НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
___Комп'ютерних наук___
(назва кафедри)
_____ __Голуб Б.Л._
(підпис) (ПІБ)
"_3" червня 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Програмне забезпечення інформаційної системи страхування»

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
____к.н.т., доцент_____ _____ ____Вайганг Г.О._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)
Керівник бакалаврської кваліфікаційної роботи
____ст. викладач_____ _____ ____Міловідов Ю.О._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)
Виконав _____ _____Джало Олександр Володимирович ____
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
к.н.т., доцент _____ Голуб Б.Л.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"_3" червня 2025 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи студенту
Джало Олександр Володимирович
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи Програмне забезпечення інформаційної системи автострахування

Затверджена наказом ректора НУБіП України від "16" грудня 2024р. №2249 С

Термін подання завершеної роботи на кафедру 2025.05.25
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи

Опис предмету дослідження, опис програмного забезпечення

Перелік питань, які потрібно розробити:

Системний аналіз предметної області

Аналіз предметної області

Розробка програмного забезпечення

Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання "16" грудня 2024 р.

Керівник бакалаврської кваліфікаційної роботи _____ Міловідов Ю.О.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Джало О.В.
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІС – Інформаційна система.
ПЗ – Програмне забезпечення.
БД – База даних.
СУБД – Система управління базами даних.
SQL – Structured Query Language, мова запитів до баз даних.
PDF – Portable Document Format, формат документів.
VIN – Ідентифікаційний номер транспортного засобу.
Т/З – Транспортний засіб.
CVV – Тризначний код безпеки банківської карти.
API – Application Programming Interface, програмний інтерфейс.
C# – Мова програмування.
UML – Unified Modeling Language, мова моделювання.
IDE – Integrated Development Environment, середовище розробки.
AI – штучний інтелект.

ВСТУП

На сьогоднішній день розвиток інформаційних технологій та автоматизація процесів в галузі страхування визнаються ключовими чинниками для покращення ефективності та конкурентоспроможності компаній. Щороку автострахування залучають все більше клієнтів, що очікують швидкого й зручного обслуговування при оформленні страхових послуг. У багатьох випадках споживачам доводиться стикатися із складним та неефективним інтерфейсом платформи для оформлення страховки, нав'язливою бюрократичною процедурою й важким доступом до представників страхових компаній. Тому на сьогодні постала необхідність у розробці ефективної й функціональної системи для поліпшення обслуговування клієнтів і оптимізації бізнес-процесів для забезпечення простоти користування сервісом.

Спеціалізоване програмне забезпечення для автострахування в інформаційній системі дозволяють здійснити повний цикл укладення страхового полісу від реєстрація користувача до підтвердження оплати. Генерація документа для підтвердження наявності страхового полісу на автомобіль та взаємодія з представниками страхових компаній. Система чинить чітке роздільне межування між ролями - користувачем, менеджером та адміністратором та наділяючи їх гнучким управлінням і рівнем безпеки в системі.

Ця програма наддасть можливість користувачу зреагувати на обрання або скористатися допомогою менеджера для вибору відповідних страхових полісів у залежності від потреб користувача і проведення оплати. Менеджер маю підтверджувати факт оплати страхового полісу. По завершенню цього процесу менеджер підтверджуватиме можливість користувача сформувати PDF-файл із своїм страховим полісом. В документі буде вказано дані про автомобіль та його власника. Адміністратор системи буде володіти контролем над самою системою:

додати компанії до системи та укладе угоду з постачальниками послуг для надання страхованих полісів; формою звітів; керують користувачами та розподіллять їх права в системі.

Актуальність

Маючи великий попит на цифрові послуги в галузі страхування, актуальність створення інформаційної системи автострахування є беззаперечною. Більшість випадків страхових компаній досі не мають зрозумілі програмні рішення, які дозволяють здійснити самостійний або за допомогою менеджерів підібрати підходящі варіанти, який міг би задовольнити клієнта.

Інформаційна система створюється для усунення основних проблем пов’язані з недоліками шляхом простого використання, функціональності, безпечності інформаційній системі, що дозволяє оптимізацію діяльності користувачів, менеджерів, та розширити можливість адміністрування та допомогти покращити користувацький досвід клієнтів.

Тема

Розроблювальна система присвячена розробці програмного забезпечення інформаційної системи автострахування, що дозволяє забезпечити реєстрацію, вибір, оформлення та підтвердження страхових полісів, маючи чітку та сформовану структуру ролей користувачів.

Зв'язок роботи з науковими програмами, планами, темами

Розробка роботи над науковими планами, темами та програмами, зберігала чітку та зрозумілу послідовність дій та її виконання згідно плану науково-дослідних робіт факультету інформаційних технологій Національного університету біоресурсів і природокористування України.

Об’єкт дослідження

Об’єкти досліджень проводились над такими сервісами, як: **polis.ua** – призначений для великого обсягу страхування, як від страхування життя так і

авто; **Monobank** та **ПриватБанк** основна система розрахована на обслуговування клієнтів пов’язані з банком, але система так само має можливість оформити страховий поліс маючи не зручний інтерфейс підбору страхового полісу на авто.

Предмет дослідження

Предметом дослідження є структура функціональність програмного забезпечення, яка має поетапний шлях до оформлення страхового полісу, адміністрування системи та взаємодія між ролями системи.

Задачі дослідження

1. Провести аналіз аналогів інформаційних систем у галузі страхування.
2. Визначення функціональності інформаційної системи.
3. Проектування бази даних реалізованої у середовищі PostgreSQL.
4. Реалізація програмний додаток, а саме клієнтський, адміністративний інтерфейс на базі Windows Forms використовуючи мову програмування C#.
5. Створення механізму реєстрації, авторизації, вибір полісу, підтвердження сплати та генерація електронного полісу за допомогою формату PDF.
6. Обмеження прав доступу відповідності рівня доступу до системи.
7. Провести тестування, усунути неполадки та розробити оцінку її надійності, масштабованості та зручність використання.
8. Розроблення та створення звітності системи для адміністратора.

Методи дослідження

На Початковий етап розробки програмного забезпечення для інформаційної системи автострахування було проведено аналіз існуючих аналогів серед яких: polis.ua, онлайн-страхування через додаток Monobank та послуги автострахування додатком ПриватБанк. Ці всі три сервіси мають зручний інтерфейс та великий

об’єм страхових компаній, які надають свої послуги. Але кожен з них має певні недоліки у використанні або мають певні обмеження.

Polis.ua виконує переважно функцію агрегатора, не маючи забезпечення збереження історії сплати або зворотного зв’язку з менеджерами під час купівлі або після. **Monobank** реалізований без реєстрації, а саме маючи лише акаунт клієнта банку, що зменшує коло користувачів. ПриватБанк – оформлення обмеження під час купівлі страхового полісу, а саме обмежує партнерське коло страхових компаній і має відсутність сформований акт про оформлення страхового полісу, що має бути одразу після підтвердження сплати. Також, кожна система не має ролевого доступу до керування контентом, прикладом є те що, жоден менеджер або адміністратор не має можливість додати контент або допомогти тим чи іншим способом клієнту.

Маючи такі обмеження, прийнято рішення створити власну інформаційну систему для поєднання всі можливі функціональності, які були обмеження або не додані аналогами. Саме для цього буде зроблено простоту оформлення та гнучкість адміністрування. Для цього буде використано такі технічні та дослідницькі підходи:

- **Проєктування бази даних**: використано методи нормалізації та концептуального моделювання, створення структури бази даних за допомогою СУБД PostgreSQL, яка буде враховувати ролі, а також таблиці зберігання компаній, полісів, замовлення, платежі та звіти.
- **Об'єктно-орієнтований підхід**: застосування принципів об'єктно-орієнтованого програмування, маючи ціль створити програмну логіку з використанням мови C# у середовищі Windows Forms, що має забезпечувати модульність, а саме головне повторне використання коду та легкість у масштабуванні.

- **Система доступу**: реалізація системи багаторівневого доступу до функцій системи, залежно від ролі користувача, дозволяж забезпечити безпеку та ізоляцію даних.
- **PDF**-генерація: автоматиче створення страхового полісу у форматі **PDF**, після підтвердження замовлення. Досліджено кілька бібліотек для роботи з PDF у .Net-середовищі. Обрана найбільш стабільна та зручна – iTextSharp, дозволяючи створити стилізований документ на основі шаблону.
- **Обробка сплати**: оплати було змодельовано з урахуванням можливості його інтеграції з **API** платіжних систем. У дипломній роботі реалізовано умовну оплату, яка в реальному застосуванні може бути легко замінена на реальний платіжний шлюз.
- **Формування звітів**: адміністратор має можливість згенерувати звіт на основі замовлень за потребою компаній.
- **Тестування**: проведено ряд тестувань з кожних компонентів системи з метою перевірки стабільності та цілісності даних та правильність їх обробки.
- **UI/UX**-дослідження: застосування простого та зручного принципу використанні для кожного користувача. Клієнт має для себе лише необхідні йому дії для оформлення полісу, менеджер має окремий кабінет, а адміністратор – доступ до всього. Це дає змогу зменшити навантаження на систему.

Теоретична значимість

Теоретична значимість полягає у розробці концептуального підходу для створення інтегрованої інформаційної системи, що має охоплювати процеси автоматизації, обробки та управління даними. У рамках роботи досліджено та реалізовано взаємодію таких теоретичних аспектів, як: побудова реляційної бази

даних, моделювання ролей користувачів та доступ, об’єктно-орієнтоване програмування та проєктування інтерфейсу.

Окрім цього, було виділено увагу для побудови архітектури, що дає змогу поєднати функції менеджменту страхових полісів, генерації їх у форматі PDF та забезпечення багаторівневої взаємодії між користувачем, менеджером та адміністратором. Таким чином, дослідження робить внесок у побудові прикладного ПЗ для галузевих інформаційних систем.

Практична значимість

Практична значимість інформаційної системи автострахування полягає в автоматизації ключових етапів оформлення страхових полісів, дозволяючи зменшити час обробки запиту та зручність для кінцевого споживача.

Система дозволяє:

- Швидке оформлення полісу в інтерактивному середовищі;
- Оперативну зміну або додавання інформації про страхові продукти;
- Адміністрування, що має забезпечувати контролювання повного циклу функціональності та аналітичності системи.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Одна із сучасних проблем в сфері страхування є швидкість оформлення та швидка комунікація із менеджерами, які б могли допомагати клієнту під час оформлення. Головна роль при обслуговуванні клієнтів повинна грати – швидкість. Основні розробки програмних забезпечень не розраховують та не думають про швидкість оформлення, те що повинно бути першою метою для розробників.

Предметною областю для розроблювальної ІС є сфера автострахування, що охоплює діяльність страхових компаній та їх послуги для транспортних засобів. Розвиток технологій та зростання обсягу транспортного потоку потребує ефективному обліку, аналізу та обслуговування договорів страхування, реєстрації страхових випадків та зберігання інформації, що має забезпечувати особисту конфіденційність користувача.

Автострахування має важливу роль у складовій безпеки на дорогах та економічний захист власників авто. Забезпечування компенсації збитків у випадку аварій, пошкодження чи викрадання авто, так само має гарантувати відповідальність між третіми особами. Велика кількість клієнтів, транспортних засобів та динамічні зміни законодавства, використовуючи сучасні програмні засоби є обов’язково для ефективності.

Рішення спрямоване на спрощування щоденної роботи працівників страхових компаній та створення швидкого сервісу для клієнтів. Одна з ключових особливостей системи є багаторівнева структура, що дозволяє обмежити доступ до різних функціональних частин програми, залежності від ролі користувача.

Система має в демонстрації українську мову, оскільки програмний продукти орієнтовано передусім на український ринок страхових послуг. У подальшому

розвиток систему може бути масштабований та адаптований під інші мови та стандарти, що дозволить розширити ринок та вийти на інший рівень.

Розробка програмного забезпечення відповідає актуальним запитам ринку, забезпечує ефективність бізнес-процесів у сфері автострахування.

1.2 Аналіз вимог до програмної системи

Функціональні вимоги повинні мати перелік тих функцій та сервісів, що насамперед повинна виконуватись системо.

Нефункціональні вимоги визначаються для функції програмного забезпечення, що оцінюються для критерії якості роботи.

Основуючись на аналізі предметної області, буде розроблено та вписано функціональні вимоги та не функціональні вимоги.

Функціональні вимоги:

1) Авторизація користувача з урахуванням ролі(звичайний користувач, менеджер, адміністратор);
2) Реєстрація нових клієнтів за допомогою валідації ведення даних;
3) Користувач самостійно може переглядати власні страхові договори;
4) Створення нового страхового договору;
5) Розрахунок вартості страхування;
6) Підтвердження або відхилення заявки менеджером системи;
7) Формування страхового полісу на основі сплати користувача;
8) Перегляд статистики продажу оформлених страхових полісів;
9) Формування та експорт договорів у форматі PDF.

Нефункціональні вимоги:

1) Інтерфейс користувача повинен бути зручний, зрозуміли у використанні;
2) Чіткий поділ доступу до функціональності відповідно до ролі користувача;

3) Швидкість обробки запитів до бази даних за великою кількістю запитів;

4) Цілісність та актуальність даних при одночасному доступі;
5) Мінімальна кількість збоїв;
6) Безпечне зберігання особистих даних користувачів.

1.3 Моделювання предметної області

Для моделювання предметної області було прийнято розумним рішенням розробити мовою моделювання опис предметної області. Уніфікована мова моделювання, грає основну роль у розробці.

Моделювання предметної області дозволяє зрозуміти логіку функціонування розроблюваної ІС автострахування. Виявити основні компоненти та ролі користувачів та взаємодію між ними.

ІС автострахування буде спрямована на автоматизацію процесів між користувачами, менеджерами та адміністраторами. Основна мета створення програмного продукту, що дозволяє перш за все, зручність та швидкість оформлення страхових полісів, контроль статусу оплати та адміністрування.

Для розробки було прийнято рішення розробити мовою моделювання використовуючи UML діаграми поведінки: діаграма прецедентів, активності та послідовності.

Діаграма Прецедентів візуально зображає різноманітні сценарії взаємодії між акторами (користувачами) і прецедентами (випадками використання); описує функціональні аспекти системи (бізнес логіку).

Діаграми Прецедентів відіграють важливу роль не тільки у комунікації між збирачами вимог до проекту і потенційними користувачами. Діаграми Прецедентів дописані бізнес логікою і детальними специфікаціями прецедентів, як джерельна інформація, успішно використовують учасники розробки проекту на всіх його фазах (зародження, дизайн, програмування, тестування,

документування..). Добре продумані і завершені специфікації прецедентів легко перевтілюються у Тестові Випадки. [1]

На рис. 1 буде представлено діаграму прецедентів для розроблювальної ПЗ.

Ця діаграма представляє з себе наявність 4 актора, а саме: користувач(основний клієнт системи), менеджер (є посередником між компанією та послугами), компанія (представляє умовно послуги), адміністратор (керуюча система управління). Представлено в діаграмі також прецеденти: Оформлення страхового полісу, вибір страхового полісу, перегляд страхових пропозицій, провести оплату, продаж страхового полісу, оформлення полісу, управління страховими пропозиціями, створення нових користувачів, реєстрування фірми.

Рис. 1 Діаграма прецедентів

1. **Клієнт (користувач):**

Кінцевий споживач послуг автострахування, звернення якого є отримання послуг та придбання страхового полісу:

Робота з оформленням страхового полісу включає в себе такі компоненти:

- Вибір страхового полісу;
- Перегляд пропозицій.

2. **Менеджер:**

Співробітник, який відповідає за обробку заявок та супровід процесу страхування.

- Робота з продажем страхових полісів, що включає себе:
    - Оформлення полісу (етап підготовки та підтвердження даних);

3. **Адміністратор:**

- Робота з страховими полісами;
- Нові користувачі;
- Робота з компаніями.

1.4 Огляд інформаційних джерел та існуючих рішень

Аналізуючи існуючі рішення для програмного забезпечення автострахування, можна було зрозуміти, що з актуальних рішень є електронні сервіси. За моїм рішенням було прийнято розробити десктопний застосунок.

На рис. 2 та рис. 3 буде зображено початковий варіант аналога, про який потрібно знати. Електронний сервіс, можна вивести, як гігант в свої сфері, чим він і служить для користування функціональності.

