НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

Програмне забезпечення працівника організації із страхування автотранспорту

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Мельников Д.О.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи студенту

Мельникову Дмитру Олександровичу

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення працівника організації із страхування автотранспорту

затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

Термін подання завершеної роботи на кафедру 2025.05.28
(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення,

Перелік питань що розглядаються:
1. Системний аналіз предметної області
2. Проєктування інформаційного та програмного забезпечення
3. Розробка інформаційного та програмного забезпечення
4. Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання «_____»_____2025 р.
Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Мельников Д.О.
(підпис) (ПІБ студента)

ЗМІСТ

ВСТУП....4
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ....6
1.1 Опис предметної області....6
1.2 Аналіз вимог до програмної системи....8
1.3 Моделювання предметної області....12
1.4 Огляд інформаційних джерел та існуючих рішень....18
1.5 Постановка завдання....24
2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ....25
2.1 Логічна модель даних у вигляді ER-діаграми....25
2.2 Діаграма класів та кооперацій....28
2.3 Діаграма пакетів....34
2.4 Діаграма компонентів....36
3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ....40
3.1 Система управління інформаційною базою....40
3.2 Розробка інформаційної бази....42
3.3 Вибір інструментарію для створення прикладного програмного забезпечення....46
3.4 Алгоритмізація та програмування програмних модулів....48
4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ....51
4.1 Тестування системи....51
4.2 Вимоги до апаратного та програмного забезпечення....55
4.3 Склад інсталяційного пакету....60
ВИСНОВКИ....62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ....64
ДОДАТОК А....67
ДОДАТОК Б....71

ВСТУП

У сучасних умовах цифровізації бізнесу стрімко зростає потреба в автоматизації внутрішніх процесів підприємств різних галузей, зокрема й у сфері страхування. Автострахування, як один із найбільш динамічних напрямів, вимагає швидкої та точної обробки даних, гнучкого управління договорами, зручної комунікації з клієнтами. Використання застарілих або частково автоматизованих рішень не дозволяє забезпечити необхідний рівень точності, швидкості й зручності обслуговування, що створює ризики помилок, дублювання даних та втрати інформації.

Розв'язання цієї проблеми можливе шляхом впровадження сучасного програмного забезпечення, яке забезпечує централізований облік клієнтів, транспортних засобів, договорів і страхових випадків, а також ефективну взаємодію працівників компанії з клієнтськими запитами. Найбільш доцільною архітектурною моделлю для реалізації таких вимог є веб-додаток, який забезпечує доступ з будь-якого пристрою, мінімальні витрати на інфраструктуру та зручність оновлення.

Об’єктом дослідження є система цифрової взаємодії між працівниками автострахової компанії та клієнтами у контексті обліку, реєстрації та обробки страхових заявок. **Предметом** дослідження є методи проєктування та реалізації веб-додатку для автоматизації роботи працівника страхової компанії з обробки заявок та управління базою клієнтів.

Метою роботи є розробка та впровадження веб-додатку, що дозволяє ефективно виконувати ключові операції зі страховими договорами, забезпечуючи надійне зберігання даних, доступність та простоту використання.

Для досягнення поставленої мети передбачено виконання таких завдань:

- провести аналіз предметної області та визначити функціональні вимоги до системи;
- спроєктувати архітектуру веб-додатку та структуру бази даних;

- реалізувати клієнтську та серверну частини програмного забезпечення;
- протестувати систему на відповідність технічним і функціональним вимогам;
- сформулювати рекомендації щодо подальшого розширення функціональності та впровадження системи в реальних умовах.

Робота містить опис етапів розробки веб-додатку на основі технологій PHP, HTML, CSS, JavaScript та реляційної бази даних. Також проаналізовано аналогічні рішення, подано діаграми, які відображають логіку функціонування системи, і наведено результати тестування, що підтверджують її функціональність і придатність до використання.

Апробація розробленого веб-додатку здійснювалася шляхом тестування функціоналу в умовах, наближених до практичного використання. Результати розробки можуть бути адаптовані до потреб інших страхових компаній та слугувати основою для подальшого розвитку інформаційних систем у цій галузі.

Пояснювальна записка містить чотири розділи, у яких послідовно розглянуто аналіз предметної області, проєктування інформаційної системи, реалізацію програмних модулів і тестування. Робота включає діаграми, фрагменти коду, приклади роботи системи, а також перелік використаних джерел і додатки. Загальний обсяг записки становить __ сторінок.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сфера автострахування є важливою складовою ринку фінансових послуг, яка охоплює діяльність зі страхового захисту власників транспортних засобів від ризиків, пов’язаних із пошкодженням, втратою або знищенням майна внаслідок аварій, стихійних лих чи протиправних дій третіх осіб. Для ефективного функціонування страхової компанії в цій галузі необхідна постійна робота з великою кількістю даних: облік клієнтів, оформлення договорів, супровід страхових подій, фінансовий аналіз і генерація звітності. Зважаючи на обсяг інформації та високу інтенсивність обміну даними, ручна або частково автоматизована обробка є неефективною та супроводжується значними ризиками помилок або дублювання.

Автоматизація ключових бізнес-процесів дозволяє підвищити точність, швидкість обробки інформації та забезпечити надійний контроль за виконанням страхових операцій. Основою предметної області є організація процесів, пов’язаних з реєстрацією клієнтів, оформленням полісів, обробкою заявок, веденням баз даних, супроводом страхових випадків та аналізом результатів діяльності. Центральним елементом цих процесів є інтерактивна взаємодія між працівником компанії та клієнтом через веб-інтерфейс, що забезпечує введення, збереження, перевірку та перегляд даних у режимі реального часу.

Враховуючи вищенаведене, предметною областю даного дослідження є діяльність автострахових компаній у частині обробки клієнтських даних, управління полісами та супроводу страхових подій. Програмне забезпечення, розроблене в межах цієї роботи, орієнтоване на автоматизацію типових функцій працівника страхової компанії, що дозволяє не лише зменшити час обробки заявок, а й підвищити рівень контролю та прозорості виконання страхових операцій.

У таблиці 1.1 наведено структурований перелік основних задач, які охоплює предметна область. Ці задачі формують основу інформаційної моделі системи та визначають функціональні вимоги до розроблюваного програмного забезпечення.

Таблиця 1.1

Основні задачі предметної області в автострахуванні

№	Назва задачі	Опис задачі
1	Оформлення страхових полісів	Реєстрація нових полісів: введення даних про клієнта, автомобіль, умови страхування.
2	Обробка заявок	Прийом і обробка заявок, перевірка даних, оновлення інформації, підтвердження або відхилення.
3	Ведення клієнтської бази	Управління обліковими записами клієнтів, полісами, транспортними засобами та історією заявок.
4	Управління страховими випадками	Облік повідомлень про настання страхових подій, ведення відповідної документації.
5	Моніторинг та звітність	Генерація звітів, фінансовий контроль, аналіз статистики щодо оброблених заявок і виплат.

Візуальне представлення задач предметної області наведено на рис. 1.1, де відображено основні функціональні блоки системи, їхню логічну взаємодію та зв’язки між користувачами й процесами.

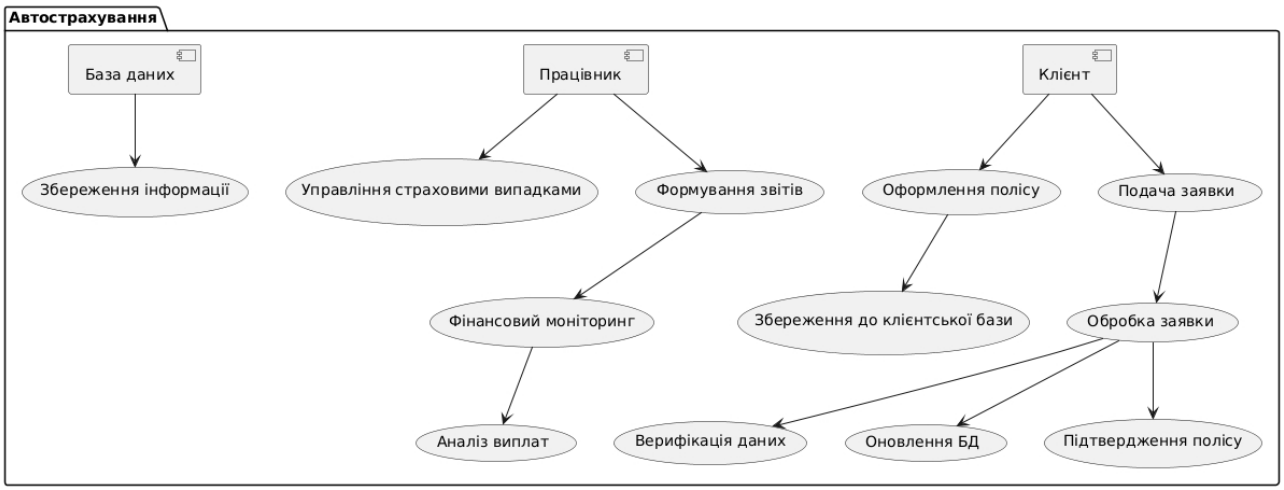

Рис. 1.1 Основні задачі предметної області автострахування

Опис предметної області дозволив визначити основні напрямки функціонування системи автострахування та сформулювати перелік ключових задач, які мають бути автоматизовані. Аналіз логіки роботи страхової компанії дав змогу виявити базові процеси, що потребують інтеграції до інформаційної системи, і заклав підґрунтя для формування вимог до архітектури та функціональності розроблюваного веб-додатку.

1.2 Аналіз вимог до програмної системи

Розроблюване програмне забезпечення призначене для автоматизації повсякденних операцій, що виконуються працівником автострахової компанії. Його функціональність має охоплювати повний цикл обробки клієнтських звернень – від реєстрації користувачів і оформлення страхових полісів до супроводу заявок і ведення клієнтської бази. Враховуючи специфіку предметної області, система повинна бути надійною, безпечною, зручною у користуванні та сумісною з різними типами пристроїв [20].

Ключовими аспектами функціональних вимог є забезпечення ідентифікації користувача, облік і оновлення даних клієнтів, обробка страхових заявок, керування профілем працівника та підтримка адаптивного інтерфейсу. Також важливою є можливість зручного перегляду історії страхування та змінення статусів заявок відповідно до їхнього стану опрацювання. Деталізований перелік основних функцій наведено у таблиці 1.2, яка представлена як узагальнена специфікація функціональних вимог до програмної системи.

Таблиця 1.2

Специфікація функціональних вимог

№	Основні вимоги	Призначення та використання
1	2	3
1	Авторизація користувача	Забезпечення входу до системи лише уповноваженим працівникам.
2	Облік клієнтів	Додавання нового клієнта, перегляд, редагування чи видалення існуючих записів.

Таблиця 1.1 (продовження)

1	2	3
3	Створення страхового поліса	Формування поліса з урахуванням типу транспортного засобу, терміну дії, суми покриття.
4	Обробка заявок	Перегляд та зміна статусу заявок.
5	Редагування профілю працівника	Зміна пароля, контактної інформації, перегляд особистих дій у системі.
6	Адаптивний інтерфейс	Повноцінна робота з додатком через мобільні або планшетні пристрої.

Функціональні вимоги визначають основні можливості програмної системи, необхідні для ефективного виконання повсякденних завдань працівника автострахової компанії. Вони охоплюють усі ключові етапи взаємодії з клієнтом – від створення облікового запису до формування страхового полісу. Серед базових функцій важливе місце займає авторизація, що забезпечує доступ до системи лише для перевірених користувачів, зберігаючи конфіденційність даних клієнтів. Також реалізовано функції внесення персональної інформації клієнта, перегляду історії страхування та редагування записів, що сприяє покращенню якості обслуговування.

Окрему роль відіграє модуль створення страхового полісу, який дозволяє вводити параметри договору, дані про транспортний засіб і автоматично формувати поліс. Обробка клієнтських заявок передбачає перегляд, перевірку та зміну їхнього статусу. Для зручності працівника реалізовано можливість редагування власного профілю. Адаптивність інтерфейсу гарантує стабільну та зручну роботу системи на різних пристроях, що особливо важливо за умов віддаленого доступу чи роботи в мобільному середовищі.

Загалом, ці вимоги забезпечують реалізацію основних бізнес-процесів компанії та створюють передумови для надійної, безпечної й зручної експлуатації системи в реальних умовах.

Нефункціональні вимоги визначають якісні характеристики програмного забезпечення, які не стосуються безпосередньо функціоналу, але критично впливають на зручність, надійність, продуктивність, безпеку та підтримуваність

системи. У контексті розробки веб-додатку для працівника автострахової компанії ці вимоги є основою для побудови стійкої, безпечної та масштабованої інформаційної платформи, що відповідає сучасним очікуванням користувачів.

Продуктивність передбачає швидке реагування на дії користувача та обробку запитів у межах однієї секунди, а надійність – тривалу безперебійну роботу без збоїв. Безпека гарантується завдяки авторизації, шифруванню даних і захисту від типових кіберзагроз. Важливою є також зручність інтерфейсу, що дозволяє працівнику ефективно взаємодіяти із системою без додаткового навчання. Масштабованість забезпечує можливість розширення функціоналу без зміни архітектури, портативність – коректну роботу на різних пристроях і браузерах, супроводжуваність – легкість у підтримці та оновленні завдяки модульному й задокументованому коду, а відмовостійкість – збереження введених даних та автоматичне відновлення після збою.

У таблиці 1.3 наведено узагальнену специфікацію основних нефункціональних вимог, які мають бути реалізовані в системі.

Таблиця 1.3

Специфікація нефункціональних вимог

№	Вимога	Опис
1	Продуктивність	Відповідь системи на користувацькі дії не повинна перевищувати 1 с.
2	Надійність	Мінімізація збоїв і втрачених даних при багатогодинній роботі.
3	Безпека	Захист персональних даних, авторизація, права доступу.
4	Зручність інтерфейсу	Простий та логічний дизайн, мінімум дій для виконання ключових задач.
5	Масштабованість	Легка адаптація системи до зростаючих обсягів даних або функцій.
6	Портативність	Підтримка роботи в різних браузерах і на різних пристроях.
7	Супроводжуваність	Структурованість і коментованість коду для подальшої розробки.
8	Відмовостійкість	Автоматичне збереження та відновлення в разі збою.

Візуальне представлення зазначених вимог подано на рисунку 1.2, який відображає ієрархію нефункціональних характеристик та їхній вплив на якість функціонування системи.

Рис. 1.2 Основні нефункціональні вимоги до програмної системи

Нефункціональні вимоги є критично важливими для створення якісного, ефективного та безпечного програмного продукту. Саме вони визначають параметри, за якими оцінюється зручність щоденної роботи працівника, продуктивність обслуговування клієнтів і стійкість до помилок та кіберзагроз. Наприклад, у разі недостатньої швидкодії або складного інтерфейсу система втрачає свою ефективність, незалежно від реалізованих функцій.

Без належного рівня безпеки зростає ризик витоку конфіденційної інформації, а за відсутності масштабованості – система втрачає актуальність у динамічних умовах ринку. Крім того, супроводжуваність та відмовостійкість є основою довготривалого життєвого циклу розробки та довіри користувачів до системи.

Таким чином, дотримання нефункціональних вимог є фундаментом надійної, зручної та конкурентоспроможної інформаційної системи, що здатна

гнучко адаптуватися до потреб автострахового бізнесу в умовах цифрової трансформації.

1.3 Моделювання предметної області

Предметна область у контексті створення інформаційної системи охоплює сукупність процесів, об'єктів і взаємозв’язків, що відображають реальну діяльність організації та потребують автоматизації. У даному випадку йдеться про цифрове представлення основних функцій автострахової компанії, пов’язаних із наданням страхових послуг, супроводом клієнтів, оформленням полісів і обробкою страхових подій. Саме ці процеси становлять інтерес для програмної реалізації та є базою для подальшого проєктування програмного забезпечення.

Моделювання предметної області є ключовим етапом розробки, оскільки на цьому етапі визначаються основні сутності, їхні атрибути та взаємозв’язки. У межах системи визначено такі сутності: Клієнт – фізична особа, яка звертається до компанії з метою страхування; Працівник – користувач, що адмініструє заявки та договори; Транспортний засіб – об'єкт страхування з характеристиками (марка, рік випуску, номер тощо); Заявка – початкове звернення клієнта; Страховий поліс – документ, що фіксує умови договору; Страховий випадок – подія, що вимагає розгляду компенсації; Платіж – інформація про фінансову операцію. Формалізація цих сутностей дозволяє структуровано організувати дані та відобразити логіку роботи системи.

Для графічного подання логіки предметної області використовується стандартизована мова моделювання UML, яка забезпечує чітке представлення архітектури, поведінки та взаємодій системи. UML підтримує основні принципи об’єктно-орієнтованого проєктування, зокрема абстрагування, багатоаспектне моделювання та ієрархічну деталізацію. Наприклад, діаграми класів дозволяють структурувати об’єкти системи й описати їх атрибути, діаграми прецедентів ілюструють взаємодію працівника із системою, а діаграми послідовностей – динамічний перебіг процесів, зокрема оформлення полісу [1].

Ретельне моделювання предметної області сприяє кращому розумінню функціональних вимог, дозволяє виявити слабкі місця в логіці обробки інформації, забезпечує коректну побудову структури бази даних і закладає основу для створення надійного, масштабованого й ефективного програмного рішення, що відповідає сучасним вимогам цифровізації страхових послуг.

Діаграма прецедентів є ключовим інструментом для моделювання функціональної поведінки інформаційної системи, оскільки дозволяє наочно представити взаємодію користувачів із системою через типові сценарії використання. Такий підхід допомагає чітко окреслити, які дії може виконувати актор і які функції повинні бути підтримані системою. При цьому прецеденти зосереджуються на описі функціональних вимог без деталізації їхньої технічної реалізації, що сприяє формуванню спільного бачення між усіма учасниками процесу розробки на ранніх етапах.

У межах даного проєкту було побудовано діаграму прецедентів для модуля працівника страхової компанії, який працює з автострахуванням. Основним актором є Працівник, що виконує основні дії в системі: обробку заявок, перегляд полісів, внесення змін у профіль, ведення обліку страхових випадків. Діаграма охоплює такі прецеденти: перегляд списку заявок, перевірка даних клієнта, підтвердження заявки (з можливим розширенням – надіслати повідомлення клієнту), відхилення заявки (з розширенням – вказати причину), перегляд інформації про поліс, редагування профілю працівника, перегляд страхових випадків.

Особливістю діаграми є наявність відношень типу <<extend>>, які демонструють розширення основного функціоналу. Наприклад, після підтвердження заявки система може автоматично надіслати повідомлення клієнту, а при відхиленні – запропонувати вказати причину. Такі механізми дозволяють адаптувати систему до практичних сценаріїв використання без ускладнення основної логіки.

Межі системи позначено прямокутником із назвою «Модуль обробки страхових заявок», усередині якого знаходяться всі прецеденти. Актор

розташований за межами системи та пов’язаний з відповідними варіантами використання. Побудова такої діаграми дозволяє структуровано представити функціональні вимоги, полегшує комунікацію між аналітиками, розробниками та замовником і слугує основою для створення сценаріїв використання, функціональних специфікацій і тестових сценаріїв. Діаграму прецедентів подано на рис. 1.3.

