НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “ Комп'ютерні системи та мережі ”

спеціальності 123 “ Комп’ютера інженерія ”

на тему: Система автоматичного сповіщення неактивних клієнтів

Виконав : студент __4__ курсу, групи _ІВ-92
(шифр групи)

_____Юзина Сергій Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент Трочун Є.В._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ст, викладач., Виноградов Ю. М._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“ Комп'ютерні системи та мережі ”

спеціальності 123 “ Комп’ютера інженерія ”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Юзини Сергія Сергійовича

1. Тема проєкту *Система автоматичного сповіщення неактивних клієнтів*
керівник проєкту Трочун Є.В., асистент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 31 травня 2023 року №2102-с

2. Термін здачі студентом закінченого проєкту *7 червня 2023 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз предметної області.

Розділ 2. Аналіз способів реалізації.

Розділ 3. Деталі розробки системи.

Розділ 4. Аналіз розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю.М.		

7. Дата видачі завдання «12» грудня 2022 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*12.12.2022-18.12.2022*	
2.	*Вивчення та аналіз завдання*	*19.12.2022-19.03.2023*	
3.	*Розробка архітектури та загальної структури системи*	*20.03.2023-02.04.2023*	
4.	*Розробка структур окремих підсистем*	*03.04.2023-15.04.2023*	
5.	*Програмна реалізація системи*	*24.04.2023-13.05.2023*	
6.	*Оформлення пояснювальної записки*	*15.05.2023-27.05.2023*	
7.	*Захист програмного продукту*	*06.06.2023*	
8.	*Передзахист*	*09.06.2023*	
9.	*Захист*	*19.06.2023*	

Студент-дипломник _____ Сергій ЮЗИНА
(підпис)

Керівник проєкту _____ Євген ТРОЧУН
(підпис)

АНОТАЦІЯ

У цій роботі було детально розглянуто системи сповіщення, способи доставки сповіщень, їхні переваги та недоліки. Розглянуто основні причини для сповіщення клієнтів. Розроблений простий інтерфейс для демонстрації роботи програми. Як результат було створено систему автоматичного сповіщення неактивних клієнтів.

Способом реалізації стала мова програмування Java в середовищі розробки IntelliJ Idea, бази даних MySQL.

Ключові слова: Автоматична система сповіщення, неактивний клієнт.

ANNOTATION

In this project, we have examined in detail the notification systems, methods of notification delivery, their advantages and disadvantages. The main reasons for notifying customers are considered. A simple interface was developed to demonstrate the program's operation. As a result, a system of automatic notification of inactive customers was created.

The method of implementation was the Java programming language in the IntelliJ Idea development environment, MySQL database.

Keywords: Automatic notification system, inactive client.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система автоматичного	3		
			сповіщення неактивних			
			клієнтів			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Система автоматичного	55		
			сповіщення неактивних			
			клієнтів			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Система автоматичного	1		
			сповіщення неактивних			
			клієнтів			
			Структурна схема системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Система автоматичного	1		
			сповіщення неактивних			
			клієнтів			
			Функціональна схема			
			(діаграма класів)			
	А4	***ІАЛЦ.467200.006 Д3***	Система автоматичного	1		
			сповіщення неактивних			
			клієнтів			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Система автоматичного	10		
			сповіщення неактивних			
			клієнтів			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА
Зм	***Лист***	***№ докум.***	***Підп***	***Дата***	
Розроб		Юзина С.С.			*Система автоматичного сповіщення неактивних клієнтів* Опис альбому
Перев		Трочун Є.В.			

Літ.	Аркуш	Аркушів
	1	1

КПІ ФІОТ
ІВ-92

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система автоматично сповіщення неактивних клієнтів*»

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на автоматичного сповіщення неактивних клієнтів. Система автоматичного сповіщення неактивних клієнтів має широкий спектр застосувань в різних сферах бізнесу. Основна мета такої системи полягає в підтримці зв'язку з клієнтами, які втратили зацікавленість або зупинили взаємодію з певною організацією.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка ефективної системи автоматичного сповіщення неактивних клієнтів, яка допоможе підтримувати та зміцнювати зв'язок з клієнтами.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Виявляти неактивних клієнтів.
- Надати можливість адміністраторам налаштовувати правила та параметри сповіщень
- Відправляти сповіщення неактивним клієнтам згідно з встановленими правилами та налаштуваннями.
- Надати можливість отримувати актуальну інформацію про клієнтів та їх активність.
- Надати вичерпну та зрозумілу документацію для розробленого продукту.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- IntelliJ Idea 2019 IDE версії або вище.

5.3. Вимоги до апаратної частини

- ЦП з архітектурою x86_64 або arm64
- ROM не менше ніж 4 ГБ.
- RAM не менше ніж 1 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	*12.12.2022-18.12.2022*
Вивчення та аналіз завдання	*19.12.2022-19.03.2023*
Розробка архітектури та загальної структури системи	*20.03.2023-02.04.2023*
Розробка структур окремих частин системи	*03.04.2023-15.04.2023*
Програмна реалізація системи	*24.04.2023-05.05.2023*
Виправлення помилок	*06.05.2023-13.05.2023*
Оформлення пояснювальної записки	*15.05.2023-27.05.2023*

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система сповіщення неактивних клієнтів*»

Київ – 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API	(Application Programming Interface) Інтерфейс програмування додатків
GUI	(Graphical User Interface) Графічний інтерфейс користувача
ID	(Identifier Ідентифікатор)
JDBC	(Java Database Connectivity) Підключення до баз даних Java
MVC	(Model-View-Controller) Модель-Вид-Контролер
SMS	(Short Message Service) Служба коротких повідомлень
SQL	(Structured Query Language) Мова структурованих запитів
UML	(Unified Modeling Language) Уніфікована мова моделювання

ВСТУП

В сучасному бізнесі збереження та відновлення клієнтів є одним з ключових факторів успіху. Не лише тому, що залучення нового клієнта вимагає більших витрат, ніж утримання існуючого, але і тому, що довіра та лояльність, які розвиваються протягом тривалого взаємодії з клієнтами, створюють більш стійке та прибуткове партнерство.

Втрата клієнтів, особливо великих і постійних, може призвести до серйозних фінансових втрат і може порушити стратегічне планування та прогнозування компанії. Тому виявлення паттернів в поведінці клієнтів, які можуть вказувати на їхню потенційну неактивність або втрату, є надзвичайно важливим для запобігання таким втратам.

З цього погляду, робота з неактивними клієнтами, що включає їх ідентифікацію, аналіз причин втрати активності, та взаємодію з метою повернення, є дуже важливою. Цей процес потребує стратегічного підходу, а також використання високотехнологічних інструментів та технік, таких як системи автоматичного оповіщення, машинне навчання та аналітика даних, щоб максимізувати ефективність цих зусиль.

Також, ураховуючи постійну зміну ринкових умов, конкурентного середовища, а також вплив нових технологій, поведінка клієнтів стає все менш передбачуваною. Це підкреслює важливість не тільки утримання існуючих відносин з клієнтами, але й адаптації до їхніх змінюваних потреб та вимог.

Важливість такого комплексного підходу до відновлення неактивних клієнтів висвітлюється в контексті сучасної бізнес-моделі, де взаємодія з клієнтами є неперервним процесом. Не вистачає просто продати товар або послугу. Сьогодні успіх вимагає побудови тривалих відносин з клієнтами, які ґрунтуються на довірі та взаємній користі.

Враховуючи, що кожен клієнт є унікальним, системи автоматичного оповіщення можуть допомогти компаніям підходити до кожного клієнта індивідуально, враховуючи їх історію взаємодії, пріоритети, вподобання та поведінку. Це не лише покращує клієнтський досвід, але і створює почуття особистого підходу, що може значно сприяти відновленню неактивних клієнтів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні поняття

Неактивний клієнт - це термін, що часто використовується в бізнесі та маркетингу для позначення клієнта, який перестав користуватися послугами або купувати продукти компанії протягом певного періоду часу. Часовий інтервал, що визначає "неактивність", може варіюватися в залежності від виду бізнесу та специфіки товарів або послуг.

Для роздрібного бізнесу, наприклад, клієнт може вважатися неактивним, якщо він не здійснював покупок протягом останніх 6 місяців. Однак у випадку з послугами, такими як інтернет-провайдинг або страхування, клієнт може вважатися неактивним, якщо він не користувався послугою або не платив за неї протягом декількох місяців.

Ця інформація важлива для компаній, оскільки вона допомагає зрозуміти, які клієнти можуть потребувати додаткової уваги або спеціальних програм, щоб знову стати активними.

Коли ви маєте фізичний бізнес або магазин і натрапляєте на одного зі своїх клієнтів, якого давно не бачили у своєму магазині, перше, що ви скажете: "Привіт, давно не бачилися!". Але що, якщо ваш бізнес ведеться онлайн і ви не можете зустрітися зі своїми клієнтами особисто чи втратите ви цих неактивних клієнтів назавжди?

Саме тоді вступає в дію стратегія реактивації клієнтів. Так само, як підтримувати добрі стосунки з існуючими клієнтами, ви повинні мати план реактивації клієнтів, які втратили активність.

За даними Invesp [1], відновлення взаємодії з неактивними клієнтами обходиться в 5 разів дешевше, ніж залучення нових, тому реактивація клієнтів є набагато більш економічно ефективним підходом.

Згідно зі статистикою, залучення нових клієнтів є значно витратнішим завданням порівняно з утриманням вже існуючих. Перше правило будь-якого успішного бізнесу полягає в зосередженні зусиль на утриманні клієнтів та побудові з ними лояльних стосунків. Цей підхід дозволяє компаніям уникати великих витрат, пов'язаних з маркетинговими кампаніями та залученням нових клієнтів.

Продовжуючи розвивати вже існуючі взаємовідносини з клієнтами, бізнес може використовувати такі стратегії, як особистий підхід, надання переваг і привілеїв для постійних клієнтів, підтримка відділу обслуговування клієнтів та забезпечення якісного сервісу. Ці дії позитивно впливають на задоволеність клієнтів і збільшують їх лояльність до компанії.

За даними досліджень, задоволений клієнт частіше повторно звертається до певної компанії, рекомендує її своїм знайомим та навіть може бути більш готовим придбати нові продукти або послуги, які компанія пропонує. Таким чином, утримання клієнтів сприяє збільшенню обсягів продажів та збільшенню доходів компанії.

Невід'ємною частиною успішної стратегії утримання клієнтів є аналіз та збір даних про них. Інформація про покупців дозволяє компанії краще розуміти їх потреби, попереджати можливі проблеми та пропонувати персоналізовані рішення. Це сприяє покращенню якості обслуговування і збільшенню задоволеності клієнтів.

Утримання існуючих клієнтів виявляється набагато ефективнішою та економічно вигідною стратегією порівняно з постійним залученням нових клієнтів. Це дозволяє компаніям зменшити витрати на маркетинг і рекламу, а також зосередити зусилля на покращенні якості обслуговування та

взаємовідносин з клієнтами. Задоволені та лояльні клієнти, котрі отримують персоналізований підхід та переваги, стають більш вірогідними повторними покупцями, рекомендують компанію своїм знайомим та сприяють збільшенню прибутку. Тому, важливо для бізнесу зрозуміти, що утримання клієнтів має високий пріоритет і варто приділяти достатню увагу побудові стійких взаємовідносин з ними.

1.2 Причини реактивації клієнтів

Реактивація неактивних клієнтів є важливим етапом в стратегії утримання клієнтів, оскільки це може мати значний вплив на зростання бізнесу. Відновлення зв'язку з клієнтами, які раніше не проявляли активності, може мати кілька переваг:

Неактивні клієнти можуть приносити потенційні прибутки компанії. Вони вже мають досвід співпраці з вашою компанією, і відновлення зв'язку може побудувати нові можливості для продажу продуктів або послуг. Це може привести до збільшення обсягів продажів та зростання доходів.

Реактивація неактивних клієнтів може допомогти зберегти репутацію компанії. Якщо клієнти покинули вашу компанію через незадовільний досвід або проблеми, відновлення зв'язку та надання відповідних рішень можуть змінити їхню думку про вашу компанію і зберегти їх як потенційних клієнтів у майбутньому.

На додачу це може покращити їхню лояльність до компанії. Шляхом виявлення причин їхньої неактивності і пропонування персоналізованих рішень або привілеїв, ви демонструєте їм, що вони є важливими для компанії. Це може збільшити їхню готовність знову співпрацювати і стати активними клієнтами.

Реактивація неактивних клієнтів дозволяє компанії зайняти конкурентну перевагу. Багато компаній зосереджуються на залученні нових клієнтів, проте

реактивація неактивних клієнтів дозволяє вам використовувати вже наявну базу клієнтів і конкурувати за їхню увагу. Це може бути ефективним способом збільшення своєї частки на ринку та зміцнення позицій компанії.

Отже, реактивація має великий потенціал для покращення результатів бізнесу. Це дає можливість збільшити доходи, зберегти репутацію, покращити лояльність клієнтів і отримати конкурентну перевагу на ринку. Зосередження на відновленні зв'язку з неактивними клієнтами може бути вигідною стратегією для будь-якої компанії, яка бажає збільшити свій успіх і підвищити задоволення своїх клієнтів.

1.3 Канали зв’язку з клієнтами

Існує багато різних каналів зв'язку з клієнтами, і вибір певних каналів залежить від специфіки вашого бізнесу, цільової аудиторії та ресурсів, які можна реалізувати, наприклад:

- Електронна пошта є одним з основних каналів комунікації з клієнтами. Достатньо надсилати персоналізовані повідомлення, акційні пропозиції, новини та іншу важливу інформацію через електронну пошту.
- Телефон - це прямий і особистий засіб спілкування з клієнтами. Використання телефону дозволяє надавати індивідуальну підтримку, відповідати на запитання та вирішувати проблеми клієнтів.
- Соціальні медіа стали популярними каналами зв'язку з клієнтами. Можна використовувати платформи, такі як Facebook, Twitter, Instagram і LinkedIn, для надання інформації, відповідей на запитання, розв'язання скарг і залучення до діалогу з вашої аудиторією.

- Розробка мобільного додатку для бізнесу дозволяє забезпечити постійний доступ клієнтів до компанії, надавати персоналізовану інформацію, сповіщення та зручність в користуванні.
- Фізичні пункти обслуговування. Якщо бізнес має фізичну присутність, наприклад, магазини або офіси, можна забезпечити зв'язок з клієнтами безпосередньо через особисту зустріч, консультацію або послуги підтримки на місці.

Вибір каналів зв'язку залежить від потреб аудиторії та можливостей компанії. Зазвичай ефективна стратегія включає використання комбінації різних каналів для максимального охоплення та задоволення потреб клієнтів.

1.3.1 SMS -повідомлення

Використання телефонного зв'язку, включаючи смс-повідомлення, є ефективним каналом зв'язку з клієнтами, зокрема в контексті реактивації неактивних клієнтів. Особистий характер телефонних розмов і можливість миттєвої комунікації забезпечують кілька переваг для відновлення зв'язку та зацікавлення клієнтів. Телефон дозволяє зв'язатися з клієнтами безпосередньо і спілкуватися з ними один на один. Це створює можливість виявити їхні потреби, відповісти на запитання та вирішити проблеми безпосередньо, що може позитивно вплинути на їхню увагу та відновлення активності[2].

Люди ймовірніше побачать, що їм надсилають - коефіцієнт відкритих SMS сягає 98% [3]. Це надзвичайно високий показник, що свідчить про ефективність SMS-маркетингу як засобу комунікації з аудиторією. Завдяки такій високій швидкості прочитання повідомлень, є змога миттєво доставляти важливу інформацію, пропозиції або розсилати акційні промокоди. Будь-які оновлення, новини або повідомлення про знижки будуть ймовірніше зауважені, що дозволяє збільшити взаємодію з клієнтами. За допомогою SMS-маркетингу

можна швидко та ефективно комунікувати з клієнтами, забезпечуючи надійну і доступну спілкування в реальному часі.

Є змога використовувати смс-повідомлення для надсилання персоналізованих повідомлень неактивним клієнтам. Це може бути коротке, але змістовне повідомлення, яке залучає їхню увагу, нагадує про вашу компанію та пропонує їм спеціальну пропозицію або знижку для стимулювання повторних покупок. Також смс-повідоплення дозволяє швидко відповісти на запити та запитання клієнтів. Надання швидкої та професійної підтримки може позитивно вплинути на сприйняття вашої компанії та підтримати реактивацію неактивних клієнтів.

Як один із принципів застосування можна здійснювати слідування за неактивними клієнтами та нагадувати про компанію час від часу. Можна запланувати дзвінки або надсилання смс-повідомлень з пропозиціями або нагадуваннями про нові продукти, акції або події, що може спонукати їх до повернення.

Використання телефонного зв'язку, зокрема смс-повідомлень, в реактивації клієнтів може забезпечити особистий підхід, швидку відповідь та можливість побудувати знову контакт з неактивними клієнтами. Це може стимулювати їхню увагу, залучення та повторні покупки.

1.3.2 Електронна пошта

Електронна пошта є потужним і ефективним каналом зв'язку з клієнтами, особливо в контексті реактивації неактивних клієнтів. Використання електронної пошти для реактивації клієнтів має кілька переваг і може бути частиною успішної стратегії утримання клієнтів.

Вона дозволяє надсилати персоналізовані повідомлення неактивним клієнтам. Можна використовувати їхні імена, інформацію про їхні попередні покупки або інші відомості, щоб створити зв'язок і зацікавити їх.

Слугує платформою для надсилання спеціальних пропозицій та акцій неактивним клієнтам, щоб залучити їхню увагу і стимулювати до повторних покупок. Це можуть бути знижки, купони або інші переваги, які стимулюють клієнтів повернутися до вашої компанії.

