НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

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

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення для допомоги громадянам України з інфекційними захворюваннями у Німеччині**»**

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ (підпис) **Ганна ВАЙГАНГ**

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ (підпис) **Іван ПАРХОМЕНКО**

Виконав _____ (підпис) **Микола БІГУН**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

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

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Бігуну Миколі Миколайовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення для допомоги громадянам України з інфекційними захворюваннями у Німеччині»

затверджена наказом від «19» грудня 2025 р. № 3204 «С»
Термін подання завершеної роботи на кафедру «___» _____ 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
тема роботи, матеріали аналізу предметної області, вимоги до вебзастосунку, сценарії роботи клієнта, адміністратора й виконавця, структура бази даних
Перелік питань, що підлягають дослідженню:
аналіз предметної області, вимоги до системи, проєктування БД, реалізація інтерфейсу, службових панелей, тестування, рекомендації з експлуатації.
Перелік графічних документів (за необхідності).
Презентація програмного забезпечення
Дата видачі завдання «___» _____ 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Іван ПАРХОМЕНКО**

Завдання взяв до виконання _____ (підпис) **Микола БІГУН**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Таблиця 1 – Перелік умовних позначень

Позначення	Розшифрування
API	Application Programming Interface, програмний інтерфейс прикладної системи.
БД	База даних.
СУБД	Система управління базами даних.
ER	Entity–Relationship, модель сутностей і зв’язків.
UML	Unified Modeling Language, уніфікована мова моделювання.
REST	Архітектурний підхід до організації HTTP API.
RBAC	Role-Based Access Control, контроль доступу на основі ролей.
OSM	OpenStreetMap, відкрита картографічна платформа.
UI	User Interface, користувацький інтерфейс.
GDPR	Загальний регламент захисту даних Європейського Союзу.

ВСТУП

Інформаційні системи у сфері громадського здоров’я мають не лише інформувати, а й допомагати людині швидко зорієнтуватися в доступних сервісах. Для українців у Німеччині це особливо актуально через мовний бар’єр, розподіленість медичних і соціальних послуг та потребу в перекладі під час звернення до фахівців.

Актуальність теми полягає в потребі єдиного вебзастосунку, що об’єднує каталог організацій, карту, первинний скринінг, подання звернення, заявку на переклад і зворотний зв’язок. Таке рішення скорочує час пошуку допомоги та робить маршрут дій зрозумілішим для користувача.

Мета роботи — розробити DATACHECK Connect: вебзастосунок для пошуку релевантної допомоги, перегляду організацій на карті, створення звернень, координації асистованого перекладу та адміністрування заявок.

Об’єктом дослідження є процес цифрової маршрутизації громадян України до медичних, соціальних, юридичних і психологічних сервісів у Німеччині. Предметом дослідження є програмні засоби, моделі даних, інтерфейси та алгоритми веб-системи, що автоматизує пошук допомоги, приймання звернень і службову обробку заявок.

Для досягнення мети застосовано системний аналіз, об’єктно-орієнтоване та ER-моделювання, проєктування реляційної БД, компонентне проєктування, ручне функціональне тестування і аналіз користувацьких сценаріїв. Реалізація виконана з використанням JavaScript, Node.js, Express.js, PostgreSQL, pg, dotenv, HTML, CSS, Leaflet та OpenStreetMap.

Практичне значення полягає у створенні прототипу платформи, що поєднує довідковий каталог, електронні форми звернення, адмін-панель і кабінет виконавця. Архітектура допускає розширення довідників, ролей, аналітики та механізмів захисту даних.

Пояснювальна записка складається зі вступу, чотирьох основних розділів, висновків, списку використаних джерел і додатків. У першому розділі виконано

системний аналіз предметної області. У другому розділі наведено проєктування інформаційного та програмного забезпечення. Третій розділ присвячено реалізації бази даних, серверної та клієнтської частин. У четвертому розділі подано тестування, вимоги до середовища, склад інсталяційного пакету та рекомендації щодо експлуатації.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

DATACHECK Connect охоплює взаємодію клієнта, організацій-надавачів послуг, адміністратора та виконавця перекладу. Клієнт працює з простими сценаріями: каталогом, картою, скринінгом, заявкою на переклад і формою відгуку.

Потреби клієнта можуть поєднувати медичні, соціальні, юридичні та психологічні питання. Тому система не обмежується одним довідником, а пов’язує категорію допомоги з конкретною організацією, містом і контактами.

У проєкті виділено чотири основні категорії підтримки: медична, соціальна, юридична та психологічна. Медична категорія включає інфекційні консультації, підбір клініки, запис на прийом і переклад під час медичного візиту. Соціальна категорія описує супровід сімей, навігацію по виплатах, документи, житло та гарячі лінії. Юридична категорія пов’язана з питаннями страхування, правового статусу, звернень до державних установ. Психологічна категорія охоплює індивідуальну та групову підтримку.

Довідник організацій у системі містить назву, контакти, статус, сайт, адресу, місто та координати. Координати використовуються для побудови інтерактивної карти, де користувач може не лише побачити точку організації, а й натиснути на послуги, що надаються цією організацією. Такий підхід поєднує географічний пошук і змістовний пошук, що є важливим для користувачів, які орієнтуються за містом перебування.

Публічна частина не містить службових кнопок: адміністративна панель і кабінет виконавця відкриваються лише за прямими посиланнями `/admin` та `/executor`. Це відокремлює сценарії клієнта від службових режимів.

Адміністратор відповідає за перегляд звернень, контроль задач перекладу, додавання й редагування послуг, створення й редагування користувачів системи, а також видалення звернень, які не мають залишатися в журналі. Виконавець працює в окремому кабінеті, бачить доступні заявки на переклад,

бере їх у роботу та після виконання завершує заявку. Таким чином, життєвий цикл задачі має зрозумілий статусний перехід і не потребує ручного втручання в базу даних.

Форма відгуку розміщена окремо в лівому меню. Клієнт може поставити оцінку та залишити коментар, а адміністратор бачить середній показник якості у службовій панелі.

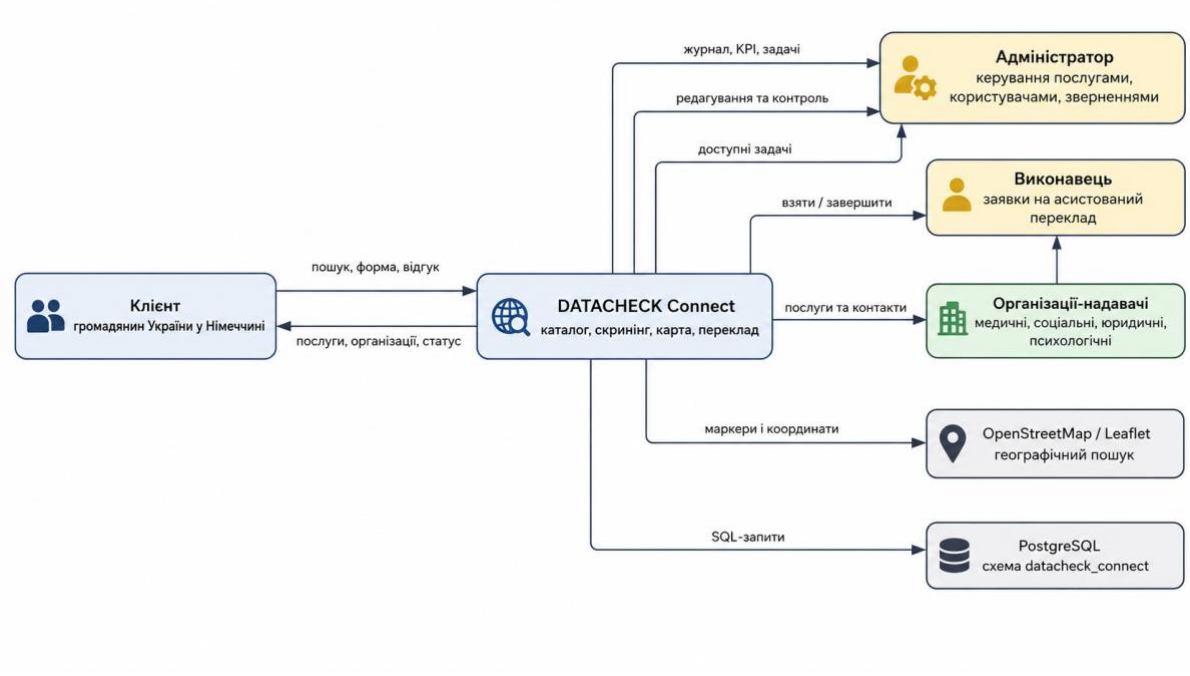

Рис. 1.1 – Контекстна діаграма взаємодії DATACHECK Connect

Таблиця 1.1 – Основні ролі системи

Роль	**Мета роботи в системі**	**Основні дії**
Клієнт	Знайти потрібну допомогу та подати звернення.	Пошук послуг, перегляд карти, скринінг, заявка на переклад, відгук.
Адміністратор	Координувати каталог, звернення і службових користувачів.	Перегляд журналу, редагування послуг, створення користувачів, видалення звернень.
Виконавець	Опрацювати задачі асистованого перекладу.	Авторизація, перегляд доступних заявок, взяття в роботу, завершення заявки.
Організація	Надавати одну або кілька послуг у конкретному місті.	Наявність контакту, адреси, координат і прив’язки до послуг.

1.2 Аналіз вимог до програмної системи

Вимоги до DATACHECK Connect сформовано на основі сценаріїв роботи клієнта, адміністратора та виконавця. Система повинна мати публічну частину для клієнта, у якій доступні тільки ті функції, що не потребують службових прав. До таких функцій належать перегляд головної сторінки, каталог послуг, карта організацій, форма скринінгу, форма заявки на переклад і форма відгуку. Публічний інтерфейс має бути мультимовним, оскільки цільова аудиторія може використовувати українську, англійську або німецьку мову.

Функціональна вимога до каталогу полягає у можливості відображати послуги з урахуванням категорії, міста, формату надання та ключового слова. У проєкті реалізовано фільтрацію за медичною, соціальною, юридичною і психологічною категоріями, містами Berlin, Hamburg, München, Köln і Leipzig, а також форматами online, offline і hybrid. Кожна послуга має назву, опис, організацію, контакти, адресу й доступні мови.

Карта повинна бути інтерактивною та працювати на основі OpenStreetMap через Leaflet. Вимога полягає не лише у відображенні маркерів, а й у можливості натискати на послуги всередині організацій. Тому маркер містить popup із назвою організації, адресою та списком кнопок послуг. Натискання на послугу відкриває деталі, що зв’язує картографічний сценарій з довідковим каталогом.

Форма скринінгу повинна збирати структуровані дані, достатні для створення клієнта, згоди, звернення та запису скринінгу. Важливо, що згода на обробку даних є окремим елементом сценарію, оскільки система працює з персональною та чутливою інформацією. Після відправлення форми дані мають зберігатися в PostgreSQL, а нове звернення повинне відображатися в адмін-панелі.

Форма асистованого перекладу повинна створювати задачу, яка доступна в службовому контурі. Користувач вказує псевдонім або ім’я, місто, дату, час, спеціалізацію лікаря, мову перекладу та коментар. Після відправлення система створює клієнта, звернення і translation_request. Далі виконавець може взяти

задачу в роботу, а після виконання завершити її. Це забезпечує повний цикл від створення до закриття задачі.

Адмін-панель повинна мати авторизацію, KPI-блок, журнал звернень, фільтри, таблицю задач перекладу, форму додавання та редагування послуг, а також форму створення і редагування користувачів системи. Окрема панель адміністратора користувачів не використовується: функції керування користувачами перенесено в основну адмін-панель, що спрощує службову структуру.

Кабінет виконавця повинен мати мінімальний набір функцій, необхідний для роботи із заявками на переклад. Виконавець бачить доступні задачі, власні задачі, кількість завершених заявок, кнопки взяття в роботу та завершення. Інші адміністративні можливості виконавцю не потрібні, тому в його інтерфейсі не відображаються керування послугами, користувачами або видалення звернень.

Нефункціональні вимоги охоплюють зрозумілість інтерфейсу, адаптивність до екранів різного розміру, збереження цілісності даних, захист службових маршрутів, контроль ролей, використання конфігурації через `.env`, можливість локального запуску, підтримку Docker PostgreSQL і придатність структури до подальшого розширення.

Таблиця 1.2 – Функціональні вимоги до системи

Код	**Вимога**	**Реалізація в проєкті**
FR-1	Пошук і фільтрація послуг	Каталог послуг у public/app.js, дані з /api/services та /api/bootstrap.
FR-2	Інтерактивна карта організацій	Leaflet + OpenStreetMap, функції initLeafletMap і renderMap.
FR-3	Онлайн-скринінг	POST /api/screening, таблиці client, consent, client_request, screening.
FR-4	Асистований переклад	POST /api/translation-tasks, таблиця translation_request, кабінет виконавця.
FR-5	Адміністрування послуг	POST /api/admin/services, PUT /api/admin/services/:id.
FR-6	Керування користувачами в адмін-панелі	POST /api/admin/users, PUT /api/admin/users/:id.

