МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ПРОГРАМУВАННЯ

Пояснювальна записка

до дипломної роботи
_____бакалавра_____
(освітньо-кваліфікаційний рівень)

на тему «Інтеграція веб-віджетів онлайн-страхування для оптимізації роботи страхових компаній»

Виконав: студент 4 курсу, групи ІПЗ-19д напряму підготовки спеціальності 121 „Інженерія програмного забезпечення ”
_____ Гніліцький О.С.
(підпис)

Керівник, доцент, к.т.н._____Іванов В.Г.
(підпис)

Рецензент, доцент, д.т.н._____ Лифар В.О.
(підпис)

КИЇВ – 2023

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ПРОГРАМУВАННЯ

Освітньо-кваліфікаційний рівень бакалавр
Спеціальність 121 „ Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТП, д.т.н., доцент
_____Лифар В.О.
«___» _____ 2023 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТА
ГНІЛІЦЬКОГО ОЛЕКСІЯ СЕРГІЙОВИЧА

1. Тема роботи Інтеграція веб-віджетів онлайн-страхування для оптимізації роботи страхових компаній
керівник роботи: к.т.н.,доцент, Іванов Віталій Геннадійович_____
затверджені наказом вищого навчального закладу від “26”04 2023 року №240/15.15-ОД
2. Строк подання студентом роботи 11 червня 2023 р.
3. Вихідні дані до роботи: Об’єктом даної роботи є процес створення віджета онлайн-страхування для інтеграції у веб-ресурси страхових компаній
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
5. Перелік графічного матеріалу немає
6. Дата видачі завдання 24 березня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Одержання завдання на виконання роботи	24.03.23	
2	Укладання і погодження з керівником плану і етапів виконання роботи	28.03.23	
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	1.04.23	
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	7.04.23	
3	Проектування інфологічної моделі задачі що реалізується.	15.04.23	
5	Укладання та тестування програмного продукту	1.05.23	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	25.05.23	
7	Укладання, оформлення та погодження з консультантом розділу «Охорона праці»	25.05.23	
7	Здача готової пояснювальної записки на кафедру	03.06.23	
8	Укладання доповіді і презентації	22.06.23	

Студент _____ Гніліцький О.С.
(підпис)

Керівник роботи _____ Іванов В.Г.

РЕФЕРАТ

Робота містить: 77 сторінок основного тексту, 5 сторінок додатків, 30 рисунків, 1 таблицю, 25 використаних джерел.

Метою випускної кваліфікаційної роботи є вивчення особливостей обробки даних в страхових компаніях для оформлення онлайн-страхування і реалізація веб-віджета спрямованої на оптимізацію роботи страхових компаній.

Були проаналізовані веб-додатки страхових компаній та маркетплейси які надають послуги з оформлення онлайн-страхування, обрані інструментальні засоби і технології, які дозволять швидко і якісно вирішити поставлену задачу.

В результаті виконаної роботи було спроектовано і реалізовано веб-віджет, який позволяє швидко та легко оформити страховку, обробку даних на боці страхової компанії, а також бути легко масштабованим віджетом виходячи з вимог замовника.

Система задовольняє всім вимогам, пред'явленим в технічному завданні.

Вироблено опис процесу розробки і тестування системи. Реалізовано, а також описаний користувальницький інтерфейс, зроблені знімки екранних форм програмного засобу. Продемонстровано результат виконаної роботи.

ЗМІСТ

ВСТУП

Страхування є однією з найбільш важливих послуг в сучасному світі, а страхові компанії змушені постійно вдосконалюватись та шукати нові шляхи, які допоможуть їм збільшити свою ефективність та залучити більше клієнтів. Оформлення страховки онлайн за допомогою віджетів на сайті страхової компанії є однією з найбільш важливих технологічних інновацій останніх років, що дозволяє значно спростити та прискорити процес оформлення страховки для клієнтів.

Такі віджети мають багато переваг. Наприклад, вони дозволяють користувачам швидко та зручно знайти всю необхідну інформацію про страхові продукти, порівняти різні пропозиції, а також оформити страховку безпосередньо на сайті компанії. Це зменшує час, необхідний для оформлення страховки та збільшує задоволення клієнтів від взаємодії зі страховою компанією. Крім того, такі віджети дозволяють знизити витрати страхової компанії на організацію роботи з клієнтами та підвищити ефективність її діяльності в цілому.

Ця робота допоможе страховим компаніям визначити необхідні критерії для розробки віджетів для страхових компаній. Розробка та впровадження віджетів на сайтах страхових компаній може значно спростити процес оформлення страховки для клієнтів та збільшити кількість продажів для компанії.

При розробці віджетів необхідно враховувати потреби користувачів та забезпечити максимально простий та зручний інтерфейс. Крім того, важливо забезпечити безпеку та захист персональних даних клієнтів, що вводяться в процесі оформлення страховки.

У результаті розробки та впровадження віджетів на сайті страхової компанії, можна забезпечити ефективнішу роботу компанії, покращити обслуговування клієнтів та збільшити прибуток. Тому, розробка віджетів для страхових компаній є важливою задачею, яка варта уваги та інвестицій.

Об’єктом дослідження є інтеграція веб-віджетів.

Предметом дослідження є аналіз сучасних проблем автоматизації сучасних страхових компаній, які не встигають за технологічним прогресом.

Метою роботи є дослідження ринку страхових компаній та покращення їх роботи.

Для реалізації цієї мети необхідно виконати наступні завдання:

Проаналізувати проблеми та перспективи розвитку страхових компаній;

аналіз ринку та вже існуючих віджетів онлайн продаж;

виконати постановку задачі;

вибрати і описати методологію проектування;

провести проектування схеми і опис бази даних;

спроектувати клієнтську частину;

обґрунтувати вибір інструментальних засобів;

реалізувати клієнтську частину;

описати призначений для користувача інтерфейс;

виконати необхідні дії для тестування.

1 АНАЛІЗ МЕТОДІВ І ЗАСОБІВ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ СТРАХОВИХ КОМПАНІЙ

1.1. Особливості організації роботи страхових компаній в умовах інтернет-середовища

Страхування - це процес, у якому страхова компанія, звана страховиком, бере на себе фінансовий ризик і зобов'язується відшкодувати втрати чи збитки, що виникли внаслідок певних подій, замість сплаченої страхової премії. Воно призначене для захисту людей або організацій від фінансових втрат, які можуть виникнути у разі нещасного випадку, хвороби, шкоди майну та інших ризиків.

Страхування ґрунтується на принципі колективного ризику, де багато людей або організацій сплачують страхові внески, щоб загальний фонд міг покрити можливі втрати для окремих застрахованих осіб у разі настання страхового випадку.

Існує безліч видів страхування, включаючи автомобільне страхування, медичне страхування, страхування нерухомості, страхування життя, страхування від нещасних випадків та інші. Кожен вид страхування має свої особливості та умови, визначені в полісі страхування, який є контрактом між страховиком та застрахованою особою чи організацією.

Страховик - це юридична або фізична особа, яка надає страховий захист та виплачує страхові відшкодування у разі настання страхового випадку. Він здійснює страхування від певних ризиків та приймає всі фінансові зобов'язання перед застрахованою особою чи організацією.

Страховики можуть бути представлені страховими компаніями та іншими фінансовими інститутами, які мають ліцензію на здійснення страхової діяльності. Вони аналізують ризики, встановлюють страхові премії, укладають страхові договори та виплачують відшкодування у разі настання страхового випадку відповідно до умов договору.

Страховик також відіграє роль фінансового посередника, збираючи страхові премії від застрахованих осіб та інвестуючи їх задля забезпечення фінансової стійкості компанії та виплати страхових відшкодувань.[1]

1.2 Поведінка споживача страхових послуг

Поведінка користувача, що взаємодіє зі страховими компаніями, може відрізнятися в залежності від конкретної ситуації, типу страхування та індивідуальних переваг користувача. Ось кілька типових аспектів поведінки користувачів щодо страхових компаній:

1. Пошук інформації: Користувачі часто шукають інформацію про різні види страхування, умови страхування, страхові компанії та їх репутацію. Вони можуть переглядати веб-сайти страхових компаній, читати відгуки, порівнювати пропозиції та проконсультуватися з іншими людьми, щоб ухвалити поінформоване рішення щодо вибору страхового покриття.

2. Укладання страхових договорів: Після вивчення різних варіантів користувачі можуть ухвалити рішення про купівлю страхового поліса. Вони можуть звернутися до страхової компанії, надати необхідні дані та інформацію про застрахований об'єкт чи собі самому, та заповнити відповідні документи. У цьому процесі користувачі зазвичай уважно вивчають умови страхування, премії, покриття, виключення та вимоги до страхового випадку.

3. Сплата страхових премій: Застраховані особи регулярно сплачують страхові премії відповідно до умов страхування. Вони можуть вибрати різні варіанти оплати, такі як щомісячні, щоквартальні або річні платежі, і вибрати найбільш зручний спосіб оплати, наприклад онлайн-платежі, автоплатежі або чеки.

4. Подання заяв про страховий випадок: У разі настання страхового випадку користувачі повинні звернутися до страхової компанії та подати заяву про страховий випадок. Це включає надання необхідних документів, опис події, що відбулася, і пояснення причин збитків. Користувачі очікують, що їхня заява буде опрацьована справедливо та відповідно до умов страхування.

5. Взаємодія з агентами та представниками страхової компанії: У процесі страхування користувачі можуть спілкуватися з агентами чи представниками страхової компанії. Ці фахівці можуть допомогти користувачам з вибором відповідного страхового покриття, відповісти на запитання, надати інформацію про деталі полісу та допомогти вирішити можливі проблеми чи суперечки.

6. Оцінка та виплата страхового відшкодування: При настанні страхового випадку страхова компанія проводить оцінку збитків або збитків. Користувачі чекають на чесну оцінку та своєчасну виплату страхового відшкодування відповідно до умов договору. Вони можуть надавати додаткові документи або інформацію, якщо це потрібно для розгляду заявки.

7. Оновлення та зміна полісу: Протягом терміну дії страхового поліса користувачі можуть вимагати зміни покриття, додавання або видалення об'єктів страхування або зміни особистої інформації. Вони можуть зв'язатися зі страховою компанією або агентом, щоб обговорити свої потреби та внести відповідні зміни до полісу.

8. Оцінка якості обслуговування: Після взаємодії зі страховою компанією користувачі можуть оцінити якість наданих послуг. Вони можуть залишити відгуки або оцінки про свій досвід, ґрунтуючись на якості обслуговування, процесі врегулювання збитків, своєчасності відповідей та інших факторах.

Важливо відзначити, що поведінка користувачів щодо страхових компаній може бути індивідуальною та залежати від їх особистих переваг, досвіду та рівня знань про страхування.[2]

1.3. Особливості конкурентного середовища на ринку страхових послуг

Конкретне середовище у сфері страхових послуг може мати свої особливості, які можуть змінюватись в залежності від місця розташування та умов ринку. Ось кілька особливостей, які можуть бути характерними для середовища у сфері страхових послуг:

1. Конкуренція між страховими компаніями: Сфера страхових послуг зазвичай представлена безліччю страхових компаній, конкуруючих між собою над ринком. Конкуренція може бути інтенсивною, і страхові компанії намагаються запропонувати привабливі умови, ширший спектр страхових продуктів та конкурентоспроможні тарифи, щоб залучити клієнтів.

2. Регулювання та законодавство: Сфера страхування зазвичай регулюється урядовими органами та законодавством, які встановлюють вимоги до фінансової стійкості страхових компаній, захисту прав споживачів та процедур

врегулювання збитків. Кожна країна чи регіон може мати свої специфічні правила та нормативи, яким мають відповідати страхові компанії.

3. Технологічні інновації: В останні роки страхові компанії активно впроваджують технологічні інновації, такі як цифрові платформи, онлайн-продажі та мобільні програми, щоб зробити процес купівлі страхових полісів та управління страховими послугами зручнішим для клієнтів. Технології, такі як штучний інтелект, аналітика даних та роботизовані процеси, також використовуються для підвищення ефективності страхових компаній.

4. Фінансові ризики та інвестиції: Страхові компанії стикаються з фінансовими ризиками, пов'язаними з виплатою страхових відшкодувань та управлінням страховими резервами. Вони також інвестують страхові премії, щоб забезпечити фінансову стійкість та отримання доходів. Управління фінансовими ризиками та ефективне управління інвестиціями є важливими аспектами для страхових компаній. Вони повинні оцінювати та керувати інвестиційними портфелями, щоб забезпечити достатні фінансові ресурси для виплати страхових відшкодувань. Це може включати різноманітні стратегії інвестування, такі як інвестиції в цінні папери, нерухомість або інші активи з урахуванням ризиків та прибутковості.

5. Споживчі вимоги та очікування: У сфері страхових послуг споживачі мають свої вимоги та очікування від страхових компаній. Вони бажають простоти та прозорості в процесі придбання полісів, швидкого та ефективного врегулювання збитків, а також хорошої якості обслуговування. Клієнтоорієнтованість та задоволення потреб клієнтів стають ключовими факторами успіху для страхових компаній.

6. Технічні та актуарні аспекти: У сфері страхування важливим аспектом є актуарна наука, яка займається оцінкою ризиків та встановленням страхових тарифів. Страхові компанії повинні мати актуарних фахівців, які аналізують дані, прогнозують ймовірність настання страхових випадків та визначають адекватні премії. Технічна експертиза також відіграє важливу роль в оцінці збитків та проведенні розслідувань.

Загалом сфера страхових послуг є динамічним та конкурентним середовищем, де страхові компанії прагнуть надати якісні послуги, ефективно управляти фінансовими ризиками та відповідати вимогам клієнтів та законодавства. Всі ці фактори формують особливості та основи роботи страхових компаній у конкретному середовищі.[3]

1.4. Перспектива використання віджетів онлайн продуктів в страхових компаніях

Застосування віджетів для страхових компаній має великий потенціал для подальшого розвитку та покращення роботи на страховому ринку. Онлайн-продукти, що використовують віджети, можуть забезпечити користувачів швидким та зручним способом отримання страхових послуг, зменшення часу, який потрібен на оформлення полісу, а також зменшення кількості помилок та неточностей в процесі страхування.

Крім того, використання віджетів може підвищити рівень задоволеності клієнтів страхових компаній та забезпечити їм більшу впевненість у своїй страховій політиці. Це може призвести до збільшення популярності та лояльності клієнтів до страхових компаній, що в свою чергу збільшує їхні прибутки та конкурентоспроможність на ринку.

Необхідно зазначити, що розробка та впровадження віджетів вимагає значних витрат на розробку та технічну підтримку, а також забезпечення безпеки персональних даних клієнтів. Для успішної реалізації проекту важливо планувати розробку та впровадження віджетів відповідно до потреб компанії та створювати продукти з урахуванням побажань та потреб користувачів.

На відміну від традиційних страхових продуктів, онлайн-страхування може мати обмеженість у покритті та інших важливих аспектах, які можуть вплинути на вибір страхової компанії. Також, існує ризик, що недосвідчені користувачі можуть зробити неточності при заповненні онлайн-форми, що може призвести до незадоволення клієнтів.

Використання віджетів для онлайн-страхування має великий потенціал для

покращення роботи страхових компаній та забезпечення більшої задоволеності клієнтів. З міцним плануванням та відповідною технічною підтримкою, страхові компанії можуть успішно впроваджувати нові онлайн-продукти та віджети, що дозволять їм залишатися конкурентоспроможними та привабливими для клієнтів.

Тому, щоб віджети стали дійсно ефективним інструментом для страхових компаній, необхідно розглядати їх як комплексну систему, яка повинна бути постійно вдосконалювана та адаптована до змін у вимогах ринку та потреб користувачів. Для цього необхідно проводити постійні дослідження та аналізувати дані про взаємодію користувачів з віджетами, щоб виявляти недоліки та вдосконалювати їх функціональність.

Загалом, використання віджетів в страхових компаніях має великий потенціал для покращення якості та швидкості надання страхових послуг, збільшення популярності та лояльності клієнтів, а також для збільшення прибутків та конкурентоспроможності компаній на ринку. Проте, успішна реалізація проекту вимагає великої уваги до питань безпеки та конфіденційності даних, а також до потреб та побажань користувачів.[4]

1.5. Аналіз існуючих систем автоматизації страхових компаній

Аналіз маркетплейсів з продажу страхових продуктів та сайтів страхових компаній становить інтерес для розуміння поточного ринку та пропозицій у сфері страхування. Розглянемо один із популярних маркетплейсів - hotline.finance, щоб проаналізувати його функціонал та особливості.

Hotline.finance пропонує широкий вибір страхових продуктів та послуг від різних страхових компаній. При попаданні на сайт та виборі основних параметрів та продукту страхування користувачеві надається список страхових компаній, які пропонують вибраний продукт, а також відповідну інформацію (рис. 1.1.).[5]

Основний функціонал маркетплейсу включає:

1. Пошук та порівняння
2. Інформація про страхові компанії

3. Онлайн оформлення
4. Підтримка клієнтів

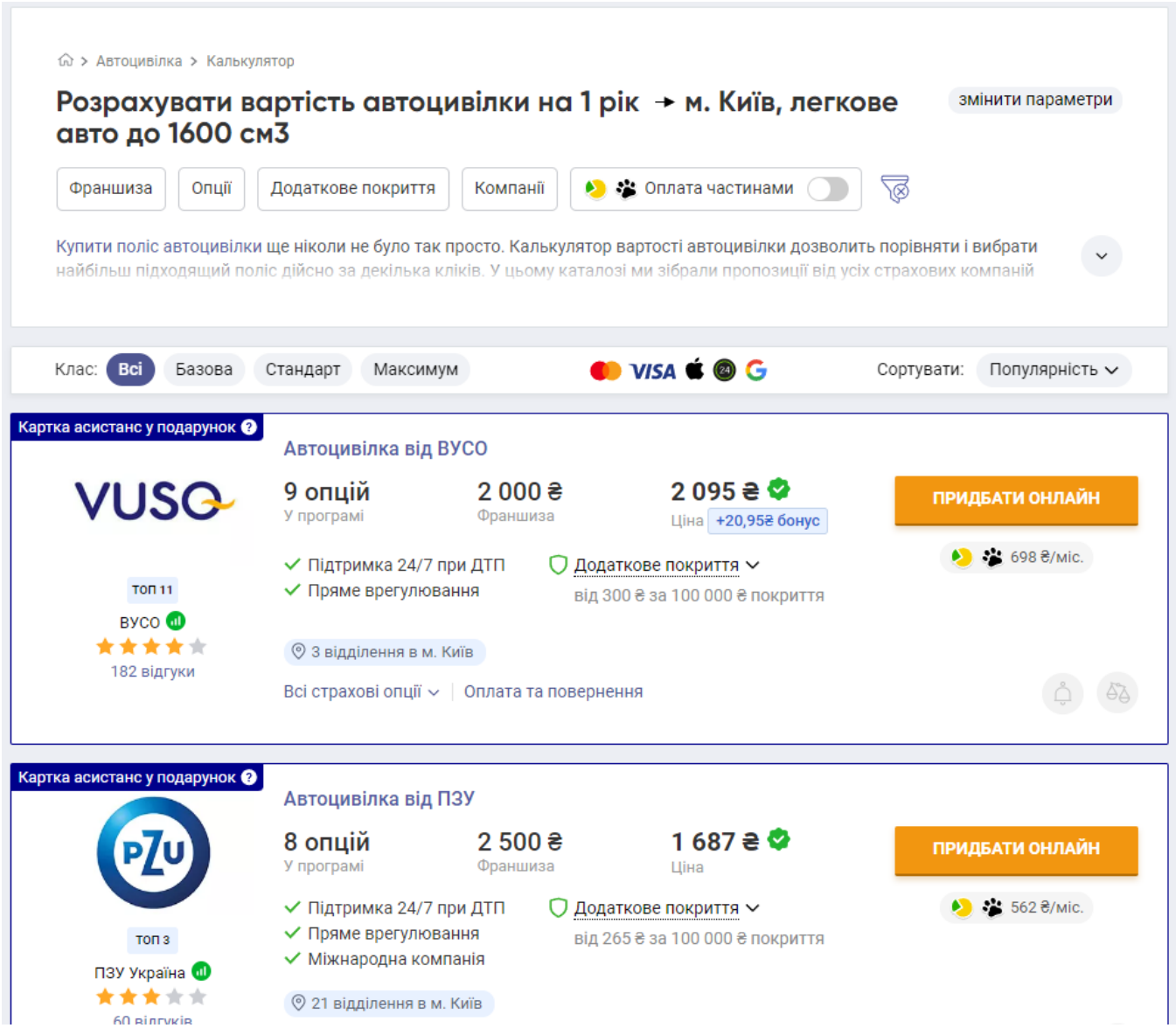


Рисунок 1.1. – Сторінка Hotline.finance з списком можливих страхових полісів.

Після того як ознайомився з умовами та додатковою інформацією, переходимо безпосередньо до оформлення страхового полісу. Оформлення відбувається кроками, можемо бачити кроки на рис. 1.2.

Аналіз маркетплейсів зі страхування допомагає користувачам порівняти пропозиції різних компаній, вибрати найбільш підходящий продукт та отримати найкращі умови страхування. Також важливо звернути увагу на функціональність і

зручність користувальницького інтерфейсу маркетплейсу, які також впливають на досвід користувача та його задоволеність.

Крім маркетплейсів, варто також проаналізувати веб-сайти страхових компаній, які пропонують свої продукти та послуги безпосередньо. Вони часто мають власні онлайн-платформи, де користувачі можуть отримати інформацію про страхові продукти, розрахувати вартість поліса, оформити його і зв'язатися з представниками компанії.