Можна використовувати електронну пошту для повідомлення клієнтам про важливі оновлення, новини або зміни в вашому бізнесі. Це можуть бути нові продукти, послуги, події або будь-які інші зміни, які можуть зацікавити клієнтів і підштовхнути їх до активності.

Використання розсилок та автоматизованих систем електронної пошти дозволяє легко взаємодіяти з великою кількістю неактивних клієнтів. Є можливість налаштувати автоматичні розсилки, які будуть надсилати послідовність повідомлень певним клієнтам з метою їх реактивації[2].

Електронна пошта є потужним і гнучким каналом зв'язку, який можна використовувати для ефективної реактивації неактивних клієнтів. Варто зосередитися на створенні персоналізованих повідомлень, привабливих акцій і важливої інформації, використанні автоматизації та аналізу метрик для досягнення успіху у реактивації клієнтів через електронну пошту.

1.4 Проблематика

Хоча системи оповіщення неактивних клієнтів мають багато переваг, вони також мають деякі виклики та проблеми:

- Компанії мають бути обережні, щоб не надсилати занадто багато повідомлень або електронних листів, що можуть сприйматися як спам. Це може розірвати відносини з клієнтами і відштовхнути їх.
- Збір та використання даних клієнтів для цілей оповіщення повинно відбуватися з повагою до їх приватності та у відповідності до

законодавства про захист даних, такого як загальний регламент про захист даних (GDPR) в Європі.

- Потреба в персоналізації. Для ефективного залучення неактивних клієнтів компаніям часто потрібно створювати персоналізовані повідомлення, що відображають інтереси та попередній досвід клієнтів. Це вимагає використання складного аналітичного програмного забезпечення і може бути викликом для невеликих або менш технологічно розвинених компаній.
- Визначення ефективності кампаній повернення клієнтів може бути важким. Компанії потребують встановити відповідні метрики та засоби відстеження, щоб з'ясувати, чи дійсно їхні зусилля приводять до повернення неактивних клієнтів.

Спам-проблематика в оповіщеннях клієнтів є серйозною проблемою, з якою стикаються багато компаній. Несправедливе або небажане надсилання повідомлень може негативно вплинути на сприйняття бренду, завдає шкоди взаємодії з клієнтами та порушує їхню приватність. Щоб уникнути спаму та забезпечити ефективне та етичне спілкування з клієнтами необхідно[4]:

По-перше, перед надсиланням будь-яких повідомлень клієнтам, переконайтись, що є згода та підтвердження. Забезпечення відповідних механізмів, таких як підписка на розсилку або підтвердження електронною поштою, дозволяє впевнитися, що клієнти свідомо погоджуються отримувати повідомлення.

По-друге, повідомлення, які надсилають клієнтам, повинні містити чітку та зрозумілу інформацію про те, що вони мають очікувати. Пояснити, які типи повідомлень будуть надсилатись, яка буде їхня регулярність та як клієнти можуть відписатися в разі потреби.

По-третє, спам-повідомлення зазвичай характеризуються загальними та неперсоналізованими повідомленнями. Щоб уникнути цього, варто звернути

увагу на персоналізацію та релевантність. Використовуйти дані про клієнтів та їхні покупки, щоб надавати індивідуальні та цікаві повідомлення, які відповідають їхнім потребам і попереднім взаємодіям з компанією.

По-четверте, важливо, щоб клієнти мали можливість легко та безпроблемно відписатися від отримання повідомлень. Варто чітко показати, як вони можуть скасувати підписку або внести зміни у свої налаштування розсилки. Необхідно зробити цей процес простим та доступним.

По-п’яте, переконайтеся, що повідомлення мають цінність для клієнтів. Варто надати корисну інформацію, спеціальні пропозиції або знижки, які дійсно цікаві та вигідні для клієнтів.

Використання клієнтських даних, персоналізація та забезпечення високої якості та релевантності повідомлень допомагають уникнути спаму і забезпечити позитивний досвід спілкування з клієнтами. Враховуючи ці аспекти, можна ефективно комунікувати з клієнтами, уникнути спаму та зберегти їхню довіру та лояльність до компанії.

Не менш важливо мати на увазі, що використання певних слів і фраз, які часто сприймаються спам-фільтрами як ознаки небажаних повідомлень, може негативно впливати на доставку та сприйняття повідомлень клієнтами.

Варто утриматись від використання загальних фраз, як "Найкраща ціна" або "Дешево". Замість цього, треба надати конкретну інформацію про вигоди або переваги пропозиції. Не використовувати надмірні знаки оклику (!!!) або великі літери у заголовках чи тексті повідомлень, оскільки це може сприйматися як небажані рекламні техніки. Застосовувати мову, що не прямо спрямована на фінансовий прибуток, як "Чистий прибуток".

ВИСНОВОК ДО РОЗДІЛУ 1

Поєднання SMS -повідомлень і електронної пошти як способу зв'язку з клієнтами може бути дуже ефективним і результативним. Комбінування цих двох каналів надає можливість взаємодіяти з клієнтами на різних рівнях і забезпечувати повністю охоплюючу комунікацію.

SMS-повідомлення є короткими і прямими сповіщеннями, які миттєво досягають клієнтів. Вони здатні привернути увагу клієнта, особливо за умови, що мобільні пристрої завжди під рукою. Використання смс-повідомлень може бути ефективним для нагадування про спеціальні пропозиції, акції або важливі події. Вони також можуть бути використані для надсилання персоналізованих повідомлень та підтримки клієнтів.

Електронна пошта, з іншого боку, надає більше простору для розгорнутого спілкування з клієнтами: надсилати детальніші повідомлення, включати зображення, посилання та інші матеріали. Електронна пошта є ефективним каналом для надання більшої кількості інформації, такої як новини про компанію, продукти, оновлення та бонусні програми. Вона також дозволяє створювати персоналізовані повідомлення, використовуючи дані про покупки та інші відомості про клієнтів.

Комбінування SMS-повідомлень і електронної пошти дозволяє створити повноцінну комунікаційну стратегію з клієнтами. Використовувати смс-повідомлення для негайних сповіщень і нагадувань, а електронну пошту - для більш детальної інформації та персоналізованих повідомлень. Крім того, використовувати дані про відкриття та перегляд посилань в електронних листах, щоб аналізувати ефективність та оптимізувати комунікаційні зусилля.

Важливо при цьому дотримуватися правил етики і надавати клієнтам можливість відписатися від отримання повідомлень, якщо вони бажають.

РОЗДІЛ 2. АНАЛІЗ СПОСОБІВ РЕАЛІЗАЦІЇ

2.1 Мова Java

2.1.1 Переваги Java

Обирання мови програмування для проекту є критично важливим етапом, який вимагає глибоких роздумів та уважного аналізу. Від цього вибору залежить не лише ефективність розробки, але й успіх проекту в цілому.

Обрано мову програмування Java для розробки програми для автоматичного оповіщення клієнтів з кількох причин[5].

По-перше, Java є однією з найпопулярніших та найпоширеніших мов програмування у світі. Вона має велику спільноту розробників, багато документації та різноманітних ресурсів, що робить її легко доступною для вивчення та розвитку навичок.

По-друге, Java є кросплатформеною мовою програмування, що означає, що програми, написані на Java, можуть працювати на різних операційних системах, таких як Windows, macOS та Linux. Це дає мені можливість створювати програму для оповіщення клієнтів, яка буде сумісна з різними пристроями та платформами, що є важливим для забезпечення гнучкості та широкого охоплення моєї аудиторії.

По-третє, Java має потужну екосистему, включаючи багато різних фреймворків та бібліотек. Один з найпопулярніших фреймворків для розробки додатків на Java - Spring. Spring надає широкі можливості для розробки програм для оповіщення клієнтів, включаючи підтримку веб-розвитку, інтеграцію зовнішніх сервісів, комунікацію в режимі реального часу та безпеку. Використання Spring дозволяє мені ефективно використовувати різноманітні компоненти та функціональні можливості, що сприяє швидкій розробці та забезпечує надійність мого додатку.

Крім того, Java також має вбудовану систему безпеки, яка допомагає уникнути багатьох типових проблем безпеки програм. Це особливо важливо для програм, які обробляють конфіденційну інформацію клієнтів. За допомогою інструментів та практик безпеки, доступних в Java, я можу забезпечити захищеність та конфіденційність даних моїх клієнтів.

Отже, вибір Java для розробки програми для автоматичного оповіщення клієнтів базується на її популярності, кросплатформеності, потужній екосистемі та вбудованій системі безпеки. Вона дозволяє мені створити надійний, гнучкий та безпечний додаток, який забезпечить задоволення потреб моїх клієнтів.

При розробці програм на Java використовуються деякі основні принципи, які сприяють створенню якісного, ефективного та легкозрозумілого коду. Ось кілька основних принципів при розробці програм на Java:

- Об'єктно-орієнтоване програмування (ООП): Java є об'єктно-орієнтованою мовою програмування, що означає, що програми розробляються з використанням об'єктів, які взаємодіють один з одним. ООП дозволяє організувати код в модульні, повторно використовувані компоненти, що спрощує розробку, тестування та підтримку програм.
- Принцип SOLID є абревіатурою, яка описує п'ять основних принципів доброго проектування програм, що сприяють збереженню гнучкості, розширюваності та підтримки коду. Ці принципи включають принцип єдиної відповідальності (Single Responsibility Principle), принцип відкритості/закритості (Open/Closed Principle), принцип підстановки Лісков (Liskov Substitution Principle), принцип розділення інтерфейсу (Interface Segregation Principle) та принцип інверсії залежностей (Dependency Inversion Principle).
- Контроль якості коду. Розробники Java активно використовують інструменти контролю якості коду, такі як статичні аналізатори (наприклад, SonarQube), тестування одиниць (JUnit) та інструменти для

автоматичного перевірки стилю коду (наприклад, Checkstyle або PMD). Ці інструменти допомагають виявляти та виправляти проблеми у коді, підвищуючи якість та надійність програм.

- Рекомендується використовувати документацію у вигляді коментарів (JavaDoc) для пояснення функціональності, параметрів та поведінки коду. Це допомагає іншим розробникам розуміти ваш код та використовувати його належним чином.
- Java має потужну систему керування винятками, яка дозволяє обробляти та відновлювати помилки під час виконання програми. Важливо правильно керувати винятками, використовувати виключення лише для ситуацій, що вимагають обробки, та вживати належних заходів для відновлення нормальної роботи програми.
- Тестування вважається важливою складовою розробки на Java. Використання фреймворків тестування, таких як JUnit або TestNG, допомагає створювати автоматизовані тести, які перевіряють функціональність та коректність коду. Це забезпечує стабільність програми та допомагає виявляти та виправляти помилки.

Ці принципи визначають основи розробки програм на Java і сприяють створенню якісного та ефективного коду. Використання цих принципів допомагає створювати програми, які є зрозумілими, легко зберігаються та підтримуються протягом тривалого часу.

2.1.2 Java Swing

Java Swing — це частина Java Foundation Classes (JFC), яка включає у себе набір бібліотек для створення графічного користувацького інтерфейсу (GUI) в Java. Swing надає класи для вікон, кнопок, текстових полів, випадаючих меню, таблиць, списків, слайдерів та багато іншого.

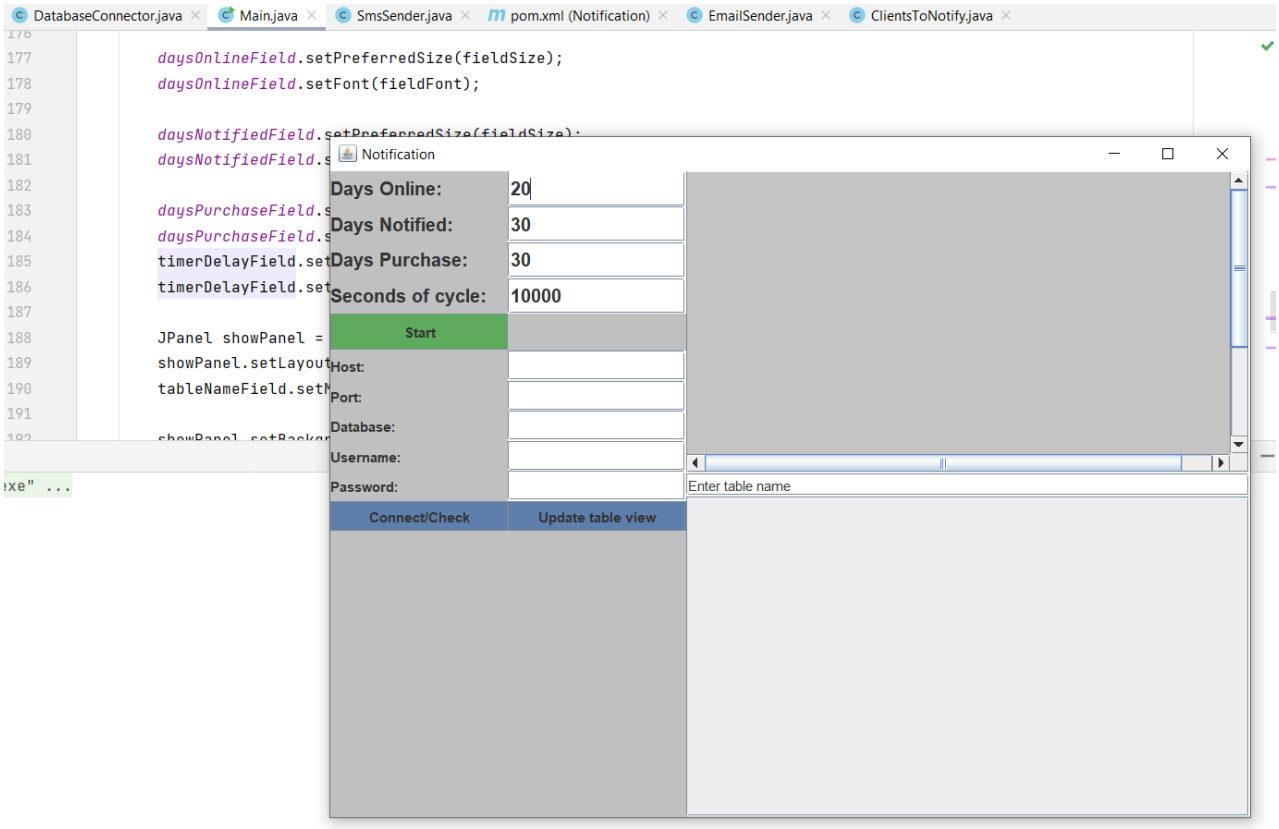


Рисунок 2.1 – зовнішній вигляд Swing-додатку

Одна з ключових переваг Swing полягає в його незалежності від платформи, яка досягається завдяки використанню Java для малювання елементів інтерфейсу на екрані, а не специфічного API операційної системи, як це робить AWT (Abstract Window Toolkit). Завдяки цьому, програми, розроблені на Swing, виглядають і працюють однаково на будь-якій платформі з підтримкою Java.

До того ж, Swing відрізняється значною здатністю налаштовуватись. Він пропонує велику кількість готових до використання компонентів GUI, але також дозволяє створювати власні компоненти або налаштовувати вигляд та поведінку існуючих. Понад те, Swing надає можливість змінювати вигляд GUI за допомогою різних "скинів" або "планок", що дозволяє адаптувати вигляд інтерфейсу без зміни коду.

Ще одна важлива особливість Swing - підтримка шарування компонентів, що дозволяє додавати кілька компонентів в одне і те ж місце ієрархії компонентів, створюючи шари. Ця функція надає більшу гнучкість при проектуванні GUI[6].

Також включає підтримку додаткових графічних особливостей, включаючи використання зображень в кнопках, створення плаваючих вікон, контекстних меню та інтеграцію з drag-and-drop. Незважаючи на те, що Swing не є найновішою технологією для створення GUI в Java він все ще широко використовується, особливо у великих та старіших проектах.

Загалом, Swing - це дуже потужний інструмент для розробки GUI. Хоча він може бути дещо складнішим у вивченні та використанні, ніж деякі інші інструменти, він надає велику гнучкість та контроль. Для розробників, що прагнуть створити власні незалежні від платформи GUI-застосунки, Swing є міцним та добре дослідженим вибором.

2.1.3 Об'єктно-орієнтоване програмування в Java

Об'єктно-орієнтоване програмування (ООП) - це парадигма програмування, яка базується на концепції об'єктів, які представляють реальні або віртуальні об'єкти з реального світу. У Java, як мові, що підтримує ООП, основні принципи об'єктно-орієнтованого програмування включають наступне[7]:

- Класи визначають структуру, поведінку та властивості об'єктів. Об'єкти є конкретними представниками класу і мають стан (поля) та поведінку (методи), які описують їх функціональність.
- Інкапсуляція означає згортання даних та методів, що оперують над цими даними, в єдиному об'єкті. Це дозволяє захистити дані від прямого доступу та забезпечити контроль над ними через публічні та приватні методи. Інкапсуляція сприяє безпеці, модульності та підтримці коду.

- Наслідування дозволяє створювати нові класи на основі вже існуючих, успадковуючи їх властивості та методи. Це дозволяє створювати ієрархію класів, що полегшує повторне використання коду, створення більш абстрактних та спеціалізованих класів та поліпшує модульність програми.
- Поліморфізм дозволяє об'єктам одного класу використовуватися як об'єкти іншого класу. Це означає, що можна використовувати багатоформальність (наприклад, методи) для різних типів об'єктів. Поліморфізм покращує гнучкість, розширюваність та повторне використання коду.
- Абстракція дозволяє скрити деталі реалізації та представити лише сутність або інтерфейс об'єкта. Це дозволяє розробникам працювати на вищому рівні абстракції, не займаючись дрібними деталями реалізації. Абстракція полегшує розуміння та використання коду, спрощує розробку та підтримку програм.

