СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет *Інформаційних технологій та електроні*

Кафедра *Інформаційних технологій та програмування*

ПОЯСНЮВАЛЬНА ЗАПИСКА

до магістерської дипломної роботи

Магістр

(освітньо-кваліфікаційний рівень)

на тему *Платформа для управління взаємодією з клієнтами (CRM) для страхової компанії*

Виконав: студент 2 курсу, групи ІСТ-23дм
126 «Інформаційні системи та технології
(шифр і назва спеціальності)
Гніліцький О.С.
(прізвище та ініціали)
Керівник Меняйленко О.С.
(прізвище та ініціали)
Рецензент Ратов Д.В.
(прізвище та ініціали)

Київ - 2024

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет *Інформаційних технологій та електроніки*
Кафедра *Інформаційних технологій та програмування*
Освітньо-кваліфікаційний рівень *Магістр*
Спеціальність *126 Інформаційні системи та технології*
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТП
_____ д.т.н., проф., Захожай О. І.
(підпис)
"___" _____ 2024 року

ЗАВДАННЯ

На магістерську дипломну роботу студента
Гніліцький Олексій Сергійович

1. Тема роботи *Платформа для управління взаємодією з клієнтами (CRM) для страхової компанії*
Керівник роботи Проф., д.т.н. Меняйленко Олександр Сергійович ,
(прізвище, ім'я, по-батькові, науковий ступінь, вчене звання)

Затверджений наказом університету від "*06*"*грудня* 2024року №*361/15.15-С*.
Строк подання студентом роботи *15 грудня 2024 р.* 3. Вихідні дані роботи *Об'єктом данної роботи є процес створення платформи для управління взаємодією з клієнтами (CRM) страхової компанії*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   - 4.1 Вступ
   - 4.2 Аналітичний огляд питання.
   - 4.3 Основна частина, де досліджено та проаналізовано методи для реалізації проекту, а також сформовано вирішення для розв'язку поставленої задачі.
   - 4.4 Практична частина, в якій зроблено огляд технологій та інструментарію, які використовуються для реалізації проекту.
   - 4.5 Висновки
   - 4.6 Перелік використаних джерел.
5. Перелік графічного матеріалу (немає)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *20 жовтня 2024 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1.	Отримання завдання на виконання роботи	20.10.2024	
2.	Укладення та погодження з керівником плану і етапів виконання роботи	24.10.2024	
3.	Погодження з керівником оптимального шляху виконання завдання.	28.10.2024	
4.	Аналіз технічних засобів існуючих систем	01.11.2024	
5.	Реалізація практичної частини	07.11.2024	
6.	Написання, оформлення та узгодження пояснювальної записки з керівником	24.11.2024	
7.	Здача пояснювальної записки на кафедру	05.12.2024	
8.	Підготовка доповіді та презентації	17.12.2024	

Студент _____ Гніліцький О.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Меняйленко О.С.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська дипломна робота: 90 стор., 2 табл., 21 рис., 17 джерел.

Об’єкт дослідження – розробка платформи для управління взаємодією з клієнтами (CRM) для страхової компанії.

Мета роботи – дослідження та розробка платформи CRM, що дозволяє ефективно керувати взаємодією з клієнтами страхової компанії, автоматизувати бізнес-процеси, покращити якість обслуговування та забезпечити надійний захист даних клієнтів.

Завдання, що поставлені для досягнення мети:

1. Проаналізувати вимоги страхової компанії до функціональності CRM-системи.
2. Оцінити існуючі CRM-платформи та їхні можливості для інтеграції з іншими системами, такими як ERP, платіжні платформи та системи управління ризиками.
3. Розробити власну платформу CRM для страхової компанії з урахуванням специфічних вимог бізнес-процесів.
4. Забезпечити належний рівень безпеки даних та конфіденційності, відповідаючи вимогам законодавства.
5. Створити інтерфейс користувача, що забезпечить простоту і зручність роботи співробітників компанії та клієнтів через особистий кабінет.
6. Впровадити аналітичні функції для прогнозування потреб клієнтів та визначення потенційних можливостей для продажів.

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

CRM, ІНТЕГРАЦІЯ, ERP, ВЗАЄМОДІЯ З КЛІЄНТАМИ, СТРАХУВАННЯ, БЕЗПЕКА ДАНИХ, АНАЛІТИКА, ПЕРСОНАЛІЗАЦІЯ, СТРАХОВІ ПРОДУКТИ, УПРАВЛІННЯ РИЗИКАМИ, UI/UX, БАЗА ДАНИХ, ЗАЯВКИ, КЛІЄНТСЬКИЙ СЕРВІС, ЗАХИСТ ДАНИХ, ПРОГРАМУВАННЯ, КАБІНЕТ КЛІЄНТА, ІНТЕРФЕЙС, ТЕСТУВАННЯ, JAVA, GRADLE, SMSS, GIT

ABSTRACT

Master's Thesis: 90 pages, 2 tables, 21 figures, 17 references.
Object of the research – development of a CRM platform for managing customer interactions for an insurance company.
The aim of the work – to research and develop a CRM platform that allows for effective management of customer interactions in an insurance company, automates business processes, improves service quality, and ensures reliable data protection for clients.

Tasks set to achieve the goal:

1. Analyze the requirements of the insurance company for the functionality of the CRM system.
2. Evaluate existing CRM platforms and their capabilities for integration with other systems such as ERP, payment platforms, and risk management systems.
3. Develop a custom CRM platform for the insurance company, taking into account the specific business process requirements.
4. Ensure an adequate level of data security and confidentiality, in compliance with legal requirements.
5. Create a user interface that ensures simplicity and convenience for employees and customers through the personal account.
6. Implement analytical functions for predicting customer needs and identifying potential sales opportunities.

The process of developing and testing the system is described. The user interface is also implemented and described, along with screenshots of the software forms. The results of the completed work are demonstrated.
CRM, INTEGRATION, ERP, CUSTOMER INTERACTION, INSURANCE, DATA SECURITY, ANALYTICS, PERSONALIZATION, INSURANCE PRODUCTS, RISK MANAGEMENT, UI/UX, DATABASE, APPLICATIONS, CUSTOMER SERVICE, DATA PROTECTION, PROGRAMMING, CLIENT ACCOUNT, INTERFACE, TESTING, JAVA, GRADLE, SMSS, GIT.

ЗМІСТ

ВСТУП 8

Розділ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАТІ ТА ВИМОГ ДО CRM-СИСТЕМ ДЛЯ СТРАХОВОЇ КОМПАНІЇ 10

1.1. Характеристика страхової діяльності та особливості взаємодії з клієнтами 10

1.2. Особливості конкурентного середовища на ринку страхових послуг .. 12

1.3. Огляд існуючих CRM-рішень на ринку 13

1.4. Аналіз потреб страхової компанії щодо функціональності CRM 15

1.5. Аналіз переваг та недостатків платформи CRM для страхової компанії 16

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ 19

2.1. Підходи до створення інформаційних систем. 19

2.2. Agile - Гнучкий підхід розробки програмного забезпечення 22

2.3. Scrum – фреймворк Agile 24

2.4. Розробка структури системи 24

2.5. Визначення компонентів системи 26

2.6. Мікросервісна архітектура 31

2.7. Model-View-Controller 33

2.8. Розробка бази даних 35

2.9. Етап датологічного та фізичного проектування бази данних 37

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСОБУ 40

3.1. Середовище розробки 40

3.2. Опис призначеного для користувача інтерфейсу 42

3.3. Тестування 49

ВИСНОВКИ..... 53

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 55

ДОДАТКИ..... 57

ВСТУП

У сучасному світі страхова діяльність є однією з ключових складових економіки, що забезпечує захист майнових, фінансових та особистих інтересів клієнтів. Зростання конкуренції на ринку страхових послуг вимагає від компаній постійного вдосконалення бізнес-процесів, персоналізації підходів до клієнтів і підвищення ефективності взаємодії з ними. Саме тому важливим інструментом для досягнення цих цілей є впровадження систем управління взаємодією з клієнтами (CRM).

CRM-система дозволяє оптимізувати роботу з клієнтами, зберігати інформацію про них, автоматизувати комунікацію та аналітику, що сприяє зростанню рівня задоволеності споживачів. Для страхових компаній особливе значення має можливість швидкого реагування на запити клієнтів, ефективного управління страховими договорами та врегулювання страхових випадків.

Розробка спеціалізованої CRM-платформи, яка враховує специфіку страхової діяльності, дозволить значно покращити якість обслуговування клієнтів і підвищити конкурентоспроможність компанії. Впровадження такої системи забезпечить зниження витрат, прискорення бізнес-процесів і формування довготривалих відносин із клієнтами.

Актуальність теми дослідження зумовлена потребою в інноваційних рішеннях для підвищення ефективності управління взаємодією зі споживачами в умовах цифрової трансформації. Мета дипломної роботи полягає у розробці концепції CRM-платформи, яка буде відповідати потребам страхової компанії, забезпечуючи інтеграцію з іншими інформаційними системами та задовольняючи вимоги сучасного ринку.

Об'єктом дослідження є взаємодія між страховими компаніями та їхніми клієнтами. Предметом дослідження є автоматизація бізнес-процесів і управління клієнтськими відносинами за допомогою CRM-систем.

У дипломній роботі буде розглянуто особливості страхової діяльності, проаналізовано сучасні CRM-рішення, розроблено вимоги до

функціональності платформи та запропоновано архітектуру спеціалізованої CRM-системи для страхової компанії.

Розділ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАТІ ТА ВИМОГ ДО CRM-СИСТЕМ ДЛЯ СТРАХОВОЇ КОМПАНІЇ

1.1. Характеристика страхової діяльності та особливості взаємодії з клієнтами

Страхова діяльність є важливим елементом фінансової системи, що спрямований на управління ризиками. Її головною метою є забезпечення фінансового захисту фізичних і юридичних осіб від можливих ризиків шляхом укладання договорів страхування. Основні принципи страхової діяльності включають:

- Еквівалентність: Вартість страхового покриття повинна відповідати рівню ризику.
- Диверсифікація ризиків: Розподіл ризиків між великою кількістю страхувальників.
- Фінансова стійкість: Здатність страхової компанії виконувати зобов'язання перед клієнтами.
- Довгостроковість: Поліси часто укладаються на тривалий період, що вимагає ефективного управління фінансовими ресурсами.

Види страхової діяльності:

- Особисте страхування: Захист життя, здоров’я та працездатності (медичне страхування, страхування життя).
- Майнове страхування: Захист майна від пошкоджень або втрат (автострахування, страхування нерухомості).
- Страхування відповідальності: Покриття збитків, завданих третім особам (цивільна відповідальність, професійна відповідальність).
- Перестрахування: Розподіл великих ризиків між кількома страховими компаніями для зменшення фінансового навантаження.

Особливості взаємодії страхової компанії з клієнтами визначаються високим рівнем персоналізації послуг та необхідністю формування довіри. Кожен етап взаємодії, від першого контакту до врегулювання страхових випадків, спрямований на забезпечення клієнтів якісним обслуговуванням та відчуттям надійності.

Спершу страхова компанія встановлює контакт із клієнтом, аналізує його потреби та надає консультації щодо доступних страхових продуктів. Важливо, щоб цей процес був прозорим і зрозумілим для клієнта, адже від цього залежить довіра до компанії. Особливу увагу приділяють укладенню договорів, де обговорюються всі умови страхування, можливі ризики та обмеження.

Під час дії договору компанія підтримує зв'язок із клієнтом, забезпечуючи регулярні консультації та доступ до необхідної інформації. Все частіше для цього використовуються цифрові інструменти, такі як мобільні додатки чи онлайн-кабінети, що дозволяють клієнтам швидко отримувати інформацію та керувати своїми полісами.

У разі настання страхового випадку компанія має забезпечити оперативне та чітке вирішення питання. Процес обробки заявок і виплати компенсацій повинен бути максимально простим і швидким, адже саме в цей момент перевіряється надійність страховика.

Загалом взаємодія з клієнтами базується на прозорості, відповідальності та клієнтоорієнтованості. Надійне виконання зобов'язань і сучасний підхід до обслуговування формують довгострокові партнерські відносини та позитивний імідж страхової компанії.

CRM-системи сприяють покращенню взаємодії зі споживачами, зокрема шляхом:

- Автоматизації обробки запитів і управління страховими договорами.
- Збирання та аналізу даних про клієнтів для створення персоналізованих пропозицій.
- Забезпечення єдиного інформаційного простору для роботи з клієнтами.

Страхова діяльність вимагає ефективного управління взаємодією з клієнтами, оскільки якість обслуговування напряму впливає на довіру та лояльність клієнтів. Особливості страхових послуг, такі як тривалість відносин, потреба в персоналізації та багатоканальна комунікація, обумовлюють актуальність впровадження CRM-систем для автоматизації процесів і підвищення конкурентоспроможності страхової компанії. [1]

1.2. Особливості конкурентного середовища на ринку страхових послуг

Середовище страхових послуг відрізняється динамічністю та має низку характерних особливостей, які залежать від специфіки ринку, географічного розташування та регуляторних умов. Основні риси цього середовища можна окреслити наступним чином:

По-перше, конкуренція між страховими компаніями є ключовим фактором, що впливає на розвиток ринку. Велика кількість учасників змушує компанії постійно вдосконалювати свої послуги, пропонувати більш вигідні умови, широкий вибір страхових продуктів та гнучку тарифну політику, щоб залучати та утримувати клієнтів.

По-друге, діяльність страховиків жорстко регулюється законодавством і контролюється відповідними державними органами. Регуляторні вимоги включають забезпечення фінансової стабільності компаній, дотримання стандартів захисту прав клієнтів, а також прозорість у процесах врегулювання збитків. При цьому кожна країна має свої унікальні норми, які страхові компанії зобов’язані виконувати.

Технологічний прогрес також відіграє важливу роль у трансформації ринку страхових послуг. Сучасні компанії активно впроваджують цифрові технології: онлайн-сервіси, мобільні додатки, автоматизовані системи аналізу даних. Використання штучного інтелекту, роботизованих процесів і великих даних сприяє покращенню обслуговування клієнтів та оптимізації внутрішніх бізнес-процесів.

Важливим аспектом є управління фінансовими ризиками. Страхові компанії повинні не лише гарантувати виплату відшкодувань, але й ефективно

розпоряджатися резервами. Інвестиційна діяльність компаній має забезпечувати надійність і прибутковість, враховуючи різноманітні фінансові інструменти, такі як цінні папери чи нерухомість.

Очікування клієнтів постійно зростають. Сьогодні споживачі прагнуть отримати прозорі умови, доступність послуг, простоту укладення договорів і швидке врегулювання страхових випадків. Компанії, які орієнтуються на клієнта і задовольняють його потреби, досягають успіху на ринку.

Не менш важливими є технічні та актуарні аспекти страхування. Аналіз ризиків, формування тарифів та прогнозування страхових подій вимагають професійного підходу. Актуарні розрахунки та експертна оцінка допомагають забезпечити фінансову стабільність компаній і захист інтересів клієнтів.

Отже, сфера страхових послуг поєднує високу конкуренцію, технологічний розвиток, жорстке регулювання та орієнтацію на потреби клієнтів. Всі ці фактори формують сучасне середовище страхування, де компанії мають адаптуватися до змін, пропонуючи якісні та інноваційні рішення.[2]

1.3. Огляд існуючих CRM-рішень на ринку

Ринок CRM-рішень постійно розвивається, пропонуючи компаніям інструменти для оптимізації взаємодії з клієнтами, підвищення ефективності бізнес-процесів і вдосконалення управління. Ці рішення адаптуються до потреб різних компаній — від малих підприємств до великих корпорацій, а також враховують специфіку окремих галузей.

Одним із найпопулярніших напрямів є хмарні CRM-системи. Вони забезпечують доступ до даних із будь-якого місця та надають компаніям гнучкість у використанні. Такі системи, як Salesforce, HubSpot CRM і Zoho CRM, вирізняються зручністю інтеграції, багатофункціональністю та можливістю масштабування. Вони особливо корисні для компаній, що прагнуть швидко адаптувати свої процеси до змін ринку, не інвестуючи в складну інфраструктуру.

Для великих корпорацій актуальні рішення, які пропонують розширені функції інтеграції з іншими системами, такими як ERP чи аналітичні платформи. Наприклад, Microsoft Dynamics 365 та SAP Customer Experience здатні підтримувати управління складними процесами, автоматизувати продажі, маркетинг і забезпечувати високий рівень клієнтського сервісу. Такі платформи часто використовуються для побудови довгострокової стратегії взаємодії з клієнтами.

Малі та середні підприємства надають перевагу CRM-системам, які поєднують доступність, простоту використання та необхідний набір базових функцій. Bitrix24, Pipedrive та Insightly — це платформи, які дозволяють ефективно управляти продажами, маркетингом і проєктами, часто пропонуючи функціонал «все в одному». Вони особливо корисні для компаній, що шукають економічно вигідні рішення для оптимізації роботи без значних інвестицій.

Крім того, деякі CRM-рішення спеціалізуються на окремих галузях. Наприклад, Veeva CRM, орієнтована на фармацевтичний сектор, або Propertybase, створена для управління бізнесом у сфері нерухомості. Такі системи враховують специфічні потреби окремих ринків, що дозволяє компаніям ефективніше управляти своїми процесами.

У сучасному CRM-просторі також важливу роль відіграють технологічні інновації. Впровадження штучного інтелекту, автоматизації процесів, аналітики великих даних та інтеграції з мобільними додатками сприяє підвищенню ефективності роботи. Багато CRM-систем пропонують можливості персоналізації послуг для клієнтів, що дозволяє підвищити їхню лояльність.

Таким чином, ринок CRM-рішень демонструє значну різноманітність пропозицій, які враховують специфіку бізнесу, розмір компанії та особливості галузі. Вибір системи залежить від цілей організації, її ресурсів і потреб у функціональності. Ефективна інтеграція CRM може стати ключовим чинником підвищення конкурентоспроможності та забезпечення довгострокового успіху бізнесу.

Параметр	Універсальні CRM	Галузеві CRM	Індивідуальні розробки
Вартість	Середня	Висока	Залежить від масштабу
Функціональність	Широка, але загальна	Адаптована до галузі	Повна відповідність
Термін впровадження	Швидкий	Середній	Тривалий

Таблиця 1.1. - Порівняння існуючих рішень

Основна проблема готових рішень — необхідність адаптації до специфіки локального ринку та процесів компанії. У свою чергу, індивідуальні розробки часто вимагають значних інвестицій, але забезпечують найкращу інтеграцію з внутрішніми бізнес-процесами. [3]

1.4. Аналіз потреб страхової компанії щодо функціональності CRM

Для визначення вимог до CRM-системи страхової компанії слід врахувати такі ключові аспекти її діяльності:

1. Ключові функції, які має виконувати CRM:
   - Управління даними клієнтів (контакти, історія взаємодій, договори).
   - Автоматизація процесу обробки заявок.
   - Підтримка сегментації клієнтів та їхнього аналітичного профілю.
   - Управління взаємодією з клієнтами через різні канали зв’язку.
2. Інтеграція з іншими системами:
   - ERP-системами для фінансового та ресурсного управління.
   - Платіжними платформами для обробки транзакцій.
   - Системами для оцінки ризиків та врегулювання страхових випадків.
3. Аналітика та прогнозування:
   - Оцінка потреб клієнтів на основі даних.
   - Визначення потенційних ризиків та можливостей для продажів.

4. Користувацький досвід:

- Простота використання інтерфейсу.
- Можливість самостійного доступу клієнтів до інформації через особистий кабінет.

5. Безпека даних:

- Дотримання вимог локального законодавства щодо захисту персональних даних.
- Сучасні інструменти шифрування та захисту інформації.

Проблеми, які вирішує CRM:

- Втрата даних через неефективне управління.
- Низька швидкість обробки заявок.
- Недостатнє розуміння потреб клієнтів.
- Складність взаємодії між відділами компанії.