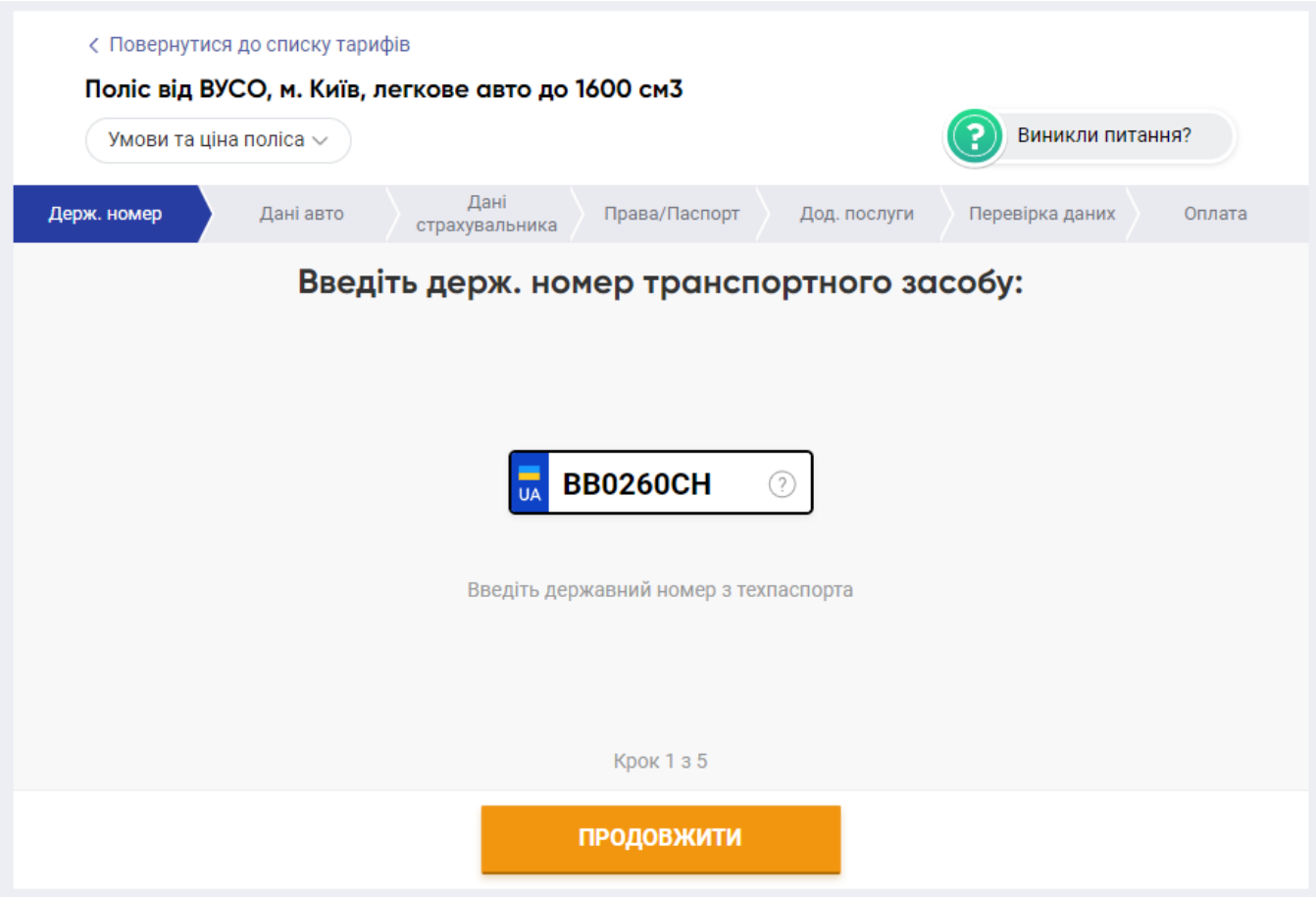


Рисунок 1.2. – Перший крок оформлення страхового полісу.

Переглянемо сторінку оформлення страхового полісу страхової компанії pzu.com.ua на малюнках бачимо приблизно схожу ситуацію з hotline.finance. Однак на стартовій сторінці перед безпосереднім оформленням страхового поліса, є набагато більше інформації про конректний продукт, з яким може ознайомитися користувач і прояснити для себе всю необхідну інформацію (рис.1.3.). На рис 1.4. бачимо анологічну ситуацію оформлення поліса за кроками.[6]

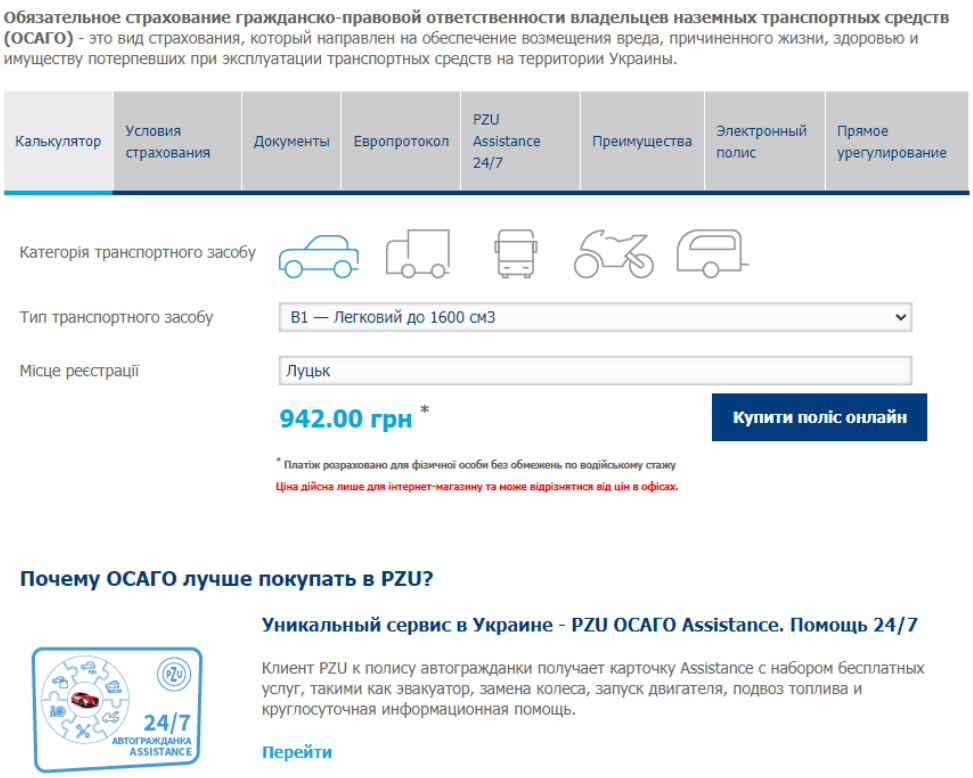


Рисунок 1.3. – Сторінка pzu.com.ua початок оформлення страхового полісу.

Рисунок 1.4. – Оформлення страхового полісу по крокам.

Після перегляду сторінок різних маркетплейсів та сайтів страхових компаній можемо бачити певну закономірність та схожі елементи надання онлайн-продуктів. Для створення свого віджету можна підкреслити два важливі пункти:

1. Дизайн та навігація віджету. Чистий та інтуїтивно зрозумілий дизайн. Хороша навігація та зручність віджету підвищують загальний користувальницький досвід та ймовірність завершення покупки.

2. Оформлення полісу по крокам – це є гарною практикою, яка полегшує процес для користувача. Розбиття процесу на кроки дозволяє впорядкувати інформацію та зробити його більш зрозумілим та зручним. Переваги крокового оформлення страхового поліса включають:

- Ясність і структурованість
- Поступове завантаження інформації
- Зручніть та легкість використання
- Контроль та перевірка даних

Крім того, важливо проаналізувати переваги та особливості кожного маркетплейсу або веб-сайту страхової компанії. Деякі можуть пропонувати додаткові послуги, такі як онлайн-консультації з фахівцями, можливість подання заяв на шкоду через інтернет або навіть інноваційні технології, такі як використання штучного інтелекту для рекомендації найбільш відповідних страхових продуктів.

1.6. Аналіз переваг та недостатків віджетів та маркетплейсу

Розміщення онлайн-продажів (віджетів онлайн страхування) на сайті самої страхової компанії має свої плюси та мінуси. Ось деякі з них:

Плюси розміщення онлайн-продажів на сайті страхової компанії:

1. Зручність для клієнтів: Онлайн-продаж дозволяє клієнтам купувати страхові поліси прямо на веб-сайті страхової компанії, що надає їм зручність і економить час. Клієнти можуть ознайомитися з доступними варіантами страхування, отримати цитати, порівняти тарифи та умови, а потім одразу здійснити покупку.

2. Більший контроль для клієнтів: Онлайн-продаж дозволяє клієнтам самостійно досліджувати страхові продукти, читати умови полісів і приймати поінформовані рішення без впливу агентів або представників страхової компанії. Це дає більший контроль клієнту над процесом вибору та купівлі страхування.

3. Швидка обробка та видача полісів: Онлайн-продаж дозволяє автоматизувати процес обробки та видачі полісів. Клієнти можуть заповнити необхідні дані, вибрати потрібні опції та одразу отримати свій поліс, що скорочує час очікування та спрощує процес для всіх сторін.

4. Скорочення витрат на маркетинг та продажі: Розміщення онлайн-продажів на власному сайті страхової компанії може скоротити витрати на маркетинг та продажі. Онлайн-продаж дозволяє залучати клієнтів безпосередньо і зменшити залежність від традиційних каналів продажів, таких як агенти або посередники.

Мінуси розміщення онлайн-продажів на сайті страхової компанії:

1. Обмежена консультація: Онлайн-продаж може обмежувати можливість отримання консультації та порад від професійних агентів або представників страхової компанії. Деяким клієнтам може бути необхідно обговорити свої індивідуальні потреби або отримати додаткові пояснення перед покупкою страхового полісу. У разі складних ситуацій або необхідності налаштування індивідуальних умов клієнтам може знадобитися особиста взаємодія з агентами або представниками страхової компанії.

2. Обмеження вибору страхових продуктів: Незважаючи на те, що сайт страхової компанії може пропонувати широкий спектр страхових продуктів, він завжди обмежений асортиментом, який надає сама компанія. Клієнти можуть відчувати обмеження у виборі та можливості порівняння різних страхових полісів та тарифів, оскільки інформація про конкурентні продукти може бути обмежена.

3. Можливі проблеми з онлайн-процесом: Онлайн-продажу можуть зіткнутися з технічними проблемами, такими як збої в роботі сайту, проблеми з онлайн-платежами або недоступність служби підтримки. Це може створити незадоволення у клієнтів та ускладнити процес купівлі страхування.

4. Нестача персоналізації: Можливості для персоналізації страхових полісів та умов на сайті страхової компанії можуть бути обмежені. Деякі клієнти можуть шукати більш гнучкі та індивідуальні варіанти страхування, які можуть бути складними для надання через онлайн-платформу.

5. Необхідність самостійного дослідження: Онлайн-продажу вимагають від клієнтів самостійного дослідження та розуміння страхових умов, термінів та полісів. Деяким людям може бути важко розібратися у всіх деталях без професійної допомоги чи консультації.

Загалом розміщення онлайн-продажів на сайті страхової компанії має свої переваги, такі як зручність, швидкість та зниження витрат. Однак воно також супроводжується деякими обмеженнями, пов'язаними з консультацією, вибором продуктів та потенційними проблемами з онлайн-процесом. Важливо, щоб страхові компанії знаходили баланс між зручністю онлайн-продажу та доступністю персоналізованої підтримки для клієнтів. Деякі компанії можуть пропонувати онлайн-чати, телефонну підтримку або електронну пошту клієнтам, які потребують допомоги або додаткової інформації.

Для успішної реалізації онлайн-продажів на сайті страхової компанії необхідно забезпечити зручність, прозорість та безпеку процесу. Це включає ясне уявлення інформації про продукти та послуги, зрозумілий опис полісів та умов, зручну систему оплати та захист персональних даних клієнтів.

Зрештою, рішення про розміщення онлайн-продажів на сайті страхової компанії залежить від стратегії компанії, переваг клієнтів та специфіки ринку. Сучасні споживачі все більше орієнтуються на онлайн-покупки та очікують зручності та простоти у процесі страхування. Тому, з урахуванням плюсів та мінусів, багато страхових компаній активно розвивають онлайн-продажі та інвестують у розробку інноваційних веб-платформ для надання страхових послуг.

Розміщення онлайн-продажів (віджетів онлайн страхування) на сайтах маркетплейсу має свої плюси та мінуси. Розглянемо деякі з них:

Плюси розміщення онлайн-продажів на сайтах маркетплейсу:

1. Широка аудиторія: Сайти маркетплейсу зазвичай мають велику відвідуваність та залучають широку аудиторію потенційних клієнтів. Розміщення онлайн продажів на таких платформах дозволяє страховим компаніям досягти більшої кількості людей, ніж вони могли б залучити через свій власний веб-сайт.

2. Зручність та доступність: Маркетплейси зазвичай пропонують зручний інтерфейс та інтуїтивно зрозумілий процес покупки. Клієнтам зручно і просто порівнювати різні страхові продукти, отримувати цитати та купувати в одному місці.

3. Репутація та довіра: Деякі маркетплейси мають встановлену репутацію та довіру у споживачів. Розміщення продуктів страхування на таких платформах може допомогти страховим компаніям покращити свою видимість та залучити клієнтів, які довіряють цьому маркетплейсу.

4. Маркетингові можливості: Маркетплейси часто пропонують маркетингові інструменти та можливості просування продуктів. Це може включати рекламні кампанії, спеціальні пропозиції чи співпрацю з іншими продавцями на платформі. Такі маркетингові можливості можуть допомогти страховим компаніям залучити більше клієнтів та збільшити продажі.

Мінуси розміщення онлайн-продажів на сайтах маркетплейсу:

1. Конкуренція: Сайти маркетплейс часто пропонують широкий вибір страхових продуктів від різних компаній. Страхові компанії стикаються із конкуренцією з боку інших продавців на платформі, що може ускладнити залучення клієнтів та знизити прибутковість. Конкуренція може призвести до зниження цін та зменшення маржі, особливо якщо на платформі присутні інші страхові компанії, що пропонують подібні продукти.

2. Обмежені можливості персоналізації: Сайти маркетплейсу зазвичай пропонують стандартизовані формати та шаблони для представлення продуктів. Це може обмежувати можливості страхових компаній у персоналізації та наданні індивідуальних умов або варіантів страхування, що може бути важливим для деяких клієнтів.

3. Залежність від платформи: Розміщення товарів на маркетплейсе робить страхову компанію залежною від платформи та її правил. Зміни у політиці платформи або обмеження, накладені на продавців, можуть вплинути на здатність страхової компанії надавати послуги або обмежити її гнучкість в управлінні продажами та маркетингом.

4. Відсутність прямого контакту з клієнтами: Розміщення товарів на маркетплейсі може зменшити прямий контакт страхової компанії з клієнтами. Це означає, що компанія може мати менше можливостей для збирання зворотного зв'язку, розуміння потреб клієнтів та надання індивідуальної підтримки.

Загалом розміщення онлайн-продажів (віджетів онлайн страхування) на сайтах маркетплейсу має свої переваги та недоліки. Для страхових компаній це може бути ефективним способом досягти більшої аудиторії та використовувати маркетингові можливості маркетплейсу. Однак вони також повинні враховувати конкуренцію, обмеження персоналізації та залежність від платформи. При ухваленні рішення про розміщення продуктів на маркетплейсі страхові компанії повинні ретельно оцінити свої цілі, стратегії та потреби клієнтів.

1.7. Постанова задачі

Розробка веб-віджету це складний процесс комплексної роботи, який вимагає великого досвіду від розробника цього впровадження, так як повино вміти користуватися богатьоми інструментами для разробки і розумінням роботи сервісів автоматизації. Тому, составимо повне технічне завдання по розробці віджету «ОСАГО » - страхування транспортных засобів, страхового полісу, розділене на блоки від аналізу до тестування та впровадження до веб-ресурсів страхових компаній.

1. Провести аналіз проблем та потреб страхових компаній та ринку вже створених віджетів, щоб зрозуміти, що можна запропонувати нового страховим компаніям. Також необхідно врахувати потреби користувачів та спростити оформлення онлайн-продукту.

2. Підготувати документацію для страхової компанії з описом роботи віджету та його зовнішнього вигляду (ui-шаблон). Після погодження можна

приступити до технічного аналізу та розробки віджету.

3. С творення архітектури проекту за допомогою Gradle та наповнення всіма необхідними бібліотеками. Створення базового проекту за допомогою Spring-Boot.

4. Розробка інтерфейсу віджету за допомогою фреймворку React.js. Віджет повинен містити три кроки:

Перший крок віджету містить поля для визначення параметрів страхування та розрахунку вартості страхування. Для цього створюється компонент з полями вводу та кнопкою для розрахунку вартості страхування. При введенні користувачем даних в поля вводу, React автоматично оновлює внутрішній стан компонента та перерендерює інтерфейс.

Другий крок містить поля для введення персональних даних такі як: ПІБ, дата народження, адреса, номер телефону, електронна пошта та інші. Для цього створюється окремий компонент з полями вводу для персональних даних та кнопкою для створення договору. Після заповнення всіх полів користувачем, віджет виконує запит на сервер для створення договору та отримання даних для підтвердження договору на наступному кроці.

Третій крок містить правила та умови страхування, кнопку для оплати. Після оплати договор автоматично підтверджується.

5. Розробка основної логіки, обробка заповнених полів, розрахунок вартості страхування за допомогою предоставленої формули страхової компанії, збереження договору.

6. Розробка сервісів впровадження веб-віджету на сторінки страхових компаній за допомогою js-скриптів.

7. Тестування системи на тестовому середовищі, виявлення та виправлення помилок. Демонстрація системи страховій компанії.

8. Впровадження та налаштування веб-віджету на сторінки страхових компаній.

9. Надання аналітичної та технічної підтримки страхової компанії та веб-віджету.

Для покращення ефективності розробки веб-віджету, можна використовувати agile-методології розробки, такі як Scrum або Kanban, для поетапної розробки та ітеративного вдосконалення віджету на основі зворотного зв'язку від користувачів та страхових компаній.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Методологія проектування інформаційних систем

Існує кілька методологій проектування інформаційних систем, і вибір конкретної методології залежить від вимог проекту та переваг команди розробників. Ось деякі з найпоширеніших методологій:

1. Waterfall (Каскадна модель): Це класична методологія, яка передбачає послідовне виконання фаз проекту, таких як визначення вимог, проектування, розробка, тестування та впровадження. Кожна фаза завершується перед переходом до наступного. Ця методологія підходить для проектів із чіткими та стабільними вимогами.

2. Agile (Гнучка методологія): Agile є ітеративною та інкрементною методологією, яка акцентує увагу на гнучкому та швидкому реагуванні на зміни вимог. Проект розбивається на короткі цикли розробки, які називаються спринтами, протягом яких відбувається розробка, тестування та зворотний зв'язок. Agile методології включають Scrum, Kanban та Extreme Programming (XP).

3. Spiral (Спіральна модель): Це ітеративна модель, яка поєднує елементи каскадної моделі та прототипування. Проект розробляється по спіралі, де кожен оборот становить одну ітерацію, що включає визначення вимог, проектування, розробку, тестування та оцінку ризиків. Ця модель підходить для проектів, які потребують безперервного аналізу та оцінки ризиків.

4. RAD (Rapid Application Development, Швидка розробка додатків): Методологія RAD спрямована на швидку розробку та впровадження додатків. Вона включає сильну взаємодію із замовниками, швидке прототипування та ітеративне створення системи. RAD підходить для проектів з обмеженими термінами та вимогами, що змінюються.

5. DevOps (Development and Operations, Розробка та експлуатація): DevOps поєднує розробку та експлуатацію інформаційних систем, з метою створення більш ефективного процесу розробки та забезпечення безперервного постачання. Він включає автоматизацію процесів розробки, тестування, розгортання та моніторингу системи.

Кожна методологія має свої особливості, переваги та недоліки. Важливо вибрати методологію, яка найкраще відповідає потребам та характеру проекту.

Крім вибору методології, важливо враховувати такі аспекти при проектуванні інформаційних систем:

- Стратегія та бізнес-мети проекту.
- Визначення вимог користувачів та зацікавлених сторін.
- Аналіз та проектування даних, включаючи структуру бази даних та методи зберігання.
- Вибір відповідних технологій та інструментів розробки.
- Планування та управління проектом, включаючи розподіл ресурсів та контроль прогресу.
- Тестування та забезпечення якості розробки.
- Розгортання та підтримка системи після впровадження.[7]

2.2. Agile - Гнучка методологія

Гнучка методологія розробки інформаційних систем, також відома як Agile (гнучкий), є ітеративним та інкрементним підходом до розробки програмного забезпечення. Вона акцентує увагу на гнучкості, співпраці та швидкій адаптації до вимог замовника, що змінюються. Agile підхід дозволяє команді розробників створювати цінність для замовника на ранніх стадіях проекту та безперервно покращувати продукт через ітерацію розробки (рис. 2.1.).

Ось основні принципи та практики гнучкої методології розробки інформаційних систем:

1. Командна робота: Agile підхід передбачає сильну взаємодію та співпрацю між членами команди розробників, а також із замовником та зацікавленими сторонами. Команда працює разом для досягнення спільних цілей проекту.

2. Ітеративний підхід: Розробка ведеться через серію коротких циклів розробки, які називаються спринтами або ітераціями. Кожна ітерація включає планування, розробку, тестування і зворотний зв'язок від замовника. Це дозволяє

швидко створювати робочий функціонал та отримувати зворотний зв'язок для його покращення.

3. Прозорість та зворотний зв'язок: Agile підхід ставить акцент на прозорість та регулярний зворотний зв'язок. Всі члени команди та замовник мають доступ до інформації про прогрес розробки та можуть надавати зворотний зв'язок, що сприяє більш точному визначенню вимог та покращенню якості продукту.

4. Постійне покращення: Команда прагне безперервного поліпшення процесу розробки та якості продукту. Після кожної ітерації проводиться ретроспектива, де команда аналізує свою роботу і шукає способи поліпшитися.