Код	Вимога	Реалізація в проєкті
FR-7	Видалення звернень	DELETE /api/admin/requests/:id.
FR-8	Завершення заявки виконавцем	POST /api/executor/tasks/:id/complete.
FR-9	Форма відгуку	Окремий блок лівого меню, POST /api/feedbacks.

1.3 Моделювання предметної області

Під час моделювання предметної області виділено сутності, які відображають головні об’єкти системи: клієнт, згода, звернення, скринінг, категорія підтримки, послуга, організація, локація, користувач системи, заявка на переклад і відгук. Таке виділення відповідає фактичній структурі бази даних і програмним модулям, оскільки кожна сутність має власну роль у життєвому циклі звернення.

Клієнт створює звернення і може залишити відгук. Згода фіксує правову підставу для обробки персональних даних. Звернення є центральною сутністю процесу, оскільки воно має статус, пріоритет, категорію, зв’язок із клієнтом і можливого відповідального користувача. Скринінг деталізує потребу звернення і може містити JSON-відповіді, які не завжди зручно жорстко нормалізувати в окремі колонки.

Категорія підтримки групує послуги та допомагає будувати фільтри. Послуга описує конкретний вид допомоги, але сама по собі не містить місця надання. Для цього використано зв’язок із організацією і локацією. Організація може надавати багато послуг, а одна послуга може надаватися різними організаціями, тому потрібна асоціативна таблиця organization_service_item.

Користувач системи представляє адміністратора, координатора, перекладача або переглядача. У поточній реалізації службові ролі використовуються для контролю доступу до адмін-панелі та кабінету виконавця. Заявка на переклад пов’язана із зверненням і виконавцем, має

власний статус і дати. Відгук пов’язаний з клієнтом і зверненням, що дозволяє аналізувати якість взаємодії в контексті конкретного кейсу.

Поведенкова модель системи описує послідовність дій від відкриття інтерфейсу до завершення звернення. Спершу клієнт обирає потрібний сценарій: пошук послуг, карту, скринінг або переклад. Якщо користувач створює звернення, система реєструє дані в БД. Якщо потрібен переклад, формується окрема задача. Далі службові користувачі змінюють статуси, а клієнт може залишити відгук.

Важливою частиною моделі є життєвий цикл статусів. Для звернень і задач використовуються статуси, що відображають реальний стан процесу: нове, у скринінгу, у роботі, виконане, закрите або скасоване. Завдяки цьому адмін-панель і кабінет виконавця можуть відображати різні списки задач без складних додаткових правил.

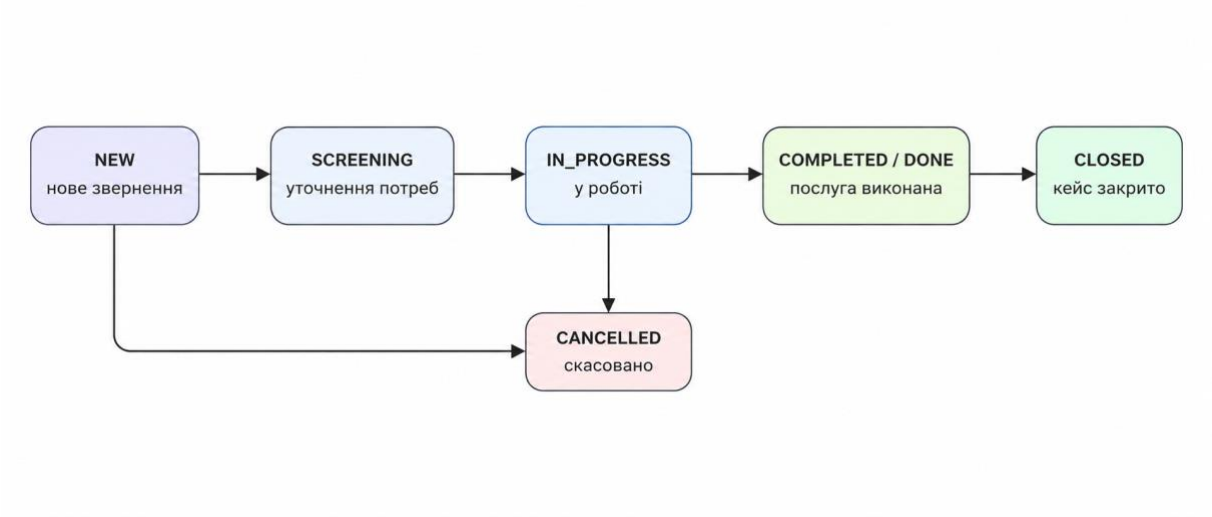

Рис. 1.2 – Узагальнений життєвий цикл статусів звернення

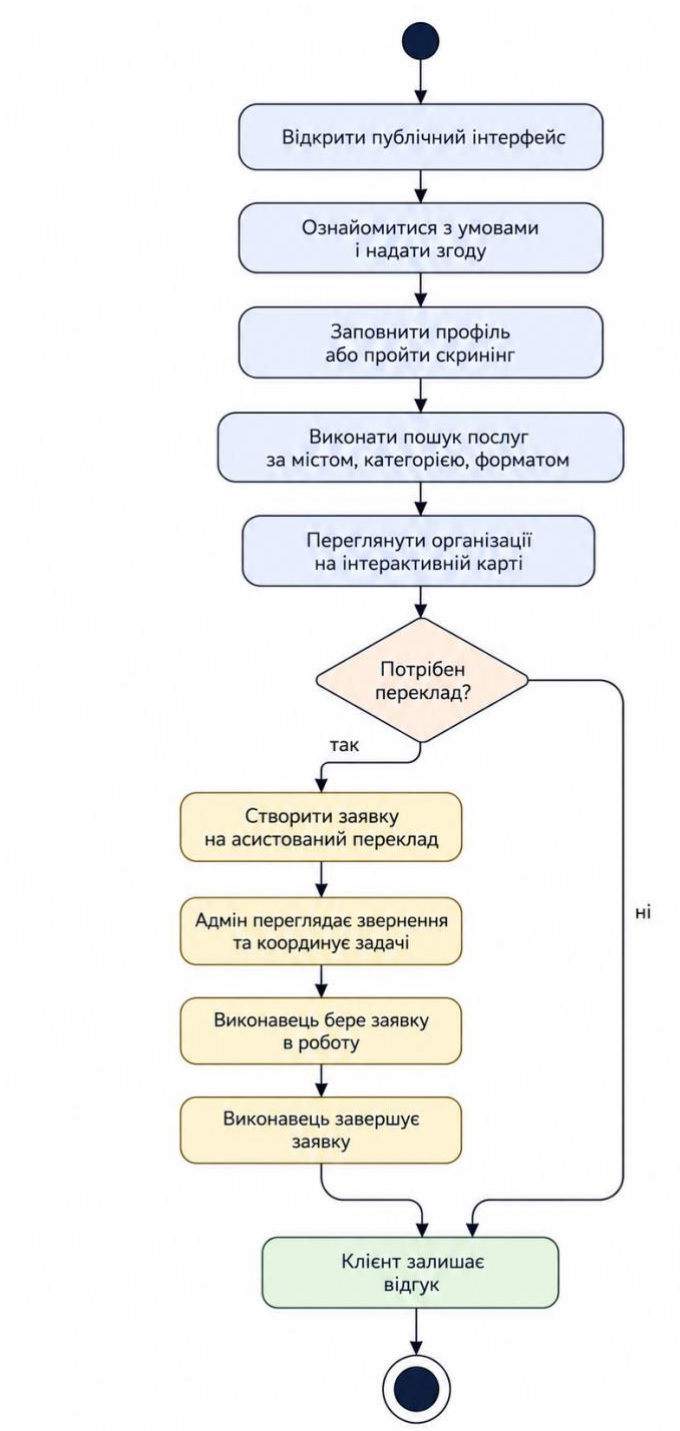

Рис. 1.3 – Діаграма активності роботи клієнта та службових ролей

1.4 Огляд інформаційних джерел та існуючих рішень

Під час аналізу предметної області було розглянуто типові підходи до побудови сервісів пошуку допомоги, довідників організацій, форм електронного звернення та картографічних каталогів. Існуючі рішення зазвичай розв’язують окрему частину задачі: довідник організацій надає контакти, карта

показує географію, форма звернення приймає дані, а службова CRM або таблиця допомагає координатору обробляти кейси. Недоліком такого підходу є розділення даних між різними інструментами.

DATACHECK Connect орієнтований на об’єднання цих функцій у єдиному застосунку. Каталог послуг, карта, скринінг, переклад, відгук, адмін-панель і кабінет виконавця працюють з однією базою даних. Це зменшує дублювання записів, спрощує контроль статусів і дозволяє координатору бачити більш повну картину роботи з клієнтом.

Для картографічної частини обрано OpenStreetMap і Leaflet. Такий вибір пояснюється відкритістю картографічних даних, доступністю бібліотеки, простотою інтеграції в статичний веб-інтерфейс і достатньою функціональністю для навчального та прототипного проєкту. Leaflet дозволяє створювати карту, додавати тайловий шар, маркери, popup-вікна, керувати масштабом і центром карти.

Для серверної частини використано Node.js і Express.js. Цей підхід є доцільним для прототипу, оскільки дозволяє швидко створити REST API, віддавати статичні файли, налаштовувати middleware, обробляти JSON-запити і підключатися до PostgreSQL через бібліотеку pg. Для конфігурації використано dotenv, що відокремлює налаштування середовища від коду.

PostgreSQL обрано як реляційну СУБД, оскільки система має чіткі сутності й зв’язки: клієнти, звернення, послуги, організації, користувачі, переклади і відгуки. Реляційна модель дозволяє реалізувати зовнішні ключі, обмеження, унікальність, індекси та транзакційність. Для напівструктурованих відповідей скринінгу використовується JSONB, що поєднує нормалізовану основу й гнучкість анкети.

У порівнянні з повноцінними CRM-системами, розроблений застосунок є легшим і точніше відповідає предметній задачі. Він не перевантажений універсальними модулями продажів або маркетингу, а зосереджений на маршрутизації допомоги, перекладі та довіднику організацій. У порівнянні з

простим статичним сайтом, DATACHECK Connect має реальну базу даних, службові панелі та API для зміни стану системи.

Таблиця 1.3 – Порівняння підходів до реалізації

Підхід	Переваги	Недоліки для задачі
Статичний довідник	Простий у створенні, не потребує сервера.	Не забезпечує звернення, статуси, службову обробку і відгуки.
Окрема карта організацій	Зручно показує географію.	Не містить повного циклу заявки і ролей.
Універсальна CRM	Має готові ролі, кейси й звіти.	Може бути надмірною та складною для клієнтського публічного сценарію.
DATACHECK Connect	Поєднує каталог, карту, скринінг, переклад, адмін-панель і БД.	Потребує підтримки сервера, бази даних і актуалізації довідників.

1.5 Постановка завдання

З урахуванням аналізу предметної області поставлено завдання створити програмну систему, яка надає клієнтові простий публічний інтерфейс, а службовим користувачам — окремі робочі вікна для адміністрування і виконання задач. Система повинна працювати як веб-застосунок, запускатися локально через Node.js, зберігати дані в PostgreSQL і мати можливість ініціалізації структури БД та початкових даних.

Необхідно реалізувати публічний каталог послуг із фільтрами, мультимовні тексти інтерфейсу, інтерактивну карту OpenStreetMap, форму скринінгу, форму заявки на асистований переклад, форму відгуку в лівому меню, приховані від головної сторінки службові маршрути `/admin` і `/executor`, авторизацію за логіном і паролем, адмін-панель із керуванням послугами, користувачами і зверненнями, а також кабінет виконавця із завершенням заявки.

До складу серверної частини повинні входити маршрути API для отримання початкових даних, перевірки стану БД, авторизації, створення

скринінгу, створення задачі перекладу, збереження відгуку, взяття задачі виконавцем, завершення задачі, додавання й редагування послуги, додавання й редагування користувача, видалення звернення. Кожна службова операція повинна перевіряти роль користувача.

До складу інформаційного забезпечення повинні входити таблиці клієнтів, згод, категорій, послуг, організацій, адрес, користувачів системи, звернень, скринінгу, рекомендованих послуг, зв’язку організацій з послугами, заявок перекладу та відгуків. Фізична схема повинна мати первинні й зовнішні ключі, обмеження цілісності, перелічувані типи, індекси та тригери для службового поля updated_at.

Результатом виконання завдання має бути працюючий прототип DATACHECK Connect із пояснювальною запискою, що описує предметну область, вимоги, моделі, технології, структуру проєкту, базу даних, програмні модулі, тестування, запуск і можливості подальшого розвитку.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Інформаційна модель DATACHECK Connect побудована навколо сутності client_request, яка відображає звернення клієнта. Звернення пов’язане з клієнтом, категорією підтримки та за потреби з відповідальним користувачем системи. Така структура дозволяє однаково обробляти звернення, створені зі скринінгу, із форми перекладу або службовим користувачем.

Сутність client містить профіль клієнта: зовнішній код, ім’я, прізвище, дату народження, телефон, email, бажану мову, країну, місто й службові нотатки. У прототипі частина клієнтів може мати псевдонім, що зменшує кількість обов’язкових персональних даних. Окрема сутність consent фіксує факт надання згоди, її версію, правову підставу, дату підтвердження та можливе відкликання.