Аналіз предметної області та доступних рішень показав, що для страхової компанії CRM-система має бути адаптованою до специфіки її діяльності. Вона повинна забезпечувати комплексний підхід до управління взаємодією з клієнтами, автоматизувати бізнес-процеси та підтримувати прийняття управлінських рішень. Наступні етапи роботи зосереджені на розробці архітектури такої системи, враховуючи визначені вимоги та особливості.[4]

1.5. Аналіз переваг та недостатків платформи CRM для страхової компанії

Впровадження CRM-платформи в страховій компанії відкриває значні можливості для оптимізації бізнес-процесів, але водночас супроводжується певними викликами. Розглянемо детально переваги та недоліки такого рішення.

Переваги використання CRM

CRM-платформи сприяють покращенню взаємодії з клієнтами завдяки можливості зберігати детальну інформацію про кожного з них. Система

дозволяє створити профіль клієнта, який включає його контактні дані, історію полісів, заявок та навіть скарг. Це забезпечує більш персоналізоване обслуговування, що підвищує довіру до компанії.

Автоматизація процесів є ще однією важливою перевагою. CRM може автоматично оновлювати базу даних, нагадувати клієнтам про продовження полісів, обробляти заявки та створювати звіти. Це не лише економить час співробітників, але й мінімізує ризик помилок, які можуть виникати при ручній обробці даних.

Аналітичні можливості CRM дозволяють страховим компаніям глибше розуміти своїх клієнтів. Система аналізує їхню поведінку, оцінює ефективність маркетингових кампаній та допомагає визначати стратегічні напрями розвитку бізнесу. Завдяки таким даним компанія може оперативно реагувати на зміни на ринку.

Сприяння збільшенню продажів є ще одним ключовим аспектом. CRM допомагає ідентифікувати можливості для крос-продажів та створювати таргетовані пропозиції. Наприклад, клієнту, який уже придбав автомобільне страхування, можна запропонувати страхування майна.

Окрім цього, сучасні CRM-системи інтегруються з іншими програмними рішеннями, такими як ERP-системи, IP-телефонія чи маркетингові платформи. Це створює єдину екосистему, яка підвищує зручність роботи для співробітників.

Недоліки використання CRM

Незважаючи на численні переваги, впровадження CRM-платформи супроводжується значними витратами. Придбання програмного забезпечення, його адаптація до специфіки компанії, навчання персоналу та технічна підтримка можуть стати фінансово обтяжливими.

Процес інтеграції CRM із уже існуючими системами страхової компанії нерідко виявляється складним і тривалим. Особливо це стосується випадків, коли компанія використовує специфічні або застарілі платформи, які не підтримують сучасні стандарти обміну даними.

Ще однією проблемою є залежність від технологій. Якщо CRM-платформа працює в хмарі, проблеми з доступом до сервера можуть уповільнити роботу компанії. Аналогічно, збій у локальній версії програми може призвести до втрати доступу до важливих даних.

Крім того, впровадження нової системи часто стикається з опором персоналу. Співробітники можуть неохоче приймати нові технології, що вимагає додаткових зусиль для їх навчання та адаптації.

Зберігання великої кількості конфіденційної інформації про клієнтів також потребує особливої уваги до безпеки. Недостатнє налаштування системи захисту може призвести до витоку даних, що матиме катастрофічні наслідки як для компанії, так і для її клієнтів.

CRM-платформа є потужним інструментом для оптимізації роботи страхової компанії та підвищення якості обслуговування клієнтів. Проте для успішного впровадження потрібно ретельно оцінити фінансові та технічні ресурси, а також врахувати ризики, пов'язані з безпекою даних та опором з боку співробітників. Ефективне впровадження CRM може суттєво змінити бізнес, створивши умови для сталого розвитку та посилення конкурентних позицій компанії.[5]

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Підходи до створення інформаційних систем.

Підходи до створення інформаційних систем — це набір методів, принципів і стратегій, що використовуються для планування, проектування, розробки, впровадження та підтримки інформаційних систем. Ці підходи забезпечують структурований процес, який допомагає досягати поставлених цілей, враховуючи потреби бізнесу, технічні обмеження та ресурси.

Існує безліч підходів до проектування інформаційних систем, і вибір конкретного залежить від специфіки проєкту та уподобань команди розробників:

Нижче наведено найпопулярніші з них:

1. Каскадна модель розробки програмного забезпечення (*Waterfalll)* — це традиційний підхід до управління проектами, при якому етапи розробки виконуються послідовно. Цей підхід отримав свою назву через те, що кожен етап плавно "перетікає" в наступний, як вода в каскаді. Ключова особливість цієї моделі полягає в тому, що повернення до попередніх етапів у процесі виконання проекту є складним і небажаним.

Каскадна модель має як переваги, так і недоліки. Її основними перевагами є простота управління процесом розробки, можливість точніше коригувати терміни та бюджети, а також визначення витрат на самому початку, що дозволяє працювати безперервно. Крім того, тестування можна проводити, використовуючи документацію, що зменшує потребу у залученні досвідчених тестувальників.

Однак у цієї моделі є й недоліки. Помилки виявляються лише на пізніх етапах, що ускладнює та здорожчує їх виправлення. Клієнти отримують можливість оцінити результати тільки після завершення всіх етапів, і це може призвести до змін та перевищення бюджету. Також каскадна модель передбачає створення великого обсягу технічної документації, що уповільнює процес та вимагає більше часу на узгодження.[6]

2. Гнучка модель розробки програмного забезпечення **(Agile)** - це підхід, який підкреслює гнучкість, швидку ітерацію, взаємодію розробника і замовника та постійне вдосконалення продукту за допомогою регулярного зворотного зв'язку. Він дозволяє швидко реагувати на мінливі вимоги і підходить для проектів, де невизначеність і швидкі зміни є нормою.

Переваги гнучкої моделі розробки програмного забезпечення включають гнучкість до змін, здатність швидко реагувати на нові вимоги та швидко досягати результатів за допомогою ітеративного процесу. Безперервне тестування та зворотній зв'язок з клієнтами покращують якість продукту і дозволяють виправляти помилки на ранній стадії. Крім того, гнучка модель підтримує самоорганізацію команд і підвищує їхню ефективність роботи.

Недоліком цієї моделі є невизначеність графіків і бюджетів через постійні зміни, що вимагає залучення досвідчених експертів для ефективної роботи. Часом зміни можуть створювати проблеми з документацією та маргіналізувати важливі аспекти проекту, роблячи процес більш хаотичним.

3. Швидка розробка додатків (RAD) - це методологія розробки програмного забезпечення, яка фокусується на швидкому створенні додатків за допомогою прототипування, безперервного зворотного зв'язку та мінімальної документації. Його мета - скоротити час розробки та швидко надавати функціонуючі продукти.

Перевагами є швидка розробка завдяки створенню прототипів та зворотному зв'язку, гнучкість у внесенні змін протягом усього процесу та краще узгодження з вимогами замовника завдяки постійній взаємодії.

Недоліки полягають у тому, що методологія не підходить для великих проектів, вимагає досвідчених фахівців для ефективної роботи і може бути складною в управлінні через мінімальну кількість документації.

4. Спіральна модель (Spiral) -це методологія розробки програмного забезпечення, яка поєднуєв собіелементимоделіводоспадута прототипування. Вонадозволяєзабезпечитибезперервнийзворотний зв'язок і коригування

продукту на кожному етапі шляхом повторення циклу(спіралі),вякомукожнастадіярозробки проходить черезетапипланування, аналізу ризиків, розробки таоцінки.

Спіральна модель має такі переваги, як ефективне управління ризиками завдяки регулярному аналізу, можливість вносити зміни на будь-якому етапі, чітке планування для забезпечення контролю над проектом і підвищення якості завдяки безперервному тестуванню та зворотному зв'язку.

До недоліків можна віднести високі витрати, пов'язані з частим тестуванням і аналізом ризиків, складність управління, особливо у великих проектах, і потребу в досвідчених фахівцях для її ефективної реалізації.

Окрім вибору підходу, при проектуванні інформаційних систем слід також враховувати наступні фактори:

- Бізнес-цілі та вимоги: обрана технологія повинна відповідати стратегічним цілям організації та задовольняти потреби користувачів і зацікавлених сторін.
- Технічна здійсненність і сумісність: переконайтеся, що технологія є технічно здійсненною і сумісною з існуючими системами, інфраструктурою та ресурсами.
- Вартість і цінність: оцініть загальну вартість володіння, включаючи розробку, впровадження та обслуговування, а також цінність, яку технологія приносить бізнесу.
- Досвід і задоволеність користувачів: зручність використання є важливим аспектом ефективного впровадження, тому технологія повинна забезпечувати дружній інтерфейс і підвищувати загальну задоволеність користувачів.
- Планування та управління проектом: зокрема розподіл ресурсів та контроль прогресу, для забезпечення вчасного виконання проекту.

- Тестування та забезпечення якості розробки для виявлення помилок та підвищення надійності системи. [7]

2.2. Agile - Гнучкий підхід розробки програмного забезпечення

Agile - це підхід до розробки програмного забезпечення, який наголошує на адаптивності та співпраці. Agile виник як відповідь на обмеження традиційних методологій, які часто були занадто жорсткими, щоб впоратися зі швидкими змінами на ринку. Такий метод дає дозвіл командам розробників реалізувти цінності для клієнтів на фундаментальних стадіях проекту та постійно вдосконалювати продукт за допомогою ітеративного процесу розробки.

Рисунок 2.1. – Agile

Характеристики гнучкої метадології:

- Короткі цикли розробки: проекти поділяються на короткі фази, які називаються ітераціями або спринтом, і зазвичай тривають від одного до чотирьох тижнів.
- Безперервна комунікація: регулярний обмін інформацією між командою та зацікавленими сторонами має важливе значення для узгодження вимог та очікувань.

- Безперервна доставка: функціональне програмне забезпечення доставляється в кінці кожної ітерації, що забезпечує швидкий зворотній зв'язок.
- Адаптивність: зміни до проекту приймаються навіть на пізніх стадіях розробки.
- Орієнтованість на клієнта: потреби та задоволення потреб клієнта є головним пріоритетом на всіх етапах проекту. [8]

Етапи розробки програмного забезпечення з використанням гнучких методологій

- Виявлення вимог: збір та визначення пріоритетності вимог замовника.
- Планування спринту: визначення того, що має бути досягнуто в наступному спринті.
- Розробка: команда працює над вимогами, узгодженими під час спринту.
- Рецензування та зворотній зв'язок: наприкінці спринту робота презентується клієнту для зворотного зв'язку.
- Ретроспектива: команда розмірковує над тим, що пройшло добре, а що можна покращити для наступного спринту.

Різниця між гнучкими та традиційними методологіями полягає в підході до розробки. У традиційних методологіях, таких як модель водоспаду, кожен етап повинен бути завершений, перш ніж переходити до наступного етапу. Це створює жорстку структуру, яка ускладнює внесення змін на пізніших етапах. Гнучкі методології, з іншого боку, засновані на ітераціях і дозволяють регулярно вносити зміни, адаптуватися до змін і вдосконалюватися протягом усього циклу розробки.

2.3. Scrum – фреймворк Agile

Скрам - це «структурований підхід», який базується на ітераційній та інкрементальній моделі.

Над кожним проектом працює універсальна команда експертів, до якої входять дві людини: власник продукту та скрам-майстер. Перший пов'язує команду із замовником і стежить за розвитком проекту (це не формальний керівник команди, а куратор). Другий допомагає першому організувати бізнес-процеси. Він організовує пленарні зустрічі, вирішує рутинні проблеми, мотивує команду і стежить за дотриманням підходу Scrum.

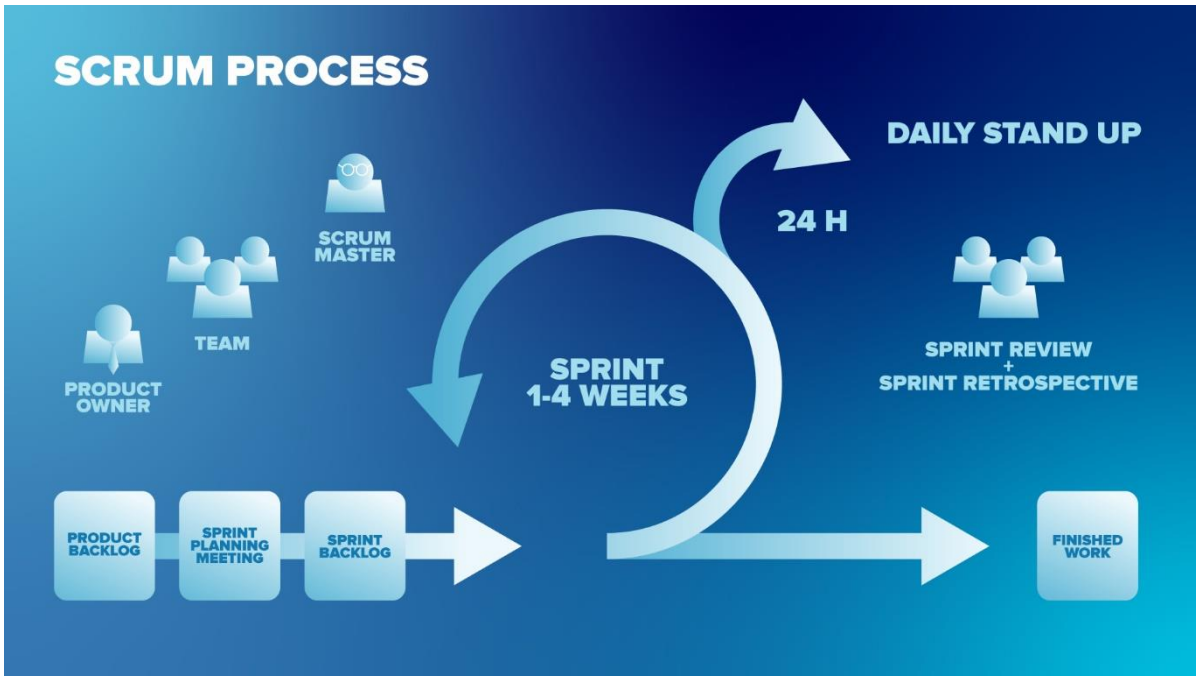


Рисунок 2.2. – Scrum sprint

Підхід Scrum ділить робочий процес на рівні спринти, які зазвичай тривають від одного тижня до одного місяця, залежно від проекту та команди. Перед початком спринту формулюються завдання для цього спринту, результати обговорюються в кінці спринту, і команда починає новий спринт. Спринти можна порівнювати між собою, що дуже корисно для управління продуктивністю.

2.4. Розробка структури системи

Розробка структури системи є важливим етапом проектування. Це пов'язано з тим, що він визначає, як компоненти системи взаємодіють для реалізації функціональності та виконання вимог.

Елементи структури системи:

1. Користувацький інтерфейс (UI) - це точка взаємодії користувача з системою. Інтерфейс включає в себе екранні форми, кнопки, меню, сповіщення та інші елементи, через які користувач може здійснювати операції з програмою. Важливо дотримуватися принципів зручності для користувачів (UX/UI), забезпечити простоту, інтуїтивність, доступність і адаптивність інтерфейсу на різних пристроях.

2. Серверна частина (Backend) - відповідає за обробку логіки програми, обробку запитів від користувачів та взаємодію з базою даних та іншими сервісами. Це "мозок" системи, де відбуваються основні операції обробки даних. Вона може бути розділена на різні модулі, такі як аутентифікація користувачів, бізнес-логіка, модулі обробки даних та інтеграція зі сторонніми сервісами.

3. База даних (Database) використовується для зберігання та управління інформацією, що використовується в системі. Це може бути що завгодно - від даних користувачів до транзакцій і журналів. Бази даних повинні бути оптимізовані для швидкого доступу, масштабованості та безпеки. Для ефективної роботи системи важливо, щоб макети баз даних, структури таблиць, індекси та зв'язки між даними були правильно визначені.

4. Зовнішні сервіси та API інтегруються з основною системою через API (інтерфейс прикладного програмування). Це дозволяє системі взаємодіяти з іншими додатками, такими як платіжні системи, поштові сервіси, сервіси автентифікації або сторонні бази даних.

5. Безпека - один з найважливіших аспектів розробки системи. Вона захищає дані, забезпечує конфіденційність і запобігає несанкціонованому доступу.

6. Після розгортання системи важливо забезпечити її моніторинг і підтримку для виявлення проблем у реальному часі та їх усунення.

7. Для успішної розробки важливо застосовувати автоматизовані процеси тестування (Unit тестування, інтеграційне тестування, тестування навантаження) для забезпечення якості та стабільності системи.

2.5. Визначення компонентів системи

Для розробки серверної частини використаємо мову програмування Java — це багатоплатформна, об’єктно-орієнтована та мережево-орієнтована мова, яку можна використовувати як платформу саму по собі. Це швидка, безпечна та надійна мова програмування для кодування будь-чого: від мобільних додатків і корпоративного програмного забезпечення до програм для великих даних і серверних технологій. [9]

Переваги:

- простота вивчення
- об'єктно-орієнтований підхід
- надійні механізми безпеки
- незалежність від платформи роблять

Для розробки клієнтської частини використаємо один із найпопулярніший фреймворків - React

React використовує модель об’єктів документа (DOM) для створення і керування інтерфейсами користувача. Методологія, яку React використовує для створення інтерфейсів користувача, є відходом від звичайних шаблонів або директив HTML. Натомість React розбиває їх на компоненти, що дозволяє розробникам створювати модульний код, який можна використовувати багаторазово та підтримувати.

Сильні сторони:

- React дозволяє створювати модульні, багаторазові та незалежні компоненти, що полегшує підтримку кодової бази
- віртуальний DOM React значно покращує продуктивність додатків, зменшуючи пряму взаємодію з реальним DOM.
- Кррозширення React для розробки мобільних додатків за допомогою React Native є революційним кроком у крос-платформенній розробці, що дозволяє створювати мобільні додатки на JavaScript.

Для для інтеграції бекенда й фронтеда в проєкті буде використовуватися SpringBoot.

Spring Boot — це інструмент із відкритим кодом, який спрощує використання фреймворків на основі Java для створення мікросервісів і веб-програм. [10]

Він має на меті оптимізувати процес розробки, надаючи попередньо налаштовані шаблони, які називаються «початковими», які обробляють такі типові завдання, як налаштування підключення до бази даних, налаштування веб-сервера або ввімкнення функцій безпеки. Ці початкові елементи дозволяють розробникам зосередитися на написанні логіки своєї програми, а не витрачати час на налаштування базової інфраструктури.

Ключові функції Spring Boot:

- Конвенція замість конфігурації: Spring Boot використовує розумні налаштування за замовчуванням та конвенції, мінімізуючи потребу в налаштуванні нових проектів. Це дозволяє розробникам швидко розпочати роботу і зменшує кількість рутинного коду.
- Автономні додатки: за допомогою Spring Boot розробники можуть створювати самодостатні Java-додатки на вбудованих веб-серверах (наприклад, Tomcat, Jetty, Undertow).
- Автоматична конфігурація: Spring Boot автоматично налаштовує додатки відповідно до бібліотек, доданих до проекту, зменшуючи потребу в ручному налаштуванні.
- Управління залежностями: Spring Boot надає попередньо сконфігуровані пакети залежностей, щоб гарантувати сумісність між компонентами та спростити процес додавання нових залежностей до проекту, та метрики додатків, що полегшує управління додатками у продуктивному середовищі.

Microsoft SQL Server (MS SQL Server) — це система управління базами даних (СУБД), розроблена компанією Microsoft. Вона забезпечує надійне та масштабоване рішення для роботи з даними у додатках різного рівня складності, від невеликих локальних систем до великих корпоративних платформ.

Однією з ключових переваг MS SQL Server є підтримка мови структурованих запитів SQL (Structured Query Language), що дозволяє виконувати операції створення, читання, оновлення та видалення даних. СУБД забезпечує ефективне управління транзакціями, що гарантує цілісність даних навіть у разі непередбачених збоїв.

MS SQL Server підтримує інтеграцію з іншими продуктами Microsoft, такими як .NET Framework, Power BI, і Azure, що дозволяє легко використовувати її у складних розподілених системах. Особливості шифрування даних та функції контролю доступу забезпечують високий рівень безпеки, що є критично важливим для сучасних додатків.