5. Гнучкість та адаптація: Agile методологія сприяє гнучкості та адаптації до змін вимог. Якщо вимоги змінюються у процесі розробки, команда може швидко реагувати та перерозподілити свої пріоритети.[8]

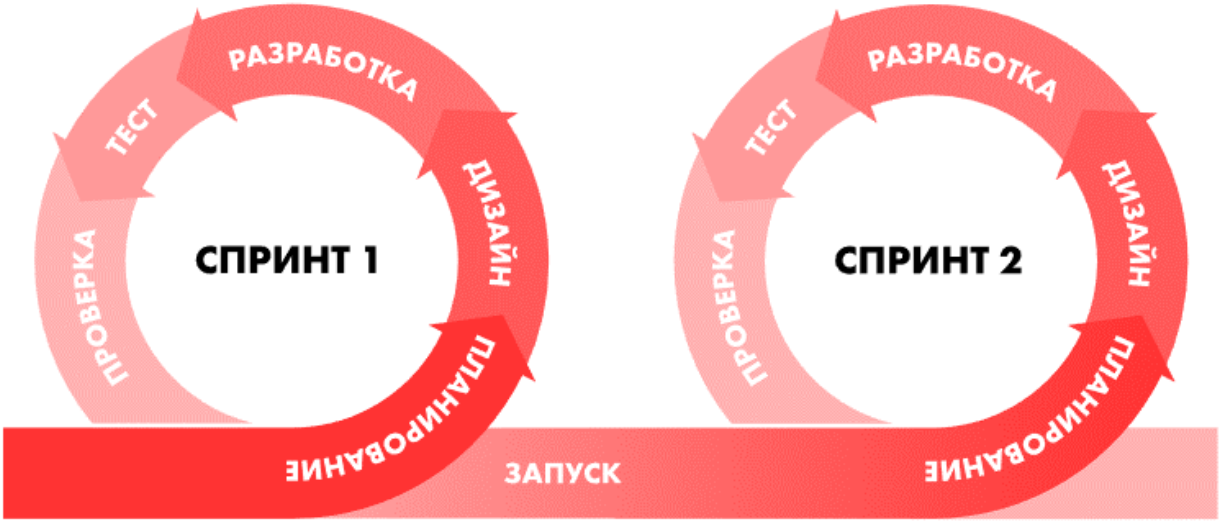


Рисунок 2.1. – Підхід гнучкою методологію

Розглянемо популярне гнучке методологія розробки інформаційних систем, заснованих на Agile, таке як Scrum

Scrum є одним з найбільш широко використовуваних фреймворків Agile. Він базується на концепції спринтів, які є фіксованими тимчасовими інтервалами (зазвичай 2-4 тижні), протягом яких команда розробників доставляє робочий функціонал. Scrum включає ролі, такі як Scrum-майстер, власник продукту і

розробники, а також різні події, такі як планування спринту, щоденні стендапи і рев'ю спринту (рис.2.2.).[9]

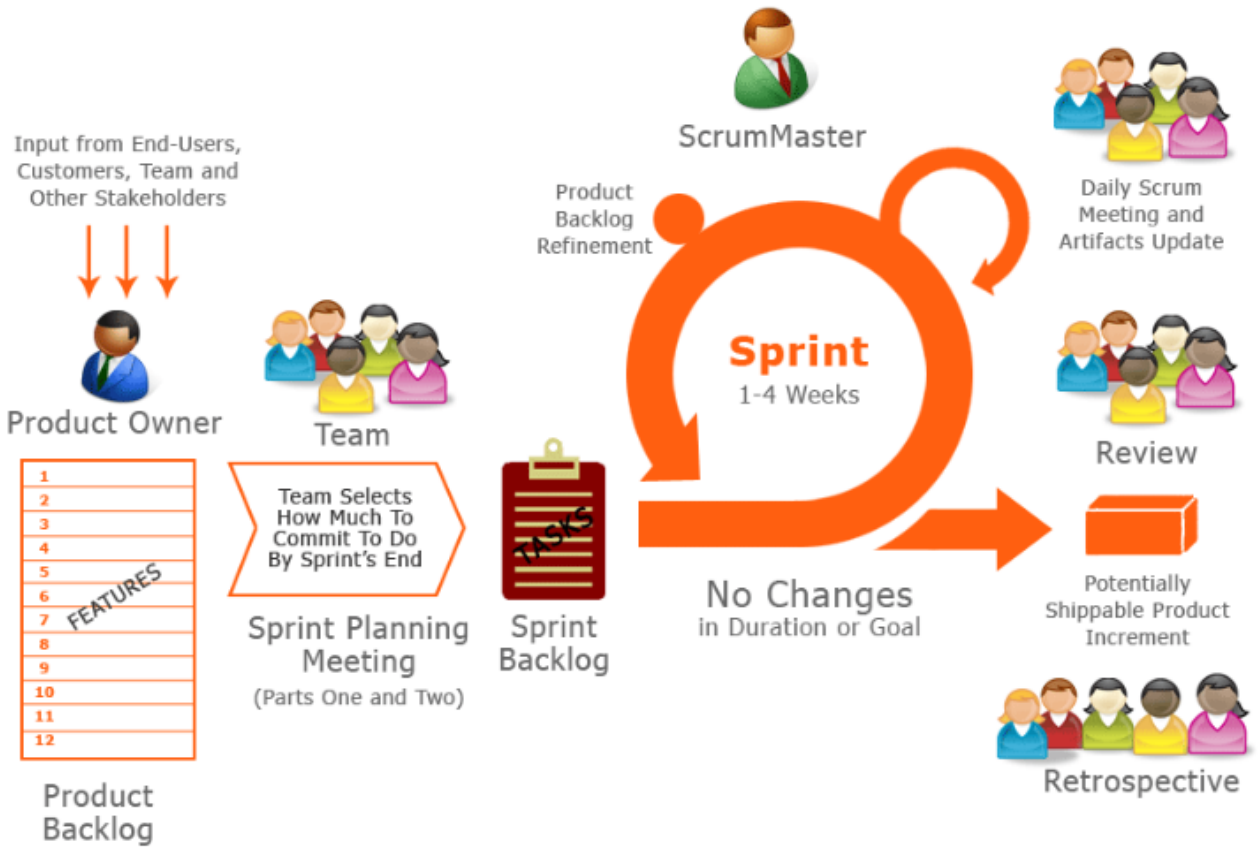


Рисунок 2.2. – Концепція Scrum

2.3. Проектування архітектури системи

Проектування архітектури інформаційних систем є важливим етапом розробки, де визначається структура, компоненти та взаємодії системи. Архітектура інформаційної системи описує її основні складові та способи їх взаємодії, щоб забезпечити досягнення необхідних функціональних та нефункціональних характеристик.

1. Ідентифікація потреб та вимог: Спочатку необхідно визначити потреби та вимоги користувачів та замовника, щоб зрозуміти, яка функціональність та характеристики мають бути включені до архітектури системи.

2. Визначення компонентів системи: Архітектура системи має визначити основні компоненти, з яких вона складатиметься. Компоненти можуть містити програми, бази даних, інтерфейси, сервери та інші елементи.

3. Визначення структури та організації: Важливим аспектом архітектури є визначення структури системи та способів організації компонентів. Можуть використовуватись різні архітектурні стилі, такі як клієнт-сервер, триланкова архітектура, мікросервісна архітектура і т.д.

4. Встановлення взаємодій: Архітектура системи має визначити способи взаємодії між компонентами системи. Це може містити визначення протоколів обміну даних, інтерфейсів, API та інших способів комунікації.

5. Забезпечення масштабованості та продуктивності: Архітектура системи має бути спроектована з урахуванням масштабованості та продуктивності. Це включає врахування можливості розширення системи, обробки великих обсягів даних та ефективного використання ресурсів.

6. Забезпечення безпеки: Архітектура системи повинна враховувати заходи безпеки та захисту даних. Це може включати автентифікацію, авторизацію, шифрування та інші механізми захисту.

7. Документування та комунікація: Важливим аспектом проектування архітектури є документування створеної архітектури.[10]

2.4. Етап визначення потреб і вимог віджету

Основні вимоги до віджету в рамках проекту включають наступні функціональні та дизайн-аспекти:

1. Вивантаження даних з бази ЦБД (МТСБУ): Віджет повинен мати можливість отримувати дані про автомобіль з бази даних Моторного страхового бюро України (МТСБУ) за номером авто та автоматично заповнювати необхідну інформацію для оформлення страхового полісу.

2. Вибір даних із існуючих систем довідників: Віджет повинен мати доступ до системних довідників, що дозволяє заповнювати поля з використанням наявної інформації.

3. Підказки на полях та кроках віджету: Віджет повинен надавати зі зрозумілими підказками та поясненнями для полів та кроків оформлення страхового полісу, що сприяє зручності та швидкості заповнення.

4. Використання картинок: Віджет може використовувати картинки для покращення візуального вигляду полів, підказок та інших елементів. Картинки повинні бути оптимізовані щодо розміру, забезпечуючи ефективне завантаження сторінок.

5. Можливість ручного введення: Віджет повинен дозволити користувачам вручну вводити дані, якщо це необхідно, забезпечуючи гнучкість та універсальність використання.

6. Валідація полів: Віджет повинен проводити перевірку введених даних на основі заданих правил та форматів, які запобігають помилкам та некоректним даним.

7. Оптимізація скриптів та дизайну CSS: Віджет повинен бути оптимізований з точки зору продуктивності, мінімізації завантаження сторінки

8. Підтвердження за допомогою sms-коду: Віджет повинен надати можливість підтвердження деяких дій або транзакцій за допомогою відправки sms-коду на мобільний телефон користувача. Це забезпечує додатковий рівень безпеки та захисту інформації.

9. Відправлення нотифікацій і квитанцій за допомогою notification-service: Віджет повинен мати можливість надсилати повідомлення та квитанції користувачам за допомогою сервісу сповіщень (notification-service). Це дозволяє інформувати користувачів про стан їхніх транзакцій та надавати документальне підтвердження.

10. Google analytics – аналітка для компанії, за кроками та певними діями, для стеження на яких кроках можуть виникати проблеми при оформленні та подальше вдосконалення їх.

Щодо дизайну, важливі такі аспекти:

1. Колір та форма кнопок модулів введення: Віджет повинен мати привабливі та легкозрозумілі кнопки з належною видимою формою.

2. Фон полів вводу: Віджет може мати візуально привабливий та зручний фон для полів, що покращує відчуття користувача під час введення даних.

3. Колірність іконок: використання іконок із приємною кольоровістю додає видимої привабливості та допомагає виділити важливі елементи віджету.

4. Візуалізація підказок: Віджет повинен використовувати привабливі та зрозумілі візуальні елементи для підказок, які сприяють легкому сприйняттю інформації користувачем.

5. Фон підказок полів вводу: Зручний та візуально привабливий фон підказок полів надає естетики та зручності використання віджету.

6. Налаштування єдиного шрифту віджету: використовуйте єдиний шрифт у всіх віджетах, щоб створити єдиний стиль і забезпечити зручну читабельність тексту для користувачів.

7. Загальний фон віджету (фон): Важливо звернути приємний фон для віджету, який не відмовляє користувачам і відповідає загальному стилю та брендуванню страхової компанії.

2.5. Етап визначення компонентів системи

Для розробки віджетів будемо використовувати язык програмування Java для бекенда и фреймворк React.js для фронтенда. Проект буде використовувати Spring-Boot, для інтеграції бекенда и фронтеда.

Java - це мова програмування, яка має великий досвід в розробці великих масштабних проектів. Вона дозволяє забезпечити надійність, швидкість та масштабованість. Більш того, багато компаній обирають Java як свою основну мову програмування для розробки додатків.[11]

Spring Boot - це фреймворк для розробки додатків мовою Java, який полегшує та прискорює процес створення самостійних та мікросервісних додатків. Він надає все необхідне для розробки веб-застосунків, керування залежностями та автоматичне конфігурування, простота використання , вбудовані сервери програм, підтримка мікросервісної архітектури.

Інтеграція з іншими модулями Spring - Spring Boot щільно інтегрований з іншими модулями Spring, такими як Spring MVC, Spring Data, Spring Security та іншими. Це дозволяє розробникам використовувати потужні функціональні можливості Spring Framework у своїх додатках.

Spring Boot дозволяє розробникам зосередитися на бізнес-логіці своїх програм, забезпечуючи зручну та ефективну платформу для розробки. Він скорочує час та зусилля, необхідні для налаштування та конфігурування програми, дозволяючи вам швидко приступити до розробки функціональності та доставки готової програми у виробництво.[12]

React є одним з найпопулярніших фреймворків для розробки інтерфейсу користувача. Він має такі переваги:

Переваги React:

- Є дуже продуктивним та швидким фреймворком, оскільки використовує віртуальну модель DOM (Virtual DOM) та вміє оновлювати лише змінені компоненти замість перемальовування всього інтерфейсу.
- Дозволяє використовувати JSX, який надає зручний спосіб написання коду в стилі HTML і робить його більш читаним.
- Забезпечує зручну багатокомпонентну архітектуру, яка спрощує супровід коду та його масштабування.
- Може використовуватися з різними інструментами та бібліотеками, що робить його дуже гнучким та розширюваним.

Деякі недоліки React:

- Робота з React може бути складною для новачків у програмуванні, оскільки потребує знань HTML, CSS, JavaScript та різних інструментів та бібліотек.
- Не підходить для розробки програм з великим обсягом даних, тому що в цьому випадку може виникнути проблема з продуктивністю.
- Необхідно знати та використовувати деякі додаткові інструменти для роботи з React, наприклад Webpack, Babel та інші.[13]

PostgreSQL - це потужна реляційна система управління базами даних (СУБД), яка надає розширені можливості для зберігання, організації та обробки даних. Вона є однією з найпопулярніших і найнадійніших СУБД у світі та широко використовується в багатьох додатках та системах. В цілому, PostgreSQL – це потужна та гнучка реляційна СУБД, яка пропонує широкий спектр функцій та

можливостей. Вона ідеально підходить для розробки та управління різними типами додатків та систем, і її активна спільнота підтримує та розвиває цю технологію, роблячи її ще більш привабливою для розробників та організацій. Для адміністрування будемо використовувати pgAdmin4.[14]

pgAdmin 4 - це безкоштовний та відкритий інструмент адміністрування баз даних PostgreSQL. Він надає графічний інтерфейс для управління та моніторингу баз даних PostgreSQL, а також виконує широкий спектр завдань, пов'язаних із розробкою та адмініструванням баз даних (рис. 2.3.).[15]

5

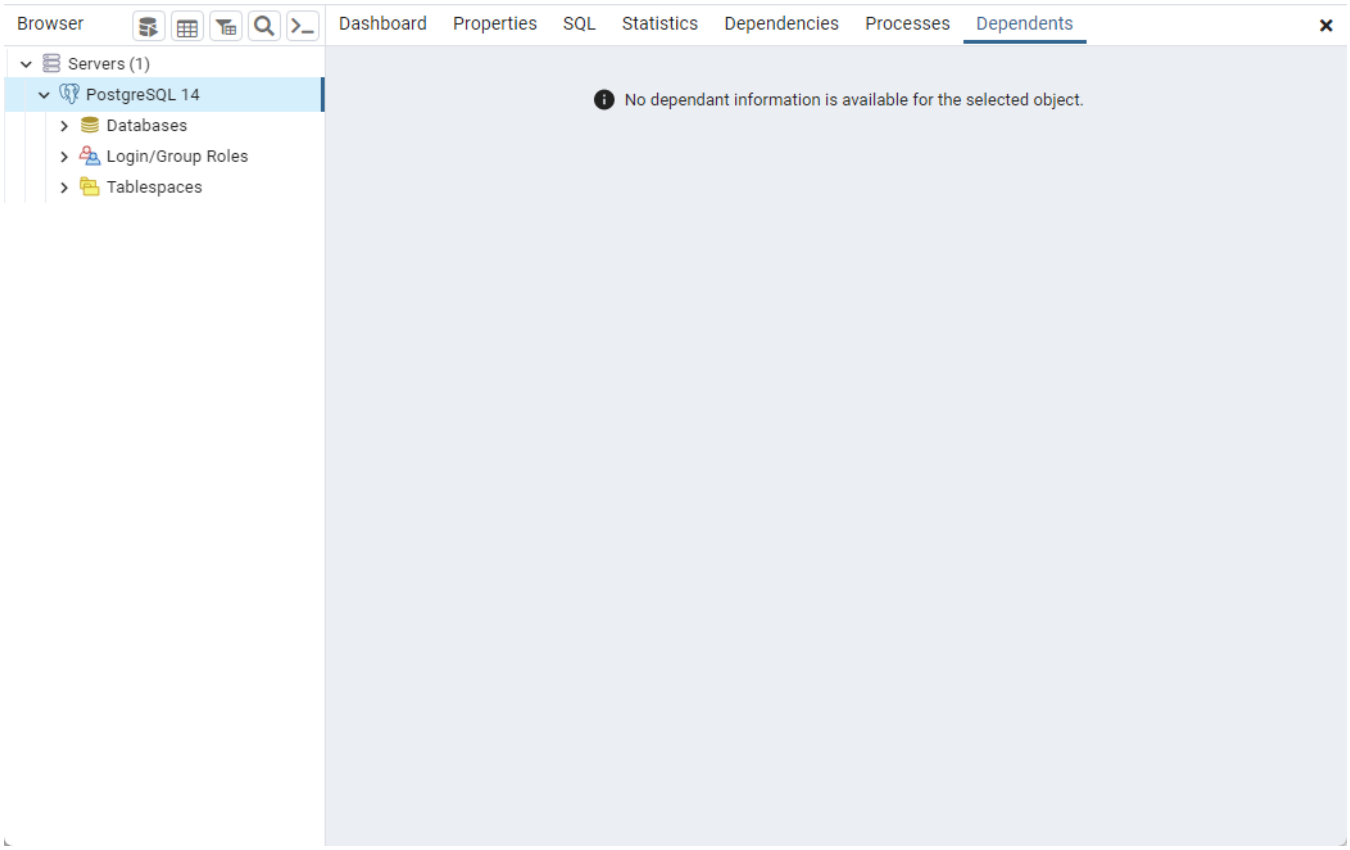


Рисунок 2.3. – Інтерфейс pgAdmin 4

Gradle – це система автоматизації збирання, яка широко використовується в розробці програмного забезпечення. Вона надає зручний та гнучкий спосіб управління залежностями, компіляцією, тестуванням та розгортанням проекту.

Деякі особливості та переваги Gradle - декларативна конфігурація,гнучкість та масштабованість, управління залежностями, підтримка безлічі мов та платформ, інкрементальне складання, легка інтеграція із середовищами розробки.

Він широко використовується в різних сценаріях розробки, включаючи створення веб-додатків, мобільних додатків, бібліотек та багато іншого.

Gradle підтримує різні парадигми складання, включаючи імперативну та декларативну моделі. Ви можете визначати свої завдання та налаштування складання на основі ваших потреб та уподобань.

Однією з головних особливостей Gradle є його здатність опрацьовувати великі та складні проекти з ефективним управлінням залежностями. Ви можете легко визначити залежності від зовнішніх бібліотек та модулів, а також керувати версіями та оновленнями.

Крім того, Gradle інтегрується з різними інструментами розробки та середовищами розробки, що забезпечує синхронізацію з робочим процесом. Ви можете використовувати Gradle у командному рядку або інтегрувати його у свою улюблену IDE для зручності та ефективності розробки.

В цілому, Gradle є потужним інструментом складання, який надає розробникам широкий спектр можливостей для управління та автоматизації процесу складання проекту. Він дозволяє створювати ефективні та гнучкі складальні скрипти, що сприяє більш ефективній та продуктивній розробці програмного забезпечення.[16]

Git - це розподілена система управління версіями, яка широко використовується розробниками для відстеження змін у коді та спільної роботи над проектами. Вона надає можливість ефективно керувати кодовою базою, відстежувати зміни, вносити виправлення та об'єднувати галузі розробки.

Git є одним з найпопулярніших і широко використовуваних інструментів керування версіями. Він надає потужні можливості для контролю змін у коді, спільної роботи та ефективного управління розробкою проектів. Ознайомлення з основами Git та його використання дозволять вам ефективно працювати з кодом та керувати версіями ваших проектів.[17]

2.6. Мікросервіси

Мікросервіси - це архітектурний підхід, при якому програма розбивається на набір невеликих, незалежних та самодостатніх сервісів, які взаємодіють один з одним через легковажні протоколи, такі як HTTP/REST. Кожен мікросервіс є окремою і обмеженою функціональністю, що виконує конкретне завдання або надає певну послугу.

Мікросервісна архітектура дозволяє досягти більш гнучкої та масштабованої розробки інформаційних систем. Вона сприяє ефективнішому поділу праці, прискореної постачання товару ринку, поліпшенню масштабованості та забезпечення отказоустойчивости. Однак, при використанні мікросервісної архітектури слід враховувати додаткові складності в управлінні, взаємодії та тестуванні багатьох незалежних сервісів.[18]

API (Application Programming Interface) - це набір певних правил і протоколів, що дозволяють різним програмним додаткам взаємодіяти друг з одним. API визначає способи та формати обміну даними між різними компонентами системи або між різними системами.

API може надавати різні функціональні можливості, такі як отримання даних, надсилання даних, виконання певних операцій та інші. API може бути представлений у вигляді набору методів, функцій, класів, бібліотек або веб-сервісів, які інші програми можуть використовувати для доступу до функціональності системи.

Одним із найпоширеніших типів API є веб-API, яке дозволяє взаємодіяти з веб-сервісами через мережу. Веб-API зазвичай використовують стандартні протоколи та формати, такі як HTTP та JSON, для передачі даних та комунікації між клієнтом та сервером.

API також можуть бути класифіковані як публічні (доступні всім розробникам) або приватні (обмежені доступом). Публічні API зазвичай надаються

розробниками або компаніями для створення екосистеми навколо своїх продуктів або сервісів, дозволяючи іншим розробникам інтегрувати їхню функціональність у свої програми.

Використання API дозволяє розробникам повторно використовувати функціональність, спрощувати інтеграцію між різними додатками та створювати більш гнучкі та масштабовані системи. API є важливою частиною розробки програм та широко використовуються в різних сферах, включаючи веб-розробку, мобільні програми, хмарні сервіси та інші.[19]

REST (Representational State Transfer) - это архитектурный стиль для построения распределенных систем, основанный на принципах и ограничениях, описанных в его родительской работе Роя Филдинга (Roy Fielding) в 2000 году. REST представляет собой легковесный и гибкий подход к проектированию веб-API.

Принципы REST включают:

1. Клиент-серверная архитектура: Система разделена на клиентские и серверные компоненты, которые взаимодействуют друг с другом посредством обмена представлениями ресурсов.

2. Без состояния (Stateless): Каждый запрос клиента к серверу должен содержать всю необходимую информацию для его обработки. Сервер не должен сохранять состояние клиента между запросами.

3. Кэширование: Клиенты или промежуточные серверы могут кэшировать ответы сервера для повторного использования, если это разрешено сервером.

4. Единообразный интерфейс: REST предоставляет унифицированный интерфейс для взаимодействия с ресурсами, включая использование уникального идентификатора ресурса (URI), стандартных методов HTTP (GET, POST, PUT, DELETE) и форматов представления данных (например, JSON, XML).

