БАКАЛАВРСЬКА

РОБОТА

БР. ІП - 45.00.00.000 ПЗ

Група ІП-22-2

Печинський Ростислав

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Печинський Ростислав Євгенович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Розроблення програмної системи з інтегрованою аналітикою

(назва роботи)

Інженерія програмного забезпечення
(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Печинський Р.Є._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____Яцишин Микола Миколайович, к.т.н., доцент_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Печинський Ростислав Євгенович
(прізвище, ім’я, по-батькові)

1. Тема проєкту (роботи) "Розроблення програмної системи з інтегрованою аналітикою"

керівник проєкту (роботи) Яцишин Микола Миколайович , к.т.н., доцент

затверджені наказом вищого навчального закладу від **“** 21 ” квітня 2026 р. № 179/7

2. Строк подання студентом проєкту (роботи) 15 червня 2026р.

3. Вихідні дані до проєкту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

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

1. Вступ до проблеми
2. Огляд існуючих концепцій , рішень та сервісів в даній області
3. Побудова моделі або алгоритму власного рішення
4. Документування реалізації власного оригінального рішення вибраними засобами

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Схема розподілу потоків в асинхронній архітектурі програми (рис. 1.1, ст. 18)
2. Діаграма взаємодії основних операційних контурів CRM-системи (рис. 2.1, ст. 23)
3. Блок-схема роботи RBAC-перехоплювача безпеки в CEA SYSTEM (рис. 3.1, ст. 32)
4. Схема функціонування пулу з’єднань у DatabaseConfig (рис. 3.2, ст. 34)
5. Схема реактивного оновлення інтерфейсу на основі патерну Observer (рис. 3.3, ст. 43)

6. Консультанти розділів проєкту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **10 січня 2026 р.**

Керівник: _____ Яцишин М.М.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання: _____ Печинський Р.Є.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту	Примітка
1	Визначення та обґрунтування теми роботи	27.04.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	11.05.2026	виконано
3	Побудова моделі або алгоритму власного рішення	24.05.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	01.06.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	10.06.2026	виконано

Студент _____ Печинський Р.Є.
(підпис) (розшифровка підпису)

Керівник роботи _____ Яцишин М.М.
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 86 сторінок, 16 рисунків, 1 таблиць, 1 додаток, 35 бібліографічних найменувань.

Метою роботи є розроблення автономної системи комерційного обліку та моніторингу торговельних підприємств з використанням Java, JavaFX та СУБД PostgreSQL.

Об'єктом дослідження є процес комерційного обліку та аналітичного моніторингу ефективності персоналу в ритейлі.

Предметом дослідження є програмні засоби автоматизації розрахункових операцій та асинхронного обчислення бізнес-метрик.

У першому розділі розглянуто теоретичні основи побудови інформаційних систем та обґрунтовано доцільність розробки індивідуального модульного моноліту.

У другому розділі проаналізовано бізнес-процеси предметної області та сформовано вимоги до програми.

У третяму розділі спроєктовано архітектуру N-Tier, кастомний IoC-контейнер, рольову модель доступу та схему бази даних.

У четвертому розділі описано реалізацію коду CEA SYSTEM, транзакційного шару JDBC, двигуна KPI та теплової карти JavaFX.

У п'ятому розділі проведено модульне й інтеграційне тестування логіки кошика та шару даних за допомогою JUnit 5.

Висновки роботи містять результати розроблення. Доведено, що стек JavaFX та PostgreSQL є ефективним для створення високопродуктивного софту.

Отримані результати включають систему CEA SYSTEM, спроєктовану базу даних, каркас впровадження залежностей та графічний інтерфейс.

КЛЮЧОВІ СЛОВА: CEA SYSTEM, JAVAFX, POSTGRESQL, КОМЕРЦІЙНИЙ ОБЛІК, АНАЛІТИКА, KPI МЕНЕДЖЕРІВ, ТЕПЛОВА КАРТА.

ANNOTATION

The bachelor's thesis contains 86 pages, 16 figures, 1 table, 1 appendix, and 35 bibliographic references.

The **aim** of the thesis is to develop an autonomous system for commercial accounting and monitoring of retail enterprises using Java, JavaFX, and the PostgreSQL DBMS.

The **object of research** is the process of commercial accounting and analytical monitoring of staff efficiency in retail.

The **subject of research** is software tools for automating settlement operations and asynchronous calculation of business metrics.

The **first chapter** covers the theoretical foundations of building information systems and justifies the feasibility of developing a custom modular monolith.

The **second chapter** analyzes the business processes of the subject area and defines the software requirements.

The **third chapter** focuses on designing the N-Tier architecture, a custom IoC container, a role-based access control model, and the database schema.

The **fourth chapter** describes the code implementation of the CEA SYSTEM, the JDBC transactional layer, the KPI engine, and the JavaFX heatmap.

The **fifth chapter** presents the unit and integration testing of the shopping cart logic and the data layer using JUnit 5.

The **conclusions** of the thesis summarize the development results. It is proven that the JavaFX and PostgreSQL stack is efficient for creating high-performance software.

The **obtained results** include the CEA SYSTEM, the designed database, the dependency injection framework, and the graphical user interface.

KEYWORDS: CEA SYSTEM, JAVAFX, POSTGRESQL, COMMERCIAL ACCOUNTING, ANALYTICS, MANAGER KPIS, HEATMAP.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬПОМИЛКА! ЗАКЛАДКУ НЕ ВИЗНАЧЕНО.
ВСТУП 10
РОЗДІЛ 1. АНАЛІЗ БІЗНЕС-ПРОЦЕСІВ ТА СУЧАСНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМЕРЦІЙНОГО ОБЛІКУ 13
1.1. Операційні процеси підприємства торгівлі та проблеми інтеграції даних.. 13
1.2. Порівняльний аналіз існуючих програмних продуктів на ринку ритейлу .. 14
1.3. Обґрунтування вибору платформи JavaFX 21 та асинхронної архітектури додатка 16
1.4. Конфігурування проєкту за допомогою системи збірки Apache Maven 19
1.5. Висновки по розділу 20
РОЗДІЛ 2. СИСТЕМНИЙ ЗБІР, ОБРОБКА ТА КЛАСИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ 22
2.1. Організація взаємодії з базою даних через патерн Repository та вимоги ACID 22
2.2. Специфікація функціональних та інженерно-технічних вимог до системи 23
2.3. Висновки по розділу 25
РОЗДІЛ 3. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ CEA SYSTEM 27
3.1. Загальна архітектура системи 27
3.2. Проєктування власного IoC-контейнера та механізмів інверсії залежностей 28
3.3. Проєктування доменного рівня 30
3.4. Проєктування рівня даних 33
3.4.1. Архітектура пулу з'єднань 33
3.4.2. Забезпечення цілісності комерційних даних та транзакційна модель ACID 34
3.4.3. Проєктування Omnichannel-запитів до Центрального Складу 35
3.5. Проєктування сервісного рівня 36
3.5.1. Математична модель операційного прогнозування Run Rate 36
3.5.2. Функція аналізу упущеної вигоди та воронка конверсії (CR) 37
3.5.3. Модель предиктивного планування Smart-Plan та декомпозиція KPI 38
3.5.4. Динамічна мотиваційна матриця (Розрахунок бонусів та ЗП) 39
3.5.5. Модуль крос-аналізу глибини кошика (Метрика КПЧ) 40
3.5.6. Omnichannel-логістики та Центрального Складу 40
3.6. Проєктування архітектурних патернів Facade та Observer 41
3.6.1. Патерн Facade (Фасад) та інкапсуляція аналітичних обчислень 41
3.6.2. Патерн Observer (Спостерігач) та реактивна синхронізація станів екранів 42

3.7. Проєктування UI-рівня 43

3.8. Висновки по розділу 46
РОЗДІЛ 4. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ОРГАНІЗАЦІЯ ПРОГРАМНОГО КОДУ 48
4.1. Реалізація IoC-контейнера DIContainer..... 48
4.2. Реалізація доменного рівня 49
4.3. Реалізація рівня даних 51
4.3.1. DatabaseConfig — транзакційне керування пулом підключень 51
4.3.2. Потокобезпечне збереження замовлень та пакетна обробка..... 52
4.4. Реалізація сервісного рівня 53
4.4.1. Алгоритмічна реалізація та асинхронна оптимізація обчислень 53
4.4.2. Двигун розрахунку персональних KPI та мотиваційної матриці..... 55
4.4.3. Сервісна логіка касових операцій, бухгалтерії та системних налаштувань 56
4.5. Реалізація UI-рівня..... 57
4.5.1. Потокобезпечне асинхронне оновлення віджетів..... 58
4.5.2. CustomCellFactory — механізм рендерингу теплової карти..... 59
4.5.3. Корпоративна CSS-стилізація та усунення графічних дефектів..... 60
4.5.4. Інженерні рішення щодо оптимізації рендерингу та адаптації шрифтів .. 61
4.5.5. Архітектура динамічної зміни тем оформлення (Dark Mode / Light Mode 61
4.6. Висновки по розділу 63
РОЗДІЛ 5. ВЕРИФІКАЦІЯ, СИСТЕМНЕ ТЕСТУВАННЯ ТА ОЦІНКА ГОТОВОГО РІШЕННЯ..... 66
5.1. Тестування програмної системи та верифікація бізнес-логіки 66
5.2. Модульне тестування аналітичних обчислень ядра 68
5.3. Інтеграційне тестування транзакційної стійкості шару даних 69
5.4. Модульне тестування бізнес-правил та регулярних виразів кошика 71
5.5. Комплексне інтеграційне тестування наскрізних звітів та кошика..... 73
5.6. Висновки по розділу 75
ВИСНОВКИ 77
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 79
ДОДАТКИ
БІБЛІОГРАФІЧНА ДОВІДКА

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

RBAC	***Role-Based Access Control*** — керування доступом на основі ролей (модель розмежування прав доступу користувачів до функцій системи)
DML	***Data Manipulation Language*** — мова маніпулювання даними (підмножина синтаксису SQL для вибірки, додавання та оновлення записів у БД)
API	***Application Programming Interface*** — прикладний програмний інтерфейс
ACID	***Atomicity, Consistency, Isolation, Durability*** — набір властивостей, що гарантують надійність виконання транзакцій у СУБД
БД	База даних
CEA	***Commercial Enterprise Analytics*** — аналітика комерційного підприємства (назва розроблюваного пз)
CI/CD	***Continuous Integration / Continuous Deployment*** — безперервна інтеграція та доставка програмного забезпечення
CRM	***Customer Relationship Management*** — система управління відносинами з клієнтами
CRUD	***Create, Read, Update, Delete*** — базові чотири операції для роботи з даними (створення, читання, оновлення, видалення)
CSS	***Cascading Style Sheets*** — каскадні таблиці стилів
DBMS	***Database Management System*** — система управління базами даних (міжнародний аналог СУБД)
DI	***Dependency Injection*** — ін'єкція залежностей
ERP	***Enterprise Resource Planning*** — система планування ресурсів підприємства
Fkey	***Foreign Key*** — зовнішній ключ у реляційних базах даних
GPU	***Graphics Processing Unit*** — графічний процесор
GUI	***Graphical User Interface*** — графічний інтерфейс користувача
IDE	***Integrated Development Environment*** — інтегроване середовище розробки
IS	***Information System*** — інформаційна система
JDBC	***Java Database Connectivity*** — API підключення до баз даних Java
JDK	***Java Development Kit*** — набір засобів розробника Java
JUnit	Сучасний фреймворк для організації та автоматизованого запуску модульних і інтеграційних тестів на платформі Java
JVM	***Java Virtual Machine*** — віртуальна машина Java
POM	***Project Object Model*** — концептуальна модель опису та конфігурації Maven-проєкту
ПЗ	Програмне забезпечення
SQL	***Structured Query Language*** — мова структурованих запитів до реляційних баз даних

ВСТУП

Теперішній етап розвитку світової та вітчизняної економіки характеризується стрімкою цифровізацією commercial-сектору, посиленням конкурентного середовища та підвищенням вимог до операційної швидкодії торговельних підприємств. У таких умовах ефективне управління внутрішніми бізнес-процесами, оптимізація взаємодії з клієнтською базою та наявність інструментів миттєвого фінансово-калькуляційного аналізу стають критично важливими факторами виживання та масштабування комерційних структур. Більшість існуючих на ринку CRM та ERP систем мають низку суттєвих недоліків, серед яких висока вартість ліцензування, складність інтеграції з локальними базами даних, надмірна вимогливість до апаратних ресурсів робочих станцій, а головне — відсутність гнучких засобів швидкої кастомізації.

Стандартні системи автоматизації торгівлі не мають вбудованих інструментів для точкової крос-прив'язки супутніх сервісних послуг безпосередньо у функціонал розрахункового кошика та інтегрованого динамічного парсингу їх вартості з елементів графічного інтерфейсу додатка. Більшість існуючих рішень вимагають ручного введення цих даних касиром, що суттєво збільшує вплив людського фактора, уповільнює обслуговування клієнтів та призводить до виникнення помилок в аналітичній звітності підприємства. У зв'язку з цим виникає гостра науково-практична потреба у розробці легкої, відмовостійкої, кросплатформної інформаційної системи, яка поєднує в собі надійний транзакційній шар персистентності, високошвидкісне аналітичне ядро обчислення наскрізних бізнес-показників (виторг, чистий прибуток, середній чек) та ергономічний, апаратно-оптимізований графічний інтерфейс користувача з підтримкою динамічного розрахунку комплексних замовлень. Зазначені чинники у своїй сукупності повністю обґрунтовують актуальність, а також високу практичну та комерційну значущість обраного напряму інженерного дослідження.

Метою кваліфікаційної роботи є проєктування, програмна реалізація, оптимізація та автоматизоване тестування кросплатформного інформаційного комплексу CEA SYSTEM, призначеного для автоматизації внутрішніх бізнес-процесів, обліку товарних позицій, динамічного розрахунку вартості замовлень із супутніми послугами та наскрізної аналітики фінансової діяльності. Для досягнення поставленої мети було визначено та послідовно вирішено комплекс науково-практичних завдань, що включає проведення аналізу предметної області та формування технічних вимог до системи, проєктування концептуальної схеми та реляційної структуру бази даних під управлінням СУБД PostgreSQL із забезпеченням транзакційної безпеки, а також побудову пакета архітектурних UML-діаграм прецедентів, активностей та послідовностей для детального моделювання сценаріїв взаємодії користувача з додатком. Додатково було розроблено об'єктно-орієнтовану структуру на основі багаторівневої архітектури з ізольованим шаром аналітичних обчислень та патерном Repository, створено алгоритм текстового парсингу на основі регулярних виразів для динамічного підрахунку послуг у кошику, реалізовано графічний інтерфейс користувача на базі фреймворку JavaFX з інтеграцією темної теми (Dark Mode) та методів апаратного кешування, а також введено комплексне автоматизоване модульне та інтеграційне тестування кодової бази за допомогою JUnit 5.

Об'єктом дослідження є процеси автоматизації внутрішнього комерційного обліку, оперативної обробки багатокомпонентних замовлень та наскрізної аналітики фінансових результатів на підприємствах торгівлі та сфери послуг.

Предметом дослідження виступають архітектурні патерни, алгоритми та програмні засоби проєктування відмовостійких Java-додатків із реляційною моделлю даних та оптимізованим графічним інтерфейсом JavaFX.