Однією з основних переваг MS SQL Server є його масштабованість. Він пропонує декілька редакцій, зокрема:

- SQL Server Express, яка є безкоштовною і підходить для невеликих проєктів.
- SQL Server Standard, оптимальну для середніх за розміром додатків.
- SQL Server Enterprise, яка містить усі доступні функції та орієнтована на великі корпоративні системи.

Також слід відзначити вбудовані механізми резервного копіювання, реплікації даних та можливості для побудови високодоступних кластерів.

MS SQL Server широко використовується у сферах електронної комерції, управління ресурсами підприємства (ERP), бізнес-аналітики та фінансових системах. Завдяки функціоналу машинного навчання та аналітики, ця СУБД дозволяє компаніям приймати більш обґрунтовані рішення на основі даних.

MS SQL Server є важливим інструментом для розробників і підприємств, які потребують ефективного управління даними. Його багатий набір функцій дозволяє створювати надійні, масштабовані та продуктивні системи, які відповідають сучасним вимогам до програмного забезпечення.

Microsoft SQL Server Management Studio (SSMS) — це інтегроване середовище для управління, конфігурації та адміністрування SQL Server. SSMS надає користувачам інтуїтивно зрозумілий інтерфейс для взаємодії з базами даних, що значно спрощує виконання широкого спектра завдань, пов’язаних з їх обслуговуванням.

SSMS також забезпечує доступ до функцій налаштування серверів, таких як безпека, автентифікація користувачів та управління ролями. Завдяки підтримці інструментів аналізу даних та інтеграції зі службами SQL Server Reporting Services (SSRS) та SQL Server Integration Services (SSIS), SSMS є універсальним інструментом для адміністраторів баз даних (DBA) і розробників.

SSMS забезпечує простоту у використанні завдяки графічному інтерфейсу, що значно знижує поріг входу для нових користувачів. Підсвітка синтаксису та інструменти автозавершення допомагають зменшити кількість помилок при написанні запитів.

Інструмент також підтримує ведення логів активності, що дозволяє легко відстежувати зміни у базах даних. Завдяки функціям аналізу продуктивності, SSMS допомагає ідентифікувати та усувати проблеми, пов’язані з повільною роботою запитів.

SSMS є стандартним інструментом для адміністраторів і розробників SQL Server. Він дозволяє виконувати як рутинні завдання (створення резервних копій, моніторинг серверів), так і більш складні операції, наприклад,

оптимізацію продуктивності SQL-запитів. SSMS також зручно використовувати для навчання, оскільки він надає доступ до всіх компонентів SQL Server у єдиному середовищі.

SSMS є важливим компонентом екосистеми SQL Server. Він надає повний набір інструментів для ефективного управління базами даних, забезпечуючи інтеграцію з іншими продуктами Microsoft. Завдяки своїй гнучкості, багатофункціональності та простоті використання, SSMS залишається незамінним інструментом для професіоналів, які працюють з SQL Server.[11]

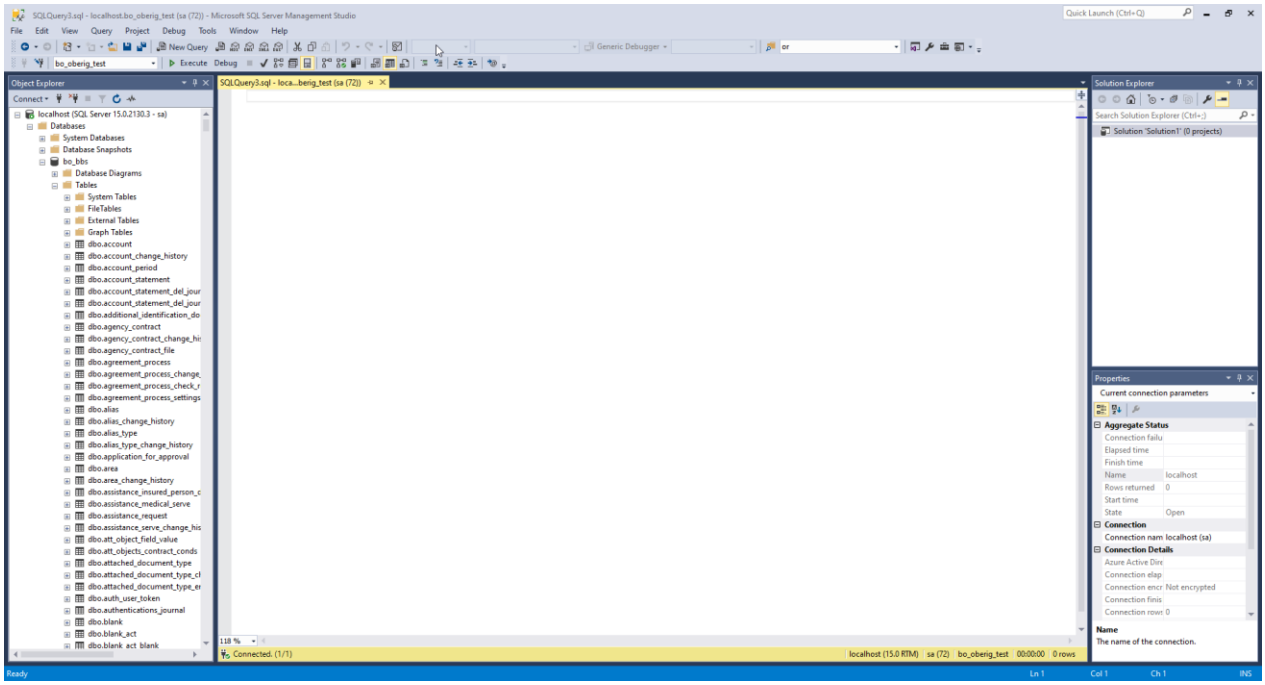


Рисунок 2.3. – Інтерфейс SSMS

Gradle — це інструмент автоматизації збірки для розробки багатомовного програмного забезпечення. Він контролює процес розробки в завданнях компіляції та упаковки для тестування, розгортання та публікації. Gradle забезпечує гнучкість у налаштуванні процесів побудови та управлінні залежностями, використовуючи декларативний стиль та сценарії, що забезпечує високу продуктивність і можливість масштабування.

Переваги використання Gradle:

- Легкість у використанні

Gradle має чітко організований та інтуїтивно зрозумілий синтаксис, що спрощує процес роботи з ним. Розробникам легко налаштувати та запустити проект, а його потужні можливості допомагають зменшити навантаження на команду та спростити розробку.

- Гнучкість і налаштування

Gradle пропонує широкий спектр налаштувань, що дозволяє адаптувати його під індивідуальні потреби проекту. Інструмент підтримує розширення і налаштування, що дає змогу оптимізувати роботу і досягти високої продуктивності для будь-якого типу проекту.

- Інтеграція з іншими інструментами

Gradle легко взаємодіє з популярними інструментами розробки, такими як IDE (IntelliJ IDEA, Eclipse, Android Studio) та системами контролю версій (Git, SVN). Це дозволяє ефективно керувати проектом і виконувати процеси побудови без зайвих ускладнень.

- Підтримка різних мов програмування

Як багатомовний інструмент, Gradle сумісний з різними мовами програмування, що надає розробникам широку свободу вибору і дозволяє застосовувати його у проектах з різними мовами та технологіями.

Git — це швидка, масштабована розподілена система контролю версій із надзвичайно багатим набором команд, який забезпечує як операції високого рівня, так і повний доступ до внутрішніх елементів. [12] Git зазвичай використовується як для розробки програмного забезпечення з відкритим кодом, так і для комерційного програмного забезпечення, що приносить значні переваги окремим особам, командам і компаніям.

- Git дозволяє рбачити всю хронологію своїх змін, рішень і прогресу будь-якого проекту в одному місці. З моменту доступу до історії проекту розробник має весь необхідний контекст, щоб зрозуміти його та почати робити внесок.

- Кожна працювати у будь-якому часовому поясі. З DVCS, таким як Git, співпраця може відбутися будь-коли, зберігаючи цілісність вихідного коду. Використовуючи гілки, розробники можуть безпечно пропонувати зміни до робочого коду.

- Компанії, які використовують Git, можуть зруйнувати комунікаційні бар’єри між командами та зосередити їх на виконанні найкращої роботи. Крім того, Git дає змогу об’єднати експертів у всьому бізнесі для співпраці над великими проектами.

2.6. Мікросервісна архітектура

Мікросервіс — це невеликий, слабко пов’язаний сервіс, призначений для виконання певної бізнес-функції, і кожен мікросервіс можна розробляти, розгортати та масштабувати незалежно.[13]

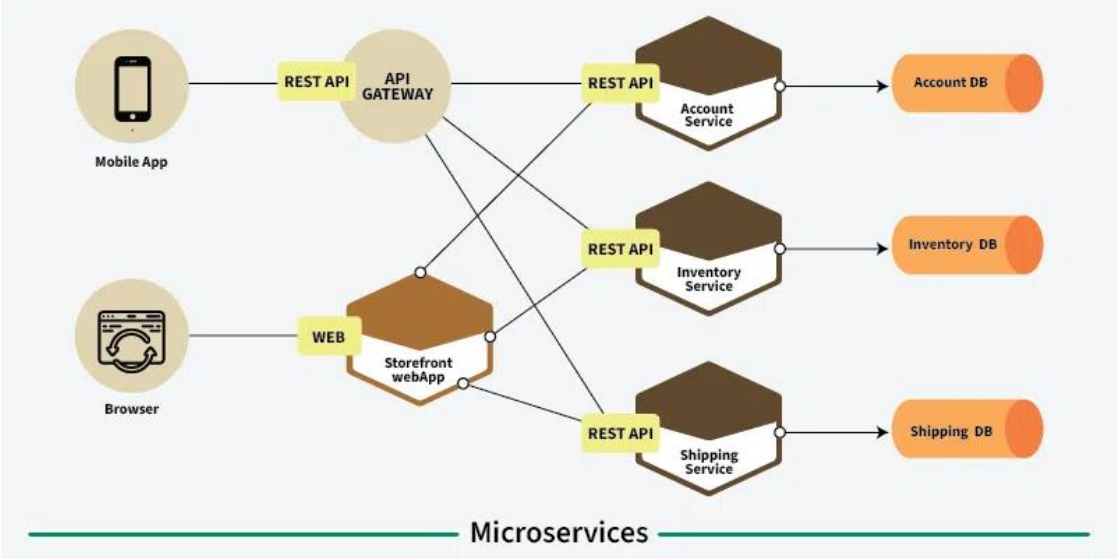


Рисунок 2.4. – Microservices

- Ця архітектура дозволяє взяти великий монолітний додаток і розкласти його на невеликі керовані компоненти/сервіси. Крім того, він вважається будівельним блоком сучасних програм.
- Мікросервіси можуть бути написані на різних мовах програмування та фреймворках, і кожен сервіс діє як міні-додаток самостійно.

Переваги:

- Програми складаються з незалежних служб, кожна з яких виконує окрему функцію, що спрощує процеси розробки та підтримки

- Кожен мікросервіс відповідає за конкретну бізнес-задачу, наприклад, автентифікацію користувачів або управління продуктами, забезпечуючи спеціалізацію підходу до розробки
- Служби взаємодіють через API, що забезпечує стандартизований обмін даними та легку інтеграцію.
- Різні мікросервіси можуть використовувати різні технології, дозволяючи командам обирати найбільш підходящі інструменти для своїх завдань.
- Мікросервіси можуть оновлюватися автономно, що мінімізує ризики під час внесення змін і підвищує загальну стабільність системи.

API - це набір програмного коду, який забезпечує передачу даних між одним програмним продуктом та іншим. Він також містить умови цього обміну даними.

Інтерфейс прикладного програмування необхідно чітко відрізняти від інтерфейсу користувача. Інтерфейс користувача отримує дані від користувача, передає їх API для обробки і повертає результати користувачеві; API не взаємодіє з користувачем, а обробляє дані, отримані від одного модуля програми, і повертає результати іншому модулю.

Веб- API - це інтерфейс прикладного програмування, призначений для спілкування між веб-сервером і клієнтом, наприклад, веб-браузером. У контексті веб-розробки цей термін може охоплювати клієнтську частину веб-додатку, включаючи використання відповідних веб-фреймворків. На стороні сервера веб-API складається із загальнодоступної кінцевої точки, яка підтримує обмін повідомленнями у форматі запит-відповідь, зазвичай у вигляді JSON або XML, через протокол HTTP. API, створений для використання всередині сервера (SAPI), не вважається веб-API, якщо він не забезпечує доступ до віддаленого веб-додатку.

REST розшифровується як Representational State Transfer (передача представницького стану) і є архітектурним стилем для створення веб-сервісів,

які взаємодіють за допомогою протоколу HTTP. Його принципи були сформульовані комп'ютерним вченим Роєм Філдінгом у 2000 році і стали популярними як масштабована і гнучка альтернатива старим методам міжмашинної комунікації. Він залишається золотим стандартом для публічних API.

Основними елементами парадигми REST API є:

- Клієнт або програмне забезпечення, яке запускається на комп'ютері або смартфоні користувача та ініціює зв'язок
- Сервер, який надає API як засіб доступу до даних та функціональності
- Будь-який контент, який сервер може надати клієнту (наприклад ресурси, наприклад, відео, текстові файли тощо). [14]

Принципи REST базуються на клієнт-серверній архітектурі, де запити є безстаними, а відповіді можуть кешуватися для підвищення продуктивності. Уніфікований інтерфейс забезпечує стандартизовану взаємодію, ресурси мають унікальні URI, а дані передаються у форматах JSON або XML. Архітектура побудована з чітким розподілом на шари, що сприяє простоті, масштабованості та ефективності системи.

Коли клієнтський запит робиться через RESTful API, він передає представлення стану ресурсу запитувачу або кінцевій точці. Ця інформація або представлення надається в одному з кількох форматів через HTTP: JSON (нотація об’єктів Javascript), HTML, XLT, Python, PHP або звичайний текст. JSON є найпопулярнішим форматом файлів для використання, оскільки, незважаючи на свою назву, він не залежить від мови, а також читається як людьми, так і машинами.

2.7. Model-View-Controller

Model-View-Controller (MVC) - це архітектурний шаблон, який розділяє додаток на три основні компоненти: моделі, види та контролери. Моделі відповідають за управління даними, логікою та бізнес-правилами. Представлення відповідають за відображення інформації для користувачів, тоді як контролери передають вхідні дані користувача моделям і

представленням для обробки. Такий підхід покращує модульність і робить додатки простішими в обслуговуванні та розширенні.

Spring MVC Framework слідує шаблону архітектури Model-View-Controller , який працює навколо переднього контролера, тобто диспетчерського сервлета . Сервлет диспетчера обробляє та відправляє всі вхідні HTTP-запити до відповідного контролера

Цей фреймворк ви можете використовувати будь-який об’єкт як команду або об’єкт підтримки форми; вам не потрібно реалізовувати специфічний для фреймворку інтерфейс або базовий клас. Зв’язування даних Spring є дуже гнучким: наприклад, воно розглядає невідповідності типів як помилки перевірки, які можуть бути оцінені програмою, а не як системні помилки. Таким чином, вам не потрібно дублювати властивості ваших бізнес-об’єктів як прості нетипові рядки в об’єктах форми просто для обробки недійсних повідомлень або належного перетворення рядків. Замість цього часто краще прив’язувати безпосередньо до ваших бізнес-об’єктів.

Вся логіка Spring MVC побудована навколо DispatcherServlet, який отримує й обробляє всі HTTP-запити (від інтерфейсу користувача) і відповіді на них.

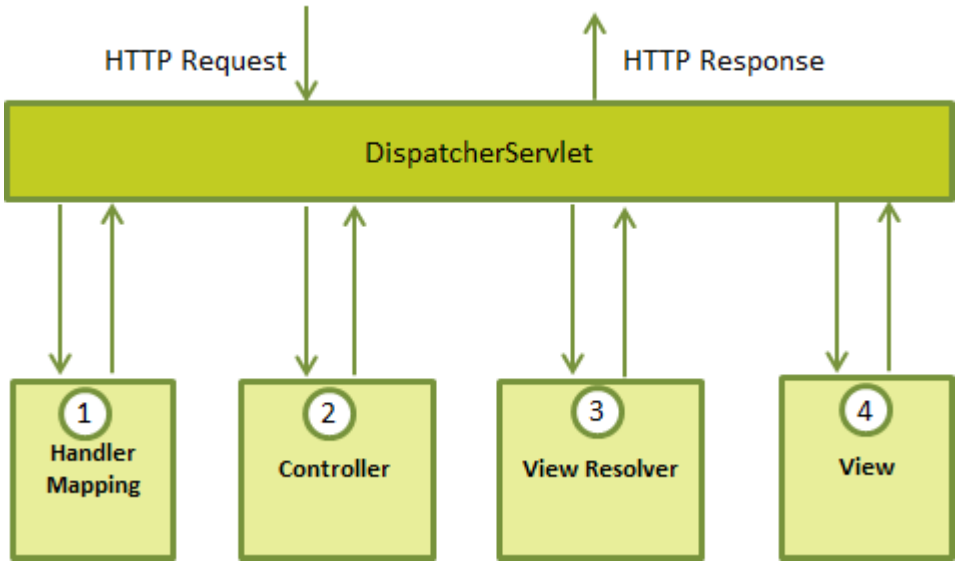


Рисунок 2.5. – Архітектура роботи MVC

1. Після отримання HTTP-запиту DispatcherServlet звертається до HandlerMapping, який визначає відповідний контролер для обробки запиту, і перенаправляє його туди.
2. Контролер обробляє запит, викликаючи відповідний метод (GET або POST), який, виконуючи бізнес-логіку, формує дані моделі та повертає DispatcherServlet ім'я представлення (View).
3. DispatcherServlet, використовуючи ViewResolver, визначає представлення на основі отриманого імені.
4. Потім він передає дані моделі у вигляді атрибутів у створене представлення, яке виводиться в браузері користувача.

2.8. Розробка бази даних

Розробка баз даних - це складний, багатогранний процес, який враховує широкий спектр технічних, бізнес- та функціональних аспектів. Він передбачає не лише створення схеми бази даних, але й забезпечення її ефективного використання, обслуговування, безпеки та масштабованості. Цей процес охоплює кілька ключових аспектів:

1. Аналіз вимог - це початковий етап, на якому зб ираються всі необхідні дані про проект. Визначаються бізнес-вимоги, які описують завдання, що має вирішувати система, наприклад, зберігання даних про клієнтів, обробка транзакцій або аналітика. Також встановлюються функціональні вимоги, які включають операції, що будуть виконуватись над даними, такі як пошук, оновлення, видалення або звітність. Технічні вимоги визначають обсяги даних, очікувану кількість запитів, час відповіді та доступність системи. Крім того, визначаються потенційні користувачі системи та їх ролі і права, наприклад, адміністратори, аналітики або клієнти.

2. Логічнне проєктування фокусується на моделюванні даних. На цьому етапі визначаються сутності, такі як клієнти та замовлення, а також атрибути, такі як імена та дати замовлень. Описуються зв'язки між сутностями (один-до-одного, один-до-багатьох, багато-до-багатьох) і встановлюються обмеження, такі як обов'язкові поля та унікальність значень.

3. Фізичне проєктування стосується реалізації бази даних у конкретній СУБД. Типи даних обираються відповідно до СУБД, наприклад,

VARCHAR, INTEGER, DATE тощо. Кожна таблиця визначає первинний ключ, який однозначно ідентифікує запис, і зовнішній ключ для зв'язку між таблицями. Індекси додаються для оптимізації пошуку та скорочення часу відповіді на запити.

4. Для забезпечення продуктивності бази даних використовуються кластерні та некластерні індекси для прискорення пошуку, розробляються стратегії кешування для зменшення навантаження, використовується шардинг для розподілу та розширення даних між декількома серверами, а також висока доступність для забезпечення високої доступності та зменшення ризику збоїв. Для забезпечення високої доступності та зниження ризику збоїв використовуються такі методи, як конфігурації реплікації.

5. Безпека бази даних є однією з основних апекрів. Важливо визначити ролі користувачів і права доступу, використовувати шифрування для захисту даних при зберіганні та передачі, проводити аудит і реєструвати зміни даних і доступ до них, а також регулярно створювати резервні копії для запобігання втраті даних.