Каталог підтримки складається із support_category, service_item, organization, address і organization_service_item. Категорія групує послуги, послуга описує вид допомоги, організація представляє надавача, адреса містить географічні дані, а асоціативна таблиця organization_service_item розв’язує зв’язок багато-до-багатьох між організаціями і послугами.

Операційний контур представлений таблицями system_user, translation_request і client_request. Користувач системи має роль, активність, логін, хеш пароля та контактні дані. Заявка на переклад пов’язана зі зверненням і виконавцем, має мови перекладу, запланований час, статус і дату завершення. Така структура підтримує сценарій “створено — взято в роботу — завершено”.

Відгук зберігається в таблиці feedback і має зв’язок з клієнтом та зверненням. Це дозволяє не лише рахувати середній рейтинг, а й аналізувати якість взаємодії в контексті конкретної послуги або категорії. Для зв’язку звернення з рекомендованими послугами використано request_service, що дає змогу зберігати причину рекомендації та порядок пріоритету.

Логічна модель нормалізує повторювані дані та відокремлює довідники від операційних записів. Послуга не дублює інформацію організації, організація не дублює список послуг у текстовому полі, а координати винесені в address. Завдяки цьому система може додавати нові організації, послуги та локації без зміни основної логіки звернень.

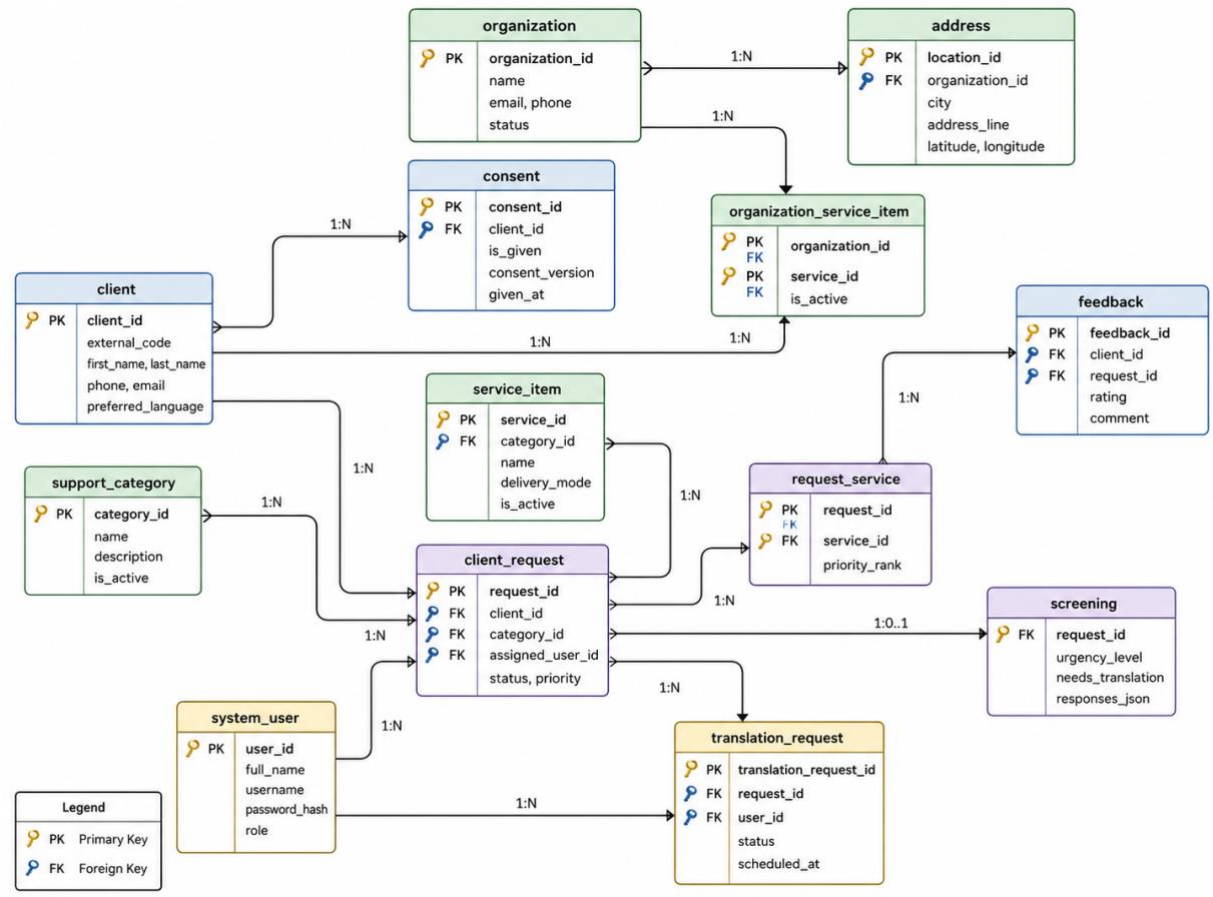

Рис. 2.1 – ER-діаграма інформаційної бази DATACHECK Connect

Таблиця 2.1 – Основні таблиці БД

Таблиця	Призначення
client	Профілі клієнтів, контактні дані, мова й місто перебування.
consent	Фіксація згоди на обробку даних і версії тексту згоди.
support_category	Довідник категорій підтримки.
service_item	Каталог послуг із категорією та форматом надання.
organization	Організації-надавачі послуг.
address	Локації організацій з координатами для карти.
system_user	Адміністратори, координатори, виконавці та інші службові користувачі.

Таблиця	Призначення
client_request	Операційні звернення клієнтів.
screening	Додаткова анкета до звернення.
translation_request	Заявки на асистований переклад.
feedback	Оцінки та коментарі користувачів.

2.2 Діаграма класів та кооперацій

На рівні класової моделі сутності інформаційної бази інтерпретуються як предметні класи, що мають атрибути й операції. Клас “Клієнт” відповідає за ініціювання звернення та відгук. Клас “Звернення” зберігає стан кейсу, пріоритет, категорію і зв’язок з клієнтом. Клас “Послуга” описує конкретну підтримку, а “Організація” і “Локація” пов’язують її з реальним надавачем.

Кооперація “Надання згоди” включає клієнта і згоду. Вона виконується на початку взаємодії, коли користувач підтверджує правомірність подальшої обробки. Кооперація “Реєстрація звернення” поєднує клієнта, звернення і категорію підтримки. Вона створює основу для подальшої маршрутизації.

Кооперація “Скринінг” пов’язує звернення з анкетою. Вона потрібна для уточнення ризиків, терміновості, потреби в перекладі та наявності інших потреб. Кооперація “Підбір послуг” поєднує звернення, категорію та послуги, що дає змогу сформувати рекомендації.

Кооперація “Надання послуги організацією” описує зв’язок між послугою, організацією та локацією. Вона є основою для інтерактивної карти. Кооперація “Опрацювання звернення” включає користувача системи і звернення, а кооперація “Координація перекладу” додатково включає заявку на асистований переклад.

Останньою є кооперація “Збір зворотного зв’язку”, де клієнт залишає відгук, прив’язаний до звернення. Такий набір кооперацій покриває повний цикл роботи системи: від початкової згоди до завершення кейсу й оцінювання якості.

Таблиця 2.2 – Прості кооперації предметної області

№	Кооперація	Класи	Призначення
1	Надання згоди	Клієнт, Згода	Фіксація правової підстави обробки даних.
2	Реєстрація звернення	Клієнт, Запит, Категорія	Створення кейсу та первинна класифікація.
3	Додатковий скринінг	Запит, Анкета	Уточнення потреб і терміновості.
4	Підбір послуг	Запит, Категорія, Послуга	Формування рекомендацій.
5	Надання послуги	Послуга, Організація, Локація	Зв’язок довідника з реальною точкою допомоги.
6	Опрацювання звернення	Користувач системи, Запит	Супровід кейсу службовою особою.
7	Координація перекладу	Запит, Заявка на переклад, Користувач	Призначення та контроль задачі перекладу.
8	Збір відгуку	Клієнт, Запит, Відгук	Оцінка якості взаємодії.

2.3 Діаграма пакетів

Пакетна структура системи відображає логічне групування модулів за відповідальністю. Першим пакетом є публічний клієнтський контур, до якого входять головна сторінка, навігація, каталог послуг, карта, скринінг, заявка на переклад і форма відгуку. Цей пакет орієнтований на користувача і не повинен містити службових кнопок для переходу до адміністративних режимів.

Другим пакетом є службовий інтерфейс, що включає адмін-панель і кабінет виконавця. Адмін-панель відповідає за керування довідником, користувачами і зверненнями. Кабінет виконавця відповідає тільки за задачі перекладу. Обидва модулі використовують спільну клієнтську логіку app.js, але активують різні режими залежно від сторінки.

Третім пакетом є серверний API. Він реалізований у server.js і надає REST-маршрути для публічних і службових операцій. У цьому пакеті

зосереджено авторизацію, перевірку ролей, обробку помилок і маршрутизацію запитів до бізнес-функцій src/apiData.js.

Четвертим пакетом є доменна логіка і доступ до даних. Файл src/apiData.js містить функції отримання послуг, звернень, задач, відгуків, створення скринінгу, перекладу, відгуку, керування послугами і користувачами. Файл src/db.js відповідає за підключення до PostgreSQL, а src/mappers.js перетворює значення БД у формат інтерфейсу.

П’ятим пакетом є інфраструктура БД і запуску. До нього входять database/schema.sql, src/seed.js, scripts/init-db.js, scripts/seed-db.js, docker-compose.yml, .env.example, package.json і скрипти run_api. Цей пакет забезпечує розгортання, ініціалізацію та повторне наповнення тестовими даними.

Таблиця 2.3 – Пакетна структура проєкту

Пакет	**Файли / модулі**	**Відповідальність**
Публічний UI	public/index.html, public/styles.css, public/app.js	Каталог, карта, скринінг, переклад, відгук.
Службові UI	public/admin.html, public/executor.html	Окремі вікна адміністратора і виконавця за прямими посиланнями.
API	server.js	REST-маршрути, static server, авторизація, ролі.
Дані та бізнес-логіка	src/apiData.js, src/db.js, src/mappers.js	Запити до БД, перетворення даних, операції домену.
Інфраструктура	database/schema.sql, src/seed.js, docker-compose.yml, package.json	Структура БД, початкові дані, запуск і залежності.

2.4 Діаграма компонентів

Компонентна модель показує, як логічні пакети перетворюються на реалізаційні модулі. Центральним компонентом є server.js, який одночасно виконує роль статичного сервера для HTML/CSS/JS-файлів і REST API для

взаємодії з PostgreSQL. Він не містить усієї бізнес-логіки безпосередньо, а делегує її до src/apiData.js.

Клієнтський компонент public/app.js відповідає за локалізацію, стан інтерфейсу, рендеринг сторінок, запити до API, ініціалізацію карти, відображення модальних вікон, роботу форм і оновлення даних після мутацій. Оскільки інтерфейс є статичним, основна клієнтська логіка міститься саме в цьому файлі.

Компонент авторизації реалізований у server.js через підписані токени. Після успішного входу користувач отримує токен із роллю, і службові маршрути перевіряють його через middleware requireRole. Таким чином, API розділяє доступ адміністратора й виконавця, не покладаючись тільки на прихованість посилань.

Компонент доступу до даних src/db.js ізолює підключення до PostgreSQL. Бізнес-функції не створюють нове підключення вручну, а користуються query або withTransaction. Це спрощує керування транзакціями під час створення клієнта, згоди, звернення, скринінгу або задачі перекладу.

Компонент schema.sql формує фізичну структуру БД. Разом із seed.js він дозволяє відновити робоче середовище з нуля. У навчальному проєкті це важливо, тому що перевіряючий може розгорнути БД, виконати npm run db:init і отримати готовий набір демонстраційних послуг, організацій, користувачів, звернень і задач.

