуску app.py система сама створює db/db.sqlite через init_db()
5. Налаштування SECRET_KEY	У app.py встановити власний app.secret_key = "..."
6. Локальний запуск	python app.py

	програма буде доступна на http://localhost:5000/
7. Проксі через nginx	Налаштувати блок location / { proxy_pass http://127.0.0.1:5000; … } у конфігу nginx
8. Запуск через gunicorn	gunicorn --workers 4 --bind 127.0.0.1:5000 app:app
9. Автоматичний запуск	Створити systemd-службу або налаштувати supervisor для процесу gunicorn

ВИСНОВКИ

У ході виконання курсового проєкту було реалізовано наступні завдання:

- Проведено аналіз предметної області та визначено ключові сутності телефонного довідника: абоненти, адреси, вулиці, поштові відділення, телефонні номери, мобільні оператори, заборгованості, ремонтні роботи, заявки на зміну номера, спеціальні служби, користувачі та заявки на реєстрацію.
- Розроблено логічну модель даних із побудовою ER-діаграми та формальним описом атрибутів і зв’язків між сутностями (1:1, 1:M, M:N).
- Проведено нормалізацію структури БД до 3НФ, що дозволило усунути дублювання даних, підвищити цілісність та оптимізувати зберігання адрес, вулиць та телефонних номерів.

  Створено фізичну модель даних у SQLite, що включає всі таблиці, зовнішні ключі, обмеження та оптимізовані зв'язки між сутностями.
- Реалізовано серверну частину на Flask, структуруючи програму на окремі модулі (app.py, service.py, шаблони, статика), що відповідає принципам модульності та архітектури.
- Реалізовано багаторівневу систему ролей (guest, user, operator, admin), включно з авторизацією, аутентифікацією, перевіркою доступу, системою заявок на реєстрацію та адмініструванням користувачів.
- Реалізовано CRUD-функціональність для всієї предметної області: абоненти, адреси, номери, борги, ремонтні роботи, спеціальні служби, мобільні оператори та інші сутності.
- Створено модуль пошуку, який підтримує розширені шаблони `*` та `?`, що дозволяє знаходити абонентів за частковими або нечіткими збігами.
- Впроваджено систему заявок на зміну номера, включно з автоматичним оновленням даних абонента після схвалення оператором.
- Створено веб-інтерфейс на основі Bootstrap 5, який включає таблиці, форми, повідомлення, адаптивні елементи управління та впорядковану навігацію, що забезпечує зручність для користувачів усіх ролей.
- Проведено тестування працездатності системи, включно з перевіркою CRUD-операцій, авторизації, пошуку, роботи SQL-звітів і валідації форм.
- Підготовлено інструкції з розгортання, що охоплюють створення та ініціалізацію БД SQLite, запуск сервера Flask та залежностей Python.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. МЕТОДИЧНІ РЕКОМЕНДАЦІЇ до виконання курсового проєкту (Комісарчук В.В., Янушевський С.В., Дячук Р.Л., Чернівці, 2025)

ДОДАТОК А – ПРОГРАМНИЙ ПРОДУКТ

app.py

```
from flask import Flask, render_template, request, redirect, url_for, session,
flash
from db.init import init_db
import service
from functools import wraps
from datetime import datetime
import os

app = Flask(__name__)
app.secret_key = "very-secret-key"


def get_current_user():
    user = session.get("user")
    if not user:
        return {"id": None, "login": "Guest", "role": "guest"}
    return user

@app.context_processor
def inject_current_user():
    return {"current_user": get_current_user()}

def allow(*roles):
    def decorator(f):
        @wraps(f)
        def wrapper(*args, **kwargs):
            user = get_current_user()
            if user["role"] not in roles:
                flash("Недостатньо прав для доступу", "warning")
                return redirect(url_for("index"))
            return f(*args, **kwargs)
        return wrapper
    return decorator


def login_required(role: str | None = None):
    def decorator(f):
        @wraps(f)
        def wrapper(*args, **kwargs):
            user = get_current_user()
            if user["role"] == "guest":
                flash("Потрібна авторизація", "warning")
                return redirect(url_for("login"))
            if role and user["role"] != role:
                flash("Недостатньо прав", "danger")
                return redirect(url_for("index"))
            return f(*args, **kwargs)
        return wrapper
    return decorator


@app.route("/")
def index():
    return render_template("index.html")


@app.route("/search")
@allow("user", "operator", "admin")
def search():
```

```
    q = request.args.get("q", "").strip()
    if not q:
        flash("Введіть критерій пошуку", "warning")
        return redirect(url_for("index"))
    results = service.search_subscribers(q)
    return render_template("search_results.html", query=q, results=results)

@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        login_ = request.form.get("login")
        password = request.form.get("password")

        user = service.get_user_by_login(login_)
        if user and service.verify_password(user, password):
            session["user"] = {
                "id": user["id"],
                "login": user["login"],
                "role": user["role"]
            }
            flash("Успішний вхід", "success")
            return redirect(url_for("index"))

        flash("Невірний логін або пароль", "danger")

    return render_template("login.html")


@app.route("/logout")
def logout():
    session.pop("user", None)
    flash("Ви вийшли із системи", "info")
    return redirect(url_for("index"))

@app.route("/request-access", methods=["GET", "POST"])
@allow("guest")
def request_access():
    if request.method == "POST":
        login = request.form["login"].strip()
        password = request.form["password"].strip()

        if not login or not password:
            flash("Логін та пароль обов'язкові", "danger")
            return redirect(url_for("request_access"))

        service.create_registration_request(login, password)
        flash("Заявку відправлено адміністратору", "success")
        return redirect(url_for("index"))

    return render_template("registration_request.html")

@app.route("/admin/requests")
@allow("admin")
def admin_requests():
    requests_list = service.get_registration_requests()
    return render_template("admin_requests.html", requests=requests_list)

@app.route("/admin/requests/approve/<int:req_id>")
@allow("admin")
def admin_approve_request(req_id):
    ok = service.approve_request(req_id)
    if ok:
        flash("Заявку схвалено, користувача створено", "success")
    else:
```

```
        flash("Заявку не знайдено", "danger")
    return redirect(url_for("admin_requests"))

@app.route("/admin/requests/reject/<int:req_id>")
@allow("admin")
def admin_reject_request(req_id):
    service.reject_request(req_id)
    flash("Заявку відхилено", "info")
    return redirect(url_for("admin_requests"))


@app.route("/admin/users")
@allow("admin")
def admin_users():
    users = service.get_all_users()
    return render_template("admin_users.html", users=users)


@app.route("/admin/users/create", methods=["GET", "POST"])
@allow("admin")
def admin_create_user():
    if request.method == "POST":
        login = request.form["login"].strip()
        password = request.form["password"].strip()
        role = request.form["role"]

        if not login or not password:
            flash("Логін і пароль обов'язкові", "danger")
            return redirect(url_for("admin_create_user"))

        service.create_user(login, password, role)
        flash("Користувача створено", "success")
        return redirect(url_for("admin_users"))

    return render_template("admin_create_user.html")


@app.route("/admin/users/<int:user_id>/delete")
@allow("admin")
def admin_delete_user(user_id):
    service.delete_user(user_id)
    flash("Користувача видалено", "info")
    return redirect(url_for("admin_users"))


@app.route("/admin/users/<int:user_id>/role", methods=["POST"])
@allow("admin")
def admin_change_role(user_id):
    new_role = request.form["role"]
    service.update_user_role(user_id, new_role)
    flash("Роль оновлено", "success")
    return redirect(url_for("admin_users"))


@app.route("/subscribers")
@allow("guest", "user", "operator", "admin")
def subscribers():
    subs = service.get_all_subscribers()
    return render_template("subscribers.html", subscribers=subs)


@app.route("/subscriber/<int:sub_id>")
@allow("guest", "user", "operator", "admin")
def subscriber_view(sub_id):
```

```
    sub = service.get_subscriber(sub_id)
    if not sub:
        flash("Абонента не знайдено", "danger")
        return redirect(url_for("subscribers"))
    phones = service.get_phones_by_subscriber(sub_id)
    debts = service.get_debts_by_subscriber(sub_id)
    return render_template(
        "subscriber_view.html",
        subscriber=sub, phones=phones, debts=debts
    )

@app.route("/subscriber/add", methods=["GET", "POST"])
@allow("operator", "admin")
def subscriber_add():
    if request.method == "POST":
        lastname = request.form.get("lastname")
        firstname = request.form.get("firstname")
        middlename = request.form.get("middlename")

        street_name = request.form.get("street_name")
        street_type = request.form.get("street_type", "вул.")
        building = request.form.get("building")
        apartment = request.form.get("apartment")

        addr_id = None
        if street_name and building:
            addr_id = service.create_address(street_name, street_type, building,
apartment)

        service.create_subscriber(lastname, firstname, middlename, addr_id, None)
        flash("Абонент створений", "success")
        return redirect(url_for("subscribers"))

    return render_template("subscriber_form.html", mode="add")

@app.route("/subscriber/edit/<int:sub_id>", methods=["GET", "POST"])
@allow("operator", "admin")
def subscriber_edit(sub_id):
    sub = service.get_subscriber(sub_id)
    if not sub:
        flash("Абонента не знайдено", "danger")
        return redirect(url_for("subscribers"))

    if request.method == "POST":
        lastname = request.form.get("lastname")
        firstname = request.form.get("firstname")
        middlename = request.form.get("middlename")

        street_type = request.form.get("street_type")
        street_name = request.form.get("street_name")
        building = request.form.get("building")
        apartment = request.form.get("apartment")

        office_number = request.form.get("post_office")

        service.update_subscriber(sub_id, lastname, firstname, middlename)

        address_id = service.update_or_create_address(
            sub["address_id"],
            street_name,
            street_type,
            building,
            apartment
        )
```

```
        post_office_id = service.update_or_create_post_office(
            sub["id_post_office"],
            office_number,
            address_id
        )

        service.update_subscriber_address_and_postoffice(sub_id, address_id,
post_office_id)

        flash("Абонента та адресу оновлено", "success")
        return redirect(url_for("subscriber_view", sub_id=sub_id))

    return render_template("subscriber_form.html", mode="edit", subscriber=sub)

@app.route("/subscriber/delete/<int:sub_id>")
@allow("admin")
def subscriber_delete(sub_id):
    service.delete_subscriber(sub_id)
    flash("Абонента видалено", "info")
    return redirect(url_for("subscribers"))



@app.route("/subscriber/<int:sub_id>/phones/add", methods=["POST"])
@allow("operator", "admin")
def phone_add(sub_id):
    number = request.form.get("number")
    ptype = request.form.get("type") or "mobile"
    operator_id = request.form.get("operator_id") or None
    operator_id = int(operator_id) if operator_id else None

    if not number:
        flash("Номер не може бути порожнім", "warning")
    else:
        service.create_phone(number, ptype, sub_id, operator_id, 1)
        flash("Номер додано", "success")

    return redirect(url_for("subscriber_view", sub_id=sub_id))


@app.route("/subscriber/<int:sub_id>/phones/delete/<int:phone_id>")
@allow("operator", "admin")
def phone_delete(sub_id, phone_id):
    service.delete_phone(phone_id)
    flash("Номер видалено", "info")
    return redirect(url_for("subscriber_view", sub_id=sub_id))


@app.route("/debts")
@allow("guest", "user", "operator", "admin")
def debts():
    q = request.args.get("q", "").strip()

    if q and get_current_user()["role"] != "guest":
        debtors = service.search_debtors(q)
    else:
        debtors = service.get_subscribers_with_debts()

    return render_template("debts.html", debtors=debtors, query=q)

@app.route("/subscriber/<int:sub_id>/debts/delete/<int:debt_id>")
@allow("operator", "admin")
def debt_delete(sub_id, debt_id):
```

```
    service.delete_debt(debt_id)
    flash("Борг видалено", "info")
    return redirect(url_for("subscriber_view", sub_id=sub_id))


@app.route("/subscriber/<int:sub_id>/debts/add", methods=["POST"])
@allow("operator", "admin")
def debt_add(sub_id):
    amount = float(request.form.get("amount", 0))
    date_start = request.form.get("date_start") or
datetime.now().strftime("%Y-%m-%d")
    deadline = request.form.get("deadline") or None
    service.create_debt(sub_id, amount, date_start, deadline, "active")
    flash("Борг додано", "success")
    return redirect(url_for("subscriber_view", sub_id=sub_id))

@app.route("/requests")
@allow("operator", "admin")
def requests_list():
    reqs = service.get_all_number_change_requests()
    return render_template("requests.html", requests=reqs)

@app.route("/requests/change/<int:sub_id>/<old_number>")
@allow("user")
def request_number_page(sub_id, old_number):
    return render_template("request_number_form.html", sub_id=sub_id,
old_number=old_number)

@app.route("/requests/add", methods=["POST"])
@allow("user", "operator", "admin")
def request_add():
    sub_id = int(request.form.get("subscriber_id"))
    old_number = request.form.get("old_number")
    new_number = request.form.get("new_number")
    date_request = datetime.now().strftime("%Y-%m-%d")
    service.create_number_change_request(sub_id, old_number, new_number,
date_request)
    flash("Заявку на зміну номера створено", "success")
    return redirect(url_for("requests_list"))

@app.route("/requests/approve/<int:req_id>")
@allow("operator", "admin")
def request_approve(req_id):
    req = service.get_request(req_id)
    if not req:
        flash("Заявку не знайдено", "danger")
        return redirect(url_for("requests_list"))

    service.apply_number_change(req)

    service.delete_request(req_id)

    flash("Заявку прийнято. Номер оновлено.", "success")
    return redirect(url_for("requests_list"))


@app.route("/requests/reject/<int:req_id>")
@allow("operator", "admin")
def request_reject(req_id):
    service.delete_request(req_id)
    flash("Заявку відхилено.", "info")
    return redirect(url_for("requests_list"))
```

```
@app.route("/repairs")
def repairs():
    q = request.args.get("q", "").strip()

    if q and get_current_user()["role"] != "guest":
        repairs = service.search_repairs(q)
    else:
        repairs = service.get_all_repairs()

    return render_template("repairs.html", repairs=repairs, query=q)

@app.route("/repair/add", methods=["GET", "POST"])
@allow("operator", "admin")
def repair_add():
    if request.method == "POST":
        street_type = request.form["street_type"]
        street_name = request.form["street_name"]
        building = request.form["building"]
        apartment = request.form["apartment"]
        date_start = request.form["date_start"]
        date_end = request.form["date_end"]
        description = request.form["description"]

        address_id = service.create_address(street_name, street_type, building,
apartment)

        service.create_repair(address_id, date_start, date_end, description)
        flash("Ремонтні роботи додано", "success")
        return redirect(url_for("repairs"))

    return render_template("repair_add.html")

@app.route("/repair/edit/<int:repair_id>", methods=["GET", "POST"])
@allow("operator", "admin")
def repair_edit(repair_id):
    repair = service.get_repair(repair_id)
    if not repair:
        flash("Ремонт не знайдено", "danger")
        return redirect(url_for("repairs"))

    if request.method == "POST":
        street_type = request.form["street_type"]
        street_name = request.form["street_name"]
        building = request.form["building"]
        apartment = request.form["apartment"]
        date_start = request.form["date_start"]
        date_end = request.form["date_end"]
        description = request.form["description"]

        service.update_address(repair["id_address"], street_name, street_type,
building, apartment)

        service.update_repair(repair_id, repair["id_address"], date_start,
date_end, description)

        flash("Дані ремонту оновлено", "success")
        return redirect(url_for("repairs"))

    return render_template("repair_edit.html", repair=repair)

@app.route("/repair/delete/<int:repair_id>")
@allow("operator", "admin")
```

```
def repair_delete(repair_id):
    service.delete_repair(repair_id)
    flash("Ремонт видалено", "info")
    return redirect(url_for("repairs"))


@app.route("/services")
@allow("guest", "user", "operator", "admin")
def services_list():
    services = service.get_all_special_services()
    return render_template("services.html", services=services)

@app.route("/sql/reports", methods=["GET", "POST"])
@allow("user", "operator", "admin")
def sql_reports():
    result = None
    selected = None
    params = {}

    if request.method == "POST":
        selected = request.form.get("query_id")

        params = {
            "lastname": request.form.get("lastname", "").strip(),
            "firstname": request.form.get("firstname", "").strip(),
            "street_name": request.form.get("street_name", "").strip()
        }

        result = service.run_builtin_query(selected, params)

    return render_template("sql_reports.html",
                           selected=selected,
                           result=result)
# ============================================================

@app.route("/sql/custom", methods=["GET", "POST"])
@allow("operator", "admin")
def sql_custom():
    cols = []
    rows = []
    sql_text = ""

    if request.method == "POST":
        sql_text = request.form.get("sql_text", "")
        if sql_text.strip():
            try:
                cols, rows = service.run_custom_sql(sql_text)
                flash("Запит виконано", "success")
            except Exception as e:
                flash(f"Помилка: {e}", "danger")

    return render_template("sql_custom.html",
                           cols=cols, rows=rows, sql_text=sql_text)

if __name__ == "__main__":
    if not os.path.exists('db/db.sqlite'):
        print('Створення БД')
        init_db()

        app.run(debug=True)
```

