ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система оцінки ступеня ризику банківських клієнтів на базі Java»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Іван САМУСЕВИЧ
(підпис) Ім'я, ПРІЗВИЩЕ здобувача

Виконав: здобувач вищої освіти гр. КНД-42
Іван САМУСЕВИЧ
Ім'я, ПРІЗВИЩЕ

Керівник: Сергій ЩЕРЯКОВ, к.т.н, доцент
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук
Ступінь вищої освіти Бакалавр
Спеціальність 122 Комп'ютерні науки
Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою
122 Комп'ютерних наук
Віктор ВИШНІВСЬКИЙ
«____» _____ 20__ р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Самусевич Іван Сергійович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Система оцінки ступеня ризику банківських клієнтів на базі Java»

керівник кваліфікаційної роботи Сергій ІЩЕРЯКОВ, к.т.н, доцент
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.
2. Строк подання кваліфікаційної роботи «30» травня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: вимоги до структури GUI-додатків та принципи проектування програмного забезпечення.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   1. Проектування та реалізація графічного інтерфейсу користувача з використанням JavaFX.
   2. Розробка логічної структури бази даних клієнтів і транзакцій.
   3. Дослідження загальних фреймворків та бібліотек
5. Перелік ілюстративного матеріалу: презентація
6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз науково-технічної літератури	28.03.25 - 14.05.25	Виконано
2	Проектування структури бази даних	15.04.25 - 19.04.25	Виконано
3	Розробка структури графічного інтерфейсу в JavaFX	20.04.25 - 28.04.25	Виконано
4	Реалізація модуля аналізу транзакцій та обчислення ризику	29.04.25 - 04.05.25	Виконано
5	Тестування функціоналу та оптимізація системи	05.05.25 - 12.05.25	Виконано
6	Написання вступу, висновків та оформлення тексту за ДСТУ	13.05.25 - 16.05.25	Виконано
7	Підготовка реферату, додатків, титулки та повного оформлення	17.05.25 -19.05.25	Виконано
8	Підготовка презентації та виступу до захисту	20.05.2025 р.	Виконано

Здобувач вищої освіти _____ Іван САМУСЕВИЧ

(підпис) (Ім’я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи _____ Сергій ІЩЕРЯКОВ

(підпис) (Ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 57 сторінок, 11 рисунків, 4 таблиці, 20 джерел.

Мета роботи – розробити програмний інструмент для банків, який на основі аналізу підозрілих клієнтських операцій автоматично оцінює ступінь ризикованості клієнта.

Об’єкт дослідження – процес виявлення та оцінки ризикових банківських клієнтів на основі їх фінансової активності.

Предмет дослідження – методи й алгоритми автоматизованої оцінки ступеня ризику клієнта банку та їх реалізація у вигляді десктопної Java FX-програми з базою даних PostgreSQL.

Короткий зміст роботи – У кваліфікаційній роботі розроблено програмний засіб для банківських установ, призначений для оцінювання ступеня ризику клієнтів на основі їх фінансових операцій. Програма реалізована мовою Java з використанням фреймворку JavaFX і бази даних PostgreSQL. Система аналізує транзакції, нараховує бали ризику згідно з заданими правилами та класифікує клієнтів за рівнями ризику. Інтерфейс забезпечує фільтрацію, візуалізацію та експорт результатів. Функціональність програми протестовано на наборі умовних даних, що підтвердило її коректність та придатність для використання в якості інструменту внутрішнього фінансового моніторингу.

КЛЮЧОВІ СЛОВА: Java, JavaFX, PostgreSQL, банківський ризик, клієнт, фінансовий моніторинг, транзакція, бальна система, класифікація, програмний засіб.

ЗМІСТ

ВСТУП

У сучасних умовах фінансової нестабільності, зростання кількості шахрайських операцій та ризиків, пов’язаних із діяльністю банківських установ, питання оцінки надійності та ризикованості клієнтів набуває особливої актуальності. Банки щодня стикаються з необхідністю аналізу великої кількості фінансових транзакцій, серед яких можуть ховатися ознаки підозрілої активності, зв’язків із підсанкційними особами або суб’єктами, пов’язаними з державою-агресором.

У відповідь на ці виклики у межах даної кваліфікаційної роботи розробляється **програмний засіб**, який дозволяє банківським установам оцінювати ступінь ризику клієнтів на основі чітко визначених правил. Програма аналізує фінансові операції клієнтів, зокрема великі зняття готівки, часті поповнення рахунку, підозрілі перекази тощо. Кожна операція оцінюється за відповідною шкалою, формуючи загальний бал ризику. На підставі цієї суми клієнту присвоюється відповідний рівень ризику: **немає ризику, низький, середній або високий ризики**.

Програмний засіб реалізовано з використанням мови **Java**, графічного фреймворку **Java FX** та системи керування базами даних **PostgreSQL** у середовищі **NetBeans**. Його використання дозволяє автоматизувати частину процедур фінансового моніторингу та покращити ефективність роботи відповідних підрозділів банку.

Цілі та задачі дослідження

Основною ціллю дипломної роботи є розробка програмного засобу для банківських установ, який дозволяє визначати ступінь ризикованості клієнтів на основі аналізу їх фінансових операцій згідно з заздалегідь встановленими правилами.

- Для досягнення цієї мети були поставлені наступні задачі:

- Провести аналіз підходів до оцінки ризику клієнтів у банківській сфері.
- Визначити критерії та правила для виявлення підозрілих транзакцій.
- Розробити структуру бази даних для зберігання інформації про клієнтів та їх операції.
- Реалізувати графічний інтерфейс користувача з використанням Java FX.
- Створити функціонал для розрахунку загального балу ризику на основі оцінених операцій.
- Забезпечити присвоєння рівня ризику (немає ризику, низький, середній, високий) клієнту згідно з підсумковим балом.
- Провести тестування програмного засобу та оцінити його ефективність.

Об’єктом дослідження виступають інформаційні процеси, пов’язані з аналізом фінансових операцій клієнтів банківських установ з метою виявлення ризикових ознак у їх поведінці та прийняття обґрунтованих рішень щодо рівня потенційної небезпеки.

Предметом дослідження є методи та програмні засоби реалізації алгоритмів виявлення підозрілих транзакцій, оцінювання ризику клієнтів на основі порогово-бальної системи, а також класифікації клієнтів за ступенем фінансової ризикованості.

Дослідження спрямоване на формалізацію та автоматизацію процесу оцінювання ризику клієнтів шляхом розробки спеціалізованого програмного засобу, що ґрунтується на визначених правилах аналізу транзакцій та обчислення сумарного ризик-індекс.

1 ТЕОРЕТИЧНІ ОСНОВИ ОЦІНКИ РИЗИКУ КЛІЄНТІВ У БАНКІВСЬКІЙ СФЕРІ

1.1 Сутність і значення оцінки фінансових ризиків клієнтів банку

Фінансові ризики в банківській діяльності є невід'ємною складовою кредитно-фінансових операцій. Одним з найбільш критичних аспектів управління ризиками є своєчасне виявлення клієнтів, чиї дії можуть становити загрозу для безпеки банку, зокрема — через підозрілі, нетипові або потенційно шахрайські транзакції.

У світовій практиці управління фінансовими установами ключовим принципом є ризик-орієнтований підхід (risk-based approach), який передбачає аналіз не лише окремих операцій, а й загальної поведінки клієнта в динаміці. Такий підхід дозволяє не просто реагувати на факти зловживань постфактум, а профілактично виявляти потенційно небезпечних клієнтів і приймати рішення щодо подальшої співпраці, обмеження функціоналу або ініціювання фінансового розслідування.

У контексті банківської діяльності під ризикованим клієнтом мається на увазі особа (фізична чи юридична), чиї фінансові операції свідчать про високу ймовірність:

- відмивання коштів;
- фінансування тероризму;
- залучення до тіньових схем;
- корупція;
- співпраці з підсанкційними або ворожими суб’єктами (наприклад

громадянами РФ, компаніями, що підлягають санкціям тощо);

- фінансової нестабільності або фіктивної діяльності.

У 2024 році Національний Банк України посилив вимоги до внутрішніх процедур фінансового моніторингу, зобов’язавши банки реалізовувати більш гнучкі та точні системи ідентифікації ризиків. При цьому ефективність такого моніторингу значною мірою залежить від автоматизованих засобів аналізу, які мають забезпечити швидке, точне й об'єктивне визначення рівня ризику клієнта.

В умовах зростання обсягу транзакцій та розвитку цифрових послуг ручна перевірка клієнтів є малоефективною, а тому потребує заміни на програмно-аналітичні інструменти, здатні:

- присвоювати бали ризику на основі заданих правил;
- класифікувати клієнтів за ступенем ризику (немає ризику, низький, середній, високий).
- агрегувати дані з баз клієнтів;

Таким чином, оцінка ризикованості клієнтів за допомогою спеціалізованого програмного забезпечення є критичним елементом безпеки банківської системи. Розробка такої програми дозволяє не лише підвищити точність виявлення ризиків, а й оптимізувати роботу аналітиків, які відповідають за фінансовий моніторинг.

1.2 Методи виявлення підозрілих фінансових операцій

Виявлення підозрілих фінансових операцій є ключовим елементом фінансового моніторингу банківської діяльності. Своєчасне виявлення таких транзакцій дозволяє запобігти злочинам, пов’язаним з легалізацією (відмиванням) доходів, отриманих злочинним шляхом, фінансуванням тероризму, ухиленням від

оподаткування, шахрайством та іншими порушеннями законодавства. У сучасній практиці використовуються декілька підходів до виявлення аномальної фінансової активності клієнтів, кожен з яких має свої переваги та недоліки.

Правилоорієнтовані методи (Rule-based approach)

Цей підхід базується на застосуванні заздалегідь визначених експертних правил, що описують ознаки потенційно підозрілих операцій. Наприклад:

- зняття готівки понад встановлений поріг (наприклад, 200 000 грн);
- багаторазові транзакції на однакову суму в короткий період часу;
- операції з контрагентами з країн, що перебувають під санкціями;
- різке зростання обсягу транзакцій без очевидного економічного обґрунтування.

Перевагами цього підходу є простота реалізації, прозорість логіки виявлення та можливість легкого налаштування під регуляторні вимоги. Однак його обмеженням є низька гнучкість — нові шахрайські схеми, що не враховані в наборах правил, можуть залишатися непоміченими.

Статистичні методи

Статистичний підхід ґрунтується на математичному аналізі історичних даних про фінансову активність клієнтів. Зокрема:

- обчислення середніх значень та стандартних відхилень для транзакцій клієнта;
- виявлення відхилень від звичних шаблонів поведінки;
- побудова нормального (типового) профілю операцій клієнта.

Такі методи дозволяють більш адаптивно виявляти нетипову поведінку, проте потребують великих обсягів якісних даних та часто не є придатними для невеликих банківських систем. Їх ефективність також залежить від рівня

сегментації клієнтів (наприклад, за типом або категорією ризику).

Кластерний аналіз

Методи кластерного аналізу (unsupervised learning) дають змогу групувати клієнтів за подібністю фінансової поведінки. Застосовуються алгоритми:

- k-середніх (k-means),
- DBSCAN,
- ієрархічна кластеризація.

Такі методи допомагають виявляти клієнтів, які «випадають» з типового кластеру поведінки, або змінюють свою модель дій. Вони добре працюють на великих масивах даних, але потребують значних обчислювальних ресурсів та глибокої попередньої обробки інформації. У контексті десктопного додатку їх використання є обмеженим через складність реалізації та обсяг даних.

Метод чорних/санкційних списків

Окремий напрямок виявлення ризикових клієнтів — це перевірка осіб та організацій за списками підсанкційних, терористичних або небажаних суб’єктів. Зазвичай використовуються:

- національні реєстри (НБУ, СБУ);
- міжнародні списки (OFAC, EU, UN);
- внутрішні «чорні списки», що формуються банком самостійно.

Технічно реалізація цього методу може базуватись на регулярному оновленні списків у базі даних або інтеграції з API відповідних джерел. Метод дає швидкий результат і дозволяє миттєво блокувати ризикові операції або зупиняти обслуговування клієнта.

Класифікаційні методи на основі порогових балів (Scoring approach)

Суть цього підходу полягає в тому, що кожній ознаці підозрілої поведінки надається певна вага у вигляді бального значення. Після обробки всіх операцій клієнта за певний період сума нарахованих балів дозволяє віднести клієнта до відповідної групи ризику.

Цей метод є зручним для програмної реалізації, добре масштабується, дозволяє швидко адаптуватися до нових умов шляхом зміни бальних коефіцієнтів або порогових значень. Саме цей підхід було реалізовано у рамках даної роботи, оскільки він поєднує в собі простоту, зрозумілість та достатню ефективність для виявлення клієнтів із ризикованою поведінкою.

З-поміж усіх методів, найбільш придатними для використання у невеликих автоматизованих рішеннях (зокрема у десктопному програмному забезпеченні) є правилоорієнтований підхід та порогово-бальна модель. Вони не потребують великих обсягів даних, не вимагають складної попередньої обробки і можуть бути ефективно застосовані для оперативного виявлення підозрілих клієнтів з мінімальними витратами ресурсів.