Ці принципи об'єктно-орієнтованого програмування визначають спосіб організації та розробки програм на Java. Вони полегшують створення модульного, повторно використовуваного та легкозрозумілого коду, забезпечують гнучкість, розширюваність та підтримку програм протягом тривалого часу.

2.1.4 Принцип SOLID

Принцип SOLID - це набір основних принципів доброго проектування програм, які сприяють створенню гнучких, масштабованих та підтримуваних кодових баз. SOLID - це абревіатура, де кожна літера відповідає конкретному принципу[8]:

Принцип єдиної відповідальності (Single Responsibility Principle - SRP). Цей принцип стверджує, що кожен клас або модуль повинен мати лише одну

причину для зміни. Кожен об'єкт або клас повинен бути відповідальним лише за одну частину функціональності. Це полегшує зміну, тестування та розуміння коду, а також зменшує ймовірність негативних впливів змін на інші частини системи.

Принцип відкритості/закритості (Open/Closed Principle - OCP). За цим принципом класи повинні бути відкритими для розширення, але закритими для зміни. Це означає, що замість зміни коду існуючих класів, варто розширювати їх через успадкування або використовувати композицію, щоб додати нову функціональність. Це забезпечує стійкість системи до змін та сприяє її розширюваності.

Принцип підстановки Лісков (Liskov Substitution Principle - LSP). Цей принцип стверджує, що об'єкти одного класу повинні бути замінювані об'єктами підкласу без зміни правильності програми. Це означає, що не варто порушувати контракт базового класу при використанні його підкласів. Цей принцип допомагає забезпечити сприятливу для утримання систему та поліпшити перевикористання коду.

Принцип розділення інтерфейсу (Interface Segregation Principle - ISP). За цим принципом клієнти не повинні залежати від інтерфейсів, які вони не використовують. Це означає, що великі інтерфейси повинні бути розбиті на менші та більш специфічні, щоб кожен клієнт отримував тільки необхідну йому функціональність. Це сприяє зменшенню залежності між компонентами та забезпечує більшу гнучкість та перевикористання коду.

Принцип інверсії залежностей (Dependency Inversion Principle - DIP). Цей принцип стверджує, що класи повинні залежати від абстракцій, а не від конкретних реалізацій. Замість того, щоб прямо залежати від інших класів, необхідно використовувати інтерфейси або абстрактні класи, що дозволяє замінювати реалізації без зміни коду клієнта. Це забезпечує слабку залежність між класами, сприяє перевикористанню та спрощує тестування.

Дотримання принципів SOLID допомагає створювати гнучкі, модульні та легкозрозумілі кодові бази. Ці принципи сприяють стійкості до змін, полегшують розширення та підтримку коду, покращують перевикористання та поліпшують якість програм.

Принцип SOLID (Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle) надає набір принципів, які допомагають створювати гнучкі, розширювані та легкозрозумілі кодові бази. Вони сприяють модульності, перевикористанню та підтримці коду протягом тривалого часу.

Об'єктно-орієнтоване програмування (ООП) визначає парадигму, в якій програми будуються з використанням об'єктів, які взаємодіють один з одним. ООП дозволяє структурувати код у вигляді класів та об'єктів, сприяючи модульності, перевикористанню та легкості розуміння коду.

Розуміння та використання цих концепцій допомагає розробникам створювати високоякісні програми на Java, які є надійними, гнучкими та легко зберігаються та підтримуються протягом тривалого часу. Вони допомагають забезпечити правильне керування винятками, ефективне використання принципів SOLID та використання переваг об'єктно-орієнтованого програмування для створення високоякісного коду.

2.1.5 Недоліки мови Java

Мова програмування Java стала однією з найбільш популярних мов програмування у світі завдяки своїй портативності, масштабованості та гнучкості. Однак, як і у всього, в Java є недоліки, які стають помітними, коли ми вивчаємо її глибше. В цьому підпункті ми детально проаналізуємо деякі з цих недоліків, щоб мати більше розуміння потенційних викликів, з якими можуть зіткнутися розробники, коли вони працюють з Java.

Java відома своєю порівняно високою продуктивністю порівняно з деякими іншими мовами програмування високого рівня. Однак, порівняно з компільованими мовами, такими як C++ або Go, Java може бути трохи повільнішою. Одна з причин такої повільності полягає в тому, що Java використовує JVM для виконання коду. Коли виконується компіляція програми Java, вона перетворюється на байт-код, який виконується на JVM. Віртуальна машина виконує цей байт-код, але процес інтерпретації може займати певний час. Існує також поняття Just-In-Time (JIT) компіляції, яке використовується в JVM для оптимізації виконання коду.[9] Коли програма виконується, JVM аналізує її і вибирає шляхи оптимізації. Оптимізований код потім кешується і використовується для подальших викликів. Однак, цей процес JIT-компіляції також може вплинути на продуктивність, оскільки він займає певний час перед початком виконання оптимізованого коду.

Хоча Java може бути повільнішою порівняно з компільованими мовами, варто відзначити, що різниця у продуктивності може бути непомітною в багатьох сценаріях програмування. Більшість програм, особливо ті, які працюють з мережевими операціями або базами даних, витрачають більшу частину часу на зовнішні операції, а не на саме виконання коду.

Java використовує систему керування пам'яттю, яка забезпечує автоматичний збір сміття. Це означає, що не потрібно явно виділяти та звільняти пам'ять для об'єктів. Замість цього, JVM автоматично відслідковує, коли об'єкти більше не використовуються в програмі, і звільняє пам'ять, яку вони займають. Автоматичний збір сміття має свої витрати з точки зору ресурсів. JVM повинна постійно відслідковувати та аналізувати об'єкти, щоб визначити, чи вони все ще потрібні. Цей процес може призвести до певної паузи в роботі програми для збору сміття. Окрім того, система керування пам'яттю Java може використовувати більше пам'яті, ніж у деяких інших мовах програмування. Це пов'язано з тим, що JVM використовує певні механізми для оптимізації використання пам'яті та підтримки автоматичного збору сміття.

Необхідність у використанні більшої кількості пам'яті може бути проблемою, особливо в розрахунку на обмежені ресурси пам'яті, наприклад, на пристроях з обмеженою пам'яттю або високонавантажених серверних середовищах.

Мові Java для виконання відносно простих завдань може знадобитись значна кількість коду. Це пов'язано з особливостями синтаксису та структури мови. Java має певні обов'язкові конструкції, такі як декларації класів, методів, імпорти, обробники виключень і т. д. Ці конструкції часто потребують багато ключових слів та додаткового коду, що може збільшити обсяг програми. Це може ускладнити читання та написання коду, особливо для новачків у програмуванні. Наприклад, для написання простої програми вводу-виводу може знадобитись кілька рядків коду, включаючи створення об'єктів, обробку виключень і закриття потоків.[9] Хоча це забезпечує більшу безпеку та надійність програм, це може здаватись надмірним для простих завдань.

Java зосереджується на високорівневому програмуванні та абстракціях, що робить її відмінним вибором для розробки багатьох видів додатків. Проте, вона не надає можливостей для більш низькорівневого програмування, як, наприклад, пряме управління пам'яттю або використання покажчиків, як в мовах C або C++. Це обмеження може становити проблему у випадках, коли потрібен прямий доступ до апаратних ресурсів, оптимізоване управління пам'яттю або використання низькорівневих бібліотек. Наприклад, в деяких областях, таких як вбудовані системи або розробка драйверів пристроїв, може бути необхідно використовувати мови програмування з низькорівневими можливостями для досягнення більшої ефективності та контролю.

Загальні недоліки Java включають обмежену продуктивність порівняно з низькорівневими мовами, використання більше пам'яті через автоматичний збір сміття, ускладнене читання та написання коду, відсутність можливостей для більш низькорівневого програмування.

2.2 Мова C#

2.2.1 Переваги C#

При виборі мови програмування для створення програмного забезпечення, важливо врахувати ряд чинників, включаючи набір доступних функцій, продуктивність, безпеку, підтримку спільноти та інші аспекти. Це особливо важливо для програм для автоматичного оповіщення клієнтів, що вимагають надійності, масштабованості, високої продуктивності та можливості інтеграції з різноманітними системами.

Мова програмування C# є однією з найпотужніших та найбільш гнучких мов, які доступні сьогодні. Створена Microsoft як частина платформи .NET, C# поєднує в собі простоту та виразність з потужними можливостями для створення широкого спектра застосунків - від простих консольних програм до складних веб-сервісів та мобільних додатків.

C# був розроблений з метою створення мови, яка була б простою для розуміння і вивчення, але при цьому дуже потужною. Мова має чітку структуру та синтаксис, що дуже схожий на такі мови, як Java і C++. Це означає, що програмісти, які вже знайомі з цими мовами, можуть легко освоїти C#. Для нових програмістів, C# пропонує добре організовану структуру, що полегшує навчання.

Допомагає зробити код більш читабельним. Синтаксис C# містить ключові слова, які легко зрозуміти, і це спрощує розуміння того, що робить код. Наприклад, ключове слово *foreach* в C# дозволяє легко перебрати всі елементи в колекції, що робить код більш зрозумілим, ніж традиційний цикл *for*. C# намагається уникнути певних особливостей, які можуть призвести до помилок. Наприклад, C# вимагає від програмістів явно вказувати, коли вони хочуть використати перевантаження операторів або конвертувати типи даних, що знижує ризик ненавмисних помилок. Тому, використання C# для створення

програми автоматичного оповіщення може бути дуже доцільним вибором, залежно від конкретних вимог і обставин проекту.

C# відноситься до мов програмування з сильною типізацією, яка обумовлює, що кожна змінна або об'єкт визначається конкретним типом, що не змінюється непомітно. Ця властивість створює додатковий рівень безпеки коду, вимагаючи від програміста ясно визначити тип даних для кожної змінної при її декларації[10]. Такий підхід знижує ймовірність виникнення помилок, що спричинені неправильним обробленням типів даних. Крім того, C# має автоматичне керування пам'яттю, що відбувається через механізм збирача сміття (Garbage Collector). Цей збирач сміття здійснює автоматичне видалення об'єктів, що вже не використовуються програмою. Така особливість не тільки спрощує життя розробникам, але й надає додаткову безпеку, уникнення помилок, які виникають від ручного керування пам'яттю. Ці помилки зазвичай є поширеними при використанні мов програмування, що дозволяють безпосереднє управління пам'яттю, таких як C++.

Отже, комбінація сильної типізації та автоматичного керування пам'яттю в C# допомагає забезпечити надійність та безпеку коду, що є невід'ємними елементами при створенні програм для автоматичного оповіщення клієнтів.

Багатозадачність є одним з ключових аспектів сучасного програмування, особливо при розробці програм для автоматичного оповіщення клієнтів, де різні процеси мають працювати паралельно або асинхронно. C# надає декілька засобів для реалізації багатозадачності.[11] C# дозволяє створювати множину потоків виконання в рамках одного процесу. Кожен потік може виконувати різні завдання незалежно від інших, що дозволяє паралельне виконання завдань. C# включає в себе вбудовану підтримку для асинхронного програмування за допомогою ключових слів *async* та *await*. Це дозволяє програмі виконувати довготривалі операції (наприклад, доступ до мережі або великі обчислення) без блокування основного потоку виконання, що підтримує високу продуктивність і реактивність програми. Task Parallel Library надає засоби для розподілу

обчислень між різними ядрами процесора та для організації паралельних обчислень на високому рівні. За допомогою TPL програміст може ефективно використовувати всі ресурси процесора, що значно збільшує швидкість виконання обчислювально важких завдань. Мова має спеціалізовані колекції, які призначені для безпечного використання в багатопоточному середовищі. Ці колекції, такі як *ConcurrentDictionary* або *BlockingCollection*, допомагають уникнути типових проблем, пов'язаних з багатопоточністю, таких як умови гонки.

Ці механізми багатозадачності в C# дозволяють програмам ефективно використовувати системні ресурси, покращувати продуктивність та забезпечувати високий рівень реактивності, що особливо важливо для програм автоматичного оповіщення клієнтів.

2.2.2 Недоліки C#

C# є потужною мовою програмування, з великою кількістю бібліотек та широким спектром застосувань. Однак, незважаючи на свої переваги, вона також має деякі недоліки, які варто враховувати при розгляді її використання для проектів. У цьому розділі ми детально проаналізуємо ці недоліки, їх вплив та розглянемо, які фактори можуть вплинути на рішення щодо використання C# у нашому проєкті. Проаналізовані недоліки надають контекст та перспективу, що допоможе у виборі оптимального рішення та врахуванні потенційних обмежень.

C#, подібно до Java, використовує автоматичний збір сміття (Garbage Collection), що спрощує управління пам'яттю для програмістів. Автоматичний збір сміття дозволяє відслідковувати та звільняти непотрібні об'єкти, що полегшує процес роботи з пам'яттю. Проте, використання автоматичного збору сміття може призвести до вищого використання пам'яті порівняно з мовами, які надають більше контролю над управлінням пам'яттю, такими як C++. Збір

сміття вимагає додаткових системних ресурсів для відстеження та очищення об'єктів, які більше не використовуються в програмі. Це може призводити до збільшення використання пам'яті, особливо в сценаріях з великим обсягом створюваних об'єктів або роботи з великими наборами даних. Збільшення використання пам'яті може стати проблемою в ситуаціях, коли доступні обмежені ресурси пам'яті, наприклад, на вбудованих системах або мобільних пристроях. Крім того, велике використання пам'яті може впливати на продуктивність програми, особливо якщо операційна система починає використовувати віртуальну пам'ять або додаток конкурує за ресурси з іншими процесами. Отже, використання автоматичного збору сміття може призводити до вищого використання пам'яті, що може мати вплив на продуктивність програми та роботу з обмеженими ресурсами пам'яті[12].

Мова C# зазвичай використовується досить часто для багатьох прикладних випадків, вона може бути повільнішою порівняно з низькорівневими мовами, такими як C++ або Rust. Це пов'язано з тим, що C# зорієнтована на високорівневе програмування і має певні рівні абстракції, які забезпечують зручний синтаксис та широкі можливості. Проте, ці рівні абстракцій також можуть мати свою ціну у вигляді додаткових шарів оптимізацій та обробки, які можуть затримувати виконання програми. Наприклад, автоматична обробка винятків та розподіл пам'яті можуть вимагати додаткового часу виконання. Це може бути проблемою для додатків, які вимагають високої продуктивності або максимального використання ресурсів обчислювальної системи. У таких випадках, коли кожна мілісекунда та кожен байт важливі, низькорівневі мови можуть забезпечити кращу продуктивність.

Подібно до Java, C# може вимагати більше коду для виконання певних завдань порівняно з деякими більш сучасними, скороченими мовами програмування, такими як Python або JavaScript. Це пов'язано з особливостями синтаксису та структури мови.

Наприклад, декларації класів, методів, обробники винятків та інші конструкції мови можуть вимагати додаткових рядків коду порівняно з іншими мовами, які мають більш спрощений синтаксис. Це може зробити код C# трохи більш об'ємним та складним для читання.

2.2.3 Windows Forms

Windows Forms, відомий як WinForms, є ключовою бібліотекою в стеку технологій C#, яка відрізняється своїм багатим набором елементів керування. Ця особливість надає розробникам широкі можливості для створення візуально привабливих і функціонально повноцінних інтерфейсів користувача. Елементи керування WinForms включають, але не обмежуються такими засобами, як кнопки, текстові поля, випадаючі списки, чекбокси, радіокнопки, панелі прокрутки, вкладки і багато інших. Всі вони забезпечують базові компоненти для створення взаємодії користувача з додатком. Діалогові вікна та форми, які служать контейнерами для цих елементів керування, створюють структуру взаємодії користувача з програмою. Вони можуть бути налаштовані за розміром, стилем, поведінкою і можуть містити будь-яку комбінацію елементів керування. Меню та панелі інструментів також є важливою частиною елементів керування WinForms, дозволяючи розробникам створювати зручні навігаційні структури та доступ до ключових функцій програми.

Важливою перевагою цих елементів керування є їхня гнучкість. Вони можуть бути налаштовані за допомогою властивостей і методів, які надає WinForms, що дозволяє змінювати їх вигляд і поведінку, щоб відповідати вимогам конкретного додатка.

Простота використання Windows Forms є ще одним значним аргументом на користь цього фреймворку при створенні програм для автоматичного оповіщення клієнтів на мові C#. Спрощена модель розробки WinForms поєднує гнучкість C# з доступністю візуального дизайнера, що сприяє більш

ефективному процесу створення програм. Візуальний дизайнер, особливо в таких середовищах розробки як Visual Studio, є ключовим інструментом для створення інтерфейсу в WinForms[13]. Його «drag-and-drop» функціональність дозволяє розробникам легко маніпулювати елементами керування на формах, встановлюючи їх розміри, положення та інші властивості візуально, без необхідності писати велику кількість коду. Прикладом цього є обробка подій. Події, такі як натискання кнопки або вибір елемента в списку, є основними в інтерактивних програмах. Візуальний дизайнер WinForms спрощує процес встановлення обробників подій, дозволяючи розробникам автоматично генерувати код обробника подій просто шляхом подвійного клацання на елементі керування. Ця інтегрованість між візуальним дизайнером та кодом знижує поріг входження для нових розробників, а також забезпечує високу продуктивність для досвідчених програмістів. Вона дозволяє розробникам зосередитися на бізнес-логіці та вимогах користувачів, замість ручного кодування низькорівневих деталей інтерфейсу.

2.3 MySQL

Для успішного виконання будь-якої системи обробки даних, незалежно від її складності або обсягу, необхідна надійна, високоефективна і зручна в експлуатації база даних. MySQL є однією з найпоширеніших та надійних систем управління базами даних (СУБД) у світі. Вона має широку підтримку спільноти розробників і документацію, що робить її легко доступною для використання та розвитку. Будучи частиною LAMP-стеку (Linux, Apache, MySQL, PHP), MySQL відома своєю швидкодією та здатністю ефективно обробляти великі обсяги даних, що важливо для роботи з великою базою клієнтів.