service.py

```
import random
from db.utils import get_db
from werkzeug.security import generate_password_hash, check_password_hash

def get_user_by_login(login):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("SELECT * FROM User WHERE login = ?", (login,))
        return cur.fetchone()

def create_user(login: str, password: str, role: str):
    hashed = generate_password_hash(password)
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            INSERT INTO User (login, password, role)
            VALUES (?, ?, ?)
        """, (login, hashed, role))

def delete_user(user_id: int):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("DELETE FROM User WHERE id = ?", (user_id,))

def update_user_role(user_id: int, new_role: str):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("UPDATE User SET role = ? WHERE id = ?", (new_role, user_id))

def verify_password(user_row, password):
    if user_row is None:
        return False
    return check_password_hash(user_row["password"], password)

def get_all_users():
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("SELECT id, login, role FROM User ORDER BY login")
        return cur.fetchall()



def create_registration_request(login: str, password: str):
    hashed = generate_password_hash(password)
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            INSERT INTO RegistrationRequest (login, password, status)
            VALUES (?, ?, 'new')
        """, (login, hashed))

def get_registration_requests():
    with get_db() as conn:
```

```
        cur = conn.cursor()
        cur.execute("""
            SELECT id, login, status
            FROM RegistrationRequest
            ORDER BY id DESC
        """)
        return cur.fetchall()

def get_registration_request(req_id: int):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT id, login, password, status
            FROM RegistrationRequest
            WHERE id = ?
        """, (req_id,))
        return cur.fetchone()

def approve_request(req_id: int):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("SELECT login, password FROM RegistrationRequest WHERE id =
?", (req_id,))
        row = cur.fetchone()
        if not row:
            return False

        login = row["login"]
        password_hash = row["password"]

        cur.execute("""
            INSERT INTO User (login, password, role)
            VALUES (?, ?, 'user')
        """, (login, password_hash))

        cur.execute("""
            UPDATE RegistrationRequest
            SET status = 'approved'
            WHERE id = ?
        """, (req_id,))
        return True

def reject_request(req_id: int):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            UPDATE RegistrationRequest
            SET status = 'rejected'
            WHERE id = ?
        """, (req_id,))



def get_or_create_street(name, st_type="вул."):
```

```
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("SELECT id FROM Street WHERE name = ? AND type = ?", (name,
st_type))
        row = cur.fetchone()
        if row:
            return row["id"]
        cur.execute("INSERT INTO Street (name, type) VALUES (?, ?)", (name,
st_type))
        return cur.lastrowid

def create_address(street_name, street_type, building, apartment):
    street_id = get_or_create_street(street_name, street_type)
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            INSERT INTO Address (id_street, building, apartment)
            VALUES (?, ?, ?)
        """, (street_id, building, apartment))
        return cur.lastrowid

def update_address(address_id, street_name, street_type, building, apartment):
    with get_db() as conn:
        cur = conn.cursor()

        cur.execute("SELECT id FROM Street WHERE name=? AND type=?",
(street_name, street_type))
        row = cur.fetchone()

        if row:
            street_id = row["id"]
        else:
            cur.execute("INSERT INTO Street (name, type) VALUES (?, ?)",
(street_name, street_type))
            street_id = cur.lastrowid

        cur.execute("""
            UPDATE Address
            SET id_street=?, building=?, apartment=?
            WHERE id=?
        """, (street_id, building, apartment, address_id))

def update_or_create_address(address_id, street_name, street_type, building,
apartment):
    with get_db() as conn:
        cur = conn.cursor()

        cur.execute("SELECT id FROM Street WHERE name=? AND type=?",
(street_name, street_type))
        row = cur.fetchone()
        if row:
            street_id = row["id"]
        else:
```

```
            cur.execute("INSERT INTO Street (name, type) VALUES (?, ?)",
(street_name, street_type))
            street_id = cur.lastrowid

        if address_id:
            cur.execute("""
                UPDATE Address
                SET id_street = ?, building = ?, apartment = ?
                WHERE id = ?
            """, (street_id, building, apartment, address_id))
            return address_id

        # інакше створити нову
        cur.execute("""
            INSERT INTO Address (id_street, building, apartment)
            VALUES (?, ?, ?)
        """, (street_id, building, apartment))
        return cur.lastrowid

def get_address(address_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT Address.*, Street.name AS street_name, Street.type AS
street_type
            FROM Address
            JOIN Street ON Street.id = Address.id_street
            WHERE Address.id = ?
        """, (address_id,))
        return cur.fetchone()


def get_all_post_offices():
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT PostOffice.*, Street.name AS street_name, Street.type AS
street_type,
                   Address.building, Address.apartment
            FROM PostOffice
            LEFT JOIN Address ON Address.id = PostOffice.id_address
            LEFT JOIN Street ON Street.id = Address.id_street
            ORDER BY office_number
        """)
        return cur.fetchall()

def create_post_office_for_address(conn, address_id):
    cur = conn.cursor()

    office_number = 58000 + random.randint(1, 25)

    cur.execute("""
        INSERT INTO PostOffice (office_number, id_address)
        VALUES (?, ?)
```

```
    """, (office_number, address_id))

    return cur.lastrowid

def update_or_create_post_office(post_office_id, office_number, address_id):
    with get_db() as conn:
        cur = conn.cursor()

        if post_office_id:
            cur.execute("""
                UPDATE PostOffice
                SET office_number=?, id_address=?
                WHERE id=?
            """, (office_number, address_id, post_office_id))
            return post_office_id


        cur.execute("""
            INSERT INTO PostOffice (office_number, id_address)
            VALUES (?, ?)
        """, (office_number, address_id))
        return cur.lastrowid

def update_subscriber_address_and_postoffice(sub_id, address_id, post_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            UPDATE Subscriber
            SET id_address=?, id_post_office=?
            WHERE id=?
        """, (address_id, post_id, sub_id))



def get_all_subscribers():
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT
                s.id,
                s.lastname,
                s.firstname,
                s.middlename,

                (
                    SELECT pn.number
                    FROM PhoneNumber pn
                    WHERE pn.id_subscriber = s.id AND pn.active = 1
                    ORDER BY
                        (CASE pn.type
                            WHEN 'mobile' THEN 0
                            WHEN 'home' THEN 1
                            WHEN 'service' THEN 2
                            ELSE 3 END),
```

```
                    pn.id ASC
                LIMIT 1
            ) AS main_phone,

            st.type AS street_type,
            st.name AS street_name,
            a.building,
            a.apartment,

            (st.type || '. ' || st.name || ' ' || a.building ||
                CASE WHEN a.apartment IS NOT NULL THEN ', кв. ' ||
a.apartment ELSE '' END
            ) AS full_address,

            po.office_number

        FROM Subscriber s
        LEFT JOIN Address a ON a.id = s.id_address
        LEFT JOIN Street st ON st.id = a.id_street
        LEFT JOIN PostOffice po ON po.id = s.id_post_office

        ORDER BY s.lastname, s.firstname
    """)
    return cur.fetchall()

def get_subscriber(sub_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT
                s.*,
                a.id AS address_id,
                a.building,
                a.apartment,
                st.name AS street_name,
                st.type AS street_type,
                po.id AS post_office_id,
                po.office_number
            FROM Subscriber s
            LEFT JOIN Address a ON a.id = s.id_address
            LEFT JOIN Street st ON st.id = a.id_street
            LEFT JOIN PostOffice po ON po.id = s.id_post_office
            WHERE s.id = ?
        """, (sub_id,))
        return cur.fetchone()

def create_subscriber(lastname, firstname, middlename, address_id,
post_office_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            INSERT INTO Subscriber (lastname, firstname, middlename, id_address,
id_post_office)
            VALUES (?, ?, ?, ?, ?)
```

```
        """, (lastname, firstname, middlename, address_id, post_office_id))
        return cur.lastrowid

def update_subscriber(sub_id, lastname, firstname, middlename):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            UPDATE Subscriber
            SET lastname = ?, firstname = ?, middlename = ?
            WHERE id = ?
        """, (lastname, firstname, middlename, sub_id))

def delete_subscriber(sub_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("DELETE FROM Subscriber WHERE id = ?", (sub_id,))

def search_subscribers(pattern):
    like = pattern.replace("*", "%").replace("?", "_")

    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT DISTINCT
                s.id,
                s.lastname,
                s.firstname,
                s.middlename,

                pn.number AS phone_number,

                st.type AS street_type,
                st.name AS street_name,
                a.building,
                a.apartment,

                (st.type || '. ' || st.name || ' ' || a.building ||
                    CASE WHEN a.apartment IS NOT NULL THEN ', кв. ' ||
a.apartment ELSE '' END
                ) AS full_address,

                po.office_number

            FROM Subscriber s
            LEFT JOIN PhoneNumber pn ON pn.id_subscriber = s.id AND pn.active = 1
            LEFT JOIN Address a ON a.id = s.id_address
            LEFT JOIN Street st ON st.id = a.id_street
            LEFT JOIN PostOffice po ON po.id = s.id_post_office

            WHERE s.lastname LIKE ?
               OR s.firstname LIKE ?
               OR s.middlename LIKE ?
               OR pn.number LIKE ?
               OR po.office_number LIKE ?
```

```
        """, (like, like, like, like, like))

        return cur.fetchall()




def get_all_operators():
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("SELECT * FROM MobileOperator ORDER BY name")
        return cur.fetchall()

def create_operator(name, prefix):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("INSERT INTO MobileOperator (name, prefix) VALUES (?, ?)",
(name, prefix))
        return cur.lastrowid



def get_phones_by_subscriber(sub_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT PhoneNumber.*, MobileOperator.name AS operator_name
            FROM PhoneNumber
            LEFT JOIN MobileOperator ON MobileOperator.id =
PhoneNumber.id_operator
            WHERE PhoneNumber.id_subscriber = ?
            ORDER BY PhoneNumber.number
        """, (sub_id,))
        return cur.fetchall()

def create_phone(number, ptype, sub_id, operator_id, active=1):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            INSERT INTO PhoneNumber (number, type, id_subscriber, id_operator,
active)
            VALUES (?, ?, ?, ?, ?)
        """, (number, ptype, sub_id, operator_id, active))
        return cur.lastrowid

def delete_phone(phone_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("DELETE FROM PhoneNumber WHERE id = ?", (phone_id,))