1.3 Підходи до класифікації клієнтів за рівнем ризику

Класифікація клієнтів за рівнем ризику є однією з ключових складових системи фінансового моніторингу, що базується на ризик-орієнтованому підході. Такий підхід дозволяє банківським установам ефективніше здійснювати контроль над фінансовими операціями, запобігати відмиванню доходів, фінансуванню тероризму та іншим незаконним діям.

Грамотно організована система класифікації клієнтів дозволяє:

- фокусувати ресурси на найбільш ризикованих суб’єктах;
- своєчасно виявляти ознаки підозрілої активності;
- знижувати фінансові та регуляторні ризики для банку;
- забезпечувати відповідність законодавчим і нормативним вимогам НБУ та міжнародних інституцій (FATF, Basel Committee, ЄС тощо).

Основи ризик-орієнтованого підходу

Відповідно до рекомендацій **FATF (Financial Action Task Force)** та положень Національного банку України, всі фінансові установи мають впроваджувати **Risk-Based Approach (RBA)** — підхід, який передбачає оцінювання рівня ризику кожного клієнта на основі певних факторів.

Серед основних факторів, що враховуються під час оцінювання ризику:

- **Тип клієнта**: фізична особа, юридична особа, іноземна компанія, PEP (Politically Exposed Person — політично значуща особа), бенефіціар нерезидентної структури тощо;
- **Географічний фактор**: країна громадянства, місце ведення бізнесу або реєстрації (наприклад, держави з високим рівнем корупції або під санкціями);
- **Характер діяльності**: участь у ризикових секторах (наприклад, готівковий обіг, ігровий бізнес, обмін валют);
- **Типові або нетипові транзакції**: зняття великих сум готівки, регулярні перекази без очевидної ділової логіки;
- **Зв’язки з іншими підозрілими клієнтами** або структурами, що фігурують у внутрішніх або міжнародних списках.

Такий підхід дозволяє оцінювати не лише окремі транзакції, а й загальну поведінкову модель клієнта.

Категоризація клієнтів

У більшості банківських систем клієнти за результатами оцінки поділяються на такі **категорії ризику**:

- **Немає ризику** — типова діяльність, прозора структура власності, стабільна історія взаємодії з банком, відсутність зв'язків з підозрілими особами;
- **Низький ризик** — окремі незначні відхилення, нетипові, але пояснювані транзакції;
- **Середній ризик** — наявність регулярних операцій, що не відповідають фінансовому профілю клієнта, ризикований вид діяльності, зв'язки з компаніями з «сірих» юрисдикцій;
- **Високий ризик** — діяльність або операції клієнта пов'язані з ознаками шахрайства, відмивання коштів, або ж клієнт включений до санкційного списку.

Порогово-бальна система оцінювання

Найпоширенішим підходом до автоматизованої класифікації клієнтів є **порогово-бальна система** (scoring model). Вона полягає у присвоєнні кожній підозрілій ознаці визначеної кількості балів. Загальна сума балів визначає рівень ризику клієнта. Наприклад:

- зняття готівки понад 200 тис. грн — 7 балів;
- переказ коштів до країни під санкціями — 15 балів;
- зв'язки з підсанкційними клієнтами — 10 балів;
- кількість підозрілих операцій — 3 бали за кожну понад норму.

Класифікація за балами:

- **0 балів** — немає ризику;
- **1–9 балів** — низький ризик;

- **10–19 балів** — середній ризик;
- **20 балів і більше** — високий ризик.

Такий підхід дозволяє автоматизувати процес класифікації, швидко адаптуватися до змін у законодавстві або поведінці клієнтів і інтегрується у програмне забезпечення з мінімальними обчислювальними витратами.

Сценарний підхід

У деяких установах також використовується **сценарний підхід**, при якому створюються **моделі поведінки клієнтів** (scenarios), що відповідають відомим схемам зловживань. Наприклад:

- **Сценарій "відмивання коштів"**: регулярні грошові перекази між пов'язаними клієнтами, виведення коштів у готівку, роздроблені транзакції;
- **Сценарій "обхід санкцій"**: часті перекази у прикордонні регіони, використання клієнтів-«прокладок»;
- **Сценарій "фіктивна діяльність"**: великі обсяги руху коштів при мінімальних ознаках бізнесу.

При виявленні відповідності одному чи декільком сценаріям, клієнту може бути присвоєно підвищений рівень ризику, навіть при відсутності окремих критичних транзакцій.

Інтеграція з зовнішніми джерелами

Важливим джерелом додаткової інформації для класифікації клієнтів є **зовнішні бази даних**:

- міжнародні санкційні списки (**OFAC**, **EU**, **UN**);
- списки **PEP** (політично значущих осіб);
- внутрішні чорні списки банку;
- відкриті джерела даних про судимості, корпоративні зв'язки тощо.

Система моніторингу може виконувати автоматичну перевірку клієнтів по цих базах, і при виявленні збігів ризикова категорія клієнта автоматично підвищується.

Роль ручного підтвердження

Попри високу ефективність автоматичних систем, у більшості банків **остаточне рішення** про присвоєння рівня ризику ухвалює **спеціаліст з фінансового моніторингу**. Це дозволяє враховувати індивідуальні обставини, унікальний контекст операцій клієнта, а також помилки алгоритмічних моделей.
Комплексний підхід до класифікації клієнтів за рівнем ризику передбачає поєднання автоматизованої оцінки за бальною системою, інтеграцію з зовнішніми базами, аналіз поведінкових шаблонів та експертну верифікацію. У рамках розробленої системи реалізовано саме **порогово-бальну модель**, яка є оптимальною для десктопного програмного забезпечення й відповідає сучасним практикам у банківській галузі.

1.4 Вимоги до програмних засобів для фінансового моніторингу

Фінансовий моніторинг — це ключова складова протидії легалізації доходів, одержаних злочинним шляхом, та фінансуванню тероризму. У зв’язку з цим програмні засоби, що використовуються для автоматизації таких процесів у банках, повинні відповідати ряду технічних, функціональних та регуляторних вимог.

Функціональні вимоги

Система моніторингу повинна забезпечувати реалізацію таких функцій:

- Автоматична класифікація клієнтів за рівнем ризику згідно з вбудованими алгоритмами (порогово-бальні системи, шаблони поведінки).
- Формування звітів для регуляторів (НБУ, Держфінмоніторинг).
- Відстеження зв’язків клієнтів (контрагенти, транзакційні ланцюги).

Вимоги до інтерфейсу

Інтерфейс повинен бути:

- Інтуїтивно зрозумілим для співробітників фінмоніторингу;
- Інформативним — наочне відображення ризиків, транзакцій, балів, суми;
- Гнучким — можливість кастомізації розміру таблиці даних, фільтрація даних.

Вимоги до звітності

Система повинна:

- підтримувати експорт у форматах **XLSX**;
- вести статистику кількості клієнтів за формою ризику.

2 АНАЛІЗ І ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

2.1 Постановка задачі та визначення вимог до системи

У сучасних умовах динамічної цифровізації банківських послуг та зростання кіберзагроз особливої актуальності набуває питання забезпечення ефективного внутрішнього фінансового моніторингу. Банк щоденно обробляє тисячі фінансових операцій, серед яких можуть бути приховані транзакції, що мають ознаки відмивання коштів, фінансування терористичної діяльності або співпраці з підсанкційними особами чи структурами.

Зважаючи на обмеженість людських ресурсів та часові рамки перевірки, виникає необхідність **автоматизації процесу виявлення ризикованих клієнтів**, що базується на формалізованих критеріях та алгоритмах оцінювання.

Постановка задачі

Метою даного етапу є чітке формулювання задачі, що вирішується в межах створення інформаційної системи.

Основна задача — розробити програмний засіб, який дозволить:

- здійснювати аналіз фінансових операцій клієнтів банку за заданий період;
- виявляти транзакції з підвищеним ризиком (на основі суми, типу, частоти, країни призначення тощо);
- автоматично нараховувати клієнтам бали ризику відповідно до заздалегідь визначених правил;
- класифікувати клієнтів за чотирма рівнями ризику (немає ризику, низький, середній, високий);

- зберігати, обробляти та відображати ці дані у графічному інтерфейсі користувача;
- генерувати звіти, що можуть бути використані для подальшої експертної оцінки або передачі у відповідні підрозділи банку.

Таким чином, запропонована система виконує роль **підтримки прийняття рішень (Decision Support System)** у сфері внутрішнього банківського моніторингу, дозволяючи значно знизити навантаження на аналітичні відділи.

Функціональні вимоги до системи

Для реалізації поставленої задачі програмний продукт повинен забезпечувати реалізацію таких функцій:

1. Збір та обробка інформації:

- зчитування інформації з реляційної бази даних (PostgreSQL);
- обробка операцій клієнтів за обраний період (в межах до 3 місяців);
- підтримка фільтрації даних за ім’ям клієнта, частковим збігом тощо.

2. Нарахування балів ризику:

- за транзакціями понад порогові значення (наприклад, зняття готівки >100 000 грн);
- за операціями з підозрілими контрагентами (наприклад, з країн під санкціями);
- за підозрілими шаблонами поведінки (частота, повторюваність, нехарактерна активність);
- на основі флагів, як-от "санкційність" клієнта.

3. Визначення категорії ризику:

- автоматичне присвоєння рівня ризику відповідно до сумарного балу;
- чітке маркування клієнтів за категоріями: немає ризику, низький, середній,

високий.

4. Інтерфейс користувача (JavaFX GUI):

- відображення зведених даних по кожному клієнту у вигляді таблиці;
- підтримка фільтрів (радіокнопки "Дорівнює", "Починається з", "Містить");
- можливість вибору періоду аналізу (через DatePicker);
- візуальне виділення клієнтів залежно від рівня ризику (колірне маркування);
- експорт результатів у файл Excel.

5. Збереження даних:

- використання бази даних PostgreSQL для централізованого зберігання інформації;
- реалізація таблиць згідно з вимогами 3НФ для уникнення надлишковості;
- збереження цілісності даних за допомогою зовнішніх ключів та обмежень.

Нефункціональні вимоги

Крім функціональної коректності, система повинна відповідати ряду нефункціональних вимог:

- **Надійність:** система має гарантувати збереження даних навіть у випадках аварійного завершення роботи або втрати з’єднання з базою.
- **Продуктивність:** обробка запитів до БД має здійснюватися в межах прийнятного часу при роботі з десятками тисяч записів.
- **Масштабованість:** можливість розширення функціоналу в майбутньому (додавання нових правил, сценаріїв, інтеграція з API).
- **Сумісність:** система повинна працювати на будь-якому ПК з Java середовищем.

2.2 Проектування структури бази даних клієнтів та операцій

Одним із ключових компонентів програмного засобу для оцінки ступеня ризику банківських клієнтів є база даних, що забезпечує централізоване зберігання інформації про клієнтів і їх фінансову активність (транзакції).

На етапі проектування бази даних було враховано основні вимоги до структури, масштабованості, узгодженості та ефективності роботи з транзакційними даними.

Вимоги до проектування

При розробці структури бази даних враховувались наступні вимоги:

- **Логічна цілісність** — забезпечення правильних зв’язків між сутностями, уникнення дублювання інформації;
- **Масштабованість** — структура повинна підтримувати додавання нових типів транзакцій або розширення правил без суттєвих змін у схемі;
- **Зручність інтеграції** — структура бази повинна легко поєднуватися з програмною логікою Java-додатку через JDBC.
- Уникнення надлишковості;

Підхід до структурування даних

У процесі моделювання структури БД було визначено ключові сутності, які мають відображати предметну область:

- **Клієнти** — інформація про фізичних, юридичних осіб та інших клієнтів, що обслуговуються в банку;
- **Фінансові операції** — транзакції, які виконуються клієнтами і аналізуються на предмет ризику;

Нормалізація та логічна модель

Проєктування структури БД передбачає приведення таблиць до **третьої нормальної форми (3НФ)** для уникнення надлишковості та дублювання. Нормалізація дозволить:

- підвищити узгодженість даних;
- спростити логіку оновлення інформації;
- зменшити ймовірність логічних помилок при формуванні звітів.

Основні сутності та структура таблиць

У межах проектування бази даних було визначено дві основні сутності, які забезпечують зберігання і обробку інформації про клієнтів банку та їхні фінансові операції: **clients** та **transactions**.

Таблиця clients – Клієнти. Ця таблиця зберігає базову інформацію про кожного клієнта, що є об’єктом фінансового моніторингу. Структура таблиці включає наступні поля:

- **idclient** — унікальний ідентифікатор клієнта (первинний ключ);
- **nameclient** — найменування клієнта (наприклад, ПІБ фізичної особи, назва юридичної особи, ФОП, ТОВ тощо);
- **sanctions** — логічний прапорець, який вказує, чи входить клієнт до санкційного списку (0 — ні, 1 — так).

Таблиця transactions – Транзакції. Таблиця зберігає інформацію про всі фінансові операції, здійснені клієнтами, які можуть бути проаналізовані з метою виявлення підозрілої активності. Структура таблиці:

- **idtransaction** — унікальний ідентифікатор транзакції (первинний ключ);
- **idclient** — зовнішній ключ, що посилається на таблицю clients і вказує, ким була виконана транзакція;
- **sum** — сума фінансової операції в гривнях;

- **type** — тип транзакції, що кодується числовим значенням: 1 — поповнення рахунку, 2 — зняття готівки, 3 — переказ коштів;
- **date** — дата здійснення транзакції, представлена у форматі yyyymm (наприклад, 202503 — березень 2025 року).