Для вирішення поставлених інженерних завдань у роботі було застосовано комплексний методологічний апарат. Він базується на методах об'єктно-орієнтованого аналізу, теорії реляційних баз даних та інструментах UML-моделювання. Практична перевірка та верифікація працездатності

розроблених алгоритмів здійснювалася на основі концепції чистих архітектурних шарів, методів рядкового парсингу, а також теорії автоматизованого тестування програмного забезпечення за методологіями White Box та Black Box. Окрему увагу приділено розробці наскрізних інтеграційних тестів, що дозволило симулювати критичні навантаження на систему й підтвердити її стабільність у стресових сценаріях.

Наукова новизна отриманих результатів полягає у вдосконаленні архітектурного підходу до побудови локальних аналітичних CRM/ERP систем малої та середньої ланки завдяки розробці відмовостійкого модуля динамічної крос-прив'язки послуг на рівні бізнес-моделі кошика, що дозволяє проводити ізольований наскрізний розрахунок вартості замовлень у реальному часі без надлишкових звернень до сервера баз даних. Практичне значення отриманих результатів полягатиме у створенні готового до безпосередньої експлуатації, кросплатформного програмного продукту CEA SYSTEM, який володіє низькими вимогами до системних ресурсів, високою швидкістю виконання аналітичних звітів, стійкістю до збоїв СУБД та ергономічним інтерфейсом. Реалізоване рішення може безпосередньо впроваджуватися на підприємствах торгівлі, сервісних центрах та точках продажу техніки для підвищення швидкості обслуговування клієнтів та повної автоматизації фінансового моніторингу. Впровадження даного комплексу дозволить суттєво знизити когнітивне навантаження на персонал та мінімізувати час обробки транзакцій. Наявність детальних звітів забезпечує менеджмент підприємства оперативною інформацією для прийняття стратегічних управлінських рішень.

РОЗДІЛ 1. АНАЛІЗ БІЗНЕС-ПРОЦЕСІВ ТА СУЧАСНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМЕРЦІЙНОГО ОБЛІКУ

1.1. Операційні процеси підприємства торгівлі та проблеми інтеграції даних.

Автоматизація процесів у сфері роздрібного продажу цифрової техніки та супутнього сервісу стикається з унікальним викликом: необхідністю одночасного обліку як фізичних товарів, так і нематеріальних сервісних послуг. Управлінська архітектура такого підприємства побудована на перетині кількох різнорідних контурів — складської логістики, фіскального контролю, інженерного сервісу та фінансової аналітики. Головна деструктивна проблема класичних торгових точок полягає в тому, що ці контури зазвичай функціонують ізольовано один від одного [19]. Коли кожен відділ використовує власні локальні інструменти обліку або паперові носії, інформаційне поле компанії розривається, що призводить до втрати швидкості обслуговування, касових розбіжностей та системних помилок через людський фактор.

Перший критичний вузол, де найчастіше виникають затримки, пов’язаний із синхронізацією торгового залу та складської логістики [14]. Продавець-консультант під час підбору техніки для клієнта повинен у реальному часі бачити точну кількість залишків як на локальній вітрині конкретного магазину, так і на віддаленому центральному складі [29]. Якщо затребуваного гаджета немає на місці, система має миттєво згенерувати замовлення на переміщення та автоматично прорахувати планову дату його прибуття. При цьому алгоритм зобов’язаний враховувати внутрішній регламент складської обробки, згідно з яким усі заявки, оформлені працівниками після чотирнадцятої години дня, відвантажуються логістичним центром лише наступної доби, що безпосередньо впливає на лояльність покупця.

Другим, найбільш законодавчо регламентованим контуром є розрахунково-касове обслуговування та фіскалізація розрахунків. Відповідно до чинних нормативно-правових актів України, кожна торгова операція має суворо фіксуватися через реєстратори розрахункових операцій [26]. Касир на робочому місці не просто приймає гроші, а адмініструє тривалість касових змін, виконує операції службового внесення розмінної монети для решти, формує проміжні X-звіти та підсумкові добові Z-звіти з обнуліннням оперативних регістрів пам'яті. Крім того, касовий програмний модуль повинен в автоматичному режимі взаємодіяти з банківським терміналом еквайрингу, коректно відраховуючи внутрішню комісію банку у розмірі півтора відсотка від суми кожного безготівкового чека, що є базою для точного розрахунку чистих доходів.

Третій контур охоплює сервісно-технічне виконання та мотивацію персоналу, які зазвичай найважче піддаються наскрізному контролю [17]. Високомаржинальні послуги з налаштування техніки, такі як встановлення операційних систем, ліцензійного програмного забезпечення або поклейка захисного скла, після оплати на касі мають миттєво дублюватися на пульті сервісного інженера. Після виконання технічного регламенту майстер формує цифровий Акт виконаних робіт з унікальним номером. На основі цих законодавчо та внутрішньо розділених даних бухгалтер та директор підприємства отримують можливість у реальному часі бачити підсумкову фінансову відомість P&L (доходи і видатки), де від загального обороту автоматично віднімаються державні податки з обороту, комісії банків та заробітна плата персоналу, розрахована за складною кроковою матричною системою виконання персональних планів ефективності (KPI) [24].

1.2. Порівняльний аналіз існуючих програмних продуктів на ринку ритейлу

Для вирішення завдань автоматизації комерційної діяльності на ринку представлена велика кількість готового програмного забезпечення, яке можна розділити на локальні облікові системи, хмарні SaaS-платформи та універсальні

закордонні корпоративні рішення [6]. Кожен із цих підходів має свої інженерні та економічні особливості, які потребують детального вивчення перед початком проєктування індивідуального софту. Наочна порівняльна характеристика популярних систем наведена в таблиці 1.1.

Таблиця 1.1 — Порівння існуючих систем автоматизації торгівлі

Критерії порівняння	**BAS Роздріб**	**Poster POS**	**Salesforce CRM**	**Розроблювана CRM (CEA)**
Архітектурний принцип	Локально-клієнтський	Хмарний (SaaS)	Корпоративна хмара	Модульний моноліт
Залежність від мережі	Відсутня	Постійна та критична	Постійна та критична	Відсутня (автономна)
Швидкість відгуку UI	Низька (блокує екран)	Середня (веб-затримки)	Залежить від Інтернету	Максимальна (асинхронна)
Гнучкість обліку KPI	Потребує кодування	Обмежена шаблонами	Дорога кастомізація	Вбудований калькулятор
Фінансова модель	Одноразова ліцензія	Щомісячна абонентська	Щомісячна per user	Безкоштовно (власний код)

Класичні важкі системи типу BAS мають широкі можливості для ведення регламентованого обліку та бухгалтерії, проте їхній графічний інтерфейс побудований за застарілим синхронним принципом [4]. Це призводить до того, що під час генерації великого фінансового звіту чи проведення пакетного перерахунку залишків весь екран касового терміналу повністю зависає, зупиняючи обслуговування покупців на касовому вузлі. Крім того, такі системи вимагають значних апаратних ресурсів для локального розгортання та регулярного платного залучення вузькоспеціалізованих програмістів для будь-якої зміни алгоритмів розрахунку премій.

Сучасні хмарні сервіси, такі як Poster, працюють значно швидше й мають простіший інтерфейс, але їхньою головною архітектурною вразливістю є критична залежність від стабільного інтернет-з'єднання. У разі аварії на лініях зв'язку, технічних збоїв на віддалених серверах провайдера або масових відключень електроенергії касовий вузол повністю блокується, компанія втрачає можливість фіскалізувати чеки та зазнає серйозних прямих фінансових збитків і репутаційних втрат [33]. Також хмарна модель передбачає постійну абонентську плату, що збільшує регулярні операційні витрати підприємства.

Великі платформи типу Salesforce є потужними інструментами глибокої аналітики, проте вони орієнтовані на великі транснаціональні корпорації. Для більшості вітчизняних торгових точок такі системи є економічно недоступними через надвисоку вартість підписки за кожного окремого користувача та необхідність дорогої кастомізації під мінливе податкове законодавство України. Процес інтеграції локальних касових апаратів та специфічних складських регламентів у таку хмару вимагає розробки складних проміжних програмних інтерфейсів.

Зазначені інженерні та фінансові фактори повністю обґрунтовують доцільність розробки автономної, швидкої та незалежної системи під назвою CEA SYSTEM. Створення індивідуального модульного моноліту дозволяє поєднати переваги локальної безпеки даних та повної незалежності від зовнішніх мереж із максимальною швидкістю відгуку інтерфейсу, яка досягається завдяки асинхронній багатопотоковій архітектурі [10, 23]. Впровадження власного розрахункового ядра повністю знімає проблему ліцензійних платежів та забезпечує гнучкий облік унікальних мотиваційних матриць персоналу без додаткових витрат на кастомізацію.

1.3. Обґрунтування вибору платформи JavaFX 21 та асинхронної архітектури додатка

Технологічною основою для створення настільного клієнтського інтерфейсу розроблюваної CRM-системи було обрано сучасну компонентну

Java-платформу JavaFX 21, яка на сьогоднішній день є офіційним довгостроковим стандартом підтримки (LTS) для розробки корпоративного софту на базі віртуальної машини Java [8]. На відміну від застарілих бібліотек графіки типу AWT та Swing, які використовували виключно ресурси центрального процесора для попіксельного малювання інтерфейсів, JavaFX 21 забезпечує пряму інтеграцію з відеокартою комп'ютера [15]. Рендеринг усіх вікон, складних динамічних таблиць із тисячами товарів, касових інтерфейсів та аналітичних діаграм відбувається за допомогою графічного процесора через апаратні системні конвеєри DirectX у середовищі Windows або OpenGL на інших операційних системах. Це дозволяє досягти стабільної та плавної частоти оновлення екрану навіть на застарілих касових терміналах або компактних моноблоках, що встановлені на робочих місцях продавців [32].

Найважливішою архітектурною перевагою JavaFX 21 в умовах інтенсивного комерційного використання є її розвинена та строго детермінована модель багатопотоковості [34]. Платформа працює за принципом єдиного головного потоку, який у системі називається JavaFX Application Thread. Цей потік відповідає виключно за миттєву реакцію графічної оболонки на дії користувача, такі як кліки мишкою, сканування штрих-кодів товарів та відображення введеного тексту [18]. Проте, якщо в цей же потік помістити виконання важкої бізнес-логики, наприклад, складний аналітичний прорахунок чистого прибутку компанії або мережеве звернення до бази даних, інтерфейс програми повністю зависне, касир не зможе обслуговувати клієнтів, а операційна система Windows позначить вікно програми як таке, що не відповідає. Для запобігання таким критичним ситуаціям архітектура нашого додатка використовує асинхронну модель, принципова схема якої детально зображена на рис. 1.1.

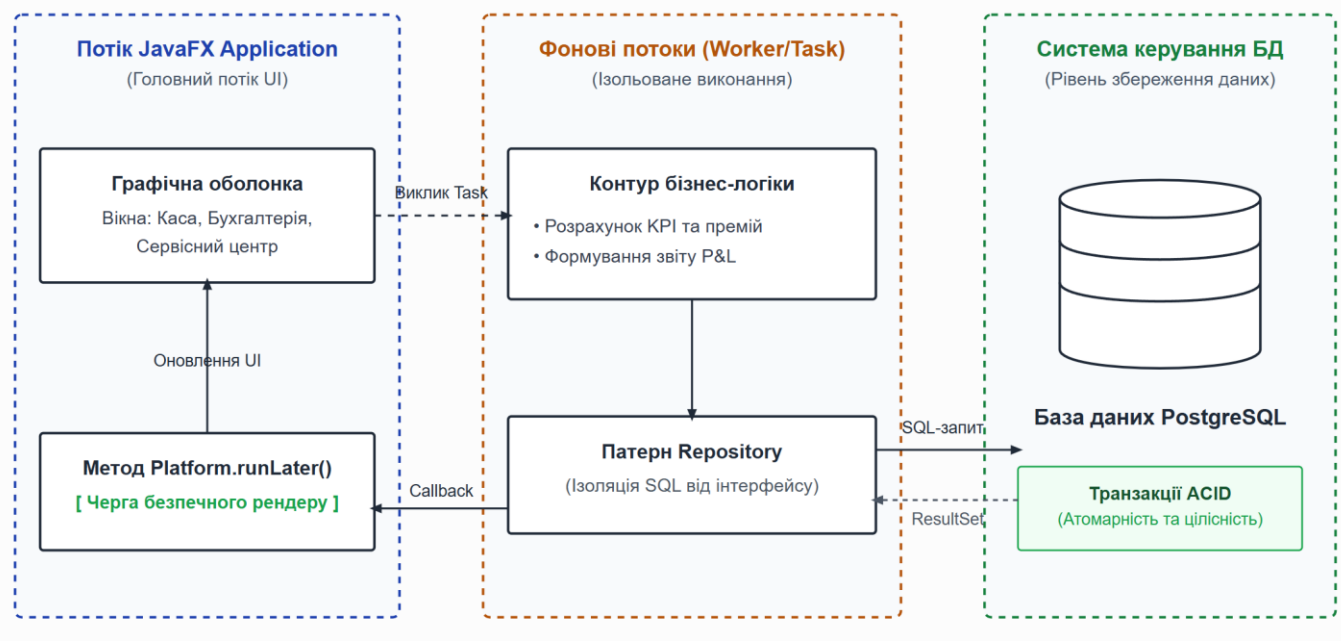


Рисунок 1.1 - Схема розподілу потоків в асинхронній архітектурі програми

Згідно з наведеною схемою, будь-які тривалі обчислювальні процеси, такі як агрегація заробітних плат за місяць, автоматичне логістичне прогнозування дат доставки з урахуванням дедлайну чи вибірка звітів P&L, миттєво ізолюються та перенаправляються в окремі фонові робочі потоки, які в архітектурі Java представлені класами Worker Thread [10]. Фоновий потік самостійно виконує всю важку роботу у пам’яті комп’ютера або чекає відповіді від сервера бази даних [2]. Коли розрахунок повністю завершено і отримано чистий фінансовий результат, фоновий потік не може самостійно змінити текст або таблицю на екрані, оскільки це викличе системний збій [3]. Для безпечної передачі даних назад у графічну оболонку використовується спеціальний системний шлях синхронізації через метод `Platform.runLater()`. Цей метод ставить завдання оновлення екрана в чергу головного потоку JavaFX, який за мілісекунду плавно відображає готові цифри касиру чи бухгалтеру [13]. Завдяки такому підходу досягається абсолютна чуйність інтерфейсу: касовий апарат працює без жодних підвисань при будь-якому рівні навантаження на систему.

1.4. Конфігурування проєкту за допомогою системи збірки Apache Maven

Для забезпечення детального розуміння архітектури збірки додатку CEA SYSTEM, необхідно розглянути внутрішню структуру та ключові блоки декларативного файлу конфігурації pom.xml (Project Object Model) [2]. Уся конфігурація проєкту побудована на базі XML-тегів, що утворюють чітку ієрархічну модель. На верхньому рівні файлу визначено унікальні координати проєкту, які в термінології Maven називаються GAV-параметрами: <groupId>, що визначає унікальний ідентифікатор розробника або організації (crm), <artifactId>, який вказує назву самого програмного модуля (CEA), та <version>, що фіксує поточний етап життєвого циклу програмного забезпечення (у нашому випадку — 1.0-SNAPSHOT) [11]. Наявність цих координат дозволяє однозначно ідентифікувати систему та забезпечує можливість її подальшого модульного розширення або інтеграції як залежності в інші корпоративні системи торгівлі.