def get_all_number_change_requests():
    with get_db() as conn:
```

```
        cur = conn.cursor()
        cur.execute("""
            SELECT r.*,
                   s.lastname, s.firstname, s.middlename
            FROM NumberChangeRequest r
            LEFT JOIN Subscriber s ON s.id = r.id_subscriber
            ORDER BY r.date_request DESC
        """)
        return cur.fetchall()

def get_request(req_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("SELECT * FROM NumberChangeRequest WHERE id=?", (req_id,))
        return cur.fetchone()

def update_request_status(request_id, status):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            UPDATE NumberChangeRequest
            SET status=?
            WHERE id=?
        """, (status, request_id))

def delete_request(request_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("DELETE FROM NumberChangeRequest WHERE id=?", (request_id,))

def apply_number_change(request_row):
    with get_db() as conn:
        cur = conn.cursor()

        sub_id = request_row["id_subscriber"]
        old_number = request_row["old_number"]
        new_number = request_row["new_number"]

        cur.execute("""
            DELETE FROM PhoneNumber
            WHERE number=? AND id_subscriber=?
        """, (old_number, sub_id))

        cur.execute("""
            INSERT INTO PhoneNumber (number, type, id_subscriber, id_operator)
            VALUES (?, 'mobile', ?, NULL)
        """, (new_number, sub_id))



def create_number_change_request(sub_id, old_number, new_number, date_request,
status="new"):
    with get_db() as conn:
        cur = conn.cursor()
```

```
        cur.execute("""
            INSERT INTO NumberChangeRequest (id_subscriber, old_number,
new_number, date_request, status)
            VALUES (?, ?, ?, ?, ?)
        """, (sub_id, old_number, new_number, date_request, status))
        return cur.lastrowid

def update_number_change_status(req_id, status):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            UPDATE NumberChangeRequest
            SET status = ?
            WHERE id = ?
        """, (status, req_id))




def get_debts_by_subscriber(sub_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT * FROM Debt
            WHERE id_subscriber = ?
            ORDER BY date_start DESC
        """, (sub_id,))
        return cur.fetchall()

def create_debt(sub_id, amount, date_start, deadline, status="active"):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            INSERT INTO Debt (id_subscriber, amount, date_start, deadline,
status)
            VALUES (?, ?, ?, ?, ?)
        """, (sub_id, amount, date_start, deadline, status))
        return cur.lastrowid

def delete_debt(debt_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("DELETE FROM Debt WHERE id=?", (debt_id,))

def update_debt(debt_id, amount, status):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            UPDATE Debt
            SET amount = ?, status = ?
            WHERE id = ?
        """, (amount, status, debt_id))

def get_subscribers_with_debts():
```

```
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT s.*, SUM(d.amount) AS total_debt
            FROM Subscriber s
            JOIN Debt d ON d.id_subscriber = s.id
            WHERE d.status = 'active'
            GROUP BY s.id
            HAVING total_debt > 0
            ORDER BY total_debt DESC
        """)
        return cur.fetchall()

def search_debtors(query: str):
    wild = query.replace("*", "%").replace("?", "_")
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT s.lastname, s.firstname, s.middlename,
                   d.amount, d.date_start, d.deadline, d.status
            FROM Debt d
            JOIN Subscriber s ON s.id = d.id_subscriber
            WHERE s.lastname LIKE ?
               OR s.firstname LIKE ?
               OR s.middlename LIKE ?
            ORDER BY s.lastname
        """, (wild, wild, wild))
        return cur.fetchall()



def get_all_repairs():
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT rw.*,
                   st.name AS street_name, st.type AS street_type,
                   a.building, a.apartment
            FROM RepairWork rw
            LEFT JOIN Address a ON a.id = rw.id_address
            LEFT JOIN Street st ON st.id = a.id_street
            ORDER BY date_start DESC
        """)
        return cur.fetchall()

def get_repair(repair_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT r.*,
                   st.name AS street_name, st.type AS street_type,
                   a.building, a.apartment
            FROM RepairWork r
            LEFT JOIN Address a ON a.id = r.id_address
```

```
            LEFT JOIN Street st ON st.id = a.id_street
            WHERE r.id = ?
        """, (repair_id,))
        return cur.fetchone()

def create_repair(id_address, date_start, date_end, description):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            INSERT INTO RepairWork (id_address, date_start, date_end,
description)
            VALUES (?, ?, ?, ?)
        """, (id_address, date_start, date_end, description))
        return cur.lastrowid

def update_repair(repair_id, address_id, date_start, date_end, description):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            UPDATE RepairWork
            SET id_address = ?, date_start = ?, date_end = ?, description = ?
            WHERE id = ?
        """, (address_id, date_start, date_end, description, repair_id))

def delete_repair(repair_id):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("DELETE FROM RepairWork WHERE id=?", (repair_id,))

def search_repairs(query: str):
    wild = query.replace("*", "%").replace("?", "_")
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT r.*,
                   st.name AS street_name, st.type AS street_type,
                   a.building, a.apartment
            FROM RepairWork r
            JOIN Address a ON a.id = r.id_address
            JOIN Street st ON st.id = a.id_street
            WHERE st.name LIKE ?
               OR a.building LIKE ?
               OR a.apartment LIKE ?
        """, (wild, wild, wild))
        return cur.fetchall()



def get_all_special_services():
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("""
            SELECT ss.*, pn.number
            FROM SpecialService ss
```

```
            LEFT JOIN PhoneNumber pn ON pn.id = ss.id_number
            ORDER BY ss.name
        """)
        return cur.fetchall()



def run_custom_sql(sql_text: str):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute(sql_text)
        try:
            rows = cur.fetchall()
            cols = [d[0] for d in cur.description] if cur.description else []
            return cols, rows
        except Exception:
            return [], []


def run_builtin_query(query_id, params):
    with get_db() as conn:
        cur = conn.cursor()

        if query_id == "1":
            cur.execute("""
                SELECT
                    name AS "Назва служби",
                    weekday AS "Дні роботи",
                    time_start AS "Початок прийому",
                    time_end AS "Кінець прийому"
                FROM SpecialService
                ORDER BY name
            """)
            return cur.fetchall()

        if query_id == "2":
            cur.execute("""
                SELECT
                    s.lastname AS "Прізвище",
                    s.firstname AS "Ім’я",
                    s.middlename AS "По батькові",
                    r.old_number AS "Старий номер",
                    r.new_number AS "Новий номер",
                    r.date_request AS "Дата заявки"
                FROM NumberChangeRequest r
                JOIN Subscriber s ON s.id = r.id_subscriber
                ORDER BY r.date_request DESC
            """)
            return cur.fetchall()


        if query_id == "3":
            cur.execute("""
                SELECT
```

```
                st.type || '. ' || st.name AS "Вулиця",
                a.building AS "Будинок",
                a.apartment AS "Квартира",
                rw.date_start AS "Початок ремонту",
                rw.date_end AS "Кінець ремонту",
                rw.description AS "Опис робіт"
            FROM RepairWork rw
            JOIN Address a ON a.id = rw.id_address
            JOIN Street st ON st.id = a.id_street
            ORDER BY rw.date_start
        """)
        return cur.fetchall()


    if query_id == "4":
        cur.execute("""
            SELECT
                ss.name AS "Назва служби",
                pn.number AS "Телефон"
            FROM SpecialService ss
            JOIN PhoneNumber pn ON pn.id = ss.id_number
            ORDER BY ss.name
        """)
        return cur.fetchall()


    if query_id == "5":
        lastname = params.get("lastname", "")
        firstname = params.get("firstname", "")

        cur.execute("""
            SELECT
                s.lastname AS "Прізвище",
                s.firstname AS "Ім'я",
                s.middlename AS "По батькові"
            FROM Subscriber s
            WHERE s.lastname LIKE ? AND s.firstname LIKE ?
        """, (lastname + "%", firstname + "%"))

        rows = cur.fetchall()
        return rows, len(rows)


    if query_id == "6":
        cur.execute("""
            SELECT
                mo.name AS "Оператор",
                COUNT(pn.id) AS "Кількість абонентів"
            FROM PhoneNumber pn
            JOIN MobileOperator mo ON mo.id = pn.id_operator
            WHERE pn.active = 1
            GROUP BY mo.id
            ORDER BY COUNT(pn.id) DESC
        """)
```

```
            return cur.fetchall()


        if query_id == "7":
            cur.execute("""
                SELECT
                    s.lastname AS "Прізвище",
                    s.firstname AS "Ім'я",
                    s.middlename AS "По батькові",
                    SUM(d.amount) AS "Загальна заборгованість"
                FROM Debt d
                JOIN Subscriber s ON s.id = d.id_subscriber
                WHERE d.status = 'active'
                GROUP BY s.id
                ORDER BY SUM(d.amount) DESC
            """)
            return cur.fetchall()


        if query_id == "8":
            cur.execute("""
                SELECT
                    s.lastname AS "Прізвище",
                    s.firstname AS "Ім'я",
                    s.middlename AS "По батькові",
                    r.old_number AS "Старий номер",
                    r.new_number AS "Новий номер",
                    st.type || '. ' || st.name || ' ' || a.building ||
                        CASE WHEN a.apartment IS NOT NULL
                             THEN ', кв. ' || a.apartment
                             ELSE '' END AS "Поточна адреса"
                FROM NumberChangeRequest r
                JOIN Subscriber s ON s.id = r.id_subscriber
                JOIN Address a ON a.id = s.id_address
                JOIN Street st ON st.id = a.id_street
                ORDER BY r.date_request DESC
            """)
            return cur.fetchall()


        if query_id == "9":
            street = params.get("street_name", "")

            cur.execute("""
                SELECT
                    s.lastname AS "Прізвище",
                    s.firstname AS "Ім'я",
                    s.middlename AS "По батькові",
                    st.type AS "Тип вулиці",
                    st.name AS "Назва вулиці",
                    a.building AS "Будинок",
                    a.apartment AS "Квартира"
                FROM Subscriber s
                JOIN Address a ON a.id = s.id_address
```

```
            JOIN Street st ON st.id = a.id_street
            WHERE st.name LIKE ?
            ORDER BY s.lastname, s.firstname
        """, (street + "%",))

        rows = cur.fetchall()
        return rows, len(rows)


    if query_id == "10":
        cur.execute("""
            SELECT
                st.type || '. ' || st.name AS "Вулиця",
                COUNT(s.id) AS "Кількість мешканців"
            FROM Subscriber s
            JOIN Address a ON a.id = s.id_address
            JOIN Street st ON st.id = a.id_street
            GROUP BY st.id
            ORDER BY COUNT(s.id) DESC
        """)
        return cur.fetchall()

    return []
```