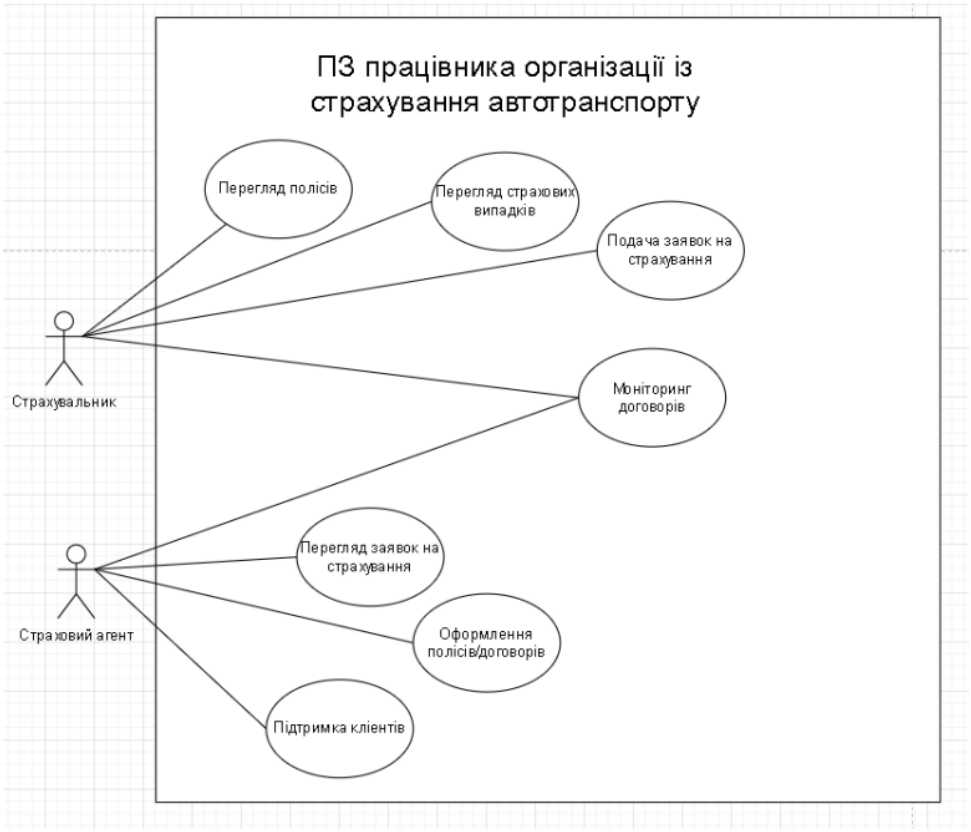

Рис.1.3 Діаграма прецендентів для веб-додатку із страхування автотранспорту

На діаграмі прецедентів зображено взаємодію двох основних акторів – Страхувальника та Страхового агента – із системою. Страхувальник здійснює подачу заявок на страхування, переглядає свої поліси та страхові випадки, а також відстежує чинні договори. Страховий агент, у свою чергу, переглядає подані заявки, формує страхові поліси та здійснює моніторинг договорів. Прецеденти охоплюють ключові процеси автострахування – від створення заявки до оформлення договору – і демонструють, як система підтримує основну діяльність користувачів на кожному етапі.

Діаграми послідовності (sequence diagrams) є важливим засобом моделювання часової взаємодії між об’єктами системи в межах конкретного сценарію використання. На відміну від діаграм прецедентів, які ілюструють загальні варіанти взаємодії користувачів із системою, діаграми послідовності деталізують порядок обміну повідомленнями між окремими компонентами, що беруть участь у реалізації функціональних процесів.

Кожен об’єкт зображається у вигляді прямокутника з підписом (ім’я та клас), а його "лінія життя" – вертикальною пунктирною лінією, що демонструє активність протягом певного часу. Передача повідомлень відображається спрямованими стрілками, які можуть бути синхронними або асинхронними, із зазначенням дій або викликів методів.

На рисунку 1.4 представлено діаграму послідовності, яка ілюструє взаємодію між трьома основними учасниками системи: Страхувальником, Страховим агентом та Системою (Страхове агентство). Дана діаграма охоплює реалізацію трьох ключових процесів: подачі нової заявки, перегляду списку заявок та створення страхового договору. У першому сценарії Страхувальник ініціює подання заявки, яка обробляється платформою, після чого система надсилає підтвердження. У процесі перегляду заявок Страховий агент надсилає запит на отримання списку, застосовує фільтри, а система, у свою чергу, повертає відповідні результати. На завершальному етапі агент ініціює створення договору на основі обраної заявки, а система генерує та повертає сформований договір із підтвердженням про завершення процедури.

Учасники системи мають наступні ролі: Страхувальник – клієнт, який ініціює звернення до платформи з метою страхування, переглядає свої дані та подає заявки; Страховий агент – працівник компанії, який опрацьовує ці заявки, переглядає список запитів, застосовує фільтри та створює поліси; Система страхового агентства – програмна платформа, що приймає запити, взаємодіє з базою даних і повертає відповідні результати або формує вихідні документи. Усі повідомлення між об’єктами подано у суворій часовій послідовності, що дозволяє повністю простежити логіку взаємодії в межах конкретного сценарію.

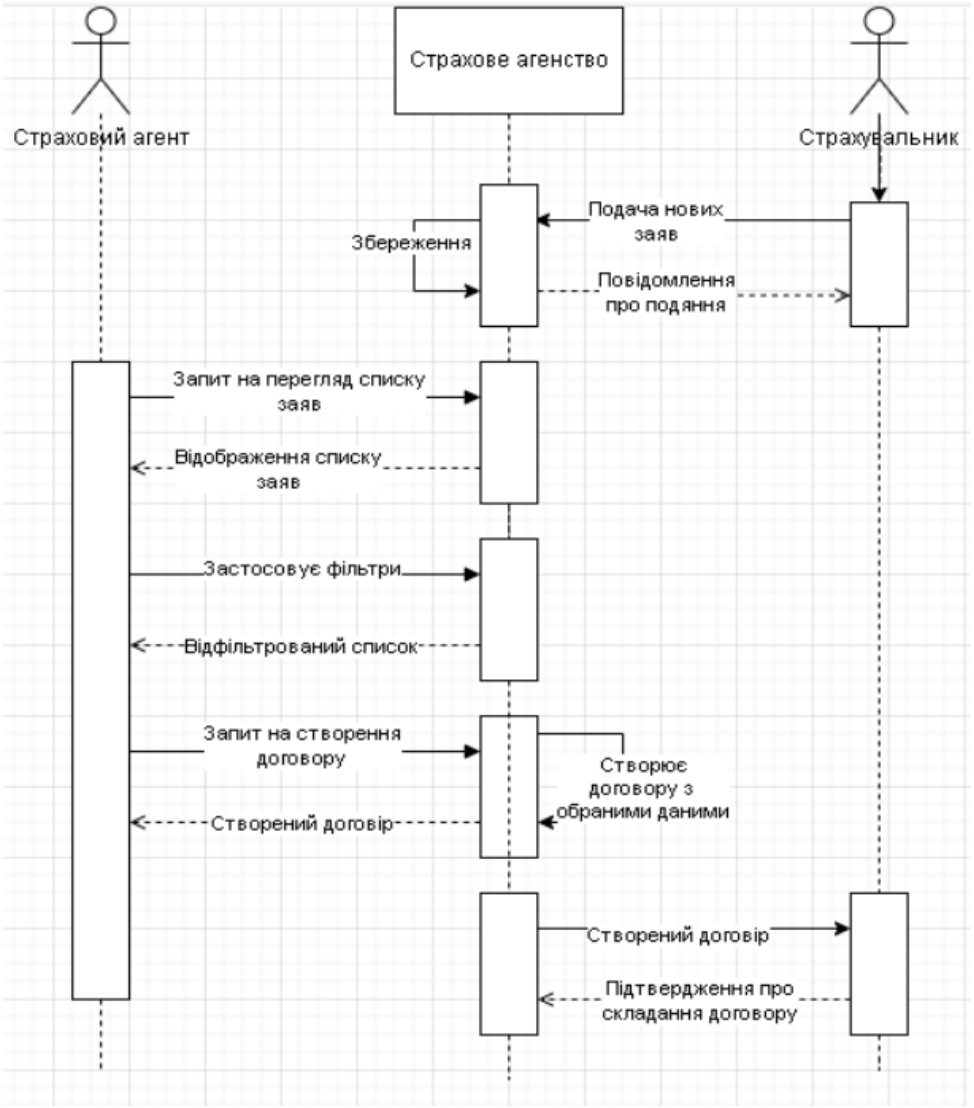

Рис.1.4 Діаграма послідовності для веб-додатку із страхування автотранспорту

Таким чином, діаграма послідовності, наведена на рис. 1.4, демонструє не лише структуру взаємодії, але й уточнює динаміку процесів у системі. Такий рівень деталізації дозволяє виявити потенційні вузькі місця, суперечності або відсутні елементи у функціональних сценаріях ще до етапу реалізації. Діаграма слугує зручним засобом комунікації між аналітиками, розробниками та зацікавленими сторонами, формуючи підґрунтя для подальшого програмування, тестування та розгортання програмного продукту.

Отже, діаграма послідовності є важливим інструментом для розуміння динамічної поведінки системи при виконанні ключових функціональних вимог, а також для проектування та розробки відповідних компонентів програмного

забезпечення. Вона допомагає візуалізувати потік подій та взаємодію між страхувальником, страховим агентом та системою страхового агентства.

Діаграма активності (рис. 1.5) використовується для графічного представлення алгоритму виконання процесу, що реалізується в межах інформаційної системи. Вона подібна до блок-схеми, однак орієнтована на візуалізацію логіки дій та переходів між ними з акцентом на умови, що визначають послідовність операцій. Кожен елемент на діаграмі відповідає певній активності або рішенню, а переходи між станами відбуваються лише після завершення попереднього етапу. Такий підхід забезпечує чітку та контрольовану логіку виконання сценаріїв взаємодії між учасниками процесу.

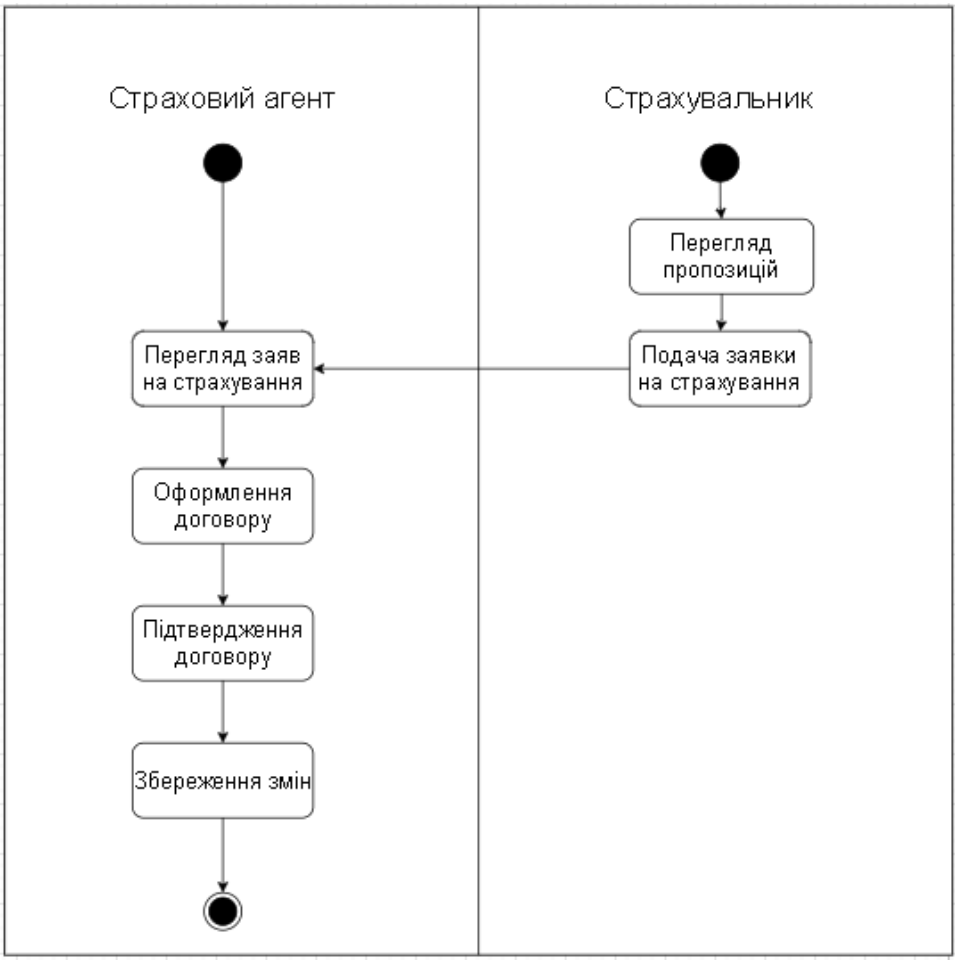

Рис. 1.5 Діаграма активності для веб-додатку із страхування автотранспорту

На діаграмі активності зображено взаємодію між Страхувальником (клієнтом) та Страховим агентом (представником компанії) у процесі

оформлення страхового договору на транспортний засіб. Процес починається з авторизації клієнта у системі та перегляду наявних страхових пропозицій. Далі клієнт приймає рішення щодо вибору програми страхування та формує заявку, заповнюючи відповідну форму із зазначенням даних про транспортний засіб. Після надсилання заявки ініціатива переходить до страхового агента.

Агент отримує заявку, перевіряє достовірність наданої інформації та приймає рішення про її схвалення або відхилення. У разі позитивного результату оформлюється проект договору, який погоджується зі страхувальником. Після узгодження умов агент зберігає всі зміни у системі, що фіксує завершення процесу. Таким чином, діаграма демонструє повну послідовність дій, а також точки прийняття рішень з боку як клієнта, так і представника компанії.

Представлена діаграма дозволяє візуалізувати ключові етапи взаємодії між користувачами та системою, включаючи перевірку даних, погодження умов та завершення страхового процесу [19]. Вона є ефективним інструментом аналізу та проєктування логіки роботи додатку, допомагає виявити критичні точки та сформувати чітке уявлення про послідовність кроків, необхідних для реалізації страхового сценарію. Діаграма активності є основою для подальшого моделювання функціональних блоків та розробки інтерфейсу користувача.

1.4 Огляд інформаційних джерел та існуючих рішень

Перед початком розробки інформаційної системи доцільно здійснити комплексний аналіз сучасних технологій, прикладних рішень та інформаційних ресурсів, що використовуються в обраній галузі. Такий підхід дозволяє не лише зібрати актуальну інформацію про функціональність уже реалізованих систем, а й виявити ключові вимоги користувачів, типові архітектурні підходи, а також обмеження, які слід враховувати під час проєктування власного програмного продукту. Аналіз аналогів також дає змогу запозичити успішні практики та уникнути повторення недоліків попередніх реалізацій [21].

У межах даного проєкту розглянуто інформаційні системи, які надають послуги автострахування у цифровому форматі, зокрема ті, що дозволяють

клієнтам самостійно оформлювати поліси, отримувати консультації, проводити онлайн-платежі та контролювати статус страхових подій. Основною метою аналізу є визначення типових функціональних компонентів таких платформ, оцінка їхньої зручності, а також пошук можливостей для оптимізації інтерфейсу та логіки взаємодії.

Прикладом вітчизняного рішення є онлайн-платформа компанії PZU Україна (рис. 1.6), яка входить до складу однієї з найбільших страхових груп Європи [10].

Рис. 1.6 Інтерфейс PZU Україна

Сервіс PZU Україна надає широкий спектр послуг у сфері автострахування, зокрема оформлення полісів ОСЦПВ та КАСКО, із можливістю самостійного придбання договорів через вебсайт, використанням онлайн-калькуляторів для розрахунку вартості, отриманням дистанційних консультацій та здійсненням оплати через інтегровані платіжні системи. Аналіз цього рішення свідчить про ефективність підходу, що поєднує повний цикл оформлення страхового продукту, підтримку в режимі реального часу та максимальну автоматизацію процесів, що відповідає очікуванням сучасних

користувачів, орієнтованих на швидкість, прозорість і зручність без потреби особистого візиту до офісу страховика.

Прикладом сучасного вітчизняного рішення є ТАС Страхування – одна з провідних українських компаній, яка пропонує ОСЦПВ і КАСКО та активно впроваджує цифрові сервіси для дистанційної взаємодії з клієнтами [11].

Сервіс ТАС Страхування забезпечує повноцінну цифрову екосистему для управління автострахуванням, пропонуючи такі функціональні можливості, як онлайн-оформлення полісів, доступ до електронних документів через мобільний застосунок, інтерактивні калькулятори для розрахунку вартості страхування та подання заяв на відшкодування через онлайн-канали. Цільовою аудиторією є власники транспортних засобів, які віддають перевагу швидкому, мобільному та зручному доступу до страхових послуг без потреби фізичного відвідування офісу.

Завдяки орієнтації на онлайн-комунікацію система забезпечує високу ефективність обслуговування, а її інтерфейс (рис. 1.7) демонструє приклад логічно структурованого та зручного у використанні середовища, що може бути використане як зразок під час проєктування власного рішення.

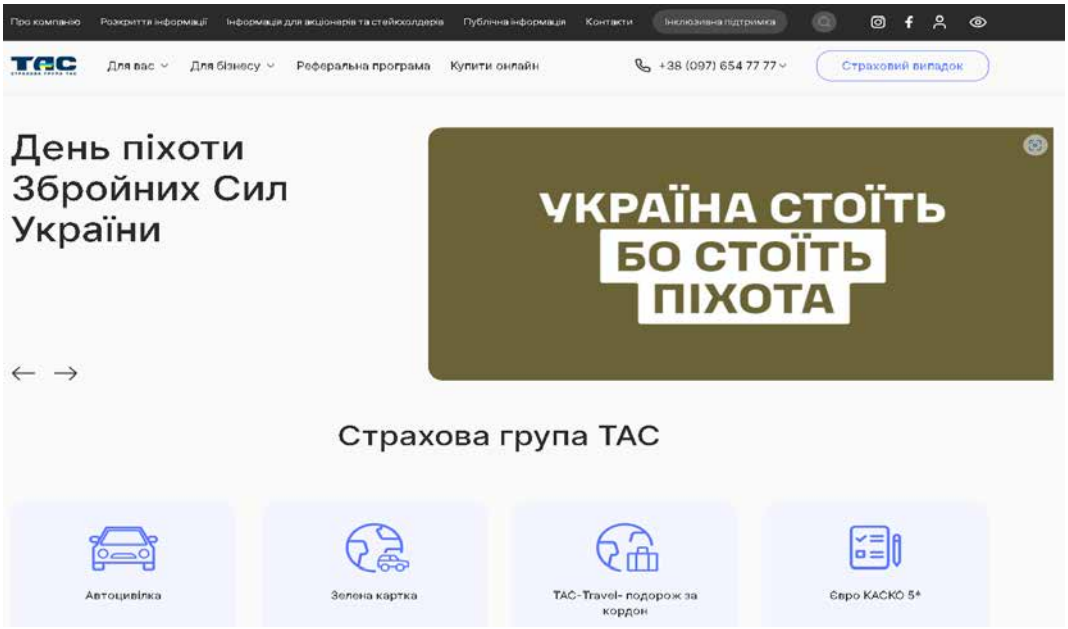

Рис. 1.7 Інтерфейс ТАС Страхування

Компанія УНІКА є представником міжнародної страхової групи UNIQA та має значний досвід роботи на європейському ринку [12]. В Україні вона спеціалізується на автострахуванні й страхуванні здоров'я, пропонуючи сучасні цифрові рішення для зручності користувачів.

Серед функціональних можливостей платформи – онлайн-придбання полісів, використання калькуляторів для розрахунку вартості страхування, оформлення договорів через мобільний застосунок, а також підтримка врегулювання страхових випадків за допомогою онлайн-сервісів. Цільовою аудиторією є автовласники та користувачі, які прагнуть швидкого й доступного способу отримання страхових послуг через інтернет. Інтерфейс системи наведено на рис. 1.8.

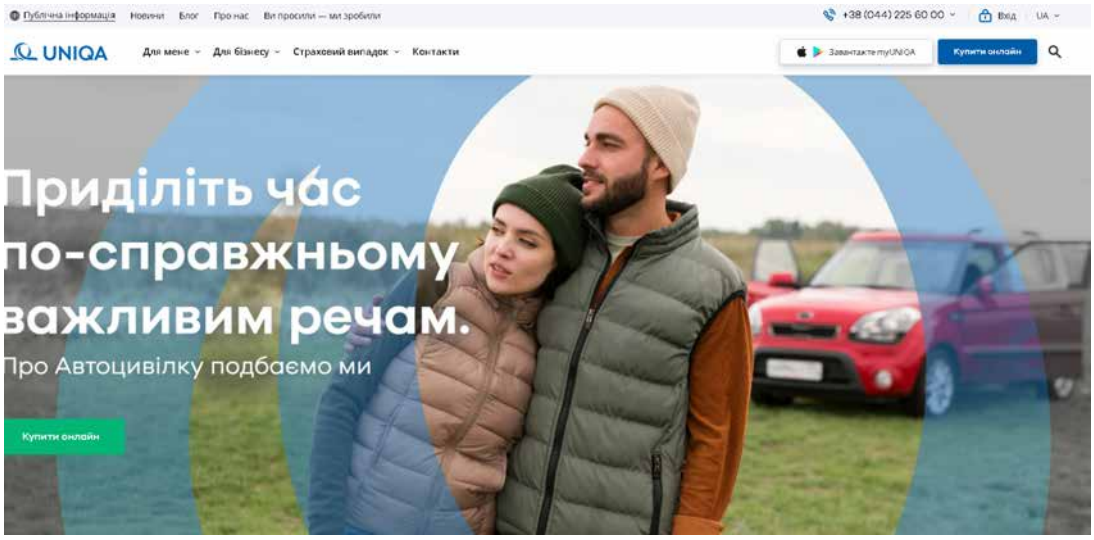

Рис. 1.8 Інтерфейс УНІКА

Після аналізу рішень, представлених на українському ринку автострахування, доцільно розглянути досвід провідних міжнародних компаній, які активно впроваджують цифрові технології для підвищення ефективності обслуговування клієнтів.