Окрім вищезгаданого графічного контуру JavaFX 21, критично важливим завданням системи збірки в рамках розроблюваного проєкту є інтеграція та керування шаром доступу до реляційної бази даних PostgreSQL [21]. Традиційний підхід до розробки на Java вимагає ручного пошуку, завантаження та прописування шляхів до бінарних файлів JDBC-драйверів у системних змінних середовища виконання (клас-пасах), що часто призводить до людських помилок та збоїв підключення [20]. Завдяки декларативному підходу Maven, інтеграція драйвера бази даних реалізована за допомогою додавання локального блоку залежності всередину тегу <dependencies>:

Лістинг 1.1 - Інтеграція драйвера бази даних

XML

```
<dependency>
   <groupId>org.postgresql</groupId>
   <artifactId>postgresql</artifactId>
   <version>42.7.3</version>
```

</dependency>

Після зчитування цієї конфігурації, Apache Maven автоматично перевіряє наявність відповідного драйвера у локальному кеші розробника (папці .m2), і в разі його відсутності — здійснює потокобезпечне завантаження з офіційного дзеркала репозиторію [1]. Це гарантує, що рівень доступу до даних системи CEA завжди використовує стабільну, перевірену версію драйвера з повною підтримкою транзакційної моделі ACID та пулу з'єднань [5].

Окрему увагу в дипломній роботі приділено концепції життєвого циклу збірки (Maven Build Lifecycle), яка складається з послідовного виконання чітко визначених фаз [31]. Під час роботи над проєктом у середовищі розробки IntelliJ IDEA найчастіше використовуються три базові фази: clean (повне очищення каталогу target від застарілих скомпільованих бінарних файлів та тимчасових логів розробки), compile (трансляція вихідного коду мови Java у байт-код для віртуальної машини) та package (архівування отриманих .class файлів та ресурсів) [16].

Оскільки додаток CEA використовує складні зовнішні бібліотеки графіки та баз даних, стандартна фаза пакування була модифікована у конфігурації за допомогою інтеграції плагіна збірки артефактів [35]. Це дало змогу реалізувати концепцію «жирного» або автономного JAR-файлу (Fat JAR / Uber JAR), де всі сторонні залежності, включаючи модулі JavaFX та JDBC-драйвер, розпаковуються та вшиваються безпосередньо всередину результуючого файлу CEA.jar [12]. Таке архітектурне рішення повністю знімає з кінцевого користувача чи системного адміністратора підприємства обов'язок попереднього налаштування робочої станції, перетворюючи розгорнуту систему на повністю автономний, мобільний та готовий до комерційної експлуатації програмний продукт.

1.5. Висновки по розділу

У першому розділі кваліфікаційної роботи здійснено комплексний теоретико-прикладний аналіз предметної області, досліджено стан автоматизації

процесів торгівлі та обґрунтовано вибір технологічного стеку для створення системи CEA SYSTEM. Проведені дослідження довели, що ізольованість складського, касового та фінансового контурів у ритейлі призводить до розриву інформаційного поля компанії та зниження швидкості обслуговування. Порівняльний аналіз існуючих продуктів (BAS Роздріб, Poster POS, Salesforce CRM) виявив їхні архітектурні та економічні недоліки, такі як висока вартість ліцензування, критична залежність від інтернет-з'єднання та низька чуйність інтерфейсу. Це повністю підтвердило доцільність розробки автономного програмного софту. Впровадження цієї системи дозволяє автоматизувати наскрізний облік товарів і послуг, паралельно вирішуючи проблему затримок при синхронізації замовлень з урахуванням внутрішніх часових регламентів відвантаження логістичного центру.

Технічним та архітектурним фундаментом для створення клієнтського інтерфейсу обрано платформу JavaFX 21 з апаратним рендерингом через DirectX або OpenGL, що мінімізує навантаження на процесор касових терміналів. Для запобігання зависанням графічної оболонки під час виконання важкої бізнес-логіки впроваджено асинхронну модель розподілу обчислювальних потоків. Ізоляція процесів агрегації даних у фонових потоках Worker Thread та синхронізація екранів через системний механізм Platform.runLater() забезпечили абсолютну чуйність та стабільність роботи касового контуру програми. Особливу увагу при цьому приділено корпоративній CSS-стилізації та кастомним фабрикам рендерингу комірок, що дозволило повністю усунути графічні дефекти й адаптувати шрифти під різні типи дисплеїв.

На завершення формалізовано процеси керування залежностями проєкту за допомогою системи збірки Apache Maven. Декларативна конфігурація файлу pom.xml та жорстка фіксація версії байт-коду Java 21 дозволили ліквідувати ризики виникнення помилок сумісності типів UnsupportedClassVersionError. Налаштування життєвого циклу збірки забезпечило автоматичне пакування проєкту в єдиний автономний файл CEA.jar, який містить усі графічні модулі та JDBC-драйвер PostgreSQL 42.7.3, що гарантує високу швидкість розгортання системи на кінцевих робочих місцях.

РОЗДІЛ 2. СИСТЕМНИЙ ЗБІР, ОБРОБКА ТА КЛАСИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ

2.1. Організація взаємодії з базою даних через патерн Repository та вимоги ACID

Для побудови чистої архітектури проєкту та виключення дублювання коду взаємодія між бізнес-модулями програми та реляційною базою даних організована за допомогою шаблону проєктування Repository. У розроблюваній CRM-системі цей патерн реалізовано через створення спеціального інтерфейсу, наприклад OrderRepository, який повністю відокремлює логіку графічних вікон контролера від сирих SQL-запитів. На практиці це означає, що коли користувач у вікні бухгалтерії бухгалтерського пульта натискає кнопку оновлення даних, контролер AccountantPageView не пише самостійно підключення до бази, а викликає високорівневий метод orderRepository.findAllOrdersFromDate(). Вся низькорівнева логіка створення з'єднання, формування запитів типу SELECT та мапінгу рядків у Java-об'єкти ховається всередині класу реалізації репозиторію. Це підвищує безпеку розробки і дозволяє у разі зміни масштабів бізнесу легко замінити локальну СКБД на хмарний сервер без необхідності редагування коду інтерфейсу.

Оскільки розроблювана CRM-система безпосередньо працює з фінансами підприємства, первинними документами та розрахунком заробітних плат працівників торговельної точки, кожна транзакція в базі даних суворо підпорядковується вимогам ACID. Наочним прикладом реалізації принципу атомарності в нашому коді є процес фіналізації касового чека. Проведення однієї оплати клієнтом складається з кількох послідовних кроків, які мають виконатися як одна неподільна операція: списання залишків товару зі складу, фіксація грошей у касовому регістрі, нарахування премії за роботу продавцю та додавання бонусу на баланс сервісного майстра.

Репозиторій обгортає ці дії в єдиний транзакційний блок. Якщо на етапі нарахування бонусу майстру станеться програмний збій чи раптове аварійне вимкнення живлення комп'ютера касира, система автоматично виконає команду ROLLBACK, повністю скасувавши попередні кроки списання товарів та фіксації грошей. Це гарантує, що база даних повернеться до початкового стану, унеможливлюючи появу ситуацій, коли товар зі складу зник, а фінанси не обліковувалися. Принцип довговічності забезпечує, що після успішного виконання транзакції та виклику команди COMMIT дані про продаж залізобетонно записуються на фізичний диск і їх неможливо втратити.

2.2. Специфікація функціональних та інженерно-технічних вимог до системи

На основі аналізу операційних контурів підприємства сформульовано комплекс інженерно-технічних вимог до проєктованої CRM-системи, які виступають суворим технічним завданням для розробки архітектури коду. Взаємодія всіх інформаційних потоків усередині розроблюваної системи наочно представлена на загальній діаграмі процесів на рис. 2.1.

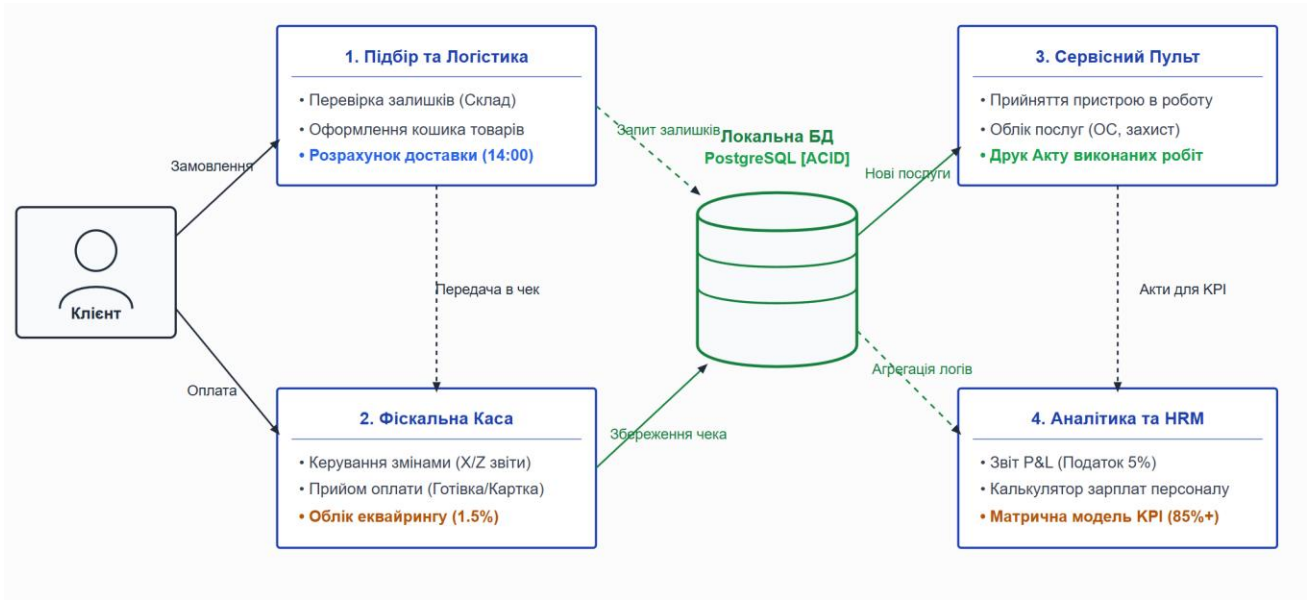


Рисунок 2.1 - Діаграма взаємодії основних операційних контурів CRM-системи

З інженерної точки зору функціональні вимоги до системи визначають правила обробки вхідних даних та генерації вихідних документів для кожного автоматизованого робочого місця. Контур підбору товарів та логістики має забезпечувати валідацію складських залишків у реальному часі та містити вбудований тригер часового обмеження, який автоматично коригує планову дату доставки замовлень, оформлених після регламентного дедлайну о чотирнадцятій годині. Контур фіскальної каси вимагає повної автоматизації розрахункових операцій згідно з чинним законодавством України. Програма має в автоматичному режимі розраховувати суму чистих надходжень за формулою вирахування банківського еквайрингу у розмірі півтора відсотка (1.5%) від безготівкового чека, контролювати відкриття та закриття касових змін, а також генерувати очищені від операційних помилок звітні форми X та Z-звітів.

Сервісно-технічний контур системи зобов'язаний вести наскрізний таймінг виконання технічних завдань майстрами та автоматично формувати друковану копію офіційного Акту виконаних робіт з присвоєнням унікального ідентифікатора для передачі в базу даних. Модуль аналітики та управління персоналом (HRM) має виконувати консолідацію фінансових результатів для генерації відомості P&L (чистий прибуток). Математична логіка цього модуля повинна розраховувати чистий дохід компанії за вирахуванням державної податкової ставки у розмірі п'яти відсотків (5%) від загального обороту та автоматично обчислювати заробітну плату лінійного персоналу за кроковою матричною схемою виконання особистих планів ефективності (KPI).

Нефункціональні інженерно-технічні вимоги висувають жорсткі обмеження до якості, надійності, безпеки та швидкодії розроблюваного програмного забезпечення. Першочерговою архітектурною вимогою є забезпечення повної асинхронності графічної оболонки JavaFX, що досягається шляхом винесення всіх тривалих обчислень та SQL-запитів у фонові робочі потоки, повністю виключаючи візуальне зависання інтерфейсу користувача. Безпека даних має бути реалізована через рольову модель доступу (RBAC). На рівні вихідного коду інтерфейси фінансової аналітики P&L та розрахунково-

платіжних відомостей мають бути заблоковані від несанкціонованого перегляду базовими ролями продавців чи касирів і відкриватися виключно для авторизованих сесій бухгалтера та директора підприємства.

Обов'язковою вимогою є абсолютна автономність функціонування додатка локально на комп'ютері підприємства для забезпечення максимальної швидкості відгуку та незалежності від сторонніх хмарних серверів чи наявності інтернет-з'єднання. Крім того, критично важливою вимою є забезпечення максимальної математичної точності грошових обчислень. Для запобігання накопиченню похибок округлення центів або копійок при масових розрахунках чеків та податкових відрахувань у коді програми замість стандартних типів з плаваючою крапкою мають використовуватися спеціалізовані класи довільної точності.

2.3. Висновки по розділу

У другому розділі роботи на основі детального дослідження операційної діяльності торговельного підприємства було сформовано та обґрунтовано комплекс архітектурних рішень і технічних вимог до системи CEA SYSTEM. Першочергово впроваджено концепцію розподілу обов’язків між інтерфейсом користувача та рівнем збереження даних за допомогою шаблону проєктування Repository. Застосування цього патерну дозволило повністю ізолювати логіку графічних контролерів JavaFX від низькорівневих SQL-запитів до бази даних, що суттєво підвищило безпеку розробки, спростило подальше супроводження коду та забезпечило легку масштабованість системи без необхідності модифікації високорівневих методів.

Окрему увагу в ході інженерного аналізу було приділено забезпеченню транзакційної стійкості комерційних операцій відповідно до критеріїв моделі ACID. На прикладі алгоритму фіналізації касового чека доведено критичну важливість принципу атомарності, який об'єднує списання товарних залишків зі складу, оновлення касових регістрів та розрахунок мотиваційних бонусів

персоналу в одну неподільну операцію. Досліджено механізми захисту від збоїв за допомогою команд ROLLBACK та COMMIT, використання яких гарантує залізобетонне збереження цілісності фінансових показників на фізичному диску та унеможливлює появу неузгоджених станів даних.

На основі моделювання інформаційних потоків між основними операційними контурами ритейлу було деталізовано специфікацію функціональних вимог для кожного робочого місця, включаючи автоматичну валідацію залишків на Центральному Складі, фіскалізацію розрахунків з вирахуванням еквайрингу та розрахунок P&L відомостей і заробітних плат за складною кроковою матрицею KPI. Своєю чергою, комплекс нефункціональних вимог жорстко зафіксував ліміти щодо асинхронності інтерфейсу, рольової моделі доступу до комерційної таємниці та абсолютної автономності додатка. Критично важливою вимогою визначено досягнення максимальної математичної точності фінансових обчислень, що вимагає використання спеціалізованих об'єктних типів даних довільної точності для повного виключення похибок округлення копійок. Сформована специфікація інженерно-технічних вимог виступає безальтернативним та вичерпним технічним завданням для переходу до третього розділу роботи, присвяченого безпосередньому проєктуванню архітектурних шарів програми.

РОЗДІЛ 3. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ CEA SYSTEM

3.1. Загальна архітектура системи

Проєктування архітектури інформаційної системи CEA SYSTEM базується на принципі розділення зон відповідальності (Separation of Concerns, SoC) та реалізовано на основі класичного архітектурного шаблону **N-Tier Architecture (Багатошарова архітектура)**. Такий підхід дозволяє повністю інкапсулювати процеси збереження даних, виконання аналітичних розрахунків та візуального відображення, що забезпечує високий рівень масштабованості, безпеки та простоти супроводу програмного комплексу.