5. Слои: Клиенты не должны знать о промежуточных слоях, которые могут использоваться для обработки запросов или повышения масштабируемости системы.

RESTful API использует HTTP-протокол для обмена данными между клиентом и сервером. Клиент отправляет запросы на сервер, указывая метод HTTP (например, GET для получения ресурса, POST для создания ресурса) и URI, и сервер отвечает соответствующими статусами HTTP и представлением запрошенного ресурса.

RESTful API позволяет создавать легко понятные, расширяемые и масштабируемые веб-API. Он широко применяется в различных областях, включая разработку веб-приложений, мобильных приложений и интернета вещей (IoT).[20]

2.7. Model-View-Controller

Spring MVC (Model-View-Controller) є частиною Spring Framework, що надає архітектурний шаблон розробки веб-додатків. Він заснований на патерні проектування MVC, який розділяє додаток на три основні компоненти: модель (Model), представлення (View) та контролер (Controller) (рис.2.5)

Spring MVC модель представляє дані, які використовуються в додатку. Вона може бути представлена у вигляді об'єктів Java або POJO (Plain Old Java Object). Модель містить бізнес-логіку і стан даних, які повинні бути відображені в інтерфейсі користувача.

Подання відповідає за відображення даних користувачеві. Воно може бути представлене у різних форматах, таких як HTML, JSON, XML та інші. Spring MVC надає можливість використовувати шаблонизатор (наприклад, JSP, Thymeleaf) для створення динамічного подання даних.

Контролер є посередником між моделлю та поданням. Він обробляє вхідні запити від клієнта, витягує необхідні дані з моделі та передає їх у виставу для відображення. Контролер також приймає дані від користувача та оновлює модель відповідно до цих даних.

Spring MVC забезпечує централізовану обробку запитів та управління життєвим циклом програми. Він надає механізми маршрутизації запитів, обробки форм, обробки винятків та інших завдань, пов'язаних з веб-розробкою. Крім того, Spring MVC забезпечує інтеграцію з іншими модулями Spring Framework, такими як Spring Security, Spring Data та іншими.

Spring MVC пропонує гнучкий та розширюваний підхід до розробки веб-додатків на Java. Він дозволяє розділити логіку програми на логічні шари, забезпечує легкість тестування та підтримки, а також забезпечує можливість використання різних технологій та інструментів для розробки інтерфейсу користувача. [12]

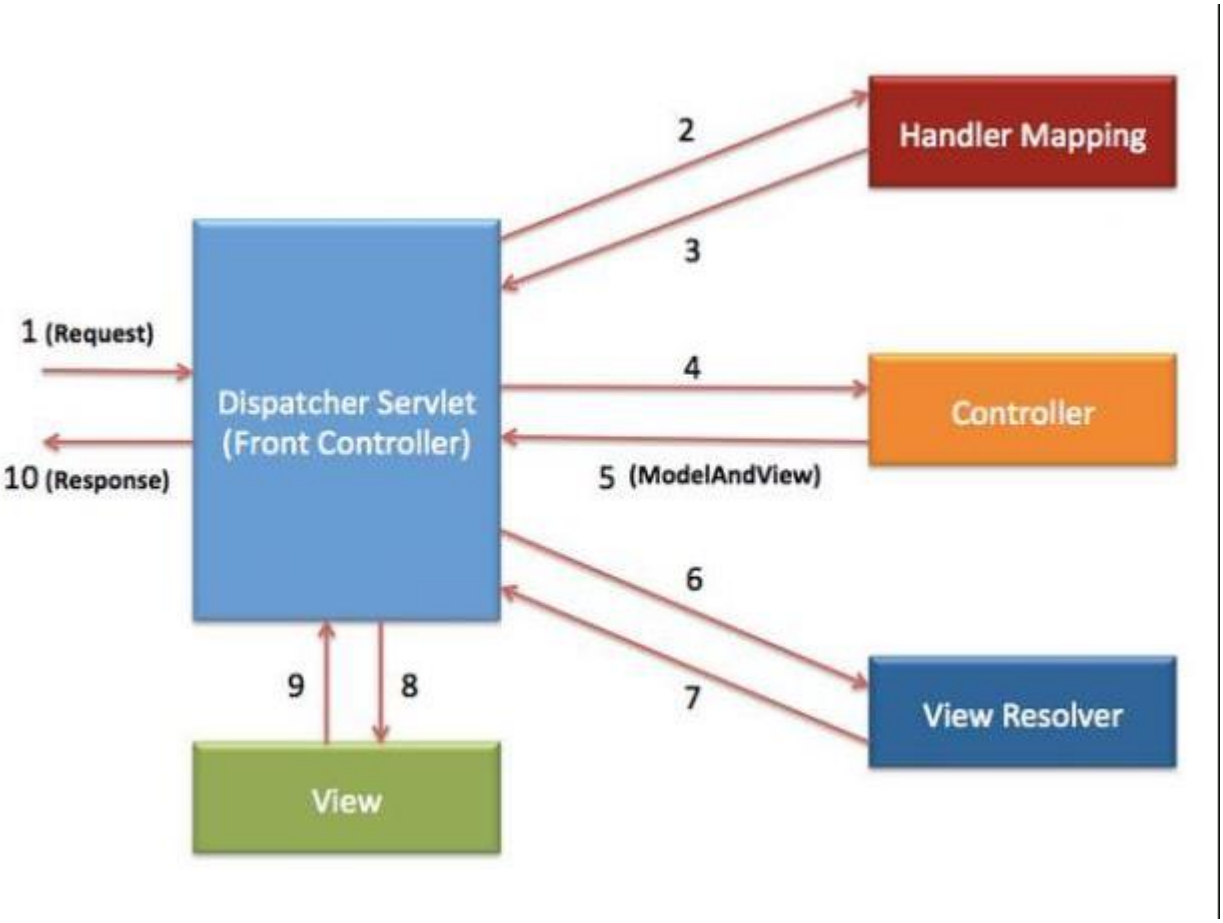


Рисунок 2.5. – Архітектура роботи Spring MVC

2.8. Проектування бази даних

Проектування бази даних є важливим етапом розробки інформаційної системи. Воно включає визначення структури, організації та зв'язків між даними, а також вибір відповідних моделей даних та методів доступу до даних. Ось деякі ключові аспекти проектування бази даних:

1. Аналіз вимог: Спочатку необхідно провести аналіз вимог та зрозуміти, які дані мають бути збережені та як вони будуть використовуватися в системі. Це включає визначення сутностей, їх атрибутів та зв'язків між ними.

2. Вибір моделі даних: На основі вимог та аналізу, необхідно вибрати відповідну модель даних. Найбільш поширеними моделями даних є реляційна модель, ієрархічна модель, мережна модель та об'єктно-орієнтована модель. Реляційна модель найчастіше використовується у сучасних інформаційних системах.

3. Нормалізація даних: Нормалізація даних є процесом організації даних у базі даних для усунення надмірності та забезпечення ефективного зберігання та обробки даних. Нормалізація дотримується певних правил (наприклад, правил нормальних форм), які допомагають усунути дублювання інформації та забезпечити цілісність даних.

4. Визначення схеми бази даних: Схема бази даних визначає структуру таблиць, їх полів та зв'язків. Необхідно визначити таблиці для кожної сутності та атрибути, які зберігатимуться в кожній таблиці. Також важливо визначити первинні та зовнішні ключі для встановлення зв'язків між таблицями.

5. Фізичне проектування: Фізичне проектування бази даних включає вибір конкретних технологій та інструментів для реалізації бази даних. Це може включати вибір СУБД (системи управління базами даних), визначення типів даних, індексування та оптимізацію запитів для забезпечення ефективності та продуктивності роботи з базою даних.

6. Забезпечення безпеки та цілісності: При проектуванні бази даних необхідно врахувати заходи безпеки та цілісності даних. Це може містити визначення прав доступу, шифрування, аудит дій, контроль цілісності даних та інші механізми захисту даних.

7. Індексування та оптимізація: Для забезпечення швидкого доступу до даних та оптимізації продуктивності запитів часто застосовується індексування. Індекси дозволяють прискорити пошук та сортування даних у базі даних. Необхідно проаналізувати типи запитів, які виконуватимуться, та створити відповідні індекси для оптимальної продуктивності системи.

8. Резервне копіювання та відновлення: Щоб забезпечити збереження даних та можливість відновлення у разі збоїв або втрати даних, необхідно

розробити стратегію резервного копіювання бази даних. Це може включати регулярне створення резервних копій та тестування процедур відновлення.

9. Масштабованість: Під час проектування бази даних слід враховувати можливість масштабування. Це може бути вертикальне масштабування (збільшення ресурсів на одному сервері) або горизонтальне масштабування (розподіл даних та навантаження на декілька серверів). Необхідно вибрати відповідні методи масштабування залежно від потреб системи.

10. Документація та супровід: Важливим аспектом проектування бази даних є документування її структури, правил та особливостей. Це допоможе розробникам, адміністраторам та іншим зацікавленим сторонам зрозуміти та підтримувати базу даних.

Проектування бази даних є складним та відповідальним процесом, який вимагає врахування безлічі факторів та дотримання принципів надійності, ефективності та безпеки. Ретельне проектування бази даних із самого початку дозволяє створити стабільну та продуктивну систему зберігання та доступу до даних.[21]

2.9. Етап інфологічного проектування

Інфологічна модель є одним з етапів процесу проектування бази даних і відображає логіку даних, що використовуються в системі. Вона описує сутності (сутності), їх атрибути (атрибути) і взаємозв’язки (відносини) між сутностями без прив’язки до конкретної бази даних фізичної реалізації.

Основна мета інфологічної моделі - налаштування, які сутності необхідні для представлення даних у системі, які атрибути відображають характеристики цих сутностей і які взаємозв'язки між сутностями.

Для побудови інфологічної моделі використані такі основні елементи:

1. Сутність (Entity): Відображає конкретний об'єкт або поняття, про яке збираються дані. Наприклад, можуть бути сутності "Клієнт", "Замовлення", "Товар" і т.д. д. Кожна сутність має свій унікальний ідентифікатор і набір атрибутів, які описують його характеристики.

2. Атрибут (Attribute): Відображає конкретну характеристику або властивість сутності. Наприклад, атрибути сутності "Клієнт" можуть включати "Ім'я", "Прізвище", "Адреса" та ін. д. Кожен атрибут має свій тип даних і обмеження.

3. Взаємозв'язок (Relationship): Відображає зв'язок між двома або більше сутностями. Наприклад, взаємозв'язок "Має" може існувати між сутностями "Клієнт" і "Замовлення", що вказує на те, що один клієнт може мати багато замовлень. Взаємозв'язок може бути один до одного, один до багатьох або багато добавок, у залежності від характеру відношення між сутностями.

Інфологічна модель часто представлена у вигляді схеми або діаграми, де сутності відображаються у вигляді прямокутників, атрибути - як овали, а взаємозв'язки - як стрілки, що з'єднують сутності. Діаграма інфологічної моделі надає видиме виявлення про структуру і зв’язки даних у системі.

Інфологічна модель є інструментом для розуміння домену даних і визначення вимог до бази даних. Вона уникнути дублювання даних, забезпечить узгодженість даних і полегшить подальший процес фізичного проектування бази даних.

На основі інфологічної моделі можна розробити фізичну модель, яка створює спосіб збереження фізичних даних у базах даних, включаючи структуру таблиці, індекси, обмеження цілісності та інші аспекти (рис 2.6)

Узагальнюючи, інфологічна модель є етапом проектування бази даних, що визначає структуру та організацію даних у системі. Вона служить основою для подальшого фізичного проектування та імплементації бази даних.

У представленій базі даних виділяються наступні основні сутності:

1. Договор: Ця сутність представляє інформацію про страховий договір, який укладено між страховою компанією та клієнтом. Вона зберігає дані про страховий поліс, дату дії договору, страхову суму та інші деталі, пов'язані з договором.

2. Користувач: Ця сутність містить інформацію про користувачів системи, що оформляють страхові поліси. Вона зберігає особисті дані клієнтів, такі

як ім'я, контактні дані, адреси тощо. Крім того, вона може містити дані про реєстрацію та авторизацію користувачів.

3. Замовлення: Ця сутність відображає інформацію про замовлення онлайн-продуктів страхової компанії. Вона зберігає дані про обрані страхові пакети, додаткові опції, вартість замовлення та інші деталі, пов'язані з процесом оформлення страхового полісу.

4. Конфігурація онлайн-продукту: Ця сутність містить основну інформацію та налаштування, що стосуються онлайн-продукту страхової компанії. Вона може включати дані про доступні страхові пакети, умови страхування, тарифи, обмеження та інші параметри, що визначають характеристики продукту.

Ці сутності мають взаємозв'язки між собою. Наприклад, договір може мати зв'язок з користувачем, який уклав договір, та замовленням, що стосується цього договору. Крім того, конфігурація онлайн-продукту може бути пов'язана із замовленням для викоронання вибору відповідного продукту. Зв'язки між цими сутностями допомагають зберігати та відстежувати взаємозв'язану інформацію в базі даних.

Важливо враховувати, що це лише основні сутності, і в базі даних можуть бути інші таблиці та зв'язки, які деталізують додаткові аспекти роботи страхової компанії та веб-віджету онлайн-страхування.

Правильна організація бази даних з урахуванням вищезазначених сутностей дозволить ефективно зберігати, керувати та використовувати дані, що стосуються договорів, користувачів, замовлень та конфігурації продукту. Це сприятиме оптимізації роботи страхової компанії, покращенню процесу оформлення страхових полісів та наданню зручного та функціонального веб-віджету для клієнтів.

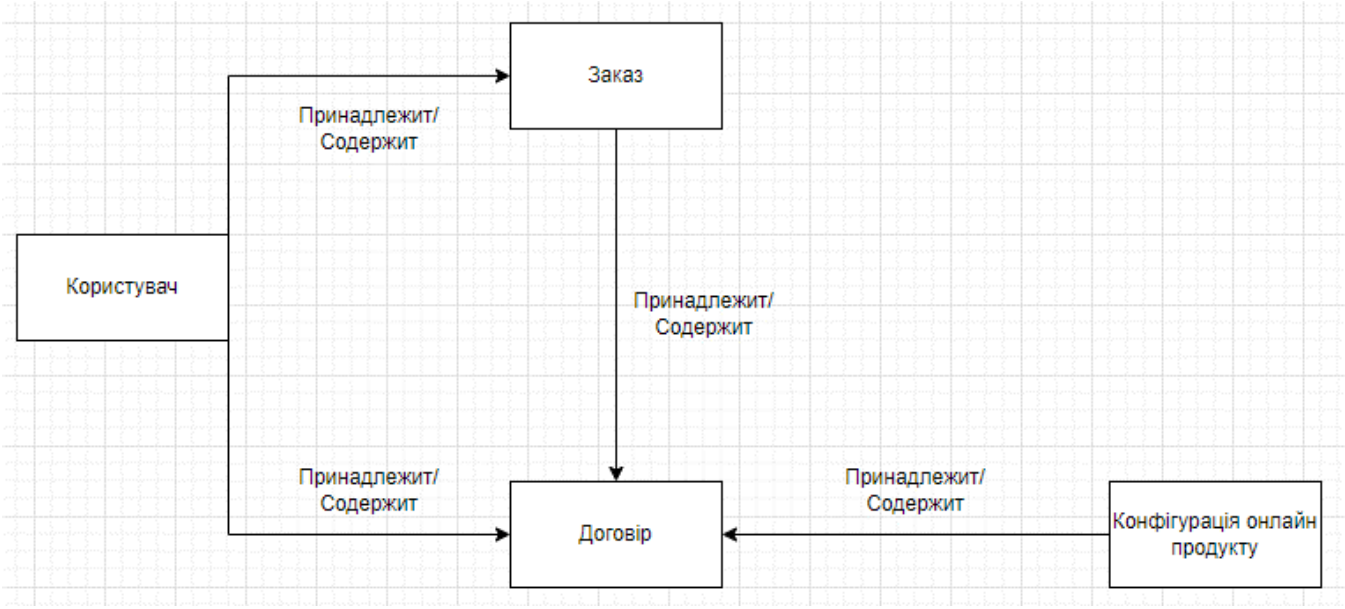


Рисунок 2.6. – Інфологічна модель предметної області

Додаткові сутності, які ви вказали, відіграють важливу роль у керуванні текстовими повідомленнями, що використовують у системі.

Сутність "Ресурс повідомлень" зберігає ключі, за якими можна отримати текстові ресурси, такі як повідомлення про помилки, підказки або інші повідомлення, які відображаються у веб-віджеті. Це дозволяє зручно керувати текстовими ресурсами, внести зміни та оновлювати їх без необхідності внесення змін до вихідного коду програми.

Сутність "Локалізований ресурс повідомлень" містить назви ресурсів у різних мовних версіях або локалях. Це дозволяє системі підтримувати багатомовність та забезпечувати відображення текстів у відповідній мові в залежності від обраної локалізації користувача. Локалізований ресурс повідомлень забезпечує зручну та ефективну роботу зі зміною мови в інтерфейсі, забезпечує зручність використання системи для користувачів з різних країн та культур.

Ці допоміжні сутності доповнюють основну базу даних та дозволяють ефективно керувати та оновлювати текстові ресурси, а також забезпечують можливість локалізації системи для різних мовних версій. Вони сприяють покращенню зручності використання веб-віджету для користувачів з різних країн та культур, а також забезпечують збереження консистентності та якості текстових повідомлень в системі.

2.10. Етап датологічного та фізичного проектування

Етап датологічного проектування є одним із ключових етапів процесу проектування бази даних і передає фізичне проектування. На цьому етапі запускаються конкретні деталі структури даних, такі як таблиці, поля, типи даних, обмеження цілісності та інші атрибути бази даних.

Після опису сутностей і установки зв'язків отримаємо наступну даталогічну модель основної бізнес-логіки (рис. 2.8 – 2.9)

Та даталогічна модель допоміжних таблиць(рис. 2.7).

Рисунок 2.7. – Даталогічна модель допоміжних таблиць

Рисунок 2.8. – Даталогічна модель основних сутностей

Рисунок 2.8. – Даталогічна модель основних сутностей

2.11 Етап фізичного проектування

Розробка фізичної моделі даних є важливим етапом у процесі створення бази даних. На цьому етапі зв'язки, що були сформовані на логічному рівні, перетворюються в таблиці, а атрибути стають стовпцями цих таблиць. У реляційних СУБД для ключових атрибутів створюються унікальні індекси, а домени перетворюються в типи даних, прийняті в конкретній СУБД.

На рівні фізичної моделі даних також реалізуються обмеження, які були визначені на логічному рівні. Це можуть бути індекси, декларативні обмеження цілісності, тригери, збережені процедури та інші засоби, що надають можливість забезпечити цілісність даних. Вибір конкретних засобів залежить від можливостей та функціональності конкретної СУБД.

При розробці фізичної моделі даних виникає багато питань, таких як ефективність таблиць, вибір індексів та необхідність програмного коду для забезпечення цілісності даних. Важливо ретельно перевірити, чи правильно організовані таблиці, чи вибрані оптимальні індекси та який обсяг програмного коду необхідний для забезпечення цілісності даних.

Остаточним результатом фізичного проектування є сама база даних, яка реалізована на певній програмно-апаратній платформі. Вибір цієї платформи може суттєво вплинути на продуктивність роботи з базою даних. Наприклад, можна вибирати різні типи комп'ютерів, змінювати конфігурацію апаратної частини, встановлювати певну кількість процесорів та оперативної пам'яті, налаштовувати дискові підсистеми і так далі. Крім того, важлива є конфігурація самої СУБД на обраній програмно-апаратній платформі.

Таким чином, рішення, які приймаються на кожному етапі моделювання та розробки бази даних, мають велике значення для подальших етапів. Тому важливо правильно приймати рішення ще на ранніх стадіях моделювання, щоб забезпечити оптимальну продуктивність та цілісність даних.

Дані кожної таблиці бази данних предоставлені в додатку А.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСОБУ

3.1. Середовище розробки

IntelliJ IDEA — це інтегроване середовище розробки (Integrated Development Environment, IDE) для мови програмування Java, розробленої компанією JetBrains. Воно надає широкий набір інструментів та функціональності, що полегшують розробку Java-додатків.

Основні особливості IntelliJ IDEA для розробки Java включають:

1. Редактор коду: Має потужні функції редактора, такі як автодоповнення, перевірка синтаксису, форматування коду та розумні рекомендації, які сприяють швидкому та точному написанню коду.

2. Підтримка фреймворків: IntelliJ IDEA підтримує різні популярні Java-фреймворки, такі як Spring, Hibernate, JavaFX та інші. Вона надає інструменти для автоматичної настройки та розробки проектів на основі цих фреймворків.

3. Відладка: IntelliJ IDEA має вбудовану підтримку відладки, яка дозволяє крокувати по коду, ставити точки зупинки, переглядати значення змінених та аналізувати стек викликів для виявлення помилок і проблем у програмі.

4. Управління залежностями: Інтегрована система управління залежностями, як Maven або Gradle, дозволяє зручно додавати та керувати залежностями в проектах.

5. Підтримка контролю версій: IntelliJ IDEA підтримує різні системи контролю версій, такі як Git, SVN та Mercurial. Вона надає інструменти для комітів, відгалужень, злиття гілок та інші операції, пов’язані з управлінням версіями.

6. Автоматичний рефакторинг: IntelliJ IDEA має набір функцій автоматичного рефакторінгу, які можуть змінити структуру коду без зміни його функціональності. Це включає зміну змінних та методів, вилучення повторюваного коду, оптимізацію імпорту та багато іншого.

7. Аналіз коду: IntelliJ IDEA надає потужні інструменти аналізу коду, які допомагають виявити ваші проблеми, недоліки та некоректне використання API. Це ідеально вдосконалити якість коду та забезпечити його надійність.

8. Підтримка інших версій Java: IntelliJ IDEA підтримує різні версії Java, включаючи Java 8, 9, 10, 11 та більш нові. Вона надає можливості рефакторінгу, автодоповнення та інші функції, специфічні для кожної версії.

9. Інтеграція з іншими інструментами: IntelliJ IDEA може бути легко інтегрована з іншими інструментами розробки, такими як середовище виконання сервера (наприклад, Apache Tomcat), системи збірки (наприклад, Maven або Gradle) та інші засоби, що полегшують процес розробки.