Geico – один із провідних постачальників автострахування у США, відомий широким використанням цифрових технологій для спрощення процесу оформлення полісів і конкурентними цінами [13]. Сервіс пропонує онлайн-калькулятори для миттєвого розрахунку вартості страхування, мобільний

застосунок для управління полісами та подачі заяв на відшкодування, функції порівняння різних варіантів страхування, а також інтерактивні інструменти для пошуку партнерських ремонтних сервісів. Geico орієнтований на американських автовласників, які цінують швидкість, доступність і функціональність онлайн-страхування. Інтерфейс платформи подано на рис. 1.9.

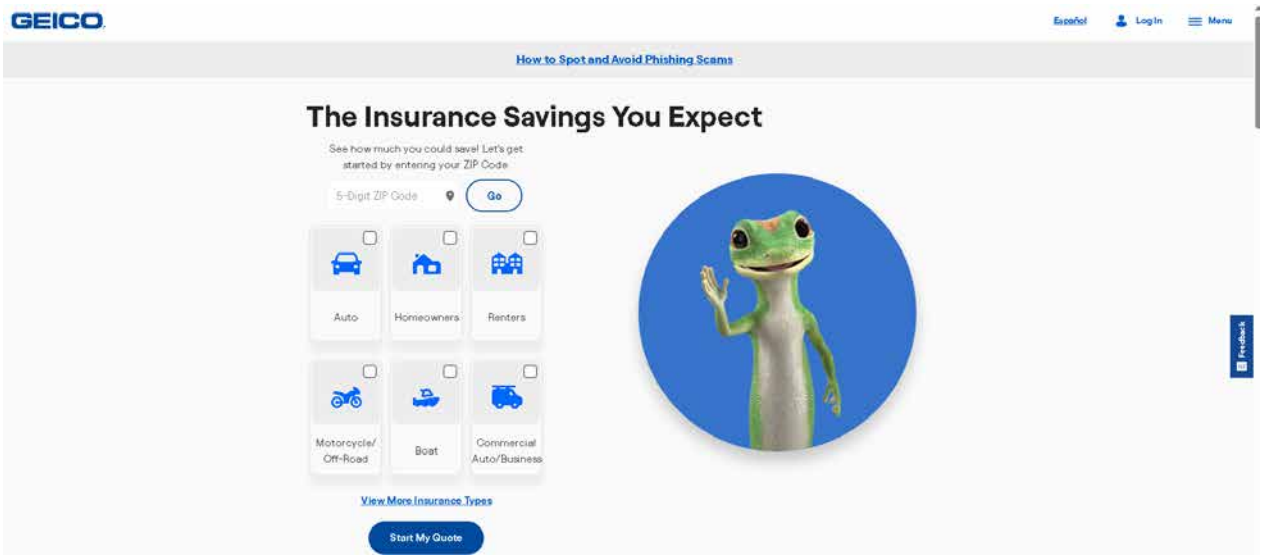

Рис. 1.9 Інтерфейс Geico

Allianz – одна з найбільших міжнародних страхових компаній, що надає широкий спектр послуг, включаючи автострахування, з орієнтацією на цифрову трансформацію взаємодії з клієнтами [14]. Платформа дозволяє оформлювати поліси онлайн, використовувати калькулятори для розрахунку вартості страхування, керувати полісами та подавати заявки на виплати через мобільні додатки, а також здійснювати врегулювання страхових випадків за допомогою онлайн-сервісів. Сервіс орієнтований на автовласників у країнах Європи та поза її межами, які очікують надійного міжнародного обслуговування з доступом до сучасних електронних інструментів. Інтерфейс системи наведено на рис. 1.10.

Для узагальнення результатів аналізу вітчизняних та міжнародних страхових платформ сформовано порівняльну таблицю (табл. 1.4), що відображає основні функціональні можливості систем. Це дає змогу виявити загальні тенденції в онлайн-сервісах автострахування та визначити доцільні функції для реалізації у проєктованому програмному забезпеченні.

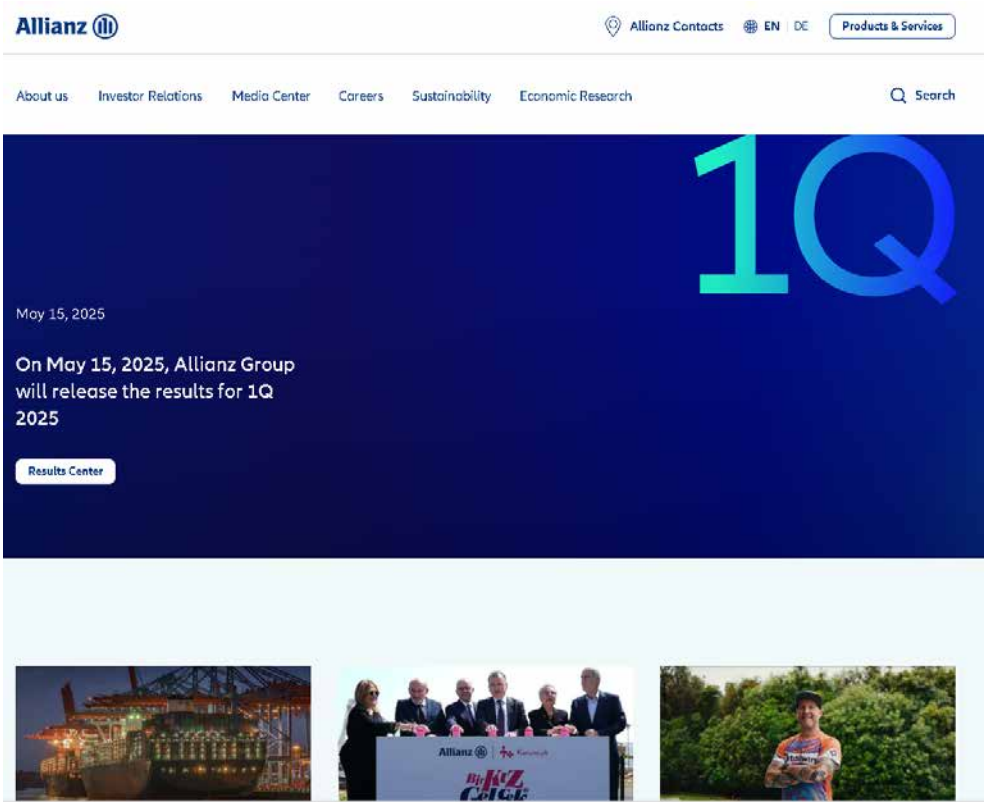

Рис. 1.10 Інтерфейс Allianz

Таблиця 1.4

Порівняльний аналіз функціональних можливостей страхових платформ

№	Компанія	Онлайн-оформлення	Мобільний додаток	Онлайн-калькулятор	Врегулювання онлайн	Підтримка клієнта	Географія
1	PZU Україна	✓	–	✓	✓	✓	Україна
2	ТАС Страхування	✓	✓	✓	✓	✓	Україна
3	УНІКА	✓	✓	✓	✓	✓	Україна / Європа
4	Geico (США)	✓	✓	✓	✓	✓	США
5	Allianz (ЄС)	✓	✓	✓	✓	✓	Європа / глобально

Узагальнюючи результати проведеного аналізу, можна зробити висновок, що успішні страхові платформи, незалежно від географії, орієнтуються на повну цифровізацію взаємодії з клієнтом: від миттєвого розрахунку вартості полісу до оформлення договору та врегулювання страхових випадків онлайн. Наявність мобільного додатку, простого інтерфейсу та багатофункціональної підтримки є

стандартом як для міжнародних, так і для провідних вітчизняних компаній. Ці фактори будуть враховані під час розробки власного програмного рішення, яке має поєднувати найкращі практики у сфері автострахування.

1.5 Постановка завдання

Процес розробки інформаційної системи розпочинається з чіткого формулювання завдання, яке визначає цілі, функціональні та технічні обмеження, а також очікуваний результат. У межах даного проєкту передбачається створення веб-додатку, що забезпечить автоматизацію ключових бізнес-процесів працівника автострахової компанії з метою підвищення ефективності обробки заявок, оформлення полісів, ведення клієнтської бази та супроводу страхових випадків.

Система має надавати працівнику зручний інтерфейс для взаємодії з клієнтськими зверненнями, перегляду та обробки поданих заявок, формування страхових договорів, редагування профілю, а також моніторингу страхових подій. Передбачається підтримка багатофункціонального доступу до системи з різних типів пристроїв, зокрема ПК та планшетів, що вимагає реалізації адаптивного веб-інтерфейсу.

З технічної точки зору система повинна бути реалізована як веб-додаток із застосуванням сучасних технологій розробки (HTML, CSS, PHP, JavaScript) і використовувати реляційну базу даних для зберігання структурованої інформації про клієнтів, транспортні засоби, поліси та страхові випадки. Система має забезпечувати базову авторизацію користувача, захист даних і стабільну роботу при типовому навантаженні.

Таким чином, завдання проєкту полягає в проєктуванні, реалізації та тестуванні інформаційної системи для працівника автострахової компанії, яка дозволить автоматизувати основні дії, зменшити вплив людського фактора, прискорити обробку звернень і підвищити якість надання страхових послуг у цифровому середовищі

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Логічне моделювання бази даних є ключовим етапом у проєктуванні інформаційної системи, що забезпечує структурну цілісність даних та правильну побудову взаємозв’язків між об’єктами предметної області. Одним із найефективніших підходів до цього завдання є побудова ER-моделі (Entity-Relationship), яка дозволяє візуалізувати сутності, їхні атрибути та логіку зв’язків між ними. Такий підхід сприяє уніфікованому уявленню про структуру даних і слугує основою для формування фізичної моделі бази даних.

У межах проєктування системи автострахування було визначено основні сутності предметної області: Client, Vehicle, InsurancePolicy, PolicyType, InsuranceClaim та Employer. ER-модель подана на рис. 2.1, де кожна сутність представлена у вигляді прямокутника з переліком атрибутів, а зв’язки між сутностями – у вигляді ліній із зазначенням типу зв’язку (один-до-одного, багато-до-одного). Наприклад, кожен транспортний засіб належить одному клієнту, але клієнт може володіти кількома транспортними засобами, що формує зв’язок типу N:1.

Для побудови ER-діаграми було використано CASE-засіб ERwin Data Modeler, який забезпечує візуалізацію логічної структури системи, дозволяє визначити атрибути, ключі, типи зв’язків і забезпечує підтримку нормалізації. Додатково можуть застосовуватись універсальні інструменти, як-от DrawIO, які дають змогу створювати діаграми у веббраузері, зберігати їх у зручному форматі та здійснювати спільне редагування.

На рисунку 2.1 подано ER-діаграму, яка відображає логічну структуру бази даних веб-додатку для автострахування. Вона ілюструє основні сутності предметної області, їх атрибути та взаємозв’язки, що забезпечують цілісне представлення інформаційної моделі. Побудована діаграма слугує

концептуальним підґрунтям для подальшої реалізації бази даних у реляційній формі та використовується для формування запитів, інтерфейсів і бізнес-логіки системи.

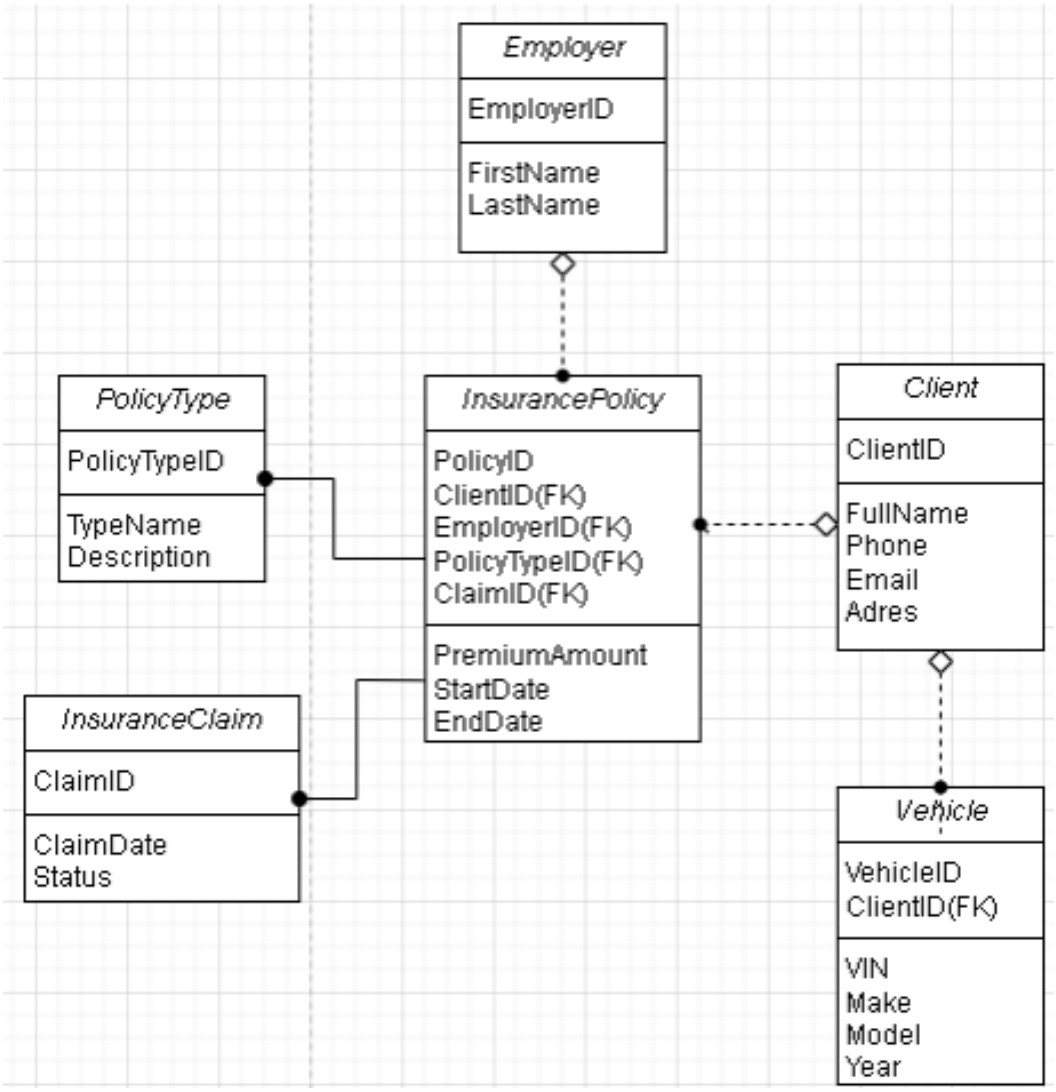

Рис. 2.1 ER-діаграма для веб-додатку із страхування автотранспорту

Побудована ER-діаграма дозволяє чітко структурувати дані, виявити ключові сутності та зв’язки між ними, що забезпечує логічну узгодженість та ефективність подальшої реалізації бази даних у межах веб-додатку для автострахування [24].

Таблиця 2.1 подає узагальнений опис сутностей, їхніх ключових атрибутів та логіки зв’язків. Кожна таблиця в базі даних містить унікальний первинний

ключ і, за потреби, зовнішні ключі для реалізації міжтабличних зв’язків [17]. Наприклад, у таблиці InsurancePolicy зберігається інформація про всі укладені страхові договори, а зв’язки з таблицями Client, Vehicle, Employer, PolicyType та InsuranceClaim забезпечуються відповідними зовнішніми ключами.

Таблиця 2.1

Опис основних сутностей логічної моделі

Сутность	Первиний ключ	Зовнішні ключі	Інші атрибути	Зв’язки з іншими сутностями
Employer	EmployerID	-	FirstName LastName	Багато-до-одного(N:1) з InsurancePolicy
PolicyType	PolicyTypeID	-	TypeName Description	Один-до-одного (1:1) з InsurancePolicy
InsuranceClaim	ClaimID	-	ClaimData Status	Один-до-одного (1:1) з InsurancePolicy
Client	ClientID	-	FullName Phone Email Adres	Багато-до-одного(N:1) з InsurancePolicy; Багато-до-одного(N:1) з Vehicle
Vehicle	VehicleID	ClientID(FK)	VIN Marke Model Year	Багато-до-одного(N:1) з Client
InsurancePolicy	PolicyID	ClientID(FK) VehicleID(FK) ClaimID(FK) PolicyTypeID(FK) EmployerID(FK)	PremiumAmount StartData EndData	Багато-до-одного(N:1) з Client; Багато-до-одного(N:1) з Employer; Один-до-одного (1:1) з PolicyType; Один-до-одного (1:1) з InsuranceClaim

З метою дотримання принципів ефективного зберігання даних і виключення надмірності, логічна модель була нормалізована до третьої нормальної форми (3НФ). Основні ознаки відповідності 3НФ:

– кожна таблиця містить унікальний первинний ключ, що забезпечує однозначну ідентифікацію записів;

- усі атрибути залежать виключно від первинного ключа, що виключає часткові та транзитивні залежності (відповідність 2НФ і 3НФ);
- інформація поділена на структурно незалежні таблиці, що мінімізує дублювання й забезпечує логічну цілісність;
- реалізовані зовнішні ключі гарантують узгодженість зв’язків між таблицями та запобігають втраті пов’язаних даних.

Таким чином, розроблена логічна модель даних відображає реальні процеси взаємодії у межах предметної області автострахування, забезпечуючи масштабовану, оптимізовану та технічно стійку основу для реалізації функціональної бази даних у веб-додатку. Вона формує концептуальну платформу для подальшого проєктування фізичної моделі та реалізації механізмів взаємодії з інтерфейсом користувача.

2.2 Діаграма класів та кооперацій

Діаграма класів є важливим інструментом для моделювання об’єктно-орієнтованої структури програмної системи. Вона дозволяє візуалізувати класи, їх атрибути, методи та логіку взаємозв’язків між ними, що сприяє глибшому розумінню логіки предметної області та структури майбутнього застосунку. У мові UML (Unified Modeling Language) діаграма класів використовується для опису статичної структури системи, демонструючи, як її основні компоненти пов’язані між собою на логічному рівні [22].

Діаграми класів застосовуються не лише на етапі розробки, а й для системного аналізу та документації програмного забезпечення [5]. Основні напрями використання діаграм класів узагальнено в таблиці 2.2. Як видно, вони слугують засобом моделювання, проєктування й аналізу, а також допомагають структурувати функціональність системи на модулі, які легко підтримувати й масштабувати.

Таблиця 2.2

Основні застосування діаграми класів

№	Ціль	Опис
1	Моделювання системи	Діаграми класів допомагають розібрати складну систему на окремі компоненти та визначити взаємозв’язки між ними.
2	Проектування системи	Діаграми класів використовуються для проектування об’єктно-орієнтованих систем. Вони допомагають визначити необхідні класи, їх атрибути та методи перед початком фази реалізації.
3	Аналіз системи	Діаграми класів допомагають зрозуміти, як система функціонує, та виявити можливі проблеми або несоответствия в структурі системи.

Структура діаграми класів базується на чотирьох ключових елементах, поданих у таблиці 2.3. До них належать назви класів, атрибути, методи та типи зв’язків між класами (асоціація, агрегація, композиція, наслідування).

Таблиця 2.3

Основні елементи діаграми класів

№	Елемент	Опис
1	Назви класів	Кожен клас на діаграмі класу має назву, яка ідентифікує його. Назви класів зазвичай вказують на їх роль або суть в системі.
2	Атрибути класу	Атрибути класу представляють дані, які зберігаються в об’єктах класу. Вони можуть бути представлені у вигляді змінних або властивостей.
3	Методи класу	Методи класу визначають поведінку об’єктів класу. Вони представляють операції, які можуть бути виконані над об’єктами цього класу.
4	Зв’язки між класами	Зв’язки між класами показують взаємозв’язки між об’єктами класів. Це можуть бути зв’язки типу “агрегація”, “композиція”, “унаслідування” або “асоціація”.