2.3.1 Переваги MySQL

MySQL володіє рядом особливостей, які сприяють її простоті використання та надають значні переваги для розробників додатків всіх рівнів.

Простота SQL синтаксису. SQL, або Structured Query Language, є стандартною мовою для взаємодії з реляційними базами даних, і MySQL використовує її без будь-яких значних модифікацій. Така простота мови дає можливість розробникам швидко навчитися основам роботи з MySQL і зосередитися на вирішенні конкретних задач, не витрачаючи час на вивчення складних мовних конструкцій.

Поряд з SQL синтаксисом, MySQL пропонує вбудовані інструменти для керування базою даних, такі як MySQL Workbench, які дозволяють виконувати широкий спектр задач - від створення та модифікації схем бази даних до виконання SQL-запитів і управління даними. Ці інструменти мають інтуїтивно зрозумілий інтерфейс і дозволяють розробникам працювати з базою даних візуально, що значно спрощує процес розробки і підтримки бази даних. MySQL також має велику кількість документації та навчальних матеріалів, доступних онлайн. Це означає, що розробники можуть легко знайти відповіді на свої запитання та вирішити проблеми, з якими вони стикаються під час роботи з MySQL.

Однією з ключових переваг MySQL є її висока швидкість обробки запитів. Це забезпечується завдяки низці технологій та алгоритмів, вбудованих в систему управління базами даних. Перше, що варто відзначити, — це оптимізація запитів. MySQL має вбудований оптимізатор запитів, який аналізує структуру запиту та вибирає найефективніший план виконання. Оптимізатор враховує такі фактори, як наявність індексів, розмір таблиці, розподілення даних, та інше, для того, щоб забезпечити максимальну швидкість виконання запиту. Вона використовує ефективні алгоритми для збереження та отримання даних. Наприклад, індексація дозволяє MySQL швидко знаходити потрібні дані, без необхідності просматривати всю таблицю. Також, MySQL використовує буферний пул для зберігання найбільш часто використовуваних даних в

оперативній пам'яті, що дозволяє швидко отримувати ці дані без звернення до диска, має ряд вбудованих механізмів для підтримки параллельної обробки запитів. Механізм многопоточності дозволяє MySQL виконувати декілька запитів одночасно, використовуючи ресурси системи максимально ефективно. Також, MySQL підтримує реплікацію, яка дозволяє розподілити обробку запитів між декількома серверами.

Одним з важливих аспектів при виборі СУБД для будь-якого проекту є безпека даних. MySQL забезпечує високий рівень безпеки за допомогою кількох механізмів і функцій. Перший з цих механізмів — це система авторизації MySQL, яка базується на іменах користувачів і паролях. При встановленні нового з'єднання з сервером MySQL, система перевіряє ім'я користувача і пароль, а також визначає, з якого хоста було ініційовано з'єднання. Це забезпечує контроль над тим, хто може отримати доступ до даних. MySQL також підтримує рівні привілеїв для користувачів, що дозволяє адміністраторам бази даних точно контролювати, які дії можуть виконувати різні користувачі. Наприклад, деяким користувачам може бути дозволено тільки читати дані, тоді як інші можуть мати повний доступ до виконання всіх операцій. Ще одна важлива особливість безпеки — це її підтримка шифрування[15]. Вона підтримує шифрування даних на диску, що забезпечує захист від несанкціонованого доступу до фізичного носія. Крім того, вона підтримує шифрування з'єднань, що запобігає перехопленню даних під час передачі між сервером та клієнтом.

Масштабованість і гнучкість є двома важливими атрибутами MySQL, що роблять її однією з найпопулярніших систем управління базами даних у світі. Ці особливості дозволяють MySQL підтримувати як малі проекти, так і великі корпоративні застосування. Щодо масштабованості, MySQL може легко впоратися з великими обсягами даних і великою кількістю запитів. За допомогою реплікації та шардування, MySQL може обробляти терабайти даних і мільйони запитів в секунду. Реплікація дозволяє копіювати і синхронізувати

дані між кількома серверами, що забезпечує високу доступність і стійкість до відмов. Шардування, з свого боку, дозволяє розподілити дані між кількома серверами, що знижує навантаження на кожний індивідуальний сервер і підвищує загальну продуктивність. З іншого боку, гнучкість MySQL виявляється у її спроможності підтримувати різні типи даних, включаючи числові, текстові, дати/час, бінарні та інші. Вона також дозволяє використовувати різноманітні SQL-запити, включаючи вибірку, вставку, оновлення, видалення, об'єднання та інші. MySQL підтримує різні стандарти SQL, що дозволяє її використовувати в різних сценаріях. В цілому, масштабованість і гнучкість MySQL роблять її ідеальним вибором для системи автоматичного сповіщення неактивних клієнтів, оскільки вона забезпечує необхідний баланс між продуктивністю, надійністю, гнучкістю та витратами.

2.3.2 Обмеження MySQL

Хоча MySQL підтримує збережені процедури, функції та тригери, вона не надає такої глибокої підтримки процедурного програмування, як деякі інші системи управління базами даних, такі як PostgreSQL або Oracle. Процедурне програмування дозволяє виконувати складні обчислювальні завдання прямо на рівні бази даних, що може покращити продуктивність додатків, які обробляють великі обсяги даних. У MySQL, хоча і є можливість створювати збережені процедури та функції, вони мають деякі обмеження. Наприклад, MySQL не підтримує пакети, які групують відповідні процедури та функції разом, як це робить Oracle. Крім того, мова процедурного програмування в MySQL, яка відома як SQL/PSM (Persistent Stored Module), не має такої різноманітності контрольних структур, як у PL/pgSQL в PostgreSQL або PL/SQL в Oracle. Це може ускладнити написання та оптимізацію складних збережених процедур в MySQL і може призвести до підвищення навантаження на сервер застосунків, оскільки більше обчислювальної роботи потребує виконання на цьому рівні.

Незважаючи на ці обмеження, MySQL все ще здатна задовольнити потреби багатьох застосунків, особливо тих, які не вимагають високого рівня процедурного програмування. Наприклад, в контексті системи автоматичного сповіщення неактивних клієнтів, більшість операцій з даними можуть бути виконані за допомогою стандартних SQL-запитів, і тому можуть не потребувати використання складних збережених процедур[15].

Одним з обмежень MySQL є відносно повільний процес відновлення даних після аварій. Це може стати серйозною проблемою для високодоступних систем, які потребують мінімізації часу простою. MySQL використовує журнал передачі (transaction log), який записує всі транзакції, що змінюють дані. У випадку системної помилки або відмови, MySQL використовує цей журнал для відновлення консистентності бази даних. Проте, цей процес може бути часоємким, особливо для великих баз даних, оскільки він вимагає повторного виконання всіх транзакцій з журналу передачі. Окрім того, MySQL може потребувати додаткового часу для перебудови індексів після відновлення даних. Це також може збільшити загальний час відновлення. Все це означає, що час простою системи MySQL після відмови може бути вищим, ніж у деяких інших СУБД, таких як PostgreSQL або Oracle, які можуть надати більш швидкі та ефективні механізми відновлення. Проте, варто зазначити, що існують стратегії та технології для зменшення часу відновлення MySQL, такі як налаштування реплікації, використання відмінних дисків для даних та журналів передачі та регулярне створення резервних копій. У контексті системи автоматичного сповіщення неактивних клієнтів, ці стратегії можуть допомогти забезпечити високий рівень доступності та надійності.

Незважаючи на це, MySQL все ще може бути ефективним вибором для багатьох застосунків, включаючи систему автоматичного сповіщення неактивних клієнтів.

.

ВИСНОВОК ДО РОЗДІЛУ 2

В ході дослідження розділу, ми глибоко вивчили основні компоненти, які становлять базу для розробки високоякісних програмних продуктів: Java, Java Swing, C# та MySQL. Кожен з цих інструментів має свої унікальні переваги, а також особливості, що мають бути враховані під час використання.

Обидві мови програмування, C# та Java, володіють різними перевагами та особливостями, що відрізняють їх одну від одної, але обидві демонструють сильні сторони як універсальні, високорівневі мови програмування.

Java є переносною мовою, виконуючою принцип "написати одного разу, запустити скрізь", завдяки машині JVM, на якій вона працює. Це робить Java привабливою для розробки великомасштабних підприємницьких застосунків та для ситуацій, коли потрібна висока переносимість коду.

C#, з другого боку, був створений Microsoft і є ключовою мовою для платформи .NET. Він демонструє сильні сторони при розробці застосунків Windows, веб-застосунків за допомогою ASP.NET і ігор за допомогою Unity. C# має гнучкий синтаксис, що заснований на C++, але більш безпечний та легший для вивчення.

Як C#, так і Java підтримують об'єктно-орієнтоване програмування, що допомагає створювати модульні та масштабовані програми. ООП включає принципи інкапсуляції, наслідування, поліморфізму та абстракції, які сприяють чистоті коду, повторному використанню та гнучкості. Обидві мови включають автоматичне керування пам'яттю через механізми збирачів сміття, зменшуючи тим самим можливість помилок управління пам'яттю. Також вони обидві підтримують обробку виключень, що є важливим механізмом для забезпечення стабільності програми.

Java Swing, з іншого боку, є потужним інструментом для створення графічного інтерфейсу користувача в Java. Його сильні сторони включають платформонезалежність, велику кількість готових компонентів GUI, а також

можливість налаштування та розширення. Його недоліки включають складність вивчення та використання.

MySQL відома своєю надійністю, швидкістю та гнучкістю. Вона є однією з найпопулярніших систем керування базами даних в світі та підтримує різні моделі даних, включаючи реляційні та об'єктно-орієнтовані. Втім, вона може мати свої виклики в плані масштабування.

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

В цьому розділі ми детально розглянемо структуру та функціонування системи сповіщень. Основою системи є п'ять ключових класів: *Main*, *DatabaseConnector*, *ClientsToNotify*, *EmailSender* та *SmsSender*. Кожен виконує свої унікальні ролі в процесі відстеження, вибору клієнтів для сповіщень, а також відправлення відповідних повідомлень через електронну пошту або SMS.

3.1 Клас Client

Клас *Client* представляє модель клієнта у вашій системі. Він містить декілька полів, кожне з яких відображає конкретну властивість клієнта. Ось детальний огляд цих полів:

- Id — це унікальний ідентифікатор, що використовується для визначення конкретного клієнта. Він автоматично генерується базою даних.
- name та lastName — ці поля містять ім'я та прізвище клієнта відповідно.
- email та number — це контактна інформація клієнта, включаючи електронну адресу та номер телефону.
- emailPref та numberPref — ці поля вказують на уподобання клієнта щодо способу зв'язку (через електронну пошту чи SMS).
- Birth — це дата народження клієнта. Вона може бути використана для вітання клієнта у його день народження.
- lastOnline — ця дата вказує, коли клієнт востаннє був онлайн.
- lastPurchase — це дата останньої покупки клієнта.
- lastNotified — це дата останнього повідомлення, відправленого клієнту.

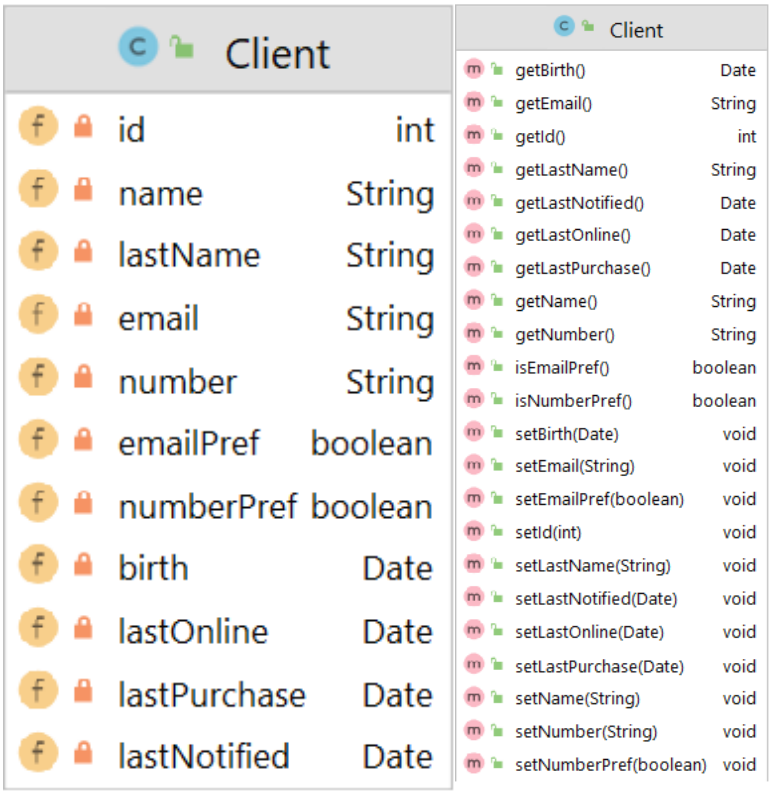


Рисунок 3.1 – UML діаграма класу *Client*

Клас містить методи (*getter* та *setter*) для кожного з цих полів, що дозволяє отримати та змінити їх значення відповідно, використовуються для отримання значень цих полів та для їх зміни.

3.2 Клас Main

Клас *Main* є основним класом програми, який ініціює з'єднання з базою даних, встановлює взаємодію з користувачем та керує процесом надсилання повідомлень.

На початку виконання програми ініціалізуються об'єкти *DatabaseConnector*, *SmsSender* та *EmailSender*, які використовуються для взаємодії з базою даних та для відправки повідомлень.

Через метод *createAndShowGUI* створюється GUI, який включає в себе різні кнопки, текстові поля, мітки та інші компоненти Swing. За допомогою цього інтерфейсу користувач може взаємодіяти з додатком.

Користувач вводить інформацію для підключення до бази даних (хост, порт, назву бази даних, ім'я користувача, пароль) і натискає кнопку *Connect/Check* для встановлення з'єднання, також може ввести назву таблиці і натиснути кнопку *Update table view*, щоб побачити поточні дані з цієї таблиці.

Користувач може ввести різні параметри (кількість днів онлайн, кількість днів сповіщення, кількість днів покупки) і натиснути кнопку *Start* для початку виконання програми. Це запускає таймер, який регулярно сповіщає клієнтів.

Коли таймер спрацьовує, програма отримує список клієнтів, які потребують сповіщення, з бази даних і відправляє їм відповідні повідомлення через електронну пошту або SMS.

Користувач може в будь-який момент натиснути кнопку *Stop* для зупинки виконання програми.

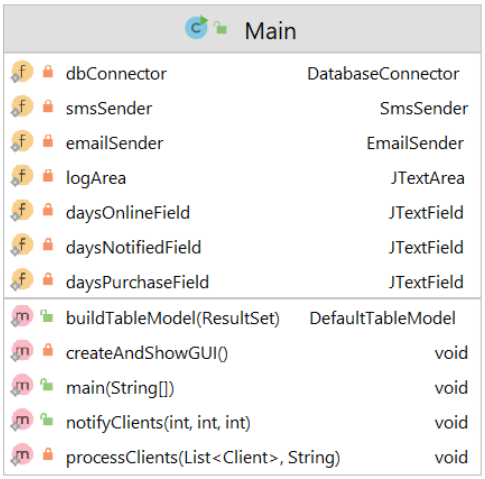


Рисунок 3.2 – UML діаграма класу *Main.*

Загалом, клас Main виконує роль контролера в моделі MVC, взаємодіючи як з моделлю (база даних та клієнти), так і з представленням (графічний інтерфейс).

- dbConnector*:* об'єкт для з'єднання з базою даних та отримання інформації про клієнтів.
- smsSender та *emailSender*: об'єкти для надсилання SMS та електронних листів відповідно.

- logArea: область тексту для відображення журналу процесу надсилання повідомлень.
- *daysOnlineField*, *daysNotifiedField*, *daysPurchaseField*: поля вводу для встановлення параметрів вибору клієнтів.

notifyClients викликає getClientsToNotify метод з DatabaseConnector, передаючи днів daysOnline, daysNotified і daysPurchase як параметри. Метод повертає об'єкт ClientsToNotify, який містить три списки клієнтів: neverPurchased, noRecentPurchase і birthdayClients. За допомогою методу processClients він обробляє кожен список клієнтів, передаючи відповідний тип повідомлення: "noPurchase" для клієнтів, які ніколи не здійснювали покупок; "noRecent" для клієнтів, які не здійснювали покупок в останній час; "birthday" для клієнтів, яким сьогодні день народження.

```
public static void notifyClients(int daysOnline, int daysNotified, int daysPurchase) {
    ClientsToNotify clientsToNotify = dbConnector.getClientsToNotify(daysOnline, daysNotified, daysPurchase);

    processClients(clientsToNotify.getNeverPurchased(), messageType: "noPurchase");
    processClients(clientsToNotify.getNoRecentPurchase(), messageType: "noRecent");
    processClients(clientsToNotify.getBirthdayClients(), messageType: "birthday");
}
```

Рисунок 3.3 – Реалізація методу *notifyClients*.

Метод *processClients* проходить по списку клієнтів та, в залежності від їхніх налаштувань, надсилає їм повідомлення через електронну пошту або SMS. Повідомлення визначається типом *messageType*.

В процесі обробки кожного клієнта метод формує рядок для журналу з інформацією про ідентифікатор клієнта, його ім'я та адресу електронної пошти або номер телефону, на які було надіслано повідомлення. Цей рядок додається до *logArea*.

Якщо повідомлення було успішно надіслано, метод *notifyClient* з *DatabaseConnector* викликається, щоб оновити інформацію про дату останнього сповіщення для відповідного клієнта в базі даних.