db / utils.py

```python
import sqlite3
from contextlib import contextmanager

DB_PATH = "db/db.sqlite"

@contextmanager
def get_db():
    conn = sqlite3.connect(DB_PATH, timeout=30, check_same_thread=False)
    conn.execute("PRAGMA foreign_keys = ON;")
    conn.execute("PRAGMA synchronous = NORMAL;")
    conn.execute("PRAGMA journal_mode = WAL;")
    conn.row_factory = sqlite3.Row
    try:
        yield conn
        conn.commit()
    except Exception:
        conn.rollback()
        raise
    finally:
        conn.close()

def execute_query(query, params=()):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute(query, params)
        return cur

def fetch_all(query, params=()):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute(query, params)
        return cur.fetchall()

def fetch_one(query, params=()):
    with get_db() as conn:
        cur = conn.cursor()
        cur.execute(query, params)
        return cur.fetchone()
```

db / init.py

```
import random, datetime
from werkzeug.security import generate_password_hash
from db.utils import get_db
from service import create_post_office_for_address

def init_db():
   with get_db() as conn:
       cur = conn.cursor()

       cur.execute("""
           CREATE TABLE IF NOT EXISTS User (
               id INTEGER PRIMARY KEY AUTOINCREMENT,
               login TEXT UNIQUE NOT NULL,
               password TEXT NOT NULL,
               role TEXT NOT NULL
           );
       """)

       cur.execute("""
           CREATE TABLE IF NOT EXISTS Street (
               id INTEGER PRIMARY KEY AUTOINCREMENT,
               name TEXT,
               type TEXT
           );
       """)

       cur.execute("""
           CREATE TABLE IF NOT EXISTS Address (
               id INTEGER PRIMARY KEY AUTOINCREMENT,
               id_street INTEGER NOT NULL,
               building TEXT,
               apartment TEXT,
               FOREIGN KEY(id_street) REFERENCES Street(id) ON DELETE CASCADE ON
UPDATE CASCADE
           );
       """)

       cur.execute("""
           CREATE TABLE IF NOT EXISTS PostOffice (
               id INTEGER PRIMARY KEY AUTOINCREMENT,
               office_number INTEGER,
               id_address INTEGER,
               FOREIGN KEY(id_address) REFERENCES Address(id) ON DELETE SET NULL
           );
       """)

       cur.execute("""
           CREATE TABLE IF NOT EXISTS Subscriber (
               id INTEGER PRIMARY KEY AUTOINCREMENT,
               lastname TEXT,
               firstname TEXT,
               middlename TEXT,
               id_address INTEGER,
```

```
            id_post_office INTEGER,
            FOREIGN KEY(id_address) REFERENCES Address(id) ON DELETE SET
NULL,
            FOREIGN KEY(id_post_office) REFERENCES PostOffice(id) ON DELETE
SET NULL
        );
    """)

    cur.execute("""
        CREATE TABLE IF NOT EXISTS MobileOperator (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            prefix TEXT
        );
    """)

    cur.execute("""
        CREATE TABLE IF NOT EXISTS PhoneNumber (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            number TEXT,
            type TEXT,
            id_subscriber INTEGER,
            id_operator INTEGER,
            active INTEGER DEFAULT 1,
            FOREIGN KEY(id_subscriber) REFERENCES Subscriber(id) ON DELETE
CASCADE,
            FOREIGN KEY(id_operator) REFERENCES MobileOperator(id) ON DELETE
SET NULL
        );
    """)

    cur.execute("""
        CREATE TABLE IF NOT EXISTS Debt (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            id_subscriber INTEGER,
            amount REAL,
            date_start TEXT,
            deadline TEXT,
            status TEXT,
            FOREIGN KEY(id_subscriber) REFERENCES Subscriber(id) ON DELETE
CASCADE
        );
    """)

    cur.execute("""
        CREATE TABLE IF NOT EXISTS NumberChangeRequest (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            id_subscriber INTEGER,
            old_number TEXT,
            new_number TEXT,
            date_request TEXT,
            status TEXT,
            FOREIGN KEY(id_subscriber) REFERENCES Subscriber(id) ON DELETE
CASCADE
```

```
            );
        """)

        cur.execute("""
            CREATE TABLE IF NOT EXISTS SpecialService (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT,
                id_number INTEGER,
                description TEXT,
                weekday TEXT,
                time_start TEXT,
                time_end TEXT,
                FOREIGN KEY(id_number) REFERENCES PhoneNumber(id) ON DELETE
CASCADE
            );
        """)

        cur.execute("""
            CREATE TABLE IF NOT EXISTS RepairWork (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                id_address INTEGER,
                date_start TEXT,
                date_end TEXT,
                description TEXT,
                FOREIGN KEY(id_address) REFERENCES Address(id) ON DELETE CASCADE
            );
        """)

        cur.execute("""
            CREATE TABLE IF NOT EXISTS RegistrationRequest (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                login TEXT,
                password TEXT,
                status TEXT
            );
        """)

        print('БД створена!')


    print('Для тесту/демонстрації генерую рандом дані')
    create_admin()
    create_mobile_operators()
    create_special_services()
    create_random_subscribers(50)
    create_random_debts(9)
    create_random_number_change_requests(5)
    create_random_repairs(6)


def create_admin():
    with get_db() as conn:
        cur = conn.cursor()
```

```
        cur.execute("SELECT COUNT(*) FROM User")
        count = cur.fetchone()[0]

        if count == 0:
            hashed = generate_password_hash("admin")
            cur.execute("""
                INSERT INTO User (login, password, role)
                VALUES (?, ?, ?)
            """, ("admin", hashed, "admin"))
            print("[OK] Створено адміністратора з логіном admin , та паролем
admin")

def create_mobile_operators():
    operators = [
        ("Kyivstar", "067"),
        ("Kyivstar", "097"),
        ("Vodafone", "095"),
        ("Vodafone", "050"),
        ("Lifecell", "093"),
    ]

    with get_db() as conn:
        cur = conn.cursor()
        cur.execute("SELECT COUNT(*) FROM MobileOperator")
        count = cur.fetchone()[0]

        if count == 0:
            cur.executemany("""
                INSERT INTO MobileOperator (name, prefix)
                VALUES (?, ?)
            """, operators)
            print("[OK] Додано мобільних операторів")
        else:
            print("[i] Оператори вже існують")

def create_special_services():
    services = [
        ("Пожежна служба", "101", "Надзвичайна служба, тушіння пожеж", "ПН - НД",
"00:00", "23:59"),
        ("Поліція", "102", "Служба поліції", "ПН - НД", "00:00", "23:59"),
        ("Швидка допомога", "103", "Медична екстрена служба", "ПН - НД", "00:00",
"23:59"),
        ("Газова служба", "104", "Аварії газових мереж", "ПН - НД", "00:00",
"23:59"),
    ]

    with get_db() as conn:
        cur = conn.cursor()

        cur.execute("SELECT COUNT(*) FROM SpecialService")
        count = cur.fetchone()[0]

        if count > 0:
            print("[i] Спецслужби вже існують")
```

```
            return

        for name, number, desc, workday, time_start, time_end in services:
            cur.execute("""
                INSERT INTO PhoneNumber (number, type, id_subscriber,
id_operator, active)
                VALUES (?, 'service', NULL, NULL, 1)
            """, (number,))
            phone_id = cur.lastrowid

            cur.execute("""
                INSERT INTO SpecialService (name, id_number, description,
weekday, time_start, time_end)
                VALUES (?, ?, ?, ?, ?, ?)
            """, (name, phone_id, desc, workday, time_start, time_end))

        print("[OK] Створено спецслужби")

def create_random_address(conn):
    streets = [
        ("Сторожинецька", "вул."),
        ("Шевченка", "вул."),
        ("Незалежності", "проспект."),
        ("Хотинська", "вул."),
        ("Гагаріна", "вул."),
        ("Головна", "вул."),
        ("Кобиляньської", "вул."),
    ]

    street_name, street_type = random.choice(streets)
    building = str(random.randint(1, 120))
    apartment = str(random.randint(1, 20))

    cur = conn.cursor()

    cur.execute("SELECT id FROM Street WHERE name=? AND type=?", (street_name,
street_type))
    row = cur.fetchone()

    if row:
        street_id = row[0]
    else:
        cur.execute("INSERT INTO Street (name, type) VALUES (?, ?)",
(street_name, street_type))
        street_id = cur.lastrowid

    cur.execute("""
        INSERT INTO Address (id_street, building, apartment)
        VALUES (?, ?, ?)
    """, (street_id, building, apartment))

    return cur.lastrowid

def create_random_subscribers(n=5):
```

```
    firstnames = [
        "Богдан", "Олександр", "Олег", "Іван", "Дмитро",
        "Євген", "Ілля", "Микита", "Роман", "Сергій",
        "Андрій", "Юрій", "Максим", "Степан", "Арсен",
        "Тарас", "Володимир", "Петро", "Гнат", "Лев"
    ]
    lastnames = [
        "Коваленко", "Шевченко", "Ткаченко", "Іванов", "Петренко",
        "Коваль", "Маргер", "Пастух", "Гуменюк", "Мельник",
        "Бондар", "Кравець", "Савчук", "Мороз", "Гриценко",
        "Сидоренко", "Ігнатенко", "Лисенко", "Гордійчук", "Проценко"
    ]
    middlenames = [
        "Олександрович", "Ігорович", "Іванович", "Михайлович", "Андрійович",
        "Богданович", "Євгенович", "Дмитрович", "Володимирович", "Сергійович",
        "Юрійович", "Петрович", "Степанович", "Романович", "Максимович",
        "Тарасович", "Гнатович", "Олегович", "Левович", "Микитович"
    ]

    with get_db() as conn:
        cur = conn.cursor()

        cur.execute("SELECT COUNT(*) FROM Subscriber")
        if cur.fetchone()[0] > 0:
            print("[i] Абоненти вже існують")
            return

        for _ in range(n):
            ln = random.choice(lastnames)
            fn = random.choice(firstnames)
            mn = random.choice(middlenames)

            addr_id = create_random_address(conn)

            po_id = create_post_office_for_address(conn, addr_id)

            cur.execute("""
                INSERT INTO Subscriber (lastname, firstname, middlename,
id_address, id_post_office)
                VALUES (?, ?, ?, ?, ?)
            """, (ln, fn, mn, addr_id, po_id))

            sub_id = cur.lastrowid

            cur.execute("SELECT id, prefix FROM MobileOperator ORDER BY RANDOM()
LIMIT 1")
            op = cur.fetchone()
            op_id = op["id"]
            prefix = op["prefix"]

            number = f"{prefix}{random.randint(0, 9999999):07d}"

            cur.execute("""
```

```
                INSERT INTO PhoneNumber (number, type, id_subscriber,
id_operator, active)
                VALUES (?, 'mobile', ?, ?, 1)
            """, (number, sub_id, op_id))

        print(f"[OK] Створено {n} випадкових абонентів")

def create_random_number_change_requests(n):
    with get_db() as conn:
        cur = conn.cursor()

        cur.execute("SELECT COUNT(*) FROM NumberChangeRequest")
        if cur.fetchone()[0] > 0:
            print("[i] Заявки вже існують")
            return

        cur.execute("""
            SELECT PhoneNumber.number, Subscriber.id as sub_id
            FROM PhoneNumber
            JOIN Subscriber ON Subscriber.id = PhoneNumber.id_subscriber
            WHERE PhoneNumber.type = 'mobile'
        """)

        numbers = cur.fetchall()

        if not numbers:
            print("[i] Немає телефонів — заявки не створено")
            return

        for _ in range(n):
            row = random.choice(numbers)
            old_number = row["number"]
            sub_id = row["sub_id"]

            cur.execute("SELECT id, prefix FROM MobileOperator ORDER BY RANDOM()
LIMIT 1")
            op = cur.fetchone()
            prefix = op["prefix"]
            new_number = f"{prefix}{random.randint(0, 9999999):07d}"

            date_request =
f"2025-{random.randint(1,12):02d}-{random.randint(1,28):02d}"

            status = random.choice(["new", "processing", "done"])

            cur.execute("""
                INSERT INTO NumberChangeRequest (id_subscriber, old_number,
new_number, date_request, status)
                VALUES (?, ?, ?, ?, ?)
            """, (sub_id, old_number, new_number, date_request, status))

        print(f"[OK] Створено {n} заявок на зміну номера")

def create_random_repairs(n):
```

```
    with get_db() as conn:
        cur = conn.cursor()

        cur.execute("SELECT COUNT(*) FROM RepairWork")
        if cur.fetchone()[0] > 0:
            print("[i] Ремонти вже існують")
            return

        cur.execute("SELECT id FROM Address")
        addresses = [row["id"] for row in cur.fetchall()]

        if not addresses:
            print("[i] Немає адрес — ремонти не створено")
            return

        for _ in range(n):
            addr_id = random.choice(addresses)

            start = datetime.date(2025, random.randint(1, 12), random.randint(1,
28))
            duration = random.randint(1, 10)
            end = start + datetime.timedelta(days=duration)
            date_start = start.strftime("%Y-%m-%d")
            date_end = end.strftime("%Y-%m-%d")

            desc = random.choice([
                "Ремонт телефонної лінії",
                "Заміна оптоволоконного кабелю",
                "Планове обслуговування мережі",
                "Усунення аварії",
                "Реконструкція узлової точки зв’язку"
            ])

            cur.execute("""
                INSERT INTO RepairWork (id_address, date_start, date_end,
description)
                VALUES (?, ?, ?, ?)
            """, (addr_id, date_start, date_end, desc))

        print(f"[OK] Створено {n} ремонтних робіт")

def create_random_debts(n):
    with get_db() as conn:
        cur = conn.cursor()

        cur.execute("SELECT COUNT(*) FROM Debt")
        if cur.fetchone()[0] > 0:
            print("[i] Борги вже існують")
            return

        cur.execute("SELECT id FROM Subscriber")
        subscribers = [row["id"] for row in cur.fetchall()]

        if not subscribers:
```

```
            print("[i] Немає абонентів — борги не створено")
            return

        for _ in range(n):
            sub_id = random.choice(subscribers)

            amount = round(random.uniform(50, 1500), 2)

            date_start =
f"2025-{random.randint(1,12):02d}-{random.randint(1,28):02d}"
            deadline =
f"2025-{random.randint(1,12):02d}-{random.randint(1,28):02d}"

            status = random.choice(["active", "paid"])

            cur.execute("""
                INSERT INTO Debt (id_subscriber, amount, date_start, deadline,
status)
                VALUES (?, ?, ?, ?, ?)
            """, (sub_id, amount, date_start, deadline, status))

        print(f"[OK] Створено {n} боргів")
```