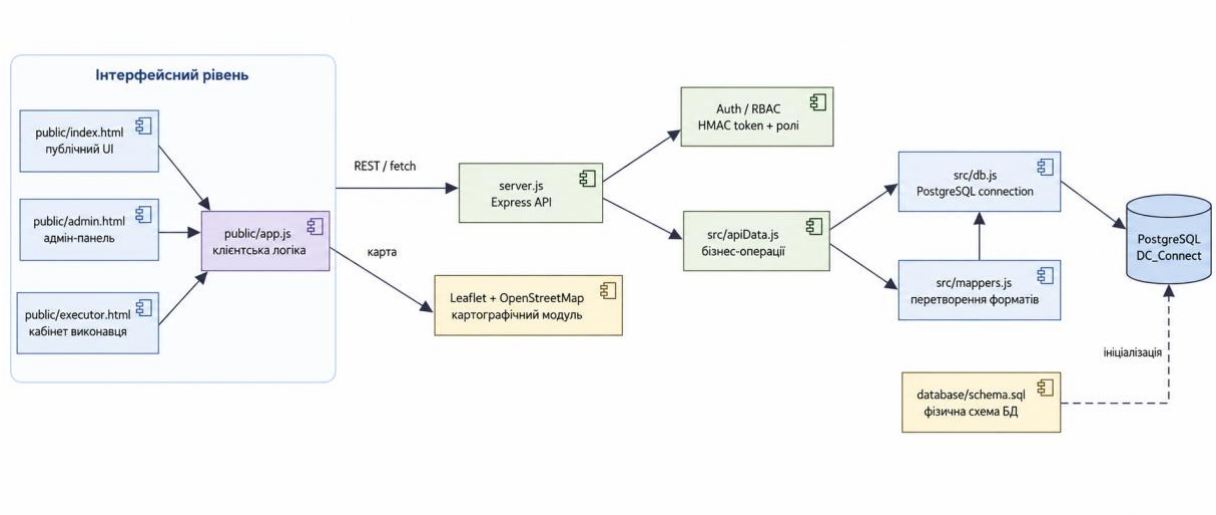

Рис. 2.2 – Компонентна діаграма структури проєкту

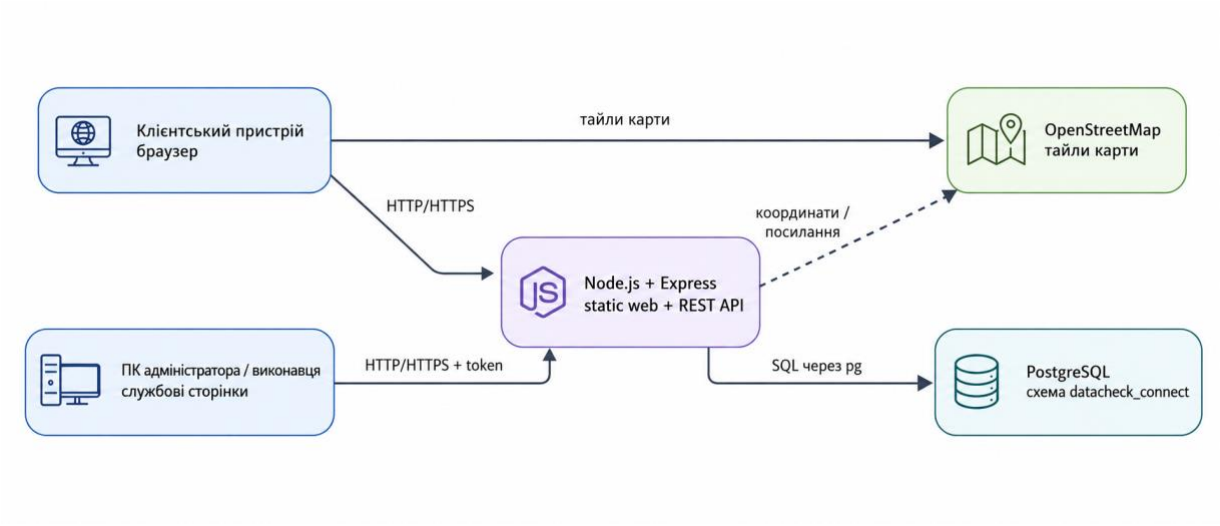

Рис. 2.3 – Діаграма розгортання DATACHECK Connect

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Для реалізації інформаційної бази обрано PostgreSQL. Основною причиною вибору є відповідність реляційній природі даних про клієнтів, звернення, послуги, організації, користувачів і задачі перекладу. Ці об’єкти мають чіткі зв’язки, тому потребують зовнішніх ключів, обмежень, індексів і транзакцій.

PostgreSQL дозволяє створювати окрему схему datacheck_connect, що відділяє таблиці проєкту від інших об’єктів бази. Це зручно для навчального розгортання, оскільки всі таблиці, типи, індекси й тригери належать до одного логічного простору. Схему можна перестворити або оновити без зміни інших баз.

У схемі використовуються перелічувані типи request_status_enum, request_priority_enum, organization_status_enum, translation_status_enum, delivery_mode_enum і user_role_enum. Завдяки цьому значення статусів, пріоритетів, ролей і форматів послуг не зберігаються довільним текстом, а контролюються на рівні БД.

Для перевірки коректності даних використовуються CHECK-обмеження. Наприклад, координати адреси повинні належати допустимим діапазонам широти й довготи, рейтинг відгуку має бути від 1 до 5, дата завершення не може бути ранішою за дату створення або запланований час. Такі обмеження зменшують ризик некоректних даних незалежно від помилок інтерфейсу.

У таблицях client, organization, system_user, client_request і screening є поле updated_at, яке автоматично оновлюється тригером set_updated_at. Це дозволяє відстежувати зміну ключових записів і є корисним для подальшого аудиту. Індекси створені на полях, що часто використовуються для фільтрації: client_id, status, category_id, assigned_user_id, request_id, service_id.

Бібліотека pg використовується для підключення Node.js до PostgreSQL. Параметри з’єднання беруться з DATABASE_URL у файлі .env. Такий підхід відокремлює код від конкретного пароля, порту або назви бази. Для локального запуску можна використати власний PostgreSQL або контейнер Docker Compose.

Таблиця 3.1 – Обґрунтування вибору PostgreSQL

Критерій	Пояснення
Реляційність	Дані мають чіткі сутності та зв’язки, які природно реалізуються таблицями.
Цілісність	Підтримуються PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, індекси і транзакції.
Гнучкість	JSONB дозволяє зберігати напівструктуровані відповіді скринінгу.
Доступність	СУБД є відкритою, поширеною і зручною для навчального розгортання.
Масштабованість	Схема може розширюватися новими довідниками, ролями та журналами.

3.2 Розробка інформаційної бази

Фізична схема БД реалізована у файлі database/schema.sql. Скрипт створює схему datacheck_connect, перелічувані типи, функцію оновлення updated_at, основні таблиці, індекси, тригери та коментарі до таблиць. Скрипт можна виконати через npm run db:init, оскільки scripts/init-db.js читає SQL-файл і виконує його в підключеній базі.

Основні бізнес-таблиці відповідають логічній моделі. client зберігає профіль користувача, consent — згоду, support_category — категорії, service_item — послуги, organization — організації, address — адреси, system_user — службових користувачів, client_request — звернення, screening — результати анкети, translation_request — задачі перекладу, feedback — оцінки клієнтів.

Асоціативні таблиці request_service і organization_service_item є важливими для нормалізації. Перша дозволяє зберігати список рекомендованих

послуг для конкретного звернення. Друга описує, які організації надають які послуги. Завдяки цьому можна додати нову організацію або змінити її перелік послуг без дублювання записів у service_item.

Початкові дані реалізовані у src/seed.js. У seed-файлі створюються категорії, десять послуг, організації, адреси з координатами, користувачі, демонстраційні звернення, заявки на переклад і відгуки. Це дозволяє одразу перевірити каталог, карту, адмін-панель і кабінет виконавця після ініціалізації БД.

У модулі src/apiData.js запити до БД згруповані за сценаріями. Функція getServices формує каталог, об’єднуючи категорії, послуги, організації та адреси. Функції getRequests, getTranslationTasks і getFeedbacks формують операційні списки для службових екранів. Функції createScreening, createTranslationTask і createFeedback записують дані публічних форм.

Для операцій, які змінюють кілька таблиць одночасно, використано транзакції. Наприклад, створення скринінгу включає створення клієнта, згоди, звернення і запису screening. Якщо один із запитів завершується помилкою, транзакція повинна бути скасована, щоб база не містила частково створених записів.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Серверна частина створена на Node.js із використанням Express.js. Node.js дозволяє виконувати JavaScript на сервері, а Express.js забезпечує мінімалістичний механізм маршрутизації HTTP-запитів, middleware і обробку помилок. Для навчального проєкту це дає можливість швидко реалізувати API без складної конфігурації фреймворку.

Фронтенд реалізований як статичний інтерфейс на HTML, CSS і JavaScript. У проєкті немає окремого frontend-build етапу, тому файли public/index.html, public/admin.html, public/executor.html, public/styles.css і

public/app.js можуть віддаватися сервером Express без транспіляції. Це спрощує розгортання й робить структуру зрозумілою для перевірки.

Для карти використано Leaflet. У public/index.html підключено стилі та скрипт Leaflet, а в app.js реалізовано ініціалізацію карти, підключення тайлового шару OpenStreetMap, створення маркерів і popup-вікон. Якщо Leaflet не завантажився, інтерфейс показує повідомлення про необхідність перевірити підключення до інтернету для тайлів карти.

Для конфігурації використано dotenv. Файл .env.example містить приклад DATABASE_URL, DB_SCHEMA, PORT, AUTH_SECRET, логінів і паролів для демо-ролей. Завдяки цьому секрети не потрібно прописувати у вихідному коді. Для запуску БД використовується docker-compose.yml, який піднімає PostgreSQL і спрощує перевірку проєкту на іншому комп’ютері.

Пакет npm містить скрипти start, dev, db:init і db:seed. start запускає server.js, dev використовує nodemon для розробки, db:init створює схему та seed-дані, db:seed оновлює початкові записи. Така структура відповідає типовій практиці організації Node.js-проєкту і дозволяє швидко повторити запуск.

Візуальний дизайн реалізований у styles.css. У ньому описано макет сторінки, ліве меню, картки, таблиці, форми, кнопки, адаптивні сітки, карту, службові панелі та стани елементів. Після останніх змін з головної сторінки прибрано блоки UI-прототипу, швидких дій і режимів інтерфейсу, щоб публічна частина виглядала як робочий сервіс, а не як демонстраційний макет.

Таблиця 3.2 – Використані технології та засоби

Засіб	**Призначення у проєкті**
Node.js	Виконання серверного JavaScript і запуск застосунку.
Express.js	Створення HTTP-сервера, REST API та static server.
PostgreSQL	Збереження клієнтів, звернень, послуг, користувачів і відгуків.
pg	Підключення Node.js до PostgreSQL.
dotenv	Завантаження налаштувань з .env.
HTML / CSS / JavaScript	Реалізація статичного інтерфейсу.

Засіб	Призначення у проєкті
Leaflet	Інтерактивна карта в браузері.
OpenStreetMap	Тайли карти та географічна основа.
Docker Compose	Швидкий запуск PostgreSQL у контейнері.
npm	Керування залежностями та скриптами.

3.4 Алгоритмізація та програмування програмних модулів

Серверний модуль server.js починається з підключення залежностей, налаштування Express і static middleware. Далі визначено допоміжні функції для асинхронних маршрутів, підпису токенів, перевірки паролів, нормалізації ролей і перевірки доступу. Це дозволяє обробляти помилки централізовано та не дублювати перевірку авторизації в кожному маршруті.

Авторизація реалізована через POST /api/auth/login. Користувач передає роль, логін і пароль. Сервер спочатку перевіряє користувача в базі через api.findAuthUser, а якщо база недоступна або користувач не знайдений, може використати демо-облікові записи з .env. Після успішного входу сервер формує підписаний токен з роллю, іменем, логіном і часом завершення дії.

Middleware requireRole перевіряє заголовок Authorization, витягує Bearer token, розшифровує payload і порівнює роль із потрібною. Якщо токен відсутній або роль неправильна, сервер повертає статус 401. Це використовується для маршрутів /api/admin/bootstrap, /api/executor/bootstrap, керування послугами, користувачами, видалення звернень і операцій виконавця.

Клієнтська логіка app.js використовує централізований стан: поточну мову, активну сторінку, фільтри каталогу, вибране місто карти, дані форм, авторизаційний токен, ім’я виконавця, список користувачів і дані з API. Після завантаження сторінки виконується запит bootstrap, який повертає послуги, організації, звернення, задачі, відгуки і таймлайн.

Алгоритм пошуку послуг працює як послідовне застосування фільтрів. Спочатку береться список services, потім перевіряється ключове слово, категорія, місто і формат надання. Результат відображається картками з назвою,

описом, організацією, містом, форматом, мовами й кнопкою перегляду деталей. Такий підхід простий, але достатній для невеликого каталогу.

Алгоритм карти групує послуги за організаціями. Для кожної організації беруться місто, адреса, координати і перелік послуг. Якщо координати відсутні, використовується fallback для міста. Далі Leaflet створює карту, додає tileLayer OpenStreetMap, маркери, popup і підлаштовує межі карти під набір координат. У popup послуги відображаються як кнопки, що робить карту інтерактивною.

Алгоритм створення скринінгу на сервері виконується в транзакції. Спершу формується клієнт, потім запис згоди, далі звернення з категорією та статусом, після чого створюється screening з JSON-відповідями. У результаті сервер повертає оновлений bootstrap, і фронтенд оновлює списки без перезавантаження сторінки.

Алгоритм задач перекладу працює схожим чином. Створення задачі формує клієнта, звернення і translation_request. Взяття в роботу змінює статус translation_request на IN_PROGRESS і пов’язує задачу з виконавцем. Завершення встановлює статус DONE, completed_at і відповідний статус звернення. У кабінеті виконавця кнопка “Завершити заявку” показується тільки для задач у роботі.

Керування послугами в адмін-панелі використовує одну форму для додавання і редагування. Якщо state.editingServiceId порожній, відправляється POST /api/admin/services. Якщо адміністратор обрав існуючу послугу, форма заповнюється даними, і при збереженні виконується PUT /api/admin/services/:id. У процесі оновлюється не лише послуга, а й організація, адреса, контакти та координати.

Керування користувачами також реалізовано в основній адмін-панелі. Адміністратор може створити службового користувача, вказати ПІБ, email, логін, телефон, роль, пароль і активність. Якщо пароль переданий, сервер перетворює його у scrypt-хеш перед збереженням. Це краще, ніж зберігати пароль відкритим текстом.