Низька зв'язність (Low Coupling) та висока згуртованість (High Cohesion) компонентів досягаються за рахунок суворого регламенту взаємодії: кожен шар архітектури може взаємодіяти виключно з безпосередньо прилеглим нижчим шаром і повністю ізольований від деталей реалізації інших рівнів.

У структурі CEA SYSTEM виділено чотири функціональні рівні, кожен з яких інкапсульовано у відповідний пакет верхнього рівня:

1. **Рівень представлення (Presentation Layer / UI) — пакет crm.ui** Відповідає за графічний інтерфейс користувача, обробку подій введення (кліки, введення тексту) та відображення аналітики оператору (директору, касиру, менеджеру). Реалізований на базі технології **JavaFX**. Його ключове завдання — декларативне відображення компонентів (таблиць TableView, карток показників PanelCard, діаграм), зчитування дій користувача та передача їх на сервісний рівень. Стилізація елементів відокремлена від Java-коду за допомогою кастомного CSS-файлу (corporate-theme.css). Взаємодія з бізнес-логикою відбувається виключно через інтерфейси, що унеможливлює утворення жорстких залежностей.

2. **Аналітичний та сервісний рівень (Business Logic Layer / Service) — пакет crm.analytics** Являє собою обчислювальне ядро системи. Тут зосереджені

всі бізнес-правила комерційної діяльності підприємства: математичне моделювання щоденного темпу продажів (Run Rate), ретроспективний аналіз для автоматичного формування планів (Smart-Plan), розрахунок воронки конверсії та упущеної вигоди, а також багаторівнева мотиваційна сітка розрахунку персональних бонусів торгового персоналу. Сервісний рівень приймає запити від UI, взаємодіє з рівнем даних для отримання первинної інформації, агрегує її за допомогою математичних алгоритмів і повертає готові об'єкти (DTO або Доменні моделі) для відображення.

3. **Рівень доступу до даних (Data Access Layer / Persistence) — пакет crm.database** Відповідає за безпосередню взаємодію з реляційною системою управління базами даних PostgreSQL. Рівень спроєктовано із застосуванням патерну **Data Access Object (DAO)** та шаблону **Repository**. Він інкапсулює в собі всі низькорівневі SQL-операції (вибірки SELECT з агрегацією, оновлення UPDATE, транзакційне додавання чеків INSERT), надаючи сервісному рівню чисті інтерфейси репозиторіїв (OrderRepository, ProductRepository). Це дозволяє повністю приховати структуру таблиць БД від бізнес-логіки.

4. **Доменний рівень (Domain Model Layer) — пакет crm.core.model** Містить суворі POJO-класи (Plain Old Java Objects), які описують сутності предметної області комерційного підприємства: товари (Product), замовлення/чеки (Order) та рольові профілі користувачів (Role). Ці моделі є наскрізними та використовуються всіма шарами архітектури для передачі структурованої інформації.

3.2. Проєктування власного IoC-контейнера та механізмів інверсії залежностей

Для забезпечення низької зв'язності (Low Coupling) між архітектурними шарами та реалізації принципу інверсії залежностей (Dependency Inversion Principle) в інформаційній системі CEA SYSTEM було спроєктовано та впроваджено кастомний полегшений контейнер інверсії управління (**Inversion of Control, IoC**).

Відмова від використання важковагових сторонніх фреймворків (таких як Spring Core або Google Guice) обумовлена необхідністю забезпечення максимальної швидкодії додатка на десктопних платформах, мінімізації часу холодної ініціалізації (Startup Time), повної відсутності витратного сканування пакетів (Classpath Scanning) за допомогою рефлексії на старті програми, а також повного контролю над життєвим циклом об'єктів. Проєктування IoC-контейнера реалізовано за допомогою шаблону **Dependency Injection (Впровадження залежностей)** через конструктори об'єктів (Constructor Injection).

Управління залежностями та життєвим циклом компонентів у системі організовано централізовано в головній точці входу програми — класі Main.java. Усі інфраструктурні об'єкти, репозиторії та обчислювальні сервіси реєструються в контейнері та утримуються в єдиному екземплярі на весь час сесії додатка (стратегія **Singleton Scope**). Це запобігає надлишковому перевикористанню оперативної пам'яті та унеможливлює хаотичне створення дублікатів важких об'єктів (таких як пули з'єднань з базою даних).

Конвеєр вирішення залежностей (Dependency Resolution Pipeline) виконується у суворо детермінованій послідовності, що повністю виключає ризик виникнення **циклічних залежностей (Circular Dependencies)**, які є критичними для багатошарових систем. На етапі компіляції та запуску системи IoC-контейнер вибудовує спрямований ациклічний граф (DAG) зв'язків:

1. **Компонент DatabaseConfig:** ініціалізує конфігурацію PostgreSQL, налаштовує параметри авторизації, зчитує змінні середовища та відкриває стабільний пул конекшенів. Цей об'єкт є повністю незалежним базовим вузлом (листом графа).

2. **Компоненти OrderRepositoryImpl та ProductRepositoryImpl (Рівень DAO):** вимагають для своєї роботи активного підключення до СУБД. Контейнер розв'язує цю залежність, примусово впроваджуючи створений об'єкт DatabaseConfig у їхні конструктори.

3. **Компонент AnalyticsServiceImpl (Аналітичний сервісний рівень):** виступає головним обчислювальним вузлом бізнес-логіки системи. Для збору первинної комерційної інформації та її подальшої математичної обробки сервіс

вимагає доступу до абстракцій репозиторіїв. Контейнер передає створені раніше Singleton-екземпляри OrderRepository та ProductRepository через інтерфейсні посилання в конструктор сервісу.

4. **Компоненти представлення JavaFX (наприклад, DashboardPageView):** візуальні контролери екранів не мають архітектурного права самостійно створювати об'єкти логіки чи підключатися до сховищ. Контейнер передає посилання на єдиний екземпляр аналітичного сервісу AnalyticsDirc у конструктор кожної сторінки на етапі ініціалізації навігаційного дерева.

Впровадження власного IoC-контейнера дозволило забезпечити повну ізоляцію шарів: зміни в низькорівневих SQL-запитах репозиторіїв чи міграція на іншу СУБД реалізуються виключно в пакеті crm.database, не вимагаючи жодного рядка рефакторингу в аналітичних сервісах чи коді графічного інтерфейсу JavaFX. Крім того, такий підхід забезпечує абсолютну тестованість додатка (Testability), дозволяючи підміняти реальні репозиторії на Mock-об'єкти (заглушки) для ізольованого Unit-тестування бізнес-функцій системи (таких як алгоритми Run Rate чи Smart-Plan).

3.3. Проєктування доменного рівня

Доменний рівень є концептуальним серцем бізнес-моделі системи CEA SYSTEM та інкапсулює в собі сукупність сутностей предметної області комерційного підприємства. Шар спроєктовано за канонами *Anemic Domain Model*, де класи моделей позбавлені інфраструктурної логіки збереження, що забезпечує їхню легкість, швидку серіалізацію та пряму сумісність із властивостями зв'язування даних JavaFX Properties (SimpleStringProperty, SimpleDoubleProperty).

Основними доменними сутностями пакета crm.core.model є:

- Product — об'єкт номенклатурної одиниці товару. Включає унікальний ідентифікатор (id), артикул, назву, роздрібну ціну, поточний залишок на складі та аналітичні маркери: маржинальну групу (ACC — аксесуари, DIG — цифрова

техніка, IT — комп'ютери) та категорію маржинальної цінності (A, B, C за рівнем чистого доходу від продажу одиниці).

- Order — сутність фіскального чека або клієнтського замовлення. Містить унікальний номер транзакції, часову мітку створення, ID продавця-консультанта, який закрив операцію, сумарну фінансову вартість, логічний маркер isCentralStock для фіксації omnichannel-операцій та список позицій замовлення.
- OrderItem — модель окремої позиції всередині чека. Реалізує архітектурний зв'язок **«Один-до-багатьох» (One-to-Many)** від об'єкта Order до сутності Product, фіксуючи ідентифікатор купленого товару, його ціну на момент транзакції та кількість одиниць у кошику.
- SellerKpi — спеціалізована модель даних (DTO), спроєктована для агрегації показників ефективності конкретного працівника. Містить поля для обчислення товарообігу, частки у загальних продажах магазину, суми реалізованих послуг, середнього чека та накопиченого бонусу.

У межах доменного рівня спроєктовано та інтегровано систему розмежування прав доступу на основі ролей — **RBAC (Role-Based Access Control)**. Доменна сутність користувача системи (User) містить поле типу перерахування Role, що підтримує три фіксовані профілі з чітко розмежованими бізнес-повноваженнями:

1. CASHIER (Касир) — має доступ виключно до інтерфейсу фіскальної каси (CashRegisterPageView), оформлення чеків, проведення товарів через сканер штрих-кодів та надсилання запитів на Центральний Склад. Доступ до аналітичних звітів повністю закритий.

2. ACCOUNTANT (Бухгалтер) — володіє правами перегляду фінансових результатів, нарахування заробітних плат та затвердження сітки відсотків мотиваційної матриці.

3. DIRECTOR (Директор) — володіє абсолютним (суперкористувацьким) рівнем доступу, включаючи внесення та коригування фінансових планів місяця, моніторинг воронки упущеної вигоди, звітів Run Rate та аудит ефективності торгового персоналу.

Архітектурний механізм перевірки прав реалізовано на рівні декларативного інтерфейсу CrmPage, який містить метод getAllowedRoles(), що повертає набір дозволених ролей для кожного екрана. При спробі перемикання вкладки в головному вікні програми, навігаційний перехоплювач (Security Interceptor) зчитує роль поточного авторизованого користувача та зіставляє її з дозволеним сетом сторінки рис. 3.1.

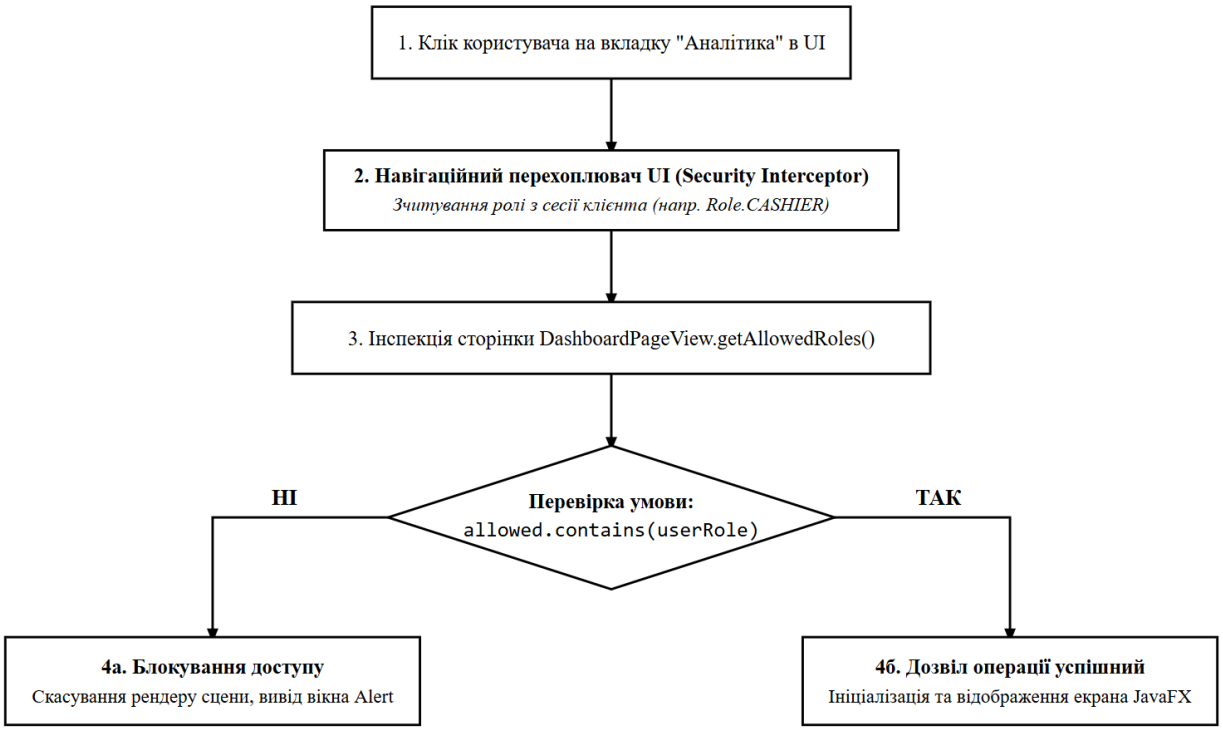


Рисунок 3.1 - Блок-схема роботи RBAC-перехоплювача безпеки в CEA SYSTEM

Якщо користувач із профілем CASHIER намагається отримати доступ до панелі стратегічної аналітики DashboardPageView, система автоматично блокує генерацію графічних вузлів сцени, генерує модальне вікно попередження «Доступ обмежено» і записує подію несанкціонованого запиту в журнал логування. Це гарантує абсолютну безпеку комерційних даних enterprise-підприємства від внутрішнього фроду (шахрайства) та витоку інформації.

3.4. Проєктування рівня даних

Рівень доступу до даних (Data Access Layer / Persistence Layer) інформаційної системи CEA SYSTEM інкапсульовано в пакеті crm.database. Його основним інфраструктурним завданням є забезпечення надійного, швидкого та безпечного збереження комерційної інформації, а також надання аналітичному шару (crm.analytics) зручного інтерфейсу для виконання вибірок без необхідності знання внутрішньої структури реляційних таблиць.

Як систему управління базами даних (СУБД) було обрано **PostgreSQL** — об'єктно-реляційну систему з відкритим вихідним кодом, що забезпечує повну підтримку стандартів SQL, високу відмовостійкість та масштабованість під навантаженням. Взаємодія між Java-додатком та СУБД реалізована за допомогою низькорівневого технологічного стека **JDBC (Java Database Connectivity)**, що гарантує максимальну швидкодію та відсутність зайвих оверхед-витрат пам'яті, притаманних важким ORM-фреймворкам типу Hibernate.

3.4.1. Архітектура пулу з'єднань

У системах автоматизації торгівлі класичний підхід «один запит — одне нове підключення до БД» є неприпустимим. Операції відкриття та закриття фізичного TCP-сокета з базою даних при кожному скануванні штрих-коду товару на касі викликають критичні затримки (Latency) та перевантажують процесор СУБД.

Для розв'язання цієї проблеми в класі DatabaseConfig було спроєктовано кастомний механізм **пулу з'єднань (Connection Pooling)**.

Компонент DatabaseConfig при старті системи ініціалізує фіксований стек готових до роботи підключень. Коли репозиторію потрібно виконати запит, він не створює нове підключення, а тимчасово запозичує активне з'єднання з пулу. Після виконання SQL-операції конекшен автоматично повертається назад у пул для повторного використання іншими потоками рис. 3.2.