Загалом, ці два методи співпрацюють для отримання списків клієнтів для сповіщення з бази даних та виконання відповідних дій сповіщення для кожного з них.

```
private static void processClients(List<Client> clients, String messageType) {
    for (Client client : clients) {
        boolean notificationSent = false;
        if (client.isEmailPref()) {
            emailSender.sendEmail(client.getName(), client.getEmail(), messageType);
            String id = String.format("%-5s", client.getId()).replace( oldChar: ' ', newChar: '.');
            String name = String.format("%-10s", client.getName()).replace( oldChar: ' ', newChar: '.');
            String email = String.format("%-20s", client.getEmail()).replace( oldChar: ' ', newChar: '.');
            logArea.append(String.format("Customer id: %s Email sent to: %s e-mail: %s\n", id, name, email));
            notificationSent = true;
        }
        if (client.isNumberPref()) {
            smsSender.sendSms(client.getName(), client.getNumber(), messageType);
            String id = String.format("%-5s", client.getId()).replace( oldChar: ' ', newChar: '.');
            String name = String.format("%-10s", client.getName()).replace( oldChar: ' ', newChar: '.');
            String number = String.format("%-20s", client.getNumber()).replace( oldChar: ' ', newChar: '.');
            logArea.append(String.format("Customer id: %s Sms sent to:   %s number: %s\n", id, name, number));

            notificationSent = true;
        }
        if (notificationSent) {
            dbConnector.notifyClient(client);
        }
    }
}
```

Рисунок 3.4 – Реалізація методу *processClients*.

Метод *createAndShowGUI* створює GUI за допомогою Swing. Він створює ряд об'єктів Swing, таких як рамки, панелі, поля введення, кнопки та інші віджети, влаштовує їх на екрані, налаштовує слухачі подій для обробки дій користувача (наприклад, натискання кнопки) та робить інтерфейс видимим.

Він також ініціює об'єкти *DatabaseConnector*, *SmsSender*, *EmailSender*, встановлює властивості різних елементів GUI (наприклад, розмір, колір, шрифт), додає *ActionListener* до різних кнопок для обробки подій та настроює *Timer* для автоматичного сповіщення клієнтів за визначеним інтервалом часу.

Цей метод служить основою для цього додатку Swing, оскільки він відповідає за встановлення взаємодії користувача та ініціювання основних об'єктів програми, він також містить декілька *ActionListener*:

- Слухач подій *updateTableButton ActionListener* зв'язаний з кнопкою "*Update table view*". Коли користувач натискає цю кнопку, викликається подія, яка отримує дані з таблиці бази даних, назва якої введена у поле *tableNameField*. Вона потім використовує метод *buildTableModel* для створення моделі таблиці з цих даних та оновлює таблицю в графічному інтерфейсі користувача.
- Слухач подій *startButton ActionListener* виконує функіонал кнопки "*Start*". Він перевіряє статус запуску: якщо програма вже запущена, він зупиняє таймер, оновлює інтервал таймера та змінює текст кнопки на "*Start*". Якщо програма не запущена, він перевіряє з'єднання з базою даних, викликає метод *notifyClients*, налаштовує інтервал таймера, запускає таймер і змінює текст кнопки на "*Stop*".
- Слухач *connectButton ActionListener* зв'язаний з кнопкою "*Connect/Check*". Коли користувач натискає цю кнопку, слухач отримує дані, введені в різні текстові поля, і використовує ці дані для створення нового об'єкта *DatabaseConnector*. Він потім перевіряє з'єднання з базою даних і відображає повідомлення, яке інформує користувача про статус з'єднання.

Метод *buildTableModel* отримує результат запиту до бази даних у вигляді *ResultSet* і перетворює його у модель таблиці *DefaultTableModel*, яку можна використовувати для відображення даних у Swing JTable.

Спочатку метод отримує метадані *ResultSet* (що включає інформацію про стовпці), а потім отримує імена всіх стовпців і додає їх до вектора. Він створює вектор для кожного рядка даних, додаючи об'єкти відповідних стовпців, а потім додає цей вектор в вектор даних. Потім він повертає новий *DefaultTableModel* із цими даними та іменами стовпців.

```
public static DefaultTableModel buildTableModel(ResultSet rs) throws SQLException {
    ResultSetMetaData metaData = rs.getMetaData();

    Vector<String> columnNames = new Vector<>();
    int columnCount = metaData.getColumnCount();
    for (int column = 1; column <= columnCount; column++) {
        columnNames.add(metaData.getColumnName(column));
    }

    Vector<Vector<Object>> data = new Vector<>();
    while (rs.next()) {
        Vector<Object> vector = new Vector<>();
        for (int columnIndex = 1; columnIndex <= columnCount; columnIndex++) {
            vector.add(rs.getObject(columnIndex));
        }
        data.add(vector);
    }

    return new DefaultTableModel(data, columnNames);
}
```

Рисунок 3.5 – Реалізація методу *buildTableModel.*

Метод *main* — стартовий метод програми, який викликає *createAndShowGUI* для створення та відображення графічного інтерфейсу.

3.3 Клас DatabaseConnector

Клас *DatabaseConnector* відповідає за взаємодію з базою даних MySQL. Він містить в собі методи для створення з'єднання з базою даних, виконання SQL-запитів, отримання даних про клієнтів та оновлення стану сповіщення клієнтів.

Коли створюється новий об'єкт *DatabaseConnector*, конструктор класу встановлює з'єднання з базою даних, використовуючи передані параметри.

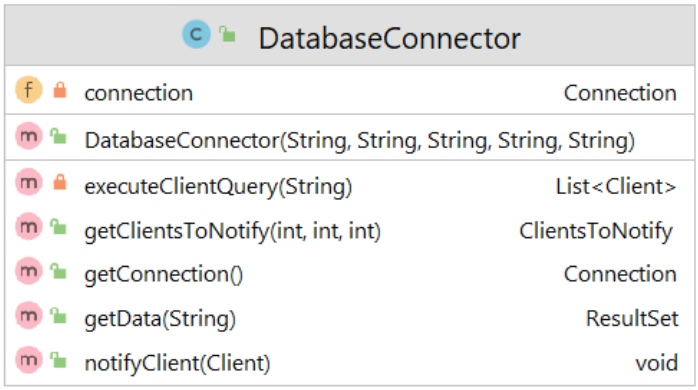


Рисунок 3.6 – UML діаграма класу *DatabaseConnector*.

Метод *getConnection* просто повертає поточне з'єднання з базою даних. З'єднання ініціалізується при створенні об'єкту *DatabaseConnector* та потім може бути використано в інших частинах програми для прямої взаємодії з базою даних.

getClientsToNotify використовує параметри для виконання трьох SQL запитів до бази даних, щоб знайти клієнтів, які потребують сповіщення.

Перший запит, *queryNeverPurchased*, знаходить клієнтів, які ніколи не здійснили покупку, були онлайн в останні *daysOnline* днів, і не отримували сповіщення протягом *daysNotified* днів.

Другий запит, *queryNoRecentPurchase*, знаходить клієнтів, які не здійснювали покупки протягом *daysPurchase* днів, були онлайн в останні *daysOnline* днів, і не отримували сповіщення протягом *daysNotified* днів.

Третій запит, *queryBirthday*, знаходить клієнтів, у яких сьогодні день народження і які не отримували сповіщення сьогодні.

Кожен з цих запитів виконується за допомогою методу *executeClientQuery*(), який повертає список клієнтів. Результати цих трьох запитів об'єднуються в об'єкт *ClientsToNotify* і повертаються.

```
public ClientsToNotify getClientsToNotify(int daysOnline, int daysNotified, int daysPurchase) {
    String queryNeverPurchased = "SELECT * FROM clients " +
            "LEFT JOIN purchases ON clients.id = purchases.client_id " +
            "WHERE purchases.client_id IS NULL " +
            "AND (DATE_ADD(clients.last_online, INTERVAL " + daysOnline + " DAY) <= CURDATE() " +
            "AND (DATE_ADD(clients.last_notified, INTERVAL " + daysNotified +
            " DAY) <= CURDATE() OR clients.last_notified IS NULL))";

    String queryNoRecentPurchase = "SELECT * FROM clients " +
            "WHERE DATE_ADD(last_purchase, INTERVAL " + daysPurchase + " DAY) <= CURDATE() " +
            "AND (DATE_ADD(last_online, INTERVAL " + daysOnline + " DAY) <= CURDATE() " +
            "AND (DATE_ADD(last_notified, INTERVAL " + daysNotified + " DAY) <= CURDATE() OR last_notified IS NULL))";

    String queryBirthday = "SELECT * FROM clients " +
            "WHERE DAY(birth) = DAY(CURDATE()) AND MONTH(birth) = MONTH(CURDATE()) " +
            "AND (DATE(last_notified) != CURDATE() OR last_notified IS NULL)";

    List<Client> neverPurchased = executeClientQuery(queryNeverPurchased);
    List<Client> noRecentPurchase = executeClientQuery(queryNoRecentPurchase);
    List<Client> birthdayClients = executeClientQuery(queryBirthday);

    return new ClientsToNotify(neverPurchased, noRecentPurchase, birthdayClients);
}
```

Рисунок 3.7 – Реалізація методу *getClientsToNotify*.

executeClientQuery виконує SQL-запит, який передається йому як рядковий параметр *query*. Запит використовується для отримання даних про клієнтів з бази даних. Результатом запиту є набір результатів *ResultSet*, який містить дані про клієнтів.

Метод створює список об'єктів *Client* і заповнює його, перебираючи набір результатів. Він читає значення кожного поля для кожного клієнта і використовує ці значення для створення нового об'єкта *Client*, який додається до списку.

Метод також обробляє *SQLException*, який може виникнути під час виконання SQL-запиту або обробки результатів.

notifyClient використовується для оновлення дати останнього сповіщення клієнта в базі даних. Він формує SQL-запит *UPDATE*, який оновлює поле *last_notified* для вказаного клієнта, встановлюючи його значення на сьогоднішню дату (*CURDATE()*). Використовує *PreparedStatement* для виконання запиту, що дозволяє безпечно передавати параметри до запиту. Він встановлює ID клієнта в запит, використовуючи метод *setInt*.

Як і інші методи в цьому класі обробляє *SQLException.*

Метод *getData* використовується для отримання всіх даних з вказаної таблиці в базі даних. Він створює SQL-запит «*SELECT * FROM tableName*», де *tableName* — це ім'я таблиці, яке передається методу як параметр.

Метод виконує запит за допомогою *Statement* і повертає набір результатів *ResultSet*, що містить дані з таблиці.

3.4 Клас ClientsToNotify

Клас *ClientsToNotify* - це простий клас-контейнер, який використовується для групування списків клієнтів, що підходять під певні критерії. Він має три приватні поля типу *List<Client>: neverPurchased, noRecentPurchase* та *birthdayClients*.

- *neverPurchased*: це список клієнтів, які ніколи не здійснювали покупок.
- *noRecentPurchase*: це список клієнтів, які не здійснювали покупок протягом певного періоду часу.
- *birthdayClients*: це список клієнтів, у яких сьогодні день народження.

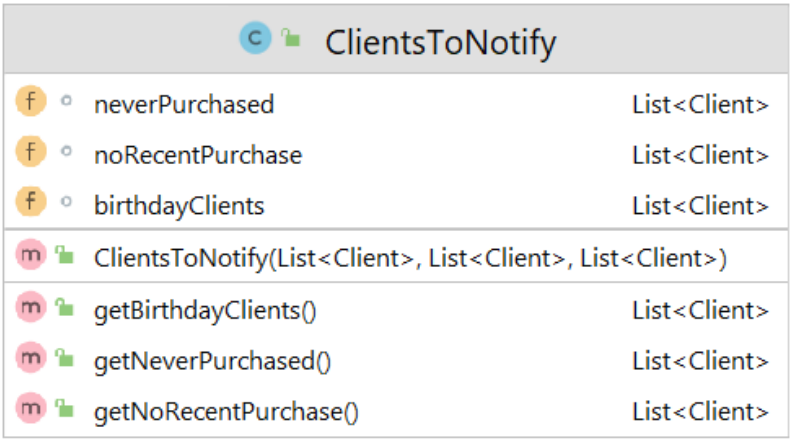


Рисунок 3.8 – UML діаграма класу *ClientsToNotify*.

Конструктор *ClientsToNotify* приймає три параметри - списки клієнтів для кожної з трьох категорій, та присвоює їх відповідним полям класу.

Крім того, в класі ClientsToNotify є три гетери , а саме: *getNeverPurchased*, *getNoRecentPurchase*, *getBirthdayClients*, які дозволяють отримати доступ до приватних полів. Кожен з гетерів повертає відповідний список клієнтів.

Цей клас в основному використовується для зберігання та передачі даних між методами та класами.

3.5 Клас EmailSender

Клас *EmailSender* є компонентом, що використовується для відправлення електронних листів. Він використовує бібліотеку *JavaMail* для роботи з протоколом SMTP, що дозволяє відправляти електронні листи.

В основному, він містить один метод *sendEmail*, який використовується для відправлення електронних листів з різними типами повідомлень.

Цей клас може бути використаний в різних частинах програми, які потребують відправлення електронних листів, наприклад, для повідомлень про події, вітання з днем народження, нагадувань про недавні покупки.

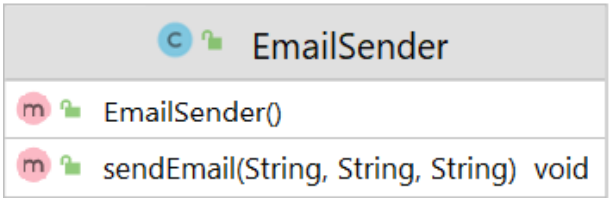


Рисунок 3.8 – UML діаграма класу *ClientsToNotify*.

Цей клас містить один метод sendEmail(String name, String to, String type), який приймає три параметри:

- *name* — ім'я клієнта, якому буде відправлено електронний лист.
- *to* — електронна адреса одержувача.
- *type* — тип повідомлення, яке буде відправлено.

Цей метод використовує бібліотеку *JavaMail* для відправлення електронних листів. Він створює *Session* з відповідними властивостями SMTP, а потім створює *MimeMessage*, що представляє саме електронне повідомлення.

Залежно від типу повідомлення (*type*), встановлюється відповідна тема (*subject*) та текст електронного листа (*htmlText*). На основі типу повідомлення може бути відправлено один із трьох типів листів: повідомлення на день народження, повідомлення для тих, хто давно не здійснював покупок, або повідомлення для тих, хто ще не здійснив жодної покупки.

Після встановлення всіх параметрів для повідомлення, викликається метод *Transport.send(message)*, який відправляє це повідомлення.

3.6 Клас SmsSender

Клас *SmsSender* є частиною системи сповіщення, розробленої для спілкування з клієнтами через SMS-повідомлення. Він створений для відправлення текстових повідомлень клієнтам у різних ситуаціях: на день народження клієнта, коли клієнт давно не робив покупок або ще ніколи не робив покупок.

Для роботи цей клас використовує зовнішній сервіс для відправлення SMS — *Telesign*.

У класі *SmsSender* використовуються три важливих реквізити: *customerId*, *apiKey* та *messageType*, які використовуються для авторизації та визначення типу повідомлення, що відправляється через *Telesign*.

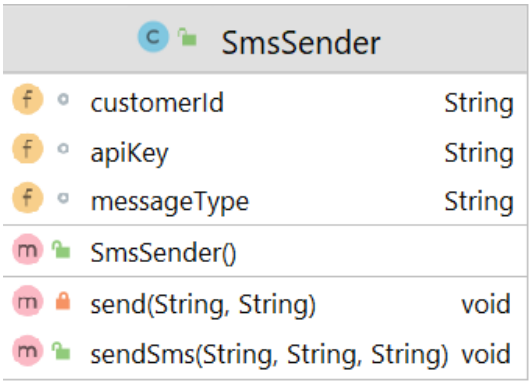


Рисунок 3.9 – UML діаграма класу *SmsSender*.

Цей клас містить два основних методи - sendSms та send.

- *sendSms* — метод, що використовується для формування повідомлень в залежності від типу повідомлення ("*birthday*", "*noRecent*", "*noPurchase*") та викликає метод *send* для відправки сформованого повідомлення.
- send є методом для відправлення SMS повідомлень. Він створює об'єкт *MessagingClient* з необхідними реквізитами (*customerId*, *apiKey*), а потім використовує метод *message* цього об'єкта для відправлення повідомлення на вказаний номер телефону.

Таким чином, клас SmsSender є інструментом в системі сповіщень, який відповідає за відправлення SMS повідомлень клієнтам.

ВИСНОВОК ДО РОЗДІЛУ 3

У даному розділі ми ознайомилися з компонентами системи сповіщень що спрямовані на вдосконалення взаємодії з клієнтами та їх втримання. Система є результатом поєднання взаємодії різних класів, кожен з яких виконує свою унікальну роль.

Клас *Main*, як вхідна точка в додатку, контролює взаємодію між класами і координує основні процеси. Він ініціює з'єднання з базою даних, створюючи об'єкт DatabaseConnector, і запускає процеси, що відбуваються у системі.

DatabaseConnector відповідальний за взаємодію з базою даних, реалізовуючи витягування даних за допомогою Java JDBC. Він розумно використовує ресурси, запитуючи в бази даних лише ті дані, які необхідні для наступних етапів виконання.

Роль класу *ClientsToNotify* полягає в структуруванні і зберіганні інформації про клієнтів, які мають отримати сповіщення. Цей клас упорядковує дані та полегшує обробку і передачу інформації, стаючи центром, де зберігаються всі дані про клієнтів для сповіщення.

У свою чергу, класи *EmailSender* та *SmsSender* реалізують безпосереднє спілкування з клієнтами, відправляючи відповідні повідомлення за допомогою електронної пошти та SMS. Вони використовують сторонні API для відправки повідомлень, підтримуючи постійний контакт з кінцевими користувачами.