Видалення звернень реалізоване через DELETE /api/admin/requests/:id. На рівні apiData.js функція deleteRequest знаходить публічний ідентифікатор, видаляє пов’язані записи, якщо це передбачено зв’язками, і повертає оновлений стан. Така функція потрібна для очищення помилкових або тестових звернень у журналі кейсів.

Таблиця 3.3 – Основні API endpoints

Метод	Endpoint	Призначення
GET	/api/health	Перевірка сервера і підключення до БД.
GET	/api/bootstrap	Початкові публічні дані.
POST	/api/auth/login	Вхід адміністратора або виконавця.
POST	/api/screening	Створення звернення зі скринінгу.
POST	/api/translation-tasks	Створення задачі перекладу.
POST	/api/feedbacks	Збереження відгуку.
GET	/api/admin/bootstrap	Дані адмін-панелі після входу.
POST	/api/admin/services	Додавання послуги.
PUT	/api/admin/services/:id	Редагування послуги.
POST	/api/admin/users	Створення користувача.
PUT	/api/admin/users/:id	Редагування користувача.
DELETE	/api/admin/requests/:id	Видалення звернення.
POST	/api/executor/tasks/:id/take	Взяття заявки виконавцем.
POST	/api/executor/tasks/:id/complete	Завершення заявки виконавцем.

3.5 Реалізація клієнтського інтерфейсу

Клієнтський інтерфейс DATACHECK Connect реалізований як односторінковий вебзастосунок без окремого етапу збірки. HTML-сторінки, стилі та JavaScript-файл public/app.js віддаються сервером Express, а перемикання між публічними й службовими екранами виконується на стороні браузера.

Головна сторінка містить короткий опис сервісу та навігацію до основних сценаріїв: каталогу, карти, скринінгу, заявки на переклад і форми відгуку. Службові переходи не показуються в публічному меню, щоб не перевантажувати інтерфейс клієнта.

Локалізацію реалізовано через об’єкт перекладів у app.js. Ключі інтерфейсу мають українські, англійські та німецькі значення, а функція t(key) повертає текст відповідно до вибраної мови. Для назв і описів послуг використовується getLocalized.

Стан інтерфейсу зберігається в об’єкті state: поточна мова, активна сторінка, фільтри каталогу, вибране місто карти, токени авторизації та дані форм. Після зміни фільтра або успішного API-запиту викликається повторний рендеринг потрібного екрана.

Каталог формує картки послуг із назвою, описом, містом, форматом, організацією, контактами та мовами. Деталі відкриваються в модальному вікні, тому список залишається компактним навіть за великої кількості записів.

Форма скринінгу збирає дані для первинної маршрутизації, але не формує медичний висновок. Заявка на переклад винесена окремо, оскільки перекладач може бути потрібний і без проходження скринінгу.

Візуальна частина описана у styles.css. У файлі задано картки, таблиці, кнопки, форми, службові панелі, адаптивні сітки та блок карти. На менших екранах компоненти переходять у вертикальне розташування.

3.6 Реалізація службових панелей і контролю доступу

Службові панелі адміністратора і виконавця відкриваються за прямими посиланнями /admin та /executor. Приховані посилання не є захистом, тому всі службові API-маршрути додатково перевіряють токен і роль користувача.

Після входу адміністратор бачить KPI-блок, журнал звернень, задачі перекладу, форму редагування послуг і блок керування користувачами. У журналі можна фільтрувати записи, переглядати статуси та видаляти тестові або помилкові звернення.

Форма послуг дозволяє змінювати не тільки назву й опис, а й категорію, формат, місто, організацію, адресу, контакти та координати. Це потрібно, бо послуга в каталозі прив’язана до реального місця надання допомоги.

Кабінет виконавця має лише необхідні дії: перегляд доступних і власних заявок, взяття задачі в роботу та завершення виконаної заявки. Редагування послуг, користувачів і звернень у цьому кабінеті не передбачене.

Контроль доступу реалізовано в server.js через middleware requireRole. Якщо запит не містить коректного Bearer-токена або роль не відповідає маршруту, сервер повертає 401 Unauthorized.

Паролі службових користувачів зберігаються у вигляді scrypt-хешів. Підписаний токен містить роль, логін, ім’я користувача та час завершення дії; перевірка підпису виконується з використанням AUTH_SECRET.

Після мутацій API повертає оновлений стан системи. Завдяки цьому додавання послуги, редагування користувача або завершення заявки одразу відображаються в інтерфейсі без ручного перезавантаження.

3.7 Реалізація початкових даних і демонстраційних сценаріїв

Початкові дані мають велике значення для перевірки проєкту, тому що система з порожньою базою не демонструє каталог, карту, звернення або задачі. У DATACHECK Connect seed-дані створюються у src/seed.js. Скрипт наповнює категорії, послуги, організації, адреси, користувачів, звернення, задачі перекладу та відгуки.

Категорії початкових даних відповідають основним напрямам підтримки: medical, social, legal і mental. Для кожної категорії задається україномовний опис. Послуги мають різні міста та формати надання, що дозволяє одразу перевірити фільтри каталогу. Наприклад, одна послуга є онлайн-консультацією, інша — офлайн-супроводом, третя — гібридною юридичною консультацією.

Для карти важливо, що seed-організації мають адреси й координати. У тестових даних використано Berlin, Hamburg, Köln, München і Leipzig. Це дає змогу перевірити центрування карти, fitBounds для кількох маркерів, фільтр за

містом і fallback-координати. Якщо адміністратор додає нову послугу з координатами, вона може бути відображена на карті після оновлення даних.

Початкові користувачі включають координатора і кількох перекладачів. Це потрібно для перевірки призначення задач і відображення виконавців у кабінеті. У реальній системі ролі могли б деталізуватися ширше, але для прототипу достатньо ADMIN/COORDINATOR і EXECUTOR/TRANSLATOR.

Демонстраційні звернення мають різні статуси й пріоритети. Це дозволяє протестувати таблицю адмін-панелі, фільтр за статусом, KPI і візуальні бейджі пріоритету. Якщо всі звернення були б однаковими, службова панель не демонструвала б логіку сортування і різних станів кейсу.

Демонстраційні задачі перекладу мають статуси REQUESTED, IN_PROGRESS і DONE. Завдяки цьому виконавець бачить різні групи задач: доступні, власні та завершені. Під час тестування можна створити нову задачу з публічної форми й перевірити, що вона переходить у доступні задачі без ручного додавання до БД.

Відгуки у seed-даних потрібні для перевірки середнього рейтингу. Якщо відгуків немає, показник дорівнює 0.0, але це не дає змоги побачити реальну аналітичну картку. Наявність кількох відгуків дозволяє перевірити, що новий відгук змінює середнє значення.

Таблиця 3.4 – Приклади початкових послуг

Код	Місто	Формат	Послуга	Організація
MED	Berlin	ONLINE	Інфекційна консультація онлайн	MediHelp Berlin
SOC	Hamburg	OFFLINE	Соціальний супровід сімей	Hamburg Welcome Point
LEG	Köln	HYBRID	Юридична консультація щодо страхування	Legal Aid NRW
PSY	München	HYBRID	Психологічна підтримка для пацієнтів	MindCare München
MED	Leipzig	OFFLINE	Підбір клініки та	Care Route Leipzig

Код	Місто	Формат	Послуга	Організація
			запис на прийом	
SOC	Berlin	ONLINE	Гаряча лінія кризової підтримки	DC Connect Hotline
MED	Hamburg	OFFLINE	Переклад на медичному візиті	Translation Support HH
LEG	München	ONLINE	Юридичний чат-бот і шаблони заяв	Justice Link Bayern
PSY	Köln	HYBRID	Група підтримки для пацієнтів та родин	Together Köln
SOC	Leipzig	HYBRID	Навігація по соціальних пільгах	Leipzig Social Hub

3.8 Організація REST API та обробка помилок

REST API у DATACHECK Connect виконує роль проміжного шару між клієнтським інтерфейсом і базою даних. Клієнтський JavaScript не звертається до PostgreSQL напряму, а формує HTTP-запити до серверних endpoint. Це дозволяє централізовано перевіряти вхідні дані, контролювати ролі, обробляти помилки та повертати інтерфейсу єдиний формат відповіді.

Публічні endpoint не потребують авторизації, оскільки вони призначені для клієнтів. До них належать /api/bootstrap, /api/services, /api/screening, /api/translation-tasks і /api/feedbacks. Проте навіть у публічних маршрутах сервер не повинен довіряти вхідним даним повністю: потрібно нормалізувати текстові значення, перевіряти обов’язкові поля та не дозволяти записувати некоректні рейтинги, статуси або координати.

Службові endpoint захищені middleware requireRole. Маршрути адміністратора потребують ролі ADMIN, а маршрути виконавця — ролі EXECUTOR. Це правило застосовується до отримання службового bootstrap, керування послугами, користувачами, видалення звернень, взяття задачі в роботу та завершення задачі. Завдяки цьому фронтенд і бекенд узгоджено підтримують розділення ролей.

Для асинхронних маршрутів використовується обгортка asyncRoute. Вона дозволяє не писати try/catch у кожному endpoint і передає помилку до централізованого error handler. Такий підхід робить код коротшим і зменшує ризик, що необроблена помилка призведе до завислого HTTP-запиту.

Централізований обробник помилок повертає JSON з полем error і status. Якщо помилка серверна, вона додатково виводиться в консоль. Для користувача це важливо, тому що фронтенд може показати зрозуміле повідомлення через showApiError, а не залишати форму без відповіді. Для розробника це зручно, бо помилки видно в терміналі під час тестування.

Після мутацій API часто повертає оновлений набір даних. Наприклад, створення скринінгу, задачі перекладу або відгуку повертає дані, які дозволяють одразу оновити стан інтерфейсу. Такий підхід простіший за часткове оновлення окремих масивів, оскільки зменшує ризик розсинхронізації між клієнтським станом і базою даних.

У майбутньому API можна деталізувати через OpenAPI-специфікацію. Це дозволило б автоматично документувати endpoint, типи параметрів, відповіді та помилки. Для командної розробки це особливо корисно, оскільки frontend і backend можуть працювати за спільним контрактом.

3.9 Перетворення даних між базою та інтерфейсом

Дані в PostgreSQL і дані в інтерфейсі мають різний формат. У БД статуси зберігаються як NEW, IN_PROGRESS, DONE або CLOSED, а в інтерфейсі зручніше працювати з коротшими або локалізованими значеннями. Тому в проєкті є src/mappers.js, який виконує перетворення статусів, категорій, міст, форматів і публічних ідентифікаторів.

Публічний інтерфейс не повинен показувати внутрішні числові первинні ключі. Тому для відображення використовується формат на кшталт REQ-101 або TR-201. Це робить записи зрозумілішими для користувача і приховує технічну структуру таблиць. На сервері такі ідентифікатори можуть

перетворюватися назад у числові ключі або використовуватися для пошуку відповідного запису.

Мапери також потрібні для міст. У seed-даних і таблицях можуть використовуватися назви Berlin, Hamburg, Köln, München і Leipzig, а в інтерфейсі вони відображаються через локалізовані підписи. Для фільтрів зручніше мати стабільні ключі berlin, hamburg, cologne, munich і leipzig. Це дозволяє уникнути помилок, пов’язаних з різними мовами або написанням umlaut.

Категорії також мають технічні ключі medical, social, legal і mental, але для користувача відображаються як медична, соціальна, юридична і психологічна підтримка. Відокремлення ключа від назви дає змогу змінити текст перекладу без зміни логіки фільтрації або структури БД.

Формат надання послуги зберігається в БД як delivery_mode_enum: ONLINE, OFFLINE або HYBRID. У клієнтському інтерфейсі він перетворюється на online, offline або hybrid і далі відображається через локалізовані підписи. Такий підхід підтримує послідовність між БД, API і UI.

Під час отримання послуг сервер формує об’єкти, які зручні для фронтенду: id, name, description, category, city, mode, languages, organization, phone, email, website, address, lat, lon. Це означає, що frontend не повинен самостійно об’єднувати таблиці. Він отримує вже підготовлений об’єкт і може одразу рендерити картку або маркер.

Перевага такого підходу полягає в тому, що складна логіка об’єднання даних залишається на сервері, де вона ближча до SQL і БД. Недоліком є те, що при зміні формату API потрібно одночасно оновлювати frontend. Для зменшення такого ризику варто документувати структуру відповіді і підтримувати стабільні назви полів.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування DATACHECK Connect виконувалося як поєднання перевірки запуску, перевірки API, ручного функціонального тестування інтерфейсу та перевірки цілісності основних сценаріїв. Оскільки проєкт є навчальним прототипом, основний акцент зроблено на наскрізних сценаріях, які демонструють роботу системи від публічної форми до службової панелі.

Перший блок тестів стосується запуску. Потрібно встановити залежності через npm install, запустити PostgreSQL, виконати npm run db:init і стартувати сервер npm start. Після цього відкривається /api/health, який повинен повернути ok: true та статус підключення до бази. Якщо база недоступна, інтерфейс може показати початкові локальні дані, але повна робота вимагає PostgreSQL.