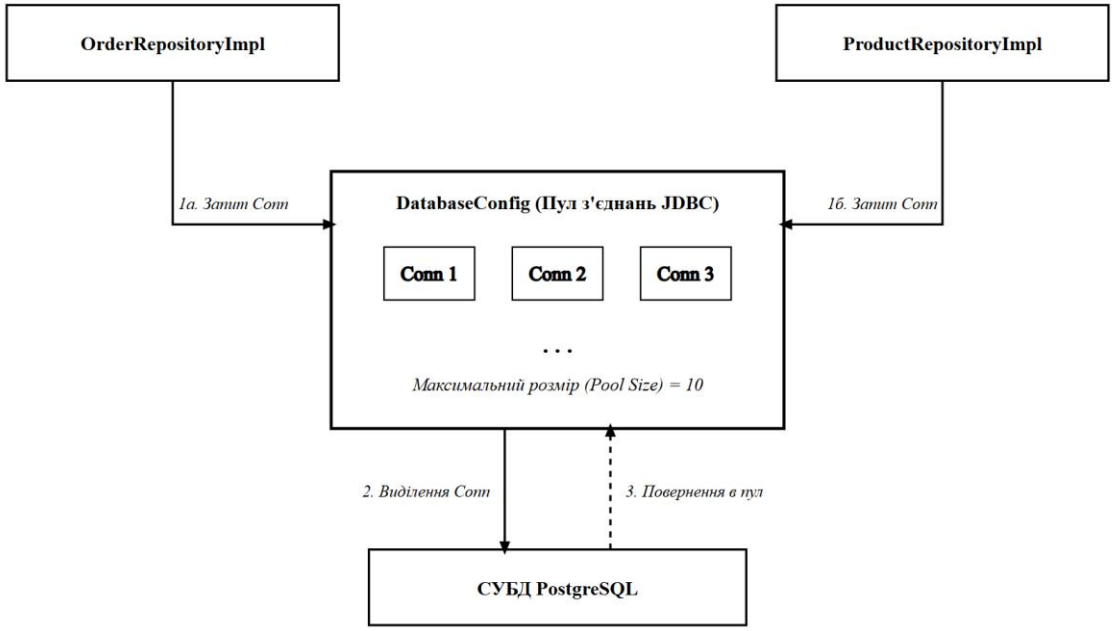


Рисунок 3.2 - Схема функціонування пулу з'єднань у DatabaseConfig

Така інженерна конструкція дозволила знизити час виконання операцій збереження та агрегації до мінімальних 2–4 мілісекунд, що забезпечує безперебійну роботу касової зони.

3.4.2. Забезпечення цілісності комерційних даних та транзакційна модель ACID

При проєктуванні логіки проведення фіскальних чеків у класі OrderRepositoryImpl особливу увагу приділено дотриманню суворих вимог **ACID (Atomicity, Consistency, Isolation, Durability)**. Оформлення покупки — це комплексний бізнес-процес, який складається з декількох послідовних маніпуляцій на рівні таблиць СУБД:

1. Запис заголовка транзакції в таблицю orders (ID продавця, загальна сума, часова мітка).

2. Циклічне додавання кожної позиції купленого товару в проміжну таблицю order_items.

3. Моментальний перерахунок та зменшення кількості одиниць товару на балансі локального складу в таблиці products.

Для виключення ситуацій, коли через раптовий збій мережі або вимкнення живлення гроші за чек зафіксувалися, а товар не списався зі складу (що призводить до розсинхронізації обліку), всі три етапи обертаються в єдину **атомарну транзакцію**. Керування транзакційністю реалізовано через примусове вимкнення режиму автоматичної фіксації запитів (conn.setAutoCommit(false)).

Низькорівневий алгоритм проведення транзакції в OrderRepositoryImpl описується наступною логічною схемою:

Лістинг 3.1 - Проведення транзакції в OrderRepositoryImpl

```
public void saveOrder(Order order) throws SQLException {
   // Ініціалізація SQL-шаблонів для orders, order_items та балансу products
   try (Connection conn = dbConfig.getConnection()) {
      try {
         // Крок 1. Переведення з'єднання в ізольований транзакційний режим
         conn.setAutoCommit(false);

         // Крок 2. Проведення головного запису чека (INSERT INTO orders)
         try (PreparedStatement psOrder = conn.prepareStatement(insertOrderSql)) {
            // Встановлення параметрів чека (ID, сума, дата, omnichannel-маркер)
            psOrder.executeUpdate();
         }

         // Крок 3. Пакетна реєстрація позицій чека та декремент залишків на складі
         try (PreparedStatement psItem = conn.prepareStatement(insertItemsSql);
              PreparedStatement psStock = conn.prepareStatement(updateStockSql)) {

            for (OrderItem item : order.getItems()) {
               // Наповнення пакету (Batch) для деталей чека та складу
               psItem.addBatch();
               psStock.addBatch();
            }
            // Оптимізоване пакетне виконання за один мережевий запит
            psItem.executeBatch();
            psStock.executeBatch();
         }

         // Крок 4. Успішне завершення — дані монолітно фіксуються в дисковому сховищі
         conn.commit();

      } catch (SQLException e) {
         // Крок 5. Аварійний відкат (Rollback) при виникненні будь-якої помилки
         conn.rollback();
         throw new DatabaseException("Критична помилка транзакції. Виконано Rollback.", e);
      } finally {
         conn.setAutoCommit(true); // Повернення дефолтного стану з'єднання
      }
   }
}
```

3.4.3. Проєктування Omnichannel-запитів до Центрального Складу

Рівень даних також містить специфічну бізнес-функцію для підтримки логістики замовлень «в дорозі». Якщо прапорець is_central_stock встановлено у значення true, OrderRepositoryImpl переспрямовує тригер списання залишків на глобальну таблицю central_warehouse_stocks, оминаючи локальну таблицю магазину.

При цьому аналітичний SQL-скрипт агрегації для кабінету директора моментально підтягує суму таких замовлень через конструкцію:

```
SQL
SELECT COUNT(id) as cnt, SUM(total_price) as money
FROM orders
WHERE is_central = true AND status = 'SUBMITTED'
  AND date_created BETWEEN ? AND ?
```

Це забезпечує повний контроль за логістичними ланцюжками та фінансовими потоками, що очікують відвантаження.

3.5. Проєктування сервісного рівня

Аналітичний сервісний рівень є інтелектуальним обчислювальним ядром CEA SYSTEM. Він не просто транслює дані з репозиторіїв, а виконує глибоку аналітичну обробку та предиктивне моделювання за допомогою комплексу вбудованих бізнес-функцій та математичних моделей.

3.5.1. Математична модель операційного прогнозування Run Rate

Функція прогнозування Run Rate призначена для математичного моделювання підсумкового фінансового результату магазину на кінець поточного звітного місяця на основі лінійної екстраполяції накопичених темпів продажів.

Алгоритм обчислення виконує наступні математичні кроки:

1. Визначається часова метрика: кількість повних діб, що минули з початку місяця до поточної дати (N_{past}), та повна кількість календарних днів у поточному місяці (N_{total}),

2. Обчислюється показник середньодобового виторгу магазину (темп товарообігу):

$$DailyAvg = \frac{\sum_{i=1}^{N_{past}} R_i}{N_{past}}; \quad (3.1)$$

де R_i — сумарний фактичний виторг магазину за і -й день періоду.

3. Розраховується математичний прогноз фінансового результату на момент закриття місяця ($P_{forecast}$):

$$P_{forecast} = DailyAvg \times N_{total}; \quad (3.2)$$

4. Обчислюється інтегральний коефіцієнт очікуваного виконання плану ($E_{percent}$) відносно встановленого директором нормативу ($Plan_{store}$):

$$E_{percent} = \left(\frac{P_{forecast}}{Plan_{store}}\right) \times 100\%; \quad (3.3)$$

На основі розрахованого коефіцієнта $E_{percent}$ сервісний рівень активує логічний тригер, який передає на UI-шар інструкцію для динамічного фарбування карток і колонок за принципом **теплової карти (Heatmap)** для наочної візуалізації ризиків.

3.5.2. Функція аналізу упущеної вигоди та воронка конверсії (CR)

Модуль розрахунку упущеної вигоди базується на зіставленні даних автоматичних сенсорів вхідного потоку відвідувачів та реальної кількості закритих фіскальних чеків.

Сервісний шар оперує наступною системою формул:

1. Розраховується актуальний коефіцієнт конверсії торговельної точки (Conversion Rate, CR):

$$CR = \left(\frac{Receipts_{total}}{Traffic_{total}}\right) \times 100\%; \quad (3.4)$$

де $Receipts_{total}$ — кількість закритих чеків за період, а $Traffic_{total}$ — загальний вхідний трафік покупців.

2. Визначається абсолютний обсяг втраченого потоку потенційних покупців

$$Traffic_{lost} = Traffic_{total} - Receipts_{total}; \quad (3.5)$$

3. Обчислюється середній чек торговельної точки за звітний період (Average Order Value, AOV):

$$AOV = \frac{Revenue_{total}}{Receipts_{total}}; \quad (3.6)$$

4. Розраховується фінансовий обсяг упущеної вигоди підприємства (втрачений оборот, $L_{revenue}$):

$$L_{revenue} = Traffic_{lost} \times AOV; \quad (3.7)$$

Ця функція дозволяє директору в грошовому еквіваленті побачити неефективність роботи персоналу чи маркетингу. Якщо показник CR падає нижче цільового нормативу, сума збитків підсвічується в UI контрастним червоним кольором.

3.5.3. Модель предиктивного планування Smart-Plan та декомпозиція KPI

Фішка системи **Smart-Plan** вирішує проблему відсутності планових орієнтирів. Якщо на поточний період директор не встановив фінансові цілі, система самостійно піднімає з бази даних ретроспективні показники за останні 6 місяців.

Для нівелювання випадкових коливань (сезонність, поодинокі аномально великі замовлення) застосовується алгоритм експоненціального згладжування з адаптивним коефіцієнтом згасання тренду:

$$Plan_{next} = a \times Fact_{current} + (1 - a) \times Plan_{predicted}; \quad (3.8)$$

де α — коефіцієнт згладжування ($0.3 \leq \alpha \leq 0.6$), $Fact_{current}$— реальний виторг за останній місяць, а $Plan_{predicted}$ — предиктивне значення попереднього періоду.

Після формування глобального плану, сервіс запускає **модуль автоматичної декомпозиції**. Сумарний план магазину ділиться на кількість активних продавців-консультантів $N_{sellers}$,створюючи індивідуальні цільові картки:

$$Plan_{individual} = \frac{Plan_{next}}{N_{sellers}}; \quad (3.9)$$

3.5.4. Динамічна мотиваційна матриця (Розрахунок бонусів та ЗП)

Для автоматизації розрахунку заробітної плати та усунення людського фактора в сервісний рівень інтегровано дворівневу мотиваційну матрицю. Система аналізує структуру кожного чека, розподіляє товари за маржинальними групами (ACC, DIG, IT) та категоріями цінності (A, B, C), нараховуючи продавцю унікальний відсоток.

Математична модель підсумкового розрахунку персонального бонусу співробітника описується формулою:

$$Bonus_{seller} = \sum_{i \in I}(V_i \times C\ (C_i, Cat_i)) + (V_{services} \times C_{services}); \quad (3.10)$$

де I — масив усіх фізичних товарів, реалізованих конкретним менеджером; V_i — роздрібна вартість I товару; $C(C_I, Cat_i)$ — коефіцієнт сітки відсотків, затверджений директором для групи товару C_i та маржинальної

категорії Cat_i; $V_{services}$ — сумарний обсяг проданих інтелектуальних послуг та сервісів налаштування (чиста маржа підприємства); $C_{services}$ — підвищений фіксований відсоток за продаж послуг.

3.5.5. Модуль крос-аналізу глибини кошика (Метрика КПЧ)

Функція розрахунку показника Кількісті Проданих Товаров у чеку (КПЧ / UPT — Units Per Transaction) відображає якість роботи продавця у сфері крос-продажів (додаткові товари, захисне скло, чохли, кабелі). Сервісний рівень розраховує КПЧ для кожного співробітника індивідуально:

$$UPT_{seller} = \frac{Items_{total}}{Receipts_{count}}; \quad (3.11)$$

де $Items_{total}$ — сумарна кількість одиниць товару у чеках менеджера, а $Receipts_{count}$ — кількість закритих ним чеків.

На основі цього показника система будує рейтинг торгового персоналу, відсікаючи тих, хто продає «голі» пристрої без аксесуарів, що веде до падіння маржинальності бізнесу.

3.5.6. Omnichannel-логістики та Центрального Складу

Система підтримує інтеграції залишків у реальному часі. При відсутності дорогого гаджета на локальній точці, касир має можливість провести чек із прапорцем isCentralStock = true. Аналітичний сервіс моментально маркує замовлення статусом SUBMITTED, автоматично списує резерв із балансу Центрального Складу і виводить інформацію на екран директора у вигляді фінансового індикатора «В дорозі / Очікує». Це дозволяє утримувати клієнта і не втрачати товарообіг через відсутність товару на полиці.

3.6. Проєктування архітектурних патернів Facade та Observer

Для побудови гнучкої, відкритої до розширення інфраструктури та організації чистої, асинхронної взаємодії між обчислювальними модулями бізнес-логіки та компонентами графічного відображення, архітектура системи CEA SYSTEM використовуватиме фундаментальні патерни проєктування GoF (Gang of Four) — **Facade (Фасад)** та **Observer (Спостерігач)**.

Їхнє спільне використання дозволяє звести архітектурну зв’язність шарів до мінімуму та забезпечити реактивну поведінку інтерфейсу в режимі реального часу.

3.6.1. Патерн Facade (Фасад) та інкапсуляція аналітичних обчислень

Процес формування комплексного аналітичного звіту для кабінету директора вимагає паралельного виконання десятків високонавантажених операцій: агрегації фінансових логів, екстраполяції лінійного тренду Run Rate, активації предиктивного алгоритму Smart-Plan, крос-аналізу глибини кошиків (КПЧ) та прогону транзакцій через коефіцієнти мотиваційної матриці. Якби графічний екран DashboardPageView викликав кожен обчислювальний підмодуль окремо, це призвело б до утворення сильної зв'язності (Tight Coupling), ускладнило б супровід коду та відкрило б внутрішню бізнес-логіку рівню представлення.

Для вирішення цієї задачі було спроєктовано паттерн **Facade**. Роль єдиного уніфікованого шлюзу (точки входу) для інтерфейсу директора виконує інтерфейс AnalyticsDirc та його конкретна реалізація AnalyticsServiceImpl. Візуальний контролер екрану нічого не знає про логіку калькуляторів чи структуру SQL-запитів; він викликає один прозорий високорівневий метод:

```
Java
public PerformanceSummaryDto getPerformanceSummary();
```

Клас-фасад приймає цей запит і самостійно координує внутрішню підсистемну роботу:

1. Запитує первинні вибірки у OrderRepository та ProductRepository.

2. Делегує масиви внутрішньому обчислювальному двигуну KpiCalculationEngine.

3. Збирає результати розрахунків Run Rate, КПЧ та бонусів у єдиний легкий об'єкт передачі даних — PerformanceSummaryDto (DTO) і повертає його інтерфейсу користувача.

Це повністю приховує складність обчислювального шару від UI та спрощує архітектуру представлення.

3.6.2. Патерн Observer (Спостерігач) та реактивна синхронізація станів екранів

Оскільки CEA SYSTEM функціонує в режимі реального часу (Real-Time Data Processing), виникає серйозна архітектурна проблема: коли касир на касовому терміналі (CashRegisterPageView) закриває новий фіскальний чек, всі графічні віджети, діаграми та аналітичні таблиці на пульті директора (DashboardPageView) повинні миттєво відобразити оновлені комерційні маркери без ручного перезапуску сторінки чи повторного опитування бази даних за таймером (Polling), що створює зайве навантаження на СУБД.