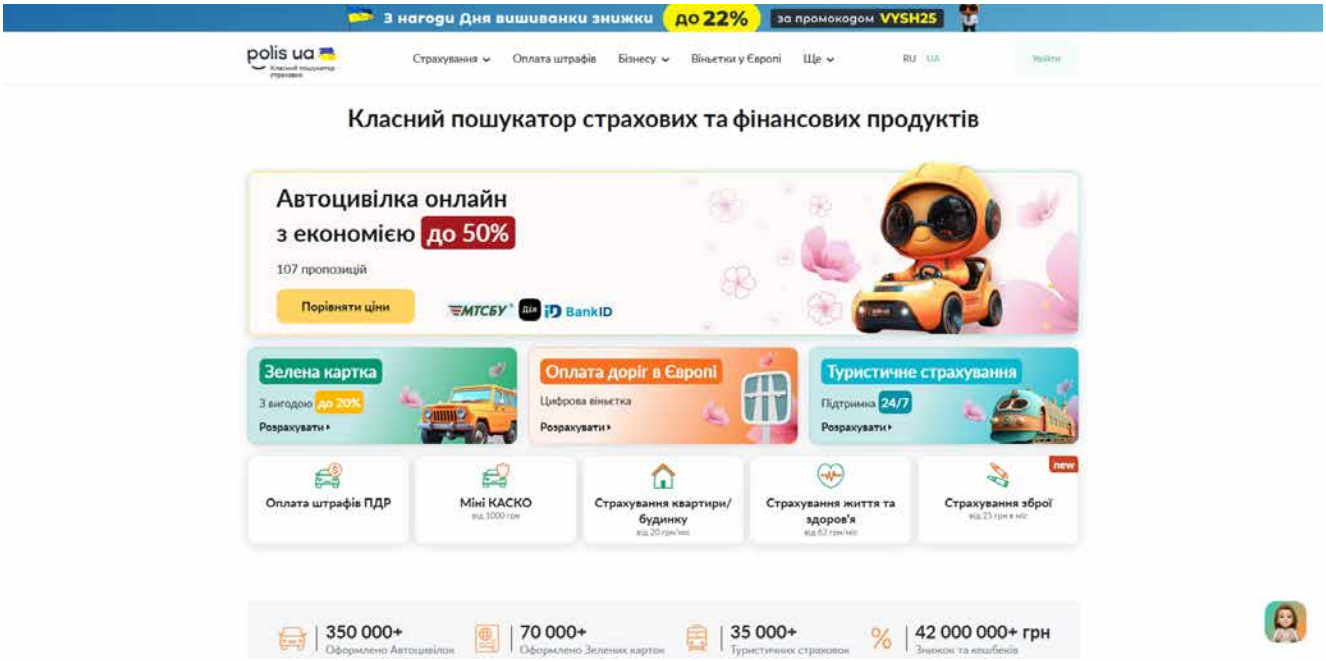

Рис. 2 Електронний сервіс страхування – Polis.ua

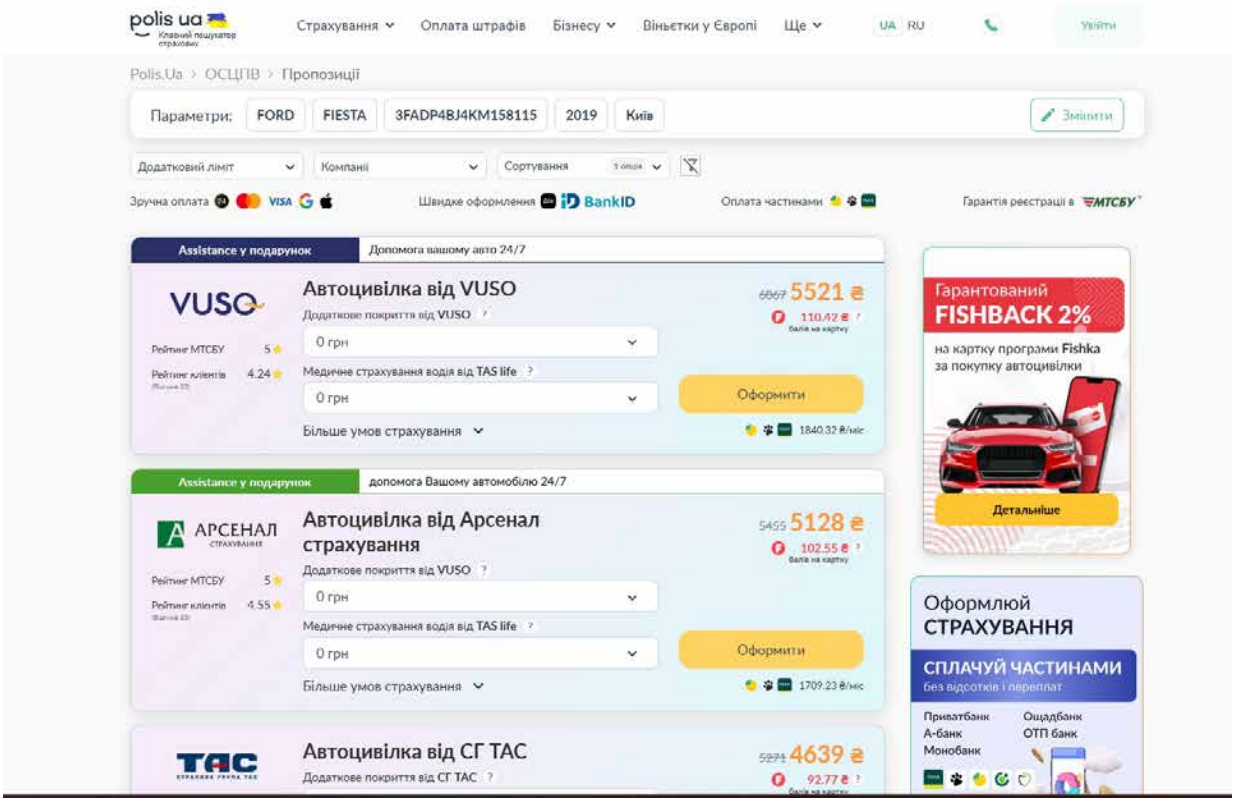

Рис. 3 Список пропозицій за заданими параметрами електронного сервісу – Polis.ua

Polis.ua – є одна із ключових напрямків дослідження, для цього використовуючи доступні сервіси можна проаналізувати те, що є у відкритому доступі веб-сервісу, реалізовуючи аналог функціональності до тієї, що розробляється для дипломного проєкту. Сиди входить ряд сервісів з оформлення страхуванні, а саме:

1. Сервіс з оформлення автострахування;
2. Управління страховими пропозиціями;
3. Електронна взаємодіє з клієнтом.

Один з найбільших та відомих поширених ресурсів є веб-платформа Polis.ua, що використовується для оформлення страхових полісів різних типів, враховуючи тип страхування ОСЦПВ, туристичне, медичне, КАСКО та страхування життя тощо.

Веб-сайт орієнтований для масового користування користувачам, що бажають самостійно обирати страхові поліси та придбати його без відвідування офісу страхових компаній. Основна форма заповнення відбувається на сайті де користувач самостійно заповнює дані, а саме: марку авто, тип транспортного засобу, місто реєстрації та його номер. Надалі генерується автоматичний перелік доступних варіантів полісу для наданого інформацію про авто користувача, які обов’язково мають договір з платформою для співробітництво, що надає перевагу тому, що дійсно страхова компанія пройшла певні перевірки та має офіційний статус страхової компанії.

Опис про страхову компанію має вигляд у зручній таблиці зі вказанням вартості, назви, наявність додаткових послуг, а точніше допомога на дорозі, технічна підтримка тощо та можливість оформлення страхового полісу одразу на сайті. Здійснення оплати проходить лише онлайн, через банківську систему або інші послуги сплати. Коли оплата пройшла, автоматично надсилається на пошту страховий поліс за яким було вказано на початку заповнення даних.

Веб-сайт забезпечує роботу в одних із сучасних браузерах, таких як Google Chrome, Mozilla, Firefox, Safari, Microsoft Edge та інших. Саме головне, що можливо виділити – це не потрібно встановлювати додаткових програмних забезпечень, що дозволяє лише швидкий та простий доступ до сервісу та широкого кола користувачів.

Платформа дозволяє зберігати надану інформацію при заповнені попередніх даних та історію страхувань, але лише для зареєстрованих користувачів, дозволяючи спростити процес повторного оформлення полісу при наступному звернені до сервісу.

Polis.ua розглядається, як один з важливих інформаційних ресурсів, дозволяючи ознайомитись зі структурою та логікою роботи подібного цифрового сервісу, його архітектуру із взаємодією з користувачами, формами зберігання або введення даних, алгоритм розрахунку та генерація результату у вигляді електронного формату PDF.

Наступний, виділяється в цій системі наступний мобільний додаток Monobank (рис. 4).

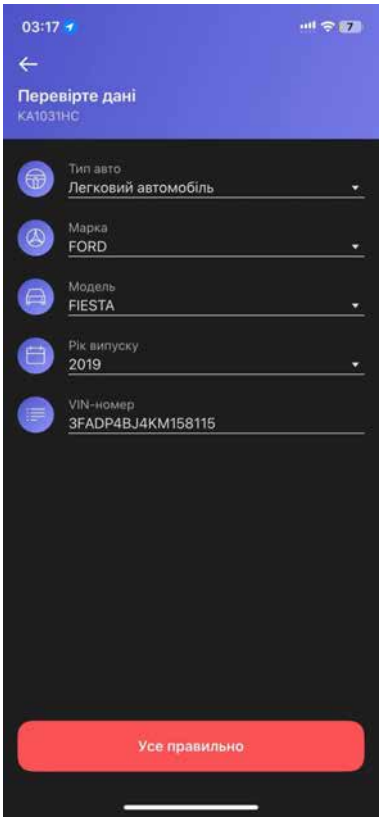

Рис. 4 Приклад інтерфейсу мобільного додатку Monobank.

Monobank – це перший в Україні банк, який повністю є мобільним банком та без традиційних фізичних відділень. Архітектура базується на сучасних технологіях, дозволяючи виконувати усі банківські операції через мобільний застосунок. Перевага такого мобільного додатку полягає в тому, що банк надає широкий спектр фінансових послуг за мінімальним залученням персоналу та без необхідності фізичної присутності клієнта у відділені.

Серед сервісу, які надає Monobank, є також і сфера страхування. В мобільному додатку, окремо розташована категорія для цих послуг, що робить його швидким та зручним у використанні. Користувач обирає тип страхування (життя, здоров’я, подорожі, автострахування тощо), ознайомлюючись з умовами та погоджуватись з ними, здійснити оплату та зберегти електрону копію безпосередньо у застосунку.

Особливість реалізаціє є повна автоматизація процесу без присутності будь-яких співробітників банку або менеджерів страхових компаній. Після обрання

типу страхування, клієнт переходить до заповнення особистої інформації про авто або інші параметри за типом страхування. В такому випадку клієнту не потребується внесення особистої інформації по типу ПІБ, місці проживання тощо, це все підтягується автоматично, якщо клієнт є вже в базі банку, але усвідомимо про те, що користуватись мобільним додатком не можливо, якщо ти не є клієнтом банку. Обираючи параметри, клієнт має можливість лише обрати та сплатити за допомогою внутрішнього платежу функціоналу застосунку. Після сплати, поліс автоматично генерує його у форматі PDF-файлу та надає можливість зберегти його собі на пристрій або відправити у якийсь месенджер.

Monobank у цій системі виконує роль, як посередник, що надає клієнту доступ до страхових сервісів партнерських компаній через єдину інтегровану платформу. Виходячи з цього, це надає можливість зменшити час оформлення та мінімізувати людський фактор. Взаємодія зі страховими компаніями виконує через API-зв’язок.

Важлива особливість системи є підтримка push – сповіщень, що дозволяє інформувати клієнта про статус полісу: спливання терміну дії страхового полісу, інформація про активацію тощо. Тобто, Monobank, забезпечує повноцінний цикл взаємодій між користувачем та страховими компаніями та їх послугами.

Monobank постійно збільшує свій спектр можливостей та сервісів, інтегруючи їх в один цілий мобільний додаток. Це дозволяє залучати нових партнерів, адаптація під нульову аудиторію банку.

На рис. 5 буде представлено спектр пропозицій страхових полісів, які пропонує додаток за заданими параметрам, а саме: тип авто, марка, модель, рік випуску та VIN – номер згідно з прикладом мобільного додатку на рис. 4.

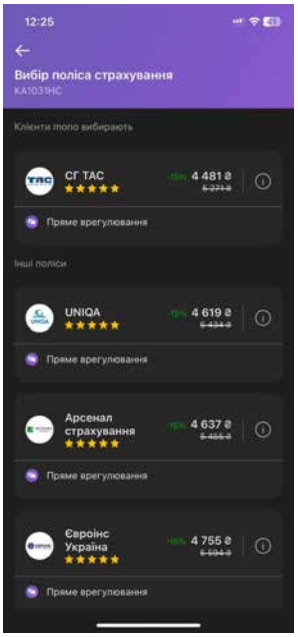

Рис. 5 Приклад пропозицій страхових полісів додатку Monobank.

Monobank додаткова перевага системи, які напряму пов’язані з банками, система перевіряє чи дійсно присутня у вас страховий поліс діючи. Приклад перевірки буде представлено на рис. 6

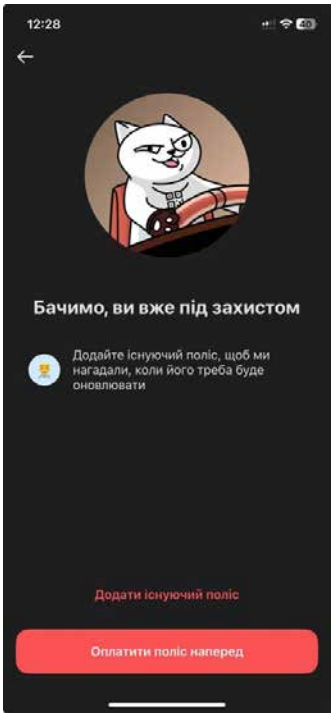

Рис. 6 Приклад дійсності та наявності страхового полісу через мобільний додаток Monobank.

Наступним та останнім буде додаток ПриватБанк (рис. 7).

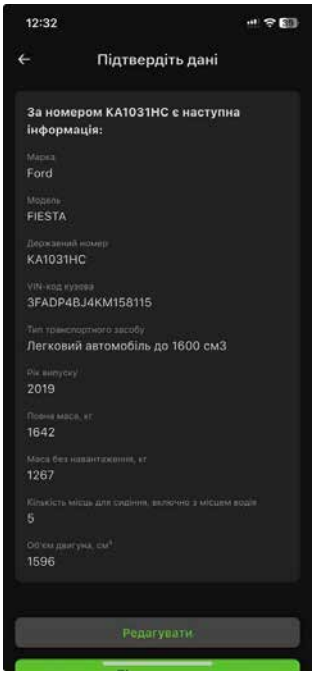

Рис. 7 Приклад інтерфейсу мобільного додатку Privat24

ПриватБанк – найбільший комерційний банк України, що є одним з найрозвиненіших цифрових екосистем у банківській сфері. Банк активно розвиває різний спектр послуг. Одним із них є сектор надання онлайн-страхування, наприклад: ОСЦПВ та «Зелена картка». Сервіс має можливість використання за допомогою: мобільного застосунку Privat24, веб-сайту або за допомогою терміналу самообслуговування.

Етап оформлення страхового полісу не є складним в цій функціональній системі:

- Обрання типу страхового полісу на головній сторінці(сайт або додаток);
- Введення основних даних, а саме інформація про автомобіль;
- Перегляд запропонованих страхових полісів від різних страхових компаній, що співпрацюють з банком.

- Користувачі ознайомлюються з умовами та обирають найбільш вигідний їм варіант та сплачують онлайн картою або іншими способами.
- Після успішної сплати, клієнту приходить у електронному форматі PDF-файл на електронну пошту або збергіється у особистому кабінеті.

ПриватБанк надає можливість не тільки автострахування, а ще й розширене страхування, для подорожі за межі країни, страхування водія та пасажира, а також КАСКО. Головний плюс в цій системі, що банк зберігає історію оформлення страхових полісів, надсилаючи запити на повторне страхування після завершення дії страхового полісу.

Банк виступає майже так само, як і Monobank. Головне для клієнтів – це надійність безпеки даних та акредитовані страхові компанії, що гарантує захищеність платежів та документального оформлення.

Автоматизація процесів перевірки за допомогою інтеграції з державними реєстрами (наприклад, МТСБУ, Сервісний центр МВС), що дозволяє зменшити помилки при оформлені або усунути їх взагалі.

Також на рис. 8 буде представлено пропозиції компаній, які співпрацюють з банком – ПриватБанк.

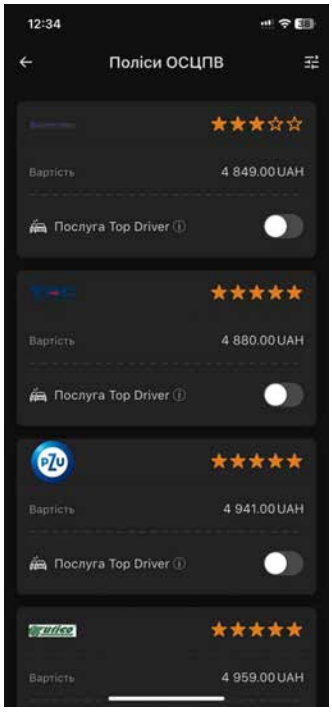

Рис. 8 Приклад пропозицій страхових полісів додатку Privat24.

Для порівння переваг та недоліків, буде створено таблицю де буде порівнюватись. Система порівняння буде створена для того, щоб розуміти, різницю між системи, існує виходом вдосконалення між ними, де це все буде представлено в (табл. 1).

Таблиця 1

Порівняння існуючих рішень

Сервіс	**Переваги**	**Недоліки**
Polis.ua	Можливість порівння пропозицій в одному інтерфейсі; Швидке оформлення; підтримка електроного підпису.	Обмеження в кількості партнерських страховиків; Немає актуальності ціни за проблем із затримкою бази; Немає інтеграції у банківські сфері або мобільні додатки.

Monobank	Цифровий процес оформлення; Інтеграція з банківським функціоналом; Майже миттєве отримання та сформування електронного полісу; Автоматизація підв'язуючись до бази МВС.	Менше коло страхових партнерів, тим самим зменшується запропонування страхових полісів; Наявність страхових продуктів не завжди задовольняє потреби користувача.
ПриватБанк	Широкий вибір страхових продуктів; Оформлення, як онлайн, так і в офісі; Архів полісів.	Менше автоматизовано, порівнюючи з Monobank; Повільна обробка замовлень; Деякі випадки потребують підтверджень від клієнтів.

1.5 Постановка завдання

Розробка ПЗ ІС автострахування, має можливість на створення ефективного інструменту автоматизації, швидкість та зручність. Система повинна мати зручну та безпечну взаємодію між трьома ключовими сторонами: користувач, менеджер, адміністратор. Більшість страхових компаній мають труднощі з ручною обробкою запитів клієнтів, зберігаючи велику кількість даних про договори страхування, т/з, особисту інформацію, розрахунки, можливості втратити інформацію, дублювання записів та помилки у документаціях.

Розроблювальна інформаційна система має вирішувати вказані проблеми, що завдяки впровадженню є чіткою структурованою моделлю з підтримкою різних ролей. Користувачі можуть реєструватись в системі, вносити персональні дані, додавати інформацію про автомобіль, оформляти страховий поліс та переглядати історію страхувань. Менеджери матимуть змогу надавати консультації та оброблювати заявки клієнтів. Адміністратор отримує повний

контроль над управлінням бази даних, в його перевазі роботи є робота з компаніями.

Інформаційна система не передбачена у використані складних механізмів пошуку або фільтрації, оскільки основна функціональність зосереджена на створенні та обслуговуванні страхових полісів, облік користувача та обмін інформацією між менеджером. Основною метою ж розробити процес взаємодій простим та надійним у використанні.

Процес роботи системи передбачує наступну логіку:

1) Реєстрація клієнта у системі, заповнюючи форму для зберігання даних на майбутнє.
2) Після реєстрації, починається наступний етап, обирання страхового полісу, та присутня можливість додати власний т/з у систему для генерації страхових полісів.
3) Менеджер має змогу допомогти клієнту обрати найбільш підходящий варіант та обробляти отриману заявку.
4) Після успішної сплати, клієнту надається можливість отримати страховий поліс у форматі PDF.
5) Адміністратор буде забезпечувати контроль усіх процесів з налаштування усіх її параметрів.