Другий блок тестів перевіряє каталог і карту. У каталозі перевіряється відображення послуг, робота пошуку, фільтрів категорії, міста і формату. На карті перевіряється завантаження тайлів, маркери організацій, popup-вікна і кнопки послуг. Якщо змінити дані послуги в адмін-панелі, вони повинні оновитися після отримання нового bootstrap.

Третій блок тестів перевіряє форми. Для скринінгу потрібно заповнити поля, підтвердити згоду і відправити форму. Після цього нове звернення має з’явитися в адмін-панелі. Для перекладу потрібно створити задачу, увійти виконавцем, взяти її в роботу, а потім завершити. Статуси повинні змінитися без ручного редагування БД.

Четвертий блок тестів перевіряє авторизацію і ролі. Доступ до /admin і /executor можливий за прямим посиланням, але службові API endpoints повинні повертати 401 без коректного токена. Адміністратор не повинен працювати як виконавець без відповідної ролі, а виконавець не повинен мати доступу до створення послуг або користувачів.

П’ятий блок тестів стосується форми відгуку. Користувач відкриває окремий блок “Форма відгуку” в лівому меню, обирає кількість зірок, вводить коментар і надсилає форму. У відповідь система додає запис feedback, оновлює список відгуків і середній рейтинг у KPI адмін-панелі.

Таблиця 4.1 – Матриця тестування основних сценаріїв

Сценарій	Очікуваний результат
GET /api/health	Сервер відповідає ok: true, БД доступна.
Відкрити каталог	Відображаються послуги з категоріями, містами і форматами.
Фільтр послуг	Список змінюється відповідно до пошуку, міста, категорії та формату.
Відкрити карту	Завантажується OpenStreetMap, маркери містять послуги організацій.
Створити скринінг	У БД створюються client, consent, client_request, screening.
Увійти як адмін	Відкривається адмін-панель з журналом і KPI.
Додати послугу	Нова послуга з’являється в каталозі та в адмін-списку.
Створити користувача	Новий користувач доступний у таблиці користувачів.
Створити задачу перекладу	Задача з’являється в адмін-панелі та кабінеті виконавця.
Завершити заявку виконавцем	Статус задачі змінюється на виконаний/підтверджений.
Надіслати відгук	У БД створюється feedback, середній рейтинг оновлюється.

4.2 Вимоги до апаратного та програмного забезпечення

Для локального запуску системи потрібен комп’ютер з операційною системою Windows, Linux або macOS, встановленими Node.js, npm і PostgreSQL. Якщо PostgreSQL не встановлено локально, можна використати Docker і Docker Compose. Для роботи з інтерфейсом потрібен сучасний браузер з підтримкою JavaScript і доступом до інтернету для завантаження тайлів OpenStreetMap.

Мінімальні апаратні вимоги для навчального запуску невисокі. Достатньо двоядерного процесора, 4 ГБ оперативної пам’яті та кількох сотень мегабайт вільного місця. Для продуктивного середовища вимоги залежать від кількості звернень, користувачів і організацій, але архітектура дозволяє окремо масштабувати сервер застосунку і сервер бази даних.

Програмне середовище складається з Node.js-сервера, PostgreSQL-бази, статичних frontend-файлів і конфігурації .env. У випадку продуктивного розгортання доцільно використовувати HTTPS, reverse proxy, резервне копіювання бази, окремий секрет AUTH_SECRET, складні паролі службових ролей і регулярне оновлення залежностей.

Оскільки система обробляє персональні дані, в експлуатації необхідно враховувати принцип мінімізації даних. У публічних формах доцільно збирати лише ті поля, які потрібні для маршрутизації звернення. Для службових користувачів потрібно застосовувати індивідуальні облікові записи, активність користувача і подальше журналювання критичних операцій.

Рекомендовано регулярно перевіряти актуальність довідника послуг, контактів організацій і координат. Якщо організація більше не надає послугу, її зв’язок із service_item потрібно деактивувати або оновити. Якщо змінюється адреса, необхідно перевірити координати, оскільки вони впливають на карту.

Таблиця 4.2 – Мінімальні вимоги до середовища

Компонент	**Вимога**
ОС	Windows 10/11, Linux або macOS.
Node.js	Версія, сумісна з сучасними npm-пакетами.
СУБД	PostgreSQL або контейнер Docker PostgreSQL.
Браузер	Chrome, Edge, Firefox або інший сучасний браузер.
Мережа	Доступ до локального сервера та інтернету для OpenStreetMap.
RAM	Від 4 ГБ для навчального запуску.
Диск	Від 500 МБ для проєкту, залежностей і тестової БД.

4.3 Склад інсталяційного пакету

Інсталяційний пакет проєкту складається з файлів вихідного коду, SQL-схеми, конфігураційних файлів, npm-залежностей і документації. Основним виконуваним файлом є server.js. Він запускається командою npm start і піднімає HTTP-сервер на порту, визначеному в .env або за замовчуванням 3000.

Папка public містить усі файли інтерфейсу. index.html використовується для публічної сторінки, admin.html для службового вікна адміністратора, executor.html для кабінету виконавця. styles.css містить стилі, а app.js — логіку рендерингу, локалізації, роботи з API, картою, формами та службовими панелями.

Папка database містить schema.sql, який створює структуру PostgreSQL. Папка src містить db.js, apiData.js, mappers.js і seed.js. Папка scripts містить init-db.js і seed-db.js. Корінь проєкту містить package.json, package-lock.json, docker-compose.yml, .env.example, README.md, run_api.bat і run_api.sh.

Процес запуску складається з кількох кроків: розпакувати проєкт, створити або скопіювати .env, встановити залежності, підняти PostgreSQL, виконати npm run db:init, запустити npm start і відкрити http://localhost:3000. Службові панелі відкриваються за прямими посиланнями http://localhost:3000/admin і http://localhost:3000/executor.

Для демонстраційного входу використовуються облікові записи admin / admin123 і executor / executor123, якщо інші значення не задано в .env або в таблиці system_user. У реальному середовищі ці паролі потрібно обов’язково змінити, а AUTH_SECRET зробити складним і унікальним.

Таблиця 4.3 – Структура інсталяційного пакету

Файл / папка	Призначення
server.js	Основний сервер Express API.
public/	Статичні HTML, CSS і JavaScript файли інтерфейсу.
database/schema.sql	SQL-схема PostgreSQL.
src/apiData.js	Бізнес-функції та запити до БД.
src/db.js	Підключення до PostgreSQL і транзакції.

Файл / папка	Призначення
src/mappers.js	Перетворення статусів, категорій і форматів.
src/seed.js	Початкові демонстраційні дані.
scripts/init-db.js	Ініціалізація БД і seed.
package.json	Залежності й npm-скрипти.
docker-compose.yml	Контейнер PostgreSQL для локального запуску.
.env.example	Приклад конфігурації.
README.md	Інструкція запуску і перевірки.

4.4 Захист даних і безпека експлуатації

DATACHECK Connect працює з персональними та потенційно чутливими даними, тому безпека врахована вже в прототипі. Публічний інтерфейс відокремлено від службових маршрутів, а адміністративні операції виконуються тільки після перевірки ролі.

Під час збору даних застосовується принцип мінімізації: у формах запитуються лише відомості, потрібні для маршрутизації звернення або організації перекладу. Згода на обробку даних фіксується окремою сутністю consent із версією тексту та часом підтвердження.

Паролі не зберігаються відкритим текстом: сервер формує scrypt-хеш і використовує його під час авторизації. Демо-облікові записи з .env придатні лише для перевірки; у робочому середовищі їх потрібно замінити індивідуальними записами.

Конфігураційні параметри DATABASE_URL, AUTH_SECRET і службові паролі не мають потрапляти до відкритого репозиторію. Для прикладу налаштувань використовується .env.example, а реальні значення задаються окремо.

Для надійної експлуатації потрібні резервні копії PostgreSQL, журналювання критичних дій і контроль використання зовнішніх сервісів, зокрема тайлів OpenStreetMap. Ці заходи зменшують ризик втрати даних і спрощують аудит роботи системи.

4.5 Напрями подальшого розвитку системи

Перший напрям розвитку — деталізація ролей і прав доступу. Окрім адміністратора та виконавця, у реальному процесі можуть бути координатор, оператор гарячої лінії, аналітик або представник організації-партнера.

Другий напрям — розширення аналітики. До KPI можна додати графіки звернень за містами, категоріями й статусами, середній час обробки заявок, навантаження виконавців та оцінки сервісу за відгуками.

Третій напрям — інтеграція зі службами повідомлень. Email, SMS або месенджери можуть використовуватися для підтвердження заявки, нагадування про візит, інформування виконавця та сповіщення адміністратора про термінові звернення.

Четвертий напрям — розвиток карти і скринінгу. Доцільно додати кластеризацію маркерів, фільтри за мовами, побудову маршруту, а також динамічну анкету, у якій питання залежать від попередніх відповідей користувача.

Окремо варто додати автоматизовані тести API, SQL-міграцій і ключових сценаріїв інтерфейсу. Це підвищить стабільність системи під час подальшого розширення.

Таблиця 4.4 – Пріоритети подальшого розвитку

Напрям	Пріоритет	Очікуваний ефект
Розширення ролей	Високий	Точніший контроль доступу і відповідальності.
Журнал аудиту	Високий	Прозорість службових операцій.
Сповіщення email/SMS	Середній	Швидше інформування клієнтів і виконавців.
Динамічний скринінг	Середній	Точніша маршрутизація звернень.
Аналітичні графіки	Середній	Кращий моніторинг роботи сервісу.
Кластеризація карти	Низький/середній	Зручніша робота з великою кількістю організацій.
Автоматизовані тести	Високий	Стабільність при подальшому розвитку.

4.6 Інструкція роботи з публічною частиною

Публічна частина DATACHECK Connect призначена для клієнта, який не має службового облікового запису. Користувач відкриває головну сторінку за адресою http://localhost:3000 і бачить загальну навігацію. У публічному меню не відображаються кнопки адміністратора або виконавця, тому користувач не стикається з незрозумілими службовими режимами.

Для пошуку допомоги клієнт переходить до каталогу послуг. У каталозі можна ввести ключове слово, обрати категорію підтримки, місто або формат надання. Якщо користувач не знає точну назву послуги, він може застосувати категорію, наприклад медичну або соціальну, і переглянути всі доступні варіанти. Картки послуг показують організацію, контактні дані й короткий опис.

Для географічного пошуку користувач переходить до карти. На карті відображаються маркери організацій. Після натискання на маркер відкривається вікно з назвою організації, адресою та кнопками послуг. Це дає

змогу швидко зрозуміти, які сервіси доступні в конкретній організації, і перейти до деталей без додаткового пошуку в каталозі.

Якщо користувач не впевнений, яка допомога потрібна, він може перейти до скринінгу. Форма скринінгу збирає первинні дані про потребу, місто, категорію, терміновість і потребу в перекладі. Перед відправленням потрібно підтвердити згоду на обробку даних. Після надсилання звернення потрапляє в адмін-панель, де його може опрацювати координатор.

Якщо користувач уже має запис до лікаря або знає, що йому потрібен перекладач, він може одразу створити заявку на асистований переклад. У формі вказується місто, дата, час, спеціалізація лікаря, мова перекладу і коментар. Після створення задача з’являється в кабінеті виконавця, а адміністратор може контролювати її статус.

Після взаємодії з сервісом користувач може залишити відгук. Для цього в лівому меню передбачено окремий блок “Форма відгуку”. Користувач обирає оцінку від однієї до п’яти зірок і вводить коментар. Відгук зберігається в БД і впливає на середній рейтинг, який відображається в службовій панелі.

4.7 Інструкція роботи адміністратора і виконавця

Адміністратор відкриває службову панель за прямим посиланням /admin. На сторінці входу він вводить логін і пароль. Після успішної авторизації сервер повертає токен, який використовується для всіх подальших службових запитів. Якщо токен відсутній або неправильний, API повертає помилку авторизації.

У верхній частині адмін-панелі розміщені метрики: кількість звернень, кількість активних послуг, кількість відкритих задач і середній рейтинг. Ці метрики дають швидке уявлення про поточний стан системи. Нижче розміщено журнал звернень, у якому адміністратор може виконати пошук, фільтрувати статуси та видаляти звернення.

Для редагування послуг адміністратор використовує форму “Редагування та додавання послуг”. Якщо потрібно створити нову послугу, форма заповнюється вручну. Якщо потрібно змінити наявну, адміністратор натискає

кнопку редагування в списку, після чого дані підставляються у форму. Після збереження зміни потрапляють у базу і стають доступними в публічному каталозі.

Для керування користувачами адміністратор використовує окремий блок тієї самої панелі. Він може додати нового виконавця або координатора, змінити роль, телефон, email, логін і активність. Якщо потрібно змінити пароль, адміністратор вводить нове значення; сервер хешує його і зберігає в БД.

Виконавець відкриває службовий кабінет за прямим посиланням /executor. Після входу він бачить доступні задачі та свої задачі. Коли виконавець натискає “Взяти задачу”, система переводить її у стан роботи та прив’язує до виконавця. Це дозволяє уникнути ситуації, коли кілька виконавців одночасно беруть одну й ту саму заявку.

Після виконання перекладу виконавець натискає “Завершити заявку”. Система змінює статус задачі, фіксує завершення і оновлює пов’язані дані звернення. Адміністратор бачить зміну в панелі, а задача більше не відображається як активна для виконавця. Такий порядок підтримує прозорий життєвий цикл задачі.