10. Розширення та плагіни: IntelliJ IDEA надає можливість розширення функціональності за допомогою плагінів. Користувачі можуть встановлювати плагіни, які надають підтримку іншим мовам програмування, фреймворків або інструментів.

IntelliJ IDEA є потужним інструментом для розробки Java-додатків, який допоможе підвищити продуктивність розробника, полегшує написання якісного коду та забезпечує широкі можливості для управління проектами. Використання IntelliJ IDEA допоможе ефективній розробці програмного забезпечення на Java.

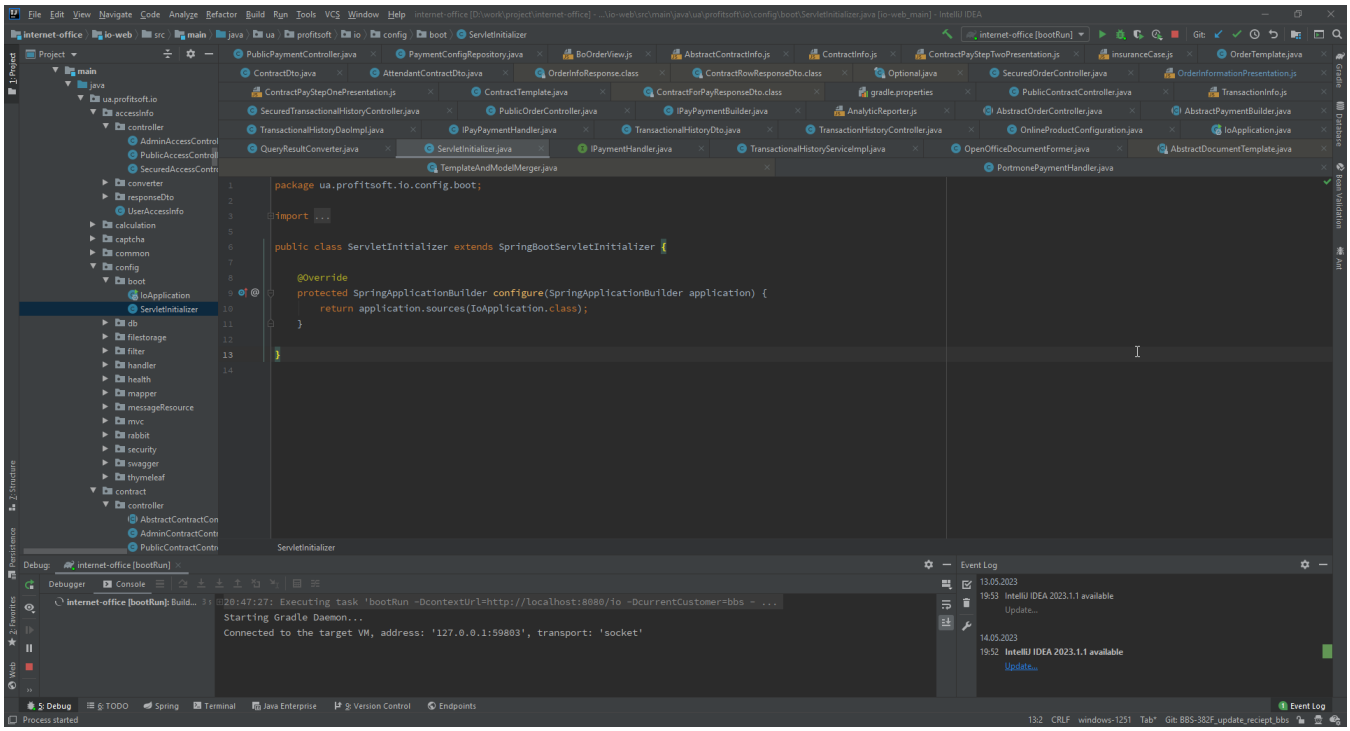


Рисунок 3.1. – Інтерфейс IntelliJ IDEA

Для початку роботи з проектом необхідно налаштувати середовище розробки з використанням Gradle версії 4.7 та JDK Java 8. Вибір саме цих старіших версій обумовлений їх стабільністю та сумісністю з наявними проектами і веб-сайтами, в які будуть впроваджуватись віджети.

Gradle є потужним інструментом для автоматизації збирання, тестування та залежностей проекту. Версія 4.7 вже доведена часом та використовується широкою спільнотою розробників.

Також, для розробки з використанням Java, вам потрібно встановити JDK версії 8. Ця версія є стабільною та має широку підтримку серед розробників. JDK забезпечує набір інструментів для розробки Java-програм, включаючи компілятор, відлагоджувач та інші необхідні компоненти.

Використання старіших версій Gradle та JDK дозволяє забезпечити сумісність інструментів розробки з існуючими проектами та забезпечити стабільну та надійну роботу з ними.[22]

3.2. Опис призначеного для користувача інтерфейсу

На стартовій сторінці веб-віджета ОСАГО, вгорі видно панель прогресу, яка показує поточний етап процесу оформлення. Далі є дві вкладки за замовчуванням: "За державним номером" і "За параметрами авто".

Оформлення за допомогою введення державного номера передбачає введення номера транспортного засобу, щоб здійснити запит до МТСБУ для автоматичного завантаження даних про транспортний засіб та автоматичного заповнення відповідних полів. Також видно поле "Місце реєстрації власника авто", яке визначає параметр "Зона", що впливає на розрахунок вартості страхового поліса. Існують 5 зон, включаючи 4 зони для поділу частин України і 5-ту зону для власників автомобілів з-за кордону.

Після цього ми переходимо на вкладку "За параметрами авто", де вже заповнені основні поля, що впливають на розрахунок страхового поліса. Другий варіант оформлення передбачає безпосередньо початок з вкладки "За параметрами авто", де вже відбувається ручне введення даних.

Така структура веб-віджета дозволяє користувачам обирати зручний спосіб оформлення страхового поліса, використовуючи або автоматичне заповнення за допомогою державного номера, або ручний ввід параметрів автомобіля. Це забезпечує гнучкість та зручність для користувачів, що дозволяє їм швидко ефективно оформити страховий поліс ОСАГО.

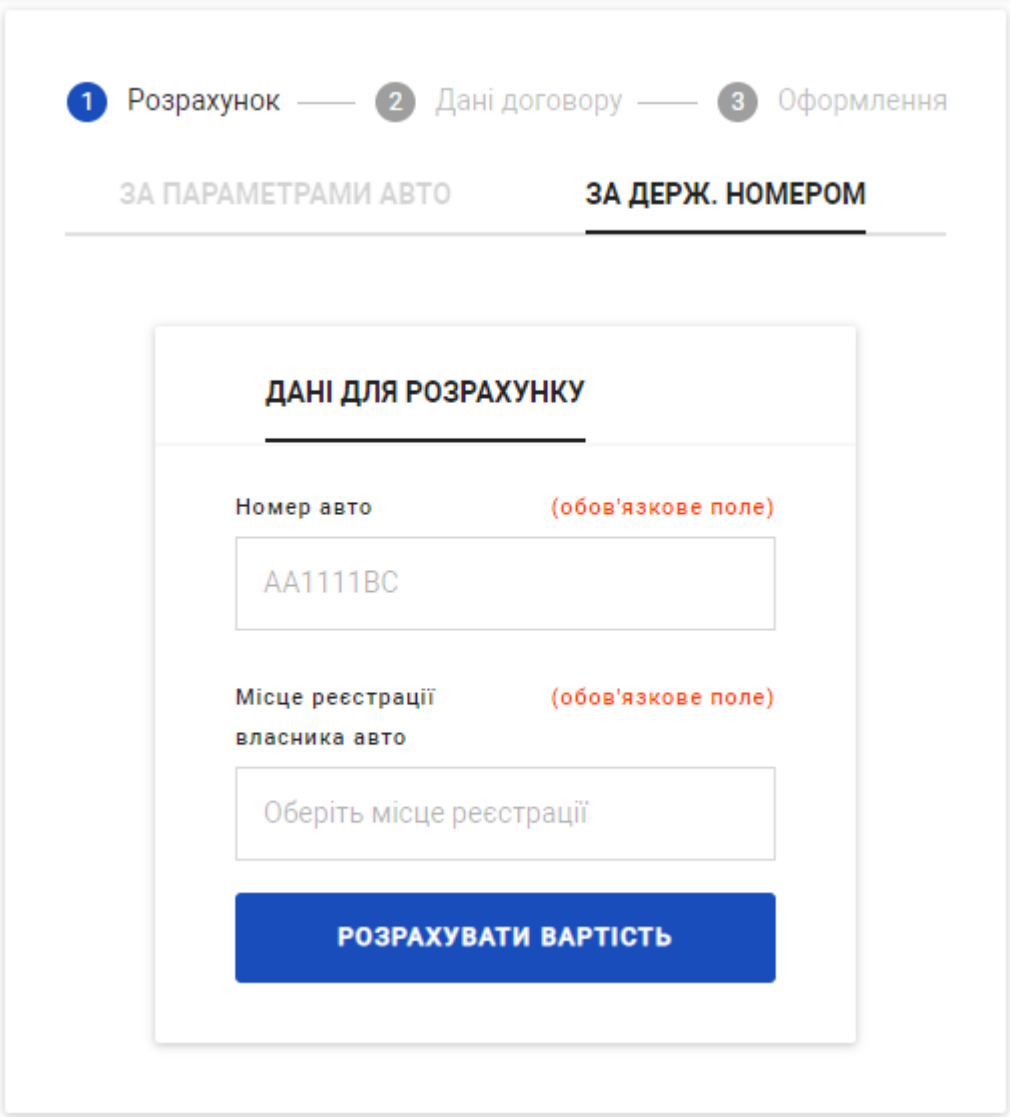


Рисунок 3.2. – Стартова сторінка віджету ОСАГО

На вкладці "За параметрами авто" доступні наступні поля:

"Ваш автомобіль" - поле, в якому вибирається тип транспортного засобу зі спадаючого списку з пропонованими варіантами. Типи транспортних засобів можуть включати легкові автомобілі, вантажівки, автобуси або причепи. Залежно

від вибраного типу, інші поля можуть змінюватися відповідно до вимог, пов'язаних з даним типом транспортного засобу.

"Об'єм двигуна" - для легкових транспортних засобів потрібно вибрати об'єм двигуна зі спадаючого списку з пропонованими варіантами. Для вантажівок, автобусів або причепів це поле може бути замінене на відповідне поле, яке відповідає вимогам до даних типів транспортних засобів.

Вибір та заповнення цих полів на вкладці "За параметрами авто" дозволяє зазначити конкретні характеристики автомобіля, які впливають на розрахунок страхового поліса ОСАГО. Враховуючи тип транспортного засобу і об'єм двигуна, страхова компанія може визначити страховий тариф та вартість поліса згідно з цими параметрами.

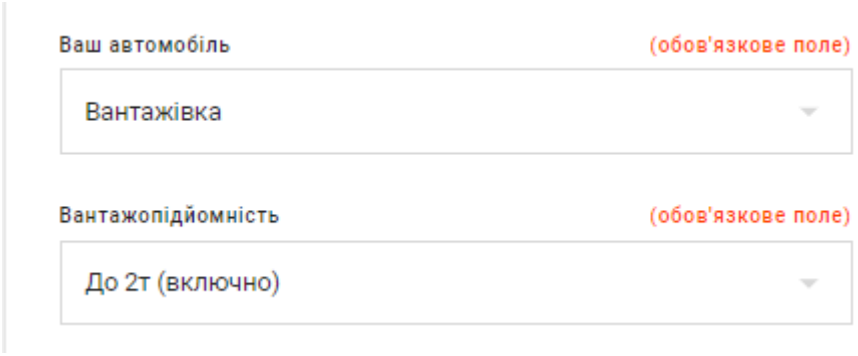


Рис. 3.3. – Конфігурація для вантажівок

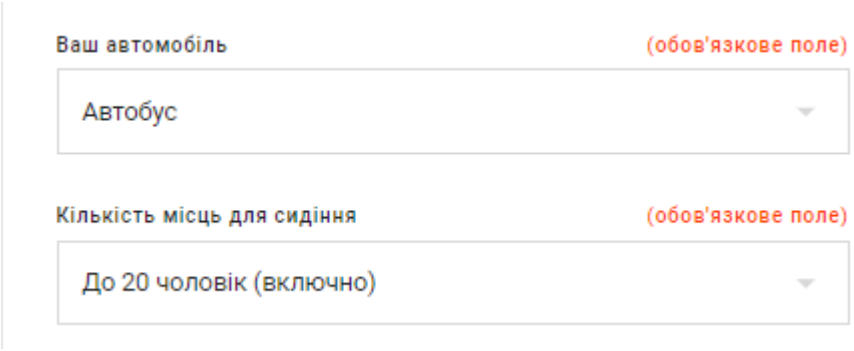


Рисунок 3.4. – Конфігурація для автобусів

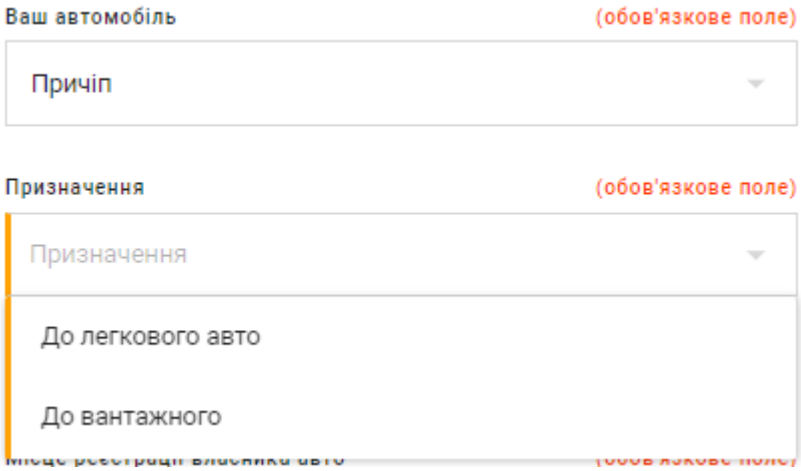


Рисунок 3.5. – Конфігурація для причіпів

"Інша держава" - чекбокс, який потрібно вибрати, якщо страхувальник не є громадянином України. У такому випадку поле "Місце реєстрації власника авто" приховується, і автоматично встановлюється Зона 5 у розрахунку.

"Місце реєстрації транспортного засобу" - потрібно вибрати місто реєстрації, яке впливає на розрахунок страхового поліса. Під час введення місця населеного пункту, система надає список населених пунктів, що відповідають введеним користувачем першим літерам.

"Таксі" - чекбокс, який вказує на наявність додаткової послуги таксі. Це впливає на вартість розрахунку страхового поліса.

"Франшиза" - це поле, в якому необхідно вибрати розмір франшизи для страхового поліса. Франшиза визначає суму, яку страхувальник повинен внести у разі настання страхового випадку перед отриманням страхової виплати від страхової компанії.

У випадку, коли страховий випадок не перевищує встановлену франшизу, страхувальник не отримує страхову виплату, а всю суму збитків сплачує самостійно. Якщо збитки перевищують встановлену франшизу, страхова компанія відшкодовує різницю між сумою збитків і розміром франшизи.

Розмір франшизи може бути різним і залежить від умов страхового поліса. Зазвичай вибір франшизи впливає на вартість страхового поліса: чим вища

франшиза, тим менше вартість поліса. Однак, вибір франшизи також пов'язаний зі збитками, які страхувальник повинен буде внести у разі страхового випадку.

"Водійський стаж" - обов'язкове поле, яке враховує водійний стаж страхувальника і впливає на розрахунок. Якщо страхувальник має недавно отримане посвідчення, що свідчить про обмежений досвід водіння, це враховується у розрахунку страхового поліса.

Після цього видно чекбокс, який пропонує додаткові послуги, такі як крос-продукти. При виборі цього чекбокса відображається вартість даної послуги.

"Крос-продукт" - це додаткова послуга або варіант, який може бути пропонований страховою компанією під час оформлення страхового поліса. Крос-продукт - це додаткові покриття або розширені можливості, які можна додати до основного страхового поліса.

При виборі крос-продукту страхувальник має можливість розширити покриття страхового поліса, отримати додаткові гарантії або додаткові послуги. Наприклад, це може бути додаткове страхування водія та пасажирів, страхування від нещасних випадків, страхування від крадіжки або пошкодження майна, асистенційні послуги на дорозі тощо.

Крос-продукти дають можливість страхувальнику персоналізувати свій страховий поліс, вибрати додаткові захисти або послуги відповідно до своїх потреб і вимог. Вартість крос-продукту може додаватися до загальної вартості страхового поліса, але він може забезпечити додатковий захист та забезпечити більш широкий спектр покриття.

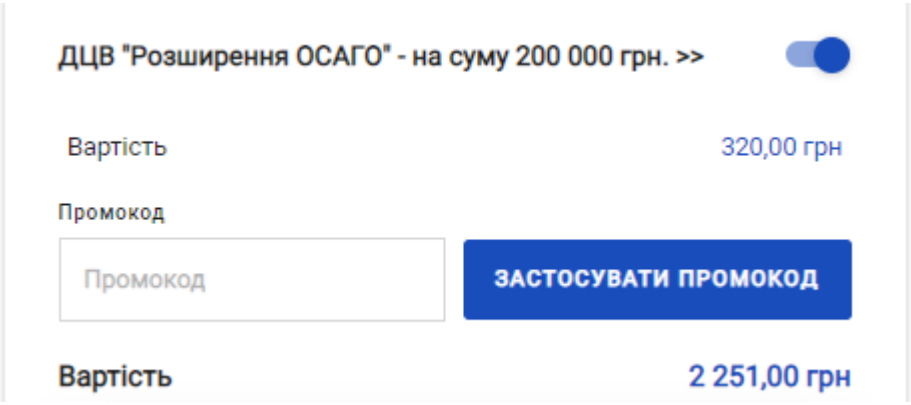


Рисунок 3.6. – Крос-продукти та загальна вартість страхового полісу

Після заповнення полів відображається поле для введення промокоду та повна вартість страхового поліса. Промокод надає можливість отримати знижку або спеціальні умови. Повна вартість страхового поліса відображає суму всіх параметрів, послуг та факторів, що впливають на розрахунок (рис. 3.7)

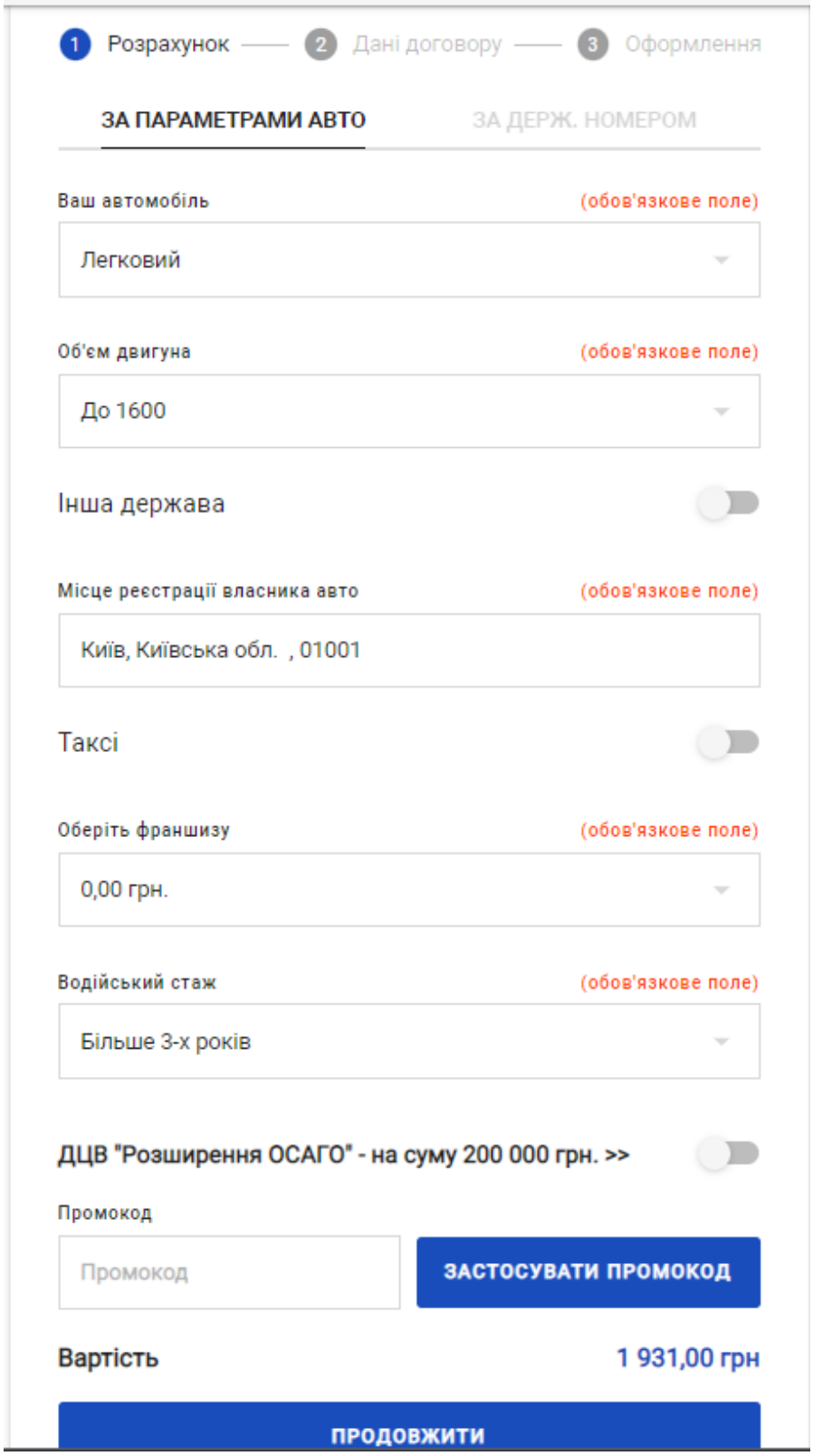


Рисунок 3.7. – Загальний вид першого кроку оформлення страхового полісу

На другому етапі оформлення договору ми вводимо наступні дані у чотирьох вкладках: "Дані по автомобілю", "Особисті дані", "Адреса" та "Дані паспорта".

На вкладці "Дані по автомобілю" ми вводимо інформацію про транспортний засіб. Маємо такі поля:

Марка і модель: поле для введення марки і моделі автомобіля. При введенні випадає список зі справочника за введеними даними.