Примітка: У розробленій програмі реалізовано саме три типи транзакцій для демонстрації функціоналу аналізу. При необхідності структура таблиці дозволяє розширення для інших типів фінансових дій.

В основу моделювання покладено принципи реляційної моделі даних та узагальнені практики побудови інформаційних систем банківського типу.

2.3 Розробка логіки обчислення балів ризику

Логіка обчислення балів ризику є центральною функціональністю програмного засобу та полягає в автоматизованому аналізі фінансових операцій клієнта відповідно до визначених критеріїв ризику. Кожне виявлене порушення або підозріла поведінка призводить до нарахування певної кількості балів, які в сукупності визначають загальний рівень ризику клієнта.

Основна концепція

Програма оперує набором **правил (формул) оцінки ризику**. Кожне правило містить **бал ризику**, що нараховується за відповідність умові. У таблиці 2.1 зображено правила оцінки

Таблиця 2.1 - Умови та формули за якими нараховуються бали

ID	Опис правила	Умова (критерій)	Бал
1	Зняття готівки понад 200 000 грн	sum > 200000	7

Продовження таблиці 2.1

2	Поповнення рахунку понад 100 000 грн	sum > 100000	5
3	Переказ коштів понад 75 000 грн	sum > 75000	3
4	Клієнт з громадянством РФ	citizenship = 'RU'	20
5	Клієнт є підсанкційною особою	sanctioned = true	20
6	Часті великі транзакції за короткий проміжок часу	>3 транзакції > 100000 за 1 день	7

Примітка: У програмі реалізовано перші 3 правила. Реалізація інших правил (наприклад, частота операцій) потребує додаткової логіки у програмному коді.

2.4 Моделювання алгоритму визначення рівня ризику клієнта

Процес визначення рівня ризику клієнта є завершальним етапом у системі фінансового моніторингу, що інтегрує результати аналізу транзакцій з урахуванням заздалегідь встановлених правил. Його мета — надати зрозумілу, обґрунтовану оцінку ризикованості кожного клієнта для подальших дій з боку банку: автоматичної перевірки, ручного аналізу або повідомлення відповідних органів.

Основна логіка алгоритму

Алгоритм визначення рівня ризику базується на кількісному підході — **сумі балів ризику**, отриманих за результатами застосування правил до клієнта. Алгоритм враховує як характеристики самого клієнта, так і його транзакції. У таблиці 2.2 зображено присвоєння ризиків відповідно кількості балів

Кроки алгоритму:

1. Ініціалізація загального балу ризику.
2. Застосування кожного правила до відповідного аспекту (транзакції або профілю клієнта).
3. Нарахування балів при виконанні умови правила.
4. Обчислення підсумкового балу.
5. Присвоєння клієнту **категорії ризику** згідно з пороговими значеннями.

Таблиця 2.2 - Присвоєння ризиків відповідно кількості балів

Кількість балів	**Присвоєний ризик**
0	Немає ризику
від 1 до 9	Низький ризик
від 10 до 19	Середній ризик
від 20 і більше	Високий ризик

Додатково може впливати на рівень ризику наявність у клієнта зв’язків з підсанкційними клієнтами або країнами, іншими клієнтами класифікованими як високоризикованим, з політично значущими особами, а також повторюваність порушень.

Особливістю реалізації є те, що алгоритм легко модифікується шляхом додавання нових правил.

На рисунку 2.1 візуалізовано алгоритм визначення рівня ризику у форматі блок-схеми

Рисунок 2.1 - Візуалізація алгоритму визначення рівня ризику (блок-схема)

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Вибір технологій розробки (Java, JavaFX, PostgreSQL, NetBeans)

Для реалізації програмної системи оцінки ступеня ризику банківських клієнтів було обрано сучасний стек технологій, що забезпечує стабільність, масштабованість, безпеку та зручність у розробці та супроводі програмного забезпечення. До складу обраного технологічного стеку увійшли: **Java** як основна мова програмування, **JavaFX і Scene Builder** — для побудови графічного інтерфейсу користувача, **PostgreSQL** — як база даних, **pgAdmin** — як система адміністрування баз даних та інтегроване середовище програмування NetBeans IDE, яке забезпечує повний цикл розробки програмного забезпечення..

Java

Java є однією з провідних мов програмування, яка широко використовується для створення надійного програмного забезпечення, зокрема у фінансовому, банківському та корпоративному секторах. На рисунку 3.1 зображено приклад коду на мові Java Її застосування в межах даного проекту обумовлене низкою технічних і організаційних переваг:

- **Широкий спектр бібліотек та фреймворків**. Java має велику кількість стандартних та сторонніх бібліотек, які охоплюють найрізноманітніші аспекти: обробку даних, роботу з базами даних, шифрування, логування, серіалізацію об'єктів тощо. Це дозволяє сконцентруватися на бізнес-логіці, а не на реалізації низькорівневих механізмів.

- **Безпека**. Мова Java включає в себе розвинені механізми контролю доступу, обмеження виконання коду, обробку винятків, що критично важливо при роботі з конфіденційною банківською інформацією. Захист від переповнення буфера, контроль пам’яті, та модульність класів дозволяють знизити ризики вразливостей.
- **Розвинене ком’юніті та підтримка**. Java є надійною платформою для створення масштабованого програмного забезпечення завдяки постійній підтримці компанії Oracle, активності розробницької спільноти та регулярним оновленням мовних і фреймворкових компонентів.
- **Платформна незалежність**. Завдяки JVM (віртуальній машині Java), створені програми можуть працювати на різних операційних системах без перекомпіляції, що спрощує розгортання та зменшує витрати на підтримку.
- **Досвід розробника**. Також вибір Java зумовлено особистим досвідом автора роботи у розробці десктопних застосунків саме цією мовою. Це дозволило значно скоротити час на опанування інструментарію та одразу зосередитись на реалізації бізнес-логіки системи.

```
private void loadClientData() {
    data.clear();

    String clientQuery = "SELECT idclient, nameclient FROM clients";
    String transactionQuery = "SELECT type, sum FROM transactions WHERE idclient = ?";

    try (Connection conn = Database.connect(); Statement clientStmt = conn.createStatement();

        while (clientRs.next()) {
            int id = clientRs.getInt(columnLabel: "idclient");
            String name = clientRs.getString(columnLabel: "nameclient");

            int transactionCount = 0;
            double totalSum = 0;
            int suspiciousCount = 0;
            int points = 0;

            try (PreparedStatement transStmt = conn.prepareStatement(sql: transactionQuery)) {
                transStmt.setInt(parameterIndex: 1, x: id);
                try (ResultSet transRs = transStmt.executeQuery()) {
                    while (transRs.next()) {
                        int type = transRs.getInt(columnLabel: "type");
                        double sum = transRs.getDouble(columnLabel: "sum");

                        transactionCount++;
                        totalSum += sum;

                        boolean isSuspicious = false;
                        int point = 0;
```

Рисунок 3.1 - Приклад коду на мові Java

JavaFX

Для реалізації графічного інтерфейсу користувача в програмному засобі було обрано **JavaFX** — сучасний фреймворк для створення кросплатформних GUI-додатків мовою Java. JavaFX поступово витіснив Swing у якості стандартного інструменту для побудови інтерфейсів у десктопних застосунках завдяки своїй гнучкості, сучасному підходу до проектування та широким можливостям кастомізації.

Що зумовило вибір JavaFX:

- **Повна інтеграція з Java**. JavaFX є частиною Java-екосистеми, що забезпечує тісну взаємодію з бізнес-логікою додатку, бібліотеками та API. Завдяки цьому відпадає потреба в сторонніх фреймворках для зв’язку між інтерфейсом і логікою програми.
- **Можливість створення сучасного UI**. JavaFX підтримує CSS-стилізацію елементів, векторну графіку SVG, шейпінг, анімації, ефекти (тіні, світіння, згладжування), що дозволяє створювати інтерфейси, які виглядають сучасно й професійно, без додаткових бібліотек.
- **FXML — декларативна мова опису інтерфейсу**. FXML дозволяє описувати вигляд вікон у вигляді окремих XML-файлів, що відділяє інтерфейс від логіки, сприяючи чистій архітектурі та зручному редагуванню.
- **Підтримка архітектури MVC**. Платформа **JavaFX** орієнтована на використання моделі **Model-View-Controller**, що сприяє кращій структуризації коду та підвищує гнучкість у розробці й підтримці ПЗ.
- **Гнучкість та масштабованість**. Для спрощення створення інтерфейсів використовується Scene Builder — візуальний редактор, сумісний із NetBeans, який дозволяє компонувати елементи UI методом перетягування без необхідності вручну писати FXML-код.

- **Інструменти розробки**. Для створення інтерфейсу використовується Scene Builder — візуальний інструмент, сумісний із NetBeans, який забезпечує побудову елементів UI методом перетягування без потреби у ручному написанні FXML-розмітки.

Порівняння з бібліотекою Swing

Під час вибору інструментарію для розробки користувацького інтерфейсу було проаналізовано альтернативні підходи, серед яких — бібліотека **Swing**, що історично використовувалась як основний засіб створення графічного інтерфейсу в Java. Проте, попри свою стабільність і сумісність із великою кількістю існуючих рішень, Swing поступається JavaFX за низкою критичних параметрів, що стало визначальним фактором у прийнятті рішення.

Нижче наведено основні аспекти, за якими **JavaFX** є більш сучасним та доцільним вибором:

- **Можливості стилізації**. JavaFX підтримує стилізацію за допомогою **CSS**, що дає змогу створювати привабливі, адаптивні інтерфейси без потреби ручного налаштування кожного компонента. Swing у цьому плані обмежений стандартними виглядами (Look and Feel), які складні в кастомізації та не підтримують гнучке оформлення.
- **Декларативне описання інтерфейсу (FXML)**. JavaFX дозволяє описувати структуру інтерфейсу за допомогою окремих XML-файлів (FXML), що спрощує поділ між логікою і виглядом. У Swing інтерфейс формується виключно програмним шляхом, що ускладнює супровід коду та його тестування.
- **Модернізована подієва модель**. JavaFX реалізує спрощену та логічну подієву модель, яка краще масштабуються при роботі з великою кількістю компонентів. У Swing події обробляються через менш структуровану модель з більшою кількістю коду, що ускладнює підтримку.

- **Анімації, графіка, мультимедіа**. JavaFX навіть у базовій реалізації підтримує анімації, 2D-графіку, векторні ефекти, обробку мультимедіа (аудіо, відео), тоді як у Swing для реалізації подібного функціоналу потрібні сторонні бібліотеки або складне ручне налаштування.
- **Сумісність із сучасними платформами**. Swing, як старіша технологія, має гіршу підтримку на сучасних дисплеях з високою роздільною здатністю, тоді як JavaFX адаптований до таких середовищ з самого початку.
- **Архітектура**. JavaFX краще підтримує архітектурні шаблони на кшталт **MVC**, що є стандартом для якісної, масштабованої розробки ПЗ.

З урахуванням зазначених переваг JavaFX було обрано як основну технологію для створення інтерфейсу, що відповідає вимогам сучасного програмного забезпечення у фінансовій сфері.