6. Етап тестування. Треба перевірити її роботу за кількома напрямками: Тести продуктивності для оцінки швидкості виконання запитів під навантаженням, виявлення структурних помилок і помилок бізнес-логіки, перевірки цілісності даних і дотримання бізнес-правил.

7. Підтримка та вдосконалення включає регулярний моніторинг продуктивності, оптимізацію запитів для забезпечення ефективності, оновлення структур у міру зміни бізнес-процесів, а також видалення старих і неактуальних даних для підвищення продуктивності та зменшення розміру бази даних.

Проектування бази даних вимагає міждисциплінарного підходу, який включає знання бізнес-процесів, програмування, адміністрування СУБД та забезпечення безпеки. Правильно спроєктована база даних є основою ефективної роботи програмного забезпечення.[15]

2.9. Етап датологічного та фізичного проектування бази данних

Етап даталогічного проєктування є одним із ключових у процесі створення бази даних і передує фізичній реалізації. На цьому етапі визначаються конкретні деталі структури даних, зокрема таблиці, їхні атрибути, типи даних, правила цілісності та інші характеристики бази даних.

Після визначення сутностей та встановлення взаємозв'язків між ними формується детальна даталогічна модель, яка описує основну бізнес-логіку системи.

Створення фізичної моделі даних є одним із ключових етапів розробки бази даних. На цьому етапі зв’язки, визначені в логічній моделі, перетворюються на таблиці, а атрибути — на стовпці цих таблиць. У реляційних СУБД для ключових атрибутів додаються унікальні індекси, а визначені раніше домени перетворюються на типи даних, які підтримує обрана СУБД.

Фізична модель також включає реалізацію обмежень, що були сформульовані під час логічного проєктування. Це можуть бути індекси, декларативні обмеження цілісності, тригери, збережені процедури та інші механізми, які забезпечують цілісність і узгодженість даних. Вибір інструментів залежить від функціональних можливостей конкретної СУБД.

На етапі розробки фізичної моделі необхідно врахувати такі аспекти, як ефективність організації таблиць, правильність вибору індексів та обсяг програмного коду, необхідного для підтримки цілісності даних. Важливо переконатися, що таблиці спроєктовані оптимально, індекси відповідають вимогам до продуктивності, а програмні механізми підтримують необхідні бізнес-правила.

Результатом фізичного проєктування є база даних, реалізована на певній програмно-апаратній платформі. Вибір платформи суттєво впливає на

продуктивність системи. Це включає вибір типу серверів, налаштування апаратної частини, визначення кількості процесорів, обсягу оперативної пам’яті та конфігурації дискових підсистем. Також важливим є налаштування самої СУБД для максимально ефективної роботи в обраній інфраструктурі.

Таким чином, рішення, прийняті на кожному етапі моделювання та проєктування, значною мірою впливають на подальшу роботу бази даних. Тому важливо ще на ранніх стадіях приймати зважені рішення, щоб забезпечити високу продуктивність і надійність системи.[16]

Детальний опис структури таблиць бази даних наведено в додатку А.

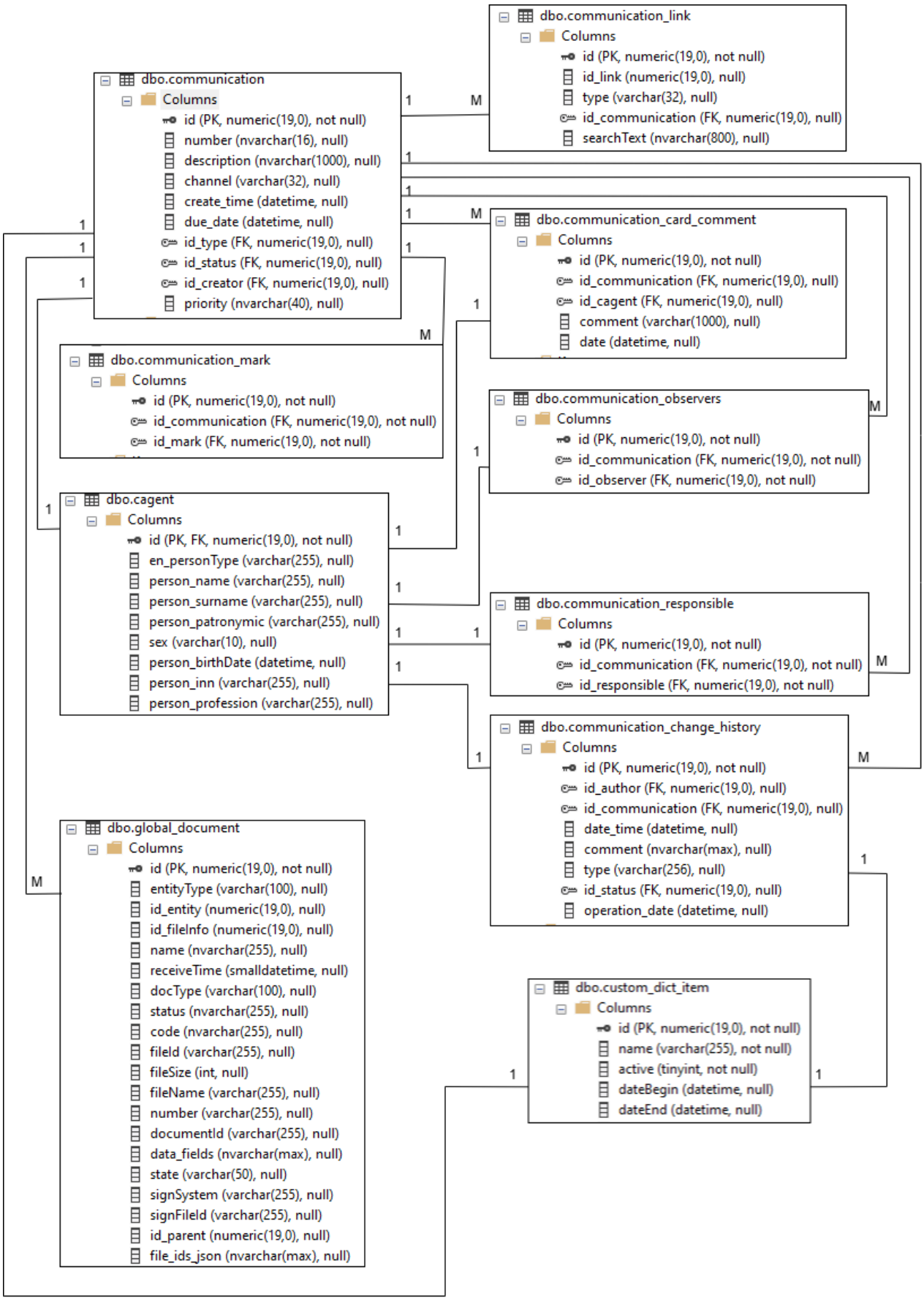


Рисунок 2.6. – Даталогічна модель

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСОБУ

3.1. Середовище розробки

IntelliJ IDEA — це інтегроване середовище розробки (IDE), створене компанією JetBrains, яке стало одним із найпопулярніших рішень для програмістів, що працюють з Java. Цей інструмент забезпечує широкий спектр функцій, які значно спрощують та прискорюють процес створення, тестування й обслуговування програмного забезпечення.

Основні функції IntelliJ IDEA:

1. Інтелектуальний редактор коду: автоматичне завершення, перевірка синтаксису, швидкі рекомендації та зручне рефакторування допомагають розробникам писати якісний код.

2. Підтримка фреймворків: середовище підтримує популярні Java-фреймворки, такі як Spring, Hibernate, JavaFX, що дає змогу автоматично налаштовувати проекти та використовувати вбудовані інструменти.

3. Засоби для відладки: IntelliJ IDEA включає відлагоджувач, який підтримує встановлення точок зупинки, виконання коду по кроках і аналіз стеку викликів.

4. Інтеграція з системами управління залежностями: завдяки підтримці Maven і Gradle, середовище забезпечує зручне керування залежностями та їх автоматичне оновлення.

5. Контроль версій: інтеграція з системами версіонування (Git, SVN) дозволяє здійснювати коміти, мерджі, відстежувати зміни та працювати з відгалуженнями.

6. Інструменти аналізу та рефакторингу: IntelliJ IDEA допомагає виявити проблеми у коді та пропонує автоматизовані інструменти для їх виправлення.

Однією з головних переваг IntelliJ IDEA є підтримка різних версій Java, починаючи з Java 8 і до новітніх релізів. Це дає змогу розробникам використовувати нові функції мови та водночас підтримувати проекти, які базуються на старіших версіях.

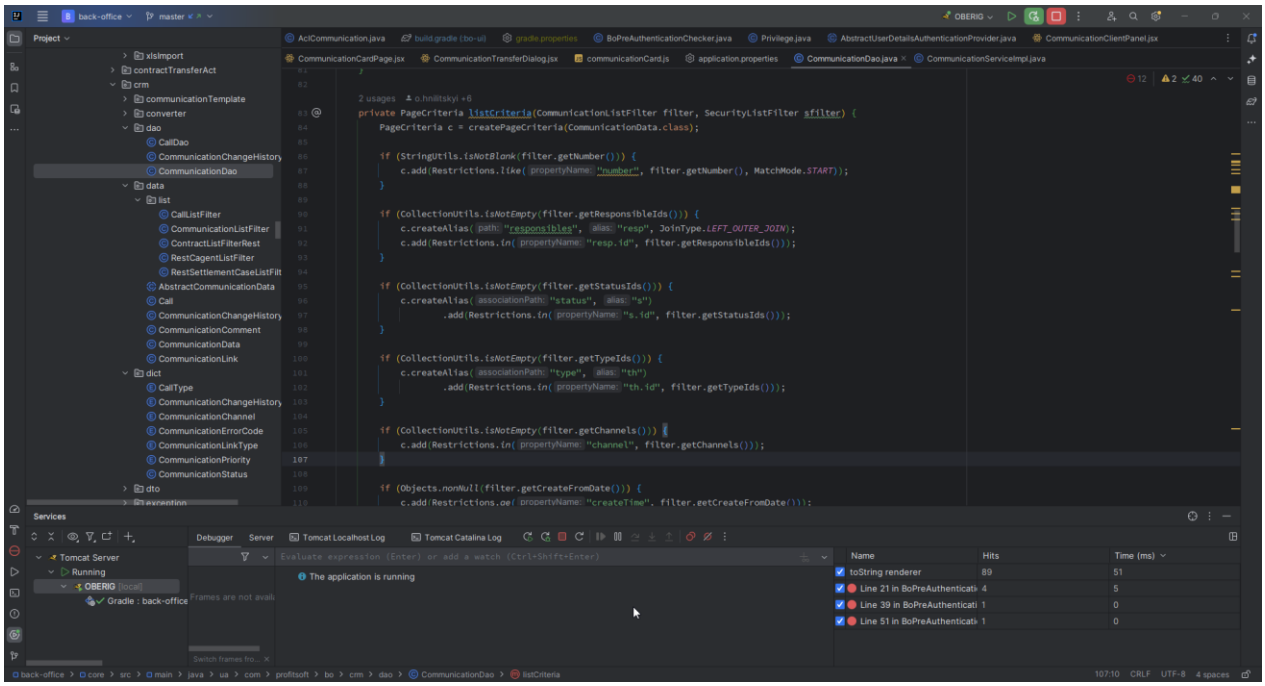


Рисунок 3.1. – Інтерфейс IntelliJ IDEA

Для ефективної розробки з використанням IntelliJ IDEA необхідно правильно налаштувати середовище. У цьому проекті використовується Gradle версії 7.3 і JDK Java 17. Цей вибір обумовлений:

- *Актуальністю*: Java 17 є довгостроковою версією підтримки (LTS), що забезпечує стабільність і регулярні оновлення безпеки.
- *Сумісністю*: сучасні фреймворки й бібліотеки активно підтримують Java 17, що дає змогу використовувати новітні функції мови.

Gradle 7.3 повністю сумісний із Java 17 і забезпечує потужну автоматизацію збирання проектів, управління залежностями та виконання тестів. JDK 17 відкриває доступ до нових можливостей, таких як шаблони для switch, запечатані класи (sealed classes) та оновлений API для роботи з пам’яттю.

IntelliJ IDEA легко інтегрується з різними сучасними інструментами, такими як сервери виконання (наприклад, Apache Tomcat), системи CI/CD (Jenkins, GitHub Actions), а також розширюється через численні плагіни. Це

робить її універсальним середовищем для командної розробки, автоматизації процесів і роботи з новими фреймворками, які підтримують Java 17.

IntelliJ IDEA залишається потужним інструментом для розробки Java-додатків. Її інтеграція з актуальними версіями Gradle і JDK 17 дає змогу розробникам використовувати сучасні можливості мови, що підвищує ефективність створення програмного забезпечення. Використання IntelliJ IDEA у поєднанні з JDK 17 і Gradle 7.3 забезпечує гнучкість і надійність у роботі над проектами будь-якої складності.[17]

3.2. Опис призначеного для користувача інтерфейсу

Початок роботи з CRM-модулем починається з меню системи, яке містить випадаючий список із функціоналом CRM. До основних компонентів цього меню належать: комунікації, шаблони комунікацій та процедури.

Головним елементом є сторінка "Комунікації", яку розглянемо детальніше.

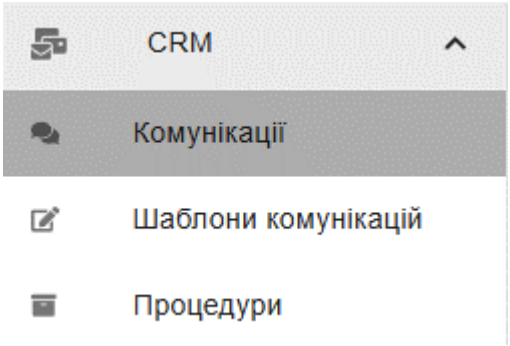


Рисунок 3.2. – Меню модуля CRM

На головній сторінці розташовані два основних блоки: фільтр (1) та таблиця з результатами пошуку комунікацій (4). Укорочений блок з полями фільтрації дозволяє здійснювати пошук комунікацій за такими критеріями, як номер комунікації, відповідальна особа, статуси, мітки, автор створення та чекбокс для фільтрації за ознакою "протерміновані чи ні" комунікації.

Також присутні функціональні кнопки (2). Перша з них — кнопка передачі комунікації, яка активна лише тоді, коли вибрано одну або кілька комунікацій за допомогою чекбоксу. Наприклад, для комунікації з номером

«009» ця кнопка дозволяє передавати або розподіляти справи між відповідальними користувачами. Друга кнопка дозволяє завантажити результати пошуку у форматі XLS. Третя кнопка відкриває модальне вікно з додатковими полями фільтрації для більш точного пошуку. У точному пошуку можна вказати такі критерії, як типи, джерела, взаємозв'язки, номер взаємозв'язку, спостерігачі, пріоритет, радіокнопка для визначення кількості відповідальних осіб, чекбокс для фільтрації за дубльованими комунікаціями, дата створення та термін виконання.

На фільтрі також відображається лічильник знайдених комунікацій (3).

Нижче розташована таблиця з результатами пошуку (4), що містить такі поля, як чекбокс для вибору комунікації, джерело, номер, тип, опис, взаємозв'язки, дата створення, термін виконання, автор створення, відповідальні особи, статус, пріоритет і мітки. Це основні поля, які відображаються у таблиці. При експорті в формат XLS, файл містить більш детальну інформацію.

У правому нижньому кутку сторінки знаходиться кнопка для створення нової комунікації (5).

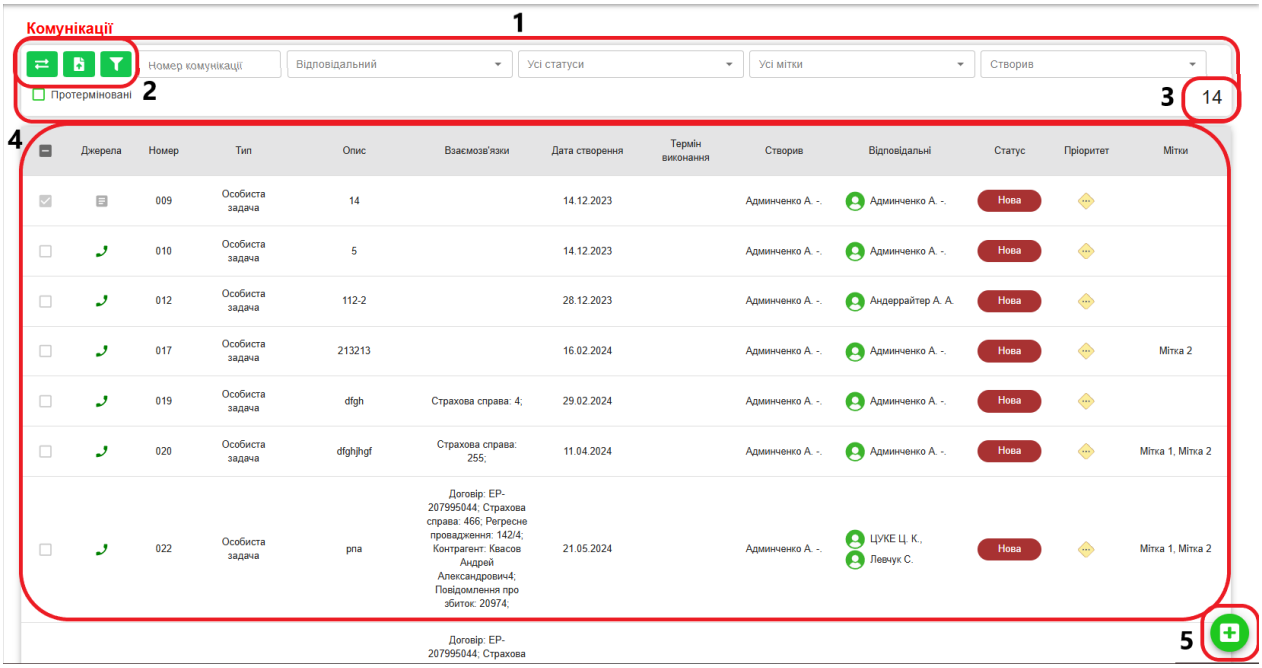
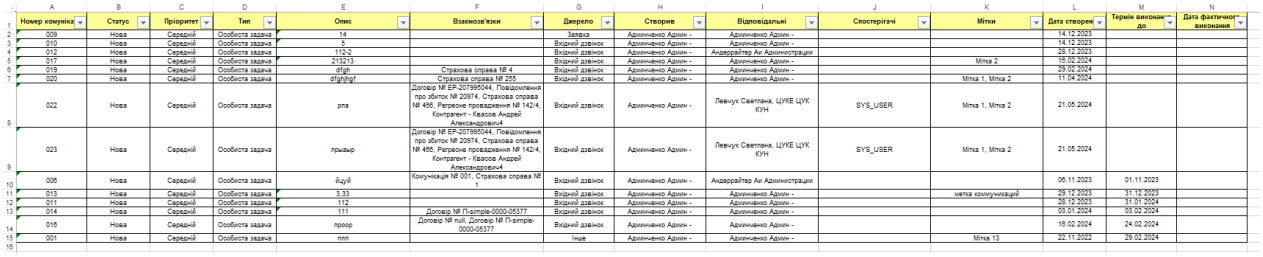


Рисунок 3.3. – Головна сторінка модуля «Комунікації»