Система має забезпечувати стабільний та безпечний обміни даних з базою даних та системою маючи обмеження по користувачам. Очікуючи, що така система дозволяє суттєво знизити навантаження на менеджерів та уникнути втрату даних, прискорюючи обробку запитів та зручність для клієнтів, що має велику роль у сфері обслуговування.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

ER-модель (Entity-relationship model або Entity-relationship diagram) – це семантична модель даних, яка призначена для спрощення процесу проектування бази даних. З ER-моделі можуть бути породжені всі види баз даних: реляційні, ієрархічні, мережні, об’єктні. В основі ER-моделі лежать поняття “сутність”, “зв’язок” та “атрибут”.

ER-модель – це представлення бази даних у вигляді наочних графічних діаграм. ER-модель візуалізує процес, що визначає деяку предметну область. Діаграма “сутність-зв’язок” – це діаграма, яка представляє в графічному вигляді сутності, атрибути і зв’язки.

Це тільки концептуальний рівень моделювання. ER-модель не містить деталей реалізації. Для тієї самої ER-моделі деталі її реалізації можуть відрізнятися.[2]

Проєктуючи інформаційну систему, одна із найважливіших пунктів є розробка логічну модель даних. Основою цього процесу є створення логічної моделі, що висвітлює сутність, атрибути та зв'язки між ними. В ІС автострахування ця модель представляє себе у вигляді ER – діаграму, що дозволяє візуалізувати структуру бази даних, реляційні залежності та ключові аспекти системи.

Розпочинаючи з проєктування, основним компонентом є саме проєктування бази даних з чого будується її логічна модель даних. Логічна модель даних надає головне поняття в структурі – це спрощує зрозуміти структуру та її логічність у системі.

Оглядаючи логічну модель даних, потрібно розуміти її аспект, поняття тощо.

Логічна модель даних має свої елемент, завдяки, якім користувач, що не розуміє структуру, йому дасть поняття такі елементи, як:

Логічна модель даних складається з основних елементів:

1) сутності (*entities*) – це об’єкт, що має визначені характеристики та може бути однозначно ідентифікований. Наприклад, у базі даних інтернет - магазину можуть бути сутності “Товар”, “Клієнт”, “Замовлення”. Кожна з цих сутностей має свої властивості – наприклад, сутність “Товар” може мати назву, опис, ціну тощо;

2) атрибути (attributes) – це властивість сутності, яка характеризує її. Наприклад, атрибути сутності "Товар" можуть бути назва, опис, ціна, категорія тощо. Атрибути сутності дозволяють детально описати її характеристики;

3) відношення (*relationships*) – це залежність між двома сутностями. Зв’язок може бути один до одного (1:1), один до багатьох (1:N) або багато до багатьох (N:M). Наприклад, у базі даних інтернет-магазину може бути зв’язок між сутностями "Клієнт" та "Замовлення", оскільки кожен замовлення може бути пов’язаний з одним конкретним клієнтом;

4) Ключ (key) – це атрибут, який використовується для однозначної ідентифікації сутності. Кожна сутність має свій ключ, який дозволяє унікально ідентифікувати кожен запис у базі даних. Наприклад, у сутності "Товар" ключем може бути його унікальний ідентифікатор або код товару. [3]

Для проєктування логічної моделі на початку обрано використовувати Erwin Data Modeler, тому, що для багатьох – це є потужним інструментом. Але виходячи з того, що для користування потрібна ліцензія, тому було вирішено обрати інший сервіс для проєктування. Обраний електронний серіс lucid.app. Самец цей сервіс візуально відображає повноцінно логічну модель даних, також додаючи до цього, логічна модель даних буде відображати тип даних, що є дуже зручним для розуміння та коректування помилок на майбутє.

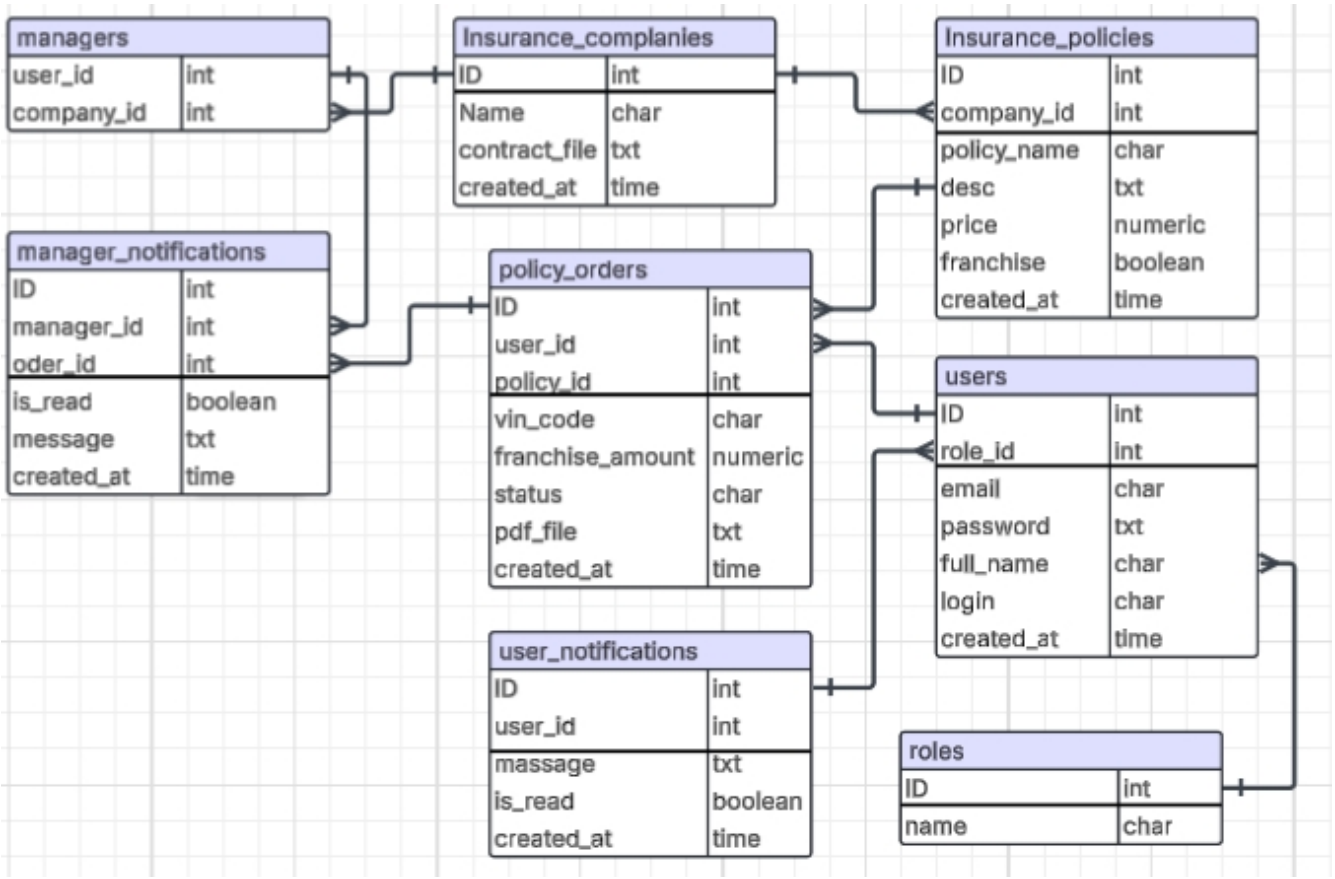

Рис. 9 Логічна модель даних

Рис. 9 показує 8 сутностей, логічної моделі даних ІС автострахування, детально описуючи:

1) *User*s – користувач системи: містить ключовий атрибут id (ідентифікатор користувача) та маючи шість неключових атрибутів: офіційне ПІБ згідно з паспортом громадянина України, електронна пошта, логін (для авторизації), role_id для визначення ролі користувача у системі(адміністратор, менеджер, користувача тобто звичайний клієнт) та останнє created_at_timestamp інформація дати та часу створення облікового запису у системі;

2) *Roles* – роль у системі: включає лише один ключовий атрибут – id (ідентифікатор ролі у системі), та один не ключовий атрибут – назва ролі (name), що містить в собі тип ролі користувача(менеджер, адміністратор, користувач;

3) *Insurance_copmanies* – страхові компанії: страхові компанії містить в собі лише один ключовий атрибут id(ідентифікатор ролі у системі) та три неключових атрибути: назва компанії, шлях до прикріплення файлу договору та дата додавання компанії в систему. Ця таблиця має логічну прив'язку полісів та менеджерів компанії;

4) *Managers* – менеджери компанії – це лише зв’язкова таблиця між користувачами компаній. Вона містить в собі два зовнішні ключі: user_id(ідентифікатори користувача, а саме менеджера який є менеджером) та company_id(ідентифікатор страхової компанії в якій працює менеджер);

5) *Insurance_policies* – страхові поліси: містить лише один ключовий атрибут – id (ідентифікатор поліса) та зовнішній ключ company_id (ідентифікатор компанії до якої буде належити страховий поліс), наступна інформація, яка буде містити: назва полісу, опис, ціна, потребність франшизи та дата створення запису. Так буде спрощено прикріплювати до існуючи компанії страховий поліс;

6) *Policy_orders* – замовлення страхових полісів: містить один ключовий атрибут – код замовлення(*id*), два зовнішніх ключі – user_id(кому належить замовлення) та policy_id (поліс, який обрано користуваче). Але для оформлення замовлення додано було більш детальну інформацію, яка потрібна для страхового полісу, а саме: vin-код транспортного засобу, сума франшизи(має містити нуль або більше), статус замовлення, шлях до PDF – файлу договору та дата створення. Це допоможе створити інтерфейс оформлення замовлення;

7) *User_notifications* – повідомлення для користувача, тобто зв’язок між користувачем та менеджером для консультації: містить один ключовий атрибут – код повідомлення (*Id*), зовнішній ключ user_id(для кого призначене повідомлення) та три неключових атрибут: текст повідомлення, статус чи прочитав користувач, якому надійшло повідомлення та дата створення. Це більш зручним методом реалізації системи спілкування між користувачем та менеджером;

8) *Manager_notifications* – Містить структуру для повідомлення менеджеру: має в собі аналогічну структуру, як з user_notifications, але створенна для менеджерів та конкретним замовленням. Присутній ключовий атрибут: id (ідентифікатор повідомлення), зовнішній ключ manager_id(ідентифікатор менеджера), та order_id (ідентифікатор замовлення), також три неключових атрибутів: текст, статус прочитаного та час створення.

2.2 Діаграма класів

Подальша дія розробки складалась з проєктування Діаграми класів. Діаграма класів необхідний для реалізації та візуалізації структури класів системи. Діаграма класів містить в собі атрибути, методи, зв’язки та класи, а також за допомогою цієї діаграми можна визначити можливі проблеми та усунути їх в подальшому для розробки.

Діаграма класів – категорія речей, що мають загальні атрибути та операції. Сама діаграма класів являє собою набір статичних, декларативних елементів моделі. Вона дає нам найбільш повне і розгорнуте уявлення про зв'язки в програмному коді, функціональність та інформацію про окремі класи. [4]

На рис. 10 буде зображено діаграму класів системи автострахування. Де:

1) клас *User* є створений для користувачів системи, собою представляє основним класом. Основним те що він містить – це дані про логін, пошту, повне ім’я, пароль, роль користувача та дату реєстрації. Згідно ролі, користувач з роллю «користувача», може оформити страховий поліс, мати консультацію зі штучним інтелектом;

2) клас *InsuranceCompany* відповідає за пердставлення страхових компаній, які маю певний договір та має право на реалізацію своїх послуг споживачам. Він містить у собі: назву компанії, шлях до файлу договору, та насамперед страхові поліси будуть прив’язані завжди до своїх страхових компаній, що в майбутньому буде полегшувати робити звіти по продажам;

3) клас *Manager* має представлення окремого класу, що реалізує зв'язок між користувачем та страховою компанією. Менеджер має особливий виняток в цій системі, тому, що це особливий користувач, що відповідає за допомогу оформлення страхового полісу та зміни статусу замовлення;

4) клас *PolicyOrder* збергіє замовлення користувачів, де фіксується дійсність сплати, інформацію про користувачів хто замовив та дані про авто, з чого формується PDF-файл;

5) клас *Administrator* містить в собі лише логін, маючи можливість редагувати, створювати тощо;

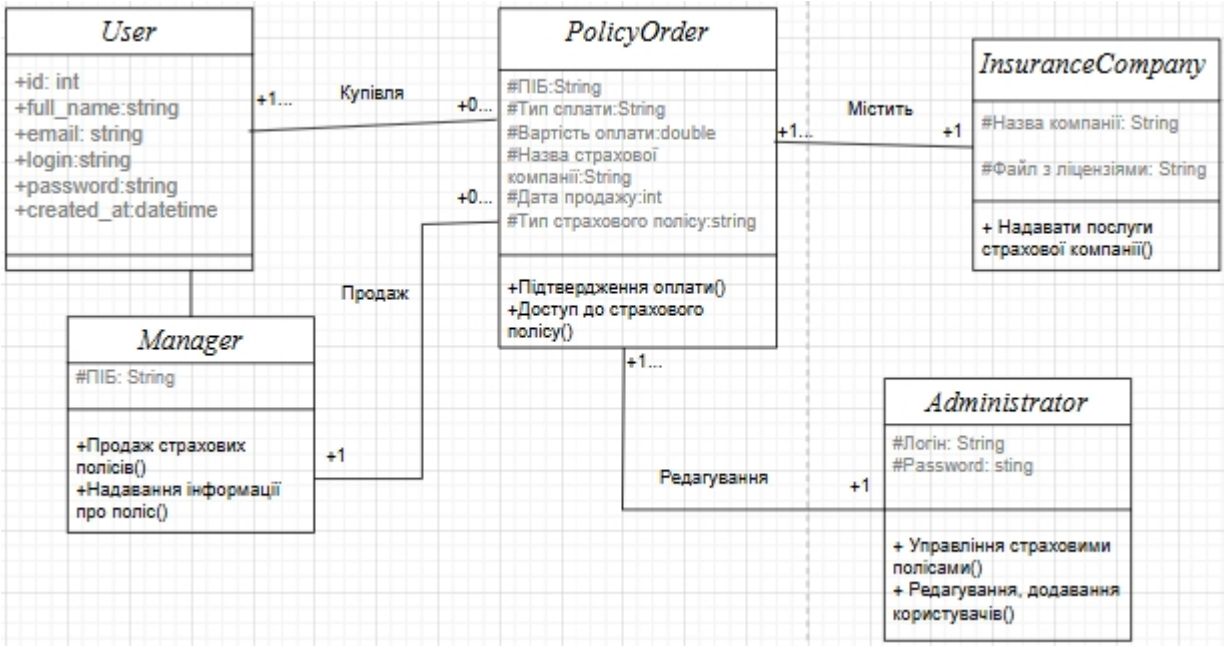

Рис. 10 Діаграма класів.

2.3 Діаграма пакетів

Діаграма пакетів полегшує зрозуміть суть архітектури та функціональності системи та взаємодії системи. За допомогою діаграми пакетів можливо переглянути логічне моделювання, за потребою.

Діаграмою пакетів є діаграма, що містить пакети класів і залежності між ними. Строго кажучи, пакети є елементами діаграми класів, тобто діаграма пакетів - це всього лише діаграма класів. Відрізняються ці діаграми практичним

призначенням і використанням. Залежність між двома елементами має місце в тому разі, якщо зміни у визначенні одного елемента можуть спричинити зміни в іншому.

Що стосується класів, то причини залежностей можуть бути різними:

1) один клас посилає повідомлення іншому;
2) один клас включає частину даних іншого класу;
3) один клас посилається на інший, як на параметр операції.

Якщо клас міняє свій інтерфейс, то повідомлення, яке він посилає, може стати неправильним.[7]

Проєктуючи діаграму пактів було прийнято рішення, використати шаблон архітектури Model-View-ViewModel, що буде зображено на рис.10.

Діаграма пакетів має свої певні переваги та недоліки у використанні. Починаючи з недоліків, діаграма має об’єм занадто велику кількість абстракцій, може не показувати взаємодію та приховує складність в межах модулів.

З переваг, діаграму можна виділити, як діаграма пакетів великих кількість класів по модулям, має приємний, але не завжди зрозуміли архітектурний вигляд проєкту та розділення системи.

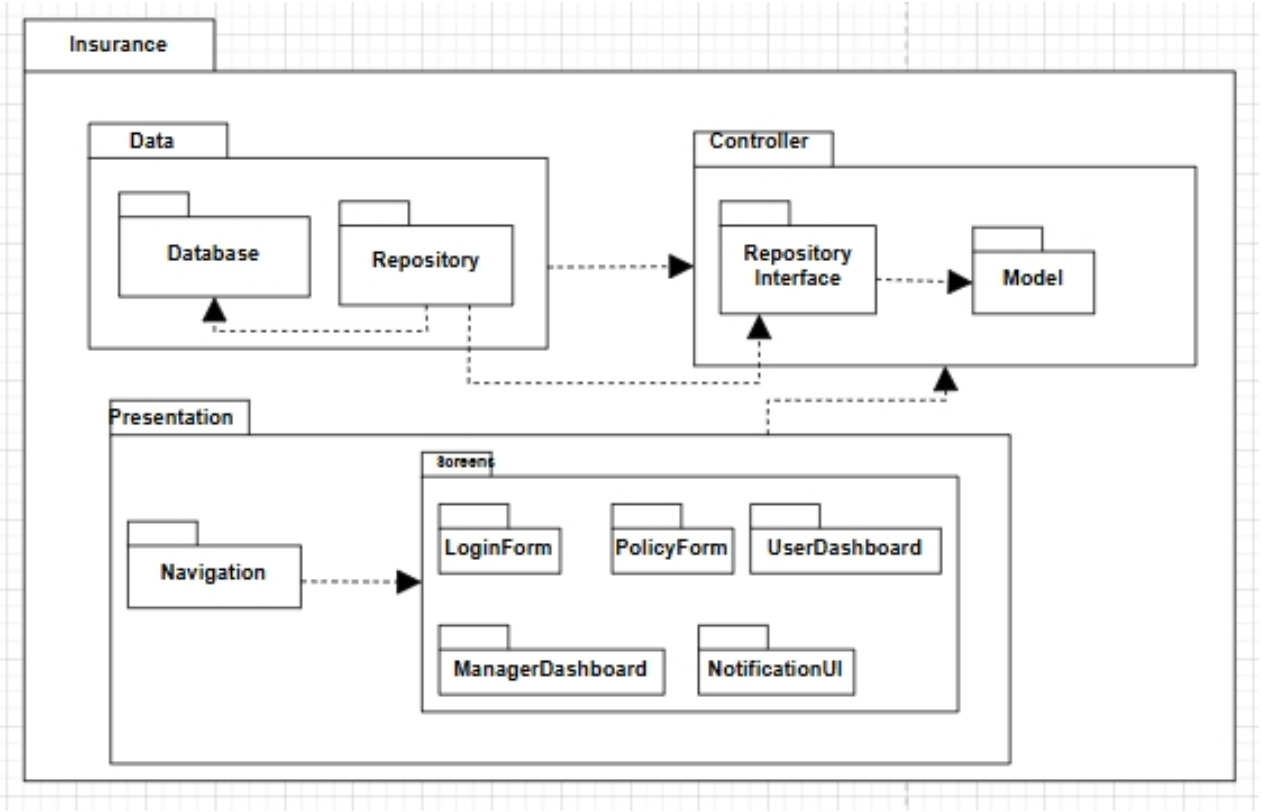

Рис. 11 Діаграма пакетів

2.4 Діаграма компонентів

Для подальшої розробки, наступним етапом використовується моделювання структури системи, а саме Діаграма компонентів. Розроблювальна система має чіткий план та структуру, що надає можливість змоделювати діаграму компонентів, що буде мати залежності, артефакти. Маючи інформацію про структуру програмного забезпечення за допомогою Діаграми компонентів – це нам надає сутність та логічність моделі.