На рисунку 3.2 зображено інтерфейс візуального редактора Scene Builder

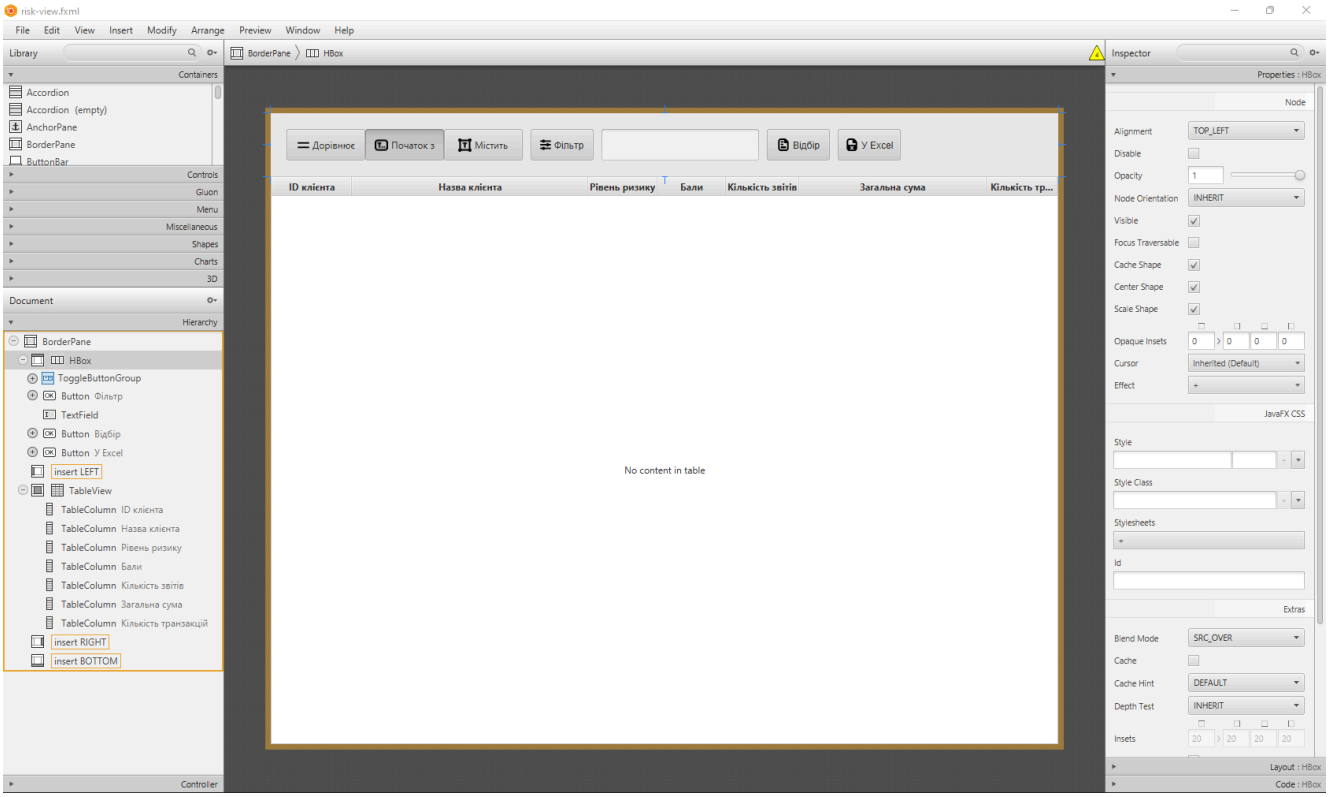


Рисунок 3.2 - Візуальний редактор інтерфейсу Scene Builder

NetBeans як середовище розробки

У межах реалізації програмного засобу було обрано **NetBeans IDE** як основне інтегроване середовище розробки (IDE), орієнтоване на створення Java-застосунків. Це середовище є офіційно підтримуваним проєктом Apache Foundation та забезпечує повноцінний інструментарій для всіх етапів розробки: від написання коду до його компіляції, тестування, налагодження та запуску.

Обґрунтування вибору NetBeans IDE базується на таких функціональних і практичних перевагах:

- **Вбудована підтримка JavaFX**. Інтеграція NetBeans із **JavaFX Scene Builder** забезпечує можливість створення графічного інтерфейсу у візуальному режимі, усуваючи потребу в ручному редагуванні FXML-файлів. Це значно пришвидшує створення інтерфейсу та підвищує зручність у його редагуванні. Крім того, NetBeans автоматично генерує відповідні контролери та забезпечує зв’язок між виглядом і логікою додатку.
- **Робота з базами даних**. У середовище вбудовані модулі для управління реляційними базами даних, включаючи **PostgreSQL**, що дає змогу:
    - переглядати структуру бази даних і її вміст;
    - виконувати SQL-запити безпосередньо з IDE;
    - налагоджувати з’єднання з БД через JDBC;
    - тестувати інтеграцію додатку з базою ще до повної реалізації функціоналу.

- **Підсвітка синтаксису, автодоповнення, рефакторинг**. NetBeans забезпечує контекстне автодоповнення, автоматичне виявлення помилок у коді, а також має вбудовані засоби для швидкого рефакторингу (перейменування змінних, оптимізація імпортів, пошук дублювань тощо). Це дозволяє підвищити продуктивність розробника, зменшити кількість технічних помилок і полегшує підтримку коду.

- **Запуск, налагодження та управління проектом одним кліком**. Завдяки зручному графічному інтерфейсу, розробник має змогу запускати додаток у будь-який момент, керувати конфігураціями виконання та миттєво відслідковувати помилки виконання. Налагоджувач (debugger) дозволяє ставити точки зупинки, переглядати вміст змінних та трасувати хід виконання програми.
- **Підтримка модульної архітектури**. NetBeans дозволяє створювати складні багатофайлові проекти з чіткою структурою. Код можна організувати в пакети, модулі або проекти, що особливо зручно при масштабуванні застосунку або роботі над великим програмним комплексом.
- **Стабільність і простота у використанні**. Незважаючи на існування інших популярних IDE, таких як IntelliJ IDEA або Eclipse, NetBeans залишається простим у налаштуванні та інтуїтивно зрозумілим середовищем, що не потребує встановлення додаткових плагінів для підтримки JavaFX чи роботи з базами даних.

З огляду на зазначені переваги, NetBeans IDE було обрано як оптимальне середовище розробки для створення десктопного програмного забезпечення, орієнтованого на фінансовий моніторинг у банківській сфері. Простота інтерфейсу, глибока інтеграція з JavaFX і підтримка роботи з БД зробили цей вибір доцільним у рамках поставленої задачі. В таблиці 3.1 зображено порівняльна характеристику середовищ для розробки

Таблиця 3.1 - Порівняльна характеристика IDE для розробки Java-додатків

Критерій	**NetBeans IDE**	**IntelliJ IDEA**	**Eclipse IDE**
Підтримка JavaFX	Вбудована, з інтеграцією Scene Builder	Часткова, потребує плагінів, ручного налаштування	Потрібне встановлення окремих модулів та Scene Builder

Продовження таблиці 3.1

Робота з FXML	Повна підтримка "з коробки", інтеграція з JavaFX	Можлива, але обмежена без плагінів або Ultimate-версії	Потрібна стороння інтеграція
Інтеграція з базами даних	Вбудовані інструменти для SQL, перегляду таблиць, підключення	Є лише в Ultimate Edition або через окремі плагіни	Можлива, але через Data Tools + ручне конфігурування
Початкове налаштування	Мінімальне, все працює одразу після встановлення	Потрібні плагіни для JavaFX, ВД, UI	Складна структура, потрібно налаштовувати проекти вручну
Зручність інтерфейсу	Інтуїтивний, дружній до новачків	Продуманий, сучасний інтерфейс	Складніша навігація, перевантажене меню
Продуктивність IDE	Висока стабільність	Висока продуктивність	Важка для старих ПК, іноді зависає при складних проектах

На рисунку 3.3 зображено інтерфейс середовища розробки NetBeans IDE

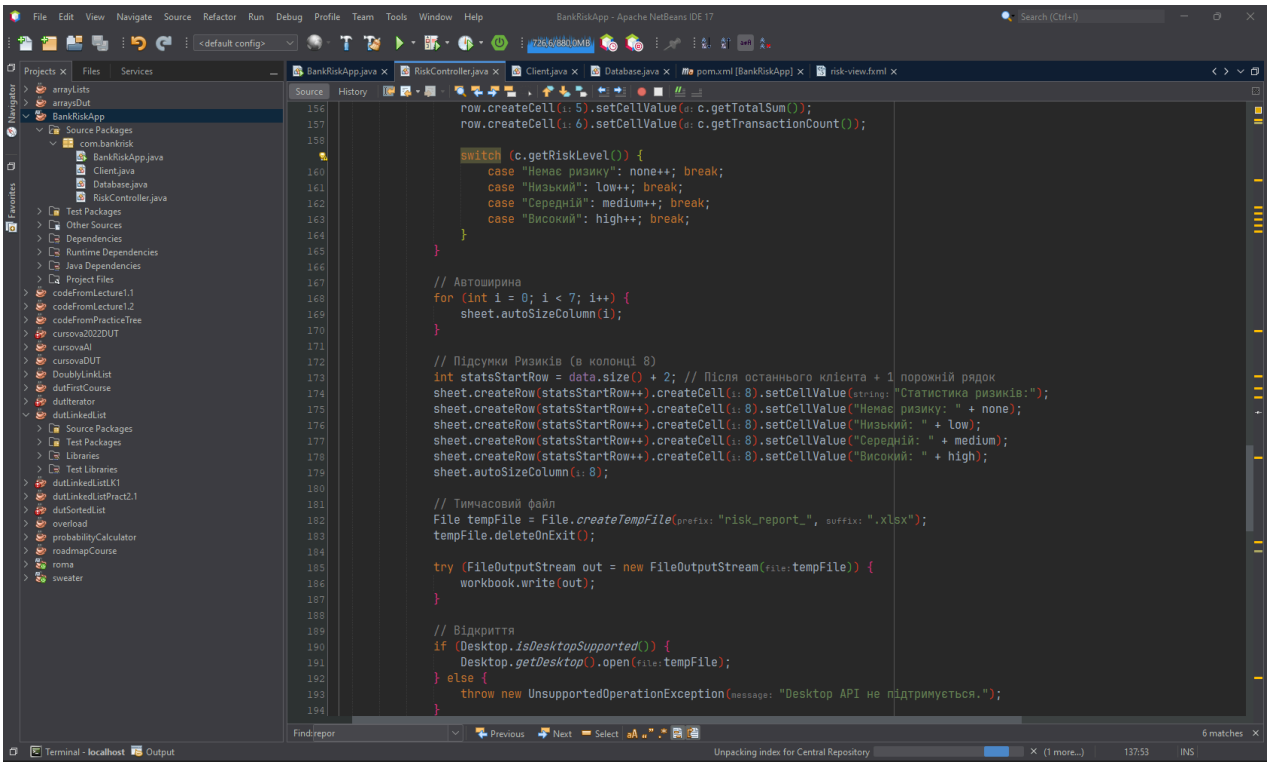


Рисунок 3.3 - Середовище розробки NetBeans IDE

PostgreSQL

Для збереження інформації про клієнтів, правила оцінки ризиків, результати аналізу та інші структуровані дані в програмному засобі використовується **PostgreSQL** — потужна об’єктно-реляційна система управління базами даних (СУБД) з відкритим вихідним кодом.

Основні переваги, що обґрунтовують вибір PostgreSQL:

- **Надійність і стабільність**. PostgreSQL має багаторічну історію промислового використання, зокрема у фінансовій сфері. Її архітектура забезпечує захист цілісності даних, підтримує транзакції (ACID) і механізми відновлення після збоїв.
- **Гнучка система контролю доступу**. За допомогою механізмів ролей та привілеїв можна організувати багаторівневий доступ до даних — до рівня таблиць, колонок або навіть окремих рядків (row-level security), що критично важливо при роботі з конфіденційною банківською інформацією.

- **Інтеграція з Java**. За допомогою стандартного драйвера **JDBC** забезпечується прозора та ефективна взаємодія Java-додатку з PostgreSQL — включаючи підключення, виконання SQL-запитів, обробку результатів і транзакцій.
- **Розширюваність**. Можливість створювати власні типи даних, функції, агрегати, індекси, а також підключати модулі, що робить PostgreSQL гнучким інструментом для розробника.

pgAdmin

Для адміністрування бази даних використовується **pgAdmin** — офіційний графічний інструмент управління PostgreSQL. Це кросплатформенне середовище з відкритим кодом, що надає розробнику потужний візуальний інтерфейс для роботи з БД.

Основні функції **pgAdmin**:

- Інтерактивне створення таблиць, стовпців, рядків, індексів, зв’язків тощо;
- Виконання SQL-запитів у вбудованому редакторі з підсвіткою синтаксису;
- Перегляд структури бази даних у вигляді дерева;
- Моніторинг активних підключень, процесів та запитів;
- Інструменти для резервного копіювання та відновлення;
- Журнали подій та повідомлень про помилки;
- Гнучке керування ролями, правами доступу та політиками безпеки.

pgAdmin значно пришвидшує розробку, налагодження та тестування запитів до БД, особливо у фазах проектування та аналізу даних. Завдяки зручному графічному інтерфейсу, pgAdmin дозволяє візуалізувати структуру бази, легко створювати таблиці, визначати зв’язки між сутностями, а також запускати складні SQL-запити без потреби взаємодії з консоллю. На рисунку 3.4 зображено інтерфейс інструменту управління БД pgAdmin.