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Номер комунікації	Статус	Пріоритет	Тип	Опис	Взаємозв'язки	Джерело	Створив	Відповідальні	Спостерігачі	Мітки	Дата створення	Термін виконання до	Дата фактичного виконання
2	009	Нова	Середній	Особиста задача	14		Заявка	Адміненко Адмін -	Адміненко Адмін -			14.12.2023		
3	010	Нова	Середній	Особиста задача	5		Вхідний дзвінок	Адміненко Адмін -	Адміненко Адмін -			14.12.2023		
4	012	Нова	Середній	Особиста задача	112-2		Вхідний дзвінок	Адміненко Адмін -	Андеррайтер Ан Администрации			28.12.2023		
5	017	Нова	Середній	Особиста задача	213213		Вхідний дзвінок	Адміненко Адмін -	Адміненко Адмін -		Мітка 2	16.02.2024		
6	019	Нова	Середній	Особиста задача	dfgh	Страхова справа № 4	Вхідний дзвінок	Адміненко Адмін -	Адміненко Адмін -			29.02.2024		
7	020	Нова	Середній	Особиста задача	dfghjhgf	Страхова справа № 255	Вхідний дзвінок	Адміненко Адмін -	Адміненко Адмін -		Мітка 1, Мітка 2	11.04.2024		
8	022	Нова	Середній	Особиста задача	pna	Договір № EP-207995044, Повідомлення про збиток № 20974, Страхова справа № 466, Регресне провадження № 142/4, Контрагент - Квасов Андрей Александрович4	Вхідний дзвінок	Адміненко Адмін -	Левчук Светлана, ЦУКЕ ЦУК КУН	SYS_USER	Мітка 1, Мітка 2	21.05.2024		
9	023	Нова	Середній	Особиста задача	прыыр	Договір № EP-207995044, Повідомлення про збиток № 20974, Страхова справа № 466, Регресне провадження № 142/4, Контрагент - Квасов Андрей Александрович4	Вхідний дзвінок	Адміненко Адмін -	Левчук Светлана, ЦУКЕ ЦУК КУН	SYS_USER	Мітка 1, Мітка 2	21.05.2024		
10	006	Нова	Середній	Особиста задача	йцуй	Комунікація № 001, Страхова справа № 1	Вхідний дзвінок	Адміненко Адмін -	Андеррайтер Ан Администрации			06.11.2023	01.11.2023	
11	013	Нова	Середній	Особиста задача	3.33		Вхідний дзвінок	Адміненко Адмін -	Адміненко Адмін -		мітка комунікацій	29.12.2023	31.12.2023	
12	011	Нова	Середній	Особиста задача	112		Вхідний дзвінок	Адміненко Адмін -	Адміненко Адмін -			28.12.2023	31.01.2024	
13	014	Нова	Середній	Особиста задача	111	Договір № П-simple-0000-05377	Вхідний дзвінок	Адміненко Адмін -	Адміненко Адмін -			03.01.2024	03.02.2024	
14	016	Нова	Середній	Особиста задача	прооор	Договір № null, Договір № П-simple-0000-05377	Вхідний дзвінок	Адміненко Адмін -	Адміненко Адмін -			16.02.2024	24.02.2024	
15	001	Нова	Середній	Особиста задача	nnn		Інше	Адміненко Адмін -	Адміненко Адмін -		Мітка 13	22.11.2022	29.02.2024	
16														

Рисунок 3.4. – Визгрузка списку комунікацій у xls-файл

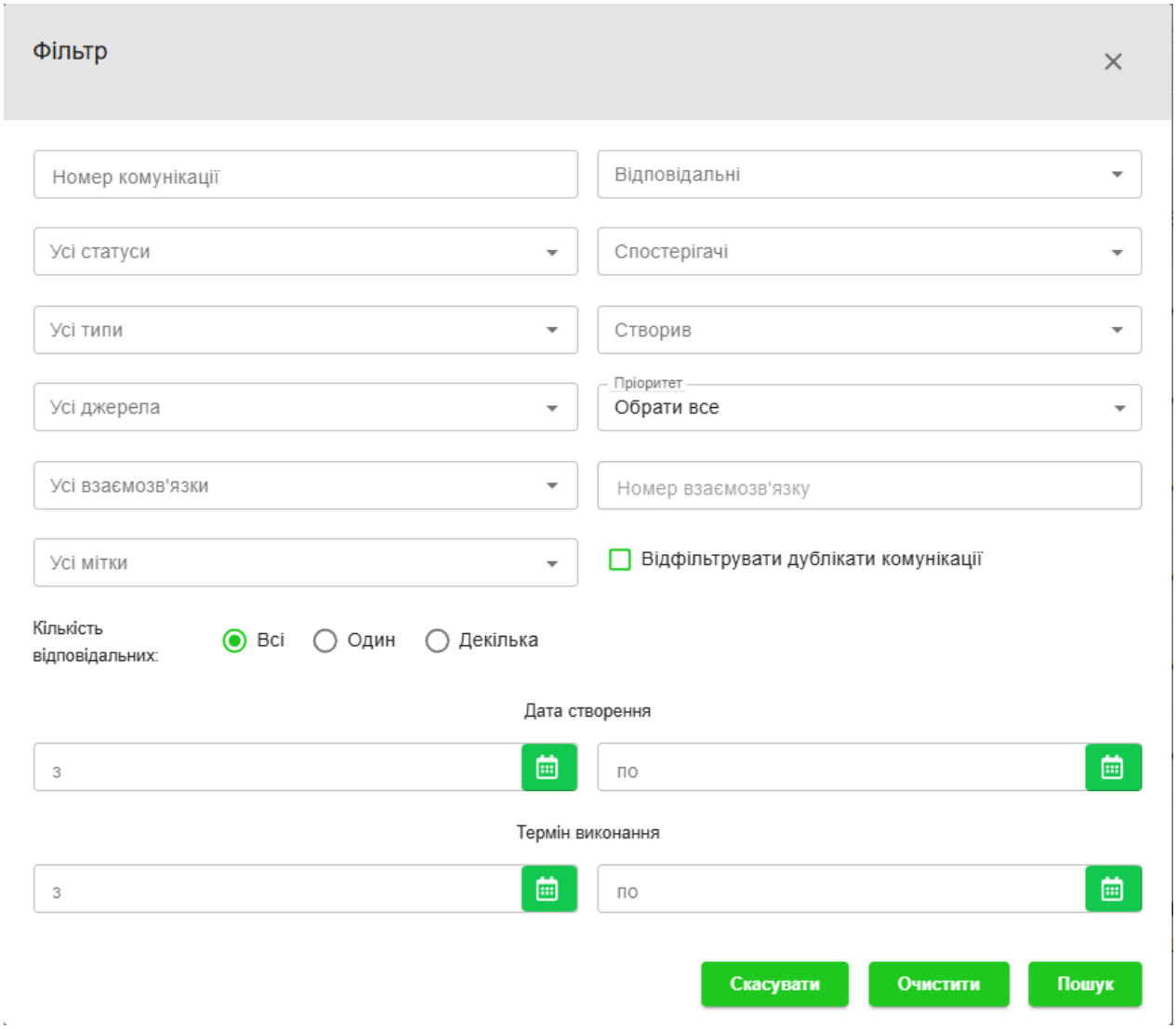


Рисунок 3.5. – Модальне вікно «Фільтр»

Переглянемо сторінку комунікації, на якій спочатку відображається номер комунікації та дата її створення. З цією комунікацією можна виконувати доступні дії: редагувати, передавати, створювати копію або видаляти. Для цих дій є відповідні кнопки внизу сторінки.

Сторінка містить три вкладки: «Загальне», «Документи» та «Коментарі». Розглянемо вкладку «Загальне». У блоці «Відомості про комунікацію»

розташовані основні поля, пов'язані з комунікацією, які ми бачили на головній сторінці. Основним блоком для роботи з комунікаціями є блок «Взаємозв'язки», у якому закріплюються різні справи або контрагенти, з якими працює страхова компанія. Завдяки цьому функціоналу ми можемо прив'язувати договори, страхові справи, повідомлення про збитки, регресні провадження та контрагентів. Для кожної з цих сутностей є гіперпосилання, через яке можна перейти і переглянути необхідну інформацію, що потрібна для роботи співробітників страхової компанії.

Якщо цей блок порожній, то через кнопку «Додати», при натисканні на яку відкриється модальне вікно, можна додати необхідні сутності, які вже є в системі. Пошук реалізований через функцію «Suggestion Search», яка автоматично пропонує користувачеві варіанти пошукових запитів під час введення тексту в рядок пошуку. Ця технологія використовується в багатьох сучасних пошукових системах і додатках, щоб підвищити зручність і швидкість взаємодії з системою.

Якщо ж у системі немає необхідних даних, то в цьому ж модальному вікні можна створити нового користувача (контрагента) або на загальній вкладці одразу оформити новий договір. Для цього передбачена кнопка «Оформити договір», яка перенесе користувача до функціоналу оформлення будь-якого можливого договору страхової компанії.

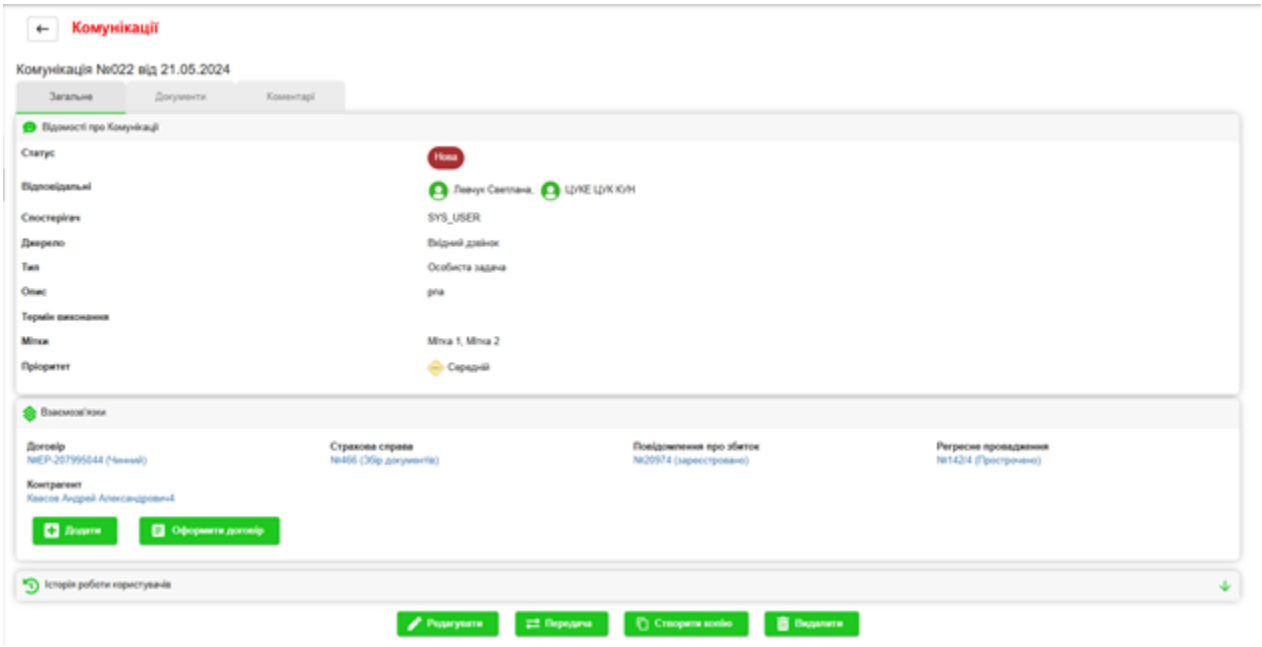


Рисунок 3.6. – Сторінка з комунікацією

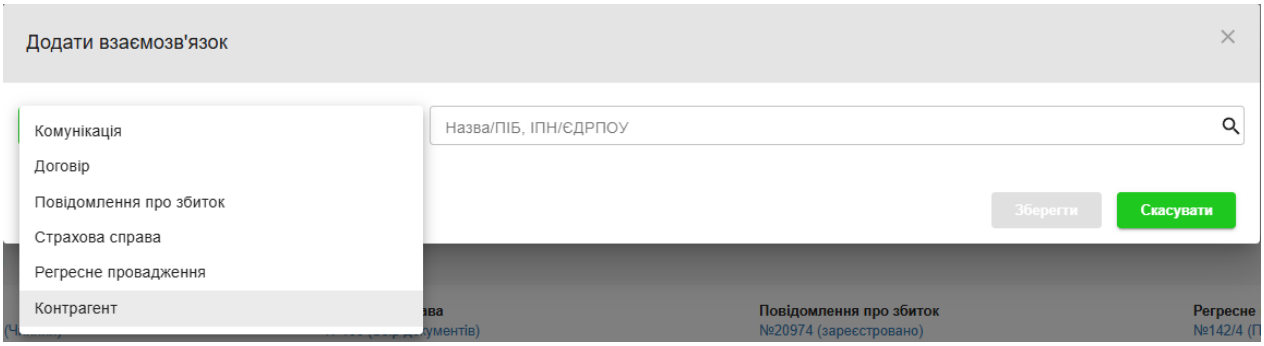


Рисунок 3.7. – Модальне вікно с додаваням взаємосз’язків

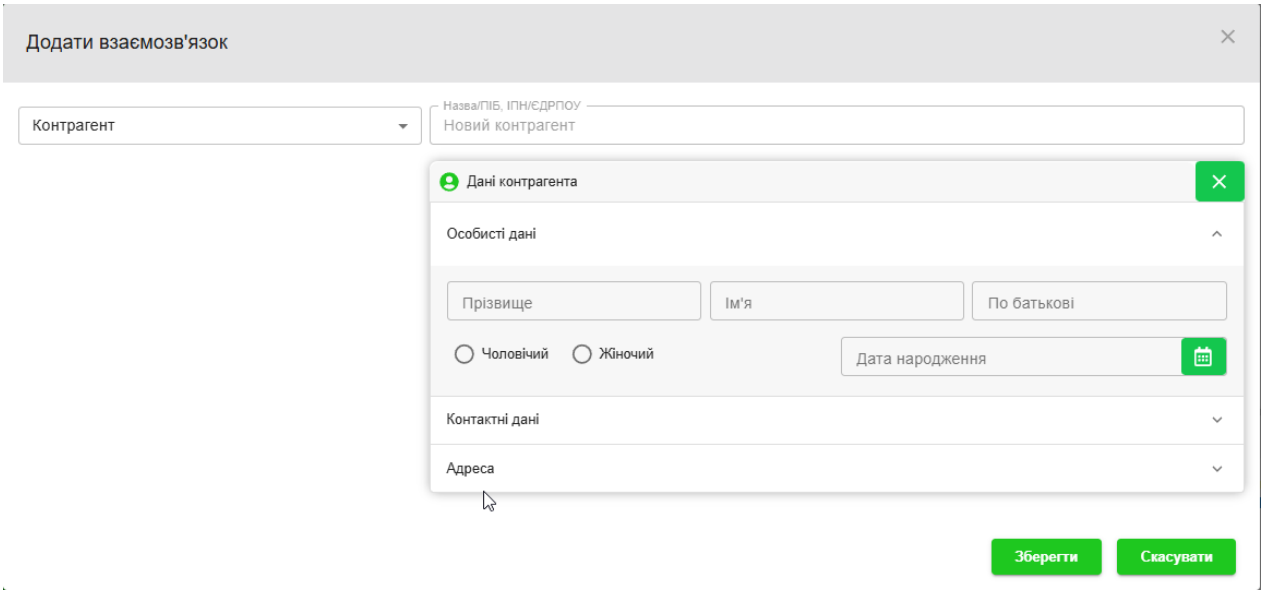


Рисунок 3.8. – Модальне вікно с додаваням взаємосз’язків, створення нового контрагента

Є менш важливий блок під назвою «Історія роботи користувачів», який має функцію розгортання. Цю можливість можна зрозуміти за стрілкою у правому верхньому куті блоку, яка змінює свій напрямок залежно від поточного стану блоку (розгорнутий чи згорнутий).

У цьому блоці відображається інформація про те, хто працював із даною комунікацією, які зміни були внесені, статус, відповідальні особи, дата змін і коментарі щодо виконаних дій. У разі внесення будь-яких змін ці дані автоматично фіксуються та відображаються в історії роботи, що дозволяє легко відстежити всі дії, виконані з комунікацією.

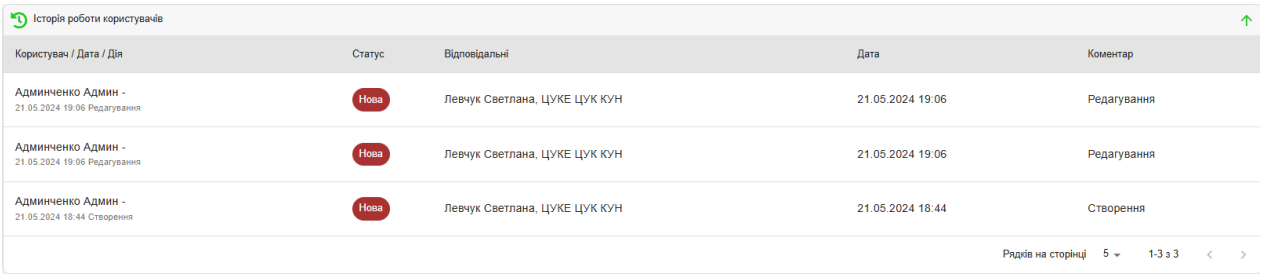


Рисунок 3.9. – Історія роботи користувачів

Друга вкладка також є надзвичайно важливою та функціональною, оскільки в ній можна додавати та зберігати будь-які види документів, пов’язані з цією комунікацією. Це дуже зручно, адже вся необхідна інформація зосереджена в одному місці, що значно спрощує її подальше використання або розподіл за потреби.

Рисунок 3.10. – Вкладка «Документи»

Третя вкладка «Коментарі» є допоміжною і призначена для перегляду та додавання коментарів, пов’язаних із комунікацією. Для додавання коментаря

передбачена кнопка «Додати коментар», натискання якої відкриває модальне вікно, де можна ввести та зберегти потрібний коментар.

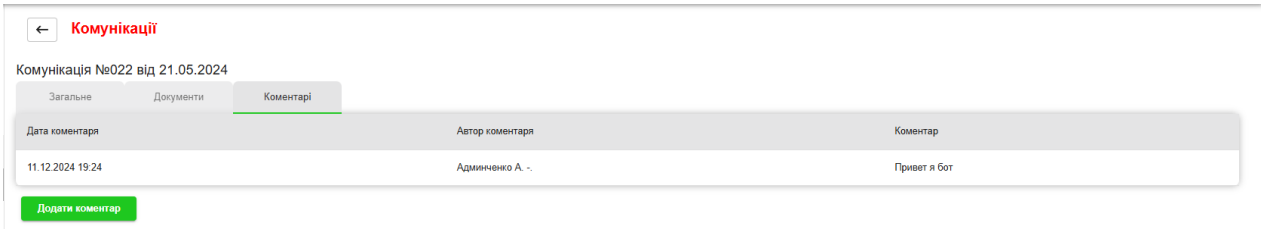


Рисунок 3.11. – Вкладка «Коментарі»

Розглянемо також додатковий функціонал CRM-модуля — шаблони комунікацій. Цей модуль створено для того, щоб спростити та пришвидшити процес створення нових комунікацій шляхом використання заздалегідь підготовлених шаблонів. Це особливо актуально для великих страхових компаній, де щоденно створюється велика кількість комунікацій, які мають схожі параметри.

На головній сторінці модуля шаблонів реалізовано інтерфейс, подібний до основної сторінки з комунікаціями. Тут є компактний фільтр для пошуку шаблонів та таблиця з результатами, що дозволяє швидко знайти потрібний шаблон і використовувати його для створення нової комунікації.

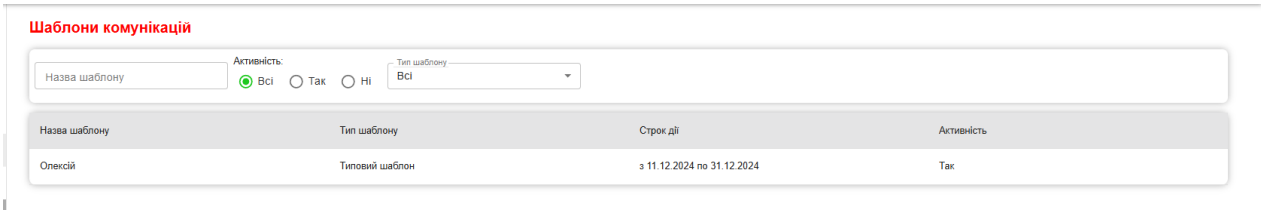


Рисунок 3.12. – Шаблони комунікацій

При створенні нового шаблону доступні два основні блоки.

Перший блок, розташований ліворуч, називається «Відомості про шаблон». У ньому відображаються основні поля, пов’язані із шаблоном, такі як назва, тип, статус тощо.