Номер кузова або VIN номер: поле для введення номера кузова, максимум 17 символів.

Якщо оформлення почалося з введення державного номеру, то дані про транспортний засіб автоматично заповняються.

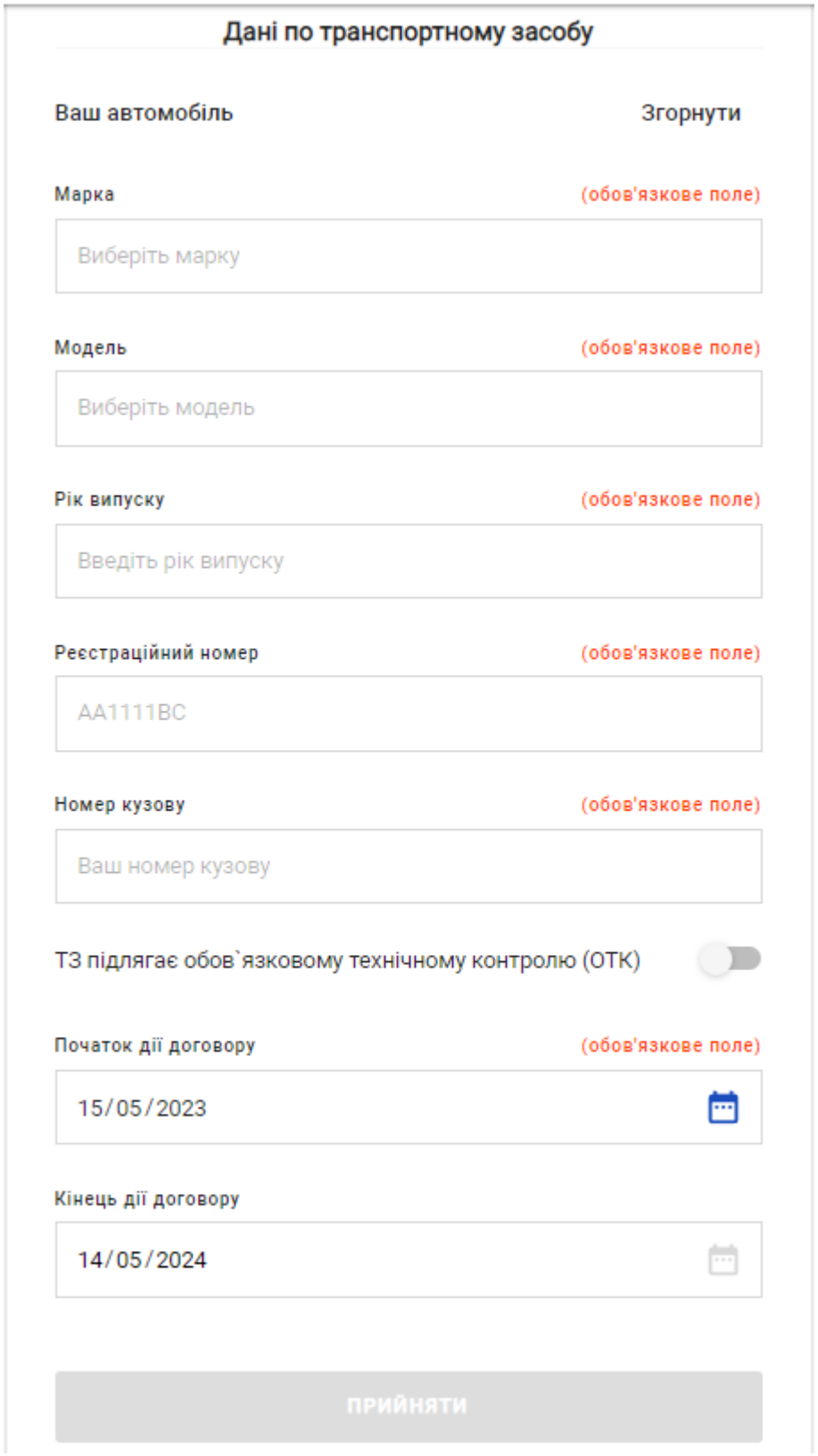


Рисунок 3.8. – Другий крок - "Дані по автомобілю"

У вкладці "Особисті дані" страхувальника заповнюються наступні поля:

Прізвище, Ім'я, По батькові - окремі поля для введення, що дає можливість вказати інформацію без по батькові у разі, якщо страхувальник є іноземцем.

Номер телефону - з вибором коду країни зі списку.

ІПН (Ідентифікаційний податковий номер) - вводиться перед датою народження, оскільки після введення цього поля, дата народження автоматично заповнюється за формулою, що визначає дату народження за введеним ІПН.

Електронна пошта - для отримання інформації про договір, страховий поліс та інші відомості.

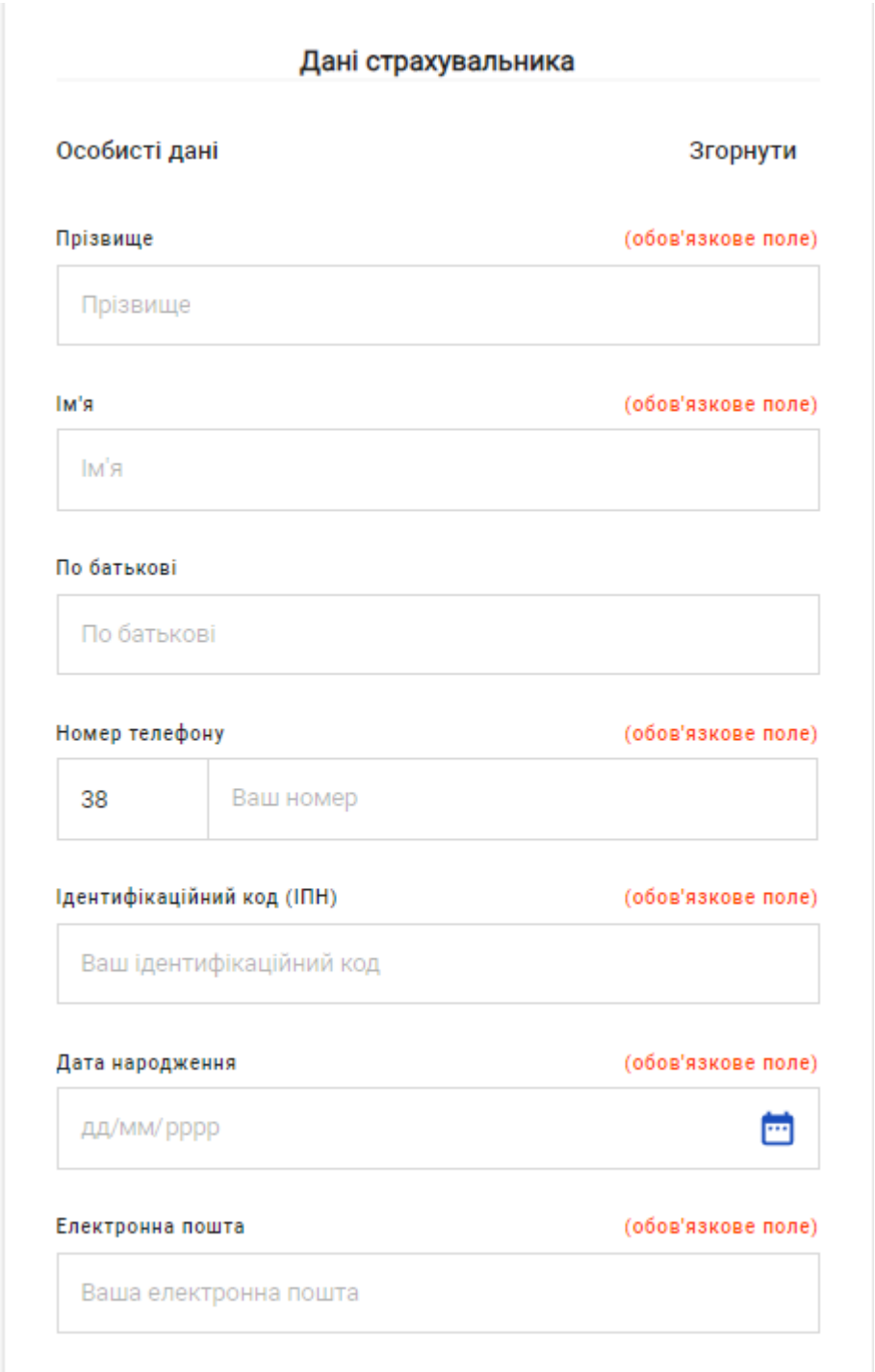


Рисунок 3.9. – Другий крок - "Особисті дані"

На наступній вкладці "Адреса" вводиться адреса страхувальника. Поля, які можна заповнити, включають:

Місто - поле для введення назви міста або населеного пункту.

Вулиця - поле для введення назви вулиці.

Будинок/квартира - поля для введення номера будинку та квартири.

Ці дані використовуються для визначення місцезнаходження страхувальника та можуть бути необхідними для обробки договору страхування.

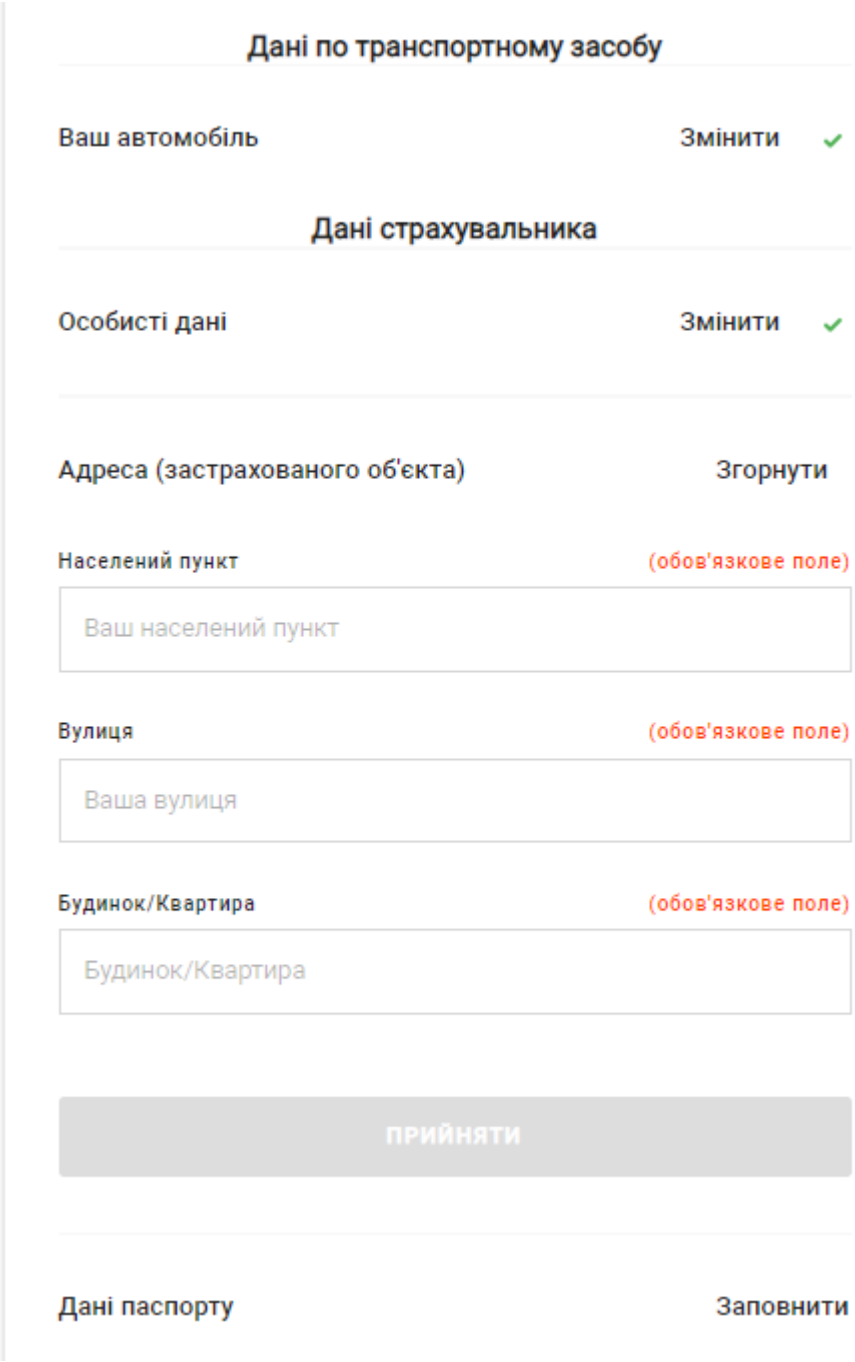


Рисунок 3.10. – Другий крок - "Адреса"

Для данних паспорта наступні поля:

Тип документа - вибір типу документа зі списку (паспорт, ID-карта, водійське посвідчення, закордонний паспорт, пенсійне посвідчення тощо).

Серія та номер документа - введення номера документа, а також серії, якщо її є.

Дата видачі - введення дати видачі документа.

Ким виданий - введення адреси паспортного столу або коду органу, що видав документ.

Ці дані використовуються для ідентифікації страхувальника та оформлення договору страхування.

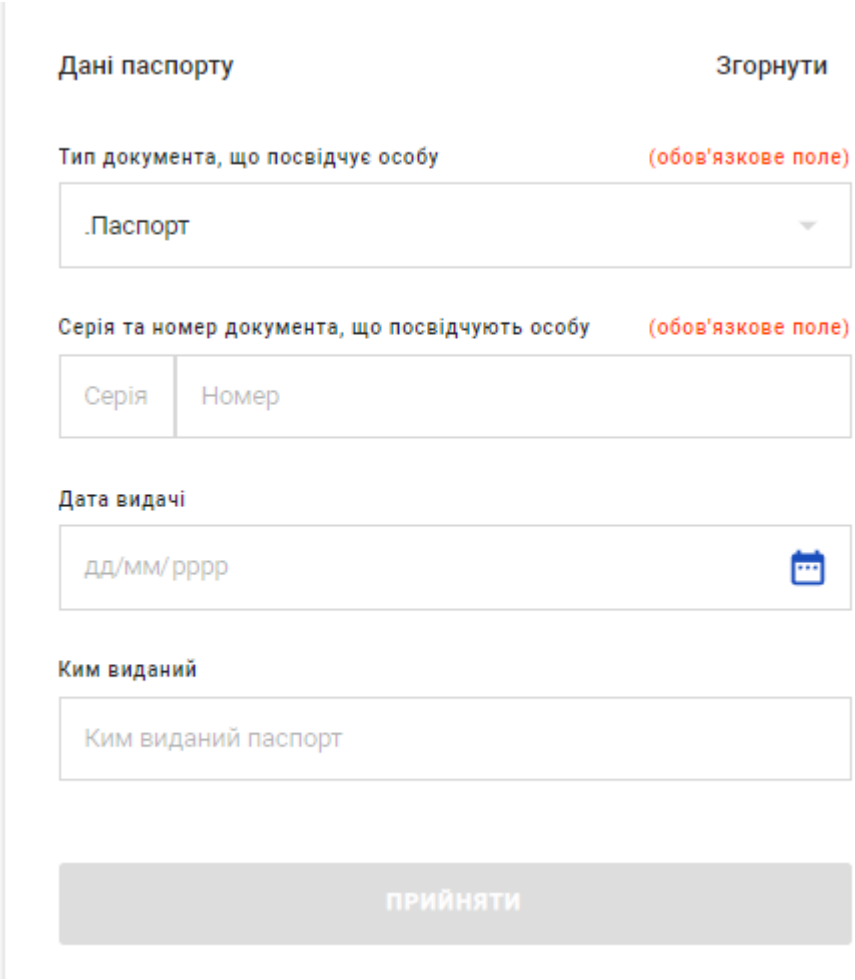


Рисунок 3.11. – Другий крок - "Дані паспорту"

На наступному етапі переходимо до оформлення та оплати страхового поліса. Перевіряємо раніше введені дані та вибираємо спосіб доставки інформації про поліс, який автоматично підставляється з раніше вказаної електронної адреси. Після підтвердження капчі переходимо безпосередньо до оформлення та оплати.

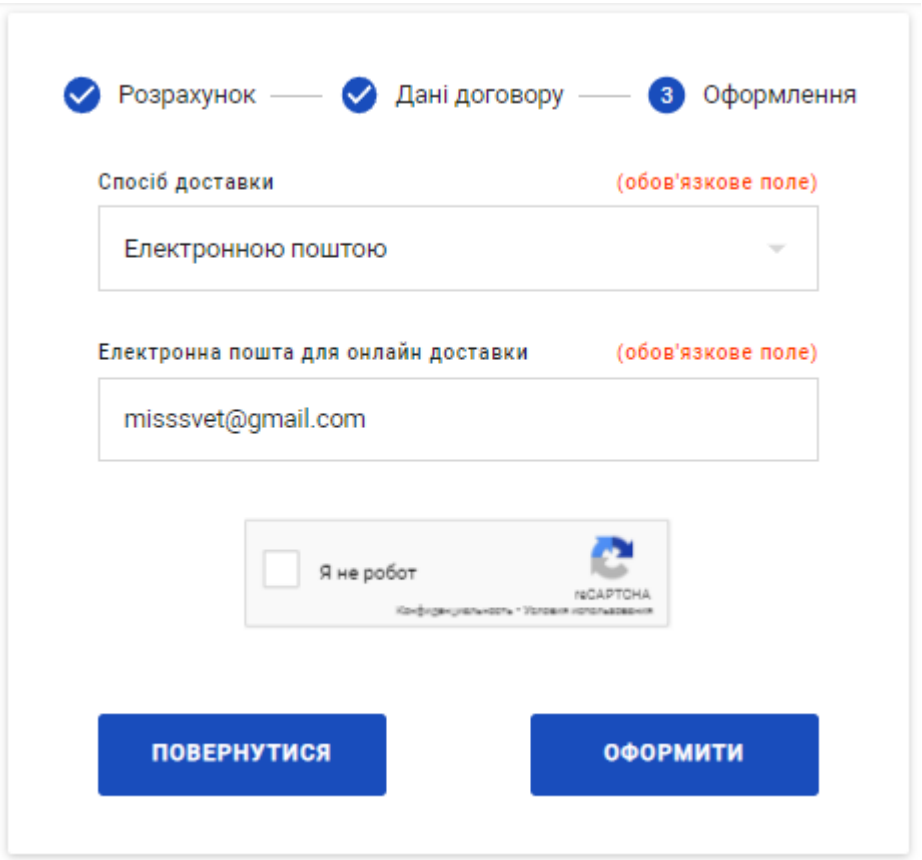


Рисунок 3.12. – Третій крок – оформлення

Якщо під час заповнення данних виникли помилки, система повідомляє про це, і оформлення страхового поліса завершується. На цьому етапі відправляється повідомлення відповідальній особі в страховій компанії про незавершене оформлення страхового поліса, і вона зв'язується зі страхувальником для продовження оформлення страхового поліса.

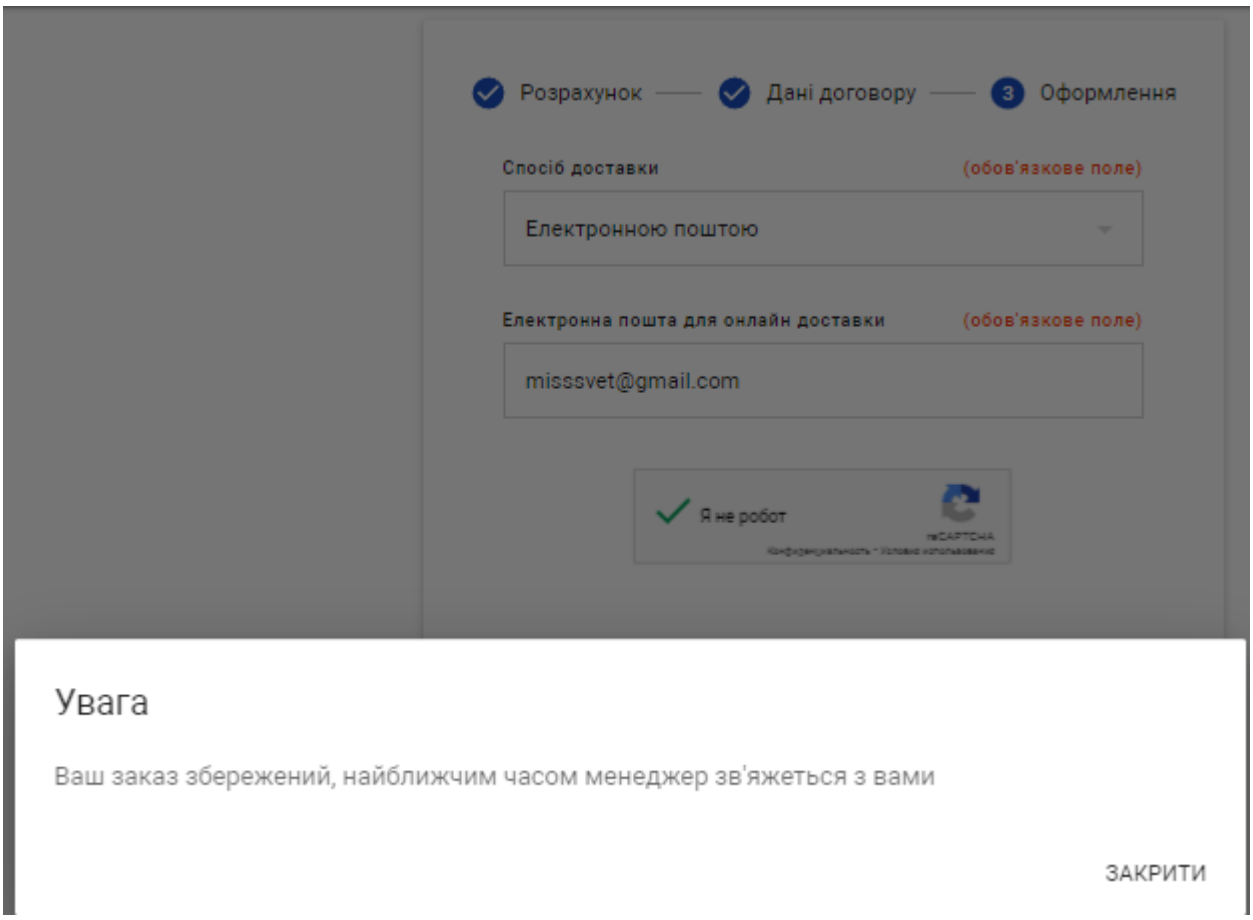


Рисунок 3.13. – Модальне вікно у разі помилок при оформлені страхового полісу

Якщо помилок немає, ви переходите до підтвердження страхового поліса через SMS-код, який автоматично надсилається на номер телефону, який був вказаний раніше при заповненні даних страхувальника. На данній сторінці ви повинні буде ввести отриманий код для завершення оформлення страхового поліса. Це додатковий етап підтвердження вашої особи та забезпечення безпеки та конфіденційності процесу оформлення. Після введення правильного SMS-коду ви переходите до завершення оформлення та отримання страхового поліса.