Одна з UML – нотатків, а саме діаграма компонентів, використовує відображення архітектури ПЗ з точки зору основуючи окремих модулів та їх взаємодій у системі. Певні види діаграм надають візуалізацію структури, складової системи між модулями та формуючи логічну модель.

Ключовою структурою діаграмою: компонентів є компоненти, що являють собою окрему частину системи та показують функціональність, інтерфейс, що

надають визначення функцій, що мають доступ до інших програм, залежності мають лише один елемент, що може використовувати один елемент або має потребу в іншому та останнє – артефакти. Артефакти мають значення в діаграмі компонентів значення фізичних та наявних файлів у системі, файли які виконуються та бібліотеки, що мають забезпечення стабільності роботи системи.

За допомогою цієї діаграми, нам надає полегшення та розуміння структури, що в межах розумного дозволяють полегшати загальне уявлення системи, надаючи можливість логічно осмислити рівні зв’язаності та виявляти можливі проблеми архітектури програми.

На рис. 12 представлено діаграму компонентів для програмного забезпечення інформаційної системи автострахування.

Центральним елементом діаграми компонентів інформаційної системи автострахування є файл який виконується (.exe), що має інтеграцію наступних складових системи. Реалізуючи наступним структуру модулів та функцій між ними, що показано в діаграмі компонентів.

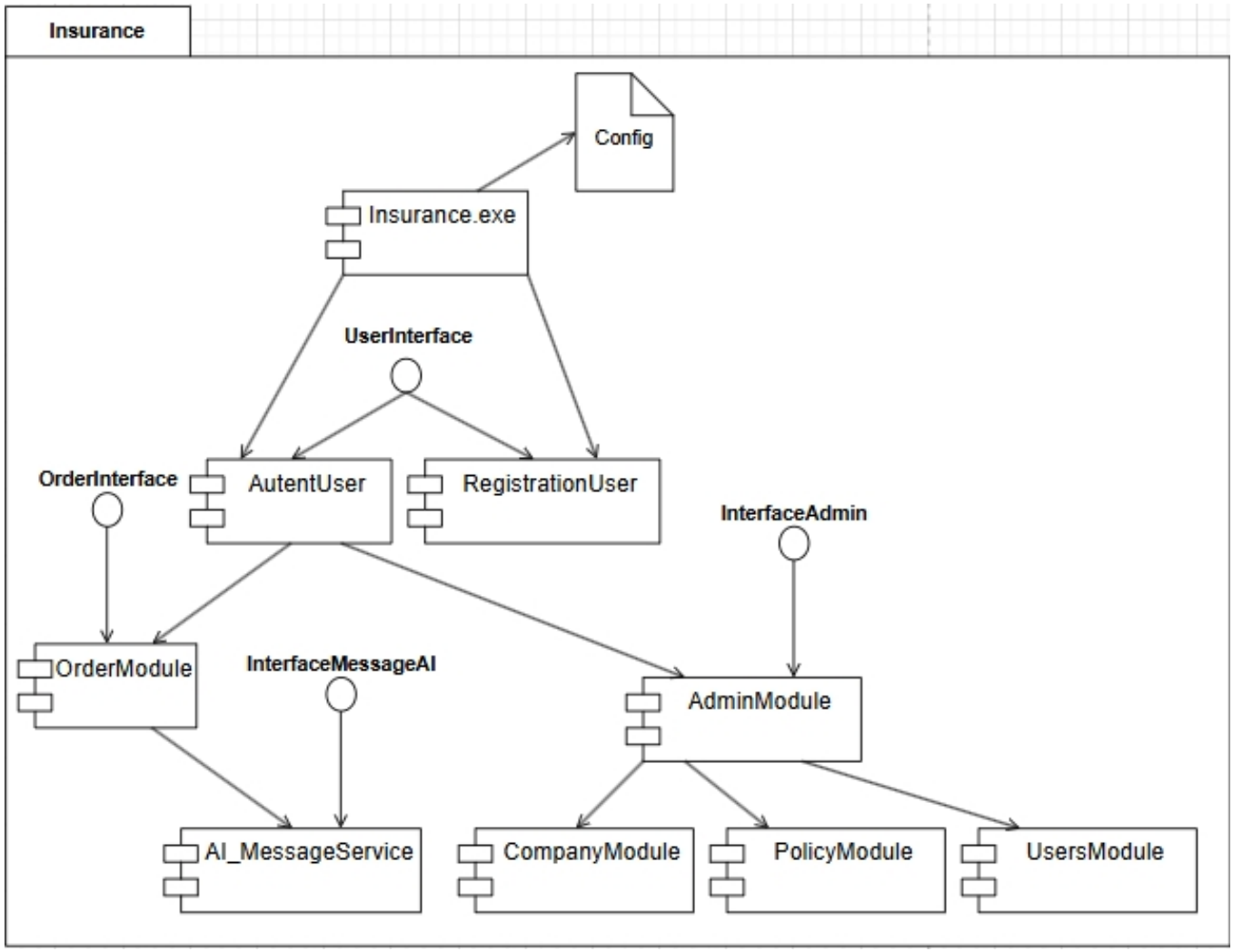

Рис. 12 Діаграма компонентів

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕПЕЧЕННЯ

3.1 Система управління інформаційною базою

Системою управління інформацію базою є один з головних виборів та аналізів із існуючих рішень управління базою даних. Реляційна база даних є одним із ключових систем завдяки яким надається можливість зберігати дані. Завдяки реляційній базі даних з’являється можливість мати конкретну структуру та зручно зберігати дані запису, ефективно мати взаємозв’язані дані, маючи реалізацію використання запитів до бази даних, як в самій системі СУБД та і використовуючи до них з програм, які під’єднанні до реляційної бази даних.

Перевагою стане такі компоненти:

1) Структуризація за допомогою таблиць у базі даних;
2) Зміни у структурі бази даних не впливають на збережений інформації у таблиці;
3) Цілісність даних маючи первинні та зовнішні ключі.

Для розробки структури бази даних є одна з найважливіших компонентів – це логічний взаємозв'язок, який зв'язує елементи, маючи зовнішні ключі, задаючи безпеку обробки та створення запитів, маючи їх коректно сформурованими.

Для розробки програмного забезпечення в якості бази даних, обрано PostgreSQL, маючи велку перевегу серед переглянутих варіантів СУБД.

PostgreSQL – це вільна і відкрита система керування базами даних (СУБД) з відкритим вихідним кодом. Є однією з найпопулярніших СУБД, тому її використовують тисячі організацій по всьому світу.

PostgreSQL базується на мові SQL (освоїти на практиці SQL ви можете на курсі QA Automation від компанії FoxmindED).

PostgreSQL належить до наступного типу СУБД – об’єктно-реляційного (тобто поєднує в собі переваги реляційних баз даних і об’єктно-орієнтованого програмування).

PostgreSQL підтримує широкий спектр функцій, які роблять її потужною і гнучкою СУБД. Ось деякі з її ключових особливостей:

1) PostgreSQL забезпечує транзакції з ACID-властивостями, що гарантує цілісність даних;

2) Підтримує автоматично оновлювані подання, що забезпечує можливість користувачам переглядати дані в базі даних у режимі реального часу;

3) Підтримує матеріалізовані подання, які зберігають копію даних подання в базі даних;

4) Підтримує тригери, які дають змогу виконувати дії під час зміни даних у таблиці;

5) Підтримує зовнішні ключі для створення зв’язку між таблицями;

6) Підтримує збережені процедури для виконання складних обчислень на стороні сервера;[5]

Також, зауважаем, що певні СУБД мають свої недоліки. У випадку СУБД PostgreSQL – це:

1) Назва Postgres мала проблему з отримання своєї назви СУБД, тому, що не могла належить тільки одній компнаії;

2) Акцентуючи на швидкість обробки та порівнючи з MySQL, Postgres потребує більше об’єму роботу;

3) MySQL має більше продуктивності, порівнянно з PostgreSQL.

3.2 Розробка інформаційної бази

Створення бази даних

На початку створення бази даних, потрібно знати мову SQL, маючи змогу надалі використовувати мову для написання та створення об’єктів, запитів тощо.

SQL — це не мова програмування. Правильніше сказати, що це мова запитів до баз даних. Структура запиту сформована в базі, а нам потрібно задати правила — як правильно до неї звернутися. [6]

Початок розробки повинен розпочинатись з створення нової бази даних, що

```
CREATE DATABASE insurance;
```

має в собі зберігати об’єкти запитів, створення таблиць, тощо.

Рис. 13 Код створення бази даних

Створення таблиць.

Аналізовуючи логічну модель згідно пункту розділу 2.1, в логічній моделі наведено існуюче рішення для створення таблиць для бази даних.

Рис. 14 Демонструє один із SQL – кодів для створення таблиці Users в базі даних. На основі логічної моделі з пункту 2.1, було розроблено код для створення таблиць. До ДОДАТКУ Б було рішення навести туди всі інші коди таблиць бази даних.

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    full_name VARCHAR(100) NOT
NULL,
    email VARCHAR(100) UNIQUE
NOT NULL,
    password_hash TEXT NOT
NULL,
    role_id INTEGER NOT NULL
REFERENCES roles(id),
    created_at TIMESTAMP
DEFAULT CURRENT_TIMESTAMP
);
```

Рис. 14 Код створення таблиці Users

Створення процедур

Процес створення процедури в базі даних є одним із головних компонентів, а саме для оптимізації та не надати повторювати певні операції.

Процедура *show_pending_policy_orders*, відображає менеджеру або адміністратору список страхових полісів, що не сплачені. Зі списку обирає лише ті поля [illegible] ідхід реалі[illegible]ь, що мают[illegible]цію в систе[illegible]

```
CREATE OR REPLACE PROCEDURE
show_policies_by_company(p_company_id INTEGER)
LANGUAGE plpgsql
AS $$
BEGIN
   RAISE NOTICE 'Список полісів компанії ID = %:', p_company_id;

   PERFORM
      id, policy_name, description, price, franchise_required, created_at
   FROM
      insurance_policies
   WHERE
      company_id = p_company_id
   ORDER BY
      created_at DESC;
END;
$$;
```

```
        created_at DESC;
END;
$$;
```

Рис. 15 Код створення процедури *show_pending_policy_orders*

Наступна процедура *show_policies_by_company*, що зображено на рис. 15. Демонструючи відображення страхових полісів, що були створені страховою компанією. Адмністратор або менеджер мають змогу перевірити, що за страхові пропозиції пропонує компанія та сформувати звіт основуючись на процедуру. Передається параметр компанії та реалізовано фільтр всіх полісах витягуючи дані з таблиці insurance_policies, що належать цій компанії.

Рис. 16 Код створення процедури *show_policies_by_company*

```
CREATE TRIGGER trg_notify_manager_on_order
AFTER INSERT ON policy_orders
FOR EACH ROW
EXECUTE FUNCTION notify_manager_on_new_order();
```

Створення тригерів.

Наступним та вирішенням завдання є створення тригера. Об’єкти, які знаходяться в базі даних, маючи умови, виконують їх автоматично та тригер прив’язується до якоїсь конкретної таблиці, що дає, якраз виконувати їх автоматично. Рис. 17 та Рис. 18 будуть представлені функція тригера та сам тригер. Розв’язуючи поступово, тригер дає автоматично створити повідомлення менеджеру, а саме повідомлення коли користувач оформив замовлення, що дозволяє оперативно дізнатися про нове замовлення не перевіряючи таблицю замовлень.

Рис. 17 Код створення тригера *trg_notify_manager_on_order*

На рис. 18 представлено функцію тригера *notify_manager_on_new_order*.

```
CREATE OR REPLACE FUNCTION notify_manager_on_new_order()
RETURNS TRIGGER AS $$
DECLARE
   manager_id INTEGER;
BEGIN
   SELECT m.user_id INTO manager_id
   FROM insurance_policies p
   JOIN managers m ON m.company_id = p.company_id
   WHERE p.id = NEW.policy_id;


INSERT INTO manager_notifications(manager_id, order_id,
message)
   VALUES (
      manager_id,
      NEW.id,
      'Нове замовлення полісу очікує обробки.'
   );

   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION notify_manager_on_new_order()
RETURNS TRIGGER AS $$
DECLARE
   manager_id INTEGER;
BEGIN
   SELECT m.user_id INTO manager_id
   FROM insurance_policies p
   JOIN managers m ON m.company_id = p.company_id
   WHERE p.id = NEW.policy_id;


   INSERT INTO manager_notifications(manager_id, order_id,
message)
   VALUES (
      manager_id,
      NEW.id,
      'Нове замовлення полісу очікує обробки.'
   );

   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Рис. 18 Код представлення тригеру *notify_manager_on_new_order*

Створення користувачів.

Система повинна мати систему безпеки, а саме маючи роль. Це означає, що Кожен користувач має мати свою категорію, свою безпеку. Попри все – це безпека, надаючи йому право на зміну записів даних в таблиці. Створючи нового користувача *user_insurance,* що має свої обмеження. З дозволенного, користувача зможе: додати, обновити та видалити.

```
CREATE USER user_insurance WITH PASSWORD 'securepassword123';

-- Надаємо йому права на вставку, оновлення та видалення даних у таблиці policy_orders
GRANT INSERT, UPDATE, DELETE ON TABLE policy_orders TO user_insurance;
```

Рис. 19 Код для створення користувача

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Обраним інструментарієм для розробки інтерфейсу користувача програмного забезпечення для автоматизованої системи автострахування обрано було *Windows Forms*.

Перевагою для розробки інтерфейсу стало немало з причин, а саме:

1) Зручність у використанні та розробки – *Forms* дає можливість швидко реалізувати інтерфейс та його функціонал системи. Також маючи велику кількість елементів завдяки чому можна все більше реалізовувати функцій;

2) *Visual Studio* має інтеграцію з *Windows Forms*, що дає змогу підтримки середовища *Visual Studio*, зручність невелика, а саме налагодження та тестування;

3) *Windows forms* дає можливість реалізувати та наталаштувати властивості елементі, що знаходяться на панелі елементів.

Мовою для програмування стала *C#,* що є зручною мовою для реалізації декстопних застосунків.

Перевага мови C#:

1) *C#* використовує наслідування об’єктів, що робить її потужною мовою, судячи з того, що це робить з цього об’єктно-орієнтований підхід до проєкту;
2) Використовуючи C# нам надається змога використовувати бібліотеки на платформі .Net, що збільшує кількість бібліотек для роботи з системою;
3) Однією із стабільних мов – є *C#* тому, що присутнє керування пам’яттю, для уникнення помилок з пошкодженням даних .

Середовищем розроброки стало Microsoft Visual Studio, що також має власні перспективи, а саме:

1) *Visual Studio* має змогу підтримувати не тільки реалізацію *C#*, *Windows Forms*, а ще багато іншого;
2) Зручне управління та з’єднання з базами даних, а саме з СУБД, що дає змогу це робити одразу в середовищі, та опрацьовувати дані, або їх змінювати;
3) IDE дає розширені функції та можливості з пошуком помилок, аналізу та тестування.

3.4 Алгоритмізація та програмування програмних модулів

Алгоритмізація та програмування програмних модулів залежить від побудованих алгоритму, щоб розуміти та бачити основні процеси блок-схеми. Завдяки побудуванню алгоритму, можливо виявити певні помилки та зрозуміти та пообдувати для себе розуміння та бачення дійсності та побудови алгоритму.

Рис. 20 демонструє алгоритми системи реєстрації, що надає інформацію про виконання пунктів програми, а саме користувач вводить прізвище ім’я та по-батькові, наступним чином введення пункту пропису email, пароль для майбутньої авторизації і останнім пунктом буде login для того, щоб ідентифікувати

користувача у системі. Також система ідентифікує та перевіряє за логіном та email, для того, щоб в системі не було здвоювання.