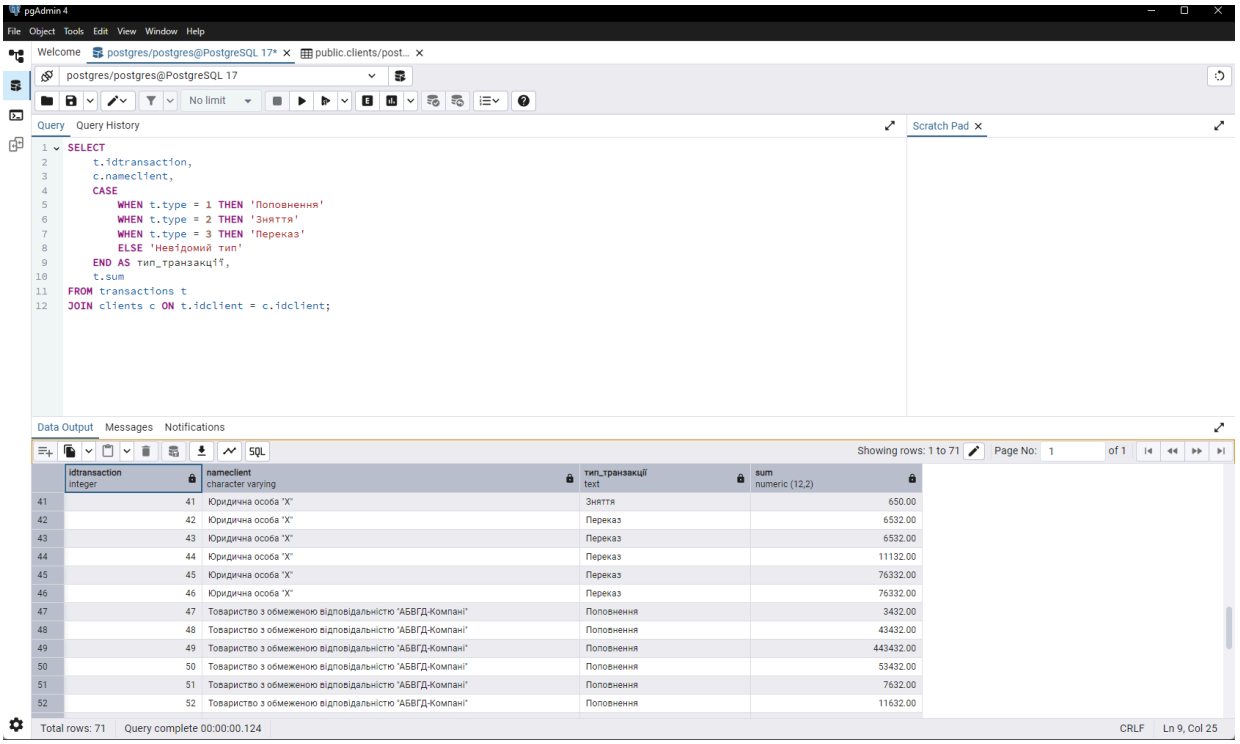


Рисунок 3.4 - Графічний інструмент управління PostgreSQL pgAdmin

3.2 Створення графічного інтерфейсу користувача

Графічний інтерфейс користувача (GUI) є важливою складовою будь-якого прикладного програмного забезпечення, оскільки саме через нього відбувається взаємодія між користувачем і функціоналом системи.

У даному проекті побудова інтерфейсу здійснювалась з використанням фреймворку **JavaFX**, що є сучасним інструментом розробки GUI у середовищі Java. Для візуального конструювання інтерфейсу застосовувався **Scene Builder** — спеціалізований інструмент, що дозволяє створювати FXML-файли без ручного редагування коду, за допомогою drag-and-drop компонування.

Основні принципи реалізації інтерфейсу

Інтерфейс спроектовано відповідно до принципів мінімалізму, інтуїтивної зрозумілості та функціональної чистоти. Він забезпечує користувачеві доступ до ключових операцій без перевантаження елементами та без потреби спеціального навчання.

Головне вікно побудовано на основі компонента **BorderPane**, який забезпечує зручну організацію інтерфейсу за допомогою п’яти зон (верхньої, нижньої, лівої, правої та центральної), що дозволяє логічно розмістити елементи керування, фільтрації та виведення результатів, забезпечуючи зрозумілу та структуровану взаємодію користувача з програмою. На рисунку 3.5 зображено вигляд компонента BorderPane.

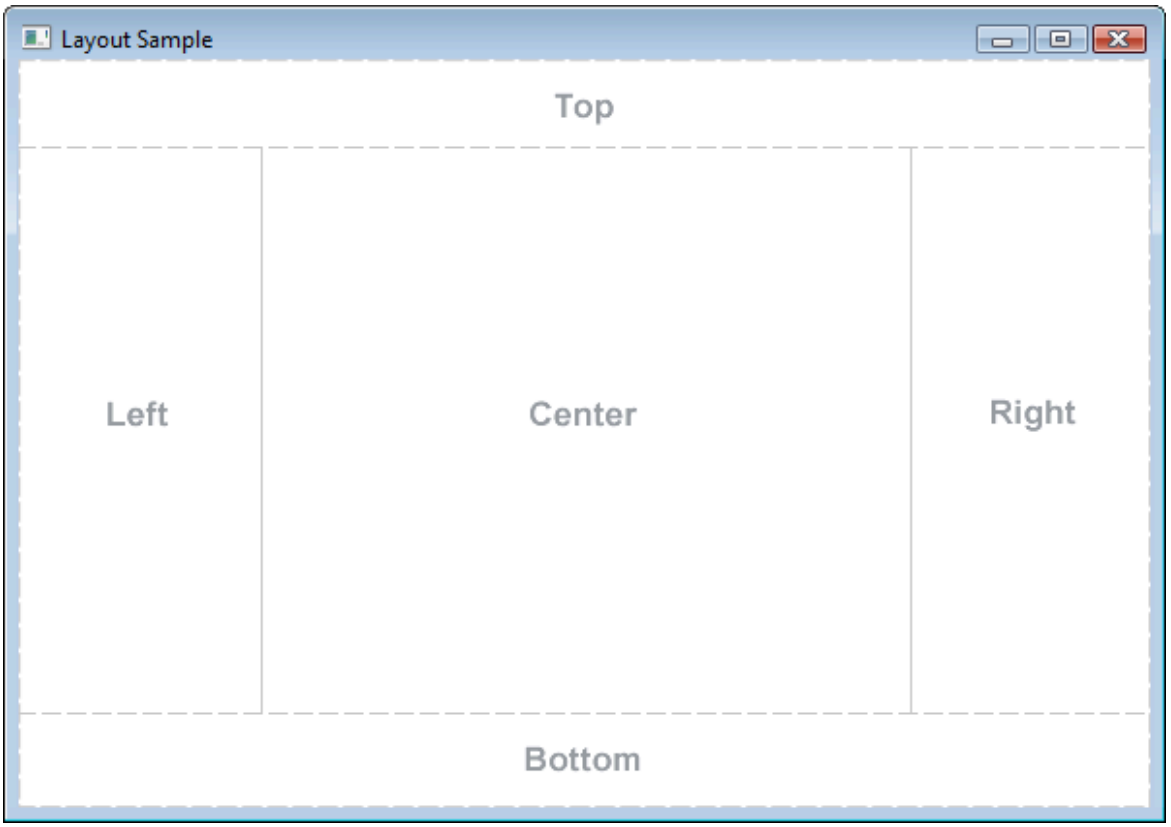


Рисунок 3.5 - Кореневий компонент BorderPane

Стандартний розмір вікна при відкритті програми є 1000 на 800 пікселів. Проте програма має адаптивний інтерфейс, тобто користувач може змінювати

розмір вікна як йому зручно і при цьому програма не буде втрачати вигляд і функціонал. Назва програми (**Title**) — **"Система оцінювання ризику банківських клієнтів"**

У верхній частині головного вікна, в області **Top** компонента **BorderPane**, розміщується **панель керування**, що реалізована за допомогою горизонтального контейнера **HBox**. Ця панель виконує функцію основного засобу взаємодії користувача з системою: саме тут зосереджені елементи фільтрації, кнопки виконання запитів до бази даних та інші елементи управління, які дозволяють формувати запити і керувати відображенням даних у таблиці.

На рисунку 3.6 зображено вигляд панелі керування.

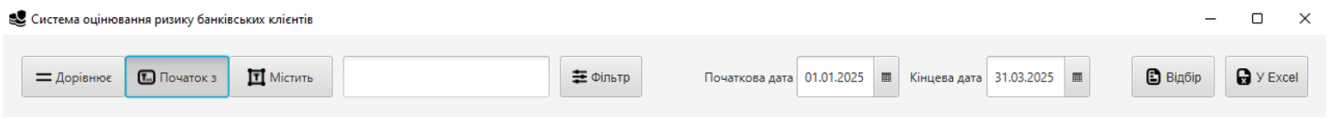


Рисунок 3.6 - Верхня панель керування, реалізована через HBox

На панелі керування розміщено такі елементи:

- **Радіокнопки (RadioButton)** — дозволяють обрати тип фільтрації для пошуку клієнтів. Передбачено, що водночас може бути вибраний один з трьох варіантів:
  - **«Дорівнює»** — точний збіг назви;
  - **«Починається з»** — перевірка початкового фрагмента;
  - **«Містить»** — частковий збіг фрагмента рядка.
- **Текстове поле введення (TextField)** — призначене для введення ключового слова або частини назви клієнта, за якою виконуватиметься фільтрація (пошук).
- **Кнопка «Фільтр»** — активує обраний спосіб фільтрації. При натисканні здійснюється обробка введених даних і оновлення відображуваної інформації.

- Вибір дати (**DatePicker**) з підписом (**Label**) «Початкова дата» — призначене для вибору початкової дати для періоду в якому будуть відбиратися дані.
- Вибір дати (**DatePicker**) з підписом (**Label**) «Кінцева дата» — призначене для вибору кінцевої дати для періоду в якому будуть відбиратися дані.
- **Кнопка «Відбір»** — виконує запит до бази даних, отримуючи актуальні дані про клієнтів та пов’язані з ними транзакції. Результати обробки виводяться в таблицю.
- **Кнопка «У Excel»** — експортує таблицю з даними у файл формату XLSX

Усі кнопки мають **вбудовані іконки**, що реалізовано для підвищення візуальної привабливості та кращої орієнтації користувача в інтерфейсі. Іконки підібрані відповідно до призначення кожної дії (наприклад, маленька таблиця для кнопки «Відбір», знак “=” для радіокнопки «Дорівнює» тощо). Також іконка встановлена для **головного вікна додатку**, що підвищує професійність його вигляду. На рисунку 3.7 зображено вигляд іконок для елементів керування.

Рисунок 3.7 - Приклад іконок для кнопок

Таблиця результатів (TableView)

Центральним елементом інтерфейсу користувача є **таблиця TableView**, яка призначена для візуального представлення аналітичної інформації про клієнтів банку. Таблиця розташована в центральній частині головного вікна (область **Center** у BorderPane) та оновлюється **динамічно** відповідно до результатів запиту, який ініціюється користувачем через кнопку **«Відбір»**.

Після натискання на кнопку «Відбір» програма звертається до бази даних, отримує актуальні дані про клієнтів, обробляє інформацію щодо транзакцій, підраховує загальні суми, бали ризику та кількість ризик-звітів. Отримані дані

передаються до таблиці у вигляді колекції **ObservableList**, яка автоматично синхронізує модель із графічним представленням.

Структура таблиці включає такі стовпці:

- **ID клієнта** – унікальний числовий ідентифікатор клієнта в системі;
- **Назва клієнта** – назва юридичної особи або компанії, ПІБ фізичної особи;
- **Рівень ризику** – текстовий опис класифікації клієнта за рівнем ризику (наприклад, «Середній ризик»);
- **Бали** – числове значення, що відображає загальну кількість балів за вибраний період, нарахованих клієнту згідно з правилами ризику;
- **Кількість звітів** – кількість випадків, коли спрацювали правила ризику;
- **Загальна сума** – сукупна сума всіх операцій клієнта;
- **Кількість транзакцій** – кількість зареєстрованих фінансових операцій клієнта у базі.

Ключові особливості реалізації:

- **Динамічне оновлення.** Всі дані таблиці повністю оновлюються при кожному натисканні кнопки "Відбір". Це дозволяє користувачу отримувати свіжу та релевантну інформацію після змін у базі або при застосуванні нових критеріїв.
- **Реактивна прив’язка (data binding).** Завдяки використанню **ObservableList**, зміни в джерелі даних миттєво відображаються в таблиці без перезавантаження вікна або повторного створення інтерфейсу.
- **Підтримка сортування та автоформатування.** Кожен стовпець може бути відсортований за зростанням або спаданням. Дані у грошових колонках форматуються з двома десятковими знаками та супроводжуються позначкою “грн”.
- **Стилізація.** Клітинки в стовпці ризику стилізуються через кольори відповідно рівня ризику, а саме: в червоний колір – “Високий”, жовтий – “Середній”, зелений – “Низький”, і без кольору для “Немає ризику”.

- **Підтримка динамічної кастомізації.** Для зручності та гнучкості реалізовано, що стовпці таблиці можна змінювати в ширину, щоб користувач міг вибрати зручний йому розмір кожного стовпця.

На рисунку 3.8 зображено вигляд таблиці заповнену трьома тестовими клієнтами.

ID клієнта	Назва клієнта	Рівень ризику	Бали	Кількість звітів	Загальна сума	Кількість транза...
654321	Товариство з обмеженою відповідальністю "Тест-Клієнт"	Високий	28	6	3113352,35 грн	17
987654	Товариство з обмеженою відповідальністю "АБВГД-Компані"	Середній	14	2	1188914,53 грн	14
123456	Юридична особа "Х"	Немає ризику	0	0	604216,29 грн	14

Рисунок 3.8 - Таблиця TableView з трьома тестовими клієнтами

3.3 Реалізація функціоналу аналізу і оцінки ризику

У цьому підрозділі детально розглянуто процес реалізації прикладної логіки аналізу ризикованості клієнтів банку, що є ключовою функціональністю створеної програмної системи. Основна мета цього функціоналу — забезпечення автоматизованого збору, обробки та оцінювання інформації про фінансові операції клієнтів з метою виявлення потенційно ризикових суб’єктів на основі визначених правил і критеріїв.

Функціонал реалізовано з використанням мови програмування Java, з застосуванням бібліотеки JavaFX для побудови графічного інтерфейсу користувача та JDBC (Java Database Connectivity) для взаємодії з базою даних, яка функціонує під управлінням PostgreSQL.