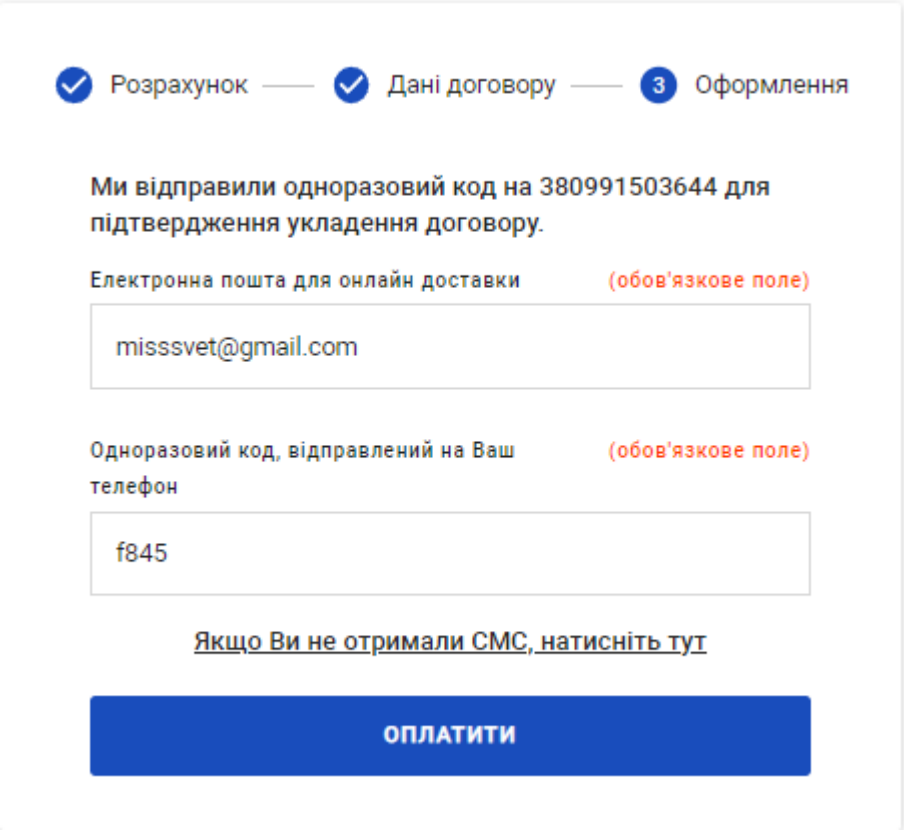


Рисунок 3.13. – Підтвердження за допомогою SMS-коду

Після успішного підтвердження з'являється модальне вікно зі списком доступних платіжних систем, через які можна оплатити страховий поліс. Після вибору платіжної системи ви переходите на веб-сайт платіжної системи для здійснення оплати. На цьому етапі ви зможете ввести необхідні дані для оплати, такі як реквізити картки, дані рахунку або будь-які інші вимоги платіжної системи. Після успішної оплати вам буде надіслане підтвердження оплати, а також ви отримаєте страховий поліс або іншу відповідну інформацію на електронну пошту або у зручний для вас спосіб.

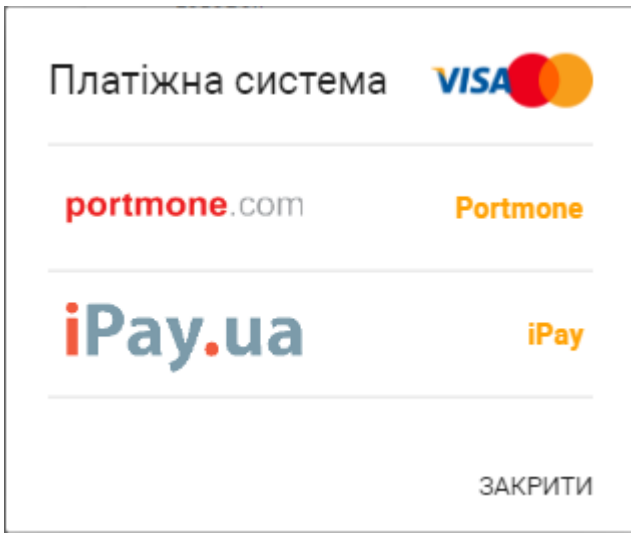


Рисунок 3.14. – Модальне вікно із списком платіжних систем

Після успішної оплати на веб-сайті платіжної системи відбувається перенаправлення до окремого віджету "Квитанція". В цьому віджеті ви можете переглянути квитанцію про здійснену оплату, яка містить інформацію про страховий поліс, дату та суму оплати, а також інші відповідні дані. Якщо оплата не пройшла успішно, ви побачите сторінку платіжної системи, де буде повідомлено про невдалу спробу оплати, і перенаправлення не відбудеться.

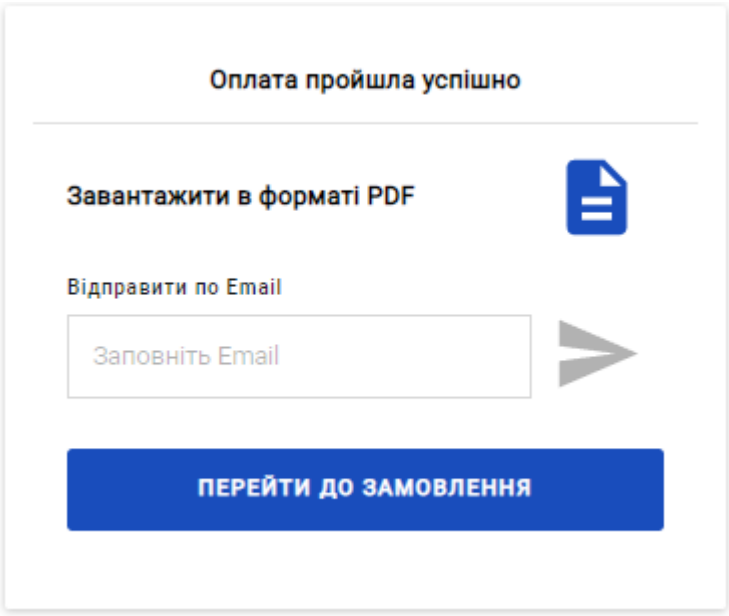
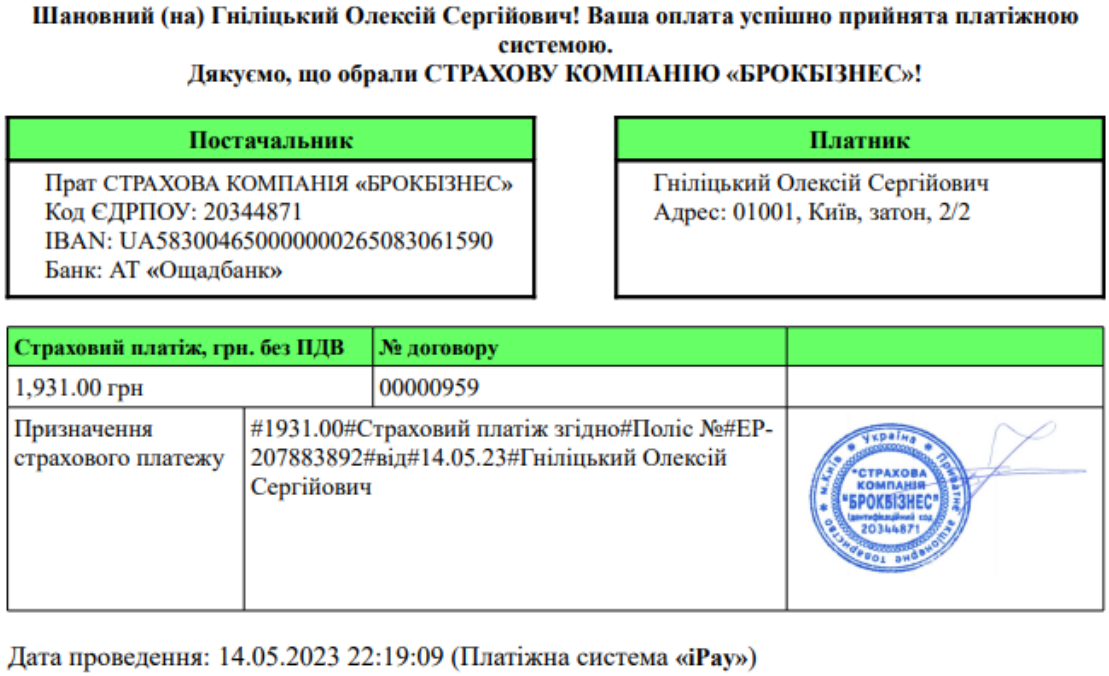


Рисунок 3.15. – Віджет "Квитанція"

На віджеті "Квитанція" ви побачите підтвердження успішної оплати, а також можливість завантажити квитанцію, що містить інформацію про проведену

транзакцію. Ви можете вибрати опцію завантаження квитанції, щоб зберегти її на своєму пристрої. Крім того, є можливість відправити квитанцію на електронну адресу, яку ви вказали під час заповнення даних страхователя. Таким чином, ви матимете можливість мати квитанцію з інформацією про успішно проведену оплату в електронному вигляді.



Шановний (на) Гніліцький Олексій Сергійович! Ваша оплата успішно прийнята платіжною системою.
Дякуємо, що обрали СТРАХОВУ КОМПАНІЮ «БРОКБІЗНЕС»!

Постачальник	Платник
Прат СТРАХОВА КОМПАНІЯ «БРОКБІЗНЕС» Код ЄДРПОУ: 20344871 IBAN: UA583004650000000265083061590 Банк: АТ «Ощадбанк»	Гніліцький Олексій Сергійович Адрес: 01001, Київ, затон, 2/2

Страховий платіж, грн. без ПДВ	№ договору	
1,931.00 грн	00000959	
Призначення страхового платежу	#1931.00#Страховий платіж згідно#Поліс №#ЕР-207883892#від#14.05.23#Гніліцький Олексій Сергійович	

Дата проведення: 14.05.2023 22:19:09 (Платіжна система «iPay»)

Рисунок 3.16. – Квітанція в форматі PDF

3.3. Тестування

Тестування інформаційної системи є першим етапом у процесі розробки, що дозволяє відзначити функціональність, якість та надійність системи перед її впровадженням. Основна мета тестування - виявлення помилок, дефектів та недоліків, що дозволяє їх виправити перед тим, як система буде використовуватися реальними користувачами.

Основні види тестування інформаційної системи включають:

1. Функціональне тестування: перевірка відповідності системи функціональним вимогам. Тестування з оплати переконатися, що система виконує очікувані функції.

2. Навантажувальне тестування: вимірювання та перевірка продуктивності системи за умов навантаження, які наближаються до реального

використання. Це дозволяє визначити проблеми з продуктивністю, масштабованістю та завантаженням системи.

3. Тестування безпеки: перевірка системи на вразливість та можливість несанкціонованого доступу. Тестування цього для забезпечення захисту конфіденційної інформації та недопущення пошкодженої безпеки.

4. Тестування користувальницького інтерфейсу: перевірка зручності використання та ергономіки інтерфейсу системи. Тестування сайту з забезпеченням зручності роботи для користувачів.

5. Тестування відновлення: перевірка здатності системи до відновлення після збоїв або випадкових помилок. Тестування цього року для впевненості в надійності системи та її здатності до самовідновлення.

6. Інтеграційне тестування: перевірка взаб. Інтеграційне тестування: перевірка взаємодії між компонентами системи для виявлення можливих проблем у процесі їх спільної роботи. Це дозволяє перевірити, чи працювати окремі модулі системи належним чином під час взаємодії між собою.

7. Тестування на прийнятті: проведення тестів замовником або кінцевим користувачем із зазначенням переконатись, що система відповідає всім вимогам та задовольняє потреби користувачів.

8. Автоматизоване тестування: використання спеціалізованих інструментів та скриптів для автоматичного виконання тестів. Це дозволяє прискорити процес тестування та підвищити його ефективність.

9. Регресійне тестування: перевірка впливу нових змін або виправлень на раніше протестовані функціональні можливості системи. Це дозволяє отримати негативні наслідки внесених змін.

10. Тестування відповідності: перевірка відповідності системи стандартам, регуляторним вимогам або бізнес-правилам. Це може включати також перевірку забезпечення конфіденційності, цільності та доступності даних, а також дотримання вимог безпеки.

Крім того, важливо розробити план тестування, застосувати набір тестових сценаріїв, підготувати тестові дані та налаштувати оточення для проведення

тестування. Після виконання тестів важливо аналізувати отримані результати та виявляти виявлені проблеми перед впровадженням системи в реальному використанні.

Загальною програмою тестування інформаційної системи є забезпечення якості та надійності системи перед її впровадженням. Виконання всіх перерахованих видів тестування допоможе виявити та виправити помилки, дефекти та дефекти системи, що забезпечує її ефективну та стабільну роботу в реальних умовах.

Для успішного тестування системі варто виконати наступні критерії:

1. Читкість вимог: перед початком тестування необхідно мати чіткі та однозначні вимоги до системи. Це допоможе побудувати відповідні тестові сценарії та перевірити, чи відповідає система поставленим вимогам.

2. Повнота покриття: важливо переконатися, що всі основні функції та компоненти системи підлягають тестуванню. Недостатнє покриття може призвести до пропуску наявних проблем.

3. Репрезентативність тестових даних: використання реалістичних тестових даних дозволяє більш точно відтворити реальні умови використання системи. Це все виявити проблеми та недоліки, які можуть виникнути в реальному середовищі.

4. Автоматизація тестування: використання автоматизованих тестових скриптів та інструментів дозволяє прискорити процес тестування, підвищити його ефективність та забезпечити більшу точність результатів.

5. Взаємодія з реальними користувачами: залучення реальних користувачів до процесу тестування і отримання відгуків та пропозицій щодо вдосконалення системи. Це дозволяєЦе дозволяє забезпечити більшу відповідність системи потребам користувачів та покращити її використання.

6. Виявлення та виправлення помилок: весь процес тестування має на меті виявлення помилок, дефектів та дефектів системи. Після виявлення проблеми необхідно внести відповідні виправлення та перевірити, чи вони були успішно усунені.

7. Документування результатів: важливо детально документувати результати тестування, виявлені проблеми та заходи, які були вжиті для їх виправлення. Це дозволяє зберегти інформацію про проведені тести та забезпечити належний контроль якості системи.

8. Повторне тестування: після внесення виправлень необхідно провести повторне тестування, щоб переконатися, що проблеми були успішно усунені і система працює належним чином.[23]

Тестування інформаційної системи - це процес, що вимагає систематичного та цілеспрямованого підходу. зазвичай до типу та масштабу проекту можуть використовуватися різні методи тестування, такі як ручне тестування, автоматизоване тестування, тестування в рамках Agile або DevOps. Вибір методик залежить від конкретних вимог та вимог проекту.

Тестування REST контролерів за допомогою Postman є розширеною практикою для перевірки функціональності та коректності роботи API. Postman - це інструмент для розробки та тестування API, який надає зручні інтерфейси для створення та виконання HTTP-запитів.

Postman також надає можливість автоматизувати тестування за допомогою колекцій тестів та скриптів, які можуть автоматично виконувати запити та перевіряти результати. Це особливо корисно для великих проектів з багатьма контролерами та складними сценаріями тестування.

Таким чином, використання Postman для тестування REST контролерів дозволяє зручно та ефективно тестувати функціональність API. Ви можете перевіряти різні типи запитів, передавати параметри, отримувати та перевіряти відповіді сервера.

Для кожного запиту в Postman ви можете встановити очікувані результати, такі як код статусу, заголовки, вміст відповіді, інформацію про помилки тощо. Це дозволяє вам зробити автоматичну перевірку на відповідність очікуваним результатам та виявлення проблем.

Postman також підтримує збереження тестових сценаріїв та колекцій, що дозволяє повторно використовувати набори тестів та легко виконувати їх. Ви

можете створити різні зміни тестових середовищ, на альтернативні середовища та автоматизувати процес тестування за допомогою скриптів, що робить процес більш гнучким та потужним.

Загалом, використання Postman дозволяє вам ефективно тестувати та перевіряти ваші контролери REST, забезпечуючи надійність та якість вашої системи. Він є потужним інструментом для розробників та QA-спеціалістів, який допоможе забезпечити коректну взаємодію між клієнтом та сервером у вашій інформаційній системі.[24]

JUnit є однією з найпопулярніших бібліотек для тестування програм на Java. Вона надає набір аннотацій та класів для написання та виконання тестів. JUnit дозволяє вам перевірити, чи працює код правильно, і забезпечує автоматизацію тестування.

JUnit допомагає створити структуровані й ефективні тести для інформаційної системи, забезпечуючи надежність, якість та швидкість розробки. Використовуючи JUnit, можна переконатися, що ваш код працює правильно та відповідає вимогам специфікацій.

JUnit є потужним інструментом для тестування Java-програм, який допомагає забезпечити якість і надійність вашого коду. Він дозволяє автоматзувати процес тестування, забезпечуючи швидку зворотну зв'язок і впевненість у правильності роботи вашої програми.

JUnit допомагає автоматизувати процес тестування, полегшує виявлення помилок та покращує якість вашого коду. Він інтегрується з популярними інструментами розробки Java, такими як IntelliJ IDEA та Eclipse, що робить його використання ще більш зручним.

Використовуючи JUnit, ви можете створювати розширені тестові набори, проводити регресі.сійні тести, визначати робочі точки входу для ваших методів та перевіряти, чи виконується очікувана функціональність. Крім того, JUnit дозволяє вам легко організувати та керувати вашими тестовими наборами.[25]

ВИСНОВКИ

У дипломній роботі було проведено дослідження та розробка процесу інтеграції веб-віджетів онлайн страхування з метою оптимізації роботи страхових компаній. Основною метою було покращення доступності та зручності процесу страхування для клієнтів, а також підвищення ефективності та автоматизації роботи страхових компаній.

У ході дослідження було вивчено сучасні тенденції та технології у сфері онлайн-страхування, а також проведено аналіз популярних маркетплейсів та веб-сайтів страхових компаній. Було виявлено, що інтеграція веб-віджетів може значно спростити та прискорити процес страхування для клієнтів, а також забезпечити більш ефективну обробку даних та автоматизацію робочих процесів для страхових компаній.

На основі проведеного аналізу та дослідження було розроблено архітектуру системи, яка передбачає інтеграцію веб-віджетів з існуючими системами страхових компаній. Для реалізації інтеграції були використані популярні технології, такі як Spring Boot, REST API та база даних PostgreSQL. Було розроблено модулі для управління користувачами, продуктами страхування та процесом оформлення полісів.

У процесі розробки було проведено тестування системи з використанням різних підходів, включаючи тестування REST-контролерів за допомогою Postman та модульне тестування з використанням JUnit. Тестування дозволило перевірити коректність роботи системи та виявити виявлені помилки та проблеми, які були виправлені у процесі розробки.

Застосування інтеграції веб-віджетів онлайн страхування виявилося дуже ефективним кроком для оптимізації роботи страхових компаній. Клієнти отримують більш зручний та швидкий доступ до страхових продуктів, а компанії отримують можливість автоматизувати безліч рутинних процесів та прискорити обробку даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Білецька Н. О. Особливості організації роботи страхових компаній в умовах інтернет-середовища // Наукові записки Національного університету "Острозька академія". Серія "Економіка". - 2015. - Том 22. - С. 100-105.
2. Гончаренко, О. (2014). Поведінка споживачів страхових послуг в умовах розвитку електронної комерції. Маркетинг і менеджмент інновацій, 3, 167-173.
3. Загоскіна, М. Фактори, що впливають на конкурентоспроможність страхових компаній / М. Загоскіна, В. Передерій // Економіка. Фінанси. Управління. - 2017. - Т. 5, № 2. - С. 23-29.
4. Карпова, Л. Особливості використання віджетів в страхових компаніях // Економіка та управління підприємствами: зб. наук. праць. Київ: Вид-во Нац. ун-ту "Львівська політехніка", 2018. Вип. 2(30). С. 145-152.
5. Hotline.finance - [Електронний ресурс]. Режим доступу: https://hotline.finance/ua/
6. PZU Insurance - [Електронний ресурс]. Режим доступу: https://www.pzu.com.ua/
7. Богуш В. Методології розробки програмного забезпечення. [Електронний ресурс]. Режим доступу: http://library.kpi.kharkov.ua/ukr/departments/k80/k80_lecture_notes.html
8. Корж Н. Використання гнучких методологій у процесі розробки програмного забезпечення. Збірник наукових праць Національного університету "Львівська політехніка". 2014. Вип. 792. С. 55-61.
9. Москаленко І., Марченко М., Козак Л. Впровадження Scrum у розробку програмного забезпечення. Вісник Київського національного університету імені Тараса Шевченка. Комп'ютерні науки та інформаційні технології. 2016. Вип. 5. С. 47-52.

10. Бас, Л., Клементс, П., Казман, Р. (2014). Архітектура програмного забезпечення: практика створення та використання. Київ: Видавництво "Діафрагма".