4.8 Підтримка та супровід програмного забезпечення

Після впровадження DATACHECK Connect потребує регулярного супроводу: оновлення каталогу послуг, підтримки службових користувачів, контролю БД та перевірки працездатності API.

Найчастіша операція — актуалізація довідника. Адміністратор має перевіряти контакти, адреси, формати роботи й доступність організацій, а також деактивувати записи, які тимчасово або остаточно не використовуються.

Також потрібно підтримувати облікові записи працівників, оновлювати npm-залежності, створювати резервні копії PostgreSQL і документувати зміни в коді, БД та інтерфейсі.

4.8.1 Регламент оновлення каталогу послуг

Каталог послуг має оновлюватися за простим циклом: перевірка даних, внесення змін в адмін-панелі, контроль відображення в каталозі та перевірка маркера на карті. Особливо уважно потрібно перевіряти адресу, координати, контакти й формат роботи організації.

Якщо послуга тимчасово недоступна, її доцільніше деактивувати, а не видаляти. Для контролю якості варто вести журнал змін із датою, назвою організації, джерелом інформації та відповідальним адміністратором.

Після значного оновлення каталогу слід виконати коротке приймальне тестування: пошук за ключовим словом, фільтр за містом, перегляд картки, відкриття маркера на карті та створення тестового звернення.

4.8.2 Підтримка ролей та керування доступом

Службовий доступ базується на ролях адміністратора та виконавця. Адміністратор керує послугами, користувачами і зверненнями, а виконавець працює тільки із заявками на переклад.

Новому користувачу потрібно надавати мінімально достатні права. Якщо працівник більше не працює із системою, його обліковий запис краще деактивувати, щоб зберегти історію пов’язаних дій.

Демо-паролі після переходу з навчального середовища в робоче потрібно замінити персональними обліковими записами.

4.8.3 Резервне копіювання та відновлення даних

PostgreSQL зберігає звернення, заявки на переклад, відгуки та довідник організацій, тому для системи потрібні регулярні резервні копії. Дамп має містити структуру схеми datacheck_connect і актуальні дані.

Копії слід зберігати окремо від основного сервера та обмежити доступ до них, оскільки вони можуть містити персональні дані. Не менш важливо періодично перевіряти відновлення копії на окремій базі.

У разі збою сервер застосунку зупиняють, зберігають поточний стан для аналізу, розгортають останню справну копію, перевіряють таблиці й запускають контрольні сценарії.

4.8.4 Моніторинг роботи серверної частини

Моніторинг має показувати, чи працює HTTP-сервер, чи доступна БД і чи повертаються дані для публічного інтерфейсу. Для базової перевірки в проєкті використовується маршрут /api/health.

У робочому середовищі помилки бажано записувати в окремий журнал: час, маршрут API, HTTP-статус і короткий технічний опис без зайвого виведення персональних даних.

Окремо потрібно контролювати службові сценарії: створення послуг, авторизацію, взяття заявки в роботу та завершення заявки виконавцем.

4.8.5 Якість даних та перевірка введення

Якість даних залежить від перевірки форм і довідників. У публічних сценаріях обов’язковими є місто, контакт, категорія, дата або час візиту, мова перекладу та згода на обробку даних.

Перевірка виконується на клієнтському і серверному рівнях. Для координат діють CHECK-обмеження, для відгуків — рейтинг від однієї до п’яти зірок, а під час редагування послуг адміністратор має уникати дублювання записів.

Повідомлення про помилки мають бути конкретними: наприклад, “заповніть контакт”, “оберіть дату візиту” або “підтвердьте згоду на обробку даних”.

4.8.6 Можливості масштабування та подальшого розвитку

Архітектура прототипу дозволяє розширити систему в кількох напрямах: поглибити аналітику, деталізувати ролі, додати сповіщення та покращити карту.

Для карти можна використати кластеризацію, маршрути й додаткові фільтри. Для скринінгу — конструктор анкет із питаннями, що залежать від попередніх відповідей користувача.

З технічного боку доцільно додати двофакторну авторизацію, журнал аудиту, обмеження кількості спроб входу та production-конфігурацію Docker Compose.

4.8.7 Організаційні та етичні аспекти експлуатації

Сервіс підтримки має бути технічно надійним і зрозумілим для користувача. Через це службові панелі приховані від публічної навігації, а форми не повинні вимагати зайвих персональних даних.

Текст згоди має пояснювати, які дані збираються, навіщо вони потрібні та хто працюватиме із зверненням. У формі відгуку бажано попереджати користувача, щоб він не вводив надмірно чутливі подробиці.

Виконавець після взяття заявки в роботу повинен завершити її або повідомити координатора про проблему. Такий порядок підтримує прозорий життєвий цикл задачі та відповідальність учасників процесу.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі розроблено та описано програмне забезпечення DATACHECK Connect, призначене для допомоги громадянам України з інфекційними захворюваннями у Німеччині. Система поєднує публічний каталог послуг, інтерактивну карту організацій, форму скринінгу, заявку на асистований переклад, форму відгуку, адмін-панель і кабінет виконавця.

Проведено системний аналіз предметної області, визначено ролі клієнта, адміністратора, виконавця й організації-надавача, сформовано функціональні та нефункціональні вимоги. Показано, що основна проблема полягає у потребі об’єднати довідкові, картографічні та операційні функції в одному інтерфейсі.

Спроєктовано інформаційне забезпечення системи. Розроблена ER-модель включає таблиці клієнтів, згод, категорій, послуг, організацій, адрес, користувачів, звернень, скринінгу, зв’язків послуг, заявок перекладу та відгуків. Фізична реалізація в PostgreSQL містить ключі, обмеження, індекси, перелічувані типи та тригери.

Спроєктовано програмну архітектуру у вигляді пакетної, компонентної та розгортувальної структури. Визначено взаємодію public/app.js, server.js, src/apiData.js, src/db.js, src/mappers.js, database/schema.sql і PostgreSQL. Реалізовано захищені службові маршрути, авторизацію за ролями, окремі вікна адміністратора та виконавця.

Розроблено клієнтський інтерфейс із мультимовністю, каталогом, картою OpenStreetMap через Leaflet, формами звернень, модальними деталями послуг і формою відгуку в лівому меню. Адмін-панель підтримує керування послугами, користувачами та зверненнями. Кабінет виконавця підтримує взяття заявки на переклад у роботу та її завершення.

Підготовлено рекомендації щодо тестування, запуску, апаратних і програмних вимог, складу інсталяційного пакету й подальшого розвитку. Мета

роботи досягнута: створено працюючий прототип веб-системи з базою даних, API, службовими ролями і документованою архітектурою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PostgreSQL Global Development Group. PostgreSQL Documentation. [Електронний ресурс]. — Режим доступу: https://www.postgresql.org/docs/

2. OpenJS Foundation. Node.js Documentation. [Електронний ресурс]. — Режим доступу: https://nodejs.org/docs/

3. Express.js. Express Documentation. [Електронний ресурс]. — Режим доступу: https://expressjs.com/

4. node-postgres. node-postgres Documentation. [Електронний ресурс]. — Режим доступу: https://node-postgres.com/

5. MDN Web Docs. HTML, CSS, JavaScript Documentation. [Електронний ресурс]. — Режим доступу: https://developer.mozilla.org/

6. Leaflet. Leaflet Documentation. [Електронний ресурс]. — Режим доступу: https://leafletjs.com/

7. OpenStreetMap Wiki. OpenStreetMap Documentation. [Електронний ресурс]. — Режим доступу: https://wiki.openstreetmap.org/

8. Docker Docs. Docker Compose Documentation. [Електронний ресурс]. — Режим доступу: https://docs.docker.com/compose/

9. OWASP Foundation. Application Security Verification Standard. [Електронний ресурс]. — Режим доступу: https://owasp.org/www-project-application-security-verification-standard/

10. European Union. General Data Protection Regulation, Regulation (EU) 2016/679. [Електронний ресурс]. — Режим доступу: https://eur-lex.europa.eu/eli/reg/2016/679/oj

11. Federal Government of Germany. Germany4Ukraine: official information portal for refugees from Ukraine. [Електронний ресурс]. — Режим доступу: https://www.germany4ukraine.de/

12. UNHCR Germany. Information for refugees from Ukraine in Germany. [Електронний ресурс]. — Режим доступу: https://help.unhcr.org/germany/

ДОДАТКИ

ДОДАТОК А

Фрагмент серверних маршрутів

```
001: function signToken(payload) {
002: function verifyToken(token) {
003: function requireRole(role) {
004: app.get('/admin', (_req, res) => {
005: app.get('/executor', (_req, res) => {
006: app.post('/api/auth/login', asyncRoute(async (req, res) => {
007: app.get('/api/health', asyncRoute(async (_req, res) => {
008: app.get('/api/bootstrap', asyncRoute(async (_req, res) => {
009: app.get('/api/admin/bootstrap', requireRole('ADMIN'), asyncRoute(async (_req, res)
     => {
010: app.get('/api/executor/bootstrap', requireRole('EXECUTOR'), asyncRoute(async (_req,
     res) => {
011: app.get('/api/services', asyncRoute(async (_req, res) => {
012: app.post('/api/screening', asyncRoute(async (req, res) => {
013: app.post('/api/translation-tasks', asyncRoute(async (req, res) => {
014: app.post('/api/feedbacks', asyncRoute(async (req, res) => {
015: app.post('/api/executor/tasks/:id/take', requireRole('EXECUTOR'), asyncRoute(async
     (req, res) => {
016: app.post('/api/executor/tasks/:id/complete', requireRole('EXECUTOR'),
     asyncRoute(async (req, res) => {
017: app.post('/api/admin/services', requireRole('ADMIN'), asyncRoute(async (req, res) =>
      {
018: app.put('/api/admin/services/:id', requireRole('ADMIN'), asyncRoute(async (req, res)
      => {
019: app.post('/api/admin/users', requireRole('ADMIN'), asyncRoute(async (req, res) => {
020: app.put('/api/admin/users/:id', requireRole('ADMIN'), asyncRoute(async (req, res) =>
      {
021: app.delete('/api/admin/requests/:id', requireRole('ADMIN'), asyncRoute(async (req,
     res) => {
022: app.post('/api/admin/tasks/:id/complete', requireRole('ADMIN'), asyncRoute(async
     (req, res) => {
023: app.get('*', (_req, res) => {
```