Алгоритм роботи системи охоплює кілька етапів:

1. ініціалізацію та налаштування графічного інтерфейсу;
2. введення користувачем діапазону дат для аналізу (періоду моніторингу);
3. запуск механізму вибірки даних з таблиць clients і transactions;
4. обчислення сукупного ризикового балу за кожним клієнтом;
5. класифікацію клієнтів за рівнями ризику (немає ризику, низький, середній, високий);
6. виведення структурованих результатів до таблиці TableView в інтерфейсі користувача.

У процесі реалізації було забезпечено розділення логіки за шарами: від візуальної взаємодії користувача до обробки даних і доступу до БД. Такий підхід відповідає принципам модульності та полегшує підтримку і масштабування програмного продукту.

Кожен з наведених етапів реалізовано з дотриманням принципів структурованого програмування, з урахуванням безпеки доступу до даних, обробки виключень, а також продуктивності виконання запитів при великій кількості клієнтів і транзакцій. В результаті, програмний засіб дозволяє швидко, надійно та наочно оцінити ступінь ризикованості кожного клієнта відповідно до заданих критеріїв, що істотно підвищує ефективність роботи фахівців банку з питань внутрішнього фінансового моніторингу.

Структура програми

Розроблений програмний продукт має чітко визначену модульну структуру, що відповідає принципам об’єктно-орієнтованого програмування та патерну MVC

(Model–View–Controller). Програма складається з окремих файлів (класів), кожен з яких виконує певну роль у системі. Нижче наведено короткий опис основних компонентів:

1. BankRiskApp.java

Головний клас програми, що є точкою входу в застосунок. Клас успадковує **Application** з JavaFX і реалізує метод **start(Stage primaryStage)**, де виконується:

- завантаження FXML-файлу (**risk-view.fxml**) через **FXMLLoader**;
- ініціалізація сцени (**Scene**) та її відображення у вікні (**Stage**);
- встановлення іконки, заголовка вікна, розмірів та інших параметрів.

2. RiskController.java

Контролер головного вікна інтерфейсу. Пов'язаний із **risk-view.fxml** через атрибут **fx:controller.** Цей клас використовується як модель для заповнення TableView, забезпечує двосторонній зв'язок із GUI через JavaFX Property.. Відповідає за:

- ініціалізацію інтерфейсу (метод **initialize()**), налаштування колонок таблиці;
- обробку подій користувача (натискання кнопок «Дорівнює», «Початок з», «Містить», «Відбір», «Фільтр», «У Excel», вибір дат, поле вводу);
- виконання запитів до бази даних;
- формування та відображення даних у таблиці **TableView<Client>**.

3. Client.java

Клас-модель, що описує сутність клієнта. Містить поля: **idclient, nameclient, transactionCount, totalSum, suspiciousReports, points, riskLevel** та відповідні *геттери.

*геттери – (від англ. get — отримати) — це спеціальні методи в об'єктно-орієнтованому програмуванні, які використовуються для **отримання значення приватного поля класу**. Вони забезпечують **інкапсуляцію**,

дозволяючи читати дані об’єкта, не надаючи прямого доступу до його внутрішньої структури. На рисунку 3.9 зображено реалізацію геттера у Java.

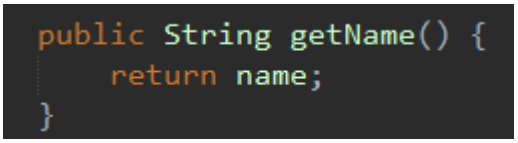


Рисунок 3.9 - Вигляд геттера у Java

Призначення геттерів:

- захист полів від прямої зміни (через **private**);
- контроль доступу (наприклад, можна повертати змінене значення);
- забезпечення зв’язку між моделлю та інтерфейсом (наприклад, у JavaFX — TableView читає дані через геттери).

У дипломному проекті геттери використовуються для відображення даних клієнтів у таблиці та для логіки експорту у звіти.

4. Database.java

Сервісний клас, що відповідає за з’єднання з базою даних PostgreSQL. Містить статичний метод **connect()**, який повертає екземпляр **Connection**.
Централізує параметри доступу до БД (URL, логін, пароль), що дозволяє спростити керування з’єднанням.

5. pom.xml

Конфігураційний файл Maven-проекту. Містить:

- інформацію про залежності (JavaFX, Apache POI для експорту Excel);
- параметри компіляції;
- плагіни для створення jar-файлу або запуску проекту;
- визначення Java-версії, модулів і шляхи до ресурсів.

6. risk-view.fxml

FXML-файл, у якому описана структура графічного інтерфейсу у форматі XML. Містить:

- елементи керування (кнопки, таблицю, текстові поля);
- розташування компонентів у **BorderPane** та **HBox**;
- ідентифікатори (**fx:id**) для зв’язку з контролером **RiskController**.

Внутрішні механізми ініціалізації інтерфейсу JavaFX

Після запуску програми головний клас **BankRiskApp** викликає метод **Application.launch(...)**, який ініціалізує середовище **JavaFX** та створює головне вікно (**Stage**). Далі за допомогою **FXMLLoader.load(...)** завантажується FXML-файл, що створює графічне дерево (scene graph), яке включає вузли (Node), контейнери (Pane, HBox, VBox, BorderPane тощо) і керуючі елементи (Button, TableView, Label тощо).

У момент завантаження FXML-файлу автоматично виконується прив’язка кожного елемента інтерфейсу до відповідного поля класу-контролера **RiskController** за допомогою анотацій **@FXML**. Після цього викликається метод **initialize(URL url, ResourceBundle rb)** — стандартний метод JavaFX-контролера, що слугує точкою входу для логіки ініціалізації.

На рисунку 3.10 зображено у вигляді блок-схеми ініціалізацію програми у системній послідовності.

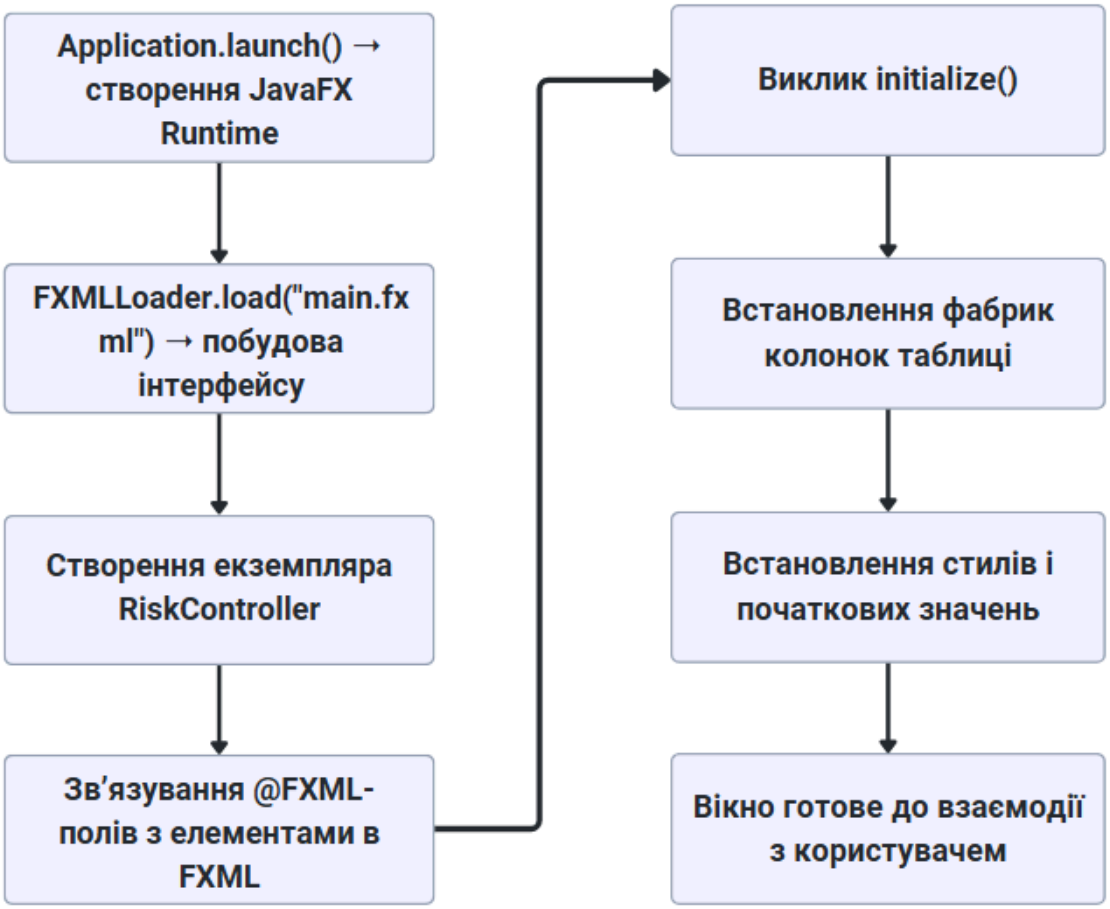


Рисунок 3.10 - Блок-схема: системна послідовність ініціалізації

Таким чином, під час запуску додатка відбувається послідовна ініціалізація інтерфейсу, його зв'язування з логікою керування та підготовка всіх візуальних компонентів до роботи. Такий підхід дозволяє чітко відділити графічний рівень від прикладної логіки та підвищує масштабованість проекту.

Введення користувачем діапазону дат для аналізу (періоду моніторингу)

Один із базових етапів взаємодії користувача з програмним засобом полягає у виборі періоду, за який буде проводитись аналіз фінансових операцій клієнтів. Цей функціонал реалізовано через два елементи типу **DatePicker**, розташовані у верхній частині інтерфейсу. Користувач може обрати початкову та кінцеву дату, що формує часовий інтервал моніторингу.

Призначення та переваги

Можливість задавати діапазон дат дозволяє:

- аналізувати лише актуальні на момент часу дані;
- виконувати вибірку історичних транзакцій;
- уникати перевантаження системи зайвими даними;
- здійснювати покроковий моніторинг клієнтської активності.

Обробка введених дат у програмі

Після того як користувач обрав дати, подальша обробка відбувається в логіці контролера. Програма перетворює введені дати на внутрішній формат **yyyymm** (рік + місяць), що дозволяє легко порівнювати значення з датами транзакцій, які зберігаються у базі даних у такому ж форматі.

Перевірка допустимого періоду

Програма автоматично обчислює кількість місяців між початковою та кінцевою датами. Якщо обраний інтервал перевищує **3 календарні місяці**, система виводить попередження і блокує виконання подальшого аналізу. Це обмеження запроваджено з метою:

- забезпечення швидкої обробки запитів;
- відповідності вимогам короткострокового моніторингу;
- уникнення затримок у роботі з великим обсягом транзакцій.

Обробка діапазону дат: встановлення стандартних значень

У програмному засобі передбачено автоматичне встановлення стандартного діапазону дат моніторингу — **від 1 січня 2025 року до 31 березня 2025 року**. Ці значення задаються програмно під час ініціалізації інтерфейсу у методі **initialize()** контролера. Відповідно, навіть якщо користувач не взаємодіє з елементами вибору дати, аналіз все одно буде виконаний для цього проміжку часу.

Таким чином, випадки неповного або порожнього введення виключено логікою роботи програми. Такий підхід гарантує:

- коректність і стабільність фільтрації даних;
- відсутність помилок при зверненні до бази;
- зручність для користувача — не потрібно вручну вводити дати для базового аналізу.

Користувач, при потребі, може змінити початкову і кінцеву дату вручну, однак у будь-якому випадку перевірка на допустимий період (не більше 3 місяців) залишається обов’язковою.

Взаємодія з логікою аналізу

Заданий діапазон дат передається як умова для подальшої фільтрації транзакцій. Усі фінансові операції, які потрапляють у межі вказаного періоду, будуть враховані у розрахунку:

- загальної суми;
- кількості транзакцій;
- кількості підозрілих випадків;
- сукупного ризик-балу клієнта.

Таким чином, механізм вибору періоду моніторингу не лише підвищує зручність використання програми, але й забезпечує контроль обсягу оброблюваної інформації. Це критично важливо для підтримки продуктивності системи та фокусованого аналізу клієнтів банку у рамках конкретного часово-фінансового вікна.

Запуск механізму вибірки даних з таблиць clients і transactions

Після задання періоду моніторингу користувач натискає кнопку **«Відбір»**, що ініціює механізм отримання та обробки даних з бази даних. Взаємодія з СУБД

PostgreSQL реалізована вручну за допомогою бібліотеки **JDBC (Java Database Connectivity)** — стандартного API Java для роботи з реляційними базами даних.

Алгоритм запуску вибірки

Натискання кнопки «Відбір» викликає метод **loadClientData()**, який виконує наступні послідовні дії:

- **Очищення попередніх результатів.** Перед початком нового аналізу очищається список **ObservableList<Client> data**, який є джерелом для заповнення TableView. Це гарантує, що таблиця відображатиме лише актуальні результати.
- **Формування SQL-запиту до таблиці clients.** Встановлюється з'єднання з базою даних за допомогою класу Database, у якому реалізовано метод **connect()**. Після цього виконується SQL-запит до таблиці clients, який вибирає **ID та ім'я всіх клієнтів**, що зберігаються в системі.