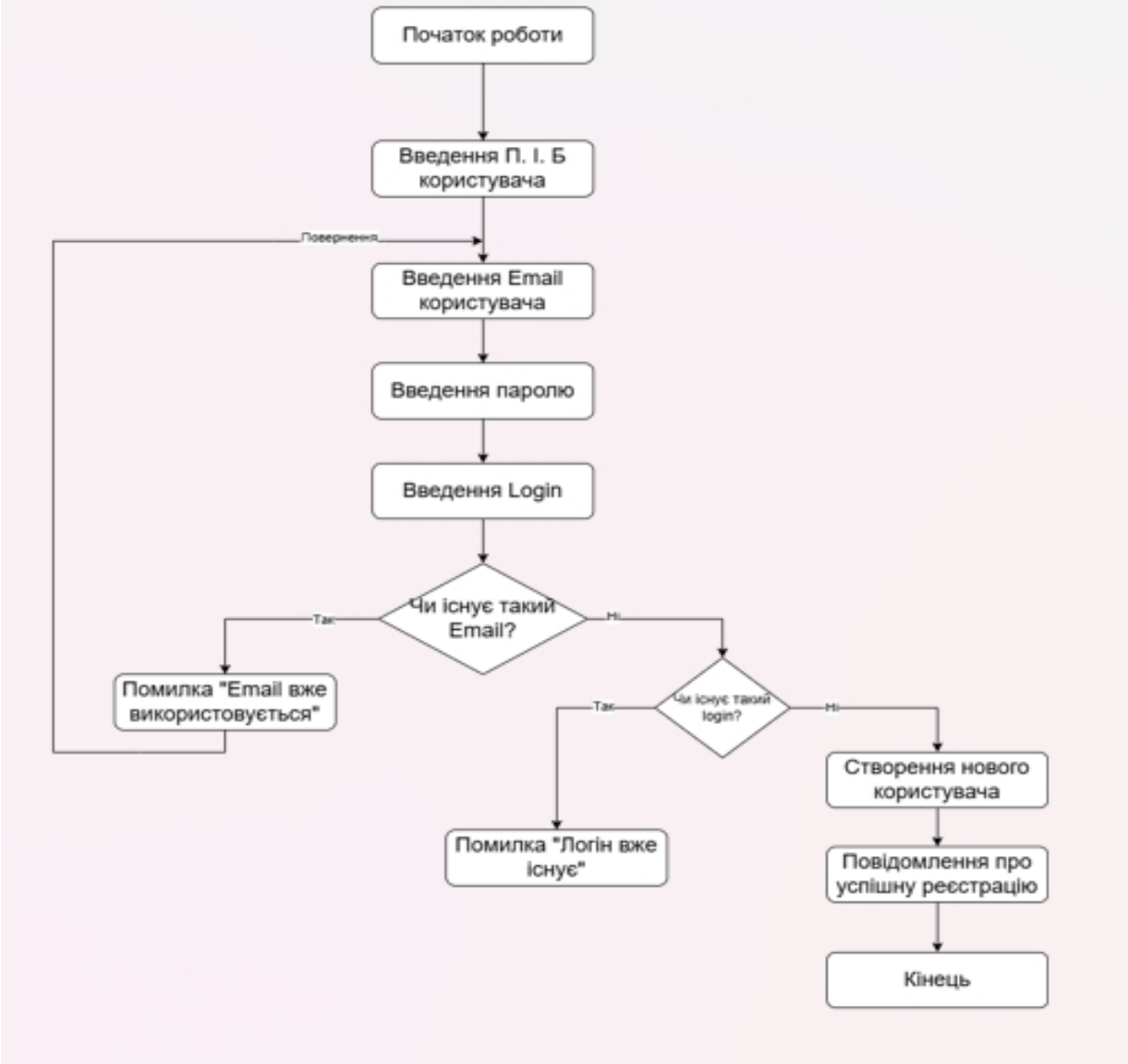

Рис. 20 Алгоритм реєстрації користувача в системі

На рис. 21 зображено частину коду для реєстрації користувача в системі автострахування. Код формує з себе запит на перевірку наявності таких користувачів в системі

```
string checkQuery = "SELECT COUNT(*) FROM users WHERE login = @login OR email = @email";
 using (var checkCmd = new NpgsqlCommand(checkQuery, conn))
 {
    checkCmd.Parameters.AddWithValue("login", login);
    checkCmd.Parameters.AddWithValue("email", email);

    long count = (long)checkCmd.ExecuteScalar();
    if (count > 0)
    {
       MessageBox.Show("Користувач з таким логіном або email вже існує.");
       return;
```

Рис. 21 Зображення реалізації перевірок Email та login в системі.

Наступним прикладом буде реалізація алгоритму оформлення замовлення користувачем. Основним етапом цього процесу є швидкість оформлення та одразу формування чеку в форматі PDF. Реалізація цих компонентів є одним із головних задач системи, щоб клієнт оформив в два кліки собі страховий поліс та забезпечив себе страховим полісом надаючи безпеку своїй автівці. Алгоритм буде зображений на рис. 22 де поступово пояснюється дії користувача під час оформлення страхового полісу.

Рис. 22. Дії користувача під час оформлення страхового полісу.

```
string vinCode = Microsoft.VisualBasic.Interaction.InputBox("Введіть VIN-код авто (17 символів):", "VIN-код", "");
if (string.IsNullOrWhiteSpace(vinCode) || vinCode.Length != 17)
{
    MessageBox.Show("VIN-код має містити 17 символів.");
    return;
}
string cardNumber = Microsoft.VisualBasic.Interaction.InputBox("Введіть номер картки (16 цифр):", "Оплата", "");
if (cardNumber.Length != 16 || !long.TryParse(cardNumber, out _))
{
    MessageBox.Show("Некоректний номер картки.");
    return;
}
string expiry = Microsoft.VisualBasic.Interaction.InputBox("Введіть термін дії (MM/YY):", "Оплата", "");
if (!System.Text.RegularExpressions.Regex.IsMatch(expiry, @"^(0[1-9]|1[0-2])\/\d{2}$"))
{
    MessageBox.Show("Невірний формат дати.");
    return;
}
string cvv = Microsoft.VisualBasic.Interaction.InputBox("Введіть CVV (3 цифри):", "Оплата", "");
if (cvv.Length != 3 || !int.TryParse(cvv, out _))
{
    MessageBox.Show("Невірний CVV.");
    return;
}
        string pdfPath = PdfGenerator.GenerateInsurancePdf(
            username: $"User#{_userId}",
            carModel: "Не вказано",
            vin: vinCode,
            insuredSum: price,
            payment: franchiseAmount,
            insuranceDate: DateTime.Now
        );
```

Рис. 23 Частина коду функціональності оформлення страхового полісу.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇСИСТЕМИ

4.1 Тестування системи

Для подальшої розробки та функціональності системи потрібно робити тестування системи. Це є обов’язковим етапом під час розробки програмного забезпечення. Проведення тестування системи надає переглянути систему з іншої сторони, виявити некоректність системи та якість готовності до використання та відповідаючи усім вимогам стандарту розробки програмного забезпечення.

Основою метою перевірки та проведення тестування є функціональність, надаючи перспективи надалі виявити помилки для подальшого усунення їх. Розуміючи використання та налаштування їх під користувача, підбиваючи усі висунені завдання та осмислюючи логічну модель даних.

Наступним етапом – є перевірка навантаженої системи, що дає розуміння та проявлення ситуації при завантаженості системи, як себе буде проявляти функціональність, та коректність роботи.

Основуючись етапом проведення функціональності, можливе виявлення помилок, що дає виявити та усунути проблеми некоректності системи.

Після виявлення усіх помилок та недостатньої функціональності, можливе проведення етапу регресійного тестування, що надає змогу оглянути та переконатись усуненню усіх поточних проблем системи, що може спричинити проблеми у використанні для користувачів. Але нажаль, усуненню проблем може з’явитись і наступні недоліки, що є звичайною послідовністю та виникненню недоліків системи. Саме тому, регресійне тестування є також одним із важливих компонентів тестування під час розробки ПЗ.

Кожна система повинна мати автоматизацію процесу, що дає змогу більше збільшити типових дій при виконанні дій в системі, наприклад: формування звіту

або його заповнення, виконуючи обчислення, авторизація тощо. Етапом автоматизації дозволяє не лише економити час, а і зменшує кількість помилок людського фактору в системі.

Переконуючись коректності за допомогою всіх минулих етапів не достатньо з приводу того, що потрібно розроблювати також перевірку за напрямки юніт-тестів, даючи змогу виконувати перевірку системи функціональності або класів також. Кожна перевірка має змогу перевірити та дає переконатися у коректних частинах коду, що буде реалізовано для програмного забезпечення інформаційної системи автострахування.

Задумане на початкових етапів розробки має пройти перевірку тестування на верифікацію та валідацію системи, даючи змогу перевірити всі вимоги, що вкладені у документацію проєкту. Перевірка передбачає з себе те, що було задумане від реалізованого програмного продукту, що в дійсності не зможе відбутися у реальному житті.

Проходячи етапи перевірки тестування системи, потрібно перевірити та протестувати систему на її безпеку, маючи змогу захищати правові та юридичні дані користувача. Безпека – це одна із важливих аспектів системи, програмний додаток має забезпечувати цілісність та безпеку даних користувачів. Це тестування спрямоване на виявлення та усунення усіх вразливих місць, передачі даних для використання зловмисником у свою користь.

Для повної реалізації та використання продукту з використанням реальних даних, дає можливість вбудовувати її у реальний світ та налаштовувати її під ці параметри. В таких ситуаціях перевірки, тестування дає можливість знаходити помилки, що були б непомічені при використанні штучних або тестових даних.

Тестування інтерфейсу для потенційного користувача системи створює перевірку існуючих елементів керування, що повинні мати логічне розташування

на панелі додатку та виявляти потенційні незручності системи елементів управління.

Завершальним етапом тестування є сумісність для інших систем через, які може користуватись користувач, це може бути, як браузер або операційна система. Розроблювальний та підготовлений програмний продукт повинен мати сумісність на інших системах тому, що це актуальність сьогодення.

Для прикладу програмного забезпечення буде наведений код для інформаційної системи автострахування, що буде представлено юніт-тест для на рис. 24.

Метою буде реалізація переконатись відповідності хешування паролю при авторизації. Для реалізації потрібно зробити повну зборку проєкту, що надалі дає можливість за допомогою Visual Studio створити xUnit для реалізації юніт-тесту в проекті. Його можна поділити на два етапи:

1) *VerifyPassword_ReturnsTrue_WhenPasswordMatches* – мета, перевірити *BCrypto.Verify*, що буде повертати значення true для передачі паролю, що відповідає хешуванню де *HashPassword(password)* створює хеш пароля, *Verify("test123", hashed)* перевіряє чи підходить пароль та результат *Assert.True(result)* очікування на успішний результат перевірки;

2) *VerifyPassword_ReturnsFalse_WhenPasswordDoesNotMatch* – етап, який перевіряє чи не підходить пароль хешуванню. Створюється хеш для пароля "*correctPassword*", "*wrongPassword*" – перевіряє пароль, що не співпадає і далі *Assert.False(result)* – тест проходить якщо значення передбачає false;

```
public class UnitTest1
{
    [Fact]
public void VerifyPassword_ReturnsTrue_WhenPasswordMatches()
    {
        var password = "test123";
var hashed = BCrypt.Net.BCrypt.HashPassword(password);

var result = BCrypt.Net.BCrypt.Verify("test123", hashed);

        Assert.True(result);
    }

    [Fact]
public void VerifyPassword_ReturnsFalse_WhenPasswordDoesNotMatch()
    {
var hashed = BCrypt.Net.BCrypt.HashPassword("correctPassword");

var result = BCrypt.Net.BCrypt.Verify("wrongPassword", hashed);

        Assert.False(result);
    }
}
```

Рис. 24 Юніт тест для методу Authorization

Наступний етап демонстрації та використання тестування користувацького тесту було прийнято рішення демонструвати на рис. 25 компонент авторизації користувача у систему.

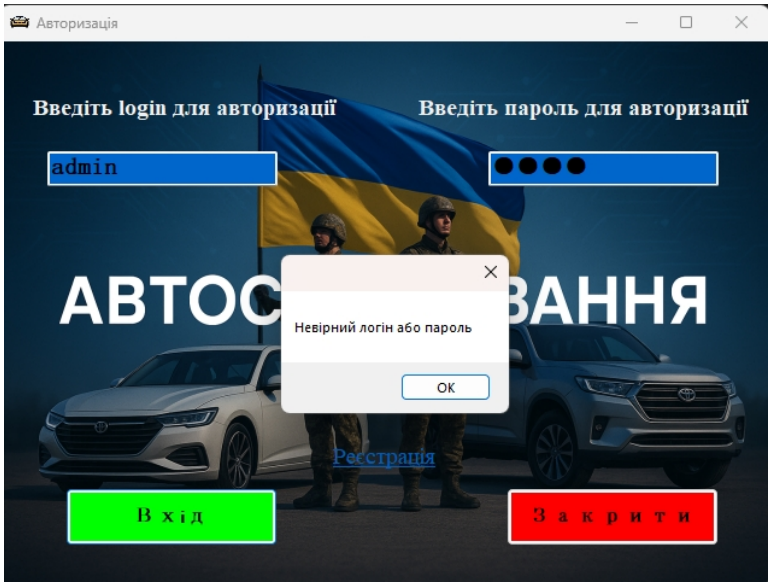

Рис. 25 Вікно авторизації та перевірка на помилкові введення даних.

Натискаючи на текст з посиланням *«Реєстрація»*, відкривається наступне вікно для реєстрації клієнта у систему (рис. 26)

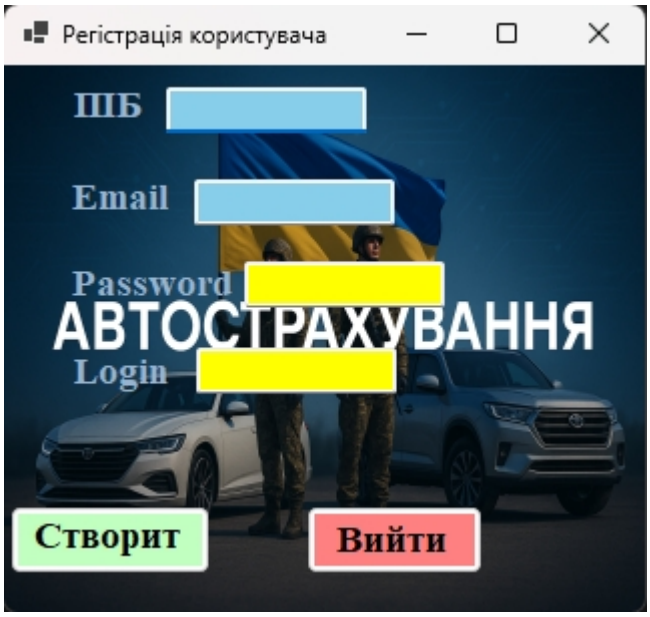

Рис. 26 Вікно реєстрації користувача.

При реєстрації, користувач переходить до вікна авторизації, заповнює логін та пароль його система інформує та запускає на головну сторінку додатку для користувача на якій реалізовано список страхових пропозицій (рис. 27).

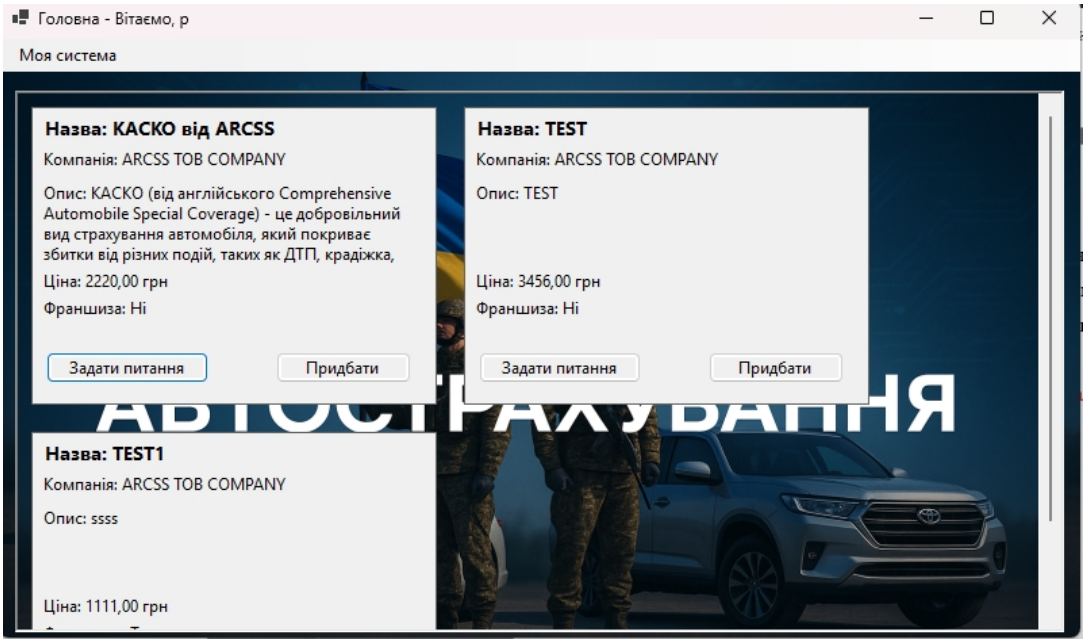

Рис. 27 Головне вікно додатку користувача

На рис. 27 зображено реалізацію списку страхових пропозицій та надає можливість консультації зі штучним інтелектом, який швидко зможе надати відповідь, яка зацікавить користувача при виборі страхового полісу.

Наступним етапом буде демонстрація чату спілкування зі штучним інтелектом, який надає допомогу користувачу (рис. 28).

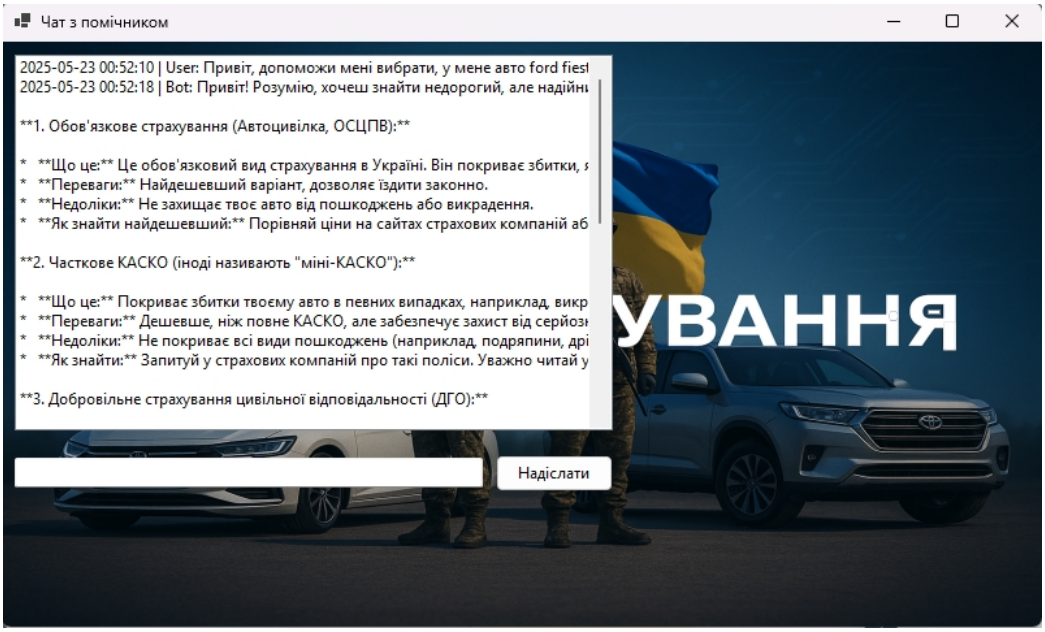

Рис. 28 Вікно спілкування зі штучним інтелектом.