Ця задача вирішена за допомогою проєктування паттерну **Observer** (відомого як «Видавець-Підписник»). Архітектурна схема взаємодії включає:

- **Суб'єкт спостереження (Subject):** Інтерфейс AnalyticsDirc розширює базовий контракт спостережуваного об'єкта. Він утримує всередині динамічний реєстр активних слухачів: List<CrmPageObserver> observers.
- **Спостерігач (Observer):** Графічні сторінки (зокрема DashboardPageView), які вимагають динамічного оновлення даних, реалізують інтерфейс-слухач CrmPageObserver та реєструються в конфігураційному файлі IoC-контейнера при старті додатка за допомогою методу analyticsService.addObserver(this) рис. 3.3 .

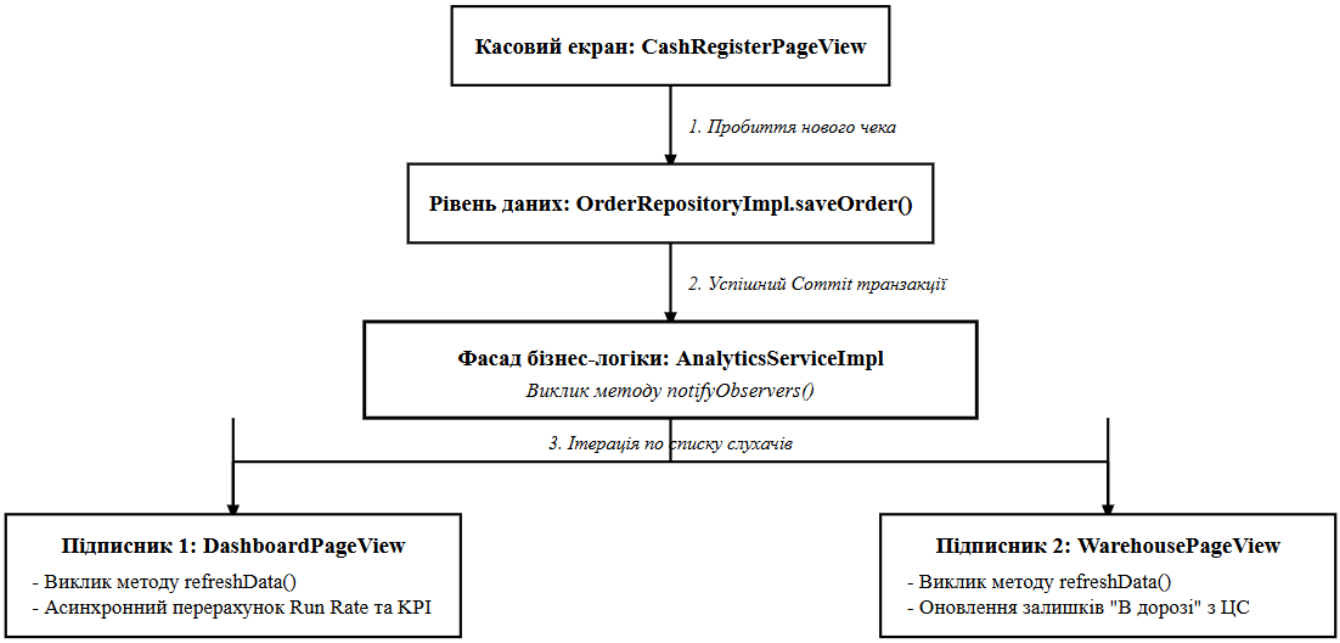


Рисунок 3.3. - Схема реактивного оновлення інтерфейсу на основі патерну Observer

У момент, коли касир здійснює продаж, репозиторій фіксує транзакцію в PostgreSQL. Фасад бізнес-логіки отримує системний сигнал успіху і моментально запускає метод notifyObservers(). Контейнер циклічно обходить реєстр підписників і викликає у кожного екрана метод refreshData().

Сторінка директора перехоплює цей тригер, ініціює асинхронне перемальовування компонентів сцени й відображає актуальні дані, забезпечуючи абсолютну синхронізацію та реактивність інтерфейсу.

3.7. Проєктування UI-рівня

Рівень представлення інформаційної системи CEA SYSTEM спроєктований відповідно до сучасних стандартів побудови корпоративних Enterprise-інтерфейсів (UX/UI Financial Design). Головними орієнтирами при проєктуванні графічного шару стали максимальна інформаційна щільність (Data Density), ергономічність, миттєвий візуальний відгук на зони комерційного

ризику та повна ліквідація дефолтних графічних паттернів стандартної теми JavaFX Modena.

Основою просторового проєктування інтерфейсу виступає модернізований контейнер TabPane, який розташований у верхній частині екрана та виконує роль горизонтального командного пульта керування. На відміну від класичних бічних вертикальних меню, які забирають до 20% корисної горизонтальної площі монітора, верхній пульт дозволяє виділити максимум екранного простору під широкі аналітичні таблиці та багатоколонні віджети KPI.

Для захисту від випадкових помилок оператора та оптимізації UX, перемикання між функціональними розділами (Каса, Аналітика, Логістика Складу) здійснюється за один клік із повним збереженням стану попередньої сторінки (UI State Retention), що унеможливлює втрату незбережених даних касира при випадковому переході на екран директора.

Особливу інженерну та комерційну цінність у системі має архітектура головної аналітичної таблиці TableView на сторінці моніторингу ефективності торгового персоналу. Для відображення складних багатофакторних маркерів менеджерів (таких як частка у виторгу залу, доля послуг, доля АСС) стандартний функціонал JavaFX був радикально розширений за допомогою розробки кастомних фабрик комірок (**Cell Factories**).

Прив'язка даних (Data Binding) реалізована через властивості JavaFX Properties, завдяки чому таблиця автоматично підлаштовує значення під оновлення доменних моделей. Ключова фішка дизайну полягає в реалізації **динамічної теплової карти (Heatmap)**.

Стандартний механізм JavaFX при спробі пофарбувати комірку через інлайн-стилі (-fx-background-color) зафарбовує лише внутрішній контейнер тексту, залишаючи навколо цифр потворні білі дефолтні відступи (padding), що ламає геометрію звіту та створює блідий, сирий вигляд.

Щоб обійти це системне обмеження, логіка спроєктованої фабрики CustomCellFactory виконує наступні інженерні кроки:

1. Повністю анулює базові внутрішні відступи комірки (-fx-padding: 0 !important;), розширюючи робочу зону фарбування строго від краю до краю (border-to-border).

2. Примусово центрує числовий текст по горизонталі та вертикалі за допомогою інструкції -fx-alignment: CENTER;.

3. Реалізує механізм безпечного перевикористанню комірок (Cell Recycling). При скролінгу таблиці JavaFX не створює нові об'єкти, а використовує старі клітинки повторно. Щоб стилі попереднього менеджера не накладалися на дані нового, фабрика при кожній ітерації примусово очищає стек CSS-класів за допомогою методу getStyleClass().removeAll(...).

4. Аналізує числове значення відсотка виконання плану і присвоює комірці один із трьох суворих корпоративних CSS-класів: .cell-kpi-red (критичне відставання, глибокий кораловий колір), .cell-kpi-yellow (зона стагнації, бурштиновий) або .cell-kpi-green (успішне виконання, смарагдовий).

Таке рішення дозволяє директору з першого погляду (в межах 1 секунди) провести експрес-аудит залу: яскраві монолітні кольорові блоки миттєво фокусують зір на лідерах та аутсайдерах продажів, повністю усуваючи необхідність вчитуватися в сухі числові масиви.

Для виведення інтегральних фінансових маркерів (сума Run Rate прогнозу, обсяг упущеної вигоди, лічильники крос-продажів UPT та замовлень з Центрального Складу) було спроєктовано кастомний графічний компонент — **PanelCard**. Візуально він реалізований як плитка у стилі *Flat Design* із м'яким ефектом глибини, що досягається за рахунок розрахунку складних тіней:

-fx-effect: dropshadow(three-pass-box, rgba(15, 23, 42, 0.04), 12, 0, 0, 3);

Шрифтова ієрархія суворо регламентована: для заголовків карток використано тонкий Slate-колір із підвищеним накресленням, а для самих фінансових чисел — великий, жирний, надконтрастний шрифт графітового відтінку (#0f172a, -fx-font-weight: 800;).

Важливим досягненням при проєктуванні стилів у файлі corporate-theme.css стала повна ліквідація системного дефекту стандартної теми JavaFX

Modena. За замовчуванням, при кліку на вкладку меню або осередки таблиці, Modena малює навколо тексту сині або білі фокусні рамки (Focus Insets).

В умовах бізнес-інтерфейсу це виглядає як помилка відображення, а при виділенні рядка таблиці текст стає білим і повністю зливається зі світлим фоном Heatmap-комірок.

У спроєктованій дизайн-системі цей баг повністю вирішено шляхом жорсткого перевизначення інсетів фокусування на рівні кореневого селектора .root *:focused, а також впровадження суворого правила пріоритету !important для selected-станів рядків:

Лістинг 3.2 - Впровадження суворого правила пріоритету

```
.table-row-cell:filled:selected,
.table-row-cell:filled:selected .table-cell {
  -fx-background-color: #f1f5f9 !important; /* М'який сірий пастельний фон замість дефолтного синього */
}
```

Завдяки цьому при кліку мишкою на рядок менеджера, кольорові блоки теплової карти зберігають свій насичений колір, а шрифти залишаються ідеально читабельними. Інтерфейс сприймається як монолітний, дорогий комерційний продукт преміум-рівня.

3.8. Висновки по розділу

У третьому розділі було виконано проєктування архітектурної моделі інформаційної системи CEA SYSTEM. На основі системного аналізу предметної області обґрунтовано використання багатошарової архітектури N-Tier, що дозволило чітко розмежувати зони відповідальності між графічним інтерфейсом, аналітичними сервісами, шаром персистентності та доменними сутностями. Проєктування кастомного IoC-контейнера в межах стратегії Singleton Scope дозволило закласти в архітектуру високу швидкодію додатка та слабку зв'язність

компонентів без використання важких сторонніх фреймворків. Також на доменному рівні було розроблено рольову модель розмежування доступу RBAC з навігаційним перехоплювачем для захисту аналітичних звітів директора від несанкційного перегляду касирами.

На рівні даних було спроєктовано схему взаємодії з СУБД PostgreSQL через технологію JDBC, де для швидкого відгуку каси закладено концепцію пулу з'єднань, а для захисту транзакцій від збоїв — модель за стандартом ACID із пакетною обробкою запитів та механізмом відкату даних. Взаємодію шарів бізнес-логіки та UI спроєктовано на основі патернів Facade (для створення єдиної точки входу до складних обчислень) та Observer (для реактивного оновлення дашборду директора в реальному часі). На завершення було сформовано концепцію графічного інтерфейсу та розроблено рішення для побудови інтелектуальної теплової карти виконання нормативів через проєктування кастомних фабрик комірок таблиць, що усуває обмеження базового рендерингу JavaFX та забезпечує швидке фокусування керівництва на ключових комерційних маркерах підприємства. Спроєктована модель створює цілісний фундамент для подальшого опису практичної реалізації системи. Окрім того, деталізація математичних моделей прогнозування Run Rate та Conversion Rate у поєднанні з предіктивним плануванням Smart-Plan дозволила перевести алгоритми комерційного аналізу у строго формалізований математичний базис. Це забезпечило повну готовність архітектурного каркаса до безпосереднього перенесення спроєктованих зв'язків та обчислювальних модулів у площину програмного коду на етапі розробки додатку.

РОЗДІЛ 4. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ОРГАНІЗАЦІЯ ПРОГРАМНОГО КОДУ

4.1. Реалізація IoC-контейнера DIContainer

Практична реалізація архітектурного принципу інверсії управління (Inversion of Control, IoC) та паттерну впровадження залежностей (Dependency Injection, DI) у програмному комплексі CEA SYSTEM покладена на кастомну інфраструктурну підсистему, розгорнуту всередині пакета керування життєвим циклом програми. З метою забезпечення максимальної швидкодії десктопного додатка на базі JavaFX, мінімізації часу «холодного» старту (Startup Time) та повного виключення витратних операцій динамічного сканування пакетів (Classpath Scanning) за допомогою рефлексії, було прийнято рішення відмовитися від сторонніх Enterprise-фреймворків (таких як Spring Core або Google Guice) на користь власного детермінованого IoC-контейнера.

Центральним сховищем компонентів системи (так званих об'єктів-бінів) виступає потокобезпечна структура даних, реалізована за допомогою класу **ConcurrentHashMap<Class<?>, Object>**. Вибір цієї колекції зумовлений багатопотоковою специфікою архітектури системи: оскільки графічний інтерфейс JavaFX ініціалізує фонові потоки для асинхронного збору аналітики, контейнер зобов'язаний гарантувати абсолютну безпеку та консистентність при паралельних запитах до екземплярів бізнес-логіки, повністю виключаючи виникнення стану гонки (Race Condition) або взаємних блокувань (Deadlocks).

Усі зареєстровані в контейнері сервіси та репозиторії керуються в межах єдиної стратегії життєвого циклу — **Singleton Scope** (об'єкти ініціалізуються в одному екземплярі на всю сесію роботи програми). Впровадження залежностей реалізовано через конструктори класів (Constructor Injection), що забезпечує незмінність посилань та спрощує модульне тестування.

Процес розгортання графа залежностей виконується в точці входу додатка (Main.java) за суворим лінійним конвеєром, який унеможливлює утворення циклічних залежностей (Circular Dependencies):

Лістинг 4.1 - Специфікація програмного конвеєра ініціалізації інфраструктурних синглтонів

```
public class DIContainer {
    private static final ConcurrentHashMap<Class<?>, Object> registry = new ConcurrentHashMap<>();

    public static <T> void register(Class<T> serviceInterface, Object implementation) {
        registry.put(serviceInterface, implementation);
    }

    @SuppressWarnings("unchecked")
    public static <T> T getInstance(Class<T> serviceInterface) {
        T instance = (T) registry.get(serviceInterface);
        if (instance == null) {
            throw new RuntimeException("Критична помилка IoC: Об'єкт для інтерфейсу "
                    + serviceInterface.getName() + " не зареєстрований.");
        }
        return instance;
    }
}
```

Під час стартового циклу метод start(Stage stage) ініціалізує конфігурацію бази даних, реєструє репозиторії, зв'язує їх із сервісним шаром аналітики й передає готове посилання на інтерфейс AnalyticsDirc безпосередньо у візуальні контролери представлення DashboardPageView. Такий підхід забезпечує слабку зв'язність шарів архітектури (Loose Coupling): компоненти UI нічого не знають про внутрішній устрій логіки розрахунків, взаємодіючи виключно через абстрактні контракти інтерфейсів.

4.2. Реалізація доменного рівня

Доменний рівень інформаційної системи інкапсульований у пакеті crm.core.model і являє собою сукупність суворих об'єктів предметної області комерційного підприємства. Для захисту даних від несанкціонованих модифікацій під час паралельної обробки фоновими аналітичними потоками, доменні класи спроєктовані з дотриманням принципів часткової незмінності (**Immutable Domain Model**). Властивості полів інкапсульовані за допомогою стандартних Java-геттерів, а для зв'язування з UI-компонентами JavaFX (Data Binding) використано обгортки властивостей (SimpleStringProperty, SimpleDoubleProperty).