Додатково діаграми класів мають низку переваг, які роблять їх незамінними під час архітектурного проєктування. Основні з них систематизовано в таблиці 2.4. Зокрема, вони сприяють візуалізації структури

системи, виявленню проблем на ранніх етапах, полегшенню командної роботи та забезпеченню модульності й розширюваності програмного продукту.

Таблиця 2.4

Переваги використання діаграм класів

№	Перевага	Опис
1	Візуалізація структури	Діаграми класів допомагають візуалізувати структуру системи, зробити її більш зрозумілою та легко сприйнятливою.
2	Виявлення проблем	Діаграми класів дозволяють виявити можливі проблеми або недоліки в структурі системи на ранніх етапах розробки.
3	Підвищення продуктивності	Використання діаграм класів дозволяє покращити комунікацію між розробниками та забезпечити однозначне розуміння структури системи.
4	Модульність та розширюваність	Діаграми класів допомагають визначити модулі та їх взаємозв’язки, що сприяє покращенню модульності та розширюваності системи.

У рамках даного проєкту було розроблено діаграму класів для веб-додатку з автострахування. Вона включає такі класи, як «Страховий агент», «Страхувальник», «Страхова заявка» та «Категорія страхового випадку», які відображають ключові ролі та об’єкти предметної області. Загальну структуру класів подано на рисунку 2.2, а детальний опис – у таблиці 2.5.

Клас «Страховий агент» представляє працівника компанії, який виконує адміністративну обробку заявки. Його методи охоплюють повний цикл взаємодії з документами. Клас «Страхувальник» моделює клієнта, який ініціює звернення. Клас «Страхова заявка» описує сутність звернення із ключовими атрибутами, такими як статус та сума. Клас «Категорія страхового випадку» систематизує типові ситуації, що є підставою для подання заявок.

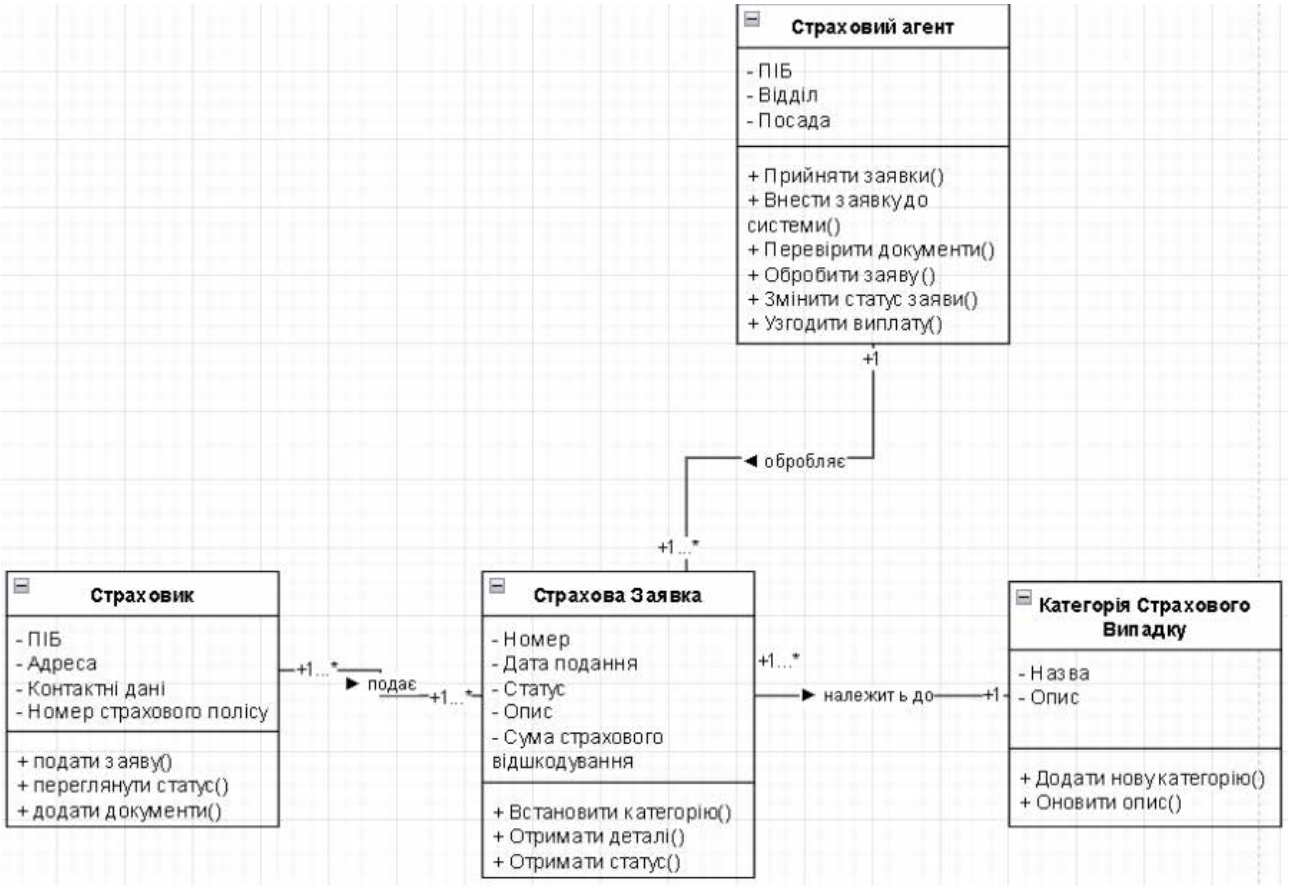

Рис. 2.2 Діаграма класів для веб-додатку із страхування автотранспорту

Таблиця 2.5

Структура діаграми класів

№	Клас	Властивості	Методи
1	Страховий агент	Страховий агент	Прийняти заявку Внести заявку до системи Перевірити документи Обробити заявку Змінити статус заявки Узгодити виплату
2	Страховик	Страховик	Подати заявку Переглянути статус Додати документи
3	Страхова заявка	Страхова заявка	Встановити категорію Отримати деталі Отримати статус
4	Категорія страхового випадку	Категорія страхового випадку	Додати нову категорію Оновити опис

Розроблена діаграма класів дозволяє чітко структурувати логіку предметної області, формалізувати поведінку об’єктів і визначити зв’язки між ними. Це створює надійну основу для реалізації об’єктно-орієнтованої архітектури програмної системи, підвищуючи її гнучкість, масштабованість і підтримуваність.

Для глибшого розуміння динаміки функціонування системи в межах окремих сценаріїв доцільно використовувати діаграми кооперації (collaboration diagrams). Вони демонструють, як окремі об’єкти взаємодіють між собою для досягнення певного результату в рамках конкретної функціональної задачі. На відміну від діаграм класів, що фіксують загальну структуру системи, діаграми кооперації зосереджуються на прикладному аспекті: хто, з ким і як саме взаємодіє у межах обраного сценарію.

Діаграма кооперації відображає екземпляри класів (об’єкти), між якими існують зв’язки – реалізація асоціацій у контексті конкретної ситуації. Повідомлення між об’єктами позначаються пронумерованими стрілками, що вказують напрям і послідовність взаємодії. Такий підхід дозволяє не лише візуалізувати логіку виконання сценарію, а й виявити можливі надмірності або пропуски у функціональній поведінці системи. Важливо, що на діаграму виносяться лише ті об’єкти, які безпосередньо беруть участь у поточній взаємодії.

На рисунку 2.3 представлено діаграму кооперації, яка моделює сценарій подання заяви страхувальником. Вона ілюструє взаємодію об'єктів «Страхувальник», «Форма подання», «Система» та «База даних» у процесі створення нової заявки. Зокрема, користувач ініціює подання, заповнює відповідну форму, яка передає дані до системи, де відбувається їх перевірка та збереження.

Рис. 2.3 Сценарій подання заяви страхувальником

На наступній діаграмі (рис. 2.4) показано сценарій прийому заявки страховим агентом. У цьому випадку об’єкти «Агент», «Система», «Список заяв» та «База даних» взаємодіють у межах процесу аналізу поданих звернень. Агент надсилає запит, система обробляє його, витягує відповідні дані з бази, після чого заявка стає доступною для перегляду.

Рис. 2.4 Сценарій прийому заяви страховим агентом

Останній приклад (рис. 2.5) демонструє сценарій обробки заявки, де беруть участь об’єкти «Агент», «Система», «Форма редагування» та «База даних». Діаграма деталізує етапи оновлення статусу заявки: перевірка, редагування, збереження результатів та підтвердження внесених змін. Взаємодія об’єктів відбувається послідовно, з дотриманням правил доступу й логіки обробки даних.

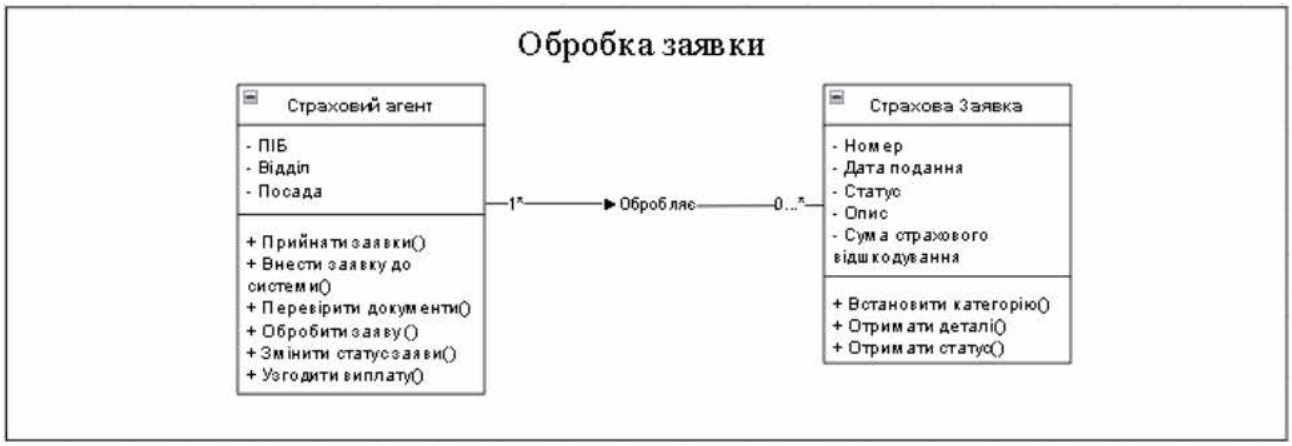

Рис. 2.5 Сценарій обробки заявки

Діаграми кооперації забезпечують наочне уявлення про сценарні взаємодії між об’єктами системи та дозволяють чітко простежити послідовність і напрям дій у межах кожного окремого процесу. Завдяки цьому можливо виявити неузгодженості ще на етапі проєктування, що підвищує надійність реалізації бізнес-логіки системи та сприяє створенню прозорої архітектури взаємодії. У подальших етапах розробки такі діаграми можуть слугувати основою для створення модулів програмного забезпечення, тестових сценаріїв і документації.

2.3 Діаграма пакетів

Одним із важливих засобів моделювання високорівневої структури програмного забезпечення є діаграма пакетів. Вона слугує для візуалізації логічної організації системи у вигляді окремих модулів (пакетів) та демонструє залежності між ними. Пакет у цьому контексті – це умовна група класів, інтерфейсів або інших програмних компонентів, які виконують схожі функції або належать до одного логічного підсистемного рівня. Така структуризація дозволяє краще контролювати архітектуру великої системи, підвищувати її модульність і спрощувати супровід.

На діаграмі пакетів кожен модуль подається у вигляді прямокутника, який може включати підпакети або окремі логічні елементи. Взаємозв’язки між пакетами позначаються пунктирними стрілками, які вказують на залежність:

наприклад, якщо один пакет використовує класи або функції з іншого. Це дозволяє аналітикам і розробникам зрозуміти, як окремі частини системи взаємодіють між собою та наскільки ці зв’язки є критичними для забезпечення стабільності й гнучкості архітектури.

На рисунку 2.6 представлено діаграму пакетів, яка відображає архітектурну модель програмного забезпечення веб-додатку для автострахування.

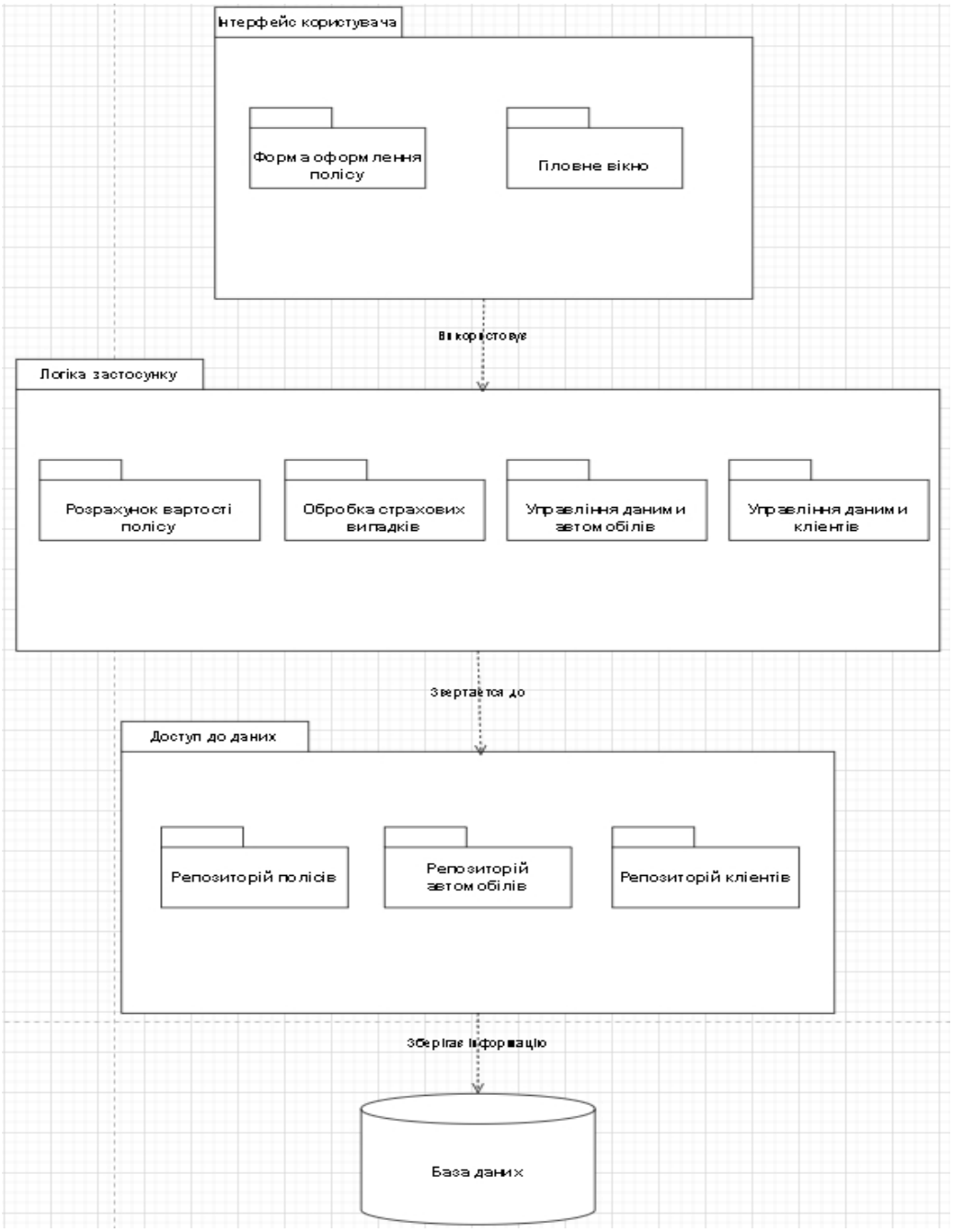

Рис. 2.6 Діаграма пакетів для веб-додатку із страхування автотранспорту

Система структурована на чотири основні рівні: презентаційний, рівень логіки застосунку, рівень доступу до даних та рівень фізичного зберігання. Така багаторівнева структура дозволяє ефективно розділити відповідальність між компонентами та забезпечити гнучке масштабування системи.

Архітектура системи побудована на основі чотирьох логічних рівнів. Презентаційний рівень відповідає за користувацький інтерфейс, включаючи головне вікно та форму оформлення полісу, яка дозволяє вводити дані про клієнта і транспортний засіб. Рівень логіки застосунку реалізує бізнес-функції системи, зокрема розрахунок вартості полісу, обробку страхових випадків, а також управління інформацією про автомобілі та клієнтів із можливістю її перегляду й редагування.

Рівень доступу до даних виступає посередником між логікою застосунку та фізичним зберіганням, реалізуючи репозиторії для основних сутностей – полісів, автомобілів і клієнтів. Заключний, фізичний рівень, представлений базою даних, яка слугує централізованим сховищем інформації та побудована з урахуванням вимог нормалізації, що забезпечує узгодженість і цілісність даних у межах системи.

Діаграма пакетів дозволяє наочно відобразити загальну архітектурну модель системи, розподіл функцій між логічними рівнями та типи залежностей між окремими модулями. Така структура забезпечує модульність, гнучкість, спрощує тестування та подальший розвиток проєкту. Використання багаторівневого підходу дозволяє забезпечити чіткий розподіл обов’язків між компонентами й адаптацію системи до змін функціональних або технологічних вимог у майбутньому.

2.4 Діаграма компонентів

Проєктування архітектури сучасного програмного забезпечення неможливе без чіткого уявлення про його складові частини та механізми взаємодії між ними. Для цього доцільно використовувати діаграму компонентів (component diagram), яка належить до структурних діаграм мови UML і

відображає взаємозв’язки між логічно відокремленими частинами системи. Такий підхід дозволяє побачити не лише те, з яких модулів складається програмний продукт, а й те, як реалізовано зв’язки та залежності між ними на рівні коду та інтерфейсів.

Кожен компонент у системі виконує окрему функцію та взаємодіє з іншими через чітко визначені інтерфейси. Це дає змогу забезпечити модульність, повторне використання компонентів та спрощення обслуговування. Завдяки цьому діаграма компонентів є корисним інструментом для архітекторів, аналітиків та розробників, особливо при роботі над масштабованими або багаторівневими системами. Залежності між компонентами візуалізуються у вигляді стрілок, що ілюструють напрямок та характер взаємодії, наприклад, коли один модуль використовує функції іншого.

На рисунку 2.7 представлено діаграму компонентів, яка відображає архітектуру веб-додатку для обробки страхових звернень у сфері автострахування. Система розділена на три основні шари: презентаційний рівень (Frontend), рівень бізнес-логіки (Backend) та рівень доступу до даних (Database). Такий поділ дозволяє розмежувати обов’язки між модулями та полегшує масштабування й супровід системи.

Презентаційний рівень відповідає за взаємодію з користувачем і включає компоненти, що реалізують зовнішній вигляд інтерфейсу. Зокрема, main-employeer.php та main-client.php є основними сторінками, через які працівники й клієнти взаємодіють із системою. Ці сторінки підключають відповідні CSS-файли (styles-employeer.css, styles-client.css), що відповідають за візуальне оформлення. Кожна дія користувача передається у вигляді запиту до центрального обробника index.php.