static / css / style.css

```css
body {
    background-color: #f5f5f5;
}

.table-hover tbody tr:hover {
    background-color: #f1f3f5;
}

h1, h2 {
    font-weight: 600;
}

.card {
    border-radius: 0.5rem;
}

.navbar-brand {
    font-weight: 600;
}

.btn {
    border-radius: 0.4rem;
}
```

static / js / script.js

```
document.addEventListener("DOMContentLoaded", () => {
   console.log("Телефонний довідник: JS завантажено.");

   const searchInput = document.getElementById("search-input");
   if (searchInput) {
       searchInput.focus();
   }

   document.querySelectorAll("[data-confirm]").forEach(el => {
       el.addEventListener("click", (e) => {
           const msg = el.getAttribute("data-confirm") || "Ви впевнені?";
           if (!confirm(msg)) {
               e.preventDefault();
           }
       });
   });
});
```

templates / admin_request.html

```
{% extends "base.html" %}
{% block content %}
<h2>Заявка на реєстрацію для отримання доступу</h2>
<table class="table table-striped mt-3">
   <thead>
       <tr>
           <th>ID</th>
           <th>Логін</th>
           <th>Статус</th>
           <th>Дії</th>
       </tr>
   </thead>
   <tbody>
   {% for r in requests %}
       <tr>
           <td>{{ r.id }}</td>
           <td>{{ r.login }}</td>
           <td>{{ r.status }}</td>
           <td>
               {% if r.status == 'new' %}
               <a href="{{ url_for('admin_approve_request', req_id=r.id) }}"
class="btn btn-sm btn-success">Схвалити</a>
               <a href="{{ url_for('admin_reject_request', req_id=r.id) }}"
class="btn btn-sm btn-danger">Відхилити</a>
               {% else %}
               <span class="text-muted">Оброблено</span>
               {% endif %}
           </td>
       </tr>
   {% endfor %}
   </tbody>
</table>
{% endblock %}
```

templates / admin_users.html

```
{% extends "base.html" %}
{% block content %}
<h2>Користувачі системи</h2>

<table class="table table-striped">
   <thead>
       <tr>
           <th>ID</th>
           <th>Логін</th>
           <th>Роль</th>
           <th>Дії</th>
       </tr>
   </thead>
   <tbody>
   {% for u in users %}
       <tr>
           <td>{{ u.id }}</td>
           <td>{{ u.login }}</td>
           <td>
               <form method="post" action="{{ url_for('admin_change_role',
user_id=u.id) }}" class="d-flex">
                   <select name="role" class="form-select form-select-sm me-2">
                   {% for r in ["guest", "user", "operator", "admin"] %}
                       <option value="{{ r }}" {% if u.role == r %}selected{%
endif %}>{{ r }}</option>
                   {% endfor %}
                   </select>
                   <button class="btn btn-sm btn-outline-secondary"
type="submit">OK</button>
               </form>
           </td>
           <td>
               <a href="{{ url_for('admin_delete_user', user_id=u.id) }}"
class="btn btn-sm btn-danger" onclick="return confirm('Видалити
користувача?')">Видалити</a>
           </td>
       </tr>
   {% endfor %}
   </tbody>
</table>
{% endblock %}
```

templates / base.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
   <meta charset="UTF-8">
   <title>Телефонний довідник</title>