ДОДАТОК Б

Фрагмент SQL-схеми

```
001: -- DATACHECK Connect
002: -- Структура бази даних на фізичному рівні
003: -- Цільова СУБД: PostgreSQL
004:
005: CREATE SCHEMA IF NOT EXISTS datacheck_connect;
006: SET search_path TO datacheck_connect, public;
007:
008: -- ========================
009: -- Перелічувані типи
010: -- ========================
011: DO $$
012: BEGIN
013:     IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'request_status_enum') THEN
014:         CREATE TYPE request_status_enum AS ENUM (
015:             'NEW',
016:             'SCREENING',
017:             'ROUTED',
018:             'IN_PROGRESS',
019:             'COMPLETED',
020:             'CLOSED',
021:             'CANCELLED'
022:         );
023:     END IF;
024:
025:     IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'request_priority_enum')
    THEN
026:         CREATE TYPE request_priority_enum AS ENUM (
027:             'LOW',
028:             'MEDIUM',
029:             'HIGH',
030:             'URGENT'
031:         );
032:     END IF;
033:
034:     IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'organization_status_enum')
     THEN
035:         CREATE TYPE organization_status_enum AS ENUM (
036:             'ACTIVE',
037:             'INACTIVE',
038:             'SUSPENDED'
039:         );
040:     END IF;
041:
042:     IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'translation_status_enum')
    THEN
043:         CREATE TYPE translation_status_enum AS ENUM (
044:             'REQUESTED',
045:             'SCHEDULED',
046:             'IN_PROGRESS',
047:             'DONE',
048:             'CANCELLED'
049:         );
050:     END IF;
051:
052:     IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'delivery_mode_enum') THEN
053:         CREATE TYPE delivery_mode_enum AS ENUM (
054:             'ONLINE',
055:             'OFFLINE',
056:             'HYBRID'
057:         );
058:     END IF;
059:
060:     IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'user_role_enum') THEN
061:         CREATE TYPE user_role_enum AS ENUM (
062:             'ADMIN',
063:             'COORDINATOR',
```

```
064:             'CASE_MANAGER',
065:             'TRANSLATOR',
066:             'VIEWER'
067:         );
068:     END IF;
069: END $$;
070:
071: -- =========================
072: -- Службова функція для updated_at
073: -- =========================
074: CREATE OR REPLACE FUNCTION set_updated_at()
075: RETURNS trigger
076: LANGUAGE plpgsql
077: AS $$
078: BEGIN
079:     NEW.updated_at := CURRENT_TIMESTAMP;
080:     RETURN NEW;
081: END;
082: $$;
083:
084: -- =========================
085: -- Основні таблиці
086: -- =========================
087:
088: -- 1. Клієнт
089: CREATE TABLE IF NOT EXISTS client (
090:     client_id            BIGSERIAL PRIMARY KEY,
091:     external_code        VARCHAR(50) UNIQUE,
092:     first_name           VARCHAR(100) NOT NULL,
093:     last_name            VARCHAR(100) NOT NULL,
094:     date_of_birth        DATE,
095:     gender               VARCHAR(30),
096:     phone                VARCHAR(30),
097:     email                VARCHAR(255),
098:     preferred_language   CHAR(2) NOT NULL DEFAULT 'uk',
099:     country              VARCHAR(100) NOT NULL DEFAULT 'Germany',
100:     city                 VARCHAR(100),
101:     consent_required     BOOLEAN NOT NULL DEFAULT TRUE,
102:     notes                TEXT,
103:     created_at           TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
104:     updated_at           TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
105: );
106:
107: -- 2. Згода на обробку даних
108: CREATE TABLE IF NOT EXISTS consent (
109:     consent_id           BIGSERIAL PRIMARY KEY,
110:     client_id            BIGINT NOT NULL REFERENCES client(client_id) ON DELETE
     CASCADE,
111:     is_given             BOOLEAN NOT NULL,
112:     consent_version      VARCHAR(30) NOT NULL,
113:     legal_basis          VARCHAR(255) NOT NULL,
114:     given_at             TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
115:     revoked_at           TIMESTAMPTZ,
116:     created_at           TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
117:     CONSTRAINT chk_consent_dates CHECK (revoked_at IS NULL OR revoked_at >=
     given_at)
118: );
119:
120: -- 3. Категорія підтримки
121: CREATE TABLE IF NOT EXISTS support_category (
122:     category_id          BIGSERIAL PRIMARY KEY,
123:     name                 VARCHAR(120) NOT NULL UNIQUE,
124:     description          TEXT,
125:     is_active            BOOLEAN NOT NULL DEFAULT TRUE,
126:     created_at           TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
127: );
128:
129: -- 4. Послуга
130: CREATE TABLE IF NOT EXISTS service_item (
131:     service_id           BIGSERIAL PRIMARY KEY,
132:     category_id          BIGINT NOT NULL REFERENCES support_category(category_id) ON
```

```
      DELETE RESTRICT,
133:     name                 VARCHAR(150) NOT NULL,
134:     description          TEXT,
135:     delivery_mode        delivery_mode_enum NOT NULL DEFAULT 'HYBRID',
136:     is_active            BOOLEAN NOT NULL DEFAULT TRUE,
137:     created_at           TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
138:     CONSTRAINT uq_service_item UNIQUE (category_id, name)
139: );
140:
141: -- 5. Організація
142: CREATE TABLE IF NOT EXISTS organization (
143:     organization_id      BIGSERIAL PRIMARY KEY,
144:     name                 VARCHAR(200) NOT NULL UNIQUE,
145:     email                VARCHAR(255),
146:     phone                VARCHAR(30),
147:     website              VARCHAR(255),
148:     working_hours        VARCHAR(120),
149:     status               organization_status_enum NOT NULL DEFAULT 'ACTIVE',
150:     notes                TEXT,
151:     created_at           TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
152:     updated_at           TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
153: );
154:
155: -- 6. Локація / адреса організації
156: CREATE TABLE IF NOT EXISTS address (
157:     location_id          BIGSERIAL PRIMARY KEY,
158:     organization_id      BIGINT NOT NULL REFERENCES organization(organization_id) ON
      DELETE CASCADE,
159:     country              VARCHAR(100) NOT NULL DEFAULT 'Germany',
160:     region               VARCHAR(100),
... фрагмент скорочено; у проєкті файл містить 351 рядок.
```

ДОДАТОК В

Фрагмент реалізації карти OpenStreetMap

```
let leafletMap = null;

function mapFallbackCoords(cityKey) {
  const coords = {
    berlin: [52.5200, 13.4050],
    hamburg: [53.5511, 9.9937],
    munich: [48.1351, 11.5820],
    cologne: [50.9375, 6.9603],
    leipzig: [51.3397, 12.3731],
  };
  return coords[cityKey] || coords.berlin;
}

function initLeafletMap(orgs) {
  const mapElement = document.getElementById('osmMap');
  if (!mapElement) return;

  if (leafletMap) {
    leafletMap.remove();
    leafletMap = null;
  }

  if (typeof L === 'undefined') {
    mapElement.innerHTML = `<div class="muted-box">${getLocalized({ uk: 'Leaflet не завантажився. Перевір підключення до інтернету для тайлів OpenStreetMap.', en: 'Leaflet did not load. Check the internet connection for OpenStreetMap tiles.', de: 'Leaflet wurde nicht geladen. Prüfe die Internetverbindung für OpenStreetMap-Kacheln.' })}</div>`;
    return;
  }

  leafletMap = L.map(mapElement, { scrollWheelZoom: true });
  L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
  }).addTo(leafletMap);

  const bounds = [];
  orgs.forEach((org) => {
    const fallback = mapFallbackCoords(org.city);
    const lat = Number.isFinite(Number(org.lat)) ? Number(org.lat) : fallback[0];
    const lon = Number.isFinite(Number(org.lon)) ? Number(org.lon) : fallback[1];
    bounds.push([lat, lon]);
    const popup = `
      <div class="map-popup">
        <strong>${escapeHtml(org.org)}</strong>
        <div>${escapeHtml(cityLabel(org.city))} • ${escapeHtml(org.address)}</div>
        <div class="popup-services">
          ${org.services.map((service) => `<button class="inline-link map-service-link" data-map-service="${service.id}" type="button">${escapeHtml(service.name)}</button>`).join('')}
        </div>
      </div>
    `;
    L.marker([lat, lon]).addTo(leafletMap).bindPopup(popup);
  });

  if (bounds.length > 1) {
    leafletMap.fitBounds(bounds, { padding: [30, 30] });
  } else if (bounds.length === 1) {
    leafletMap.setView(bounds[0], 13);
  } else {
    leafletMap.setView([51.1657, 10.4515], 6);
  }
```

```
060:   setTimeout(() => leafletMap?.invalidateSize(), 50);
061: }
062:
063: function renderMap() {
064:   if (!root.views.map) return;
065:   const orgs = getOrganizationsForMap();
066:   root.views.map.innerHTML = `
067:     <div class="map-grid">
068:       <section class="map-board">
069:         <div class="section-title-row">
070:           <div>
071:             <div class="section-title">${t('map.title')}</div>
072:             <div class="soft-text">${t('map.lead')}</div>
073:           </div>
074:           <button class="secondary-btn" id="mapClear">${t('map.clear')}</button>
075:         </div>
076:         <div class="city-filter-row space-top">
077:           ${['all', 'berlin', 'hamburg', 'munich', 'cologne', 'leipzig'].map((city)
     => `
078:             <button class="filter-chip ${state.mapCity === city ? 'active' : ''}"
     data-map-city="${city}">${city === 'all' ? t('common.all') :
     cityLabel(city)}</button>
079:           `).join('')}
080:         </div>
081:         <div id="osmMap" class="osm-map" role="application" aria-
     label="OpenStreetMap"></div>
082:       </section>
083:
084:       <section class="section-block">
085:         <div class="section-title">${t('map.cityListTitle')}</div>
086:         <div class="soft-text">${state.mapCity === 'all' ? t('catalog.allCities') :
     cityLabel(state.mapCity)}</div>
087:         <div class="city-list">
088:           ${orgs.length ? orgs.map((org) => `
089:             <div class="city-item map-org-item">
090:               <div class="icon-badge">${org.count}</div>
091:               <div>
092:                 <div class="card-title">${escapeHtml(org.org)}</div>
093:                 <div class="soft-text">${cityLabel(org.city)} •
     ${escapeHtml(org.address)}</div>
094:                 <div class="badge-row space-top">
095:                   <span class="tag">${org.count} ${t('map.services')}</span>
096:                   ${org.services.map((service) => `
097:                     <button class="tag tag-button" data-map-service="${service.id}"
     type="button">${escapeHtml(service.name)}</button>
098:                   `).join('')}
099:                 </div>
100:               </div>
101:             </div>
102:           `).join('') : `<div class="muted-box">${t('map.noCity')}</div>`}
103:         </div>
104:       </section>
105:     </div>
106:   `;
107:
108:   root.views.map.querySelectorAll('[data-map-city]').forEach((btn) =>
     btn.addEventListener('click', () => {
109:     state.mapCity = btn.dataset.mapCity;
110:     renderMap();
... фрагмент скорочено; у проєкті файл містить 124 рядки.
```

ДОДАТОК Г

Інструкція користувача та адміністратора

Для запуску проєкту потрібно розпакувати архів, відкрити термінал у папці DC_Connect, виконати npm install, запустити PostgreSQL або docker compose up -d, створити файл .env на основі .env.example, виконати npm run db:init та npm start. Після запуску публічний інтерфейс доступний за адресою http://localhost:3000.

Адміністратор відкриває http://localhost:3000/admin, вводить логін і пароль, після чого бачить службову панель. У панелі можна переглядати KPI, фільтрувати звернення, видаляти тестові або помилкові звернення, додавати й редагувати послуги, створювати і редагувати користувачів системи.

Виконавець відкриває http://localhost:3000/executor, входить у кабінет, переглядає доступні заявки на переклад, натискає “Взяти задачу”, виконує переклад у реальному процесі й після цього натискає “Завершити заявку”. Статус задачі оновлюється в базі даних і стає видимим адміністратору.

Клієнт користується тільки публічним інтерфейсом. Він може переглянути каталог, застосувати фільтри, перейти на карту, натиснути на послугу організації, пройти скринінг, створити заявку на переклад або залишити відгук через окремий блок лівого меню.

ДОДАТОК Д

Контрольні сценарії приймального тестування

Для приймального тестування DATACHECK Connect доцільно використовувати сценарії, які повторюють реальний порядок роботи користувачів. Перший сценарій перевіряє повний шлях клієнта від відкриття публічного інтерфейсу до створення звернення зі скринінгу. Перевіряється заповнення полів, згода на обробку даних, запис у БД і поява нового кейсу в адмін-панелі.

Другий сценарій перевіряє шлях створення заявки на асистований переклад. Користувач створює заявку, адміністратор бачить її в панелі, виконавець входить у кабінет, бере задачу в роботу, завершує її, а адміністратор бачить змінений статус. Цей сценарій є ключовим, тому що підтверджує взаємодію публічної частини, API, БД і кабінету виконавця.

Третій сценарій перевіряє адміністрування каталогу. Адміністратор додає нову послугу з організацією, адресою та координатами. Після збереження послуга повинна з’явитися в каталозі, а організація — на карті. Потім адміністратор редагує назву або адресу, і зміни знову відображаються в публічній частині.

Четвертий сценарій перевіряє керування користувачами. Адміністратор створює нового виконавця, задає логін і пароль, після чого цей виконавець може увійти в /executor. Якщо користувача деактивовано або роль змінено, доступ повинен змінитися відповідно до нових даних.

П’ятий сценарій перевіряє відгуки. Користувач відкриває форму відгуку, ставить оцінку, надсилає коментар і після цього адміністратор бачить зміну середнього рейтингу. Такий тест підтверджує, що блок відгуку працює як окремий публічний сценарій, а не як декоративний елемент інтерфейсу.

Таблиця Д.1 – Контрольні приймальні сценарії

№	Назва сценарію	Критерій успішності
1	Створення звернення зі скринінгу	Запис з’являється в БД і журналі адміністратора.
2	Створення та завершення заявки перекладу	Задача проходить стани REQUESTED, IN_PROGRESS і DONE.
3	Додавання послуги адміністратором	Послуга з’являється в каталозі та на карті.
4	Створення службового користувача	Новий користувач може увійти відповідно до ролі.
5	Видалення звернення	Звернення зникає з журналу і не порушує цілісність пов’язаних даних.
6	Надсилання відгуку	Запис feedback створюється, середній рейтинг оновлюється.
7	Захист службового API	Запит без токена повертає 401 Unauthorized.

ДОДАТОК Е

Приклад конфігурації середовища

Файл .env використовується для відокремлення налаштувань від вихідного коду. У ньому задається рядок підключення до PostgreSQL, назва схеми, порт сервера, секрет підпису токенів і демонстраційні облікові записи. У реальному середовищі цей файл не повинен передаватися стороннім особам або публікуватися в репозиторії.

Під час перенесення проєкту на інший комп’ютер достатньо створити .env на основі .env.example, змінити пароль PostgreSQL і виконати npm run db:init. Якщо використовується Docker Compose, параметри підключення мають відповідати контейнеру БД. Якщо використовується локальний PostgreSQL, потрібно перевірити ім’я бази, порт і користувача.

Таблиця Е.1 – Основні змінні .env

Змінна	Призначення
DATABASE_URL	Рядок підключення до PostgreSQL.
DB_SCHEMA	Назва схеми БД, за замовчуванням datacheck_connect.
PORT	Порт HTTP-сервера Express.
AUTH_SECRET	Секрет для підпису службових токенів.

ДОДАТОК Ж

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Бігун Микола Миколайович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

- Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Програмне забезпечення для допомоги громадянам України з інфекційними захворюваннями у Німеччині» є результатом моєї особистої праці.
- Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
- Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [✓] інструменти ШІ ChatGPT були використані для:
        - [ ] перекладу іншомовних джерел;
        - [✓] стилістичного редагування та перевірки граматики;
        - [ ] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

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

«___» _____ 2026 р. _____ **Микола БІГУН**
(підпис)