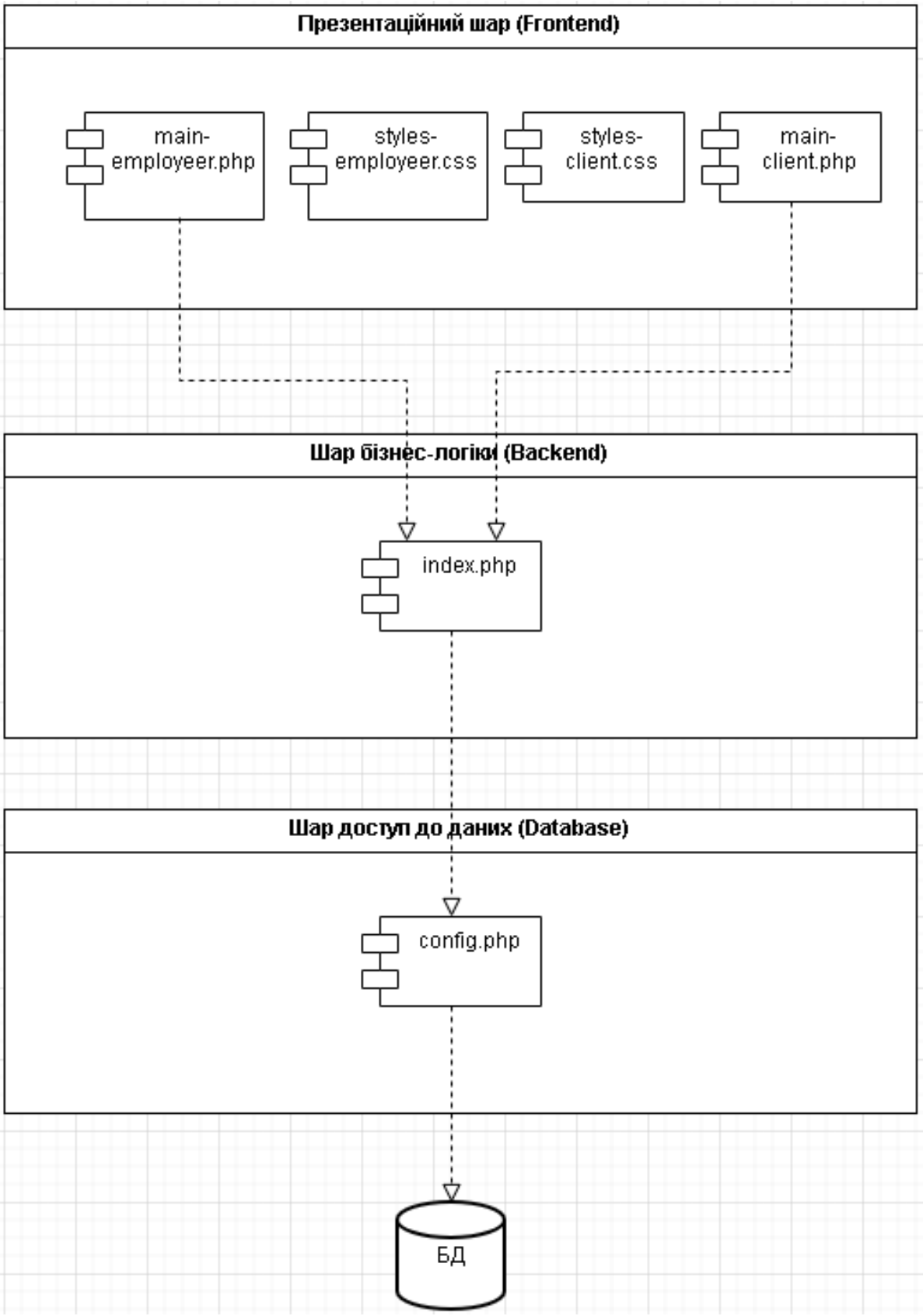

Рис. 2.7 Діаграма компонентів веб-додатку із страхування автотранспорту

Рівень бізнес-логіки, або серверна частина, виконує обробку запитів, застосовує бізнес-правила, приймає рішення на основі введених даних і формує запити до бази даних. Основним компонентом цього рівня є index.php, який реалізує функціональність для обох ролей – клієнта та працівника. Він залежить від config.php, що надає доступ до функцій підключення до бази даних та виконання SQL-запитів.

Рівень доступу до даних включає модуль config.php, що містить конфігураційні параметри для підключення до бази даних, а також допоміжні функції для її взаємодії з серверною логікою. Центральною складовою цього рівня є база даних, у якій зберігається інформація про користувачів, заявки, договори, транспортні засоби та інші сутності системи. Саме вона забезпечує збереження та цілісність інформації в усій системі.

Узагальнено потік даних у системі виглядає наступним чином: користувач ініціює взаємодію через клієнтський інтерфейс (наприклад, сторінку main-client.php), що надсилає запит до index.php. Цей компонент обробляє отриману інформацію, при потребі звертається до config.php для отримання або зміни даних у базі, після чого результат повертається у зворотному напрямку до користувача – у вигляді повідомлення, оновленої сторінки або підтвердження виконаної дії.

Діаграма компонентів дозволяє наочно уявити модульну структуру програмного забезпечення, визначити місце кожного з компонентів у загальній архітектурі та зрозуміти механізми їх взаємодії. Поділ на логічні шари забезпечує зрозуміле розмежування функцій, підвищує гнучкість системи та спрощує її тестування, масштабування й підтримку. Такий підхід є доцільним для реалізації веб-застосунків у сфері автострахування, де важлива надійність, інтегрованість і зручність у використанні.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

У сучасних інформаційних системах надійне зберігання, обробка та доступ до даних забезпечуються за допомогою спеціалізованого програмного забезпечення – системи управління базами даних (СУБД). Це програмний комплекс, що дозволяє створювати логічно структуровані сховища інформації, виконувати операції з даними, а також реалізовувати механізми контролю доступу, резервного копіювання та захисту. Однією з ключових функцій СУБД є організація ефективної та захищеної роботи з інформацією, що включає її збереження, пошук, оновлення та видалення.

Крім зручного управління даними, СУБД забезпечує їхню цілісність та захист. Зокрема, підтримуються механізми аутентифікації та авторизації користувачів, шифрування конфіденційної інформації, ведення журналів дій та системний аудит. Існують різні класифікації СУБД: за моделлю даних (реляційні, ієрархічні, мережеві), за типом запитів (SQL, NoSQL), за ліцензією (відкриті або комерційні) та за способом розміщення (централізовані, розподілені). Вибір СУБД залежить від особливостей застосунку, його масштабів і вимог до продуктивності.

У межах цього проєкту для реалізації серверної частини бази даних використано реляційну СУБД phpMyAdmin. Це потужний веб-інструмент з відкритим кодом, який дозволяє адмініструвати бази даних MySQL і MariaDB через графічний інтерфейс. На відміну від командного рядка, де всі дії виконуються вручну, phpMyAdmin забезпечує зручне керування структурою бази, таблицями, записами та зв’язками без потреби в глибокому знанні мови SQL [2, 15, 16]. Це робить інструмент доступним навіть для початківців, а також значно підвищує швидкість адміністрування.

Переваги phpMyAdmin як засобу управління базами даних наведено в таблиці 3.1. У ній висвітлено, чому цей інструмент є поширеним вибором у більшості веб-проєктів.

Таблиця 3.1

Основні переваги використання phpMyAdmin в проектах

№	Перевага	Опис
1	Зручний веб-інтерфейс	Всі основні операції виконуються через браузер без необхідності написання SQL-коду.
2	Розширений функціонал	Підтримка експорту, імпорту, створення копій, перегляду журналів, складних запитів.
3	Мультикористувацький доступ	Можливість налаштування ролей, обмеження доступу та захисту даних.
4	Відкритий код і безкоштовність	Регулярні оновлення, активна спільнота та сумісність із новими версіями СУБД.
5	Платформна сумісність	Працює на будь-якому сервері з підтримкою PHP, підтримує віддалене адміністрування.

Як показано в таблиці, phpMyAdmin є оптимальним інструментом для реалізації невеликих та середніх веб-систем, оскільки поєднує функціональність із простотою налаштування. Особливо це актуально для веб-додатків, таких як створений у цьому проєкті інструмент для автоматизації роботи працівника автострахової компанії.

Для уточнення специфіки використання phpMyAdmin саме у сфері веб-розробки, у таблиці 3.2 наведено узагальнення переваг інструменту з погляду його інтеграції в типову архітектуру веб-додатків.

Таблиця 3.2

Переваги використання phpMyAdmin у веб-розробці

№	Перевага	Опис
1	2	3
1	Простота адміністрування	Інтерфейс дозволяє керувати таблицями, записами, структурами без знання SQL.
2	Гнучке резервне копіювання	Підтримка швидкого імпорту й експорту у форматах CSV, SQL, XML, JSON.
3	Безпечна робота з правами доступу	Детальне налаштування доступу для різних ролей у командній роботі.

Таблиця 3.2 (продовження)

1	2	3
4	Віддалене управління	Можливість повноцінного адміністрування з будь-якого пристрою через веб-браузер.
5	Підтримка актуальних СУБД	Повна сумісність з MySQL та MariaDB, швидке оновлення до нових версій.
6	Легка інтеграція у PHP-середовище	Підтримка роботи на більшості веб-серверів, включаючи Apache, XAMPP, Laragon тощо.

Система управління базою даних є критичним елементом архітектури інформаційної системи. Використання phpMyAdmin у межах даного проєкту виправдане як з позиції ефективності, так і з позиції практичності впровадження у веб-середовищі. Гнучкість, доступність і підтримка широкого спектру функцій роблять цю СУБД оптимальним вибором для розробки на PHP, забезпечуючи надійну роботу з даними, їх захист і зручне адміністрування.

3.2 Розробка інформаційної бази

Процес проєктування та реалізації інформаційної бази є важливою частиною створення програмного забезпечення, оскільки саме база даних забезпечує централізоване зберігання, обробку та доступ до даних системи. У межах даного підрозділу демонструється приклад розробки бази даних на основі однієї з ключових сутностей – Users, що відповідає за облік зареєстрованих користувачів системи. Повний опис решти сутностей та їх структури подано в Додатку А, оскільки реалізація охоплює всі основні об’єкти предметної області.

На початковому етапі створено базу даних з ідентифікатором carguard, що відповідає назві розробленого веб-застосунку. Ця база є логічним контейнером, у межах якого відбувається створення всіх таблиць, визначення зв’язків, індексів, обмежень цілісності та інші операції. Створення бази проілюстровано на рисунку 3.1.

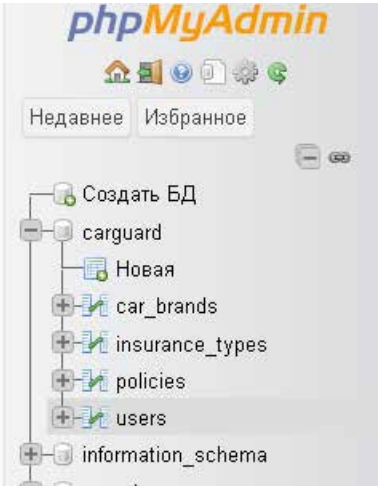

Рис. 3.1 Створена база даних “carguard”

Після створення структури БД було реалізовано підключення до неї з боку програмного коду за допомогою конфігураційного файлу. На рисунку 3.2 представлено фрагмент коду, що встановлює з’єднання з базою даних через стандартні механізми PHP (зокрема, mysqli_connect) із використанням параметрів підключення, таких як хост, логін, пароль та назва бази.

```php
<?php
$serverName = "localhost\\SQLEXPRESS";
$connectionOptions = array(
    "Database" => "CarGuardDB",
    "Uid" => "sa",
    "PWD" => "sa",
    "CharacterSet" => "UTF-8"
);

// Підключення
$conn = sqlsrv_connect(serverName: $serverName, connectionInfo: $connectionOptions);

// Перевірка
if (!$conn) {
    die(print_r(value: sqlsrv_errors(), return: true));
} else {
    echo "Підключено до SQL Server!";
}
?>
```

Рис. 3.2 Підключення до бази даних у програмному коді

Наступним кроком стало створення таблиці **users**, яка виконує роль збереження облікових даних користувачів. Таблиця містить поля для

ідентифікатора, ПІБ, email-адреси, пароля та ролі (страховик або працівник). Структуру цієї таблиці представлено на рисунку 3.3.

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно	Действие
1	**id**	int(10)			Нет	*Нет*		AUTO_INCREMENT	Изменить Удалить Ещё
2	**name**	varchar(255)	utf8mb4_general_ci		Нет	*Нет*			Изменить Удалить Ещё
3	**email**	varchar(255)	utf8mb4_general_ci		Нет	*Нет*			Изменить Удалить Ещё
4	**password**	varchar(255)	utf8mb4_general_ci		Нет	*Нет*			Изменить Удалить Ещё
5	**role**	enum('user', 'employeer')	utf8mb4_general_ci		Нет	*Нет*			Изменить Удалить Ещё
6	**verification_token**	varchar(255)	utf8mb4_general_ci		Нет	*Нет*			Изменить Удалить Ещё
7	**is_verified**	tinyint(1)			Да	*NULL*			Изменить Удалить Ещё

Рис. 3.3 Створена таблиця “users” у базі даних

Процес взаємодії з таблицею реалізовано через відповідну форму реєстрації, що дозволяє новому користувачу створити обліковий запис. На рисунку 3.4 зображено механізм, згідно з яким спочатку відбувається перевірка, чи існує вже користувач із вказаною електронною адресою. Якщо користувач не знайдений у базі, дані, введені у формі, записуються до таблиці **users** для подальшого використання у системі.

```php
session_start();
require_once 'config.php';
require_once '../public/send_mail.php';

if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['register'])) {
    $name = trim(string: $_POST['name']);
    $email = trim(string: $_POST['email']);
    $password = trim(string: $_POST['password']);
    $role = trim(string: $_POST['role']);

    // Перевірка, чи email вже існує
    $stmt = $conn->prepare("SELECT id FROM users WHERE email = ?");
    $stmt->bind_param("s", $email);
    $stmt->execute();

    if ($stmt->get_result()->num_rows > 0) {
        $_SESSION['register_error'] = "Цей email вже зареєстрований";
        header(header: "Location: reg-form-client.php");
        exit();
    }

    // Генерація токену
    $verificationCode = bin2hex(string: random_bytes(length: 32));
    $hashedPassword = password_hash(password: $password, algo: PASSWORD_DEFAULT);

    // Збереження користувача
    $stmt = $conn->prepare("INSERT INTO users (name, email, password, role, verification_token) VALUES (?, ?, ?, ?, ?)");
    $stmt->bind_param("sssss", $name, $email, $hashedPassword, $role, $verificationCode);

    if ($stmt->execute()) {
        if (sendVerificationEmail(email: $email, name: $name, verificationCode: $verificationCode)) {
            $_SESSION['success'] = "Реєстрація успішна! Перевірте вашу пошту для підтвердження.";
        } else {
            $_SESSION['register_error'] = "Помилка відправки листа підтвердження";
        }
    } else {
        $_SESSION['register_error'] = "Помилка при реєстрації";
    }

    header(header: "Location: reg-form-client.php");
    exit();
}
```

Рис. 3.4 Реєстрація та збереження нового користувача в базу даних

Результат успішної реєстрації, а саме – збережені дані нового користувача, можна переглянути безпосередньо в таблиці бази. Цей стан системи відображено на рисунку 3.5.

	id	name	email	password	role	verification_token	is_verified
Изменить Копировать Удалить	6	TuMaTuck	pershotravneva112@gmail.com	$2y$10$ZcXS8aLk9odk.z54Esq24eN2zBBA4RGBFmpoT.Q48ZT...	user	86979eadda4f3725e251af0898af0948a3ff8e1b65b2e1e38c...	NULL
Изменить Копировать Удалить	8	TuMaTuck	ipz21-d.melnykov@nubip.edu.ua	$2y$10$4OBRcFEhNbnDe0xFUk9y4eNMLaEEgAcUjLuHTPWbx1d...		c264c0319620a9f298bc7c0f3f892646cf3aa45eae64c8e0b5...	NULL
Изменить Копировать Удалить	9	TuMaTuck	melnykovdmytro@gmail.com	$2y$10$io8/Y7BS0wtHp3MIGJZjA.eTahvHIZElp1r3ifMXiO...	user	827acd41f9f1b555b8b31a1f49cb958175e0c84e79538c7c83...	NULL

Рис. 3.5 Дані які були записані в таблицю “users”

Також реалізовано механізм авторизації. На рисунку 3.6 показано логіку входу користувача: після введення електронної пошти та пароля система перевіряє правильність введених облікових даних. У разі успішної верифікації користувач автоматично перенаправляється до відповідного інтерфейсу, що залежить від його ролі, вказаної під час реєстрації. Зокрема, це може бути сторінка працівника або сторінка страхувальника.

```php
if (isset($_POST['login'])) {
    $email = $_POST['email'];
    $password = $_POST['password'];

    $result = $conn->query("SELECT * FROM users WHERE email = '$email'");
    if ($result->num_rows > 0) {
        $user = $result->fetch_assoc();
        if (password_verify(password: $password, hash: $user['password'])) {
            $_SESSION['name'] = $user['name'];
            $_SESSION['email'] = $user['email'];
            $_SESSION['user_id'] = $user['id'];

            if ($user['role'] === 'user') {
                header(header: "Location: users/mein-client.php");
            } else {
                header(header: "Location: employeer/mein-employeer.php");
            }
            exit();
        }
    }

    $_SESSION['login_error'] = 'Incorrect email or password';
    $_SESSION['active_from'] = 'login';
    header(header: "Location: index.php");
    exit();
}
```

Рис. 3.6 Реалізація входу (логіна) користувача у програмному коді

Реалізований підхід із розподілом ролей дає змогу продемонструвати логіку керування доступом у системі: вже на етапі створення облікового запису користувач обирає свою роль, що автоматично визначає інтерфейс та набір доступних функцій. Це сприяє не лише логічній ізоляції функціоналу, а й демонструє реальне застосування моделі контрольованого доступу в інформаційних системах.

Проєктування інформаційної бази починається зі створення структури даних і сутностей, таких як Users, які відіграють ключову роль у роботі застосунку. Реалізація таблиці, форми реєстрації та логіки авторизації демонструє практичне впровадження взаємодії між інтерфейсом користувача, серверною частиною та базою даних. Такий підхід забезпечує не лише зручність обслуговування користувачів, а й дозволяє легко масштабувати систему для подальших розширень функціональності.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Успішна розробка прикладного програмного забезпечення залежить не лише від чіткого планування, а й від грамотного вибору технологічного інструментарію, який відповідає поставленим вимогам до функціональності, масштабованості, надійності та зручності у розробці. Саме тому під час створення веб-додатку для працівника страхової компанії було обрано набір перевірених інструментів, що забезпечують стабільну роботу системи та комфортну взаємодію з її компонентами на всіх рівнях – від написання коду до адміністрування бази даних.

Як основне середовище для програмування було обрано Visual Studio Code (VS Code) [3, 18]. Це сучасний редактор коду, який підтримує численні мови програмування, включаючи PHP, HTML, CSS та JavaScript. Його перевагами є швидкодія, мінімалістичний інтерфейс, система розширень, вбудовані засоби діагностики помилок і автодоповнення. Завдяки плагінам, таким як PHP

Intelephense та Live Server, вдалося створити гнучке середовище, адаптоване для потреб веб-розробки [26].

Для реалізації локального серверного середовища було застосовано XAMPP – універсальний дистрибутив, що включає у себе веб-сервер Apache [16], мову програмування PHP та систему управління базами даних MySQL. Цей пакет дозволив швидко розгорнути повноцінне тестове середовище, відтворюючи роботу застосунку так, як вона функціонуватиме у продакшн-режимі. Відсутність складних налаштувань та підтримка стандартних компонентів зробили XAMPP оптимальним рішенням для прототипування.

Для управління базою даних використовувався веб-інтерфейс phpMyAdmin, який працює поверх MySQL. Він надає інтуїтивно зрозумілий інструментарій для створення таблиць, зв’язків між ними, виконання SQL-запитів та обслуговування структури бази даних. Візуальний інтерфейс значно полегшує адміністративні завдання, дозволяючи розробнику зосередитись на логіці взаємодії з даними, а не на синтаксисі запитів.

Для зручності порівняння характеристик обраних інструментів, у таблиці 3.3 наведено узагальнений огляд їх основних функціональних можливостей і ролі у структурі розробки.

Таблиця 3.3

Вибір технологічного інструментарію для створення веб-додатку

№	Інструмент	Призначення	Основні переваги
1	Visual Studio Code	Написання, редагування та відлагодження коду	Автодоповнення, підтримка PHP, розширення, інтеграція з Git
2	XAMPP	Локальний веб-сервер (Apache + PHP + MySQL)	Простота встановлення, симуляція продакшн-середовища, підтримка модулів
3	phpMyAdmin	Веб-інтерфейс для управління базами даних MySQL/MariaDB	Візуальна побудова таблиць, SQL-запити, резервне копіювання, імпорт/експорт

Крім цього, важливим критерієм вибору стали відкритість коду та безкоштовна доступність інструментів, що дозволяє уникнути додаткових витрат і забезпечити гнучкість для подальшого розширення системи. Усі інструменти легко інтегруються між собою, що дає змогу забезпечити безперебійну роботу на всіх етапах розробки – від дизайну до розгортання і тестування [25].