   <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
         rel="stylesheet">

   <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css')
}}">
</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
 <div class="container-fluid">

   <a class="navbar-brand" href="{{ url_for('index') }}">
       📞 Телефонний довідник
   </a>

   <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
           data-bs-target="#mainNavbar" aria-controls="mainNavbar"
           aria-expanded="false" aria-label="Toggle navigation">

       <span class="navbar-toggler-icon"></span>
   </button>

   <div class="collapse navbar-collapse" id="mainNavbar">

     <ul class="navbar-nav me-auto mb-2 mb-lg-0">

       <li class="nav-item">
         <a class="nav-link" href="{{ url_for('subscribers') }}">Абоненти</a>
       </li>

       <li class="nav-item">
         <a class="nav-link" href="{{ url_for('debts') }}">Боржники</a>
       </li>

       <li class="nav-item">
```

```
        <a class="nav-link" href="{{ url_for('repairs') }}">Ремонти</a>
      </li>

      <li class="nav-item">
        <a class="nav-link" href="{{ url_for('services_list')
}}">Спецслужби</a>
      </li>

    </ul>

    <ul class="navbar-nav ms-auto">

      {% if current_user.role == "guest" %}
          <li class="nav-item">
            <a class="nav-link" href="{{ url_for('login') }}">Увійти</a>
          </li>

          <li class="nav-item">
            <a class="nav-link" href="{{ url_for('request_access')
}}">Зареєструватися</a>
          </li>

      {% else %}
          <li class="nav-item d-flex align-items-center">
            <span class="navbar-text me-3">
              👤 {{ current_user.login }} ({{ current_user.role }})
            </span>
          </li>

          {% if current_user.role in ["operator", "admin"] %}
          <li class="nav-item">
            <a class="nav-link" href="{{ url_for('sql_custom')
}}">SQL-консоль</a>
          </li>
          {% endif %}

          {% if current_user.role == "admin" %}
          <li class="nav-item">
            <a class="nav-link" href="{{ url_for('admin_users')
}}">Користувачі</a>
          </li>

          <li class="nav-item">
            <a class="nav-link" href="{{ url_for('admin_requests') }}">Заявки
1</a>
          </li>

          <li class="nav-item">
            <a class="nav-link" href="{{ url_for('requests_list') }}">Заявки
2</a>
          </li>
          {% endif %}

          <li class="nav-item">
```

```
            <a class="nav-link" href="{{ url_for('logout') }}">Вийти</a>
          </li>

      {% endif %}

    </ul>

  </div>
 </div>
</nav>

<div class="container mt-3">
 {% with messages = get_flashed_messages(with_categories=true) %}
   {% if messages %}
     {% for category, msg in messages %}
       <div class="alert alert-{{ category }} alert-dismissible fade show"
role="alert">
         {{ msg }}
         <button type="button" class="btn-close"
data-bs-dismiss="alert"></button>
       </div>
     {% endfor %}
   {% endif %}
 {% endwith %}
</div>


<div class="container my-4">
   {% block content %}
   {% endblock %}
</div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.j
s"></script>

<script src="{{ url_for('static', filename='js/script.js') }}"></script>

</body>
</html>
```

templates / debts.html

```
{% extends "base.html" %}
{% block content %}

<div class="d-flex justify-content-between align-items-center mb-3">
   <h2>Абоненти з заборгованістю</h2>
   {% if current_user.role in ['operator', 'admin'] %}
   <button class="btn btn-success mb-3"
           onclick="alert('Щоб додати борг, перейдіть у профіль потрібного
абонента і скористайтесь кнопкою «Додати борг».');">
       ➕ Додати боржника
   </button>
   {% endif %}

</div>



{% if current_user.role in ["user", "operator", "admin"] %}
<div class="card card-body shadow-sm mb-4">
   <form action="{{ url_for('debts') }}" method="get">
       <label class="form-label">Пошук боржників (підтримуються символи * та ?
для пошуку)</label>
       <div class="input-group">
           <input type="text" name="q" class="form-control" placeholder="Введіть
ім'я або прізвище або по-батькові">
           <button class="btn btn-primary">Пошук</button>
       </div>
   </form>
</div>
{% else %}
<div class="alert alert-warning">
   Пошук доступний лише після <a href="{{ url_for('login') }}">входу</a> або <a
href="{{ url_for('request_access') }}">реєстрації</a>.
</div>
{% endif %}

{% if debtors %}
<table class="table table-striped table-hover mt-3">
   <thead>
       <tr>
           <th>Абонент</th>
           <th>Загальна сума боргу</th>
```

```
            <th></th>
        </tr>
    </thead>
    <tbody>
    {% for d in debtors %}
        <tr>
            <td>{{ d.lastname }} {{ d.firstname }} {{ d.middlename }}</td>
            <td>{{ d.total_debt }}</td>
            {% if current_user and current_user.role == 'admin' or
current_user.role == 'operator' %}
            <td>
                <a class="btn btn-sm btn-primary" href="{{
url_for('subscriber_view', sub_id=d.id) }}">Редагувати</a>
            </td>
            {% endif %}
        </tr>
    {% endfor %}
    </tbody>
</table>
{% else %}
<p>Немає активних боржників.</p>
{% endif %}
{% endblock %}
```

templates / index.html

```
{% extends "base.html" %}
{% block content %}

<div class="container mt-4">

   {% if current_user.role == "guest" %}

   <div class="alert alert-info shadow-sm">
       <h3 class="mb-3">👋 Вітаємо у телефонному довіднику!</h3>
       <p>Ви увійшли у систему як <strong>гість</strong>.</p>

       <p class="mt-3">Як гість, ви можете:</p>
       <ul>
           <li>переглядати <strong>абонентів</strong></li>
           <li>переглядати <strong>ремонти</strong></li>
           <li>переглядати <strong>боржників</strong></li>
           <li>переглядати <strong>спеціальні служби</strong></li>
       </ul>

       <hr>

       <p class="fw-bold">Рекомендуємо зареєструватися!</p>
       <p>Після реєстрації ви отримаєте доступ до:</p>
       <ul>
           <li><strong>пошуку</strong> абонентів, ремонтів, боржників</li>
           <li><strong>запитам на оновлення номеру</strong></li>
           <li>[debug] 10 запитам з теми</li>
       </ul>

       <div class="mt-3">
           <a href="{{ url_for('request_access') }}" class="btn btn-success
me-2">Зареєструватися</a>
           <a href="{{ url_for('login') }}" class="btn
btn-outline-primary">Увійти</a>
       </div>
   </div>

   {% elif current_user.role == "user" %}

   <div class="alert alert-primary shadow-sm">
       <h3 class="mb-3">👋 Вітаємо, {{ current_user.login }}!</h3>
       <p>Ви увійшли як <strong>авторизований користувач</strong>.</p>
```

```
        <p class="mt-3">Вам доступно:</p>
        <ul>
            <li>перегляд та пошук абонентів, боржників, ремонтів, спецслужб</li>
            <li>запити на зміну номера</li>
        </ul>
    </div>

    {% elif current_user.role == "operator" %}
    <div class="alert alert-warning shadow-sm">
        <h3 class="mb-3">🔧 Вітаємо, оператор {{ current_user.login }}!</h3>
        <p>Ваш рівень доступу: <strong>Оператор</strong>.</p>

        <p class="mt-3">Вам доступно:</p>
        <ul>
            <li>перегляд та пошук абонентів, боржників, ремонтів, спецслужб</li>
            <li>додавання / редагування абонентів</li>
            <li>додавання / видалення номерів</li>
            <li>додавання боргів</li>
            <li>перевірка на зміну номера</li>
            <li>Довільні SQL запити</li>
        </ul>
    </div>

    {% elif current_user.role == "admin" %}

    <div class="alert alert-success shadow-sm">
        <h3 class="mb-3">🛡️ Вітаємо, адміністратор {{ current_user.login }}!</h3>
        <p>Ваш рівень доступу: <strong>Адміністратор</strong>.</p>

        <p class="mt-3">Вам доступно ВСЕ у системі:</p>
        <ul>
            <li>повний перегляд усіх даних</li>
            <li>керування користувачами</li>
            <li>опрацювання заявок на реєстрацію</li>
            <li>повний CRUD абонентів</li>
            <li>керування номерами, боргами, заявками</li>
            <li>Довільні SQL запити</li>
        </ul>
    </div>
    {% endif %}

</div>

<div class="alert alert-success shadow-sm">
    <h2 class="mb-2">ℹ️ Види запитів в інформаційній системі (10 штук) з файла
теми:</h2>
    <p><a href="{{ url_for('sql_reports') }}" class="link-primary
fw-bold">Перейти до SQL-звітів</a></p>
</div>

{% endblock %}
```

templates / login.html

```
{% extends "base.html" %}
{% block content %}
<div class="row justify-content-center">
   <div class="col-md-4">
       <h2 class="mb-3 text-center">Вхід</h2>
       <form method="post" class="card card-body shadow-sm">
           <div class="mb-3">
               <label class="form-label">Логін</label>
               <input type="text" name="login" class="form-control">
           </div>
           <div class="mb-3">
               <label class="form-label">Пароль</label>
               <input type="password" name="password" class="form-control">
           </div>
           <button class="btn btn-primary w-100" type="submit">Увійти</button>
       </form>
   </div>
</div>
{% endblock %}
```

templates / registration_request.html

```
{% extends "base.html" %}
{% block content %}
<h2>Заявка на отримання прав авторизованого користувача</h2>

<form method="post" class="mt-3" style="max-width: 400px;">
 <div class="mb-3">
   <label class="form-label">Бажаний логін</label>
   <input type="text" name="login" class="form-control" required>
 </div>
 <div class="mb-3">
   <label class="form-label">Бажаний пароль</label>
   <input type="password" name="password" class="form-control" required>
 </div>
 <button class="btn btn-primary" type="submit">Відправити заявку</button>
</form>
{% endblock %}
```

templates / repair_add.html

```
{% extends "base.html" %}
{% block content %}
<h2>Додати ремонтні роботи</h2>

<form method="post" class="card card-body shadow">

   <h5>Адреса</h5>

   <div class="mb-3">
       <label class="form-label">Тип вулиці</label>
       <input type="text" name="street_type" class="form-control" value="вул.">
   </div>

   <div class="mb-3">
       <label class="form-label">Назва вулиці</label>
       <input type="text" name="street_name" class="form-control" required>
   </div>