Запит на мові SQL, який дістає дані idclient і nameclient з таблиці clients:

SELECT idclient, nameclient FROM clients;

Кожен клієнт обробляється окремо — до нього буде прив'язана відповідна вибірка з таблиці транзакцій.

Підготовка запиту до таблиці transactions

Для кожного клієнта за його idclient готується окремий параметризований SQL-запит до таблиці transactions, що дозволяє отримати всі транзакції цього клієнта:

SELECT type, sum, date FROM transactions WHERE idclient = ?;

Цей підхід забезпечує захист від SQL-ін'єкцій, дозволяє гнучко обробляти дані та знижує навантаження на систему при великій кількості клієнтів.

Фільтрація транзакцій за періодом

Після отримання всіх транзакцій конкретного клієнта, вони фільтруються відповідно до обраного періоду (**startDate, endDate**). Тільки ті операції, дата яких входить у діапазон, підлягають подальшому аналізу.

Підрахунок основних показників

Для кожного клієнта обчислюються такі значення:

- **Кількість транзакцій**;
- **Загальна сума** всіх транзакцій;
- **Кількість підозрілих операцій** (згідно з умовами);
- **Сума балів ризику**, нарахованих за ці операції.

Ці обчислення виконуються у процесі перебору транзакцій клієнта, і результати зберігаються у об’єкті Client.

Додавання об’єкта Client до списку

На основі зібраної інформації створюється екземпляр класу **Client**, який містить:

- ID клієнта,
- Назву,
- Кількість транзакцій,
- Загальну суму,
- Кількість ризик-звітів,
- Нараховані бали ризику.

Цей об’єкт додається до **ObservableList**, яка зв’язана з таблицею **TableView** у графічному інтерфейсі. Завдяки реактивній природі ObservableList, дані відразу оновлюються в таблиці без необхідності ручного оновлення інтерфейсу.

Обробка помилок

У разі виникнення помилок при роботі з базою даних (наприклад, при втраті з’єднання або неправильному SQL-запиті), передбачено обробку винятків через **try-catch**. Це дозволяє уникнути аварійного завершення програми та повідомити користувача про помилку.

Процес вибірки даних реалізовано за класичною структурою «підключення — запит — обробка — відображення». Програма формує результативну таблицю з актуальною аналітикою по кожному клієнту, що дозволяє швидко отримати оцінку рівня ризику та інші ключові показники. Завдяки поєднанню **JavaFX** і **JDBC** реалізація є надійною, продуктивною та розширюваною в майбутньому.

Обчислення сукупного ризикового балу за кожним клієнтом

Після вибірки та фільтрації транзакцій за вказаним періодом для кожного клієнта здійснюється автоматизований аналіз його фінансової активності з метою виявлення потенційно ризикових операцій. Результатом цього аналізу є **нарахування балів ризику**, які сумуються у **загальний ризиковий бал клієнта**.

Підхід до оцінювання ризику

Оцінка базується на **порогово-бальній моделі**, яка є поширеною практикою у системах фінансового моніторингу. Кожна транзакція клієнта аналізується за фіксованим набором критеріїв, які відповідають конкретним ознакам підозрілої активності. При виявленні таких ознак клієнту нараховується певна кількість балів.

Правила нарахування балів

У програмі реалізовано такі правила:

- **Поповнення рахунку** (тип транзакції = 1) понад **200 000 грн — +7 балів**
- **Зняття готівки** (тип = 2) понад **100 000 грн — +5 балів**

- **Переказ коштів** (тип = 3) понад **75 000 грн — +3 бали**

Кожна транзакція, що відповідає хоча б одному з критеріїв, вважається **підозрілою** і додає відповідну кількість балів до загального результату.

Механізм обробки

Для кожного клієнта система перебирає всі транзакції, що відповідають вибраному часовому діапазону. При кожній ітерації перевіряється:

- тип транзакції;
- її сума;
- відповідність критерію ризику.

У разі відповідності:

- збільшується **лічильник підозрілих операцій**;
- до **сукупного балу (points)** додається визначене значення.

Врахування підсанкційного статусу

Хоча безпосередньо у цьому методі санкційність (sanctions) не впливає на бали, вона використовується у наступному етапі для визначення **підсумкового рівня ризику**. Однак у разі розширення системи санкційний статус може бути інтегрований як ще одне джерело нарахування балів.

Переваги такого підходу

- **Гнучкість:** набір правил легко змінити або розширити (наприклад, додати перевірку на країну отримувача або частоту транзакцій).
- **Масштабованість:** для великої кількості клієнтів підрахунок відбувається швидко і не потребує складної математики.
- **Автоматизація:** вся логіка обчислення реалізована програмно, без втручання користувача.

Візуалізація результату

Після завершення обчислень сукупний ризиковий бал клієнта відображається у відповідному стовпці таблиці **(Бали)**. Значення використовуються для наступного кроку — **визначення рівня ризику** та візуального виділення клієнтів із підвищеним ризиком.

Класифікація клієнтів за рівнями ризику (немає ризику, низький, середній, високий)

Після завершення аналізу транзакцій та обчислення сукупного ризикового балу для кожного клієнта система переходить до **етапу класифікації** — присвоєння відповідного рівня ризику. Це дозволяє автоматично категоризувати клієнтів залежно від їх фінансової поведінки, а також забезпечити **візуальне групування** у графічному інтерфейсі програми.

Ризик-орієнтований підхід

Класифікація реалізована за **порогово-бальною моделлю**, яка ґрунтується на сукупній кількості балів ризику, що були нараховані за підозрілі транзакції клієнта протягом аналізованого періоду. Чим більше балів — тим вищий рівень ризику. Рівні ризику та їхні порогові значення наведені у Таблиці 2.2

Ці пороги обрано умовно, відповідно до внутрішніх політик фінансового моніторингу. У разі потреби їх можна адаптувати або розширити — наприклад, додати проміжні рівні або враховувати додаткові чинники (як-от підсанкційність клієнта).

Реалізація у програмному коді

Логіка присвоєння рівня ризику реалізована безпосередньо у класі-моделі **Client**. Метод **getRiskLevel()** виконує перевірку кількості балів клієнта і повертає один із рядкових результатів: "Немає ризику", "Низький", "Середній", "Високий". Ці значення використовуються:

- для відображення в окремій колонці таблиці;
- для кольорового виділення клітинок (зелене, жовте, червоне) — через встановлення стилю в **CellFactory**.

Візуалізація у таблиці

Колонка "Рівень ризику" (**riskColumn**) у таблиці TableView заповнюється відповідним рівнем, що повертається з моделі клієнта. В залежності від значення, до клітинки застосовується фон:

- **Зелений** — низький ризик;
- **Жовтий** — середній;
- **Червоний** — високий;
- **Без кольору** — якщо ризику немає.

Цей механізм дозволяє **візуально і швидко** ідентифікувати проблемних клієнтів.

Переваги підходу

- **Простота**: класифікація реалізована у вигляді звичайної перевірки порогів.
- **Гнучкість**: порогові значення можна змінити у будь-який момент.
- **Автоматизація**: присвоєння рівня відбувається без участі користувача.
- **Інформативність**: результат відображається не лише текстом, а й кольором.

Автоматична класифікація клієнтів за рівнями ризику дозволяє виявляти потенційно проблемних осіб, пріоритизувати їх для перевірки, а також значно полегшує роботу співробітникам банку завдяки наочному інтерфейсу. Це критичний елемент системи фінансового моніторингу, який забезпечує швидке ухвалення рішень щодо ризикових клієнтів.

Виведення структурованих результатів до таблиці TableView в інтерфейсі користувача

Фінальним етапом логіки аналізу ризику клієнтів є відображення оброблених даних у зрозумілому й структурованому вигляді в інтерфейсі програми. Для цього використовується компонент **TableView** із бібліотеки JavaFX.

Компонент **TableView<Client>** забезпечує:

- динамічне відображення результатів аналізу;
- можливість сортування, фільтрації й оновлення вмісту;
- зручну інтеграцію з моделлю даних (**Client**) через зв’язок з JavaFX-спостережуваними властивостями.

Це дозволяє користувачеві одразу після натискання кнопки **«Відбір»** бачити повну аналітику по кожному клієнту, включно з усіма розрахованими показниками.

Структура таблиці

У таблиці 3.2 показано, які дані відображаються у колонках таблиці TableView та їхнє призначення:

Таблиця 3.2 - Дані, які виводяться у таблицю та їх призначення

Колонка	**Джерело даних**	**Призначення**
ID клієнта	idclient	Унікальний ідентифікатор клієнта
Назва клієнта	nameclient	Найменування фізичної чи юридичної особи
Рівень ризику	getRiskLevel()	Розрахований рівень ризику клієнта
Бали	points	Сума балів за підозрілі операції

Продовження таблиці 3.2

Кількість звітів	suspiciousReports	Скільки разів клієнт потрапив під критерії ризику
Загальна сума	totalSum	Підсумована сума всіх операцій у вказаний період
Кількість транзакцій	transactionCount	Загальна кількість фінансових операцій

Технічна реалізація

Підключення даних до таблиці здійснюється за допомогою JavaFX-колонок (**TableColumn**), кожна з яких прив’язана до відповідного поля моделі **Client**. Після обробки всіх клієнтів дані додаються до **ObservableList**, яка автоматично оновлює вміст таблиці без перезапуску інтерфейсу.

Це досягається викликом:

tableMain.setItems(data);

де **data** — список усіх клієнтів з розрахованими показниками.

Візуальне оформлення та інтерактивність

Для підвищення зручності сприйняття реалізовано:

- **Форматування числових даних.** Наприклад, суми виводяться з округленням до двох знаків після коми та з додаванням гривні (наприклад, 125000.00 грн).
- **Кольорове виділення рівня ризику.** Колонка «Рівень ризику» має умовне форматування по кольорам.

- **Автоматичне оновлення.** Завдяки ObservableList зміна списку клієнтів миттєво оновлює таблицю без необхідності перезапуску або ручного оновлення.

Переваги такого підходу

- **Інформативність:** користувач бачить усі ключові показники в одному вікні.
- **Реактивність:** результат аналізу миттєво відображається після натискання кнопки.
- **Масштабованість:** можна легко додати нові колонки або змінити логіку сортування.
- **Зручність для подальшого експорту:** структура таблиці прямо відповідає формату Excel-звіту, що генерується окремо.

Виведення результатів аналізу в таблицю **TableView** є ключовим етапом у зручності використання програмного засобу. Воно дозволяє працівникам банку швидко оцінити фінансову поведінку клієнтів, виявити потенційно ризикових осіб та приймати обґрунтовані рішення в рамках системи фінансового моніторингу.

3.4 Тестування, перевірка правильності оцінювання клієнтів

Після реалізації основного функціоналу програмного засобу було проведено тестування з метою перевірки коректності роботи системи, зокрема — правильності обчислення ризикового балу клієнтів, класифікації за рівнем ризику та відповідності відображених результатів очікуваній логіці.

Мета тестування

Основною метою тестування було переконатися, що:

- система правильно інтерпретує типи транзакцій;

- бали ризику нараховуються згідно з правилами;
- фільтрація за періодом працює коректно;
- класифікація клієнтів відбувається відповідно до накопичених балів;
- дані відображаються у таблиці без помилок і затримок.

Типи проведеного тестування

1. **Модульне тестування (unit testing)** — перевірка окремих методів на коректність логіки нарахування балів, фільтрації транзакцій та присвоєння рівня ризику.
2. **Інтеграційне тестування** — перевірка взаємодії між модулями Client, Database, RiskController та базою даних.
3. **Функціональне тестування** — перевірка всього процесу від введення даних до виведення результату в інтерфейсі.
4. **Ручне тестування** — перевірка типових і граничних сценаріїв за участі тестових даних.

Підготовка тестових даних

У таблицю **clients** були внесені умовні клієнти з унікальними ідентифікаторами. Для кожного клієнта в таблиці **transactions** сформовано набір транзакцій, які:

- відповідають або не відповідають пороговим значенням для ризику;
- містять різні типи операцій (1, 2, 3);
- охоплюють різні дати — як у межах дозволеного періоду;

Це дало змогу перевірити, чи правильно:

- ігноруються транзакції за межами періоду;
- розраховуються суми;
- підозрілі транзакції впливають на бали.

Результати тестів підтвердили:

- **Правильність фільтрації** — жодна транзакція поза обраним періодом не враховується в підсумках.
- **Коректність нарахування балів** — всі правила (7/5/3 балів відповідно до типу і суми транзакції) виконуються точно.
- **Стабільність системи** — при великій кількості записів таблиця не зависає, дані оновлюються швидко.
- **Правильна класифікація** — клієнти з однаковими балами потрапляють в однакові категорії ризику, відповідно до логіки:
    - 0 балів — «Немає ризику»
    - 1–9 — «Низький»
    - 10–19 — «Середній»
    - 20+ — «Високий»

Приклади успішного тестування

- Клієнту, який здійснив три перекази по 80 000 грн (тип = 3), було нараховано 9 балів.

  Результат — **ризик: низький**.

- Клієнту, який здійснив зняття готівки на 120 000 грн (тип = 2) і переказ 90 000 грн (тип = 3), було нараховано 5 + 3 = 8 балів.

  Результат — **ризик: низький**.

- Клієнту, який виконав 2 великі зняття і 1 переказ — 5 + 5 + 3 = 13 балів.