Вибраний стек інструментів – VS Code, XAMPP і phpMyAdmin – повністю задовольнив вимоги до розробки веб-додатку, забезпечивши зручність у реалізації програмної логіки, управлінні базою даних і тестуванні функціональності. Такий підхід дозволяє ефективно впроваджувати нові функції, масштабувати рішення та гарантувати стабільну роботу системи на різних етапах її життєвого циклу. Завдяки відкритості та гнучкості обраного інструментарію, проєкт може бути адаптований до різних сценаріїв використання без значних витрат часу та ресурсів.

3.4 Алгоритмізація та програмування програмних модулів

Для ефективного управління логікою системи та зручного розподілу функціональних обов’язків між компонентами, у розробленому веб-додатку була реалізована модульна архітектура з чіткою рольовою структурою. Основна ідея полягала у впровадженні архітектурного підходу на основі ролей (Role-Based Architecture), що дозволяє ізолювати функціональність залежно від типу користувача – страхувальника або працівника страхової компанії. Такий підхід забезпечує кращу масштабованість, спрощує підтримку та дає змогу незалежно розвивати окремі частини системи.

Архітектура передбачає логічне групування коду, стилів та ресурсів у окремі каталоги відповідно до функціонального навантаження. Наприклад, кожен тип користувача має окремий модуль, який містить сторінки для відображення інтерфейсу, обробники дій (PHP-скрипти), стилі оформлення, а також допоміжні файли для взаємодії з базою даних. Така структура підвищує

читаємість коду, покращує безпеку системи та полегшує розподіл відповідальності між членами команди при командній розробці.

Загальну схему архітектури веб-додатку наведено на рисунку 3.7. Вона ілюструє логічні блоки системи, структуру взаємодії між ними, а також визначає межі відповідальності кожного з компонентів.

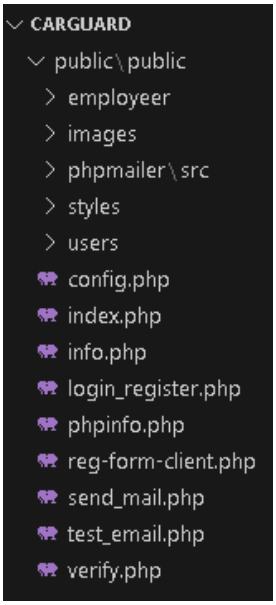

Рис. 3.7 Загальна архітектура проєкту

Для детального аналізу реалізації окремого функціонального блоку розглянемо структуру модуля **users**, що відповідає за обробку дій, пов’язаних із клієнтами системи. До нього входять як файли інтерфейсу, так і серверні обробники. Структуру каталогу модуля користувача наведено на рисунку 3.8, а коротку характеристику кожного компонента – у таблиці 3.4.

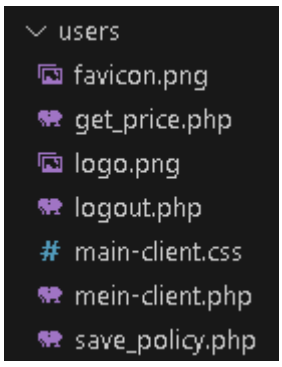

Рис. 3.8 Вміст модулю “users”

Таблиця 3.4

№	Файл	Тип	Короткий опис
1	get_price.php	PHP-скрипт	Обробляє запит на отримання вартості страхового полісу за заданим insurance_type_id. Скрипт підключається до бази даних, виконує SQL-запит до таблиці insurance_types і повертає результат у форматі JSON.
2	logout.php	PHP-скрипт	Виконує вихід користувача з системи.
3	main-client.css	CSS-скрипт	Відповідає за стилізацію веб-інтерфейсу для веб-сайту.
4	main-client.php	PHP-скрипт	Реалізує особистий кабінет користувача, включаючи оновлення профілю, відображення інформації про страхові випадки та роботу з типами страхових полісів.
5	save-policy.php	PHP-скрипт	Оформлює новий страховий поліс у базі даних.

Наведена структура демонструє, що кожен файл у модулі виконує чітко визначену функцію та реалізує окрему частину логіки взаємодії користувача із системою: наприклад, main-client.php забезпечує роботу клієнтського інтерфейсу, а save-policy.php відповідає за збереження даних до бази. Завдяки модульному підходу система зберігає логічну цілісність і легко масштабується, дозволяючи розробникам безпечно розширювати функціональність без ризику порушити роботу інших компонентів, що забезпечує її гнучкість і готовність до подальшого розвитку.

Розроблена модульна архітектура веб-додатку чітко відповідає сучасним принципам побудови масштабованих систем. Використання рольового підходу дозволило ізолювати логіку обробки дій клієнтів і працівників, спростити підтримку коду, а також підвищити безпеку за рахунок розмежування доступу. Обрана структура зберігає логічну цілісність, є гнучкою та готовою до подальшого розширення функціоналу.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Під час перевірки працездатності програмного забезпечення, розробленого для працівника автострахової компанії, особлива увага приділяється відповідності функціоналу заявленим вимогам. Надійність і стабільність у щоденній роботі є критичними показниками якості, оскільки від цього залежить ефективність обслуговування клієнтів і загальна продуктивність внутрішніх процесів. Саме тому тестування є обов’язковим етапом перед впровадженням системи в експлуатацію.

Основним методом перевірки було обрано функціональне тестування, яке орієнтоване на поведінку системи з точки зору кінцевого користувача. Тестування відбувалося шляхом моделювання дій працівника страхової компанії: виконання авторизації, оформлення заявок, перегляд та оновлення даних у профілі, управління договорами. Завдяки цьому вдалося імітувати реальні сценарії використання та виявити можливі відхилення в роботі інтерфейсу.

У межах випробувань перевірялась реакція системи на коректні й некоректні дані, логіка обробки запитів, відображення інформації та відповідність очікуваному результату. Функціональне тестування не передбачає аналіз коду, натомість воно дозволяє оцінити якість реалізації інтерфейсу, зручність навігації та повноту реалізації основних можливостей. Цей підхід є оптимальним для первинного оцінювання працездатності системи в інтерактивному середовищі.

На етапі розробки пріоритет було надано саме функціональній перевірці, оскільки вона надає швидкий і достовірний результат щодо готовності системи до роботи з реальними користувачами. Також такий підхід дозволив виявити та усунути логічні помилки ще до переходу до глибшого рівня тестування –

наприклад, перевірки безпеки або навантаження, які є актуальними на пізніших стадіях впровадження [4].

Розглянемо приклад взаємодії з системою з позиції клієнта під час подання заявки та працівника – під час її обробки. На першому етапі клієнт проходить процедуру авторизації в системі та переходить до розділу оформлення страхового полісу (рис. 4.1).

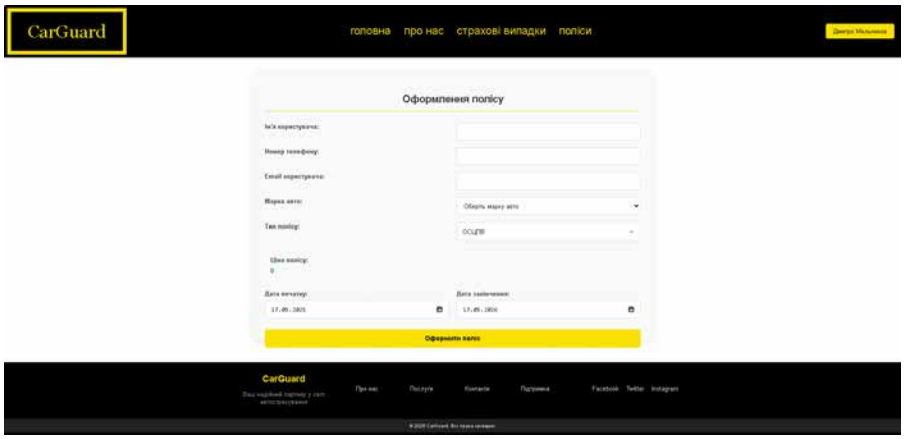

Рис. 4.1 Форма оформлення полісу

Тепер у нас є два варіанти, під час заповнення форми (рис. 4.2) якщо користувач вводить дані він може не ввести якесь поле то тоді вискочить підказка, яка сповістить його щоб він обов’язково заповнив поле, також після заповнення форми, натискається кнопка оформити поліс, і дана заявка вже передається до працівника.

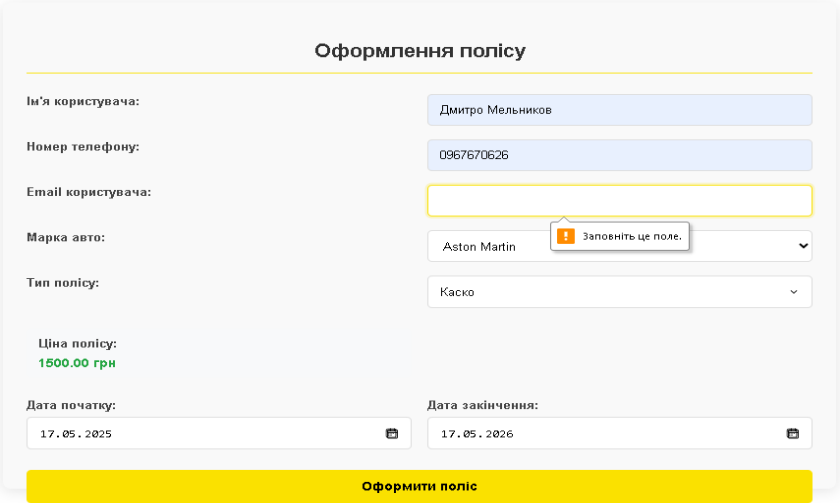

Рис. 4.2 Повідомлення про обов’язкове заповнення

Після того як користувач заповнює форму подання заявки, система перевіряє коректність введених даних. Якщо деякі обов’язкові поля залишено порожніми, автоматично з’являється повідомлення з відповідним попередженням, що спонукає завершити заповнення. У разі успішного заповнення всіх необхідних полів та натискання кнопки підтвердження, заявка передається на розгляд працівнику страхової компанії для подальшої обробки (рис. 4.3).

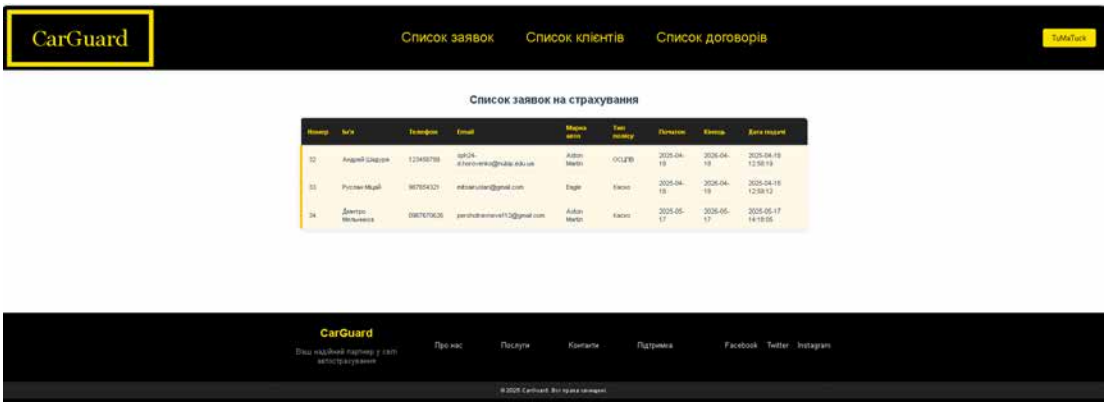

Рис.4.3 Відображення у працівника списку заявок

Як видно на рис. 4.3, заявка на страхування була успішно відправленя, тепер працівник може її переглянути та підтвердити чи відхилити. Для цього після натисканя на потрібну заявку у працівника відкривається модальне вікно де буде відображено повну інформацію про заявку, показано на рис. 4.4.

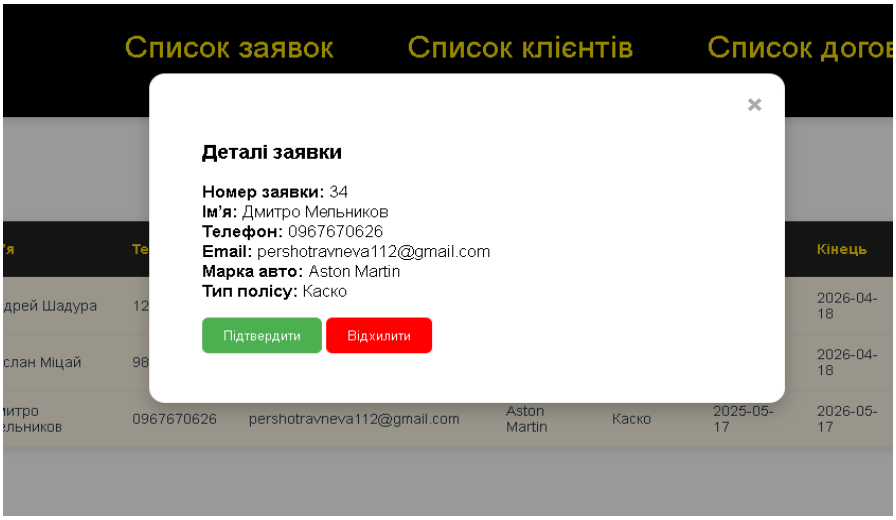

Рис. 4.4 Модальне вікно

Після вибору працівником, підтвердити чи відхилити заявку, вона попадає в список договорів, де після цього вже працівник може її редагувати і змінювати статус.

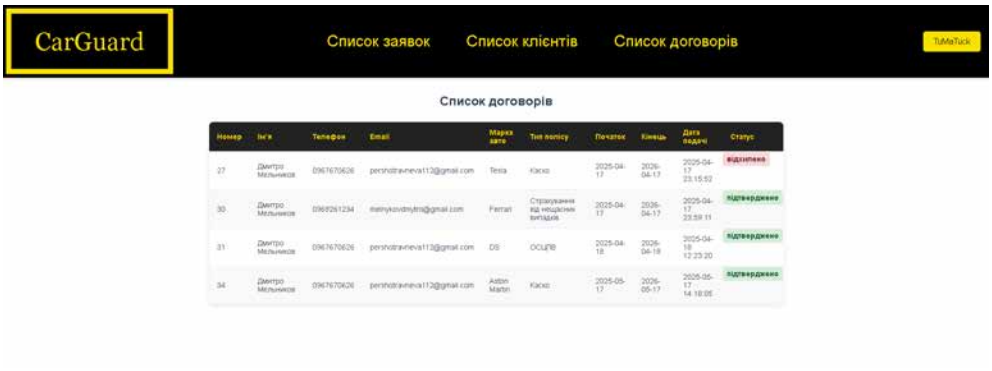

Рис. 4.5 Список договорів

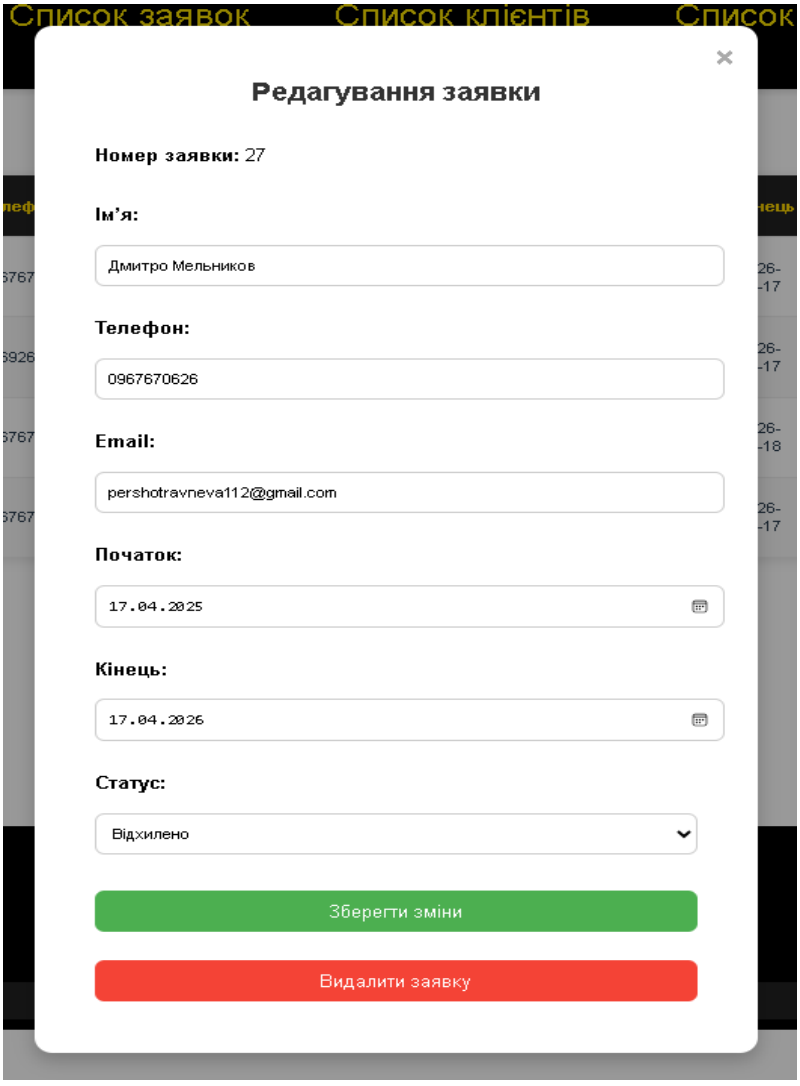

Рис. 4.6 Модальне вікно для редагування заявки

Після редагування заявки у модальному вікні, працівник може зберегти оновлену інформацію, змінити статус договору або видалити його за потреби. Інтерфейс редагування реалізований таким чином, щоб забезпечити швидкий доступ до ключових параметрів заявки без перевантаження користувача зайвими елементами [23]. Усі зміни автоматично фіксуються в базі даних, що дозволяє зберегти актуальний стан інформації та уникнути дублювання чи втрати даних.

За результатами проведеного функціонального тестування підтверджено стабільну роботу ключових модулів системи, зокрема – механізмів подачі та обробки заявок. Інтерфейс коректно реагує на дії користувача, вчасно генерує повідомлення про помилки, а обробка даних виконується відповідно до закладеної логіки. Це засвідчує технічну готовність програмного забезпечення до практичного використання в умовах реального робочого середовища.

4.2 Вимоги до апаратного та програмного забезпечення

Для ефективного функціонування веб-додатку, призначеного для працівників автострахової компанії, критично важливим є забезпечення відповідного апаратного та програмного середовища. Ці вимоги охоплюють характеристики клієнтських пристроїв, серверного обладнання, а також необхідного системного й прикладного програмного забезпечення, що забезпечують стабільну роботу системи, її безпеку, масштабованість і доступність.

Функціонування розробленої системи передбачає взаємодію декількох компонентів, які фізично можуть бути розміщені як на локальних пристроях, так і в хмарному середовищі. Основу архітектури складають веб-сервер, сервер бази даних та клієнтські пристрої. Веб-сервер відповідає за обробку запитів та логіку взаємодії, тоді як база даних забезпечує збереження інформації про користувачів, заявки, страхові поліси та пов’язані з ними сутності. Клієнт взаємодіє з системою через браузер, що дозволяє забезпечити кросплатформенний доступ та адаптивність інтерфейсу [27].

Загальну структуру розгортання системи подано на рис. 4.7. Вона ілюструє фізичну архітектуру розміщення компонентів системи CarGuard, взаємозв’язки між програмними модулями, апаратними засобами та мережею, яка об’єднує користувача, веб-сервер і сервер бази даних.