Сукупність цих класів створює розумну та гнучку систему сповіщень, яка автоматизує процеси комунікації з клієнтами, забезпечуючи їх своєчасністю і ефективністю. Вона здатна адаптуватися до різних потреб бізнесу і забезпечує надійність, необхідну для успішного функціонування онлайн магазину. Кожний клас виконує свою роль, що допомагає зберегти цілісність системи та спрямовує її на досягнення загальної мети.

РОЗДІЛ 4. АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Огляд інтерфейсу програми

В цьому розділі ми детально розглянемо зовнішній вигляд програми, розробленої для автоматизації процесу сповіщення клієнтів.

Програма працює за допомогою використання засобів для роботи з базами даних JDBC та SQL. Це дозволяє їй взаємодіяти з базами даних на рівні програмного забезпечення, отримуючи інформацію про користувачів і їх активність. Ця інформація використовується для визначення, коли і яким клієнтам слід надіслати повідомлення.

Програма має гнучкі налаштування, що дозволяють користувачеві налаштовувати параметри під свої потреби. Зокрема, можна налаштовувати параметри, такі як часові проміжки між повідомленнями, а також дні, які треба враховувати під час визначення активності користувача. Це дозволяє враховувати індивідуальні особливості бізнесу та користувачів.

Інтерфейс програми, реалізований за допомогою Swing, є інтуїтивно зрозумілим та зручним для користувача. Він включає в себе ряд текстових полів та кнопок, які дозволяють користувачеві керувати процесом роботи програми. За допомогою цих елементів користувач може вводити дані для підключення до бази даних, налаштовувати параметри відправки повідомлень та контролювати процес роботи програми.

Опишемо функції кожного компонента, що дозволить нам зрозуміти, як вони взаємодіють між собою, та як вони впливають на загальний функціонал програми. Крім того, ми проаналізуємо зручність використання цих елементів та їх вплив на досвід користувача.

У полях треба ввести наступне:

- Days online — кількість днів, які користувач не був в мережі.
- Days notified — кількість днів, які користувач не був оповіщений.
- Days purchased — кількість днів, які користувач нічого не купував.
- Seconds of cycle — кількість часу, яка необхідна для повторної перевірки.
- Host — назва хосту бази даних, з якою треба працювати.
- Port — номер порту, який необхідний для підключення.
- Username — назва користувача для доступу до бази даних.
- Password — пароль користувача для доступу до бази даних.
- Database — назва бази даних, до якої треба підключитися.

Після введення даних неодхідно провести перевірку з’єднання до бази даних, натиснувши кнопку *"Connect/Check"*. Ця кнопка використовується для встановлення з'єднання з базою даних за введеними параметрами. При успішному з'єднанні відображається повідомлення *"Connection successful!"*. В іншому випадку буде повідомлення, про неуспіше підключення до бази даних як на рисунку 4.1.

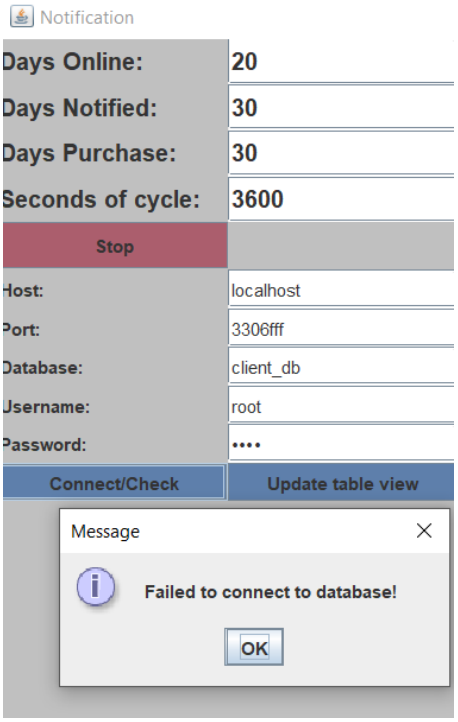


Рисунок 4.1 – Повідомлення про неуспішне підключення до бази даних.

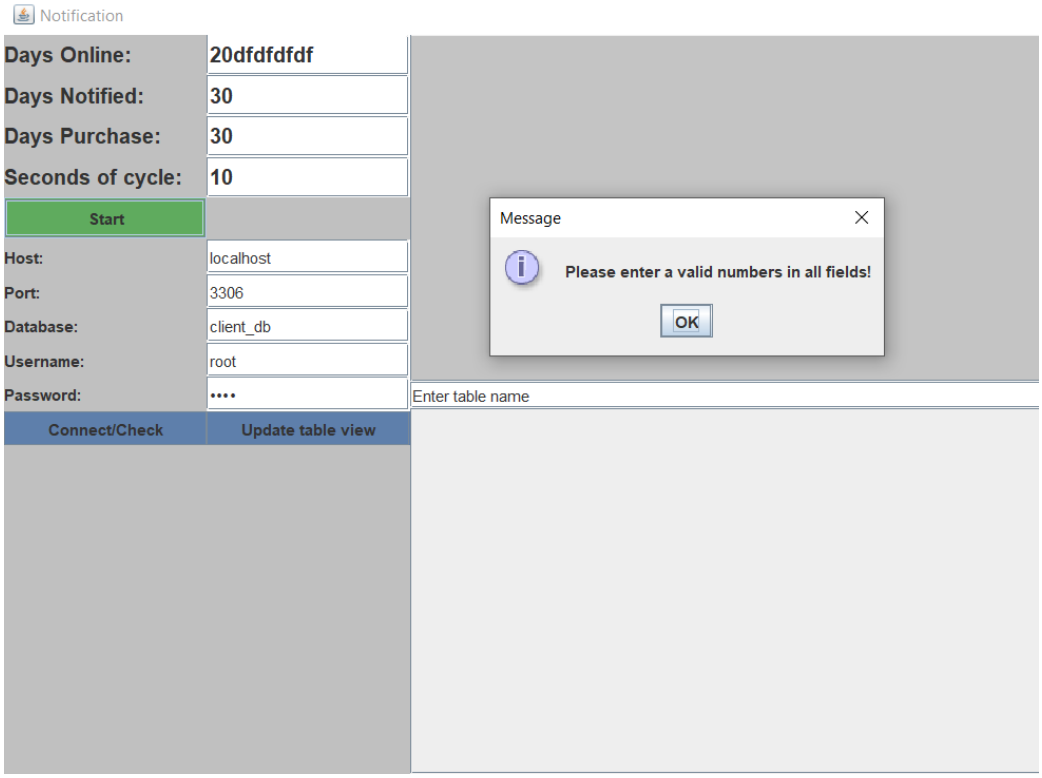


Рисунок 4.2 – Повідомлення про некоректність введених даних.

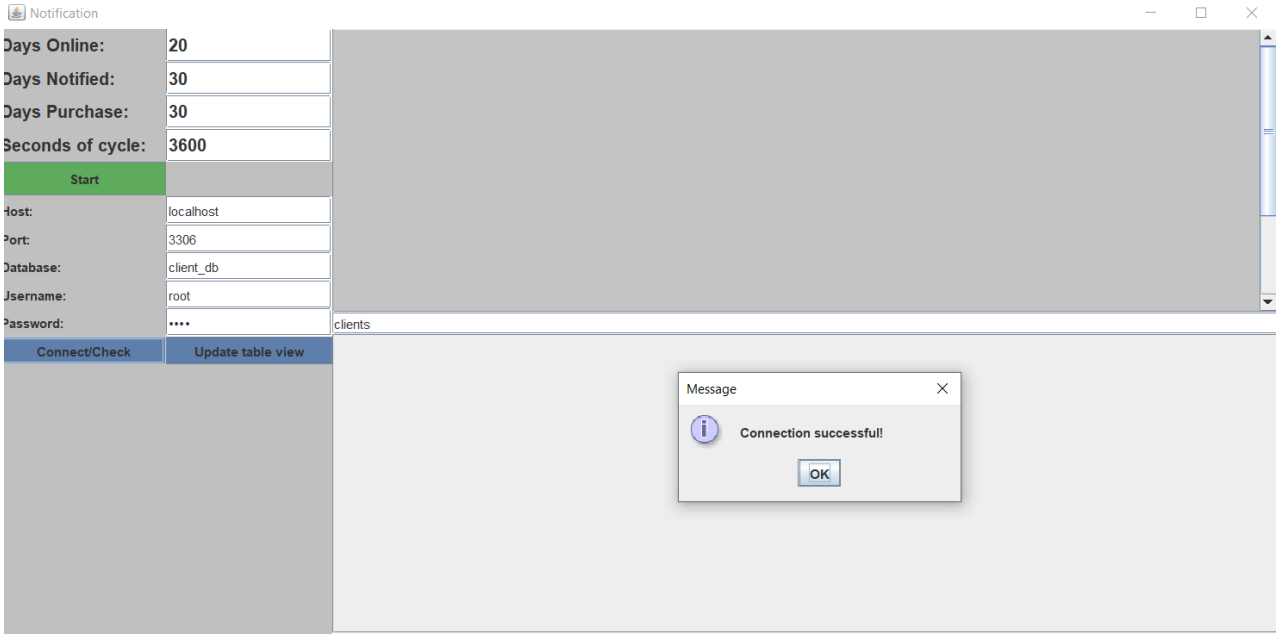


Рисунок 4.3 – Повідомлення успішне підключення до бази даних.

Тепер можна безпосередньо запускати програму. кнопка *"Start"* використовується для запуску таймера для автоматичного сповіщення. Якщо були введені невірні дані в полях: Days online, Days notified, Days purchased,

Seconds of cycle, то користувача буде повідомлено про це як на рисунку 4.2. Після запуску текст кнопки змінюється на "Stop", що дозволяє зупинити таймер.

Можна переглянути логи програми в текстовому полі зліва. Вони містять інформацію про виконання програми, включаючи успішні або невдалі спроби встановити з'єднання з базою даних, виконані запити до бази даних, та сповіщення, які були відправлені.

4.2 Огляд роботи програми

При першому запуску програми відкривається головне вікно інтерфейсу, що містить текстові поля та кнопки для введення інформації про базу даних і параметрів роботи. Після введення цих даних було натиснуто кнопку *"Connect/Check"*, що здійснило підключення до бази даних. Після успішного підключення, програма надала зворотний зв'язок про успішне підключення, надіславши відповідне повідомлення. Далі були введені параметри для відправки повідомлень.

Ці параметри вказують, що програма відправлятиме повідомлення користувачам, які були онлайн протягом останніх 20 днів, але не отримували повідомлень протягом останніх 30 днів і не здійснювали покупок протягом останніх 30 днів.

Натискаючи кнопку *"Start"*, програма запускає свою основну роботу. Вона періодично перевіряє базу даних, визначає користувачів, які відповідають вказаним критеріям, і відправляє їм повідомлення за допомогою SMS і електронної пошти.

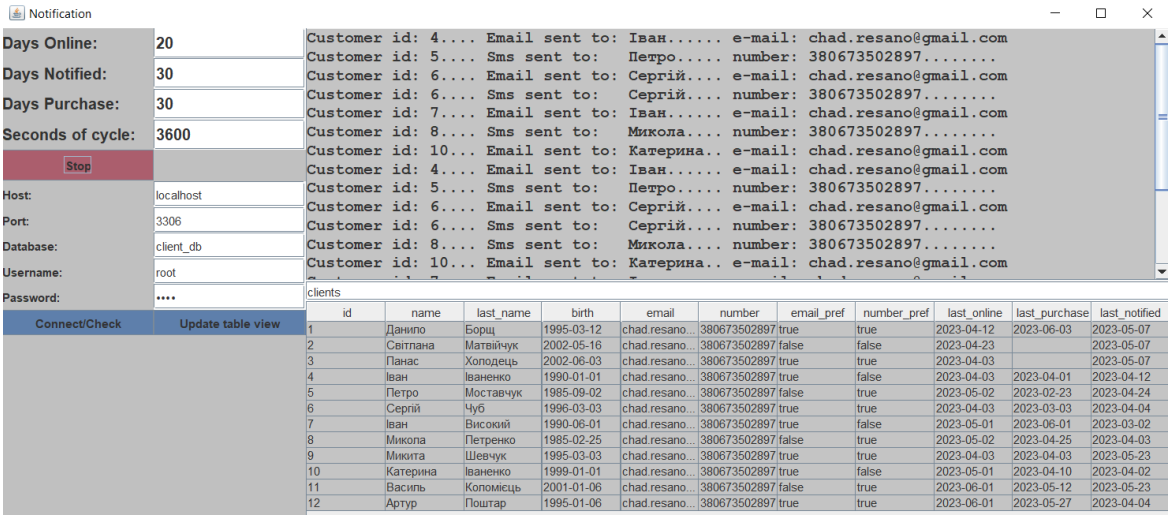


Рисунок 4.4 – Звітування програми про відправлення сповіщень.

Протягом роботи програми можна в будь-який момент переглянути стан бази даних, натиснувши кнопку *"Update table view"*.

clients

id	name	last_name	birth	email	number	email_pref	number_pref	last_online	last_purchase	last_notified
1	Данило	Борщ	1995-03-12	chad.resano...	380673502897	true	true	2023-04-12	2023-06-03	2023-05-07
2	Світлана	Матвійчук	2002-05-16	chad.resano...	380673502897	false	false	2023-04-23		2023-05-07
3	Панас	Холодець	2002-06-03	chad.resano...	380673502897	true	true	2023-04-03		2023-05-07
4	Іван	Іваненко	1990-01-01	chad.resano...	380673502897	true	false	2023-04-03	2023-04-01	2023-06-01
5	Петро	Моставчук	1985-09-02	chad.resano...	380673502897	false	true	2023-05-02	2023-02-23	2023-06-01
6	Сергій	Чуб	1996-03-03	chad.resano...	380673502897	true	true	2023-04-03	2023-03-03	2023-06-01
7	Іван	Високий	1990-06-01	chad.resano...	380673502897	true	false	2023-05-01	2023-06-01	2023-06-01
8	Микола	Петренко	1985-02-25	chad.resano...	380673502897	false	true	2023-05-02	2023-04-25	2023-06-01
9	Микита	Шевчук	1995-03-03	chad.resano...	380673502897	true	true	2023-04-03	2023-04-03	2023-05-23
10	Катерина	Іваненко	1999-01-01	chad.resano...	380673502897	true	false	2023-05-01	2023-04-10	2023-06-01
11	Василь	Коломієць	2001-01-06	chad.resano...	380673502897	false	true	2023-06-01	2023-05-12	2023-05-23
12	Артур	Поштар	1995-01-06	chad.resano...	380673502897	true	true	2023-06-01	2023-05-27	2023-04-04

Рисунок 4.5 – Зміна стану таблиці в колонці *last_notified*.

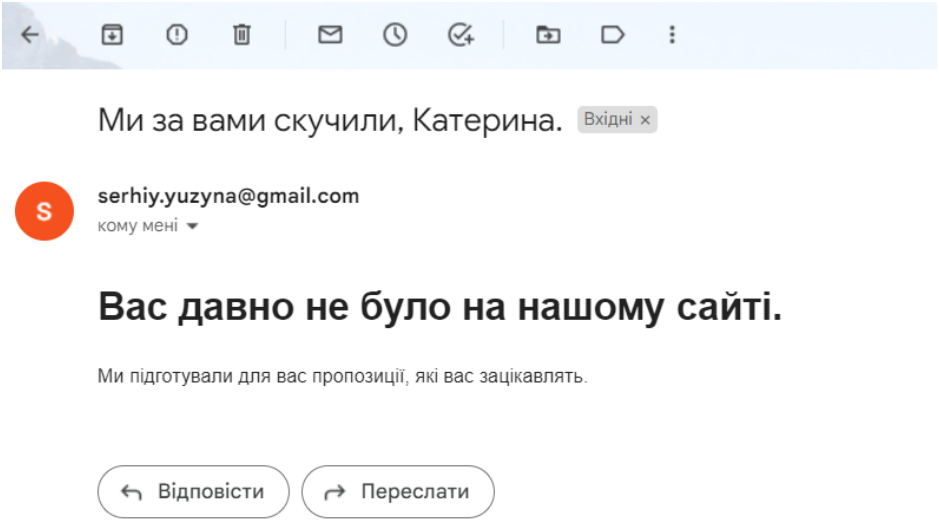


Рисунок 4.5 –Електронний лист, що надійшов користувачу.

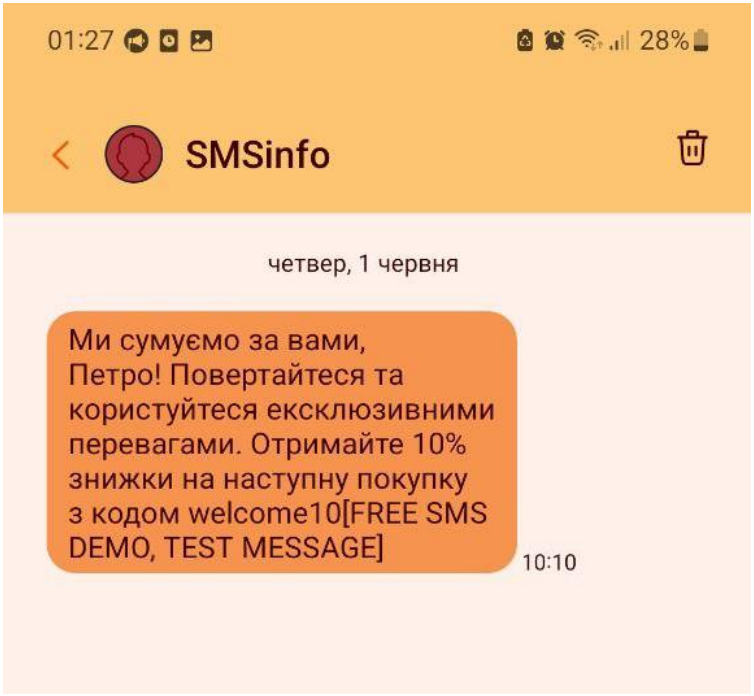


Рисунок 4.6 –SMS-повідомлення, що надійшло користувачу.