Консультуючись з штучним інтелектом, який надав допомогу при виборі страхового полісу, клієнт повертається назад на головне вікно та обирає потрібний йому страховий поліс. На рис. 29 по рис. 33 будуть представлені демонстрації заповнення даних для сплати та реалізація чеку у форматі PDF.

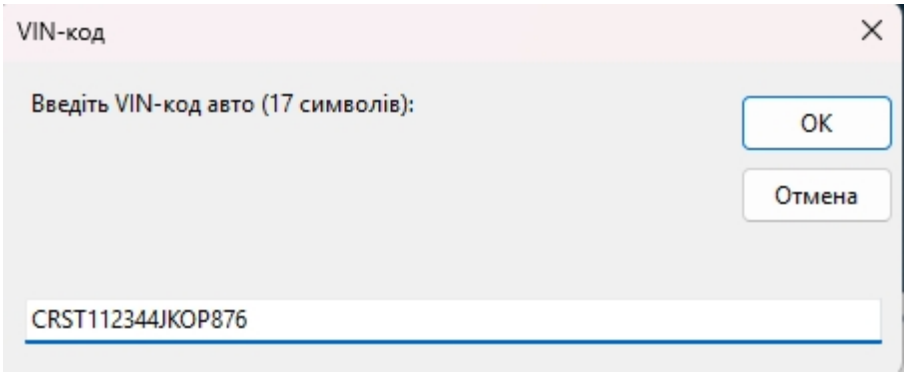

Рис. 29 Введення VIN – коду авто.

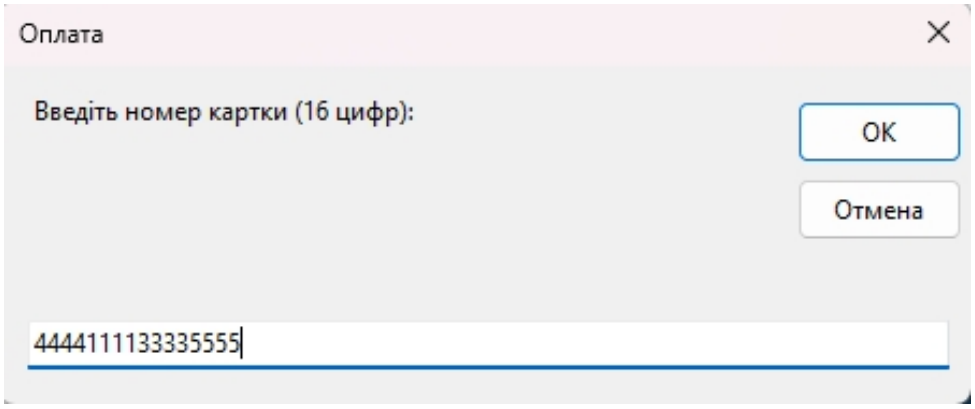

Рис. 30 Заповнення даних карти для сплати.

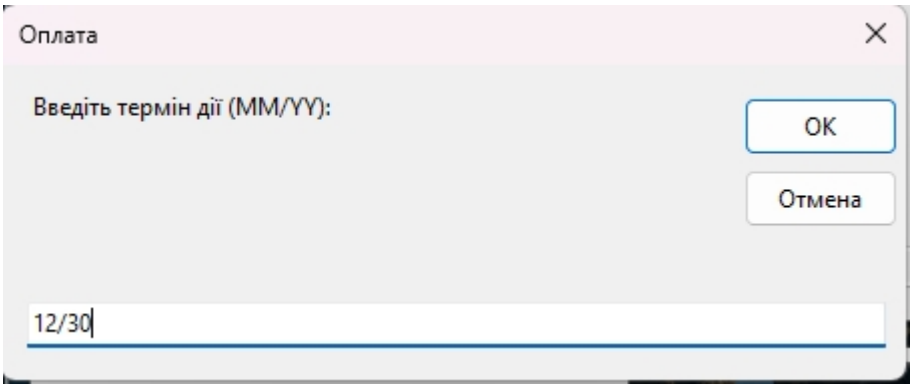

Рис. 31 Заповнення терміну дії карти користувача.

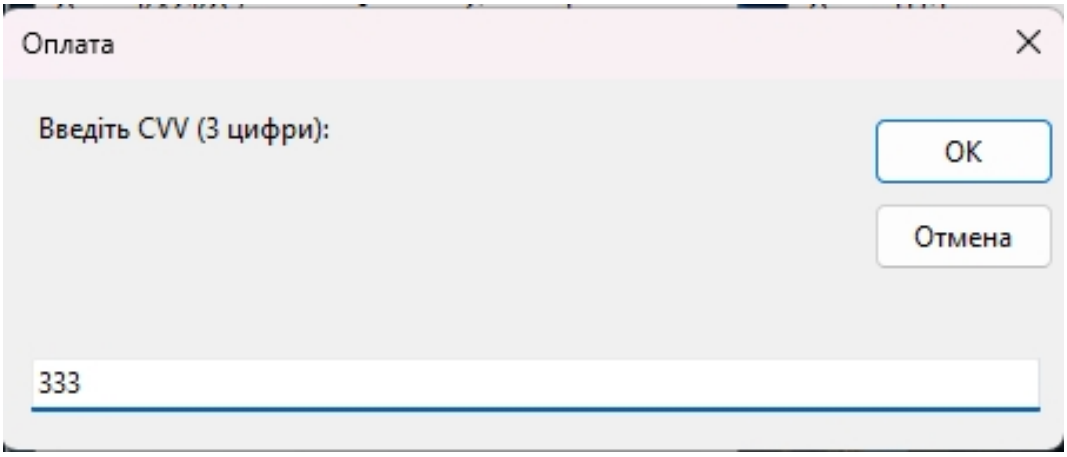

Рис. 32 Заповнення CVV коду від карти.

NAME

Company: ARCSS Insurance Group"
Str: -
Number: +38 (044) 123-45-67

USERS: User#14
VIN CODE: CRST112344JKOP876
CAR MODEL:
InsuredSUM: 2220,00 UAH
PAYMENT: 2220,00 UAH
Date: 23.05.2025

Coating	Summ	Date
CASKO	2220,00 UAH	23.05.2025 - 23.05.2026

Payment all: 0 UAH
Form payment: card
Signature: _____

Рис. 33 Формування PDF файлу чеку про отримання послуги.

За допомогою головного меню користувача, клієнт має змогу перевірити наявність його страхових полісів, що дає перевірити власну історію покупок. Перегляд страхових замовлення представлено на рис. 33

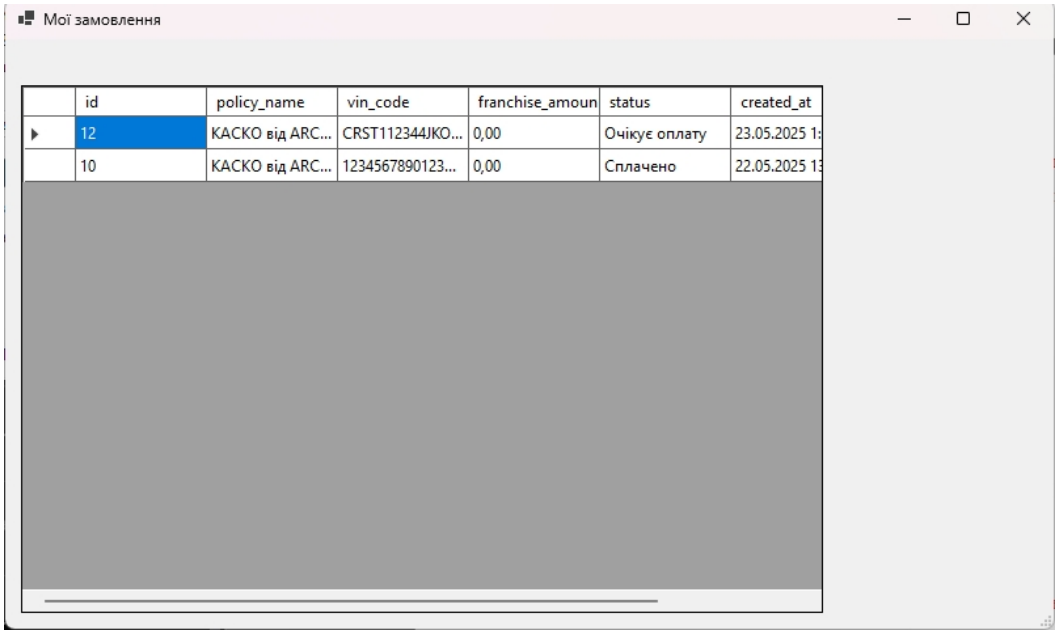

Рис. 34 Перегляд страхових замовлення користувача.

Наступним етапом буде представлення роботи адміністратора та його панелі керування. Рис. 35 має представити головне вікно адміністратора.

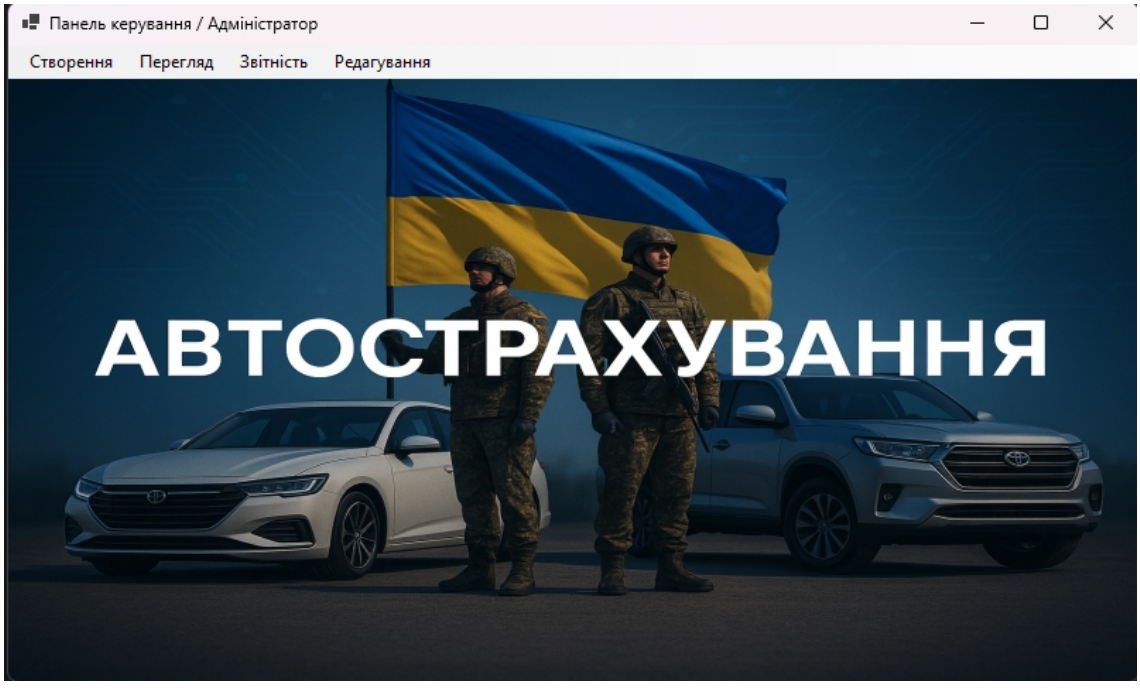

Рис. 35 Головне вікно панелі керування адміністратора.

На панелі відтворено панель керування, а саме: Створення, Перегляд, Звітність, Редагування. На рис. 36 по рис., 38 представлено реалізацію створення страхової компанії, страхового полісу та створення користувача.

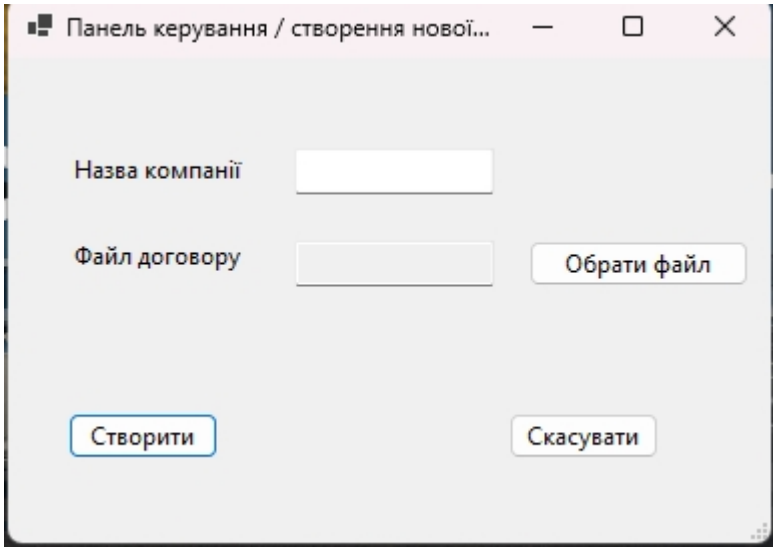

Рис. 36 Створення нової компанії у системі автострахування.

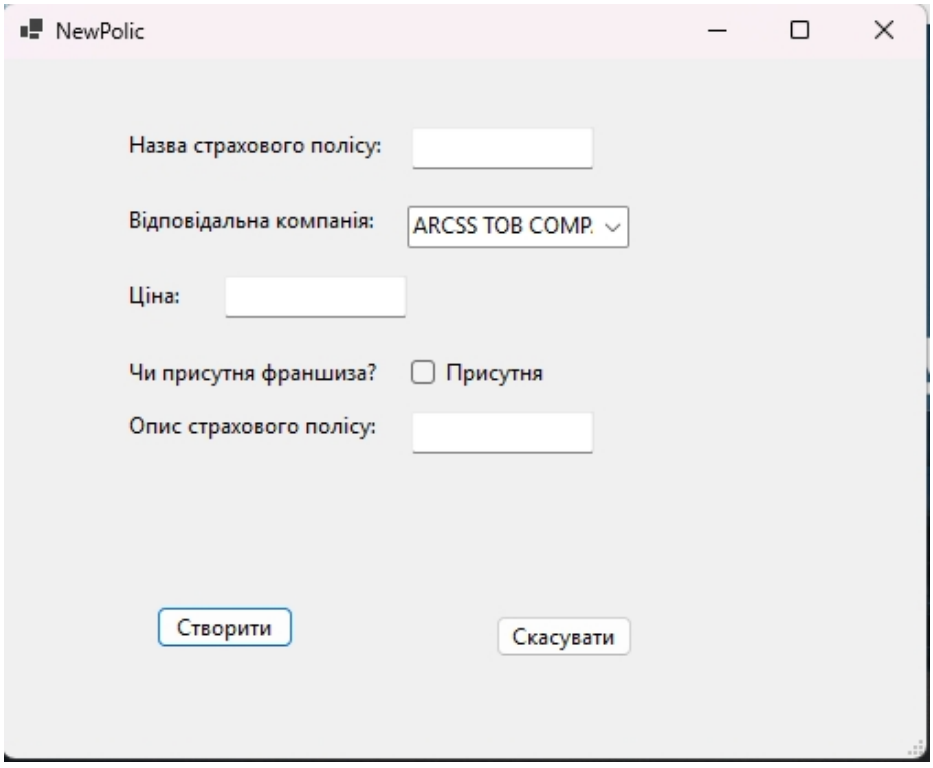

Рис. 37 Створення нового страхового полісу.

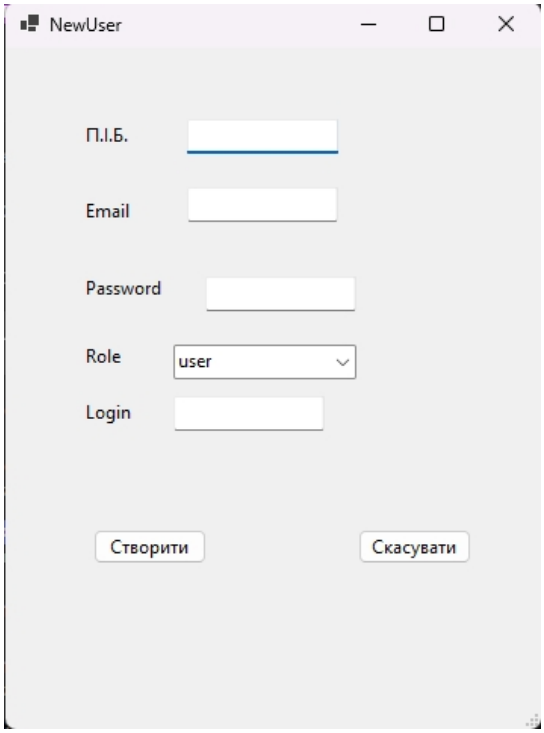

Рис. 38 Створення нового користувача в системі.

Наступну демонстрацію згідно рис. 35, буде представлено перегляд списку опублікованих страхових пропозицій, список користувачів та перегляд компаній. Все це буде представлено на рис. 39 по рис. 41.

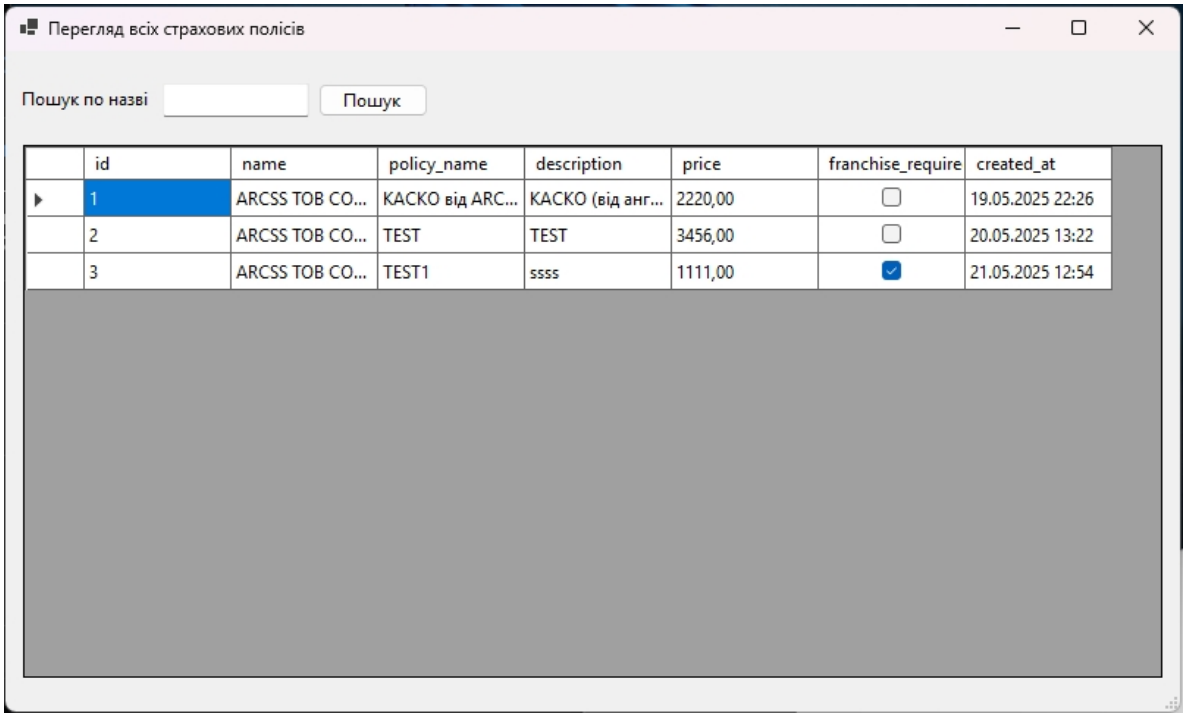

Рис. 39 Перегляд страхових полісів.