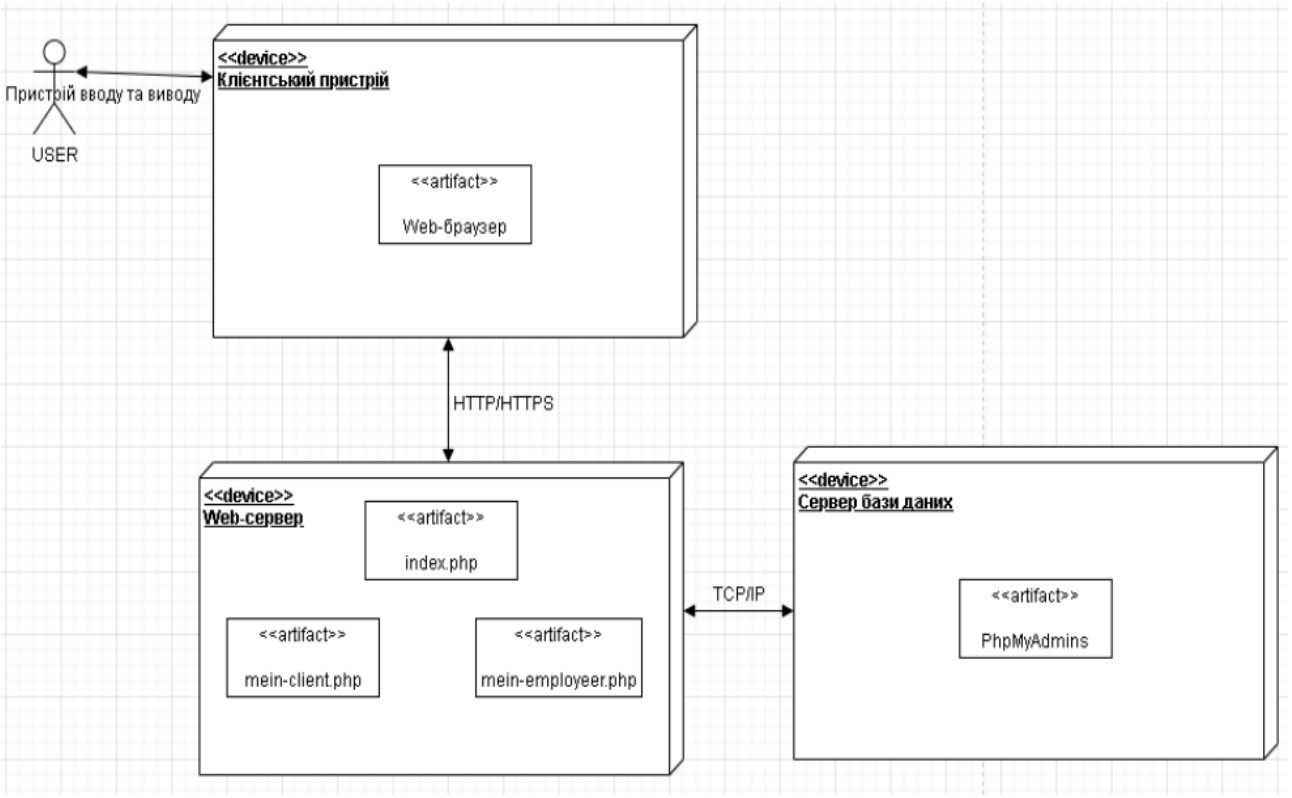

Рис. 4.7 Діаграма розгортання системи

На діаграмі розгортання (рис. 4.7) візуалізовано основні апаратні та програмні компоненти системи CarGuard, а також канали взаємодії між ними. Для детальнішого розуміння функцій кожного елемента та ролей користувачів у системі доцільно подати узагальнення у табличній формі. У таблиці 4.1 систематизовано основні завдання користувачів, функціональність програмних модулів і взаємодію між компонентами інфраструктури.

Зведені у таблиці характеристики дозволяють чітко визначити вимоги до кожного елемента інформаційної системи, забезпечити їхню узгоджену взаємодію та створити передумови для стабільної, масштабованої й безпечної експлуатації програмного забезпечення в умовах реального використання.

Таблиця 4.1

Структура взаємодії користувача з системою CarGuard

Компонент	Опис	Функціональність / Артефакти
Користувач (USER)	Взаємодіє з веб-додатком через браузер (на ПК або мобільному пристрої).	Клієнт: – Перегляд страхових продуктів – Подання заявки – Перевірка статусу – Авторизація профілю Працівник: – Перегляд заявок – Обробка заявок – Редагування
Web-сервер (device)	Обробляє запити від браузера користувача, передає їх до логіки додатку.	Артефакти: – index.php – головна сторінка – mein-client.php – інтерфейс клієнта – mein-employeer.php – інтерфейс працівника Зв’язок: пристрій введення/виводу
Сервер бази даних (device)	Забезпечує зберігання усіх структурованих даних системи: користувачів, заявок, полісів, страхових продуктів тощо.	Артефакт: PhpMyAdmin (інтерфейс до РСУБД) Таблиці: – users – car_brands – insurance_types – policies
Взаємодія між компонентами	Послідовність запитів і відповідей між користувачем, веб-сервером і базою даних.	1. Користувач надсилає HTTP/HTTPS-запит із браузера 2. Веб-сервер обробляє запит 3. Серверна логіка взаємодіє з БД 4. Результат повертається користувачу

Під час визначення вимог до обладнання було враховано необхідність підтримки сучасних веб-технологій, захисту даних, а також гнучкого масштабування. На стороні клієнта передбачено використання персональних комп’ютерів або мобільних пристроїв із сучасними браузерами та підтримкою HTML5, CSS3 і JavaScript. Серверна частина реалізована на основі технології PHP у зв’язці з Apache та СУБД MySQL, що дозволяє забезпечити надійне обслуговування запитів користувачів і зберігання даних.

Для успішного розгортання та стабільної роботи програмного забезпечення CarGuard необхідно дотримуватись певних технічних умов як на стороні клієнта, так і на стороні сервера. У таблиці 4.2 узагальнено апаратні та програмні вимоги до клієнтської і серверної частин системи, а також наведено додаткові компоненти для забезпечення безпеки й інтеграції.

Таблиця 4.2

Апаратні та програмні вимоги до функціонування системи

Компонент	Тип вимог	Характеристика / Параметри
Клієнтська частина	Апаратні	– Сучасний пристрій (ПК, ноутбук, планшет, смартфон) – ОЗП від 2 ГБ – Роздільна здатність від 1024×768
	Програмні	– Google Chrome 90+ – Mozilla Firefox 85+ – Safari 13+ – Microsoft Edge
	Вимоги до браузера	– Увімкнений JavaScript – Підтримка HTML5, CSS3, Fetch/AJAX
Серверна частина	Програмні	– PHP 8.x – (Опційно) Laravel / Express / FastAPI – Apache або NGINX
	База даних	– PhpMyAdmin (інтерфейс до MySQL / MariaDB)
Сторонні інтеграції	Інструменти	– Email API (повідомлення про статус заявки) – SMS API (підтвердження дій) – CAPTCHA для захисту форм
Безпека	Протоколи	– HTTPS – SSL/TLS-сертифікат
	Авторизація	– JWT або Session-based механізм

Виконання зазначених технічних вимог гарантує коректну роботу системи на всіх рівнях – від клієнтського інтерфейсу до серверної логіки й бази даних. Забезпечення сумісності, продуктивності та безпеки створює підґрунтя для подальшого масштабування системи та її ефективного використання в умовах реального навантаження.

З огляду на орієнтацію системи CarGuard на широку аудиторію користувачів, у тому числі мобільних і десктопних, важливо передбачити мінімальні технічні характеристики для їх пристроїв. Таблиця 4.3 узагальнює вимоги до апаратного та програмного забезпечення з погляду кінцевого користувача – як для використання на персональному комп’ютері, так і на мобільному пристрої.

Таблиця 4.3

Вимоги до системи з точки зору користувача

Категорія пристрою	Компонент	Вимоги
Десктоп / ноутбук	Операційна система	Windows 10+, macOS 10.13+, Linux з графічним інтерфейсом
	Оперативна пам’ять	Від 2 ГБ
	Вільне місце	Мінімум 100 МБ (для кешу браузера та тимчасових файлів)
	Екран	Роздільна здатність від 1024×768 (оптимально – Full HD)
Мобільні пристрої	Операційна система	Android 8.0+ або iOS 13+
	Браузер	Актуальні версії Chrome, Safari, Firefox, Edge
	Екран	HD і вище, підтримка адаптивної верстки
Програмні вимоги (веб)	З’єднання	Стабільне підключення від 1 Мбіт/с (рекомендовано 5+ Мбіт/с)
	Функції браузера	Увімкнений JavaScript, підтримка HTML5, CSS3, cookies
	Зберігання	Опціонально: доступ до локального сховища (для сесій та стану користувача)

Наведені технічні характеристики дозволяють забезпечити повноцінний доступ до системи з різних пристроїв – як настільних комп’ютерів, так і мобільних гаджетів. Завдяки підтримці стандартних вебтехнологій, таких як HTML5, CSS3, JavaScript і cookies, інтерфейс додатку зберігає стабільність і зручність незалежно від платформи користувача. Це створює підґрунтя для єдиного, уніфікованого користувацького досвіду, що не залежить від роздільної здатності екрана, операційної системи чи браузера.

Система також адаптована до роботи в середовищі з різними технічними можливостями та підтримує підключення сторонніх сервісів – зокрема API для сповіщень, інтеграцію з Captcha та інструментами авторизації. Таке середовище розгортання не лише знижує поріг входу для користувачів, а й дозволяє масштабувати додаток відповідно до внутрішніх потреб організації або зовнішнього використання в хмарних інфраструктурах.

4.3 Склад інсталяційного пакету

В моєму випадку інсталяційний пакет складається з файлу самого проєкту. Тобто всі необхідні файли, можна встановити на ПК за допомого того самого GitHub, але необхідно також встановити всі необхідні ПЗ, такі як XAMP-для локального хостингу для впроваджень та редагування. Рис. 4.8 та Рис.4.9 показують вміст проєкту та ПЗ.

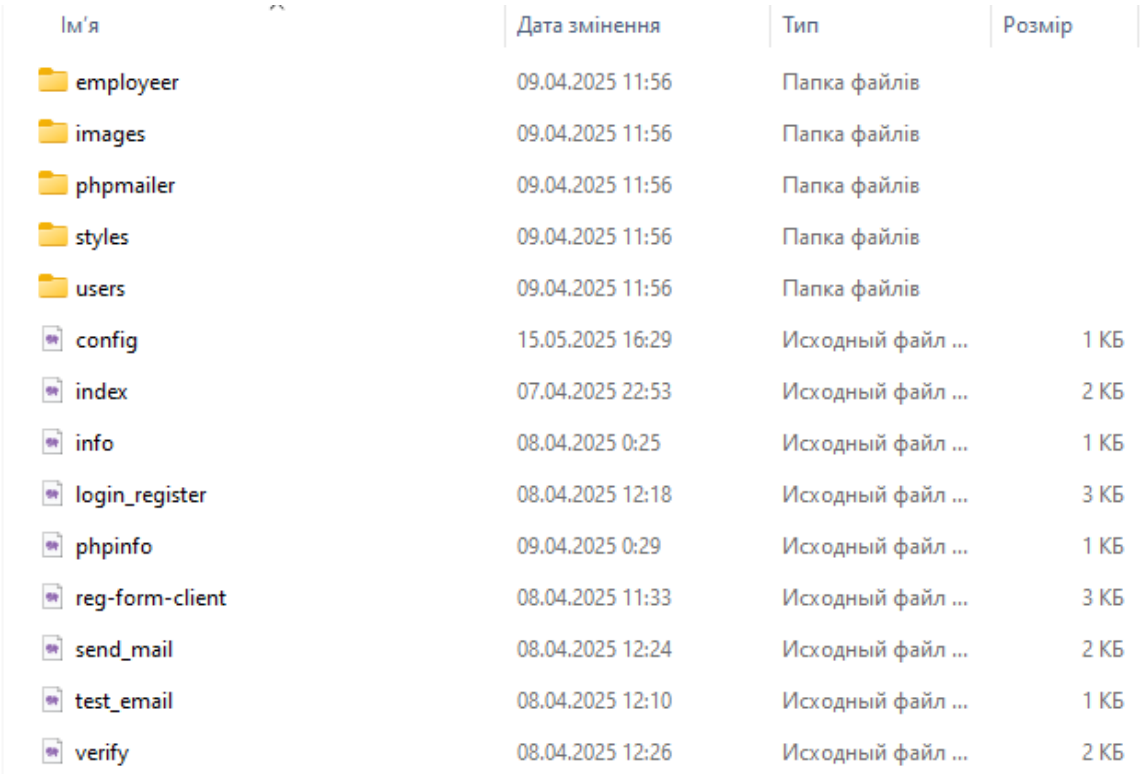

Рис. 4.8 Вміст проекту

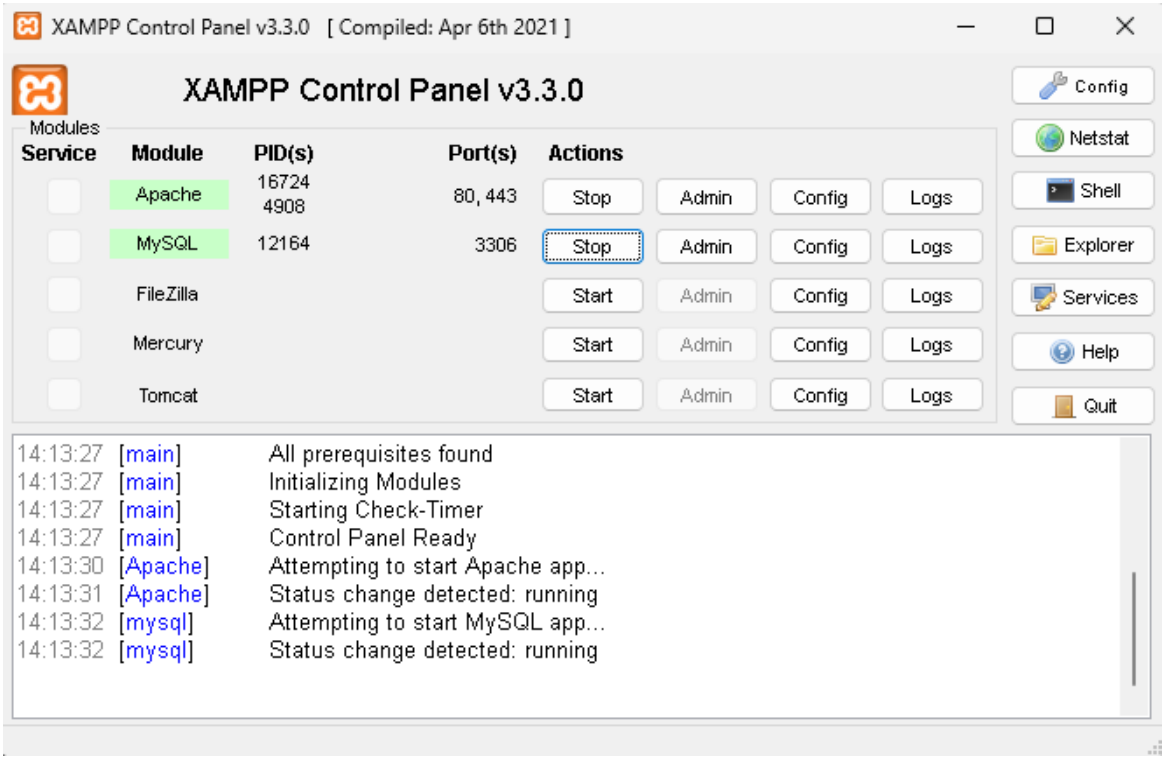

Рис. 4.9 XAMP [16]

Таким чином, до складу інсталяційного пакета входять усі необхідні файли програмного забезпечення та допоміжні компоненти, які забезпечують його швидке розгортання і налаштування. Наявність повного пакета файлів, сумісного середовища розробки та зрозумілих інструкцій гарантує зручність впровадження системи на локальному або хмарному сервері. Це дозволяє мінімізувати технічні труднощі при встановленні та забезпечує безперебійну інтеграцію програмного забезпечення в робоче середовище страхової компанії.

ВИСНОВКИ

У межах виконання бакалаврської кваліфікаційної роботи було спроєктовано та реалізовано веб-додаток «CarGuard», призначений для автоматизації ключових процесів працівника автострахової компанії. Розроблене програмне забезпечення дозволяє ефективно управляти клієнтськими заявками, оформлювати страхові поліси, супроводжувати страхові випадки та вести облік усіх сутностей, що стосуються автострахування. Система підтримує рольову модель доступу, завдяки чому функціонал чітко розподілений між клієнтами та працівниками компанії.

У результаті аналізу предметної області було виявлено основні бізнес-процеси, що потребують цифровізації, визначено перелік функціональних і нефункціональних вимог, а також проведено порівняльний аналіз сучасних інформаційних рішень у сфері автострахування. Це дозволило сформувати цілісне уявлення про архітектуру майбутнього додатку та обґрунтувати доцільність використання веб-інтерфейсу для взаємодії користувача із системою.

Проєктування здійснювалося з використанням сучасних стандартів моделювання, зокрема UML-діаграм, що дозволило сформувати логічну й фізичну структуру даних, а також описати основні сценарії взаємодії користувачів із системою. Для зберігання даних обрано реляційну модель, реалізовану на базі MySQL із використанням phpMyAdmin як інструменту адміністрування. Інтерфейс користувача реалізовано з використанням HTML, CSS та JavaScript, а серверну логіку – мовою PHP у середовищі XAMPP.

Функціональне тестування системи підтвердило її відповідність заявленим вимогам, стабільність роботи, коректну обробку помилок та зручність користування. Архітектура додатку виявилася гнучкою та масштабованою, що дозволяє надалі розширювати функціонал без істотної перебудови системи. Застосована модульна структура спростила навігацію, оновлення коду та забезпечила високий рівень підтримуваності.

Результати роботи можуть бути використані як готове рішення для автоматизації роботи в автострахових компаніях або як базис для подальшої розробки більш складних систем страхування, інтеграції з CRM-сервісами, платіжними платформами та зовнішніми базами даних. Запропоноване рішення демонструє приклад ефективного застосування інструментів веб-розробки для оптимізації процесів у сфері фінансових послуг, зокрема автострахування, що є актуальним у контексті цифрової трансформації галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is UML? [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ – Дата звернення: 12.05.2025.
2. Що таке СУБД і для чого вони потрібні [Електронний ресурс]. – Режим доступу: https://foxminded.ua/systema-upravlinnia-bazamy-danykh/ – Дата звернення: 22.04.2025.
3. Microsoft. SQL Server documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/sql/sql-server/ – Дата звернення: 03.05.2025.
4. Види функціонального та нефункціонального тестування [Електронний ресурс]. – Режим доступу: https://dan-it.com.ua/uk/blog/vidy-funkcionalnogo-i-nefunkcionalnogo-testirovanija/ – Дата звернення: 07.05.2025.
5. Діаграма класів [Електронний ресурс]. – Режим доступу: https://ua5.org/oop/392-diagrami-klasiv.html – Дата звернення: 10.05.2025.
6. Mozilla Developer Network. HTML: HyperText Markup Language [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTML – Дата звернення: 14.05.2025.
7. Mozilla Developer Network. CSS: Cascading Style Sheets [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS – Дата звернення: 14.05.2025.
8. Mozilla Developer Network. JavaScript documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript – Дата звернення: 14.05.2025.
9. PHP Manual. PHP documentation [Електронний ресурс]. – Режим доступу: https://www.php.net/manual/en/ – Дата звернення: 14.05.2025.
10. PZU Україна. Офіційний сайт страхової компанії [Електронний ресурс]. – Режим доступу: https://www.pzu.com.ua/ – Дата звернення: 24.04.2025.
11. ТАС Страхування. Офіційний вебсайт [Електронний ресурс]. – Режим доступу: https://tasinsurance.com.ua/ – Дата звернення: 14.05.2025.