   <div class="mb-3">
       <label class="form-label">Будинок</label>
       <input type="text" name="building" class="form-control" required>
   </div>

   <div class="mb-3">
       <label class="form-label">Квартира</label>
       <input type="text" name="apartment" class="form-control">
   </div>

   <h5>Дати</h5>

   <div class="row">
       <div class="col-md-6 mb-3">
           <label class="form-label">Дата початку</label>
           <input type="date" name="date_start" class="form-control" required>
       </div>
       <div class="col-md-6 mb-3">
           <label class="form-label">Дата завершення</label>
           <input type="date" name="date_end" class="form-control" required>
```

```
        </div>
    </div>

    <div class="mb-3">
        <label class="form-label">Опис</label>
        <textarea name="description" class="form-control"></textarea>
    </div>

    <button class="btn btn-success">Додати</button>

</form>
{% endblock %}
```

templates / repair_edit.html

```
{% extends "base.html" %}
{% block content %}
<h2>Редагувати ремонтні роботи</h2>
<form method="post" class="card card-body shadow">
    <h5>Адреса</h5>
    <div class="mb-3">
        <label class="form-label">Тип вулиці</label>
        <input type="text" name="street_type" class="form-control" value="{{
repair.street_type }}">
    </div>

    <div class="mb-3">
        <label class="form-label">Назва вулиці</label>
        <input type="text" name="street_name" class="form-control" value="{{
repair.street_name }}" required>
    </div>

    <div class="mb-3">
        <label class="form-label">Будинок</label>
        <input type="text" name="building" class="form-control" value="{{
repair.building }}" required>
    </div>

    <div class="mb-3">
        <label class="form-label">Квартира</label>
        <input type="text" name="apartment" class="form-control" value="{{
repair.apartment }}">
    </div>

    <h5>Дати</h5>

    <div class="row">
        <div class="col-md-6 mb-3">
            <label class="form-label">Дата початку</label>
            <input type="date" name="date_start" class="form-control" value="{{
repair.date_start }}" required>
        </div>
```

```
        <div class="col-md-6 mb-3">
            <label class="form-label">Дата завершення</label>
            <input type="date" name="date_end" class="form-control" value="{{
repair.date_end }}" required>
        </div>
    </div>

    <div class="mb-3">
        <label class="form-label">Опис</label>
        <textarea name="description" class="form-control">{{ repair.description
}}</textarea>
    </div>

    <button class="btn btn-primary">Зберегти</button>
</form>
{% endblock %}
```

templates / repairs.html

```
{% extends "base.html" %}
{% block content %}

<div class="d-flex justify-content-between align-items-center mb-3">
    <h2>Ремонтні роботи</h2>
    {% if current_user and current_user.role == 'admin' %}
        <a href="{{ url_for('repair_add') }}" class="btn btn-success mb-3">➕
Додати ремонт</a>
    {% endif %}
</div>


{% if current_user.role in ["user", "operator", "admin"] %}
<div class="card card-body shadow-sm mb-4">
    <form action="{{ url_for('repairs') }}" method="get">
        <label class="form-label">Пошук за адресою:</label>
        <div class="input-group">
            <input type="text" name="q" class="form-control"
placeholder="Наприклад: Головна">
            <button class="btn btn-primary">Пошук</button>
        </div>
    </form>
</div>
{% else %}
<div class="alert alert-warning">
    Пошук доступний лише після <a href="{{ url_for('login') }}">входу</a> або <a
href="{{ url_for('request_access') }}">реєстрації</a>.
</div>
{% endif %}


<table class="table table-striped table-hover mt-3">
    <thead>
    <tr>
        <th>Адреса</th>
```

```
        <th>Дата початку</th>
        <th>Дата закінчення</th>
        <th>Опис</th>
        {% if current_user.role in ["operator", "admin"] %}
        <th>Дія</th>
        {% endif %}
    </tr>
    </thead>
    <tbody>
    {% for r in repairs %}
    <tr>
        <td>
            {% if r.street_name %}
                {{ r.street_type }} {{ r.street_name }} {{ r.building }}
                {% if r.apartment %}, кв. {{ r.apartment }}{% endif %}
            {% endif %}
        </td>
        <td>{{ r.date_start }}</td>
        <td>{{ r.date_end }}</td>
        <td>{{ r.description }}</td>
        {% if current_user.role in ["operator", "admin"] %}
        <td>
            <a href="{{ url_for('repair_edit', repair_id=r.id) }}" class="btn
btn-sm btn-primary">Редагувати</a>
            <a href="{{ url_for('repair_delete', repair_id=r.id) }}"
               onclick="return confirm('Видалити ремонтні роботи?');"
               class="btn btn-sm btn-danger">Видалити</a>
        </td>
        {% endif %}

    </tr>
    {% endfor %}
    </tbody>
</table>
{% endblock %}
```

templates / request_number_form.html

```
{% extends "base.html" %}
{% block content %}

<h2>Запит на зміну номера</h2>

<div class="card card-body shadow-sm">

   <p><strong>Старий номер:</strong> {{ old_number }}</p>

   <form method="post" action="{{ url_for('request_add') }}">
       <input type="hidden" name="subscriber_id" value="{{ sub_id }}">
       <input type="hidden" name="old_number" value="{{ old_number }}">

       <div class="mb-3">
           <label class="form-label">Новий номер</label>
           <input type="text" name="new_number" class="form-control" required
                  placeholder="Введіть новий номер">
       </div>

       <button class="btn btn-primary">Надіслати заявку</button>
   </form>

</div>

{% endblock %}
```

templates / requests.html

```
{% extends "base.html" %}
{% block content %}
<h2>Заявки на зміну номера</h2>

<table class="table table-striped table-hover mt-3">
   <thead>
       <tr>
           <th>Абонент</th>
           <th>Старий номер</th>
           <th>Новий номер</th>
           <th>Дата заявки</th>
           <th>Дії</th>
       </tr>
   </thead>
   <tbody>
   {% for r in requests %}
   <tr>
       <td>{{ r.lastname }} {{ r.firstname }} {{ r.middlename }}</td>
       <td>{{ r.old_number }}</td>
       <td>{{ r.new_number }}</td>
       <td>{{ r.date_request }}</td>
       <td>

           <a href="{{ url_for('request_approve', req_id=r.id) }}"
              class="btn btn-sm btn-success"
              onclick="return confirm('Прийняти заявку і оновити номер?');">
               ✔ Прийняти
           </a>

           <a href="{{ url_for('request_reject', req_id=r.id) }}"
              class="btn btn-sm btn-danger"
              onclick="return confirm('Відхилити заявку?');">
               ✖ Відхилити
           </a>
```

```
        </td>
    </tr>
    {% endfor %}
    </tbody>
</table>
{% endblock %}
```

templates / search_result.html

```
{% extends "base.html" %}
{% block content %}
<h2>Результати пошуку: "{{ query }}"</h2>
{% if results %}
<table class="table table-striped table-hover mt-3">
    <thead>
        <tr>
            <th>Прізвище</th>
            <th>Ім’я</th>
            <th>По-батькові</th>
            <th>Адреса</th>
            <th>Пошт.індекс</th>
            <th>Телефон</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
    {% for s in results %}
        <tr>
            <td>{{ s.lastname }}</td>
            <td>{{ s.firstname }}</td>
            <td>{{ s.middlename }}</td>
            <td>
                {% if s.street_name %}
                    {{ s.street_type }} {{ s.street_name }} {{ s.building }}
                    {% if s.apartment %}, кв. {{ s.apartment }}{% endif %}
                {% endif %}
            </td>
            <td>{{ s.office_number or '—' }}</td>
           <td>
                {{ s.phone_number or '—' }}
                {% if current_user.role == "user" and s.phone_number %}
```

```
                    <a class="btn btn-sm btn-primary ms-2" href="{{
url_for('request_number_page', sub_id=s.id, old_number=s.phone_number)
}}">✏️</a>
                {% endif %}
            </td>
            <td>
                {% if current_user and current_user.role == 'admin' or
current_user.role == 'operator' %}
                <a href="{{ url_for('subscriber_view', sub_id=s.id) }}"
class="btn btn-sm btn-outline-primary">Редагувати профіль</a>
                {% endif %}
            </td>
        </tr>
    {% endfor %}
    </tbody>
</table>
{% else %}
<p class="mt-3">Нічого не знайдено.</p>
{% endif %}
{% endblock %}
```

templates / services.html

```
{% extends "base.html" %}
{% block content %}
<h2>Спеціальні служби</h2>

<table class="table table-striped table-hover mt-3">
    <thead>
    <tr>
        <th>Назва</th>
        <th>Номер</th>
        <th>Опис</th>
        <th>Дні прийому</th>
        <th>Графік</th>
    </tr>
    </thead>
    <tbody>
    {% for s in services %}
    <tr>
        <td>{{ s.name }}</td>
        <td>{{ s.number }}</td>
        <td>{{ s.description }}</td>
        <td>{{ s.weekday }}</td>
        <td>{{ s.time_start + " - " + s.time_end}}</td>
    </tr>
    {% endfor %}
    </tbody>
</table>
{% endblock %}
```

templates / sql_custom.html

```
{% extends "base.html" %}
{% block content %}
<h2>SQL-консоль (тільки для оператора / адміністратора)</h2>

<form method="post" class="mb-3">
 <div class="mb-3">
   <label class="form-label">SQL-запит</label>
   <textarea name="sql_text" rows="5" class="form-control">{{ sql_text
}}</textarea>
 </div>
 <button class="btn btn-primary" type="submit">Виконати</button>
</form>

{% if cols %}
 <table class="table table-striped table-sm">
   <thead>
     <tr>
       {% for c in cols %}
         <th>{{ c }}</th>
       {% endfor %}
     </tr>
   </thead>
   <tbody>
     {% for row in rows %}
       <tr>
         {% for v in row %}
           <td>{{ v }}</td>
         {% endfor %}
       </tr>
     {% endfor %}
   </tbody>
```

```
 </table>
{% endif %}
{% endblock %}
```

templates / sql_reports.html

```
{% extends "base.html" %}
{% block content %}

<h2>SQL-звіти</h2>

<form method="post" class="card card-body shadow-sm mb-4">
   <label class="form-label">Оберіть звіт:</label>
   <select class="form-select" name="query_id">
       <option value="1">1. Графік роботи спеціальних служб</option>
       <option value="2">2. Абоненти з заявками на зміну номера</option>
       <option value="3">3. Ремонтні роботи</option>
       <option value="4">4. Номери спеціальних служб</option>
       <option value="5">5. Абоненти за ПІБ</option>
       <option value="6">6. Абоненти за операторами</option>
       <option value="7">7. Абоненти з боргами</option>
       <option value="8">8. Абоненти, які змінили адресу та номер</option>
       <option value="9">9. Абоненти за вулицею</option>
       <option value="10">10. Вулиці з найбільшою кількістю абонентів</option>
   </select>

   {% if selected == "5" %}
       <div class="row mt-3">
           <div class="col">
               <input class="form-control" name="lastname"
placeholder="Прізвище">
           </div>
           <div class="col">
               <input class="form-control" name="firstname" placeholder="Ім’я">
           </div>
       </div>
   {% endif %}
```