Другий блок є копією блоку, який використовується при створенні або редагуванні комунікацій. Він відображає інформацію із загальної вкладки, що

містить основні дані про комунікацію, зокрема «Відомості про комунікацію». Це дозволяє зручно налаштовувати шаблон із врахуванням усіх ключових параметрів, необхідних для майбутніх комунікацій.

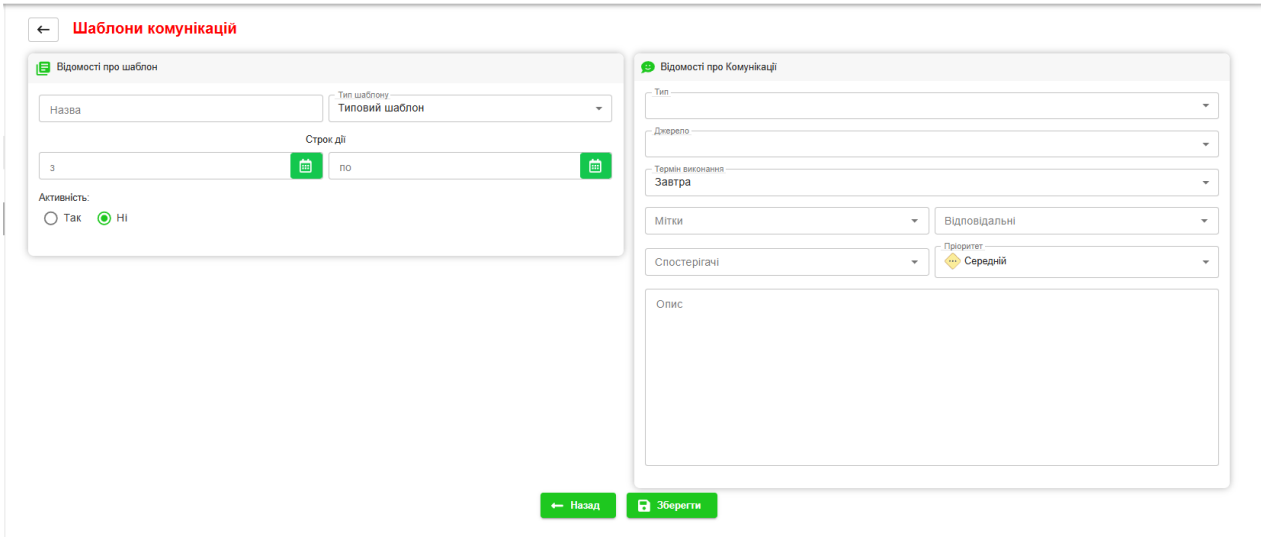


Рисунок 3.13. – Створення шаблонів комунікацій

Також є ще один допоміжний блок, це «Процедури», в якій є одна процедура по імпорту комунікацій із файлу, потрібно це також для прискорення ведення роботи із за великої кількості комунікацій та отриманням інформації в електронному вигляді яку потрібно переносити до системи ведення.

Рисунок 3.14. – Процедури комунікацій

Рисунок 3.15. – Імпорт комунікацій

3.3. Тестування

Процес тестування інформаційної системи є критичним етапом її розробки, що спрямований на оцінку функціональності, якості та надійності системи перед її введенням у експлуатацію. Основна мета тестування —

виявити помилки, дефекти та недоліки, щоб виправити їх до моменту, коли система стане доступною для кінцевих користувачів.

До ключових видів тестування інформаційних систем відносяться:

1. Функціональне тестування: перевірка відповідності роботи системи вимогам, визначеним у технічній документації. Це дозволяє впевнитися, що система виконує свої основні функції.

2. Навантажувальне тестування: оцінка продуктивності системи за умов, що імітують реальні навантаження. Це дозволяє виявити проблеми з продуктивністю, масштабованістю та стійкістю до високих обсягів даних.

3. Тестування безпеки: аналіз системи на вразливості, а також перевірка її здатності захищати конфіденційну інформацію та протистояти несанкціонованому доступу.

4. Тестування інтерфейсу користувача: оцінка зручності використання та ергономіки інтерфейсу. Це тестування спрямоване на забезпечення комфортної взаємодії користувачів із системою.

5. Тестування на відновлення: перевірка здатності системи до коректного відновлення після збоїв або аварійних ситуацій.

6. Інтеграційне тестування: перевірка взаємодії між різними компонентами системи, щоб виявити можливі проблеми у їхній спільній роботі.

7. Тестування на прийняття: оцінка системи замовником або кінцевими користувачами для перевірки відповідності вимогам і очікуванням.

8. Автоматизоване тестування: застосування спеціалізованих інструментів для автоматичного виконання тестів, що підвищує ефективність і швидкість процесу.

9. Регресійне тестування: перевірка, чи не спричинили зміни або виправлення помилок негативного впливу на раніше протестовану функціональність системи.

10. Тестування відповідності: забезпечення відповідності стандартам, нормативним вимогам і внутрішнім бізнес-правилам.

Для успішного тестування важливо розробити чіткий план, підготувати тестові сценарії, створити набір тестових даних та налаштувати середовище для проведення тестів. Аналіз отриманих результатів дозволяє вчасно виявляти та усувати виявлені проблеми.

Ретельне виконання всіх зазначених типів тестування гарантує стабільність, надійність та ефективність роботи системи після її впровадження.

Основні критерії успішного тестування:

1. Чіткі вимоги: наявність ясної та недвозначної документації щодо функціоналу системи.
2. Повне покриття: охоплення тестуванням усіх основних функцій і компонентів.
3. Реалістичні тестові дані: використання даних, що імітують реальні умови роботи системи.
4. Автоматизація: застосування скриптів для автоматичного виконання тестів, що пришвидшує процес.
5. Робота з користувачами: залучення кінцевих користувачів для отримання зворотного зв'язку.
6. Виправлення помилок: своєчасне усунення виявлених проблем і перевірка результатів.
7. Документування: створення детальної звітності про тестування.
8. Повторне тестування: перевірка системи після внесення змін для підтвердження їх ефективності.

Інструменти тестування які використовуется це Postman та Junit.

Postman — це інструмент для розробки та тестування API, який дозволяє створювати HTTP-запити, автоматизувати їх виконання та перевіряти результати. З його допомогою можна перевіряти статус-коди, заголовки, вміст відповіді, а також симулювати різні сценарії роботи API. Postman підтримує збереження тестових сценаріїв, автоматизацію та можливість адаптації до різних середовищ, що робить процес тестування гнучким і ефективним.

JUnit — популярна бібліотека для тестування Java-додатків. Вона забезпечує автоматизацію тестів, допомагає створювати структуровані тестові набори та підтримує інтеграцію з багатьма IDE. JUnit дозволяє проводити регресійні тести, перевіряти функціональність методів, організовувати тести за категоріями та автоматизувати процес перевірки, що суттєво підвищує якість розробки.

ВИСНОВКИ

У результаті виконання дипломної роботи, присвяченої розробці платформи для управління взаємодією з клієнтами (CRM) для страхової компанії, було розроблено ефективне та інноваційне рішення, яке відповідає актуальним вимогам галузі та дозволяє суттєво покращити процеси взаємодії з клієнтами. Створена CRM-система стала важливим інструментом для автоматизації операцій, полегшення управлінських процесів і підвищення якості обслуговування клієнтів у страховій компанії.

Розроблена система дозволяє страховій компанії ефективно управляти інформацією про клієнтів, відслідковувати етапи взаємодії з ними, автоматизувати продажі страхових продуктів і здійснювати ретельний аналіз даних для прогнозування поведінки клієнтів. Крім того, реалізація інтеграцій з різними комунікаційними каналами, такими як телефонія, електронна пошта та месенджери, дозволяє ефективно організувати обробку запитів клієнтів та забезпечити високий рівень їх задоволення.

Завдяки використанню сучасних технологій і підходів у розробці, система характеризується високою продуктивністю, масштабованістю та надійністю. Важливим аспектом було забезпечення безпеки обробки даних клієнтів відповідно до вимог чинного законодавства, що є надзвичайно важливим у сфері страхування, де конфіденційність інформації є пріоритетом.

Комплексне тестування системи підтвердило її стабільну роботу в різних умовах і дозволило виявити та усунути можливі дефекти до впровадження, що забезпечує надійність і безпеку продукту в реальному використанні. Крім того, система має інтуїтивно зрозумілий інтерфейс, що сприяє швидкому освоєнню її функцій співробітниками компанії без необхідності в додаткових навчаннях.

Впровадження цієї CRM-системи дозволить суттєво підвищити ефективність внутрішніх процесів компанії, зменшити час обробки запитів, покращити взаємодію між різними підрозділами та оптимізувати процеси

продажів і обслуговування клієнтів. Автоматизація рутинних завдань дозволить співробітникам фокусуватися на більш важливих аспектах роботи, таких як побудова лояльних відносин із клієнтами.

Розробка платформи CRM для страхової компанії є важливим кроком до забезпечення конкурентоспроможності та стійкості компанії на ринку. Вона дозволяє компанії краще розуміти потреби своїх клієнтів, вдосконалювати свої послуги та адаптуватися до змінюваних умов ринку. Система також дає змогу компанії знижувати витрати на обслуговування клієнтів, зменшувати ймовірність помилок та підвищувати продуктивність роботи співробітників.

Загалом, результати цієї роботи показують, що створення та впровадження CRM-платформи є важливим етапом у розвитку бізнесу страхової компанії, що сприяє підвищенню її ефективності, задоволеності клієнтів і довгостроковому успіху на ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гриньова, В.М., Сидоренко, В.В. *Управління страховою діяльністю*. — Київ: Центр навчальної літератури, 2020. — С. 45–67.
2. Базилевич, В.Д., Базилевич, К.С. *Страхування: підручник*. — 4-те вид. — Київ: Знання, 2021. — С. 112–134.
3. Петров, А.В. *CRM-системи в сучасному бізнесі: тенденції та перспективи розвитку*. — Київ: Наукова думка, 2022. — С. 45–67.
4. Гончаренко, І.М. *Інформаційні технології в страхових компаніях: впровадження CRM-систем*. — Львів: Видавництво ЛНУ, 2021. — С. 102–118.
5. Іванова, Т. О., та Ковальчук, І. В. *Інформаційні технології в управлінні страховими компаніями: впровадження та ефективність CRM-систем*. — Київ: Видавництво "Академперіодика", 2022. — С. 45–67.
6. JavaRush. *Основи сервлетів. Лекція 1* [Електронний ресурс]. — Режим доступу: https://javarush.com/ua/quests/lectures/ua.questservlets.level15.lecture01
7. LinkedIn. *What factors should you consider when choosing AIX?* [Електронний ресурс]. — Режим доступу: https://www.linkedin.com/advice/1/what-factors-should-you-consider-when-choosing-aixre.
8. StarkCloud. *Agile methods in software development.* [Електронний ресурс]. — Режим доступу: https://www.starkcloud.com/en/starkcloud-blog/agile-methods-in-software-development.
9. Amazon Web Services (AWS). *What is Java?* [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/what-is/java/.
10. Spring. *Spring Boot Project.* [Електронний ресурс]. — Режим доступу: https://spring.io/projects/spring-boot.
11. Microsoft. *SQL Server Management Studio (SSMS).* [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/ru-ru/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16l.

12. Git. *Git Documentation.* [Електронний ресурс]. — Режим доступу: https://git-scm.com/docs.

13. GeeksforGeeks. *What are Microservices?* [Електронний ресурс]. — Режим доступу: https://www.geeksforgeeks.org/microservices/#1-what-are-microservices.

14. AltexSoft. *REST API Design.* [Електронний ресурс]. — Режим доступу: https://www.altexsoft.com/blog/rest-api-design/.

15. "Розробка баз даних: ключові етапи та вимоги." [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/ru-ru/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16l

16. Elmasri, R., & Navathe, S. B. (2015). Fundamentals of Database Systems (7th ed.). Pearson Education. p. 82-115.

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

ДОДАТКИ

Дадоток А. Етап фізічного проектування

Таблиця	Атрибут	Тип	Довжина
communication	id	numeric	19
	number	nvanchar	16
	description	nvanchar	1000
	channel	varchar	32
	create_time	datetime	
	due_date	datetime	
	id_type	numeric	19
	id_status	numeric	19
	id_creator	numeric	19
	priority	nvanchar	40
communication_card_comment	id	numeric	19
	id_communication	numeric	19
	id_cagent	numeric	19
	comment	varchar	1000
	date	datetime	
communication_change_history	id	numeric	19
	id_author	numeric	19
	id_communication	numeric	19
	date_time	datetime	
	comment	nvanchar	max
	type	varchar	256
	id_status	numeric	19
	operation_date	datetime	
communication_link	id	numeric	19
	id_link	numeric	19

	type	varchar	32
	id_communication	numeric	19
	searchText	nvanchar	800
communication_mark	id	numeric	19
	id_communication	numeric	19
	id_mark	numeric	19
communication_observers	id	numeric	19
	id_communication	numeric	19
	id_observer	numeric	19
communication_responsible	id	numeric	19
	id_communication	numeric	19
	id_responsible	numeric	19
cagent	id	numeric	19
	en_personType	varchar	255
	person_name	varchar	255
	person_surname	varchar	255
	person_patronymic	varchar	10
	sex	varchar	255
	person_birthDate	datetime	
	person_inn	varchar	255
	person_profession	varchar	255
custom_dict_item	id	numeric	19
	name	varchar	255
	active dateBegin	tinyint datetime	
	dateEnd	datetime	
global_document	id	numeric	19

	entityType	varchar	100
	id_entity	numeric	19
	id_fileInfo	numeric	19
	name	nvanchar	255
	receiveTime	smalldatetime	
	docType	varchar	100
	status	nvanchar	255
	code	nvanchar	255
	fileId	varchar	255
	fileSize	int	
	fileName	varchar	255
	number	varchar	255
	documentId	varchar	255
	data_fields	nvanchar	max
	state	varchar	50
	signSystem	varchar	255
	signFileId	varchar	255
	id_parent	numeric	19
	file_ids_json	nvanchar	max