  Результат — **ризик: середній**.

- Клієнт із сумарним балом 22 отримав статус **«Високий ризик»**, що відповідає моделі.

Висновки за результатами тестування

Програмний засіб успішно пройшов тестування, продемонструвавши відповідність очікуваній логіці аналізу фінансових операцій. Алгоритми нарахування балів і класифікації ризику працюють стабільно та надійно. Система здатна ефективно обробляти різні сценарії поведінки клієнтів, що дозволяє використовувати її для підтримки процесів внутрішнього фінансового моніторингу в банківських установах.

ВИСНОВКИ

У межах кваліфікаційної роботи на тему «Система оцінки ступеня ризику банківських клієнтів на базі Java» було розроблено прикладне програмне забезпечення, яке дозволяє автоматизувати процес оцінки ризикованості клієнтів банку на основі аналізу їх фінансових операцій. У процесі виконання роботи було досягнуто наступних результатів:

- проведено теоретичний аналіз існуючих підходів до оцінки банківських ризиків, зокрема методів класифікації клієнтів, порогово-бальних систем та вимог до програмних засобів у сфері фінансового моніторингу;
- визначено вимоги до програмної системи, спроектовано структуру бази даних, що включає таблиці клієнтів та транзакцій, з урахуванням нормалізації та забезпечення зв’язності даних;
- реалізовано графічний інтерфейс користувача за допомогою фреймворку JavaFX з Scene Builder, що забезпечує зручність взаємодії з програмою;
- створено функціонал для аналізу клієнтських транзакцій, обчислення ризикових балів згідно з заданими правилами та класифікації клієнтів на категорії: немає ризику, низький, середній, високий;
- забезпечено можливість фільтрації даних, експорту результатів у формат Excel та візуалізацію ключових параметрів ризику у таблиці TableView;
- проведено тестування роботи програмного засобу на основі умовних даних, що підтвердило його коректність, функціональність і придатність до використання у банківських установах для підвищення ефективності внутрішнього фінансового моніторингу.

Таким чином, поставлені цілі та задачі кваліфікаційної роботи повністю виконані, а розроблений програмний засіб може бути використаний як основа для впровадження аналітичної системи моніторингу клієнтів у фінансових організаціях.

ПЕРЕЛІК ПОСИЛАНЬ

Нормативно-правові акти

1. Закон України «Про запобігання та протидію легалізації (відмиванню) доходів, одержаних злочинним шляхом...» № 361-IX від 06.12.2019 [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/361-20
2. Про затвердження Положення про здійснення фінансового моніторингу банками : постанова НБУ № 65 від 19.05.2020 [Електронний ресурс]. – Режим доступу: https://bank.gov.ua/ua/legislation/Resolution_19052020_65

Книги

3. Єпіфанов А. О. Управління ризиками банків : навч. посіб. / А. О. Єпіфанов. – Суми : ДВНЗ "УАБС НБУ", 2010. – 283 с.
4. Introduction to Programming Using Java. – 8th ed. – 2019. – 699 с.
5. The Art of PostgreSQL. – 2022. – 450 с.
6. Banks and Fintech on Platform Economies. – Wiley, 2021. – 272 с.
7. Head First Java. – 2nd ed. – O'Reilly Media, 2005. – 720 с.
8. Java: The Complete Reference. – 11th ed. – McGraw-Hill Education, 2018. – 1882 с.
9. Beginning NetBeans IDE for Java Developers. – Apress, 2015. – 253 с.
10. Murach's Beginning Java with NetBeans. – Mike Murach & Associates, 2015. – 660 с.
11. JavaFX in Action. – Manning Publications, 2009. – 384 с.
12. “Регулювання ринку розробки програмного забезпечення”. – BRDO, 2017. – 43 с.

Наукові статті / дослідження

13. Developing Scalable Financial Software Applications to Drive Digital Transformation in Banking and Investment // Vol. 9, Issue 3. – 2025. – С. 348–359.
14. Adoption of Design Thinking, Agile Software Development and Co-creation: A Qualitative Study towards Digital Banking Innovation Success // ResearchGate. – 2022. – Vol. 12. – 12 с.

Електронні ресурси

15. Oracle. Java Platform, Standard Edition Documentation [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/javase/8/docs/
16. OpenJFX. JavaFX Documentation [Електронний ресурс]. – Режим доступу: https://openjfx.io/
17. FATF. Risk-Based Approach Guidance for the Banking Sector [Електронний ресурс]. – Режим доступу: https://www.fatf-gafi.org/en/publications/Fatfrecommendations/Fatf-recommendations.html
18. Apache Software Foundation. Apache POI – the Java API for Microsoft Documents [Електронний ресурс]. – Режим доступу: https://poi.apache.org/
19. PostgreSQL. Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/
20. Apache NetBeans. Documentation [Електронний ресурс]. – Режим доступу: https://netbeans.apache.org/kb/docs/

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

Дипломна робота

На тему: Система оцінки ступеня ризику банківських клієнтів на базі Java

Виконав студент групи КНД-42
Самусевич Іван

Актуальність даної роботи

У сучасних умовах банки щодня стикаються з фінансовими загрозами: відмиванням коштів, шахрайством, переказами через підсанкційні канали.

Ручна перевірка клієнтів — неефективна. Тому зростає потреба в автоматизованих рішеннях, які здатні оперативно аналізувати транзакції та визначати рівень ризику клієнта.

Розроблений програмний засіб вирішує цю задачу: він спрощує моніторинг, зменшує людський фактор і підвищує безпеку банківської системи.

🎯 **Ціль дослідження:**

Розробити програму, яка автоматично аналізує фінансові операції клієнтів банку та визначає ступінь їхнього ризику на основі заданих правил.

🛠 **Основні задачі:**

- Провести аналіз методів оцінки фінансових ризиків.
- Сформувати правила для виявлення підозрілих транзакцій.
- Спроектувати структуру бази даних.
- Реалізувати інтерфейс користувача.
- Реалізувати логіку нарахування балів ризику.
- Провести тестування програми.

Предмет і об'єкт дослідження

🔎 **Об'єкт дослідження:**

Інформаційні процеси, пов'язані з аналізом фінансових операцій клієнтів банку.

🎯 **Предмет дослідження:**

Методи та програмні засоби виявлення підозрілих транзакцій і класифікації клієнтів за рівнем фінансового ризику.

💻 Технології розробки:

Java — основна мова програмування, реалізація бізнес-логіки.

JavaFX — побудова графічного інтерфейсу користувача.

Scene Builder — візуальне створення UI без ручного кодування FXML.

PostgreSQL — система управління базами даних для зберігання клієнтів та транзакцій.

JDBC — взаємодія Java-додатку з базою даних.

NetBeans — середовище розробки з підтримкою JavaFX.

Apache POI — для експорту звітів у формат Excel.

Алгоритм оцінювання клієнтів

Вибір періоду аналізу:
Користувач задає діапазон дат.

Завантаження даних:
Програма витягує з бази даних інформацію про клієнтів і їх транзакції.

Аналіз транзакцій:
Кожна операція перевіряється на відповідність умовам підозрілості:

- великі суми зняття/поповнення/переказу;
- частота операцій;
- підсанкційність клієнта.

Нарахування балів:
За кожну підозрілу операцію нараховуються бали (від 3 до 15), які сумуються.

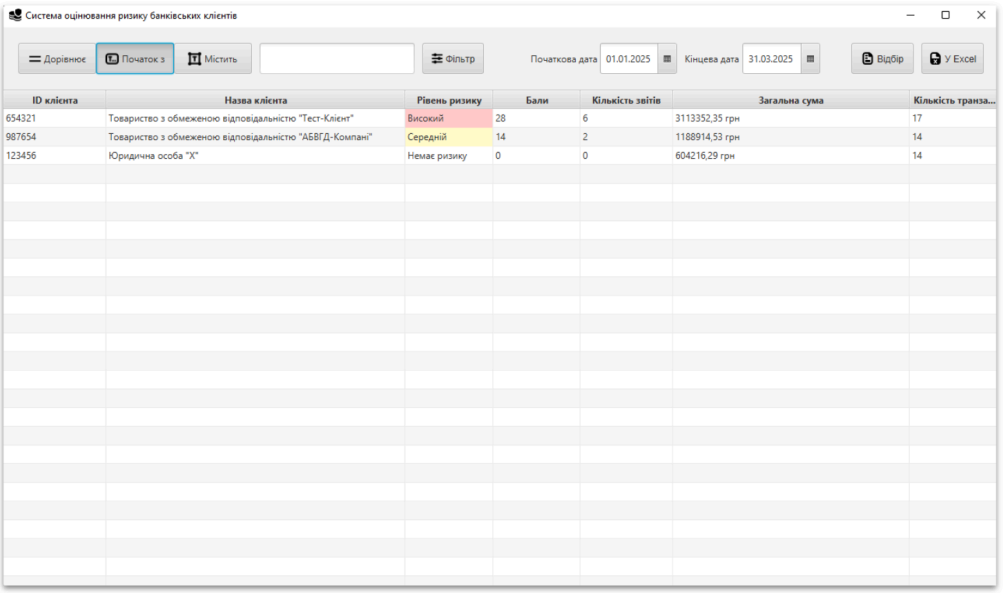
Формування рівня ризику:
Загальна кількість балів визначає рівень ризику клієнта:

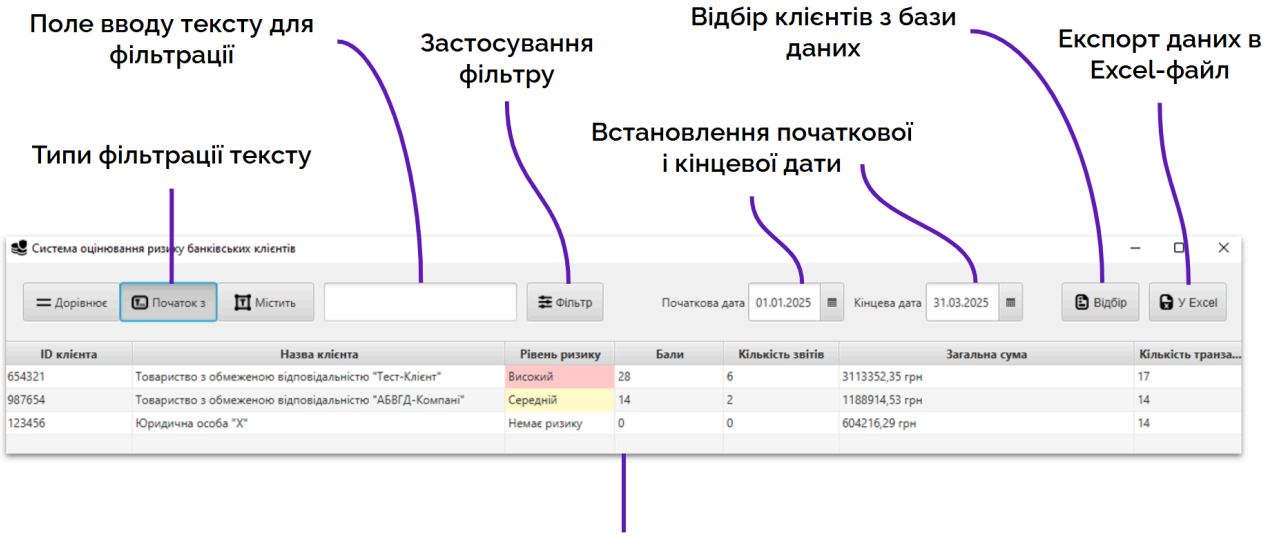
- **0 — Немає ризику**
- **1–9 — Низький**
- **10–19 — Середній**
- **20+ — Високий**

Вивід результатів:
Дані відображаються в таблиці, користувач бачить:

- Назву клієнта, ID клієнта, бали, рівень ризику, кількість транзакцій і звітів, загальну суму.

Загальний вигляд програми





✅ Тестування і перевірка коректності

Мета тестування:
Перевірити, чи програма правильно аналізує транзакції і класифікує клієнтів за рівнем ризику.

Проведені дії:

- Введено тестові дані клієнтів і транзакцій різних типів.
- Перевірено відповідність нарахованих балів умовам правил.
- Протестовано граничні значення (наприклад: 200 000 грн, 100 000 грн).
- Оцінено точність класифікації по балам.

Результат:

- Бал ризику нараховується коректно.
- Клієнти з великою кількістю підозрілих операцій правильно потрапляють до високого рівня ризику.
- Інтерфейс оновлюється автоматично після натискання кнопки "Відбір".
- Фільтрація і експорт в Excel працюють без помилок.

📌 Висновки

🔹 Розроблено десктопний програмний засіб для банківських установ, що дозволяє автоматизовано оцінювати рівень ризику клієнтів на основі їх фінансових операцій.

🔹 Реалізовано інтерфейс на JavaFX з підтримкою фільтрації, звітів і візуального відображення ризику.

🔹 Впроваджено алгоритм нарахування балів за підозрілі транзакції та класифікацію клієнтів на чотири рівні ризику.

🔹 Програма коректно обробляє дані з бази PostgreSQL і підтримує експорт результатів у формат Excel.

🔹 Результати тестування підтвердили стабільну роботу і правильність логіки аналізу.