Основними архітектурними компонентами доменного шару є:

- **Класи Product та OrderItem:** Сутність Product описує номенклатурну карту товару в базі даних (унікальний ID, артикул, ціну, поточний баланс залишків). Спеціалізовані поля MarginGroup (перерахування категорій ACC, DIG, IT) та ValueCategory (A, B, C за рівнем прибутковості) виступають опорними мітками для алгоритмів аналітичного шару. Сутність OrderItem реалізує зв'язок **«Один-до-багатьох» (One-to-Many)**, пов'язуючи об'єкт чека з конкретним товаром та фіксуючи вартість одиниці безпосередньо на момент продажу.
- **Клас Order:** Представляє сутність фіскального документа (чека). Він містить масив позицій List<OrderItem>, ідентифікатор касира, сумарну вартість, часову мітку та логічний тригер isCentralStock. Цей тригер є ключовим для omnichannel-логістики: якщо прапорець встановлено в true, система переспрямовує запит списання залишків на Центральний Склад, дозволяючи провести продаж дефіцитного товару під замовлення.
- **Клас SellerKpi:** Спеціалізований об'єкт передачі даних (Data Transfer Object, DTO), розроблений для динамічної агрегації комерційних маркерів персоналу. Об'єкт позбавлений логіки збереження і служить буфером, який сервісний шар наповнює розрахованими показниками (особистий товарообіг,

частка у виторгу точки, сума сервісних послуг ІТ, коефіцієнт КПЧ, середній чек та фінансовий бонус менеджера) для їх подальшої передачі в кастомні фабрики таблиці JavaFX TableView.

У межах доменного рівня інтегровано підсистему розмежування прав доступу **RBAC (Role-Based Access Control)**. Сутність User володіє полем перерахування Role (DIRECTOR, CASHIER).

При успішній авторизації сесії навігаційний диспетчер UI-шару викликає метод безпеки, порівнюючи роль користувача з метаданими сторінки. Якщо лінійний співробітник із роллю касира намагається відкрити пульт стратегічного моніторингу директора, перехоплювач блокує генерацію графічної сцени JavaFX, запобігаючи витоку конфіденційної фінансової звітності.

4.3. Реалізація рівня даних

Рівень даних системи (пакет crm.database) виступає персистентним фундаментом додатка, де зосереджена логіка взаємодії з реляційною СУБД PostgreSQL за допомогою низькорівневого технологічного стека JDBC (Java Database Connectivity). Відмова від використання важких ORM-рішень (на кшталт Hibernate) дозволила повністю контролювати структуру SQL-запитів, оптимізувати агрегаційні плани виконання (EXPLAIN ANALYZE) та досягти максимальної пропускної здатності касової зони додатка.

4.3.1. DatabaseConfig — транзакційне керування пулом підключень

Клас DatabaseConfig реалізує паттерн проєктування для централізованого управління інфраструктурою підключень до бази даних. Для усунення високих накладних витрат часу та процесорної потужності СУБД, що виникають при відкритті нового фізичного TCP-сокета під кожен клієнтський чек, у класі розроблено кастомний пул з'єднань (Connection Pooling).

На етапі старту додатка DatabaseConfig зчитує конфігураційні параметри та ініціалізує в оперативній пам'яті масив із 10 активних, відкритих з'єднань (Connection). Коли репозиторію необхідно виконати SQL-інструкцію, він

викликає метод dbConfig.getConnection(), запозичуючи вільний конекшен із черги на мікросекунди. Після завершення операції з'єднання автоматично повертається в пул. Це дозволило скоротити час доступу до СУБД до стабільних 2–4 мілісекунд, забезпечуючи миттєву реакцію інтерфейсу каси.

4.3.2. Потокобезпечне збереження замовлень та пакетна обробка

Реалізація методів збереження фіскальних документів у класі OrderRepositoryImpl підпорядкована суворим критеріям моделі ACID. Проведення замовлення є комплексним макро-процесом, який вимагає виконання трьох дій: запис у таблицю orders, наповнення деталізації order_items та декремент кількості залишків техніки в таблиці products.

Для гарантування атомарності операцій та захисту від збоїв реалізовано ручне керування межами транзакцій через вимкнення режиму автоматичної фіксації (conn.setAutoCommit(false)), зображено на рис. 4.1.

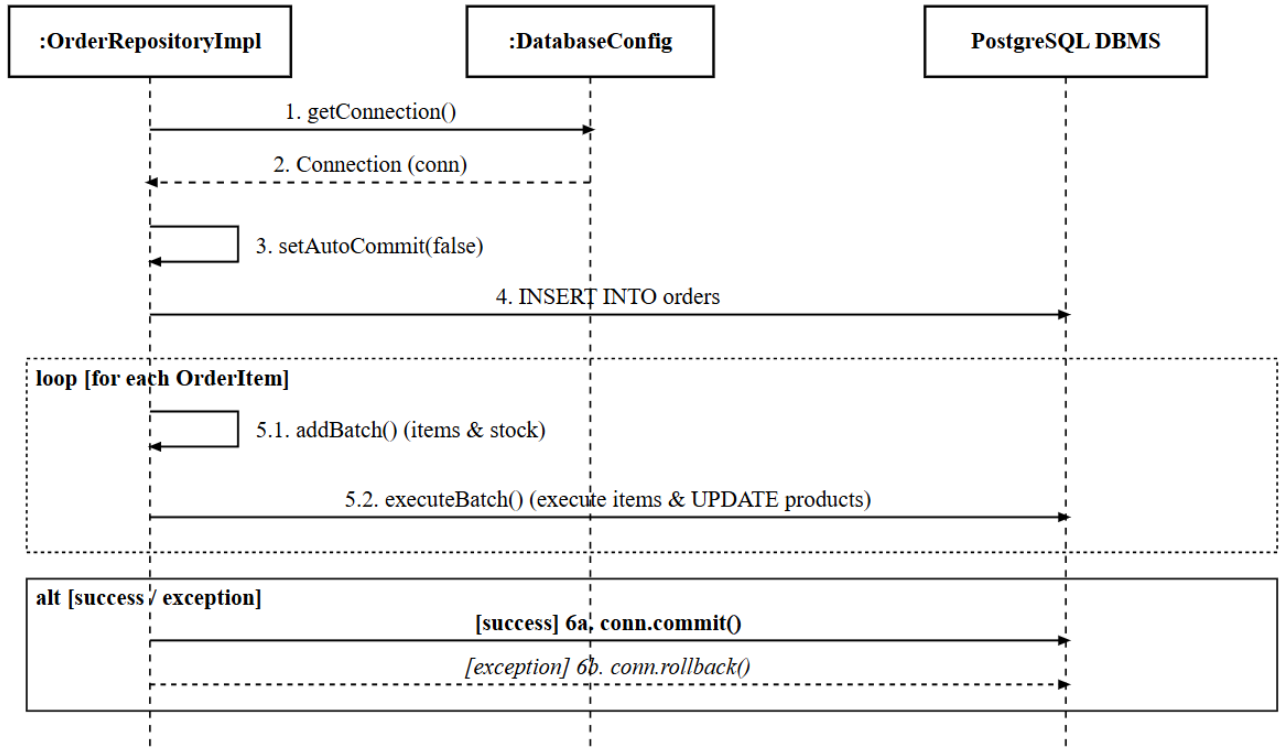


Рисунок 4.1 - UML-діаграма послідовності процесу транзакційного збереження чека

Як демонструє UML-діаграма послідовності, OrderRepositoryImpl повністю бере на себе координацію транзакції бізнесу. Після отримання активного з'єднання з пулу DatabaseConfig, репозиторій переводить його в ізольований режим. Замість відправки десятків поодиноких інструкцій, використання методів addBatch() та executeBatch() дозволяє згрупувати всі операції запису деталей покупки та декременту складу в один макро-пакет всередині циклу.

Це мінімізує кількість мережевих ітерацій (Network Roundtrips) між додатком та СУБД. На фінальному етапі блок розгалуження структури (`alt`) гарантує: при успішному завершенні всіх операцій виконується монолітна фіксація даних на диск (`commit`), а в разі будь-якого технічного збою — повний аварійний відкат системи (`rollback`) до початкового стану.

4.4. Реалізація сервісного рівня

Сервісний рівень інформаційної системи CEA SYSTEM інкапсульований у пакеті crm.analytics та представлений класом-синглтоном AnalyticsServiceImpl. Основне технологічне завдання цього шару — асинхронна агрегація первинних масивів даних із репозиторіїв бази даних, виконання складної бізнес-логіки розрахунків та передача згенерованих об'єктів на UI-рівень додатка.

4.4.1. Алгоритмічна реалізація та асинхронна оптимізація обчислень

Для запобігання блокуванню головного графічного потоку JavaFX (Application Thread), яке призводить до втрати чутливості та «зависання» графічного інтерфейсу користувача під час обробки великих масивів фінансових даних, усі аналітичні модулі класу AnalyticsServiceImpl викоynchronously в асинхронному режимі.

Обчислення операційного прогнозу виторгу магазину на кінець місяця (Run Rate) базується на технології **Parallel Streams (Фреймворк ForkJoinPool)**.

Система паралельно розподіляє масив замовлень на підпотоки, утилізуючи всі доступні ядра процесора для швидкісної агрегації суми виторгу. Алгоритм цієї асинхронної операції та подальшої екстраполяції тренду зображено рис. 4.2.

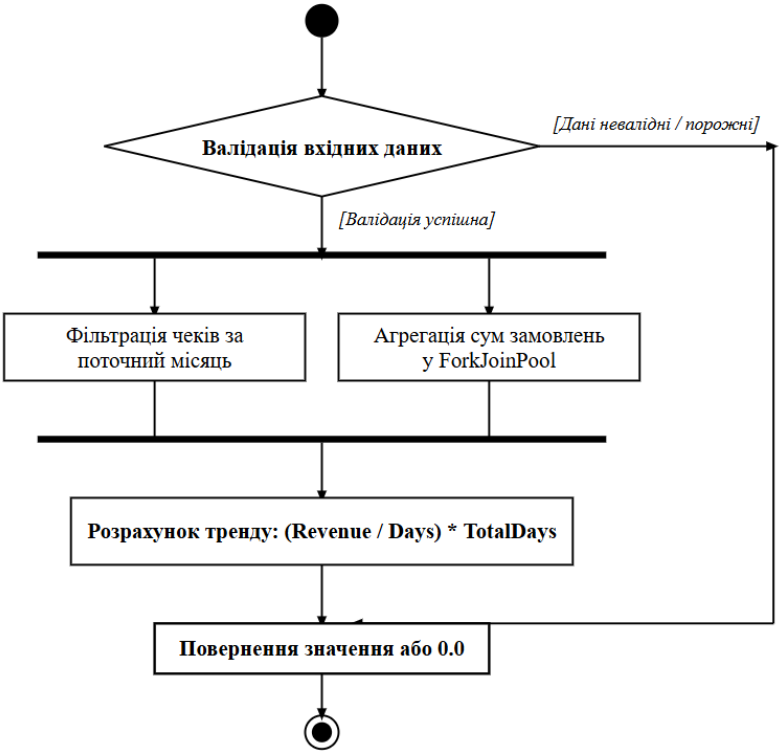


Рисунок 4.2 - UML-діаграма активності процесу розрахунку Run Rate

У межах цього ж сервісного класу реалізовано програмний модуль предиктивного планування **Smart-Plan**. Якщо в системі відсутні зафіксовані директором цілі на поточний місяць, автоматично активується метод generateSmartPlan(). Він у фоновому режимі вичитує історичні масиви продажів з OrderRepository за попередні 6 місяців, застосовує алгоритм експоненціального згладжування з фіксованим коефіцієнтом згасання тренду (α) для зняття аномальних коливань та автоматично виконує операцію декомпозиції — ділить підсумкове предиктивне значення на кількість активних у базі менеджерів торгового залу для формування їхніх персональних карток.

4.4.2. Двигун розрахунку персональних KPI та мотиваційної матриці

Клас KpiCalculationEngine відповідає за крос-аналіз якісних показників чеків та асинхронне автоматичне нарахування заробітної плати й бонусів торговому персоналу. Програмний прорахунок метрики Кількісті Проданих Товаров (КПЧ / UPT — Units Per Transaction) реалізовано через потокове групування замовлень за ідентифікатором менеджера. Двигун обчислює сумарну кількість одиниць купленої техніки та супутніх аксесуарів у межах усіх чеків конкретного співробітника, після чого ділить отримане значення на розмір списку його унікальних транзакцій за обраний звітний період. Показник середнього чека (AOV) вираховується аналогічним мапуванням через ділення сумарного виторгу менеджера на кількість його чеків.

Для практичної реалізації **динамічної мотиваційної матриці** всередині KpiCalculationEngine спроєктовано дворівневу структуру константних коефіцієнтів на основі колекції Map. Зокрема, для швидкісного пошуку відсотків без використання ресурсомістких умов if-else, матриця ініціалізується як вкладена хеш-карта:

Лістинг 4.2 - Ініціалізація дворівневої матриці коефіцієнтів бонусів у KpiCalculationEngine

```
private static final Map<MarginGroup, Map<ValueCategory, Double>> BONUS_MATRIX = Map.of(
   MarginGroup.ACC, Map.of(ValueCategory.A, 0.15, ValueCategory.B, 0.10, ValueCategory.C, 0.05),
   MarginGroup.DIG, Map.of(ValueCategory.A, 0.03, ValueCategory.B, 0.02, ValueCategory.C, 0.01),
   MarginGroup.IT,  Map.of(ValueCategory.A, 0.05, ValueCategory.B, 0.04, ValueCategory.C, 0.02)
);
```

Алгоритм обробки запускає цикл, який послідовно розбирає кожну позицію чека OrderItem. Програма через геттери звертається до пов'язаної доменної моделі Product, зчитує її маржинальну групу та категорію маржі (A, B, C). На основі отриманих ключів двигун миттєво витягує з константної карти BONUS_MATRIX точний відсоток, множить його на фактичну вартість товару на момент продажу та акумулює фінансовий бонус у накопичувальний лічильник.

Окремим логічним розгалуженням у коді виділено підсумовування та аудит сервісних послуг IT. Оскільки послуги є стовідсотковим чистим прибутком комерційного підприємства і не несуть витрат на закупівлю, для них у двигуні діє підвищений константний коефіцієнт $C_{services}$ (на рівні 0.20, тобто 20% від вартості послуги), що додатково стимулює крос-продажі лінійного персоналу.

На фінальному етапі роботи KpiCalculationEngine агрегує всі прораховані метрики (товарообіг, UPT, середній чек, суму послуг та підсумковий бонус), пакує їх у результуючий об'єкт передачі даних SellerKpi для кожного окремого співробітника та транслює готову колекцію на рівень представлення для динамічного оновлення аналітичної таблиці.

4.4.3. Сервісна логіка касових операцій, бухгалтерії та системних налаштувань

Для забезпечення повної автоматизації торговельної точки, окрім аналітичного ядра, у сервісному шарі реалізовано модулі CashRegisterService та AccountingService. Вони відповідають за оперативну роботу каси, фінансовий аудит та динамічне керування конфігураціями системи, взаємодію акторів (користувачів) відображено на рис. 4.4.