Дадоток Б. Код реалізації проекту

CommunicationCardPage.jsx

```
import Alert from 'js/components/Alert'
import BackArrow from 'js/icons/BackArrow'
import CircularProgress from 'js/components/CircularProgress'
import currentContext from 'js/stores/currentContext'
import Dialog from 'js/components/Dialog'
import DialogContent from 'js/components/DialogContent'
import DialogTitle from 'js/components/DialogTitle'
import DialogActions from 'js/components/DialogActions'
import Edit from 'js/icons/Edit'
import FormPanel from 'js/components/FormPanel'
import Grid from 'js/components/Grid'
import MenuItem from 'js/components/MenuItem'
import React, {createRef} from 'react'
```

```
import Save from 'js/icons/Save'
import Select from 'js/components/Select/Select'
import Typography from 'js/components/Typography'
import {withStyles} from '@material-ui/core/styles'
import Exchange from 'js/icons/Exchange'
import Close from 'js/icons/CloseIcon'
import Copy from 'js/icons/Copy'
import {INBOUND_CALL} from 'js/pages/communication/constants/communicationChannel'
import {
    EMPTY_CHANNEL_FIELD,
    EMPTY_CLIENT_NAME_FIELD,
    EMPTY_CLIENT_SEX_FIELD,
    EMPTY_CLIENT_SURNAME_FIELD,
    EMPTY_DESCRIPTION_FIELD,
    EMPTY_MARKS_FIELD,
    EMPTY_RESPONSIBLE_FIELD,
    EMPTY_STATUS_FIELD,
    EMPTY_TYPE_FIELD,
    INVALID_CLIENT_EMAIL_FIELD,
    INVALID_CLIENT_PHONE_NUMBER_FIELD,
    IS_DUPLICATE_COMMUNICATION,
} from 'js/pages/communication/constants/invalidErorrKeys'
import IconButton from 'js/components/IconButton'
import LinkSearchBox from
'js/pages/communication/components/link/LinkSearchBox/LinkSearchBox'
import GlobalDocumentsPanel from 'js/pages/documents/components/GlobalDocumentsPanel';
import LinearProgress from 'js/components/LinearProgress';
import {
    addNewCagentLink,
    addLink,
    createCommunication, deleteById,
    fetchSetup,
    getDocumentProps,
    transferCommunication,
    updateCommunication,
    validateBeforeSave,
} from 'js/pages/communication/actions/communicationCard'
import {
    CONTACTS_DATA_PANEL,
    PERSON_DATA_PANEL
} from 'js/pages/communication/constants/clientPanelsDataIds'
import {EXTRA_SMALL} from 'js/const/paginationRowsPerPageSizes'
import * as linkTypeKeys from 'js/pages/communication/constants/linkTypeKeys'
import {
    COMMUNICATION,
    CAGENT,
} from "js/pages/communication/constants/linkTypeKeys";
import Tabs from "js/components/Tabs";
import TabsContent from "js/components/TabsContent";
import Tab from "js/components/Tab/Tab";
import CommunicationComments from
"js/pages/communication/containers/CommunicationComments";
```

```
import CommunicationLinkPanel from
"js/pages/communication/components/link/CommunicationLinkPanel";
import CommunicationChangeHistoryPanel
   from
"js/pages/communication/components/communicationChangeHistoryPanel/CommunicationChan
geHistoryPanel";
import NonRedDelete from "js/icons/NonRedDelete";
import CommunicationPanel from
"js/pages/communication/components/communicationPanel/CommunicationPanel";
import CommunicationClientPanel from
"js/pages/communication/components/client/CommunicationClientPanel";
import communicationsContext from "../../communicationList/store/communicationsContext";
import CommunicationTransferDialog from
"../../communicationList/components/CommunicationTransferDialog";
import {MEDIUM} from "../constants/communicationPriority";
import Button from "../../../components/Button";
import DialogRow from "../../../components/DialogRow";
import CommunicationAddCommentDialog from "./CommunicationAddCommentDialog";
import {addComment} from "../actions/communicationCard";
import {formatDate} from 'js/util/DateUtil'
import BoSnackBar from "js/components/BoSnackBar";
import {defineDynamicDueDate} from
"../../communicationTemplate/utils/communicationsTemplate";

const styles = theme => ({
   actionsContainer: {
      marginTop: '-25px',
      paddingBottom: '10px',
      marginBottom: '10px',
      '& div': {
         marginTop: '3px'
      }
   },
   communicationsContainer: {
      width: '100%',
      boxSizing: 'border-box',
      marginTop: '30px',
   },
   communicationsCagentBlock: {
      width: '100%',
   },
   closeButton: {
      position: 'absolute',
      right: '8px',
      top: '8px',
      color: theme.palette.grey[500],
      width: '50px',
      height: '50px',
   },
   dialogContent: {
      paddingTop: `${theme.spacing()}px`
   }
});
```

```
const TRUE = 'true';
const TAB_INDEX_FOR_DOCUMENTS = 1;

class CommunicationCardPage extends React.Component {

    constructor(props) {
        super(props);
        this.snackbarRef = createRef();
        this.state = {
            addCagentLink: false,
            accordingErrorKeys: [],
            cagentLink: {
                birthDate: null,
                country: '',
                email: '',
                house: '',
                id: null,
                name: '',
                patronymic: '',
                phone: '',
                sex: '',
                sexErrorText: '',
                street: '',
                surname: '',
                town: '',
            },
            clientBirthDate: null,
            clientCountry: '',
            clientEmail: '',
            clientHouse: '',
            clientId: null,
            clientName: '',
            clientPatronymic: '',
            clientPhone: '',
            clientSex: '',
            clientSexErrorText: '',
            clientStreet: '',
            clientSurname: '',
            clientTown: '',
            changeHistory: [],
            changeHistoryPaginationPage: 0,
            changeHistoryPaginationPerPage: EXTRA_SMALL,
            communicationChannel: '',
            communicationComments: [],
            communicationCreateDate: null,
            communicationDescription: '',
            communicationDueDate: null,
            communicationId: '',
            communicationLinks: [],
            communicationMarks: [],
            communicationNumber: '',
            communicationObservers: [],
```

```
        communicationPhoneNumber: '',
        communicationResponsibles: [],
        communicationResponsibleDetails: [],
        communicationStatus: {},
        communicationType: {},
        communicationPriority: MEDIUM,
        documentProps: '',
        expandedClientPanel: PERSON_DATA_PANEL,
        expandedCagentLinkPanel: PERSON_DATA_PANEL,
        invalidEmail: false,
        invalidPhoneNumber: false,
        isEditing: false,
        isFailed: false,
        isFailedAddLink: false,
        isFailedSave: false,
        isFetching: false,
        isLinkSaving: false,
        isNotChosen: false,
        isOneClickButtonSave: false,
        isOpenDeleteDialog: false,
        isOpenDeleteLinkBoxDialog: false,
        isOpenSearchLinkBoxDialog: false,
        isOpenTransferDialog: false,
        isSuccessSave: false,
        selectedCommunicationLinks: [],
        selectedDeleteLinkItem: null,
        selectedLinkItem: null,
        selectedLinkType: COMMUNICATION,
        showAddCommentDialog: false,
        stateErrorKeys: [],
        tabIndex: 0,
    }
}

openSnackBar = (message, isError, isInfo) => {
    this.snackbarRef.current.open(message, isError, isInfo);
};

onChangeTab = (event, newValue) => {
    if (newValue === TAB_INDEX_FOR_DOCUMENTS) {
        this.setupDocuments();
    }
    this.setState({tabIndex: newValue})
};

componentDidMount() {
    this.setup(false)
}

setup(isCopyThis) {
    this.setState({
        isFailed: false,
        isFetching: true,
```

```
    });
    const {
        cagentId,
        callId,
        id,
        isEditing,
        link,
        responsibleId,
    } = this.props;
    fetchSetup({
        cagentId,
        callId,
        id,
        responsibleId,
    }).then(({
                cagent,
                call,
                communication,
                communicationType,
                responsible,
            }) => {
             this.setState({
                ...this.convertData({
                    call,
                    client: cagent,
                    communication,
                    link: link,
                    responsible,
                    type: communicationType,
                    isCopyThis,
                }),
                isEditing: isEditing === TRUE || !id || isCopyThis,
                isFetching: false,
             })
        })
        .catch(() => {
            this.setState({
                isFailed: true,
            })
        });
}

setupDocuments() {
    this.setState({
        isFetching: true
    });
    getDocumentProps({
        id: this.state.communicationId,
    }).then(documentProps => {
        this.setState({
            documentProps: documentProps,
            isFetching: false
        })
```

```
        }).catch(() => {
            this.setState({
                isFailed: true,
                isFetching: false,
            })
        });
    }

    onApplyUpdate = ({action, body}) => {
        const id = this.state.communicationId;
        action({id, body})
            .then((comments) =>
                this.setState({
                    communicationComments: comments,
                    isFetching: false,
                })
            ).catch(() => {
            this.setState({
                isFailed: true,
                isFetching: false,
            });
        });
    };

    handleAddCommunicationLink = (selectedLinkItem) => {
        const {
            addCagentLink,
            communicationId,
            cagentLink,
            isEditing,
            selectedCommunicationLinks,
        } = this.state;
        let newLinksArray = selectedCommunicationLinks;
        const containsLink = selectedCommunicationLinks.find(item => selectedLinkItem.id ===
item.link.id);
        const newErrorKeys = []
        const newAccordingErrorKeys = []
        const formatLinkNumber = this.formatLinkNumber(selectedLinkItem, cagentLink);

        if (addCagentLink){
            this.validateCagentFields(cagentLink, false, newAccordingErrorKeys, newErrorKeys);
            if (!newErrorKeys.length){
                this.setState({
                    isLinkSaving: true,
                });
            }
        }

        if (!newErrorKeys.length){
            if (!containsLink) {
                newLinksArray = newLinksArray.concat({
                    link: {
                        id: selectedLinkItem.id,
```

```
                number: formatLinkNumber,
                status: selectedLinkItem.status,
            },
            type: selectedLinkItem.type,
        })
    }
    if (isEditing) {
        if (selectedLinkItem.id === -1){
            addNewCagentLink(cagentLink)
                .then((cagentId) => {
                    newLinksArray.find(item => -1 === item.link.id).link.id = cagentId
                });
        }

        this.setState({
            isOpenSearchLinkBoxDialog: false,
            selectedCommunicationLinks: newLinksArray,
            selectedLinkItem: null,
            isLinkSaving: false,
        });
    } else if (!containsLink) {
        addLink(communicationId, selectedLinkItem, cagentLink, formatLinkNumber)
            .then((links) => {
                this.setState({
                    communicationLinks: links,
                    isOpenSearchLinkBoxDialog: false,
                    selectedCommunicationLinks: links,
                    selectedLinkItem: null,
                    isLinkSaving: false,
                });
            }).catch(() => {
            this.setState({
                isFailedAddLink: true,
            });
        })
    } else {
        this.setState({
            isOpenSearchLinkBoxDialog: false,
            isLinkSaving: false,
        });
    }
    this.clearCagentLinkFields();
  }
};

formatLinkNumber(selectedLinkItem, cagentLink) {
  if (selectedLinkItem.type === CAGENT) {
    if (selectedLinkItem.id === -1) {
      return `${cagentLink.surname} ${cagentLink.name} ${cagentLink.patronymic}`;
    } else {
      return selectedLinkItem.name;
    }
  } else {
```

```
            return selectedLinkItem.number;
        }
    }

    validateClientPhone(phone) {
        const regex = /^380(\d{9})$/g;
        return !regex.test(phone);
    }

    validateClientEmail(email) {
        const re = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        return !re.test(email);
    }

    validateForHighlightFields() {
        const {
            communicationChannel,
            communicationDescription,
            communicationId,
            communicationMarks,
            communicationResponsibles,
            communicationStatus,
            communicationType,
        } = this.state;
        const newErrorKeys = [];
        //Если у нас уже сущетсвует communicationId то значим мы редактируем
коммуникацию, в этом случае валидириуем отсутствие статуса
        if (!communicationStatus && communicationId) {
            newErrorKeys.push(EMPTY_STATUS_FIELD)
        }
        if (!communicationType) {
            newErrorKeys.push(EMPTY_TYPE_FIELD)
        }
        if (!communicationMarks || communicationMarks.length === 0) {
            newErrorKeys.push(EMPTY_MARKS_FIELD)
        }
        if (!communicationResponsibles || communicationResponsibles.length === 0) {
            newErrorKeys.push(EMPTY_RESPONSIBLE_FIELD)
        }
        if (!communicationChannel) {
            newErrorKeys.push(EMPTY_CHANNEL_FIELD)
        }
        if (!communicationDescription) {
            newErrorKeys.push(EMPTY_DESCRIPTION_FIELD)
        }
        return newErrorKeys;
    }

    handleSaveCommunicationWithAddingComment(comment) {
        const {
            clientBirthDate,
            clientCountry,
            clientEmail,
```

```
    clientHouse,
    clientId,
    clientName,
    clientPatronymic,
    clientPhone,
    clientSex,
    clientStreet,
    clientSurname,
    clientTown,
    communicationChannel,
    communicationDescription,
    communicationDueDate,
    communicationId,
    communicationMarks,
    communicationObservers,
    communicationPhoneNumber,
    communicationResponsibles,
    communicationStatus,
    communicationType,
    communicationPriority,
    responsibleCagents,
    selectedCommunicationLinks,
} = this.state;
this.setState({
    isFetching: true,
});
const body = {
    clientBirthDate,
    clientCountry,
    clientEmail,
    clientHouse,
    clientId,
    clientName,
    clientPatronymic,
    clientPhone,
    clientSex,
    clientStreet,
    clientSurname,
    clientTown,
    communicationChannel,
    communicationDescription,
    communicationDueDate,
    communicationId,
    communicationLinks: selectedCommunicationLinks,
    communicationPhoneNumber,
    communicationStatusId: (communicationStatus || {}).id,
    communicationTypeId: (communicationType || {}).id,
    communicationPriority,
    markIds: (communicationMarks || []).map(mark => mark.id),
    observerIds: (communicationObservers || []).map(obs => obs.id),
    responsibleIds: (communicationResponsibles || []).map(resp => resp.id),
};
updateCommunication(body)
```

```
        .then((communication) => {
          this.onApplyUpdate({ action: addComment, body: comment});
          this.setState({
              ...this.convertData({
                  communication,
                  responsibleCagents,
                  type: communicationType,
              }),
              isEditing: false,
              isFetching: false,
              isSuccessSave: true,
              showAddCommentDialog: false
          });
      })
      .catch(() => this.setState({
          isFailedSave: true,
          isFetching: false,
      }));
};

onSaveCommunication() {
    const {
        clientBirthDate,
        clientCountry,
        clientEmail,
        clientHouse,
        clientId,
        clientName,
        clientPatronymic,
        clientPhone,
        clientSex,
        clientStreet,
        clientSurname,
        clientTown,
        communicationChannel,
        communicationDescription,
        communicationDueDate,
        communicationId,
        communicationMarks,
        communicationObservers,
        communicationPhoneNumber,
        communicationResponsibles,
        communicationStatus,
        communicationType,
        communicationPriority,
        selectedCommunicationLinks,
        isOneClickButtonSave,
    } = this.state;
    const newErrorKeys = this.validateForHighlightFields();

    if (!newErrorKeys.length) {
        const body = {
            communicationId: communicationId,
```

```
            communicationLinks: selectedCommunicationLinks,
            communicationStatusId: (communicationStatus || {}).id,
            communicationTypeId: (communicationType || {}).id,
            markIds: (communicationMarks || []).map(mark => mark.id),
            responsibleIds: (communicationResponsibles || []).map(resp => resp.id),
          };

          validateBeforeSave(body).then((dto) => {
            if (dto.errorCode === IS_DUPLICATE_COMMUNICATION &&
!isOneClickButtonSave) {

this.openSnackBar(currentContext.provideMessage('communication.validation.duplicate'), true);
              this.setState({
                isOneClickButtonSave: true,
              });
            } else if (dto.errorMessage) {
              this.openSnackBar(dto.errorMessage, true);
            } else if (dto.showAddCommentDialog){
              this.setState({
                showAddCommentDialog: true
              })
            } else {
              this.setState({
                isFetching: true,
              });
              const body = {
                clientBirthDate,
                clientCountry,
                clientEmail,
                clientHouse,
                clientId,
                clientName,
                clientPatronymic,
                clientPhone,
                clientSex,
                clientStreet,
                clientSurname,
                clientTown,
                communicationChannel,
                communicationDescription,
                communicationDueDate,
                communicationId,
                communicationLinks: selectedCommunicationLinks,
                communicationPhoneNumber,
                communicationStatusId: (communicationStatus || {}).id,
                communicationTypeId: (communicationType || {}).id,
                communicationPriority,
                markIds: (communicationMarks || []).map(mark => mark.id),
                observerIds: (communicationObservers || []).map(obs => obs.id),
                responsibleIds: (communicationResponsibles || []).map(resp => resp.id),
              };

              (communicationId
```

```
                ? updateCommunication(body)
                : createCommunication(body))
                .then((communication) => {
                    this.setState({
                        ...this.convertData({
                            communication,
                            type: communicationType,
                        }),
                        isEditing: false,
                        isFetching: false,
                        isOneClickButtonSave: false,
                        isSuccessSave: true,
                    });

                    //если мы создаем карточку, то обновим веб-флоу.
                    if (!communicationId) {
                        this.props.onSubmitCreation(communication.id);
                    }
                })
                .catch(() => this.setState({
                    isFailedSave: true,
                    isFetching: false,
                }));
            }
        });
    }

    this.setState({
        stateErrorKeys: newErrorKeys,
    });
  }

  validateCagentFields = (cagentLink, isNotChosen, newAccordingErrorKeys, newErrorKeys)
=> {
    if (!cagentLink.name && !isNotChosen) {
        newAccordingErrorKeys.push(PERSON_DATA_PANEL)
        newErrorKeys.push(EMPTY_CLIENT_NAME_FIELD)
    }
    if (!cagentLink.surname && !isNotChosen) {
        newAccordingErrorKeys.push(PERSON_DATA_PANEL)
        newErrorKeys.push(EMPTY_CLIENT_SURNAME_FIELD)
    }
    if (!cagentLink.sex && !isNotChosen) {
        newAccordingErrorKeys.push(PERSON_DATA_PANEL);
        newErrorKeys.push(EMPTY_CLIENT_SEX_FIELD)
    }
    if (cagentLink.email && this.validateClientEmail(cagentLink.email)) {
        newAccordingErrorKeys.push(CONTACTS_DATA_PANEL);
        newErrorKeys.push(INVALID_CLIENT_EMAIL_FIELD)
    }
    if (cagentLink.phone && this.validateClientPhone(cagentLink.phone)) {
        newAccordingErrorKeys.push(CONTACTS_DATA_PANEL);
        newErrorKeys.push(INVALID_CLIENT_PHONE_NUMBER_FIELD)
```

```
        }
        this.setState({
            accordingErrorKeys: newAccordingErrorKeys,
            cagentLink: {
                ...this.state.cagentLink,
                sexErrorText: newErrorKeys.includes(EMPTY_CLIENT_SEX_FIELD) ?
currentContext.provideMessage('communication.sex.validation.error') : ''
            },
            invalidPhoneNumber:
newErrorKeys.includes(INVALID_CLIENT_PHONE_NUMBER_FIELD),
            invalidEmail: newErrorKeys.includes(INVALID_CLIENT_EMAIL_FIELD),
            stateErrorKeys: newErrorKeys,
        });
    };

    handleDelete = () => {
        deleteById(this.state.communicationId);
        this.props.onReturnBack();
    };

    convertData({
                    call,
                    client,
                    communication,
                    link,
                    responsible,
                    type,
                    isCopyThis,
                }) {
        let {
            changeHistory,
            channel: communicationChannel,
            client: communicationClient,
            comments: communicationComments,
            creationDate: communicationCreateDate,
            description: communicationDescription,
            dueDate: communicationDueDate,
            id: communicationId,
            links: communicationLinks,
            marks: communicationMarks,
            number: communicationNumber,
            observers: communicationObservers,
            phoneNumber: communicationPhoneNumber,
            responsibles: communicationResponsibles,
            responsibleDetails: communicationResponsibleDetails,
            status: communicationStatus,
            type: communicationType,
            priority: communicationPriority,
        } = communication || {};

        if (client && !communication) {
            communicationClient = client;
        }
```

```
    if (responsible && !communication) {
      communicationResponsibles = responsible.list;
    }
    //Назначим тип "Особиста задача" при создании, то есть когда у нас нету
айдиКоммуникации
    if (type && !communicationId) {
      communicationType = type;
    }

    if (call && !communication) {
      const {
        subscriber,
        subscriberNumber,
      } = call
      communicationPhoneNumber = subscriberNumber
      communicationChannel = INBOUND_CALL
      if (subscriber) {
        communicationClient = client
      } else {
        communicationClient = {
          ...communicationClient,
          contacts: {
            phone: subscriberNumber,
          },
        }
      }
    }

    let {
      id: clientId,
      person,
      contacts,
      address,
    } = communicationClient || {};

    let {
      birthDate: clientBirthDate,
      name: clientName,
      patronymic: clientPatronymic,
      sex: clientSex,
      surname: clientSurname,
    } = person || {};
    let {
      email: clientEmail,
      phone: clientPhone,
    } = contacts || {};
    let {
      country: clientCountry,
      town: clientTown,
      street: clientStreet,
      house: clientHouse,
    } = address || {};
```

```
    if (link) {
      let newLinksArray = communicationLinks ? [...communicationLinks] : [];
      newLinksArray = newLinksArray.concat(link);
      communicationLinks = newLinksArray;
    }

    return {
      changeHistory: changeHistory || [],
      clientBirthDate,
      clientCountry: clientCountry || '',
      clientEmail: clientEmail || '',
      clientHouse: clientHouse || '',
      clientId,
      clientName: clientName || '',
      clientPatronymic: clientPatronymic || '',
      clientPhone: clientPhone || '',
      clientSex: clientSex || '',
      clientStreet: clientStreet || '',
      clientSurname: clientSurname || '',
      clientTown: clientTown || '',
      communication,
      communicationChannel: communicationChannel || INBOUND_CALL,
      communicationComments: communicationComments,
      communicationCreateDate: isCopyThis ? null : communicationCreateDate,
      communicationDescription,
      communicationDueDate: isCopyThis ? null : communicationDueDate,
      communicationId: isCopyThis ? '' : communicationId,
      communicationLinks: communicationLinks || [],
      communicationNumber: isCopyThis ? '' : communicationNumber,
      communicationMarks,
      communicationObservers,
      communicationPhoneNumber,
      communicationResponsibles,
      communicationResponsibleDetails,
      communicationStatus,
      communicationType,
      communicationPriority: communicationPriority || MEDIUM,
      selectedCommunicationLinks: communicationLinks || [],
    }
  }

  handleChangeObserverPersonsInCommunicationPanel = (cagents) => {
    this.setState({
      communicationObservers: cagents,
    });
  };

  handleChangeResponsiblePersonsInCommunicationPanel = (cagents) => {
    this.setState({
      communicationResponsibles: cagents,
      communicationResponsibleDetails: cagents,
    });
  };
```

```
handleSelectCommunicationType = (target) => {
    this.setState({
        communicationType: target.value,
    });
};

handleSelectCommunicationPriority = (value) => {
    this.setState({
        communicationPriority: value,
    });
};

handleSelectCommunicationStatus = (target) => {
    this.setState({
        communicationStatus: target.value,
    })
};

handleSelectCommunicationChannel = (target) => {
    this.setState({
        communicationChannel: target.value,
    })
};

handleSelectCommunicationDueDate = (newDate) => {
    this.setState({
        communicationDueDate: newDate,
    })
};

handleSelectCommunicationDescription = (target) => {
    this.setState({
        communicationDescription: target.value,
    })
};

handleOpenSearchLinkBoxDialog = () => {
    this.setState({
        isOpenSearchLinkBoxDialog: true,
    })
};

handleOpenDeleteLinkBoxDialog = (item) => {
    this.setState({
        isOpenDeleteLinkBoxDialog: true,
        selectedDeleteLinkItem: item,
    })
};

handleChangeHistoryPaginationPage = (newPage) => {
    this.setState({
        changeHistoryPaginationPage: newPage,
```

```
    })
};

handleChangeHistoryPaginationRowsPerPage = (target) => {
    this.setState({
        changeHistoryPaginationPerPage: target.value,
    })
};

handleSelectCommunicationMarks = (marks) => {
    this.setState({
        communicationMarks: marks
    })
};

handleExitAddLinkDialog = () => {
    this.setState({
        isLinkSaving: false,
        selectedLinkItem: null,
    })
    this.clearCagentLinkFields();
};

handleChangeCagentLinkPanel = panel => (event, newExpanded) => {
    this.setState({
        expandedCagentLinkPanel: newExpanded ? panel : false,
    })
};

handleChangeCagentLinkBirthDate = (newDate) => {
    this.setState({
        cagentLink: {
            ...this.state.cagentLink,
            birthDate: newDate,
        }
    })
};

handleChangeCagentLinkCountry = (target) => {
    this.setState({
        cagentLink: {
            ...this.state.cagentLink,
            country: target.value,
        }
    })
};

handleChangeCagentLinkEmail = (target) => {
    const needUpdate = target.value.length < 50;
    this.setState({
        cagentLink: {
            ...this.state.cagentLink,
          email: needUpdate ? target.value : this.state.email,
```

```
        }
    })
};

handleChangeCagentLinkHouse = (target) => {
    this.setState({
        cagentLink: {
            ...this.state.cagentLink,
            house: target.value,
        }