Для тестування було використано одну окрему електронну пошту та номер телефону, однак програма може відправити сповіщення на іншу адресу. Програма працює стабільно, виконуючи всі свої функції без помилок. Вона успішно взаємодіє з базою даних, правильно ідентифікує користувачів, які відповідають визначеним критеріям, і відправляє їм повідомлення. Отже, можна з упевненістю стверджувати, що програма відповідає всім поставленим вимогам і ефективно виконує свої задачі.

4.3 Рекомендації щодо вдосконалення додатка

Хоча програма вже є ефективним інструментом для відправлення сповіщень користувачам, є кілька областей, де вона може бути поліпшена.

Нині програма використовує загальні налаштування для визначення, коли слід надсилати сповіщення. Проте користувачі можуть мати різні потреби, тому можливість налаштовувати сповіщення для кожного користувача окремо було

б вельми корисною. Також було б непогано розширити список підтримуваних форматів сповіщень, включаючи push-сповіщення, сповіщення в соціальних мережах та інші.

Поточний інтерфейс є функціональним, його можна зробити більш привабливим та інтуїтивно зрозумілим. Використання іконок, вдосконалених віджетів введення даних, а також інтеграція інтерфейсу з темою операційної системи користувача можуть значно покращити користувацький досвід.

Додати можливість відстежувати, як користувачі реагують на сповіщення (наприклад, відкривають листи, переходять за посилання тощо), може бути важливим інструментом для покращення ефективності сповіщень.

Хоча програма вже використовує автоматизацію для відправлення сповіщень, існують інші області, де можна впровадити автоматизацію. Наприклад, програма може автоматично визначати оптимальний час для відправлення сповіщень на основі поведінки користувача.

Ці поліпшення можуть в значній мірі покращити ефективність програми, збільшуючи задоволеність користувачів та підвищуючи загальну продуктивність.

ВИСНОВОК ДО РОЗДІЛУ 4

Проведений огляд програми демонструє, що цей додаток має ряд важливих особливостей, які спрощують процес надсилання сповіщень користувачам. Він надає простий та ефективний інтерфейс користувача, який включає всі необхідні поля для введення інформації та налаштування параметрів для коректної роботи з базою даних і відправлення сповіщень.

Програма має розгорнуту систему управління, яка дає змогу користувачам контролювати відправлення сповіщень, встановлювати інтервали між циклами та контролювати процес з'єднання з базою даних. Забезпечується також можливість перегляду даних з бази, що сприяє кращому розумінню того, яким чином дані обробляються та використовуються.

Однак, хоча програма вже є ефективним інструментом, є декілька областей, де вона може бути поліпшена. Зокрема, це стосується гнучкості налаштувань, підтримки різних форматів сповіщень, оптимізації користувацького інтерфейсу, відстеження відповідей на сповіщення, покращення заходів захисту даних та автоматизації деяких процесів.

В усьому, моя програма є вдалим прикладом програмного забезпечення, призначеного для автоматизації процесу надсилання сповіщень. З допомогою декількох поліпшень вона може стати ще більш ефективним інструментом для задоволення потреб користувачів.

ВИСНОВОК

Ця дипломна робота була цілеспрямованою та широко розгалуженою дослідницькою роботою, яка охоплювала різні аспекти аналізу, проектування та реалізації програмного забезпечення. Основна мета роботи полягала в розробці та аналізі системи, що є додатком для надсилання сповіщень користувачам. Виконання цієї роботи допомогло зрозуміти ключові аспекти розробки програмного забезпечення, включаючи процеси аналізу, проектування, реалізації та тестування.

Було проведено ретельний огляд предметної області, що мав на меті визначити основні потреби користувачів і специфікації програмного забезпечення. Цей крок був надзвичайно важливим, оскільки він допоміг формувати чітке уявлення про те, як має виглядати та функціонувати програма "Notification". Було визначено ключові вимоги користувачів і застосування, що послужило основою для детального проектування програмного забезпечення.

Пошук методів реалізації став наступним важливим етапом дослідження. В ході цього процесу було досліджено різні технології, інструменти та підходи, які можуть бути використані для розробки програми. Цей етап не тільки допоміг вибрати найефективніші методи реалізації, але й забезпечив глибше розуміння технологій, що стоять за розробкою програмного забезпечення.

Розбір програмного коду був надзвичайно корисним, оскільки він допоміг не лише зрозуміти, як система функціонує в деталях, але й визначити можливі області для поліпшення. Зокрема, виявлено, що хоча додаток вже є ефективним інструментом, існують можливості для оптимізації, в тому числі поліпшення гнучкості налаштувань, розширення підтримки форматів сповіщень та покращення інтерфейсу користувача.

Аналіз зовнішнього вигляду програми, нарешті, дозволив оцінити зручність та ефективність інтерфейсу користувача. Хоча інтерфейс є інтуїтивно

зрозумілим та легким у використанні, було ідентифіковано декілька областей, в яких можна зробити його ще кращим.

В цілому, ця робота надала важливий досвід у розробці програмного забезпечення, а також висвітлила важливість постійного покращення та оптимізації для досягнення найкращих результатів. Кінцева мета досягнута роботи досягнута — розроблено та протестовано програму, яка відповідає потребам користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. How to Reactivate Inactive Customers [Електронний ресурс] – Режим доступу до ресурсу: https://www.actito.com/en-BE/blog/customer-reactivation/
2. SMS Marketing and Texting Statistics – [Електронний ресурс] – Режим доступу до ресурсу: https://www.crazyegg.com/blog/sms-marketing-program/
3. List of Marketing Statistics for 2022 – [Електронний ресурс] – Режим доступу до ресурсу: https://www.hubspot.com/marketing-statistics
4. Email & SMS Spam Words to Avoid When Marketing – [Електронний ресурс] – Режим доступу до ресурсу: https://www.textmagic.com/blog/email-sms-spam-words-to-avoid-business/
5. Effective Java – [Електронний ресурс] – Режим доступу до ресурсу: https://archive.org/details/effectivejava00bloc_0
6. Creating a GUI With Swing – [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/tutorial/uiswing/index.html Brett D.
7. McLaughlin , Gary Pollice , Dave West // Head First Object-Oriented Analysis and Design 1st Edition. 2006. Vol. 636. P. 75—136.
8. Getting a SOLID start – [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start
9. Advantages and Disadvantages of Java – [Електронний ресурс] – Режим доступу до ресурсу: https://techvidvan.com/tutorials/pros-and-cons-of-java/
10. C# documentation – [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/
11. What Is C#: Definitions, Strengths & Usages – [Електронний ресурс] – Режим доступу до ресурсу: https://www.designveloper.com/blog/what-is-c-sharp/
12. Disadvantages of C# – [Електронний ресурс] – Режим доступу до ресурсу: https://www.codeguru.com/csharp/c-sharp-disadvantages/

13. Desktop Guide Windows Forms – [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-7.0
14. MySQL – [Електронний ресурс] – Режим доступу до ресурсу: https://www.oracle.com/mysql/what-is-mysql/
15. The Pros and Cons of MySQL – [Електронний ресурс] – Режим доступу до ресурсу: https://www.smartfile.com/blog/the-pros-and-cons-of-mysql/

ДОДАТОК 1

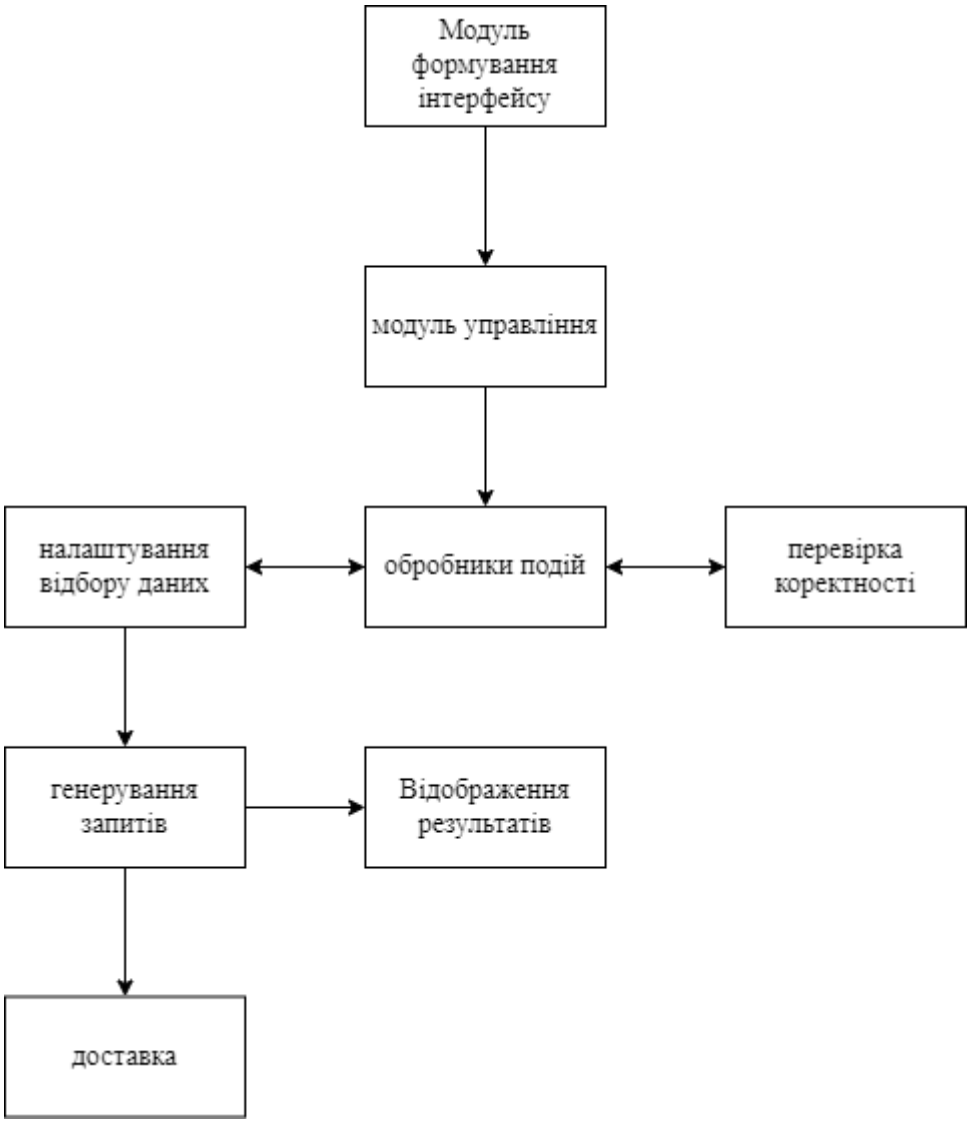
Система автоматично сповіщення неактивних клієнтів

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Юзина С. С.			Система автоматичного сповіщення неактивних клієнтів (Структурна схема системи)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Трочун Є. В.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-92	
Н. Контр.		Виноградов Ю.М					
Затвердив							

ДОДАТОК 2

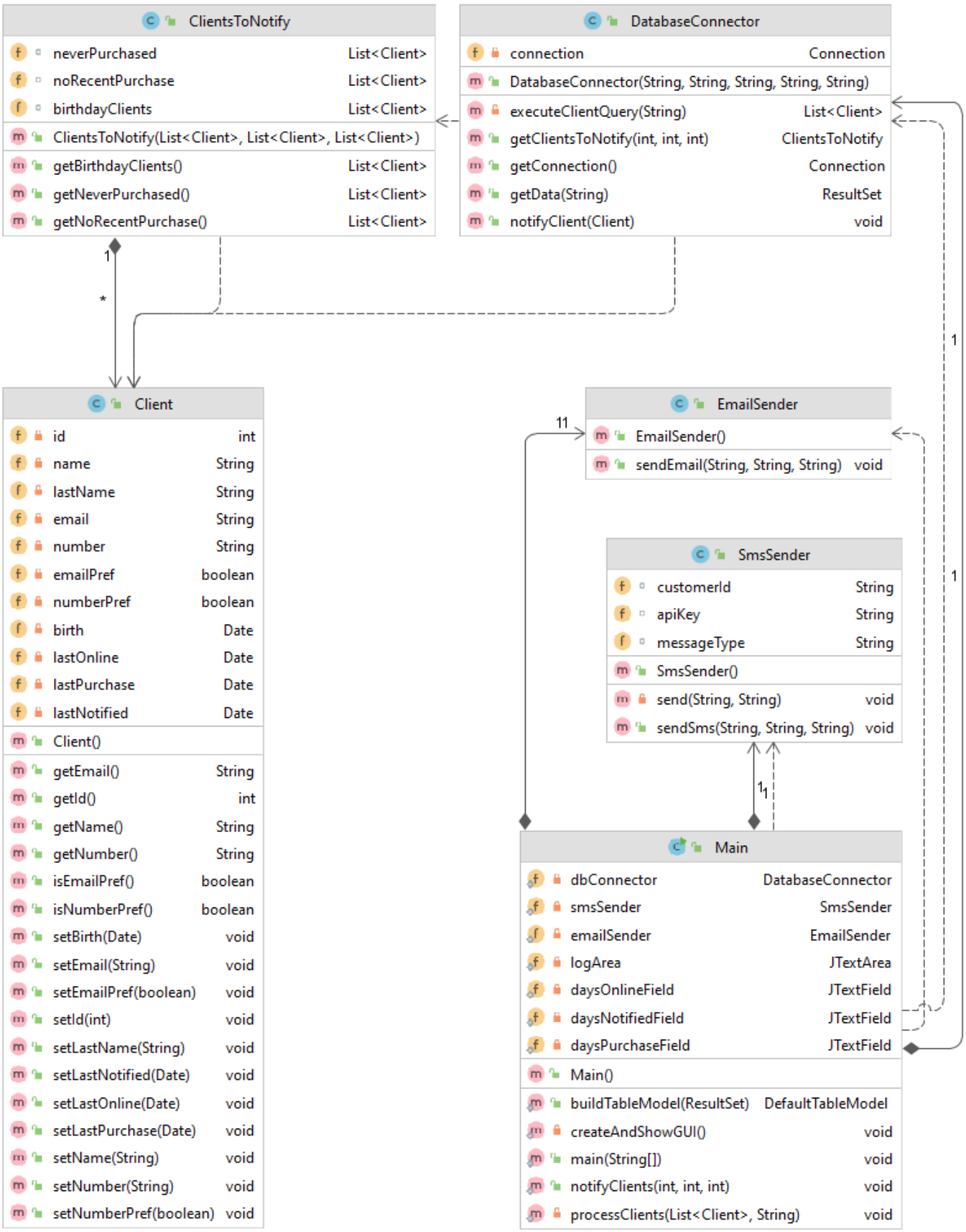
Система автоматично сповіщення неактивних клієнтів

Функціональна схема (діаграма класів)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Юзина С.С.			Система автоматичного сповіщення неактивних клієнтів Функціональна схема (діаграма класів)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Трочун Є.В.				КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-92
Н. Контр.		Виноградов Ю.М				
Затвердив						

ДОДАТОК З

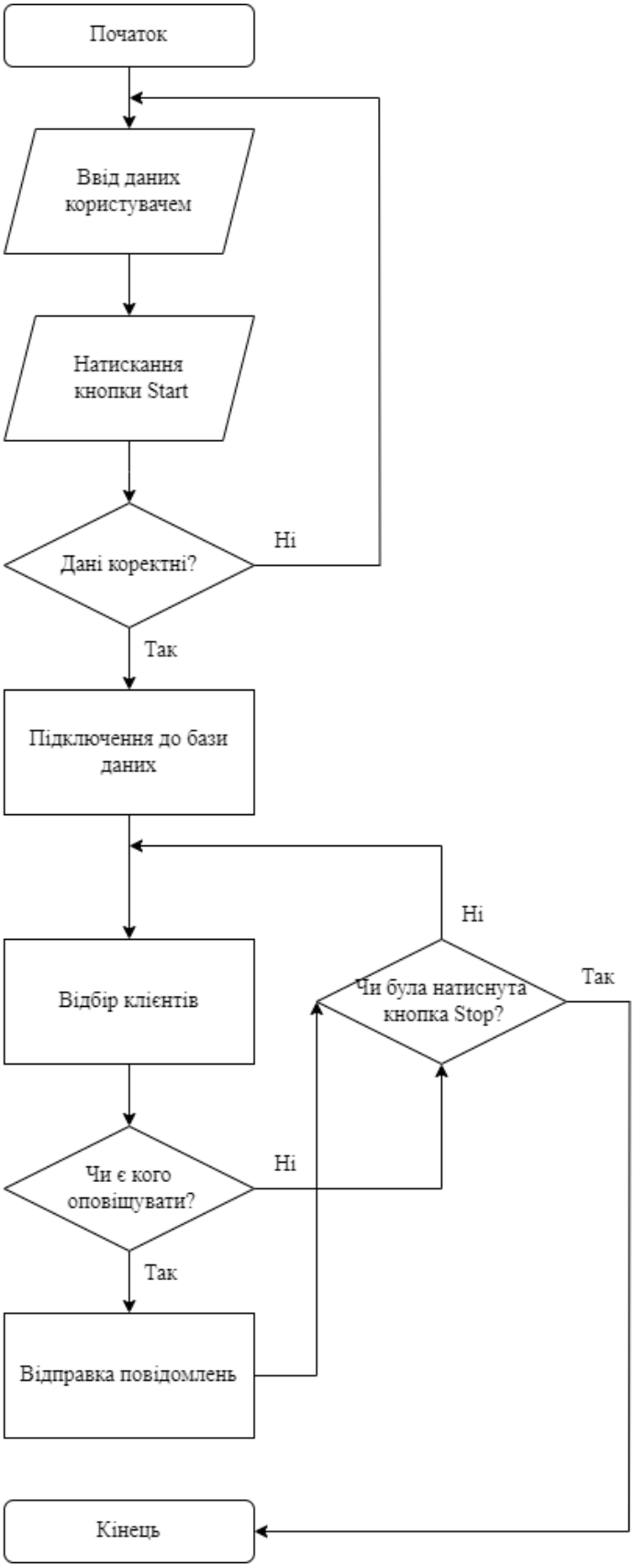
Система автоматично сповіщення неактивних клієнтів