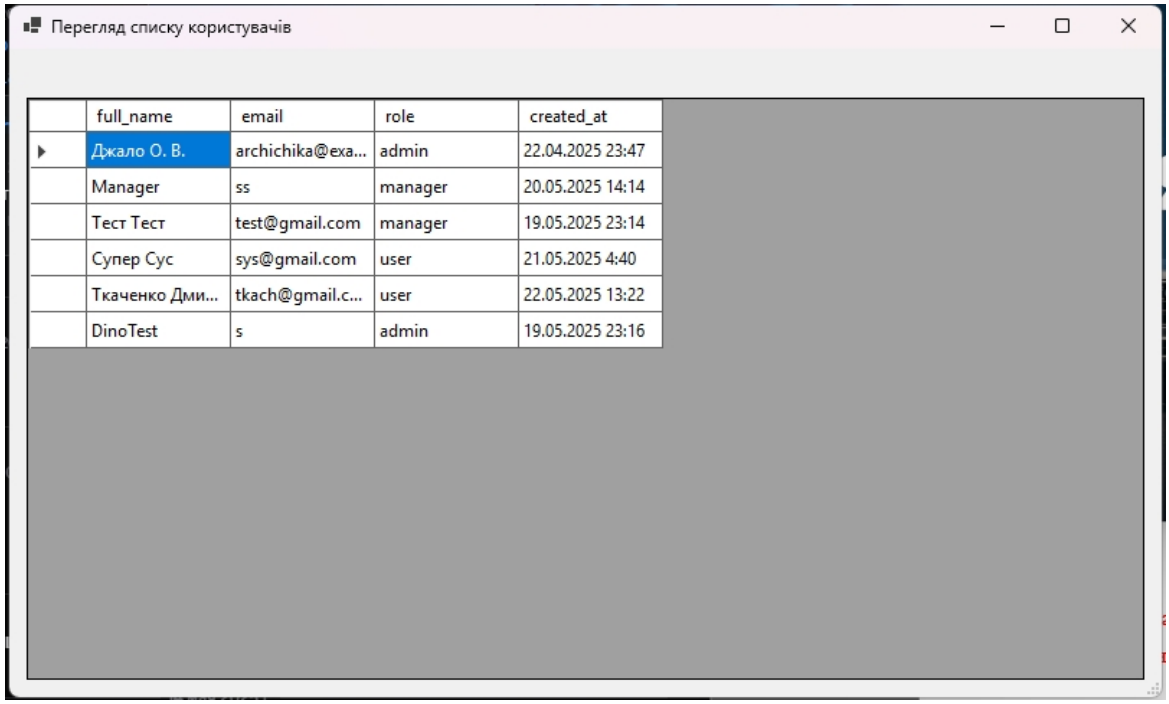

Рис. 40 Перегляд списків користувачів системи.

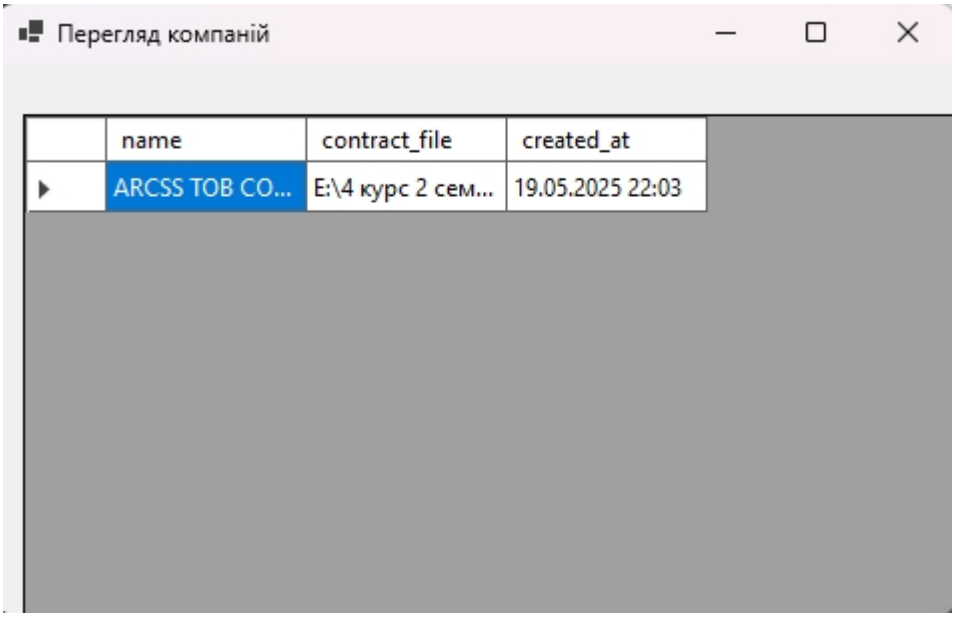

Рис. 41 Перегляд списків компанії.

Наступний пункт демонстрації буде реалізовано демонстрацію звітності за назвою компанії та загальним списком страхових полісів. Де реалізовано кількість проданих та інше. На рис. 42 та рис. 43 буде представлено два вікна звітності.

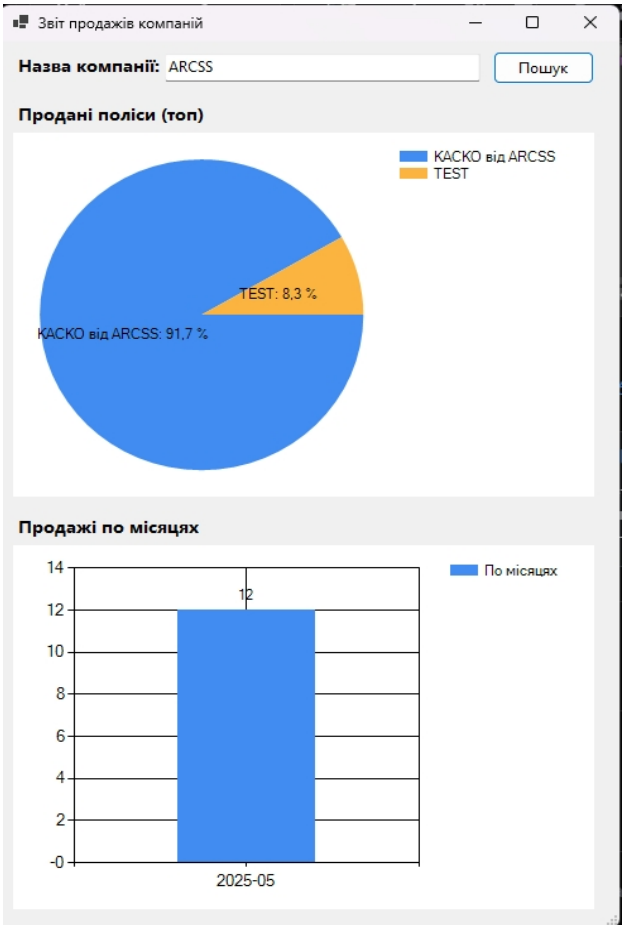

Рис. 42 Вікно звітності за назвою компанії.

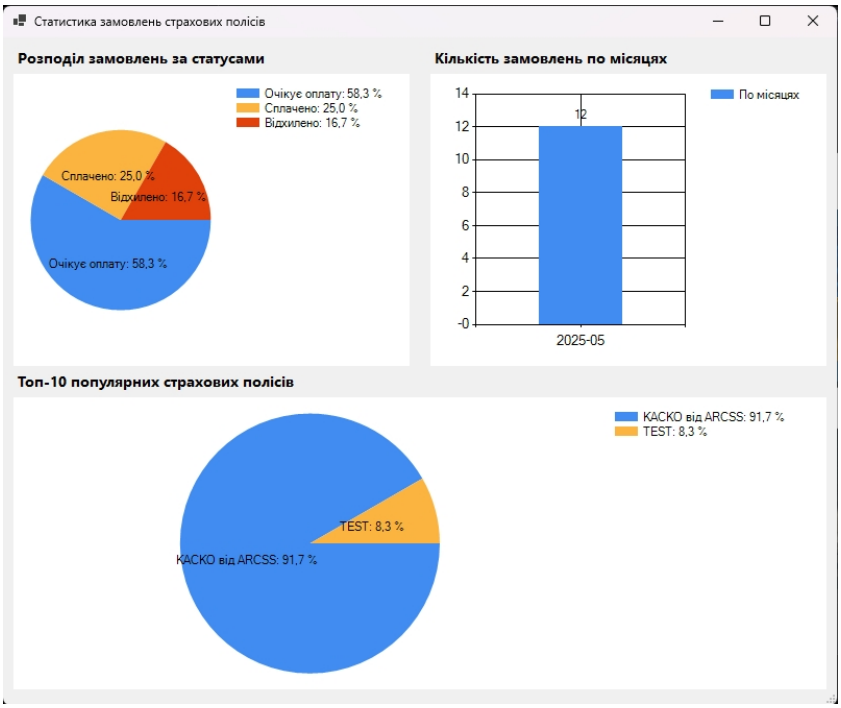

Рис. 43 Вікно звітності загально по замовленням страхових полісів.

Останнім етапом демонстрації панелі адміністратора – це редагування, воно містить в собі редагування користувачів, страхових полісів та інформацію про компанію. Все буде представлено на рис. 44, рис. 45 та рис. 46.

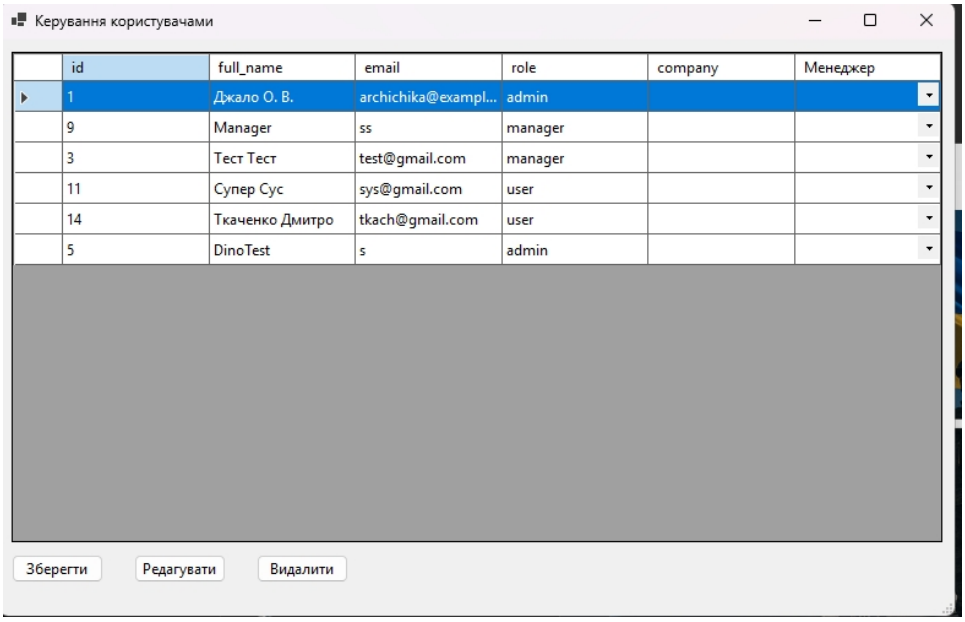

Рис. 44 Вікно керування користувачами.

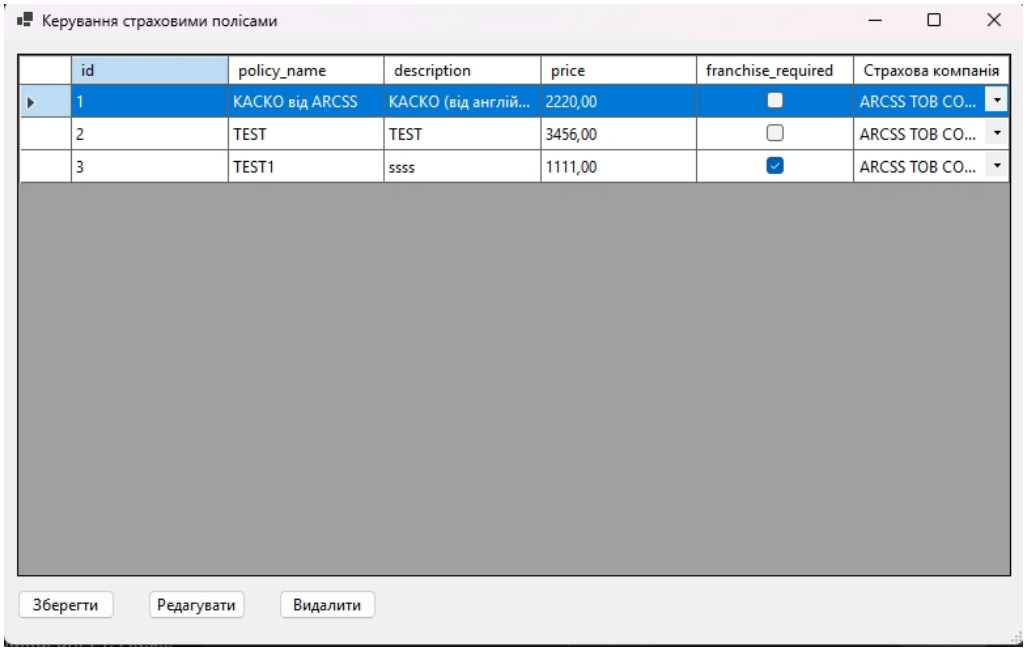

Рис. 45 Вікно керування страховими полісами.

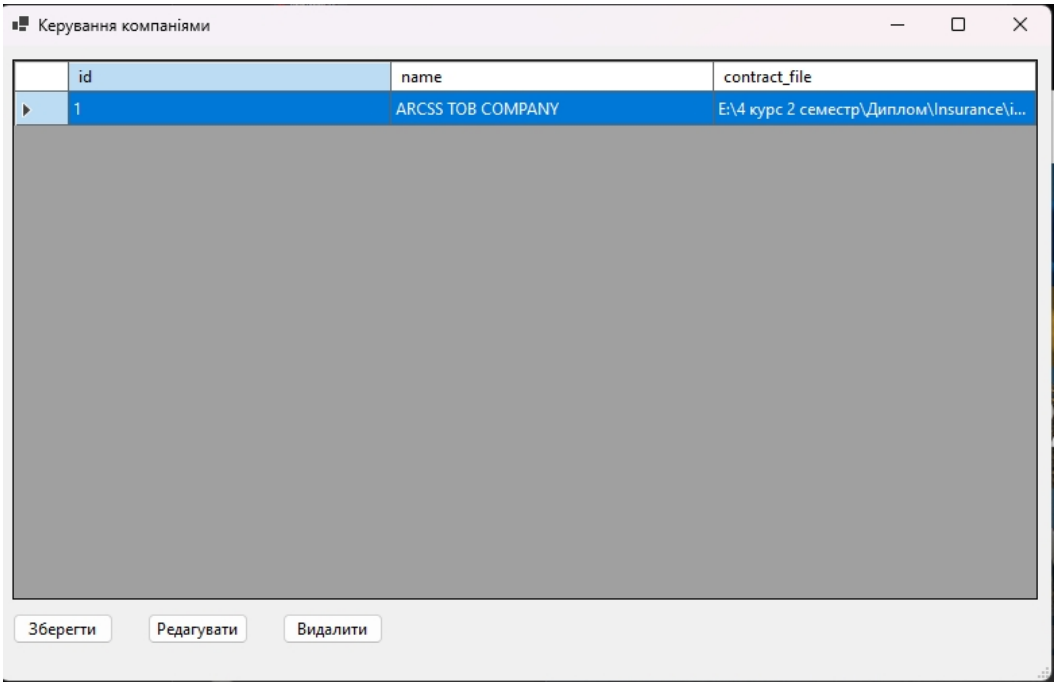

Рис. 46 Вікно керування страховими компаніями.

Останній пункт реалізації є менеджер. Демонстрація буде представлена на рис. 47, рис. 48 та рис. 49, головне вікно менеджера, створення замовлення та перегляд замовлень.

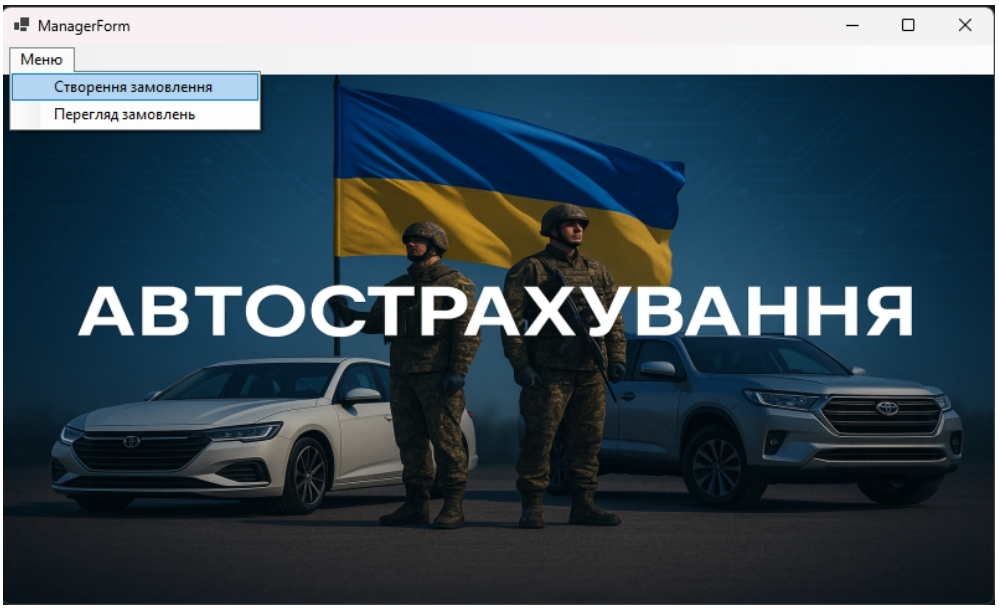

Рис. 47 Головне вікно функціональності менеджера.