```
    {% if selected == "9" %}
    <div class="mt-3">
        <label class="form-label">Вулиця:</label>
        <input class="form-control" name="street_name" placeholder="Наприклад:
Шевченка">
    </div>
    {% endif %}

    <button class="btn btn-primary mt-3">Виконати</button>
</form>

{% if result %}
<div class="card card-body shadow-sm mt-4">
    <h4 class="mb-3">Результати:</h4>

    {# --- Для запитів 5 і 9 є (rows, count) --- #}
    {% if selected in ["5", "9"] %}
        {% set rows = result[0] %}
        {% set cnt = result[1] %}

        <p><strong>Загальна кількість:</strong> {{ cnt }}</p>
    {% else %}
        {% set rows = result %}
        {% set cnt = None %}
    {% endif %}

    {% if rows and rows|length > 0 %}
        <table class="table table-bordered table-striped table-sm">
            <thead>
                <tr>
                    {% for col in rows[0].keys() %}
                        <th>{{ col }}</th>
                    {% endfor %}
                </tr>
            </thead>
            <tbody>
                {% for row in rows %}
                <tr>
                    {% for col in row.keys() %}
                        <td>{{ row[col] }}</td>
                    {% endfor %}
                </tr>
                {% endfor %}
            </tbody>
        </table>
    {% else %}
        <p>Немає даних.</p>
    {% endif %}
</div>
{% endif %}


{% endblock %}
```

templates / subscriber_form.html

```
{% extends "base.html" %}
{% block content %}
{% if mode == 'add' %}
<h2>Новий абонент</h2>
{% else %}
<h2>Редагування абонента</h2>
{% endif %}

<form method="post" class="card card-body shadow-sm">
   <div class="row">
   <div class="col-md-6">
       <h5>Персональні дані</h5>
       <div class="mb-3">
           <label class="form-label">Прізвище</label>
           <input type="text" name="lastname" class="form-control"
                  value="{{ subscriber.lastname if subscriber else '' }}">
       </div>
       <div class="mb-3">
           <label class="form-label">Ім’я</label>
           <input type="text" name="firstname" class="form-control"
                  value="{{ subscriber.firstname if subscriber else '' }}">
       </div>
       <div class="mb-3">
           <label class="form-label">По-батькові</label>
           <input type="text" name="middlename" class="form-control"
                  value="{{ subscriber.middlename if subscriber else '' }}">
       </div>
   </div>
```

```
    <div class="col-md-6">
        <h5>Адреса</h5>
        <div class="mb-3">
            <label class="form-label">Тип вулиці</label>
            <input type="text" name="street_type" class="form-control"
                   value="{{ subscriber.street_type if subscriber else 'вул.'
}}">
        </div>

        <div class="mb-3">
            <label class="form-label">Назва вулиці</label>
            <input type="text" name="street_name" class="form-control"
                   value="{{ subscriber.street_name if subscriber else '' }}">
        </div>

        <div class="mb-3">
            <label class="form-label">Будинок</label>
            <input type="text" name="building" class="form-control"
                   value="{{ subscriber.building if subscriber else '' }}">
        </div>

        <div class="mb-3">
            <label class="form-label">Квартира</label>
            <input type="text" name="apartment" class="form-control"
                   value="{{ subscriber.apartment if subscriber else '' }}">
        </div>

        <h5 class="mt-3">Поштове відділення</h5>

        <div class="mb-3">
            <label class="form-label">Номер відділення</label>
            <input type="text" name="post_office" class="form-control"
                  value="{{ subscriber.office_number if subscriber else '' }}">
        </div>
    </div>
</div>


    <button type="submit" class="btn btn-primary mt-2">Зберегти</button>
</form>
{% endblock %}
```

templates / subscriber_view.html

```
{% extends "base.html" %}
{% block content %}
<div class="d-flex justify-content-between align-items-center mb-3">
   <h2>Абонент: {{ subscriber.lastname }} {{ subscriber.firstname }} {{
subscriber.middlename }}</h2>
   <div>
       {% if current_user and current_user.role == 'admin' %}
           <a href="{{ url_for('subscriber_edit', sub_id=subscriber.id) }}"
             class="btn btn-sm btn-primary">Редагувати</a>
           <a href="{{ url_for('subscriber_delete', sub_id=subscriber.id) }}"
              class="btn btn-sm btn-danger"
              data-confirm="Видалити абонента?">Видалити</a>
       {% endif %}
   </div>
</div>

<div class="card mb-3 shadow-sm">
   <div class="card-body">
       <p><strong>Адреса:</strong>
           {% if subscriber.street_name %}
               {{ subscriber.street_type }} {{ subscriber.street_name }} {{
subscriber.building }}
               {% if subscriber.apartment %}, кв. {{ subscriber.apartment }}{%
endif %}
           {% else %}
               (не вказано)
           {% endif %}
       </p>
       <p><strong>Поштове відділення:</strong>
```

```
            {% if subscriber.office_number %}
                № {{ subscriber.office_number }}
            {% else %}
                (не вказано)
            {% endif %}
        </p>
    </div>
</div>

<div class="row">
    <div class="col-md-6">
        <h4>Телефони</h4>
        <table class="table table-sm table-bordered">
            <thead>
            <tr>
                <th>Номер</th>
                <th>Тип</th>
                <th>Оператор</th>
                <th>Дії</th>
            </tr>
            </thead>
            <tbody>
            {% for p in phones %}
            <tr>
                <td>{{ p.number }}</td>
                <td>{{ p.type }}</td>
                <td>{{ p.operator_name }}</td>
                {% if current_user.role in ["operator", "admin"] %}
                <td>
                    <a href="{{ url_for('phone_delete', sub_id=subscriber.id,
phone_id=p.id) }}"
                       class="btn btn-sm btn-danger"
                       onclick="return confirm('Видалити номер?');">
                        Видалити
                    </a>
                </td>
                {% endif %}
            </tr>
            {% endfor %}
            </tbody>
        </table>

        <form method="post" action="{{ url_for('phone_add', sub_id=subscriber.id)
}}"
              class="card card-body">
            <h5>Додати телефон</h5>
            <div class="mb-2">
                <label class="form-label">Номер</label>
                <input type="text" name="number" class="form-control" required>
            </div>
            <div class="mb-2">
                <label class="form-label">Тип</label>
                <select name="type" class="form-select">
                    <option value="mobile">Мобільний</option>
```

```
                    <option value="landline">Стаціонарний</option>
                    <option value="service">Службовий</option>
                </select>
            </div>
            <div class="mb-2">
                <label class="form-label">ID оператора (опційно)</label>
                <input type="text" name="operator_id" class="form-control">
            </div>
            <button type="submit" class="btn btn-primary btn-sm
mt-1">Додати</button>
        </form>
    </div>

    <div class="col-md-6">
        <h4>Борги</h4>
        {% if debts %}
        <table class="table table-sm table-bordered">
            <thead>
            <tr>
                <th>Сума</th>
                <th>Початок</th>
                <th>Дедлайн</th>
                {% if current_user.role in ["operator", "admin"] %}
                <th>Дії</th>
                {% endif %}
            </tr>
            </thead>
            <tbody>
            {% for d in debts %}
            <tr>
                <td>{{ d.amount }}</td>
                <td>{{ d.date_start }}</td>
                <td>{{ d.deadline }}</td>
                {% if current_user.role in ["operator", "admin"] %}
                <td>
                    <a href="{{ url_for('debt_delete', sub_id=subscriber.id,
debt_id=d.id) }}"
                       class="btn btn-sm btn-danger"
                       onclick="return confirm('Видалити борг?');">
                        Видалити
                    </a>
                </td>
                {% endif %}
            </tr>
            {% endfor %}
            </tbody>
        </table>
        {% else %}
        <p>Боргів немає.</p>
        {% endif %}

        <form method="post" action="{{ url_for('debt_add', sub_id=subscriber.id)
}}"
              class="card card-body">
```

```
            <h5>Додати борг</h5>
            <div class="mb-2">
                <label class="form-label">Сума</label>
                <input type="number" step="0.01" name="amount"
class="form-control" required>
            </div>
            <div class="mb-2">
                <label class="form-label">Дата початку</label>
                <input type="date" name="date_start" class="form-control">
            </div>
            <div class="mb-2">
                <label class="form-label">Дедлайн</label>
                <input type="date" name="deadline" class="form-control">
            </div>
            <button type="submit" class="btn btn-primary btn-sm mt-1">Додати
борг</button>
        </form>
    </div>

</div>
{% endblock %}
```

templates / subscribers

```
{% extends "base.html" %}
{% block content %}

<div class="d-flex justify-content-between align-items-center mb-3">
    <h2>Абоненти</h2>
    {% if current_user and current_user.role == 'admin' %}
        <a href="{{ url_for('subscriber_add') }}" class="btn btn-success">➕
Додати абонента</a>
    {% endif %}
</div>

{% if current_user.role in ["user", "operator", "admin"] %}
<div class="card card-body shadow-sm mb-4">
    <form action="{{ url_for('search') }}" method="get">
        <label class="form-label">Пошук абонентів по ПІБ / номеру телефона
(символи * і ? підтримуються):</label>
        <div class="input-group">
            <input type="text" name="q" class="form-control" placeholder="Введіть
дані для пошуку...">
            <button class="btn btn-primary">Пошук</button>
        </div>
    </form>
</div>
{% elif current_user.role == "guest" %}
```

```
<div class="alert alert-warning">
   Пошук доступний лише після <a href="{{ url_for('login') }}">входу</a> або <a
href="{{ url_for('request_access') }}">реєстрації</a>.
</div>
{% endif %}

<table class="table table-striped table-hover">
   <thead>
       <tr>
           <th>Прізвище</th>
           <th>Ім’я</th>
           <th>По-батькові</th>
           <th>Адреса</th>
           <th>Пошт.індекс</th>
           <th>Телефон</th>
           <th></th>
       </tr>
   </thead>
   <tbody>
   {% for s in subscribers %}
       <tr>
           <td>{{ s.lastname }}</td>
           <td>{{ s.firstname }}</td>
           <td>{{ s.middlename }}</td>
           <td>
               {% if s.street_name %}
                   {{ s.street_type }} {{ s.street_name }} {{ s.building }}
                   {% if s.apartment %}, кв. {{ s.apartment }}{% endif %}
               {% endif %}
           </td>
           <td>{{ s.office_number or '—' }}</td>
           <td>
               {{ s.main_phone or '—' }}
               {% if current_user.role == "user" and s.main_phone %}
                   <a class="btn btn-sm btn-primary ms-2" href="{{
url_for('request_number_page', sub_id=s.id, old_number=s.main_phone) }}">✏️</a>
               {% endif %}
           </td>
           <td>
               {% if current_user and current_user.role == 'admin' or
current_user.role == 'operator' %}
               <a href="{{ url_for('subscriber_view', sub_id=s.id) }}"
class="btn btn-sm btn-primary">Редагувати</a>
               {% endif %}
           </td>
       </tr>
   {% endfor %}
   </tbody>
</table>
{% endblock %}
```