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Юзина С. С.			Система автоматичного сповіщення неактивних клієнтів Алгоритм дій програмного забезпечення	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Трочун Є. В.				
Н. Контр.		Виноградов Ю.М				КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-92
Затвердив						

ДОДАТОК 4

Система автоматично сповіщення неактивних клієнтів

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 10

Київ 2023 р

```java
// Main.java
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.List;
import java.util.Vector;

public class Main {

    private static DatabaseConnector dbConnector;
    private static SmsSender smsSender;
    private static EmailSender emailSender;
    private static JTextArea logArea;
    private static JTextField daysOnlineField;
    private static JTextField daysNotifiedField;
    private static JTextField daysPurchaseField;

    public static void notifyClients(int daysOnline, int daysNotified, int
daysPurchase) {
        ClientsToNotify clientsToNotify =
dbConnector.getClientsToNotify(daysOnline, daysNotified, daysPurchase);

        processClients(clientsToNotify.getNeverPurchased(), "noPurchase");
        processClients(clientsToNotify.getNoRecentPurchase(), "noRecent");
        processClients(clientsToNotify.getBirthdayClients(), "birthday");
    }

    private static void processClients(List<Client> clients, String messageType)
{
        for (Client client : clients) {
            boolean notificationSent = false;
            if (client.isEmailPref()) {
                emailSender.sendEmail(client.getName(), client.getEmail(),
messageType);
                String id = String.format("%-5s", client.getId()).replace(' ',
'.');
                String name = String.format("%-10s", client.getName()).replace('
', '.');
                String email = String.format("%-20s",
client.getEmail()).replace(' ', '.');
                logArea.append(String.format("Customer id: %s Email sent to: %s
e-mail: %s\n", id, name, email));
                notificationSent = true;
            }
            if (client.isNumberPref()) {
                smsSender.sendSms(client.getName(), client.getNumber(),
messageType);
                String id = String.format("%-5s", client.getId()).replace(' ',
'.');
                String name = String.format("%-10s", client.getName()).replace('
', '.');
                String number = String.format("%-20s",
```

```
client.getNumber()).replace(' ', '.');
                logArea.append(String.format("Customer id: %s Sms sent to:   %s
number: %s\n", id, name, number));

                notificationSent = true;
            }
            if (notificationSent) {
                dbConnector.notifyClient(client);
            }
        }
    }


    private static void createAndShowGUI() {

        smsSender = new SmsSender();
        emailSender = new EmailSender();

        JFrame frame = new JFrame("Notification");
        frame.setSize(1200, 600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());

        JTextField hostField = new JTextField("localhost", 15);
        JTextField portField = new JTextField("3306", 5);
        JTextField databaseField = new JTextField("client_db", 15);
        JTextField usernameField = new JTextField("root", 10);
        JPasswordField passwordField = new JPasswordField("root", 10);
        JTextField timerDelayField = new JTextField("10", 5);
        JTextField tableNameField = new JTextField("Enter table name", 20);
        dbConnector = new DatabaseConnector(hostField.getText(),
portField.getText(),
                databaseField.getText(), usernameField.getText(),
passwordField.getText());
        JTable table = new JTable();
        JScrollPane tableScrollPane = new JScrollPane(table);
        table.setBackground(Color.decode("#c4c4c4"));

        JButton updateTableButton = new JButton("Update table view");
        updateTableButton.setBackground(Color.decode("#5e7fab"));
        updateTableButton.addActionListener(e -> {
            String tableName = tableNameField.getText();
            ResultSet rs = dbConnector.getData(tableName);
            try {
                table.setModel(buildTableModel(rs));
            } catch (SQLException throwable) {
                throwable.printStackTrace();
            }
        });

        Timer timer = new Timer(10000, evt -> {
            if (dbConnector.getConnection() != null) {
                int daysOnline = Integer.parseInt(daysOnlineField.getText());
                int daysNotified =
Integer.parseInt(daysNotifiedField.getText());
                int daysPurchase =
Integer.parseInt(daysPurchaseField.getText());
```

```
                notifyClients(daysOnline, daysNotified, daysPurchase);
            } else {
                JOptionPane.showMessageDialog(null, "Failed to connect to
database!");
            }
        });

        JButton startButton = new JButton("Start");
        startButton.setBackground(Color.decode("#5fab5e"));
        startButton.addActionListener(new ActionListener() {
            boolean running = false;
            public void actionPerformed(ActionEvent e) {
                if (running) {
                    timer.stop();
                    try {
                        int timerDelay =
Integer.parseInt(timerDelayField.getText());
                        timer.setDelay(timerDelay*1000);
                    } catch (NumberFormatException ex) {
                        JOptionPane.showMessageDialog(null, "Please enter a
valid number for timer delay!");
                    }
                    startButton.setText("Start");
                    startButton.setBackground(Color.decode("#5fab5e"));
                    running = false;
                } else {
                    if (dbConnector.getConnection() != null) {
                        try {
                        int daysOnline =
Integer.parseInt(daysOnlineField.getText());
                        int daysNotified =
Integer.parseInt(daysNotifiedField.getText());
                        int daysPurchase =
Integer.parseInt(daysPurchaseField.getText());
                        notifyClients(daysOnline, daysNotified, daysPurchase);
                        int timerDelay =
Integer.parseInt(timerDelayField.getText());
                        timer.setDelay(timerDelay*1000);
                        } catch (NumberFormatException ex) {
                            JOptionPane.showMessageDialog(null, "Please enter a
valid numbers in all fields!");
                        }
                        timer.start();
                        startButton.setText("Stop");
                        startButton.setBackground(Color.decode("#ab5e6d"));
                        running = true;
                    } else {
                        JOptionPane.showMessageDialog(null, "Failed to connect
to database!");
                    }
                }
            }
        });



        JButton connectButton = new JButton("Connect/Check");
```

					ІАЛЦ.467200.007 Д4		
		№ докум.	Підпис	Дата			
Розробив		Юзина С. С.			Система автоматичного сповіщення неактивних клієнтів Текс програмного коду	Літ.	Аркуш 1 / Аркушів 10
Перевірив		Трочун Є. В.					
Н. Контр.		Виноградов Ю.М				КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-92	
Затвердив							

```
        connectButton.setBackground(Color.decode("#5e7fab"));
        connectButton.addActionListener(e -> {
            String host = hostField.getText();
            String port = portField.getText();
            String database = databaseField.getText();
            String username = usernameField.getText();
            String password = new String(passwordField.getPassword());

            dbConnector = new DatabaseConnector(host, port, database, username,
password);

            if (dbConnector.getConnection() != null) {
                JOptionPane.showMessageDialog(null, "Connection successful!");
            } else {
                JOptionPane.showMessageDialog(null, "Failed to connect to
database!");
            }
        });


        logArea = new JTextArea(20, 50);
        logArea.setFont(new Font("Courier New", Font.BOLD, 16));
        logArea.setEditable(false);
        logArea.setBackground(Color.decode("#c4c4c4"));
        JScrollPane scrollPane = new JScrollPane(logArea);

        daysOnlineField = new JTextField("20", 5);
        daysNotifiedField = new JTextField("30", 5);
        daysPurchaseField = new JTextField("30", 5);

        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));


        Dimension fieldSize = new Dimension(20, 30);
        Font fieldFont = new Font("Arial", Font.BOLD, 16);

        JLabel daysOnlineLabel = new JLabel("Days Online:");
        daysOnlineLabel.setFont(fieldFont);
        JLabel daysNotifiedLabel = new JLabel("Days Notified:");
        daysNotifiedLabel.setFont(fieldFont);
        JLabel daysPurchaseLabel = new JLabel("Days Purchase:");
        daysPurchaseLabel.setFont(fieldFont);
        JLabel timerDelayLabel = new JLabel("Seconds of cycle:");
        timerDelayLabel.setFont(fieldFont);


        daysOnlineField.setPreferredSize(fieldSize);
        daysOnlineField.setFont(fieldFont);

        daysNotifiedField.setPreferredSize(fieldSize);
        daysNotifiedField.setFont(fieldFont);

        daysPurchaseField.setPreferredSize(fieldSize);
        daysPurchaseField.setFont(fieldFont);
        timerDelayField.setPreferredSize(fieldSize);
        timerDelayField.setFont(fieldFont);
```

```
        JPanel showPanel = new JPanel();
        showPanel.setLayout(new BoxLayout(showPanel, BoxLayout.Y_AXIS));
        tableNameField.setMaximumSize(new Dimension(Integer.MAX_VALUE,
tableNameField.getPreferredSize().height));

        showPanel.setBackground(Color.decode("#c4c4c4"));
        showPanel.add(scrollPane);
        showPanel.add(tableNameField);
        showPanel.add(Box.createVerticalGlue());
        showPanel.add(tableScrollPane);

        JPanel dayControlPanel = new JPanel(new GridLayout(5, 2));
        dayControlPanel.setBackground(Color.lightGray);
        dayControlPanel.add(daysOnlineLabel);
        dayControlPanel.add(daysOnlineField);
        dayControlPanel.add(daysNotifiedLabel);
        dayControlPanel.add(daysNotifiedField);
        dayControlPanel.add(daysPurchaseLabel);
        dayControlPanel.add(daysPurchaseField);
        dayControlPanel.add(timerDelayLabel);
        dayControlPanel.add(timerDelayField);
        dayControlPanel.add(startButton);


        JPanel connectionPanel = new JPanel(new GridLayout(6, 2));
        connectionPanel.setBackground(Color.lightGray);
        connectionPanel.add(new JLabel("Host:"));
        connectionPanel.add(hostField);
        connectionPanel.add(new JLabel("Port:"));
        connectionPanel.add(portField);
        connectionPanel.add(new JLabel("Database:"));
        connectionPanel.add(databaseField);
        connectionPanel.add(new JLabel("Username:"));
        connectionPanel.add(usernameField);
        connectionPanel.add(new JLabel("Password:"));
        connectionPanel.add(passwordField);
        connectionPanel.add(connectButton);
        connectionPanel.add(updateTableButton);


        JPanel topPanel = new JPanel(new GridLayout(6, 1));
        topPanel.setBackground(Color.lightGray);

        topPanel.add(dayControlPanel);
        topPanel.add(connectionPanel);

        panel.setLayout(new BorderLayout());
        panel.add(topPanel, BorderLayout.NORTH);

        frame.add(panel, BorderLayout.LINE_START);
        frame.add(showPanel, BorderLayout.CENTER);
        frame.setVisible(true);
    }

    public static DefaultTableModel buildTableModel(ResultSet rs) throws
SQLException {
```

					ІАЛЦ.467200.007 Д4		
		№ докум.	Підпис	Дата			
Розробив		Юзина С. С.			Система автоматичного сповіщення неактивних клієнтів Текс програмного коду	Літ.	Аркуш 1 / Аркушів 10
Перевірив		Трочун Є. В.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-92	
Н. Контр.		Виноградов Ю.М					
Затвердив							

```
        ResultSetMetaData metaData = rs.getMetaData();

        Vector<String> columnNames = new Vector<>();
        int columnCount = metaData.getColumnCount();
        for (int column = 1; column <= columnCount; column++) {
            columnNames.add(metaData.getColumnName(column));
        }

        Vector<Vector<Object>> data = new Vector<>();
        while (rs.next()) {
            Vector<Object> vector = new Vector<>();
            for (int columnIndex = 1; columnIndex <= columnCount; columnIndex++)
{
                vector.add(rs.getObject(columnIndex));
            }
            data.add(vector);
        }

        return new DefaultTableModel(data, columnNames);
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(Main::createAndShowGUI);
    }
}
```

```
// DatabaseConnector.java


import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class DatabaseConnector {


    private Connection connection = null;



    public DatabaseConnector(String host, String port, String database, String
user, String pass) {
        String dbUrl = "jdbc:mysql://" + host + ":" + port + "/" + database;
        try {
            connection = DriverManager.getConnection(dbUrl, user, pass);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public Connection getConnection() {
        return connection;
    }

    public ClientsToNotify getClientsToNotify(int daysOnline, int daysNotified,
```

					ІАЛЦ.467200.007 Д4		
		№ докум.	Підпис	Дата			
Розробив		Юзина С. С.			Система автоматичного сповіщення неактивних клієнтів Текс програмного коду	Літ.	Аркуш 1 / Аркушів 10
Перевірив		Трочун Є. В.					
Н. Контр.		Виноградов Ю.М				КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-92	
Затвердив							

```
int daysPurchase) {
        String queryNeverPurchased = "SELECT * FROM clients " +
                "LEFT JOIN purchases ON clients.id = purchases.client_id " +
                "WHERE purchases.client_id IS NULL " +
                "AND (DATE_ADD(clients.last_online, INTERVAL " + daysOnline + "
DAY) <= CURDATE() " +
                "AND (DATE_ADD(clients.last_notified, INTERVAL " + daysNotified
+
                " DAY) <= CURDATE() OR clients.last_notified IS NULL))";

        String queryNoRecentPurchase = "SELECT * FROM clients " +
                "WHERE DATE_ADD(last_purchase, INTERVAL " + daysPurchase + "
DAY) <= CURDATE() " +
                "AND (DATE_ADD(last_online, INTERVAL " + daysOnline + " DAY) <=
CURDATE() " +
                "AND (DATE_ADD(last_notified, INTERVAL " + daysNotified + " DAY)
<= CURDATE() OR last_notified IS NULL))";

        String queryBirthday = "SELECT * FROM clients " +
                "WHERE DAY(birth) = DAY(CURDATE()) AND MONTH(birth) =
MONTH(CURDATE()) " +
                "AND (DATE(last_notified) != CURDATE() OR last_notified IS
NULL)";

        List<Client> neverPurchased = executeClientQuery(queryNeverPurchased);
        List<Client> noRecentPurchase =
executeClientQuery(queryNoRecentPurchase);
        List<Client> birthdayClients = executeClientQuery(queryBirthday);

        return new ClientsToNotify(neverPurchased, noRecentPurchase,
birthdayClients);
    }


    private List<Client> executeClientQuery(String query) {
        List<Client> clients = new ArrayList<>();
        try {
            PreparedStatement stmt = this.connection.prepareStatement(query);
            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
                Client client = new Client();
                client.setId(rs.getInt("id"));
                client.setName(rs.getString("name"));
                client.setLastName(rs.getString("last_name"));
                client.setBirth(rs.getDate("birth"));
                client.setEmail(rs.getString("email"));
                client.setNumber(rs.getString("number"));
                client.setEmailPref(rs.getBoolean("email_pref"));
                client.setNumberPref(rs.getBoolean("number_pref"));
                client.setLastOnline(rs.getDate("last_online"));
                client.setLastPurchase(rs.getDate("last_purchase"));
                client.setLastNotified(rs.getDate("last_notified"));
                clients.add(client);
            }
        } catch (SQLException e) {
            e.printStackTrace();
```

					ІАЛЦ.467200.007 Д4		
		№ докум.	Підпис	Дата			
Розробив		Юзина С. С.			Система автоматичного сповіщення неактивних клієнтів Текс програмного коду	Літ.	Аркуш 1 / Аркушів 10
Перевірив		Трочун Є. В.					
Н. Контр.		Виноградов Ю.М				КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-92	
Затвердив							

```
        }

        return clients;
    }

    public void notifyClient(Client client) {
        String query = "UPDATE clients SET last_notified = CURDATE() WHERE id =
?";
        try {
            PreparedStatement stmt = this.connection.prepareStatement(query);
            stmt.setInt(1, client.getId());
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public ResultSet getData(String tableName) {
        try {
            Statement statement = connection.createStatement();
            return statement.executeQuery("SELECT * FROM " + tableName);
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }


}



// ClientsToNotify.java

import java.util.List;

public class ClientsToNotify {
    List<Client> neverPurchased;
    List<Client> noRecentPurchase;
    List<Client> birthdayClients;

    public ClientsToNotify(List<Client> neverPurchased, List<Client>
noRecentPurchase, List<Client> birthdayClients) {
        this.neverPurchased = neverPurchased;
        this.noRecentPurchase = noRecentPurchase;
        this.birthdayClients = birthdayClients;
    }

    public List<Client> getNeverPurchased() {
        return neverPurchased;
    }

    public List<Client> getNoRecentPurchase() {
        return noRecentPurchase;
    }

    public List<Client> getBirthdayClients() {
```

```
        return birthdayClients;
    }
}


//Client.java



import java.sql.Date;

public class Client {
    private int id;
    private String name;
    private String lastName;
    private String email;
    private String number;
    private boolean emailPref;
    private boolean numberPref;
    private Date birth;
    private Date lastOnline;
    private Date lastPurchase;
    private Date lastNotified;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }


    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getNumber() {
        return number;
    }
```

```java
public void setNumber(String number) {
    this.number = number;
}

public boolean isEmailPref() {
    return emailPref;
}

public void setEmailPref(boolean emailPref) {
    this.emailPref = emailPref;
}

public boolean isNumberPref() {
    return numberPref;
}

public void setNumberPref(boolean numberPref) {
    this.numberPref = numberPref;
}

public void setBirth(Date birth) {
    this.birth = birth;
}

public void setLastOnline(Date lastOnline) {
    this.lastOnline = lastOnline;
}

public void setLastPurchase(Date lastPurchase) {
    this.lastPurchase = lastPurchase;
}


public void setLastNotified(Date lastNotified) {
    this.lastNotified = lastNotified;
}
```