11. Oracle Java Documentation - [Електронний ресурс]. Режим доступу: https://docs.oracle.com/en/java/

12. Офіційна документація Spring Framework: Офіційний сайт Spring Framework ([Електронний ресурс]). Режим доступу: https://spring.io

13. React Legacy Documentation - [Електронний ресурс]. Режим доступу: https://legacy.reactjs.org/docs/getting-started.html

14. PostgreSQL Documentation - [Електронний ресурс]. Режим доступу: https://www.postgresql.org/docs/

15. pgAdmin Documentation - [Електронний ресурс]. Режим доступу: https://www.pgadmin.org/docs/pgadmin4/development/getting_started.html

16. Gradle User Guide - [Електронний ресурс]. Режим доступу: https://docs.gradle.org/current/userguide/userguide.html

17. Git Documentation - [Електронний ресурс]. Режим доступу: https://git-scm.com/doc

18. Багіров О. Мікросервісна архітектура: принципи та використання [Електронний ресурс] / О. Багіров. – Режим доступу: https://prog.kiev.ua/blog/2020-02-25/mikroservisna-arhitektura-principi-ta-vikoristannya

19. Рістінг М. Що таке API і як його використовувати [Електронний ресурс] / М. Рістінг. – Режим доступу: https://techrocks.ua/2020/01/08/api-guide/

20. Загальна інформація про REST API. [Електронний ресурс] – Режим доступу: https://restfulapi.net/

21. Мудрик, М. О., & Горобець, В. І. Проектування баз даних: навчальний посібник. Київ: Центр учбової літератури, 2012.

22. JetBrains Help Documentation - [Електронний ресурс]. Режим доступу: https://www.jetbrains.com/help/

23. Короб, Н., Кікоть, В. Методи і засоби тестування програмного забезпечення. К.: Наукова думка, 2015.

24. Learning Postman (n.d.). Офіційна документація Postman- [Електронний ресурс]. Режим доступу: https://learning.postman.com/

25. JUnit 5 User Guide (n.d.). Офіційна документація JUnit- [Електронний ресурс]. Режим доступу: https://junit.org/junit5/docs/current/user-guide/

ДОДАТКИ

Дадоток А.

1. Етап фізічного проектування

Таблиця	Атрибут	Тип	Довжина
contract	id	bigint	
	contract_spec_fields	json	
	contract_ins_objects	json	
	client_surname	character varying	255
	client_name	character varying	255
	client_patronymic	character varying	255
	client_birth_date	timestamp without time zone	
	client_document_serie	character varying	4
	client_document_number	character varying	9
	client_document_issue_date	timestamp without time zone	
	client_document_issued_by	character varying	255
	client_address_region	character varying	255
	client_address_area	character varying	255
	client_address_town	character varying	255
	client_address_street	character varying	255
	client_address_house	character varying	20
	contact_phone_home	character varying	12
	id_source_user	bigint	
	id_opc	bigint	
	id_order	bigint	
	payment_sum	double precision	
	client_inn	character varying	10
	contact_phone	character varying	12
	client_email	character varying	255
	contract_date_begin	timestamp without time zone	
	contract_date_end	timestamp without time zone	
	contract_date_sign	timestamp without time zone	
	ins_tarif	double precision	
	ins_sum	double precision	
	appointment_type	character varying	15
	attendant_contracts	json	
	client_document_equivale	character varying	50

	nce_type		
	client_document_type_code	character varying	50
	creation_date	timestamp without time zone	
	step	smalint	
	is_draft	boolean	
	client_full_name_latin	character varying	255
	client_address_latin	character varying	255
	client_additional_documents	json	
	client_address_country	character varying	20
	online_product_setting_code	character varying	12
	sign	character varying	50
	original_contract_number	character varying	50
	client_address_zip	character varying	5
	resident	boolean	
	contact_phone_home_country_code	character varying	4
	contact_phone_country_code	character varying	4
	partner_info	json	
	client_address_imported	character varying	255
	promocode	character varying	255
	premies	json	
	person_citizenship	character varying	4
	is_ga_sent	boolean	
orders	id	bigint	
	order_status	character varying	60
	sales_method_type	character varying	15
	delivery_type	character varying	30
	delivery_region	character varying	255
	delivery_area	character varying	255
	delivery_town	character varying	255
	delivery_street	character varying	255
	delivery_house	character varying	20
	home_phone	character varying	12
	id_user	bigint	
	delivery_email	character varying	255
	number	character varying	50
	payment_method_type	character varying	55
	post_office	character varying	500

	id_sale_center	bigint	
	delivery_address_latin	character varying	255
	delivery_country	character varying	20
	delivery_zip	character varying	5
	creation_date	timestamp without time zone	
	delivery_status	character varying	11
	bo_saving_errors	text	
	delivery_address_imported	character varying	255
online_product_configuration	id	bigint	
	name	character varying	255
	code	character varying	255
	widget_name	character varying	255
	online_product_code	character varying	12
	online_product_name	character varying	255
	online_product_delivery_block	boolean	
	online_product_payment_block	boolean	
	need_manager_confirmation	boolean	
	sales_method_types	text[]	
	delivery_types	text[]	
	shopping_cart_enabled	character varying	9
	payment_method_types	text[]	
	type	character varying	22
	id_image_file	bigint	
	object_code	character varying	255
	ins_sum_type	character varying	15
	ins_tarif	double precision	
	prolongation	boolean	
	tarif_plans	json	
	ins_sum	double precision	
	divisions	json	
	attendant_products	json	
	box_delivery_types	text[]	
	box_divisions	json	
	compare_product_conditions	character varying	20
	offer	boolean	

	offer_text	text	
	id_banner_image	bigint	
	client_authorisation_type	character varying	20
	risk_select_type	character varying	20
	field_configurations	json	
	calc_service_path	character varying	255
	commission_inclusion_by_system_settings	boolean	
	commission_inclusion_type	character varying	255
	commission_in_percents	double precision	
	commission_payer_in_percents	double precision	
users	Id	bigint	
	email	character varying	255
	contact_phone	character varying	12
	password	character varying	100
	role	character varying	10
	enabled	boolean	
	locale	character varying	2
	surname	character varying	255
	name	character varying	255
	patronymic	character varying	255
	phone_confirmation_status	character varying	30
	phone_confirmation_code	character varying	8
	last_sms_repeat_date	timestamp without time zone	
	phone_additional	character varying	12
	inn	character varying	10
	passport_serie	character varying	4
	passport_number	character varying	9
	passport_issue_date	timestamp without time zone	
	passport_issued_by	character varying	255
	foreign_passport_serie	character varying	4
	foreign_passport_number	character varying	8
	region	character varying	255
	area	character varying	255
	town	character varying	255
	street	character varying	255
	house	character varying	20
	back_office_cagent_code	character varying	255

	birth_date	timestamp without time zone	
	link_type	character varying	10
	full_name_latin	character varying	255
	address_latin	character varying	255
	contact_phone_home	character varying	12
	country	character varying	20
	registration_date	timestamp without time zone	
	zip	character varying	5
	contact_phone_country_code	character varying	4
	contact_phone_home_country_code	character varying	4
	phone_additional_country_code	character varying	4
	address_imported	character varying	255
	resident	boolean	
	person_citizenship	character varying	4
message_resourse	id	bigint	
	key	character varying	255
	description	character varying	255
	tags	character varying[]	128
localized_message_resourse	id	bigint	
	id_message_resource	bigint	
	locale	character varying	2
	value	character varying	600

2. Код реалізації проекту

WidjetProvider

```
import FetchUtil from 'js/core/util/FetchUtil';
import ResponseErrorUtil from "js/core/util/ResponseErrorUtil";

const publicURl = __webpack_public_path__ = application.property.contextUrl + "/";

const provider = {
    OsagoCalcWidget: {
```

```
    messageResourceTag: "widget.osago",
    get: (getter) => {
        if(application.property.useSingleChunk){
            getter(require(application.property.OsagoCalcWidget).default);
        } else {
            require.ensure([], function () {
                getter(require(application.property.OsagoCalcWidget).default);
            }, "js/widget/OsagoCalcWidget");
        }
    }
},

AuthWidget: {
    messageResourceTag: "widget.auth",
    get: (getter) => {
        if(application.property.useSingleChunk){
            getter(require(application.property.AuthWidget).default);
        } else {
            require.ensure([], function () {
                getter(require(application.property.AuthWidget).default);
            }, "js/widget/AuthWidget");
        }
    }
},

BoxBuyingByRuleWidget: {
    messageResourceTag: "widget.byRule",
    get: (getter) => {
        if(application.property.useSingleChunk){
            getter(require(application.property.BoxBuyingByRuleWidget).default);
```

```
        } else {
            require.ensure([], function () {
                getter(require(application.property.BoxBuyingByRuleWidget).default);
            }, "js/widget/BoxBuyingByRuleWidget");
        }
    }
},

ContractPay: {
    messageResourceTag: "widget.contractPay",
    get: (getter) => {
        if(application.property.useSingleChunk){
            getter(require(application.property.ContractPayWidget).default);
        } else {
            require.ensure([], function () {
                getter(require(application.property.ContractPayWidget).default);
            }, "js/widget/ContractPayWidget");
        }
    }
},

Profile: {
    messageResourceTag: "widget.profile",
    get: (getter) => {
        if(application.property.useSingleChunk){
            getter(require(application.property.ProfileWidget).default);
        } else {
            require.ensure([], function () {
                getter(require(application.property.ProfileWidget).default);
            }, "js/widget/ProfileWidget");
```

```
        }
    }
},

boxBuying: {
    messageResourceTag: "widget.boxBuying",
    get: (getter) => {
        if(application.property.useSingleChunk){
            getter(require(application.property.BoxBuyingWidget).default);
        } else {
            require.ensure([], function () {
                getter(require(application.property.BoxBuyingWidget).default);
            }, "js/widget/BoxBuyingWidget");
        }
    }
},

determineWidgetName: (productCode, callbak, onError) => {
    FetchUtil.postJson({
        url: publicURl + "public/widget/info.json?opcCode=" + productCode
    }).then(
        (meta) => {
            callbak(meta.name);
        },
        (response) => {
            onError({message:ResponseErrorUtil.identifyError(response)})
        }
    )
},
```

```
    /**
     * Returns widget to getter function.
     *
     * @param name widget name
     * @param getter getter function
     * @param contextLocale context locale (not necessary, if called after
ApplicationProperties render)
     * @param target target (optional) for default error handling if reject method is not
specified
     */
    get({name, getter, contextLocale, target}) {
        this.getMessageResourceLoaderPromise({
            target,
            contextLocale: contextLocale || this.contextLocale,
            tag: this[name].messageResourceTag
        }).then(() => {
            this[name].get(getter);
        });
    }
};

Object.assign(
    provider,
    {
        getWidgetByProductCode(productCode, getter){
            provider.determineWidgetName(
                productCode,
                (name)=> {
                    provider.get({name, getter});
                }
```

```
        )
      }
    }
);

export default provider;
```

OsagoWidgetController

```
import StepOneController from 'js/customer/common/osago/stepOne/StepOneController';
import StepTwoController from
'js/customer/common/osago/stepTwo/StepTwoController';
import StepThreeController from 'js/customer/common/calculator/OrderCreationStep';

import OsagoContractToWidgetDataConverter from
'js/customer/common/converter/osago/OsagoContractToWidgetDataConverter';

import OsagoService from 'js/customer/common/service/OsagoCalcService';
import DateUtil from 'js/core/util/DateUtil';

import OsagoDataToCalcRequestDtoConverter from
'js/customer/common/converter/osago/OsagoDataToCalcRequestDtoConverter';
import OsagoDataToContractRequestDtoConverter
  from
'js/customer/common/converter/osago/OsagoDataToContractRequestDtoConverter';

import AbstractCalculatorController from
'js/customer/common/calculator/AbstractCalculatorController';
import ObjectUtil from "js/core/util/ObjectUtil";
import VehicleListHelper from
"js/customer/common/util/insObjectHelper/VehicleListHelper";
```

```
import PromiseUtil from "js/core/util/PromiseUtil";
import MtsbuRegistrationPlaceHelper from "js/core/util/MtsbuRegistrationPlaceHelper";
import FieldAvailability from "js/core/util/FieldAvailability";
import NumberUtil from "js/core/util/NumberUtil";

const INITIALIZE = "INITIALIZE";
const STEP_INITIALIZE = "STEP_INITIALIZE";
const ON_STEP_CHANGE = "ON_STEP_CHANGE";
const ON_UPDATE_CROSS_STEP_FIELDS =
"ON_UPDATE_CROSS_STEP_FIELDS";



/**
 * Common OSAGO widget controller.
 */
class OsagoWidgetController extends AbstractCalculatorController {


    constructor(props, context) {
        super(props, context);

        this.stepContents = [
            StepOneController,
            StepTwoController,
            StepThreeController
        ];

        this.stepNames = [
            "stepOne",
```

```
            "stepTwo",
            "stepThree"
        ];

        this.stepLabels = [
            "label.block.widget.osago.calculation",
            "label.block.widget.osago.contractData",
            "label.block.widget.osago.makeContract"
        ];

        this.duration = {};
        this.fieldDictData = {};
        MtsbuRegistrationPlaceHelper.processResponse(this.fieldDictData, this.context,
()=>this.forceUpdate());
    }

    componentDidMount() {
        this.context.services.authorityService.promiseUser().then(
            ()=>{this.authorized = true},
            this.onCheckAuthFail
        );
        this.stepProps = {};

        PromiseUtil.when(
            this.context.services.promiseContainerService.get("contractSettings" +
this.productCode,
                ()=>
                    this.context.services.widgetAjaxService.postJson({
                        relativeUrl:'/public/widget/contract/settings.json',
                        data: {
```

```
onlineProductConfigurationCode: this.productCode,
loadDocTypes: true,
usePromocode: true,
availabilityParameters: {
   fieldCodes: [
      "drivingSkillsLessThreeYears",
      "taxi"
   ]
},
dictionaries: [
   {
      dictCode: "gosNumberTemplate",
      fieldValueCodes: ["field1", "field2"]
   }
],
useCagentConfigSettings: true,
inputDefaultValues: [
   {
      fieldCode : 'drivingSkillsLessThreeYears'
   },
   {
      fieldCode : 'taxi'
   },
   {
      fieldCode : 'IMKDDefaultSearchVehicleBy'
   },
   {
      fieldCode : 'bonus_malus'
   }
],
```

```
                fieldValueCodes:[
                   "needGovNumberCheck",
                   "needBodyNumberLengthWarning",
                   "needBodyNumberSymbolWarning"
                ],
                inputTableSettings: [
                   {
                      tableCode: "k2",
                      fieldCodes: [
                         "territory"
                      ]
                   }
                ],
                signDate: new Date()
             }
          })
      ),
      this.getProductConfiguration()
    ).then((response, productConfiguration) => {
      this.fieldDictData = {};
      MtsbuRegistrationPlaceHelper.processResponse(this.fieldDictData, this.context,
()=>this.forceUpdate());
      this.stepProps.stepOne = {
         productConfiguration,
         onUpdateCrossStepFields: this.onUpdateCrossStepFields
      };
      this.contractConverter = new
OsagoDataToContractRequestDtoConverter(this.fieldDictData, this.props.contract,
this.context);
      this.osagoService = new OsagoService(
```

```
            this.context,
            new OsagoDataToCalcRequestDtoConverter(this.fieldDictData),
            this.contractConverter,
            this.productCode,
            this.interactionUuid
        );

        this.duration = response.duration;
        this.stepProps.stepTwo = Object.assign(this.stepProps.stepTwo || {}, {duration:
response.duration, fieldValues: response.fieldValues});
        const fieldAvailabilities = {};
        response.fieldsAvailability.forEach(av=>fieldAvailabilities[av.fieldCode] =
av.availability);
        const defaultValues = {};
        response.inputSettings.forEach(is=>{
            if(is.defaultValue){
                defaultValues[is.fieldCode] = is.defaultValue.value;
            }
        });
        const drivingSkillsLessThreeYears = (defaultValues.drivingSkillsLessThreeYears
=== "YES").toString();
        this.availabilities.drivingSkillsLessThreeYears =
fieldAvailabilities.drivingSkillsLessThreeYears;
        this.availabilities.taxi = fieldAvailabilities.taxi;
        this.availabilities.promocode = response.promocodeAvailability;
        let containsForeignCountry = false;
        let containsUA = false;
        let any = false;
        const findEq = (row, value)=>row.fields.find(field=>field.value === value);
        const findNotEq = (row, value)=>row.fields.find(field=>field.value !== value);
```

```
        response.inputSettings.find(s=>{
            if(s.fieldCode === "k2"){
                s.rows.find(row=>{
                    any = findEq(row, "any");
                    containsForeignCountry = any || containsForeignCountry ||
findEq(row,"zone_7");
                    containsUA = any || containsUA || findNotEq(row,"zone_7");
                    return any || (containsForeignCountry && containsUA);
                });
                return true;
            }
        });

        this.cagentConfig = response.cagentConfig;

        this.availabilities.foreignCountry = containsForeignCountry ? containsUA ?
FieldAvailability.OPTIONAL : FieldAvailability.CONST :
FieldAvailability.DISABLED;
        this.availabilities.cagentConfig = response.cagentConfig;
        const foreignCountry = containsForeignCountry && !containsUA;
        const taxi = defaultValues.taxi === "YES";
        const bonus_malus = defaultValues.bonus_malus;
        const IMKDDefaultSearchVehicleBy =
defaultValues.IMKDDefaultSearchVehicleBy;
        this.dispatch(INITIALIZE, {drivingSkillsLessThreeYears, taxi, bonus_malus,
foreignCountry, IMKDDefaultSearchVehicleBy});
      });
      VehicleListHelper.initObjectData(this.context);
    }
```

```
    getService(){
        return this.osagoService;
    }

    getActionMap(){
        return {
            ...super.getActionMap(),
            ...{
                [INITIALIZE]: (state, action) => {
                    let dateBegin = DateUtil.formatDateTime(DateUtil.getOffsetDate(1),
DateUtil.FORMAT_DDMMYYYY_SLASH_DIVIDER);
                    let dateEnd = DateUtil.getContractDateEnd(dateBegin, this.duration);
                    let inputPromocode = ObjectUtil.getObjectProperty(this.props.contract,
"promocode");
                    if(!inputPromocode) {
                        inputPromocode = {
                            value:'',
                            accepted: false,
                            pressed: false,
                        };
                    }
                    const stepData = !this.isContractNew || this.isProlongation
                        ? new
OsagoContractToWidgetDataConverter().convert(this.props.contract)
                        : {
                            stepOne: {
                                data: {
                                    vehicleType: "car",
                                    drivingSkillsLessThreeYears: action.drivingSkillsLessThreeYears,
                                    taxi: action.taxi,
```

```
                        bonus_malus: action.bonus_malus,
                        foreignCountry: action.foreignCountry,
                        IMKDDefaultSearchVehicleBy:
action.IMKDDefaultSearchVehicleBy,
                        promocode: inputPromocode,
                        osagoCostWithPromocode: null,
                    }
                }
            };
            stepData.stepOne.data.attendant =
this.processAttendants(stepData.stepOne.data.attendant);
            stepData.stepOne.data.promocodeAvailability =
action.promocodeAvailability;
            if (this.isProlongation) {
                Object.assign(stepData.stepTwo.data, {
                    contractDateBegin: dateBegin,
                    contractDateEnd: dateEnd
                });

            }
            return {
                initialized: true,
                activeStep: 0,
                shoppingCartEnabled: action.shoppingCartEnabled,
                [`initialization.stepThree`]: true,
                stepData
            }
        },
        [ON_UPDATE_CROSS_STEP_FIELDS]: (state, action) => {
            const crossFieldsValues = {
```

```
                mark: action.mark,
                model: action.model,
                govNumber: action.govNumber,
                bodyNumber: action.bodyNumber,
            };
            const firstStepData = Object.assign({},
this.getStateProperty("stepData.stepOne.data"), crossFieldsValues);
            const secondStepVehicleData = Object.assign({},
this.getStateProperty("stepData.stepTwo.data.vehicleData"), crossFieldsValues);
            return {
                ["stepData.stepOne.data"]: firstStepData,
                ["stepData.stepTwo.data.vehicleData"]: secondStepVehicleData
            }
        },
    }
  }

  processAttendants(attendants){
    attendants = attendants || {};
    this.onlineProductConfiguration.attendantProducts.forEach(ap=>{
      const attendant = attendants[ap.id] = attendants[ap.id] || {id: ap.id};
      attendant.productCode = ap.productCode;
      if(ap.cost.value) {
        attendant.insSum = NumberUtil.parseCurrency(ap.cost.value);
        if(ap.costWithPromocode) {
          attendant.insSum =
NumberUtil.parseCurrency(ap.costWithPromocode.value);
        }
      }
```

```
            attendant.customInsSums = ap.customInsSums;
            attendant.currentCustomInsSumCode = ap.customInsSums &&
ap.customInsSums[0] ? ap.customInsSums[0].code : null;
            attendant.value = FieldAvailability.REQUIRED_WITH_CANCEL_OPTION
=== ap.availability;
        });
        return attendants;
    }

    updateNameAndSurname(linkedCagent){
        this.getStateProperty("stepData.stepTwo.data").surname =
linkedCagent.person.surname;
        this.getStateProperty("stepData.stepTwo.data").name = linkedCagent.person.name;
        this.getStateProperty("stepData.stepTwo.data").patronymic =
linkedCagent.person.patronymic;
    }

    getRequiredStepProps(){
        const stepName = this.stepNames[this.state.activeStep];
        return {
            ...super.getRequiredStepProps(),
            statePath: `${this.getStatePath()}.stepData.${stepName}`,
            availabilities: this.availabilities,
            stepData: this.state.stepData,
            stepProps: this.stepProps[stepName],
            productCode: this.productCode,
            vehicleUseInTown:
this.getStateProperty("stepData.stepOne.data.vehicleUseInTown"),
            fieldDictData: this.fieldDictData,
```

```
            costCalculationType:
this.getStateProperty("stepData.stepOne.data.costCalculationType")
        }
    }

    getClientData(){
        return ObjectUtil.getObjectProperty(this.state.stepData, "stepTwo.data") || {};
    }

    getFullData(){
        const stepData = this.state.stepData;
        return {
            ...ObjectUtil.getObjectProperty(stepData, "stepOne.data"),
            ...ObjectUtil.getObjectProperty(stepData, "stepTwo.data"),
            ...ObjectUtil.getObjectProperty(stepData, "stepTwo.vehicleData"),
            ...ObjectUtil.getObjectProperty(stepData, "stepTwo.address")
        }
    }

    /**
     * Updates fields which used at the first and the second steps.
     *
     * @param mark mark name
     * @param model model name
     * @param govNumber government number
     * @param bodyNumber body number
     * @returns {Promise} promise
     */
    onUpdateCrossStepFields = ({mark, model, govNumber, bodyNumber}) => {
```

```
        return this.dispatch(ON_UPDATE_CROSS_STEP_FIELDS, {mark, model,
govNumber, bodyNumber});
    };
}

export default OsagoWidgetController;
```