12. UNIQA Україна. Міжнародна страхова група UNIQA [Електронний ресурс]. – Режим доступу: https://uniqa.ua/ – Дата звернення: 14.05.2025.
13. Geico Insurance. Official U.S. Auto Insurance Platform [Електронний ресурс]. – Режим доступу: https://www.geico.com/ – Дата звернення: 12.05.2025.
14. Allianz Group. Global Insurance and Financial Services Provider [Електронний ресурс]. – Режим доступу: https://www.allianz.com/ – Дата звернення: 12.05.2025.
15. phpMyAdmin. Офіційний сайт [Електронний ресурс]. – Режим доступу: https://www.phpmyadmin.net/ – Дата звернення: 14.05.2025.
16. XAMPP. Apache Friends Project [Електронний ресурс]. – Режим доступу: https://www.apachefriends.org/index.html – Дата звернення: 13.05.2025.
17. Visual Studio Code Documentation [Електронний ресурс]. – Режим доступу: https://code.visualstudio.com/docs – Дата звернення: 13.05.2025.
18. ОМG. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання. – К. : Мінекономіки України, 2016. – 16 с.
19. Соммервіль І. Інженерія програмного забезпечення : пер. з англ. / І. Соммервіль. – 9-е вид. – К. : Діалектика, 2011. – 784 с.
20. Бочкор А. Ю., Павлова О. М. Аналіз, проєктування та розробка інформаційних систем : навч. посіб. – Львів : ЛНУ ім. І. Франка, 2020. – 228 с.
21. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.
22. ISO/IEC 9126:2001. Software Engineering – Product Quality – Part 1: Quality Model.
23. Канер К., Фолк Дж., Нгуєн Х. Тестування програмного забезпечення. Фундаментальні принципи. – К. : Діалектика, 2019. – 432 с.
24. Фігурний Ю. Г., Жуков І. В. Основи захисту інформації в комп’ютерних системах. – К. : Видавничий центр «Академія», 2021. – 272 с.

25. Гнатенко І. В., Цал-Цалко Ю. С. Проєктування баз даних : навч. посіб. – К. : КНЕУ, 2020. – 248 с.
26. Глушков В. М., Ситник С. М. Основи інформаційних технологій : навч. посіб. – К. : Каравела, 2020. – 276 с.
27. Кіндратенко В. О. Web-технології та Web-програмування : навч. посіб. – К. : НАУ, 2021. – 210 с.

ДОДАТОК А

Сторінок – 3

Опис таблиць бази даних Web-додатку

```
CREATE TABLE `car_brands` (
  `id` int(11) NOT NULL,
  `name` varchar(100) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
INSERT INTO `car_brands` (`id`, `name`) VALUES
(1, 'Acura'), (2, 'Alfa Romeo'), (3, 'Aston Martin'), (4, 'Audi'), (5,
     'Bentley'),(6, 'BMW'), (7, 'Bugatti'), (8, 'Buick'), (9, 'Cadillac'),
     (10, 'Chevrolet'), (11, 'Chrysler'), (12, 'Citroën'), (13, 'Dacia'),
     (14, 'Daewoo'), (15, 'Daihatsu'), (16, 'Dodge'), (17, 'Donkervoort'),
     (18, 'DS'), (19, 'Ferrari'), (20, 'Fiat'), (21, 'Fisker'), (22,
     'Ford'), (23, 'Genesis'), (24, 'Geely'), (25, 'GMC'), (26, 'Great
     Wall'), (27, 'Haval'), (28, 'Honda'), (29, 'Hummer'), (30,
     'Hyundai'), (31, 'Infiniti'), (32, 'Isuzu'), (33, 'Jaguar'), (34,
     'Jeep'), (35, 'Kia'), (36, 'Koenigsegg'), (37, 'Lada'), (38,
     'Lamborghini'), (39, 'Lancia'), (40, 'Land Rover'), (41, 'Lexus'),
     (42, 'Lincoln'), (43, 'Lotus'), (44, 'Lucid'), (45, 'Mahindra'), (46,
     'Maserati'), (47, 'Maybach'), (48, 'Mazda'), (49, 'McLaren'), (50,
     'Mercedes-Benz'), (51, 'Mercury'), (52, 'MG'),  (53, 'Mini'), (54,
     'Mitsubishi'), (55, 'Nissan'), (56, 'Noble'), (57, 'Opel'), (58,
     'Pagani'), (59, 'Peugeot'), (60, 'Plymouth'), (61, 'Polestar'), (62,
     'Pontiac'), (63, 'Porsche'), (64, 'Proton'), (65, 'RAM'), (66,
     'Renault'), (67, 'Rivian'), (68, 'Rolls-Royce'), (69, 'Rover'), (70,
     'Saab'), (71, 'Samsung'), (72, 'Saturn'), (73, 'Scion'), (74,
     'SEAT'), (75, 'Škoda'), (76, 'Smart'), (77, 'SsangYong'), (78,
     'Subaru'), (79, 'Suzuki'), (80, 'Tata'), (81, 'Tesla'), (82,
     'Toyota'), (83, 'Vauxhall'), (84, 'Volkswagen'), (85, 'Volvo'), (86,
     'Wuling'), (87, 'ZAZ'), (88, 'BYD'), (89, 'Changan'), (90, 'Chery'),
     (91, 'FAW'), (92, 'Foton'), (93, 'GAC'), (94, 'Haima'), (95, 'Hino'),
     (96, 'JAC'), (97, 'Jetour'), (98, 'JMC'), (99, 'Lifan'), (100,
     'Maxus'), (101, 'Nio'), (102, 'Ora'), (103, 'Perodua'), (104,
     'Prodrive'), (105, 'Qoros'), (106, 'Roewe'), (107, 'Seres'), (108,
     'Soueast'), (109, 'Spyker'),  (110, 'Togg'), (111, 'VinFast'), (112,
     'Wey'), (113, 'Xpeng'), (114, 'Zotye'), (115, 'Tatra'), (116, 'UAZ'),
     (117, 'Moskvich'), (118, 'Derways'), (119, 'Eagle'), (120,
     'Pininfarina'), (121, 'Puch'), (122, 'Scania'), (123, 'MAN'), (124,
     'Kamaz'), (125, 'Freightliner'), (126, 'Kenworth'), (127,
     'Peterbilt'), (128, 'Western Star'), (129, 'TAM'), (130,
     'Isdera');ALTER TABLE `car_brands`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `car_brands`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=131;
COMMIT;

CREATE TABLE `insurance_types` (
  `id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `price` decimal(10,2) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
INSERT INTO `insurance_types` (`id`, `name`, `price`) VALUES
(1, 'ОСЦПВ', 500.00),
(2, 'Каско', 1500.00),
(3, 'Страхування від нещасних випадків', 200.00);
ALTER TABLE `insurance_types`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `insurance_types`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;
```

```
COMMIT;

CREATE TABLE `policies` (
  `id` int(11) NOT NULL,
  `user_id` int(10) NOT NULL,
  `insurance_type_id` int(11) DEFAULT NULL,
  `user_name` varchar(255) DEFAULT NULL,
  `user_phone` varchar(15) DEFAULT NULL,
`user_email` varchar(255) DEFAULT NULL,
  `start_date` date DEFAULT NULL,
  `end_date` date DEFAULT NULL,
  `price` decimal(10,2) DEFAULT NULL,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp(),
  `status` varchar(20) DEFAULT 'новий',
  `car_brand_id` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
INSERT INTO `policies` (`id`, `user_id`, `insurance_type_id`, `user_name`,
      `user_phone`, `user_email`, `start_date`, `end_date`, `price`,
      `created_at`, `status`, `car_brand_id`) VALUES
(27, 6, 2, 'Дмитро Мельников', '0967670626',
      'pershotravneva112@gmail.com', '2025-04-17', '2026-04-17', 1500.00,
      '2025-04-17 20:15:52', 'відхилено', 81),
(30, 9, 3, 'Дмитро Мельников', '0969261234', 'melnykovdmytro@gmail.com',
      '2025-04-17', '2026-04-17', 200.00, '2025-04-17 20:59:11',
      'підтверджено', 19),
(31, 6, 1, 'Дмитро Мельников', '0967670626',
      'pershotravneva112@gmail.com', '2025-04-18', '2026-04-18', 500.00,
      '2025-04-18 09:23:20', 'підтверджено', 18),
(32, 6, 1, 'Андрей Шадура', '123456789', 'sph24-d.horovenko@nubip.edu.ua',
      '2025-04-18', '2026-04-18', 500.00, '2025-04-18 09:58:19', 'новий',
      3),
(33, 6, 2, 'Руслан Міцай', '987654321', 'mitsairuslan@gmail.com', '2025-
      04-18', '2026-04-18', 1500.00, '2025-04-18 09:59:12', 'новий', 119),
(34, 6, 2, 'Дмитро Мельников', '0967670626',
      'pershotravneva112@gmail.com', '2025-05-17', '2026-05-17', 1500.00,
      '2025-05-17 11:18:05', 'підтверджено', 3);
ALTER TABLE `policies`
  ADD PRIMARY KEY (`id`),
  ADD KEY `user_id` (`user_id`),
  ADD KEY `insurance_type_id` (`insurance_type_id`);
ALTER TABLE `policies`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=35;
ALTER TABLE `policies`
  ADD CONSTRAINT `policies_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES
      `users` (`id`),
  ADD CONSTRAINT `policies_ibfk_2` FOREIGN KEY (`insurance_type_id`)
     REFERENCES `insurance_types` (`id`);
COMMIT;

CREATE TABLE `users` (
  `id` int(10) NOT NULL,
  `name` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  `role` enum('user','employeer') NOT NULL,
```

```
  `verification_token` varchar(255) NOT NULL,
  `is_verified` tinyint(1) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
INSERT INTO `users` (`id`, `name`, `email`, `password`, `role`,
      `verification_token`, `is_verified`) VALUES
(6, 'Дмитро Мельников', 'pershotravneva112@gmail.com',
      '$2y$10$ZcXS8aLk9odk.z54Esq24eN2zBBA4RGBFmpoT.Q48ZTlBx.9FIc9S',
      'user',
      '86979eadda4f3725e251af0898af0948a3ff8e1b65b2e1e38c3cfe3e1fd6745c',
      NULL),
(8, 'TuMaTuck', 'ipz21-d.melnykov@nubip.edu.ua',
      '$2y$10$4OBRcFEhNbnDe0xFUk9y4eNMLaEEgAcUjLuHTPWbx1d.kqItvM82e', '',
      'c264c0319620a9f298bc7c0f3f892646cf3aa45eae64c8e0b54e315eda52246d',
      NULL),
(9, 'TuMaTuck', 'melnykovdmytro@gmail.com',
      '$2y$10$io8/Y7BS0wtHp3MIGJZjA.eTahvHIZElp1r3ifMX/iOO.mScx8GjK',
      'user',
'827acd41f9f1b555b8b31a1f49cb958175e0c84e79538c7c8327d1c89defd3c0', NULL);
ALTER TABLE `users`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `users`
  MODIFY `id` int(10) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=10;
COMMIT;
```

ДОДАТОК Б

Сторінок – 6

Програмні рішення головного екрану інтерфейсу Web-додатку

Index.php

```
<?php
session_start();
$errors = [
    'login' => $_SESSION['login_error'] ?? ''
];
$activeForm = $_SESSION['active_form'] ?? 'login';
session_unset();
function showError($error) {
    return !empty($error) ? "<p class='error-message'>$error</p>" : '';
}
function isActiveForm($formName, $activeForm) {
    return $formName === $activeForm ? 'active' : '';
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CarGuard</title>
    <link rel="shortcut icon" type="image/png"
     href="../public/images/favicon.png">
    <link rel="stylesheet" href="../public/styles/styles-log-form.css">
</head>
<body>
    <div class="form">
       <div class="center-container">
       <div class="logo">CarGuard</div>
       </div>
<div class="wrapper <?= isActiveForm('login', $activeForm); ?>">
        <form action="login_register.php" method="post">
            <h1>Вхід</h1>
        <?= showError($errors['login']); ?>
        <div class="input-box">
            <input type="email" name="email" placeholder="Пошта"
     id="email" required>
       </div>
       <div class="input-box">
            <input type="password" name="password" placeholder="Пароль"
     id="password" required>
       </div>
       <button type="submit" name="login" class="btm-log"
     id="login">Увійти</button>
       <div class="register-link">
            <p>Не зарегистровані? <a href="reg-form-
     client.php">Зарегистуватись</a></p>
       </div>
       </form>
    </div>
    </div>
</body>
</html>
```

Config.php

```
<?php
$serverName = "localhost\\SQLEXPRESS";
$connectionOptions = array(
    "Database" => "CarGuardDB",
    "Uid" => "sa",
    "PWD" => "sa",
    "CharacterSet" => "UTF-8"
);
// Підключення
$conn = sqlsrv_connect($serverName, $connectionOptions);
// Перевірка
if (!$conn) {
    die(print_r(sqlsrv_errors(), true));
} else {
    echo "Підключено до SQL Server!";
}
?>
```

login_register.php

```
<?php
session_start();
require_once 'config.php';
require_once '../public/send_mail.php';
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['register'])) {
    $name = trim($_POST['name']);
    $email = trim($_POST['email']);
    $password = trim($_POST['password']);
    $role = trim($_POST['role']);
 // Перевірка, чи email вже існує
    $stmt = $conn->prepare("SELECT id FROM users WHERE email = ?");
    $stmt->bind_param("s", $email);
    $stmt->execute();
    if ($stmt->get_result()->num_rows > 0) {
        $_SESSION['register_error'] = "Цей email вже зареєстрований";
        header("Location: reg-form-client.php");
        exit();
    }
    // Генерація токену
    $verificationCode = bin2hex(random_bytes(32));
    $hashedPassword = password_hash($password, PASSWORD_DEFAULT);
    // Збереження користувача
    $stmt = $conn->prepare("INSERT INTO users (name, email, password,
      role, verification_token) VALUES (?, ?, ?, ?, ?)");
    $stmt->bind_param("sssss", $name, $email, $hashedPassword, $role,
      $verificationCode);
    if ($stmt->execute()) {
        if (sendVerificationEmail($email, $name, $verificationCode)) {
            $_SESSION['success'] = "Реєстрація успішна! Перевірте вашу
      пошту для підтвердження.";
        } else {
            $_SESSION['register_error'] = "Помилка відправки листа
      підтвердження";
        }
    } else {
        $_SESSION['register_error'] = "Помилка при реєстрації";
```

```
    }
    header("Location: reg-form-client.php");
    exit();
}
if (isset($_POST['login'])) {
    $email = $_POST['email'];
    $password = $_POST['password'];
    $result = $conn->query("SELECT * FROM users WHERE email = '$email'");
    if ($result->num_rows > 0) {
        $user = $result->fetch_assoc();
        if (password_verify($password, $user['password'])) {
            $_SESSION['name'] = $user['name'];
            $_SESSION['email'] = $user['email'];
            $_SESSION['user_id'] = $user['id'];
            if ($user['role'] === 'user') {
                header("Location: users/mein-client.php");
            } else {
                header("Location: employeer/mein-employeer.php");
            }
            exit();
        }
    }
    $_SESSION['login_error'] = 'Incorrect email or password';
    $_SESSION['active_from'] = 'login';
    header("Location: index.php");
    exit();
}
?>
```

reg-form-client.php

```
<?php
session_start();
$errors = [
    'register' => $_SESSION['register_error'] ?? ''
];
$activeForm = $_SESSION['active_form'] ?? 'login';
session_unset();
function showError($error) {
    return !empty($error) ? "<p class='error-message'>$error</p>" : '';
}
function isActiveForm($formName, $activeForm) {
    return $formName === $activeForm ? 'active' : '';
}
?>
<!DOCTYPE html>
<html lang="ua">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CarGuard</title>
    <link rel="shortcut icon" type="image/png"
      href="../public/images/favicon.png">
    <link rel="stylesheet" href="../public/styles/styles-reg-
      form.css?v=1.0.1">
</head>
```

```
<body>
<div class="form">
    <div class="center-container">
    <div class="logo">CarGuard</div>
    </div>
<div class="wrapper <?= isActiveForm('register', $activeForm); ?>">
        <form action="login_register.php" method="post">
            <h1>Регистрація</h1>
        <?= showError($errors['register']); ?>
        <div class="input-box">
            <input type="text" name="name" placeholder="Ім'я"
     id="username" required>
        </div>
        <div class="input-box">
            <input type="email" name="email" placeholder="Пошта"
     id="email" required>
        </div>
        <div class="input-box">
            <input type="password" name="password" placeholder="Пароль"
     id="password" required>
        </div>
        <select name="role" required>
            <option value="">--Виберіть роль--</option>
            <option value="user">Страховик</option>
            <option value="employee">Працівник</option>
        </select>
        <p class="registration-note">
    Після реєстрації вам буде надіслано лист для підтвердження електронної
     адреси.
        </p>
        <button type="submit" class="btm" id="submit"
     name="register">Зарегиструвatись</button>
        <div class="login-link">
             <p>Зарегистровані? <a href="index.php">Увійти</a></p>
        </div>

        </form>
    </div>
</div>
</body>
</html>
```

mein-employeer.php

```
<?php
session_start();
require_once '../config.php';
// Перевірка авторизації
if (!isset($_SESSION['email'])) {
    header("Location: ../index.php");
    exit();
}
$email = $_SESSION['email'];
// Якщо це POST-запит для оновлення профілю
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (!isset($_POST['name'], $_POST['email'])) {
```

```
            $_SESSION['error'] = "Будь ласка, заповніть всі обов'язкові
        поля!";
        } else {
            $name = trim($_POST['name']);
            $new_email = trim($_POST['email']);
            $password = trim($_POST['password']);

            // Перевірка унікальності email
            $stmt = $conn->prepare("SELECT email FROM users WHERE email = ?
        AND email != ?");
            $stmt->bind_param("ss", $new_email, $email);
            $stmt->execute();
            $result = $stmt->get_result();

            if ($result->num_rows > 0) {
                $_SESSION['error'] = "Цей email вже використовується!";
            } else {
                // Оновлення користувача
                if (!empty($password)) {
                    $hashed_password = password_hash($password,
        PASSWORD_DEFAULT);
                    $stmt = $conn->prepare("UPDATE users SET name = ?, email =
        ?, password = ? WHERE email = ?");
                    $stmt->bind_param("ssss", $name, $new_email,
        $hashed_password, $email);
                } else {
                    $stmt = $conn->prepare("UPDATE users SET name = ?, email =
        ? WHERE email = ?");
                    $stmt->bind_param("sss", $name, $new_email, $email);
                }

                if ($stmt->execute()) {
                    $_SESSION['email'] = $new_email; // Оновлення сесії
                    $_SESSION['success'] = "Профіль успішно оновлено!";
                } else {
                    $_SESSION['error'] = "Помилка оновлення!";
                }
            }
            $stmt->close();
        }

}
// Отримання даних користувача
$query = $conn->query("SELECT name, email FROM users WHERE email =
        '$email'");
$user = $query->fetch_assoc();
$_SESSION['name'] = $user['name'];
?>
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CarGuard</title>
    <link rel="shortcut icon" type="image/png" href="favicon.png">
    <link rel="stylesheet" href="styles-main-employeer.css">
```

```
</head>
<body>
    <div class="header">
        <div class="container">
            <div class="header-line">
                <div class="logo"><img src="logo.png"
     alt="CarGuard"></div>
                <div class="nav">
                    <a class="za" href="#">Список заявок</a>
                    <a class="cl" href="#">Список клієнтів</a>
                    <a class="do" href="#">Список договорів</a>
                    <a class="tex" href="#">Тех. підтримка</a>
                </div>
                <a class="button" href="javascript:void(0);"
     onclick="toggleProfileForm()">

                    <?= htmlspecialchars($_SESSION['name']) ?? "Особистий
     кабінет" ?>
                </a>
            </div>
        </div>
    </div>
    <div class="profile-form" id="profileForm">
        <h2>Особистий кабінет</h2>
        <?php if (isset($_SESSION['error'])) { echo "<div class='message
     error'>{$_SESSION['error']}</div>"; unset($_SESSION['error']); }
             if (isset($_SESSION['success'])) { echo "<div class='message
     success'>{$_SESSION['success']}</div>"; unset($_SESSION['success']);
     } ?>
        <form method="post">
            <label>Ім'я:</label>
            <input type="text" name="name" value="<?=
     htmlspecialchars($user['name']) ?>" required>
            <label>Email:</label>
            <input type="email" name="email" value="<?=
     htmlspecialchars($user['email']) ?>" required>
            <label>Новий пароль:</label>
            <input type="password" name="password" placeholder="Введіть
     новий пароль (необов'язково)">
            <button type="submit">Оновити</button>
        </form>
        <a href="mein-employeer.php">Назад</a>
        <a href="logout.php">Вийти</a>
    </div>
    <script>
        function toggleProfileForm() {
            var form = document.getElementById('profileForm');
            form.style.display = (form.style.display === 'none' ||
     form.style.display === '') ? 'block' : 'none';
        }
        setTimeout(() => {
     document.querySelectorAll('.message').forEach(msg =>
     msg.style.display = 'none'); }, 3000);
    </script>
</body>
</html>
```