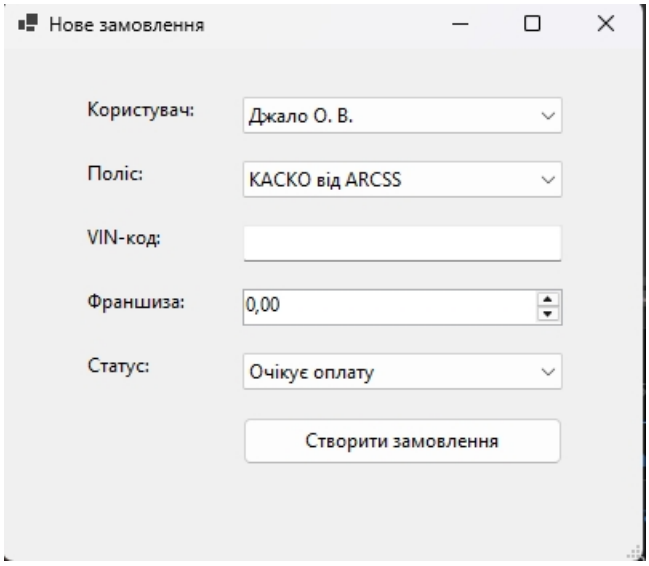

Рис. 48 Вікно створення страхового поліса.

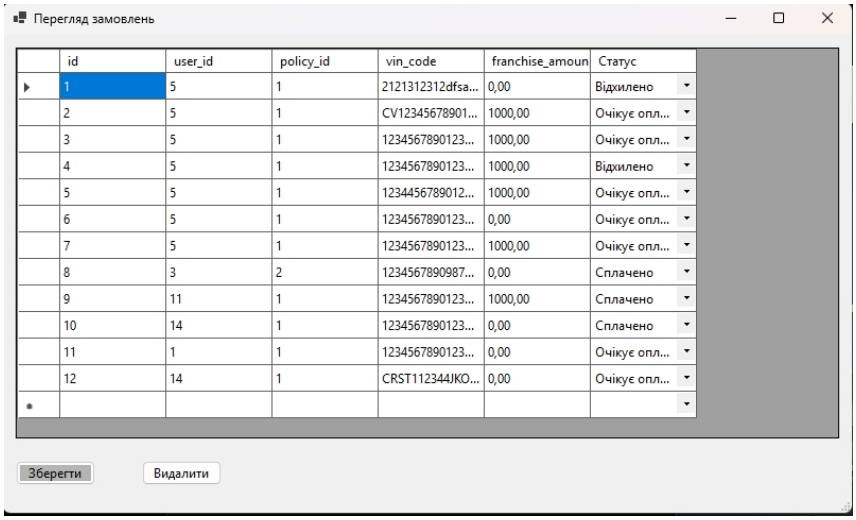

Рис. 49 Вікно перегляду замовлення.

При тестування, було прийнято рішення продемонструвати коректність перевірки при авторизації під час спроби увійти до системи, що було представлено на рис. 25. Згідно всім вимогам система працює коректо та не виявляє ніяких помилок при перевірці функціональності системи.

4.2 Вимоги до апаратного та програмного забезпечення

Програмне забезпечення має мати вимоги для функціювання та виконання усіх процесів системи. Нижче наведено мінімальний перелік вимог для використання:

1) Процесор – Intel Core i3 (двохядерний) або AMD Ryzen 3;
2) Оперативна пам’ять – 4 ГБ;
3) Пам’ять на жорсткому диску – 500 МБ вільного місця та 1 ГБ для бази даних;
4) Монітор – не меньше 15 дюймів, що дає здатність розширити вікно на 1366x768;
5) Мережа – Доступ до локальної мережі або інтернету для автоматизації процесу обміну даних.

4.3 Склад інсталяційного пакету

Склад інсталяційного пакету складає з себе розуміння фізичної реалізації програмного забепечення, проектуючи з цього було прийнято рішення створити діаграму розгортання. Цією діаграмою виходить можливість проєктувати архітектуру програми, а саме графічне представлення.

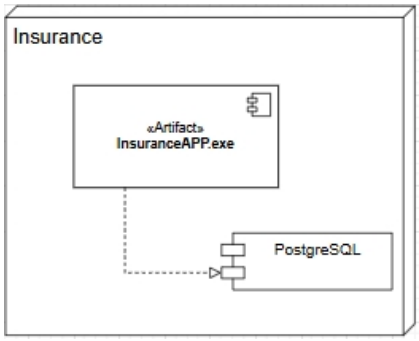

Рис. 50 Діаграма розгортання

Рис. 50 відображає архітектуру програмного забезпечення інформаційної системи автострахування у вигляді діаграми розгортання, що представляє з себе просту архітектуру, що дає зрозуміти склад інсталяційного пакету. Інсталяційний пакет – insuranceAPP.exe

ВИСНОВКИ

Виконання бакалаврської кваліфікаційної роботи на тему «Програмне забезпечення інформаційної система Автострахування», було розглянуто та проаналізовано предметну область автострахування, аналіз існуючих програмних рішень, сформульовано постановку задачі виконання, проектування бази даних, обгрунтовано вибір середовища розробки, реалізація бази даних, інтерфейс, тестування та опис вимог. Починаючи з обраної предметної області, було розглянуто, як об’єкт дослідження, та проаналізовано процеси, що повинні бути автоматизовані, реєстрація, створення страхових полісів, формування PDF-формату чек, авторизація. Однією з головних задач, а також мета – це реалізація швидкого придбання страхового полісу, а також швидка допомога у вигляді спілкування з штучним інтелектом. Сформовано діаграми прецедентів, активності, щоб мати чітке розуміння та логіки програмного забезпечення. Проаналізовано існуючи джерела аналогів системи автострахування, де було розписано переваги та недоліки та детальний опис за кожним з аналогів. Розписано проєктування бази даних та її реалізація у програмному забезпечені, що дає розуміти її значимість та логічне пояснення системи. Реалізація за допомогою середовища розробки Visual Studio та бібліотеки Windows Forms на мові програмування C#, було детально реалізовано інтерфейс та програмну логіку системи, а саме її функціональність. Та було описано вимоги використання програмного забезпечення для використання в повсякденні для наступних клієнтів системи

Результат бакалаврської кваліфікаційної роботи повноцінне робоче програмне забезпечення з використанням автоматизації процесів автострахування, даючи змогу управління базою, генерування страхових полісів, розмежування доступів, відповідаючи вимогам інформаційної безпеки. Розроблена система може бути адаптована під страхові компанії для використання в своїх цілях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Діаграми Прецедентів (Use Case UML Diagram) [Електронний ресурс] – Режим доступу: https://bit.ly/3H46cW9
2) Поняття ER-моделі. Поняття сутності (entity). Атрибути. Види атрибутів [Електронний ресурс] – Режим доступу: https://bit.ly/43pOvrm
3) Поняття сутності, атрибута, ключа, зв’язку [Електронний ресурс] – Режим доступу: https://bit.ly/4jefwUv
4) UML для бізнес-моделювання: для чого потрібні діаграми процесів [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html
5) Що таке PostgreSQL і для чого використовується? [Електронний ресурс] – Режим доступу: https://foxminded.ua/postgresql-shcho-tse/
6) Де використовується SQL і чому він так потрібен програмістам? [Електронний ресурс] – Режим доступу: https://bit.ly/4kviKUy
7) ПРАКТИЧНА РОБОТА №8 ДІАГРАМИ ПАКЕТІВ [Електронний ресурс] – Режим доступу: https://bit.ly/4kSKqmT

Додаток А

SQL-КОД

Сторінок – 3

2025

```
-- 1. Таблиця ролей
CREATE TABLE roles (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE
);

-- 2. Користувачі
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    full_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    role_id INTEGER NOT NULL REFERENCES roles(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 3. Компанії
CREATE TABLE insurance_companies (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    contract_file TEXT, -- шлях до pdf договору
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 4. Менеджери (зв'язок з компаніями)
CREATE TABLE managers (
```

```
    user_id INTEGER PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,
    company_id INTEGER NOT NULL REFERENCES insurance_companies(id)
);

-- 5. Страхові поліси (які створює адміністратор)
CREATE TABLE insurance_policies (
    id SERIAL PRIMARY KEY,
    company_id INTEGER NOT NULL REFERENCES insurance_companies(id) ON DELETE CASCADE,
    policy_name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL,
    franchise_required BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 6. Замовлення страхового полісу (створює користувач)
CREATE TABLE policy_orders (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    policy_id INTEGER NOT NULL REFERENCES insurance_policies(id),
    vin_code VARCHAR(50) NOT NULL,
    franchise_amount DECIMAL(10, 2),
    status VARCHAR(50) DEFAULT 'Очікує оплату', -- або 'Сплачено', 'Підтверджено'
    pdf_file TEXT, -- шлях до згенерованого pdf
```

```
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 7. Повідомлення менеджеру
CREATE TABLE manager_notifications (
    id SERIAL PRIMARY KEY,
    manager_id INTEGER NOT NULL REFERENCES managers(user_id),
    order_id INTEGER NOT NULL REFERENCES policy_orders(id),
    is_read BOOLEAN DEFAULT FALSE,
    message TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 8. Повідомлення клієнту
CREATE TABLE user_notifications (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id),
    message TEXT NOT NULL,
    is_read BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Додаток Б

КОД РЕАЛІЗАЦІЇ ОСНОВНИХ ФУНКЦІЙ

Сторінок – 6

2025

Реалізація функціональності авторизації та розмежування доступу вікон

```
 string login = loginadd.Text.Trim();  // Логін
string password = passadd.Text.Trim();   // Пароль
using (var conn = Database.GetConnection())
{
   conn.Open();
   string query = @"
SELECT u.id, u.password_hash, r.name
FROM users u
JOIN roles r ON u.role_id = r.id
WHERE u.login = @login";
   using (var cmd = new NpgsqlCommand(query, conn))
   {
      cmd.Parameters.AddWithValue("login", login);
      using (var reader = cmd.ExecuteReader())
      {
         if (reader.Read())
         {
            int userId = reader.GetInt32(0); // ID користувача
            string storedHash = reader.GetString(1);
            string roleName = reader.GetString(2);
            if (BCrypt.Net.BCrypt.Verify(password, storedHash))
            {
               this.Hide();
               if (roleName == "admin")
               {
                  AdminForm adminForm = new AdminForm();
                  adminForm.Show();
               }
               else if (roleName == "manager")
               {
                  ManagerForm managerForm = new ManagerForm();
                  managerForm.Show();
               }
               else if (roleName == "user")
               {
                  MainForm mainForm = new MainForm(login, userId);
                  mainForm.Show();
               }
            }
            else
            {
               MessageBox.Show("Невірний логін або пароль");
            }
         }
         else
         {
            MessageBox.Show("Невірний логін або пароль");
```

```
            }
        }
    }
}
```

Код реалізація реєстрації користувача та перевірка на коректність заповнення даних:

```
string login = textBox4.Text.Trim();       // логін
string password = textBox3.Text.Trim();     // пароль
string email = textBox2.Text.Trim();       // email
string fullName = textBox1.Text.Trim(); // повне ПІБ
if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password) ||
   string.IsNullOrEmpty(email) || string.IsNullOrEmpty(fullName))
{
   MessageBox.Show("Будь ласка, заповніть усі поля.");
   return;
}
using (var conn = Database.GetConnection())
{
   conn.Open();
   // Перевірка чи логін або емейл вже існує
   string checkQuery = "SELECT COUNT(*) FROM users WHERE login = @login OR email =
@email";
   using (var checkCmd = new NpgsqlCommand(checkQuery, conn))
   {
      checkCmd.Parameters.AddWithValue("login", login);
      checkCmd.Parameters.AddWithValue("email", email);
      long count = (long)checkCmd.ExecuteScalar();
      if (count > 0)
      {
         MessageBox.Show("Користувач з таким логіном або email вже існує.");
         return;
      }
   }
   // Вставка нового користувача з роллю user
   string insertQuery = @"
      INSERT INTO users (login, password_hash, email, full_name, role_id)
      VALUES (@login, @password, @email, @fullname, 1)";
   using (var insertCmd = new NpgsqlCommand(insertQuery, conn))
   {
      insertCmd.Parameters.AddWithValue("login", login);
      insertCmd.Parameters.AddWithValue("password", password); // бажано хеш
      insertCmd.Parameters.AddWithValue("email", email);
      insertCmd.Parameters.AddWithValue("fullname", fullName);
      int rows = insertCmd.ExecuteNonQuery();
      if (rows > 0)
      {
         MessageBox.Show("Реєстрація успішна!");
         this.Close();
```

```
        }
        else
        {
            MessageBox.Show("Сталася помилка при реєстрації.");
        }
    }
}
```

Код відображення страхових полісів:

```
flowLayoutPanel1.AutoScroll = true;
flowLayoutPanel1.Controls.Clear();
using (var conn = Database.GetConnection())
 {
    conn.Open();
    string query = @"
        SELECT ip.id, ip.policy_name, ip.description, ip.price, ip.franchise_required, ic.name
        FROM insurance_policies ip
        JOIN insurance_companies ic ON ip.company_id = ic.id";
    using (var cmd = new NpgsqlCommand(query, conn))
    using (var reader = cmd.ExecuteReader())
    {
        while (reader.Read())
        {
            int policyId = reader.GetInt32(0);
            string name = reader.GetString(1);
            string description = reader.IsDBNull(2) ? "" : reader.GetString(2);
            decimal price = reader.GetDecimal(3);
            bool franchise = reader.GetBoolean(4);
            string companyName = reader.GetString(5);

            var card = CreatePolicyCard(policyId, name, description, price, franchise, companyName);
            flowLayoutPanel1.Controls.Add(card);
        }
    }
 }
```

Код реалізації чату спілкування зі ШІ:

```
private async void BtnSend_Click(object sender, EventArgs e)

 {

    string userMessage = textBox1.Text.Trim();

    if (!string.IsNullOrWhiteSpace(userMessage))

    {

        SendMessage("User", userMessage);

        textBox1.Clear();
```

```
        string botReply = await GetGeminiResponse(userMessage);
        SendMessage("Bot", botReply);
    }
 }
private void LoadMessages()
{
   listBox1.Items.Clear();
   if (File.Exists(chatFilePath))
   {
      string[] messages = File.ReadAllLines(chatFilePath);
      foreach (var msg in messages)
      {
         listBox1.Items.Add(msg);
      }
   }
}
var content = new StringContent(JsonSerializer.Serialize(requestBody), Encoding.UTF8,
"application/json");
 string geminiApiUrl = $"https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-
flash:generateContent?key={GeminiApiKey}";
 var response = await httpClient.PostAsync(geminiApiUrl, content);
 if (response.IsSuccessStatusCode)
 {
    string responseContent = await response.Content.ReadAsStringAsync();
    using JsonDocument doc = JsonDocument.Parse(responseContent);
    if (doc.RootElement.TryGetProperty("candidates", out JsonElement candidates) &&
        candidates.GetArrayLength() > 0)
    {
        var firstCandidate = candidates[0];
```

```
        if (firstCandidate.TryGetProperty("content", out JsonElement contentElement) &&
            contentElement.TryGetProperty("parts", out JsonElement parts) &&
            parts.GetArrayLength() > 0)
        {
            var text = parts[0].GetProperty("text").GetString();
            return text?.Trim() ?? "[Відповідь порожня]";
        }
    }
    return "[Відповідь порожня]";
}
else
{
    string errorDetails = await response.Content.ReadAsStringAsync();
    return $"[Помилка отримання відповіді від Gemini API]:
{response.StatusCode}\n{errorDetails}";
}
```

Код реалізації генерування PDF файлу:

```
    string folderPath = "C:\\InsurancePDFs";
    Directory.CreateDirectory(folderPath);
    string fileName = $"insurance_{DateTime.Now.Ticks}.pdf";
    string filePath = Path.Combine(folderPath, fileName);
    using (FileStream fs = new FileStream(filePath, FileMode.Create))
    using (Document doc = new Document(PageSize.A4, 50, 50, 50, 50))
    using (PdfWriter writer = PdfWriter.GetInstance(doc, fs))
    {
        doc.Open();

        var boldFont = FontFactory.GetFont(FontFactory.HELVETICA_BOLD, 12);
        var normalFont = FontFactory.GetFont(FontFactory.HELVETICA, 12);
```

```
        Paragraph header = new Paragraph();
        header.Add(new Phrase("NAME\n\n", boldFont));
        header.Add(new Phrase("Company: ", boldFont));
        header.Add(new Phrase("ARCSS Insurance Group\"\n", normalFont));
        header.Add(new Phrase("Str: ", boldFont));
        header.Add(new Phrase("-\n", normalFont));
        header.Add(new Phrase("Number: ", boldFont));
        header.Add(new Phrase("+38 (044) 123-45-67\n\n", normalFont));
        doc.Add(header);
        doc.Add(new Paragraph($"USERS: {username}", normalFont));
        doc.Add(new Paragraph($"VIN CODE: {vin}", normalFont));
        doc.Add(new Paragraph($"CAR MODEL: {carModel}", normalFont));
        doc.Add(new Paragraph($"InsuredSUM: {insuredSum} UAH", normalFont));
        doc.Add(new Paragraph($"PAYMENT: {insuredSum} UAH", normalFont));
        doc.Add(new Paragraph($"Date: {insuranceDate:dd.MM.yyyy}\n\n", normalFont));
        PdfPTable table = new PdfPTable(3);
        table.WidthPercentage = 100;
        table.SetWidths(new float[] { 60f, 20f, 20f });
        AddCell(table, "\r\nCoating", boldFont, true);
        AddCell(table, "Summ", boldFont, true);
        AddCell(table, "Date", boldFont, true);
        AddCell(table, "CASKO", normalFont);
        AddCell(table, $"{insuredSum} UAH", normalFont);
        AddCell(table, $"{insuranceDate:dd.MM.yyyy} - {insuranceDate.AddYears(1):dd.MM.yyyy}",
normalFont);
        doc.Add(table);
        doc.Add(new Paragraph($"\nPayment all: {payment} UAH", boldFont));
        doc.Add(new Paragraph("Form payment: card", normalFont));
        doc.Add(new Paragraph("Signature: _____", normalFont));
```