    })
};

handleChangeCagentLinkName = (target) => {
    this.setState({
        cagentLink: {
            ...this.state.cagentLink,
            name: target.value,
        }
    })
};

handleChangeCagentLinkPatronymic = (target) => {
    this.setState({
        cagentLink: {
            ...this.state.cagentLink,
            patronymic: target.value,
        }
    })
};

handleChangeCagentLinkPhone = (target, phone) => {
    const needUpdate = target.value.length < 13 || phone.length > 13;
    this.setState({
        cagentLink: {
            ...this.state.cagentLink,
            phone: needUpdate ? target.value : phone,
        }
    })
};

handleChangeCagentLinkSex = (target) => {
    this.setState({
        cagentLink: {
            ...this.state.cagentLink,
            sex: target.value,
            sexErrorText: '',
        }
    })
};

handleChangeCagentLinkStreet = (target) => {
    this.setState({
```

```
        cagentLink: {
            ...this.state.cagentLink,
            street: target.value,
        }
    })
};


handleChangeCagentLinkSurname = (target) => {
    this.setState({
        cagentLink: {
            ...this.state.cagentLink,
            surname: target.value,
        }
    })
};

handleChangeCagentLinkTown = (target) => {
    this.setState({
        cagentLink: {
            ...this.state.cagentLink,
            town: target.value,
        }
    })
};

onFocusCagentLinkPhone = (target, clientPhone) => {
    if (!clientPhone) {
        this.setState({
            cagentLink: {
                ...this.state.cagentLink,
               phone: '380',
            }
        });
    }
};

clearCagentLinkFields = () => {
    this.setState({
        accordingErrorKeys: [],
        addCagentLink: false,
        cagentLink: {
            ...this.state.cagentLink,
            birthDate: null,
            country: '',
            email: '',
            house: '',
            id: null,
            name: '',
            patronymic: '',
            phone: '',
            sex: '',
            sexErrorText: '',
```

```
            street: '',
            surname: '',
            town: '',
        },
        invalidPhoneNumber: false,
        invalidEmail: false,
        stateErrorKeys: [],
    })
}

handleSaveTransferDialog = (responsibles, comment) => {
    transferCommunication({
        comment,
        communicationId: this.state.communicationId,
        responsibleIds: (responsibles || []).map(resp => resp.id),
    }).then((communication) => {
        this.setState({
            ...this.convertData({
                communication,
            }),
            isFetching: false,
            isOpenTransferDialog: false,
        })
    }).catch(() => this.setState({
        isFetching: false,
    }));
};

handleFillFromTemplate = (templateData) => {
    this.setState({
        communicationChannel: templateData.channel,
        communicationDescription: templateData.description,
        communicationDueDate: defineDynamicDueDate(templateData),
        communicationMarks: templateData.marks,
        communicationObservers: templateData.observers,
        communicationResponsibles: templateData.responsibles,
        communicationResponsibleDetails: templateData.responsibles,
        communicationType: templateData.type,
        communicationPriority: templateData.priority,
    });
}

render() {
    const {classes, updateWebFlow} = this.props;
    const {
        accordingErrorKeys,
        addCagentLink,
        cagentLink,
        changeHistory,
        changeHistoryPaginationPage,
        changeHistoryPaginationPerPage,
        clientId,
        communication,
```

```
            communicationChannel,
            communicationComments,
            communicationCreateDate,
            communicationDescription,
            communicationDueDate,
            communicationId,
            communicationLinks,
            communicationNumber,
            communicationMarks,
            communicationObservers,
            communicationResponsibles,
            communicationResponsibleDetails,
            communicationStatus,
            communicationType,
            communicationPriority,
            documentProps,
            expandedCagentLinkPanel,
            isEditing,
            isFailed,
            isFailedSave,
            isFetching,
            isLinkSaving,
            isOpenDeleteDialog,
            isOpenTransferDialog,
            isOpenSearchLinkBoxDialog,
            isSuccessSave,
            isOpenDeleteLinkBoxDialog,
            isFailedAddLink,
            invalidEmail,
            invalidPhoneNumber,
            selectedCommunicationLinks,
            selectedDeleteLinkItem,
            selectedLinkItem,
            selectedLinkType,
            stateErrorKeys,
        } = this.state;

        const {canAddLink, canCreateCopy, canCreateContract, canEdit, canDelete,
canCreateCagentLink} = communicationsContext;

        return (
            <div className={classes.communicationsContainer}>
                <div className={classes.actionsContainer}>
                    {invalidPhoneNumber && (
                        <Alert
                            id="invalidPhoneNumberAlert"

message={currentContext.provideMessage('common.phoneNumber.validation.error',
'380XXXXXXXXX')}
                            onClose={() => this.setState({
                                invalidPhoneNumber: false,
                            })}
                            variant="error"/>
```

```
            )}
            {invalidEmail && (
              <Alert
                  id="invalidEmailAlert"
                  message={currentContext.provideMessage('common.email.validation.error',
'example@gmail.com')}
                  onClose={() => this.setState({
                    invalidEmail: false,
                  })}
                  variant="error"/>
            )}
          </div>
          <div className={classes.communicationsCagentBlock}>
            {isFailed && (
              <Alert id="failedToLoadDataAlert"
message={currentContext.provideMessage('communication.alert.failedToLoadData')}
                  variant="error"/>
            )}
            {isFetching && (
              <LinearProgress/>
            )}
            {!isFailed && !isFetching && (
              <Grid container direction="column" spacing={4} >
                <Typography variant='h6'>
                  {communicationNumber && communicationCreateDate
                    ? currentContext.provideMessage('communication.view',
communicationNumber, formatDate(communicationCreateDate))
                    : currentContext.provideMessage('communication.new')}
                </Typography>
                <Tabs
                  id="communicationCardTabs"
                  indicatorColor="primary"
                  onChange={this.onChangeTab}
                  value={this.state.tabIndex}
                >
                  <Tab id="commonTab"
label={currentContext.provideMessage('common.common')}/>
                  {!isEditing && <Tab id="documentsTab"
label={currentContext.provideMessage('common.documents')}/>}

                  {!isEditing && <Tab id="commentsTab"
label={currentContext.provideMessage('common.comments')}/>}
                </Tabs>
                <TabsContent index={this.state.tabIndex}>
                  <div>
                    <Grid container item spacing={2}>
                      {/* Communication panel */}
                      <CommunicationPanel
                        communicationStatus={communicationStatus}
                        communicationMarks={communicationMarks}
                        communicationObservers={communicationObservers}
                        communicationResponsibles={communicationResponsibleDetails}
                        communicationChannel={communicationChannel}
```

```
                                communicationType={communicationType}
                                communicationPriority={communicationPriority}
                                communicationDescription={communicationDescription}
                                communicationDueDate={communicationDueDate}
                                communicationId={communicationId}
                                id="communicationPanel"
                                isEditing={isEditing}

onSelectCommunicationChannel={this.handleSelectCommunicationChannel}

onSelectCommunicationDescription={this.handleSelectCommunicationDescription}

onSelectCommunicationDueDate={this.handleSelectCommunicationDueDate}

onSelectCommunicationMarks={this.handleSelectCommunicationMarks}

onSelectCommunicationObserverPersons={this.handleChangeObserverPersonsInCommunication
Panel}

onSelectCommunicationResponsiblePersons={this.handleChangeResponsiblePersonsInCommuni
cationPanel}

onSelectCommunicationStatus={this.handleSelectCommunicationStatus}

onSelectCommunicationType={this.handleSelectCommunicationType}

onSelectCommunicationPriority={this.handleSelectCommunicationPriority}
                                openSnackBar={this.openSnackBar}
                                stateErrorKeys={stateErrorKeys}
                                useTemplate={true}
                                onFillFromTemplate={this.handleFillFromTemplate}
                            />
                        </Grid>
                        {/* Links panel */}
                        {(!!selectedCommunicationLinks.length || canAddLink ||
canCreateContract) &&
                        <CommunicationLinkPanel
                            clientId={clientId}
                            communicationId={communicationId}
                            communicationLinks={selectedCommunicationLinks}
                            isEditing={isEditing}
                            id="communicationLinksPanel"
                            onOpenDeleteLinkBoxDialog={this.handleOpenDeleteLinkBoxDialog}
                            onOpenSearchLinkBoxDialog={this.handleOpenSearchLinkBoxDialog}
                            updateWebFlow={updateWebFlow}
                        />}

                        {/* History panel */}
                        {!isEditing && communicationId &&
                        <CommunicationChangeHistoryPanel
                            changeHistory={changeHistory}
                            changeHistoryPaginationPage={changeHistoryPaginationPage}
                            changeHistoryPaginationPerPage={changeHistoryPaginationPerPage}
```

```
                        id="communicationChangeHistoryPanel"

onChangeRowsPerPage={this.handleChangeHistoryPaginationRowsPerPage}
                        onChangePage={this.handleChangeHistoryPaginationPage}
                    />}

                    {/* Actions and alerts */}
                    <Grid
                        alignItems="center"
                        container
                        direction="column"
                        item
                        justifyContent="center"
                        spacing={1}
                    >
                        {isEditing && isFailedSave && (
                            <Alert
                                id="failedToSaveAlert"

message={currentContext.provideMessage('communication.alert.failedToSave')}
                                onClose={() => this.setState({
                                    isFailedSave: false,
                                })}
                                timeout
                                variant="error"
                            />
                        )}
                        {isSuccessSave && (
                            <Alert
                                id="successSavedAlert"

message={currentContext.provideMessage('communication.alert.successSaved')}
                                onClose={() => this.setState({
                                    isSuccessSave: false,
                                })}
                                timeout
                                variant="success"
                            />
                        )}
                        <Grid
                            alignItems="center"
                            container
                            item spacing={1}
                            justifyContent="center"
                        >
                            {isEditing && communicationId && (
                                <Grid item>
                                    <Button
                                        icon={BackArrow}
                                        id="backToListButton"
                                        label={currentContext.provideMessage('common.back')}
                                        onClick={() => {
                                            this.setState({
```

```
                                    accordingErrorKeys: [],
                                    isEditing: false,
                                    ...this.convertData({
                                       communication,
                                       type: communicationType
                                    }),
                                    selectedCommunicationLinks: [...communicationLinks],
                                    stateErrorKeys: [],
                                 })
                              }}
                           />
                        </Grid>
                     )}
                     {canEdit &&
                     <Grid item>
                        <Button
                           icon={(() => {
                              if (isEditing) {
                                 return isFetching
                                    ? CircularProgress
                                    : Save
                              }
                              return Edit
                           })()}
                           id={isEditing ? "communicationSaveButton" :
"communicationEditButton"}
                           label={currentContext.provideMessage(isEditing ?
'common.save' : 'common.edit')}
                           onClick={() => isEditing
                              ? this.onSaveCommunication()
                              : this.setState({
                                 addCagentLink: false,
                                 isEditing: true,
                              })}
                        />
                     </Grid>
                     }
                     {communicationId && !isEditing && (
                        <Grid item>
                           <Button
                              icon={Exchange}
                              id="openTransferDialogButton"
                              label={currentContext.provideMessage('common.transfer')}
                              onClick={() => this.setState({
                                 isOpenTransferDialog: true,
                              })}
                           />
                        </Grid>
                     )}

                     {communicationId && !canCreateCopy && !isEditing && (
                        <Grid item>
                           <Button
```

```
                                    icon={Copy}
                                    id="copyButton"

label={currentContext.provideMessage('communication.createCopyItemButton')}
                                    onClick={() => this.setup(true)}
                                />
                            </Grid>
                        )}

                        {communicationId && canDelete && !isEditing && (
                            <Grid item>
                                <Button
                                    icon={NonRedDelete}
                                    id="openDeleteDialogButton"
                                    label={currentContext.provideMessage('common.delete')}
                                    onClick={() => this.setState({
                                        isOpenDeleteDialog: true,
                                    })}
                                />
                            </Grid>
                        )}
                    </Grid>

                </Grid>
            </div>

            {!isEditing && <FormPanel variant="simpleShadow">
                <div id="communicationCardPageDocuments"/>
                <GlobalDocumentsPanel {...documentProps}
theme={currentContext.theme}/>
            </FormPanel>}

            {!isEditing && <CommunicationComments
                comments={communicationComments}
                id="communicationComments"
                isEditing={isEditing}
                onApplyUpdate={this.onApplyUpdate}
            />}

        </TabsContent>
    </Grid>
)}
</div>

<Dialog
    fullWidth={true}
    id="linkSearchBoxDialog"
    maxWidth="lg"
    onExit={this.handleExitAddLinkDialog}
    open={isOpenSearchLinkBoxDialog}
>
    <DialogTitle
headerMessage={currentContext.provideMessage('common.addLink')}>
```

```
                    <IconButton
                      className={classes.closeButton}
                      disabled={isLinkSaving}
                      id="searchLinkBoxCloseButton"
                      inherit
                      onClick={() => {
                        this.clearCagentLinkFields();
                        this.setState({
                          isOpenSearchLinkBoxDialog: false
                        });
                      }}>
                       <Close/>
                    </IconButton>
                  </DialogTitle>
                  <DialogContent>
                    <div className={classes.dialogContent}>
                      <Grid container spacing={1}>
                        {isFailedAddLink && (
                          <Grid item container xs={12} spacing={6}>
                            <Alert
                              id="failedToAddLinkAlert"

message={currentContext.provideMessage('communication.alert.failedToAddLink')}
                              onClose={() => this.setState({
                                isFailedAddLink: false
                              })}
                              timeout
                              variant="error"/>
                          </Grid>)}

                        <Grid item container xs={4}>
                          <Select
                            id="linkTypeSelect"
                            onChange={({target}) => {
                              this.clearCagentLinkFields();
                              this.setState({
                              selectedLinkType: target.value,
                            })}}
                            value={selectedLinkType}
                          >
                            {Object.values(linkTypeKeys).map(key => (
                              <MenuItem
                                key={key}
                                value={key}
                              >

{currentContext.provideMessage(`communication.link.type.${key}`)}
                              </MenuItem>
                            ))}
                          </Select>
                        </Grid>
                        <Grid item container xs={8}>
                          <LinkSearchBox
```

```
                communicationNumber={communicationNumber}
                id="linkSearchBox"
                linkType={selectedLinkType}
                onSelect={(item) => {
                  this.setState({
                    selectedLinkItem: item,
                  })
                }}
                selectedItem={selectedLinkItem}
                addCagentLink={addCagentLink}
              />
                {addCagentLink ?
                <CommunicationClientPanel
                  accordingErrorKeys={accordingErrorKeys}
                  canCagentEdit={true}
                  canCagentSearchBox={false}
                  clientBirthDate={cagentLink.birthDate}
                  clientCountry={cagentLink.country}
                  clientEmail={cagentLink.email}
                  clientHouse={cagentLink.house}
                  clientName={cagentLink.name}
                  clientPatronymic={cagentLink.patronymic}
                  clientPhone={cagentLink.phone}
                  clientSex={cagentLink.sex}
                  clientSexErrorText={cagentLink.sexErrorText}
                  clientStreet={cagentLink.street}
                  clientSurname={cagentLink.surname}
                  clientTown={cagentLink.town}
                  expandedClientPanel={expandedCagentLinkPanel}
                  isEditing={true}
                  isNotChosen={false}
                  isViewCloseButton={true}
                  onChangeClientBirthDate={this.handleChangeCagentLinkBirthDate}
                  onChangeClientCountry={this.handleChangeCagentLinkCountry}
                  onChangeClientEmail={this.handleChangeCagentLinkEmail}
                  onChangeClientHouse={this.handleChangeCagentLinkHouse}
                  onChangeClientName={this.handleChangeCagentLinkName}
                  onChangeClientPanel={this.handleChangeCagentLinkPanel}

onChangeClientPatronymic={this.handleChangeCagentLinkPatronymic}
                  onChangeClientPhone={this.handleChangeCagentLinkPhone}
                  onChangeClientSex={this.handleChangeCagentLinkSex}
                  onChangeClientStreet={this.handleChangeCagentLinkStreet}
                  onChangeClientSurname={this.handleChangeCagentLinkSurname}
                  onChangeClientTown={this.handleChangeCagentLinkTown}
                  onClickCloseButton={this.handleExitAddLinkDialog}
                  onFocusClientPhone={this.onFocusCagentLinkPhone}
                  panelSize={12}
                  stateErrorKeys={stateErrorKeys}
                  updateWebFlow={updateWebFlow}
                /> : <div/>
                }
            </Grid>
```

```
                    </Grid>
                  </div>
                </DialogContent>
                <DialogActions>
                  {isLinkSaving ?
                    <Grid container justifyContent="center">
                      <CircularProgress/>
                    </Grid> :
                    <Grid container spacing={2} justifyContent="flex-end">
                      {selectedLinkType === CAGENT && !addCagentLink &&
canCreateCagentLink && (
                        <Grid item>
                          <Button
                            id="addCagentLink"
                            label={currentContext.provideMessage('common.cagent.new')}
                            onClick={() => {
                              this.setState({
                                addCagentLink: true,
                                selectedLinkItem: {
                                  id: -1,
                                  type: CAGENT
                                }
                              });
                            }}
                          />
                        </Grid>
                      )}
                      <Grid item style={{marginLeft: 'auto'}}>
                        <Button
                          disabled={!selectedLinkItem}
                          id="saveLinkButton"
                          label={currentContext.provideMessage('common.save')}
                          onClick={() => {
                            this.handleAddCommunicationLink(selectedLinkItem)
                          }}
                        />
                        <Button
                          id="cancelSavingLinkButton"
                          label={currentContext.provideMessage('common.cancel')}
                          onClick={() => {
                            this.setState({
                              addCagentLink: false,
                              isOpenSearchLinkBoxDialog: false,
                            })
                          }}
                        />
                      </Grid>
                    </Grid>
                  }
                </DialogActions>
              </Dialog>

              <Dialog
```

```
            fullWidth={true}
            id="linkDeleteDialog"
            maxWidth="sm"
            open={isOpenDeleteLinkBoxDialog}
          >
            <DialogTitle
              headerMessage={currentContext.provideMessage('common.deleteLink')}
              id ='deleteLinkBoxCloseButton'
              onClose={() => this.setState({isOpenDeleteLinkBoxDialog: false})}
            />
            <DialogContent>
              <DialogRow>
                <Typography id="deleteLinkQuestionTypography" variant="body1">
                  {currentContext.provideMessage('communication.deleteLink.question')}
                </Typography>
              </DialogRow>
            </DialogContent>
            <DialogActions>
              <Button
                id="confirmDeleteLinkButton"
                label={currentContext.provideMessage('common.yes')}
                onClick={() => {
                  this.setState({
                    isOpenDeleteLinkBoxDialog: false,
                    selectedCommunicationLinks: [...selectedCommunicationLinks].filter(link
=> link.link.id !== selectedDeleteLinkItem.link.id),
                    selectedDeleteLinkItem: null,
                  })
                }}/>
              <Button
                id="cancelDeleteLinkButton"
                label={currentContext.provideMessage('common.no')}
                onClick={() => {
                  this.setState({
                    isOpenDeleteLinkBoxDialog: false,
                    selectedDeleteLinkItem: null,
                  })
                }}/>
            </DialogActions>
          </Dialog>

          <Dialog
            fullWidth={true}
            id="communicationDeleteDialog"
            maxWidth="sm"
            open={isOpenDeleteDialog}
          >
            <DialogTitle
              headerMessage={currentContext.provideMessage('communication.delete')}
              id="deleteCommunicationCloseButton"
              onClose={() => this.setState({isOpenDeleteDialog: false })}
            />
            <DialogContent>
```

```
                <DialogRow>
                  <Typography id="deleteCommunicationQuestionTypography"
variant="body1">
                    {currentContext.provideMessage('communication.delete.question')}
                  </Typography>
                </DialogRow>
              </DialogContent>
              <DialogActions>
                <Button
                  id="cancelDeleteCommunicationButton"
                  label={currentContext.provideMessage('common.no')}
                  onClick={() => {
                    this.setState({
                      isOpenDeleteDialog: false,
                    })
                  }}
                />
                <Button
                  id="confirmDeleteCommunicationButton"
                  label={currentContext.provideMessage('common.yes')}
                  onClick={this.handleDelete}
                />
              </DialogActions>
            </Dialog>

            <CommunicationTransferDialog isOpenTransferDialog={isOpenTransferDialog}
                                         handleCloseTransferDialog={() => {
                                           this.setState({
                                             isOpenTransferDialog: false,
                                           })
                                         }}
                                         handleSaveTransferDialog={this.handleSaveTransferDialog}/>

            <CommunicationAddCommentDialog
              dialogTitle={currentContext.provideMessage('communication.addComment')}
              onApply={comment =>
this.handleSaveCommunicationWithAddingComment(comment)}
              onClose={() => this.setState({ showAddCommentDialog: false })}
              open={this.state.showAddCommentDialog}
            />

            <BoSnackBar ref={this.snackbarRef}/>
          </div>
        )
      }
}

export default withStyles(styles, {withTheme: true})(CommunicationCardPage)
```