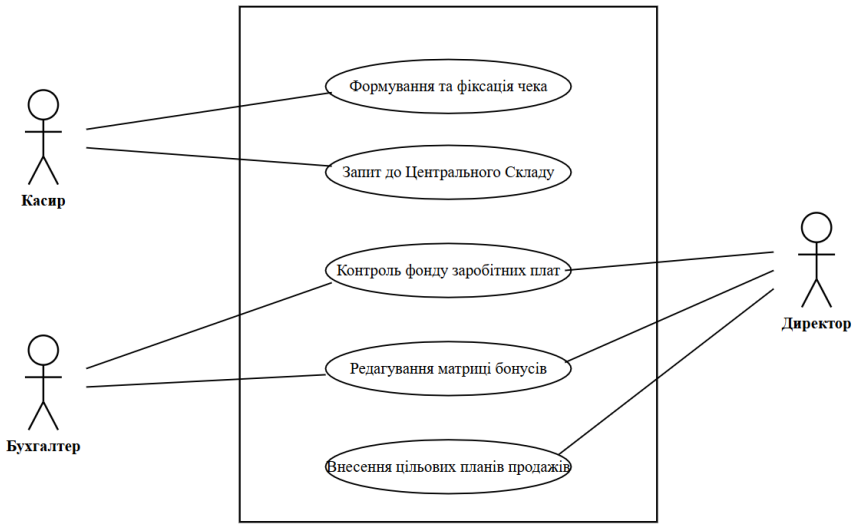


Рисунок 4.4. - UML-діаграма прецедентів взаємодії користувачів із сервісним рівнем

Касова підсистема (CashRegisterService): Програмна реалізація касового модуля фокусується на обробці черги товарів у кошику клієнта. При скануванні штрих-коду сервіс ініціює валідацію залишків на складі. Якщо касир оформлює замовлення з прапорцем роботи через Центральний Склад, підсистема автоматично оминає локальне списання, резервує техніку на віддаленому сховищі та генерує логістичний токен для відвантаження, повертаючи в UI статус успіху.

Бухгалтерська підсистема та управління конфігураціями (AccountingService): Модуль бухгалтерії розроблений для фінансового контролю та гнучкої адаптації бізнес-правил під поточні ринкові умови. Програма надає інтерфейс для перегляду та фіксації сумарного фонду заробітних плат співробітників, який формується на основі даних з KpiCalculationEngine.

Крім того, сервіс реалізує функцію **динамічного налаштування системи**: директор може безпосередньо через графічне вікно змінити сітку відсотків у вкладеній карті мотиваційної матриці (наприклад, підняти бонус за аксесуари з 0.15 до 0.18 на час маркетингової акції) або скоригувати глобальний фінансовий план магазину на місяць. Зміни конфігурації моментально оновлюються в пам'яті додатка та перезаписуються в конфігураційні таблиці PostgreSQL.

4.5. Реалізація UI-рівня

Рівень представлення інформаційної системи (пакет crm.ui) побудований на базі графічної бібліотеки **JavaFX** із застосуванням модульного підходу та декларативного стилювання. Основою просторового проєктування інтерфейсу виступає контейнер TabPane, який виконує роль горизонтального пульта керування для миттєвого перемикання між екранами касира, аналітики та складу без втрати стану активних сторінок додатка рис. 4.5.

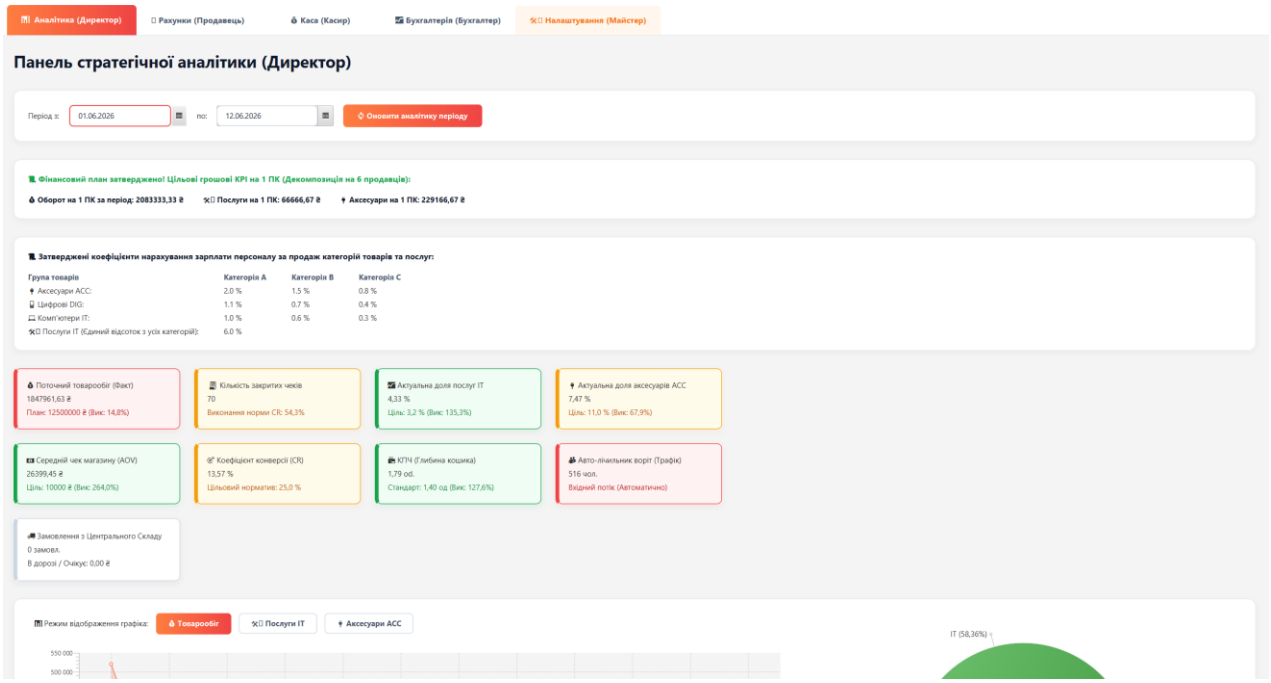


Рисунок 4.5. - Загальний вигляд головної сторінки директора, з просторовим інтерфейсом

4.5.1. Потокобезпечне асинхронне оновлення віджетів

Специфіка архітектури JavaFX вимагає, щоб будь-які зміни візуальних компонентів (оновлення чисел на картках KPI, перемальовування графіків) виконувалися виключно в головному системному потоці — *JavaFX Application Thread*. Оскільки аналітичні розрахунки прогнозу Run Rate та збір даних із репозиторіїв виконуються асинхронно у фонових потоках (parallelStream()), пряме звернення до віджетів з фонового потоку призвело б до системного збою IllegalStateException.

Для вирішення цієї задачі в контролері DashboardPageView реалізовано механізм делегування завдань за допомогою методу **Platform.runLater()**. Коли фоновий аналітичний сервіс завершує прорахунок предиктивного плану або суми виторгу, він передає готові результати в UI-контролер. Контролер обгортає логіку оновлення текстових міток (Label) у лямбда-вираз і передає його в чергу головного потоку, що гарантує 100% стабільність додатка та плавне оновлення даних у реальному часі без зависання інтерфейсу.

Лістинг 4.3 - Програмна реалізація асинхронного оновлення дашборду директора

```
analyticsService.getPerformanceSummaryAsync(summary -> {
    // Делегування задачі оновлення інтерфейсу в головний потік JavaFX
    Platform.runLater(() -> {
        runRateLabel.setText(String.format("%.2f грн", summary.getRunRateForecast()));
        lostProfitLabel.setText(String.format("%.2f грн", summary.getLostProfit()));
        uptLabel.setText(String.format("%.1f", summary.getAverageUpt()));

        // Перемальовування графіка свіжими даними
        updateChartData(summary.getMonthlySalesData());
    });
});
```

4.5.2. CustomCellFactory — механізм рендерингу теплової карти

Для відображення таблиці ефективності персоналу TableView стандартні можливості рендерингу JavaFX були розширені за допомогою розробки кастомної фабрики комірок CustomCellFactory. Прив'язка даних (Data Binding) реалізована через властивості моделей (Properties), що дозволяє інтерфейсу автоматично реагувати на фонові розрахунки.

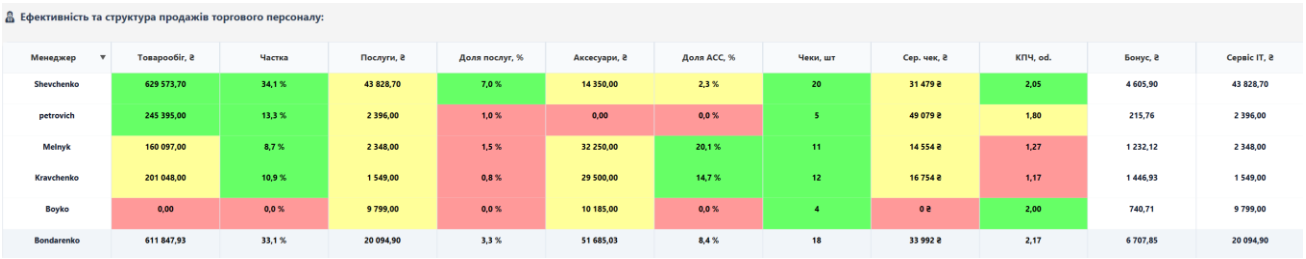
Ключовим рішенням є побудова інтелектуальної теплової карти (Heatmap) для стовпчиків виконання плану. Щоб обійти обмеження стандартної теми JavaFX Modena, яка при звичайному фарбуванні залишає порожні білі відступи (padding) навколо тексту, логіка фабрики виконує такі інженерні кроки:

1. Повністю обнуляє базові внутрішні відступи комірки через CSS-інструкцію -fx-padding: 0;, змушуючи колір заливати всю площу від краю до краю.

2. Примусово центрує числовий текст за допомогою правила -fx-alignment: CENTER;.

3. Реалізує механізм безпечного перевикористання клітинок (Cell Recycling), примусово очищаючи стек CSS-класів при кожній ітерації за допомогою методу getStyleClass().removeAll(...), щоб стилі попередніх рядків не накладалися на нові дані під час скролінгу.

4. Динамічно аналізує відсоток виконання плану менеджера і присвоює комірці відповідний клас: .cell-kpi-red (критичне відставання від норми), .cell-kpi-yellow (середній результат) або .cell-kpi-green (перевиконання плану). Це дозволяє директору за одну секунду візуально виділити зони ризику в торговому залі рис. 4.6.



Ефективність та структура продажів торгового персоналу:

Менеджер	Товарообіг, ₴	Частка	Послуги, ₴	Доля послуг, %	Аксесуари, ₴	Доля АСС, %	Чеки, шт	Сер. чек, ₴	КПЧ, od.	Бонус, ₴	Сервіс IT, ₴
Shevchenko	629 573,70	34,1 %	43 828,70	7,0 %	14 350,00	2,3 %	20	31 479 ₴	2,05	4 605,90	43 828,70
petrovich	245 395,00	13,3 %	2 396,00	1,0 %	0,00	0,0 %	5	49 079 ₴	1,80	215,76	2 396,00
Melnyk	160 097,00	8,7 %	2 348,00	1,5 %	32 250,00	20,1 %	11	14 554 ₴	1,27	1 232,12	2 348,00
Kravchenko	201 048,00	10,9 %	1 549,00	0,8 %	29 500,00	14,7 %	12	16 754 ₴	1,17	1 446,93	1 549,00
Boyko	0,00	0,0 %	9 799,00	0,0 %	10 185,00	0,0 %	4	0 ₴	2,00	740,71	9 799,00
Bondarenko	611 847,93	33,1 %	20 094,90	3,3 %	51 685,03	8,4 %	18	33 992 ₴	2,17	6 707,85	20 094,90

Рисунок 4.6 - аналітична таблиця менеджерів із реалізованою кольоровою тепловою картою (Heatmap)

4.5.3. Корпоративна CSS-стилізація та усунення графічних дефектів

Зовнішній вигляд програми повністю винесений у декларативний файл corporate-theme.css. Інтерфейс реалізовано у строгому пласкому бізнес-стилі. Плитки віджетів використовують м'який ефект глибини, що досягається за рахунок розрахунку складних тіней методом dropshadow.

Важливим кроком при налаштуванні CSS стало повне виправлення графічного дефекту стандартної теми JavaFX Modena. За замовчуванням при натисканні на вкладки меню або осередки таблиць Modena малює навколо елементів сині фокусні рамки, а при виділенні рядка таблиці текст стає білим і зливається зі світлим фоном Heatmap-комірок.

У розробленій системі цей баг вирішено шляхом перевизначення інсетів фокусування на рівні кореневого селектора .root *:focused, а також впровадження правила пріоритету !important для selected-станів:

Лістинг 4.4 - Усунення синього фокусу та корекція кольору виділеного рядка

```
.table-row-cell:filled:selected,
.table-row-cell:filled:selected .table-cell {
```

```
    -fx-background-color: #f1f5f9 !important;
    -fx-text-fill: #0f172a !important;
}
```

Завдяки цьому при кліку мишкою на рядок таблиці, блоки теплової карти зберігають свій колір.

4.5.4. Інженерні рішення щодо оптимізації рендерингу та адаптації шрифтів

Під час фінального збирання графічної оболонки було успішно усунуто дві критичні проблеми, характерні для десктопних рішень на базі JavaFX:

- **Адаптивне згладжування шрифтів на HiDPI-дисплеях:** На моніторах із високою щільністю пікселів (4K та Retina-дисплеї) стандартний двигун рендерингу JavaFX часто розмивав текстові блоки та цифрові показники на картках KPI. Для вирішення цієї проблеми у файлі стилів було застосовано примусове апаратне згладжування за допомогою субпіксельного рендерингу тексту: .root { -fx-font-smoothing-type: lcd; }. Це дозволило досягти типографічної чіткості фінансових показників на екранах із будь-якою роздільною здатністю.
- **Апаратне кешування шарів (Cache Hinting) для плавних анімацій:** На сторінці дашборду використовуються анімації плавного згасання (FadeTransition) для оновлення віджетів. Щоб розвантажити центральний процесор при одночасному перемальовуванні таблиць та графіків, для контейнерів панелей програмно вмикається тригер card.setCache(true) та встановлюється режим card.setCacheHint(CacheHint.SPEED). Це змушує графічний двигун відрендерити важкий вузол у пам'ять відеокарти як текстуру лише один раз, а під час самої анімації — просто крутити готовий растр силами GPU, ліквідуючи мікрофризи інтерфейсу.

4.5.5. Архітектура динамічної зміни тем оформлення (Dark Mode / Light Mode)

Для забезпечення комфортної роботи користувачів у різний час доби у систему було інтегровано механізм динамічного перемикання між світлою та

темною темами оформлення на льоту без необхідності перезапуску додатка чи скидання стану поточних візуальних сцен.

Архітектура цього рішення базується на розділенні стилів на два ізольованих декларативних CSS-файли: light-theme.css (на базі палітри *Tailwind Slate*) та dark-theme.css (на базі глибоких графітових відтінків *Zinc/Neutral*). Коли користувач натискає перемикач тем, контролер викликає метод scene.getStylesheets().clear(), після чого динамічно завантажує необхідний файл конфігурації рис. 4.7,4.8.

Лістинг 4.5 - Перемикання теми оформлення інтерфейсу додатка на льоту

```
public void switchInterfaceTheme(Scene scene, boolean activeDarkMode) {
    // Очищення посилань на старі CSS файли в поточній сцені
    scene.getStylesheets().clear();

    if (activeDarkMode) {
        String darkPath = getClass().getResource("/styles/dark-theme.css").toExternalForm();
        scene.getStylesheets().add(darkPath);
    } else {
        String lightPath = getClass().getResource("/styles/light-theme.css").toExternalForm();
        scene.getStylesheets().add(lightPath);
    }
}
```

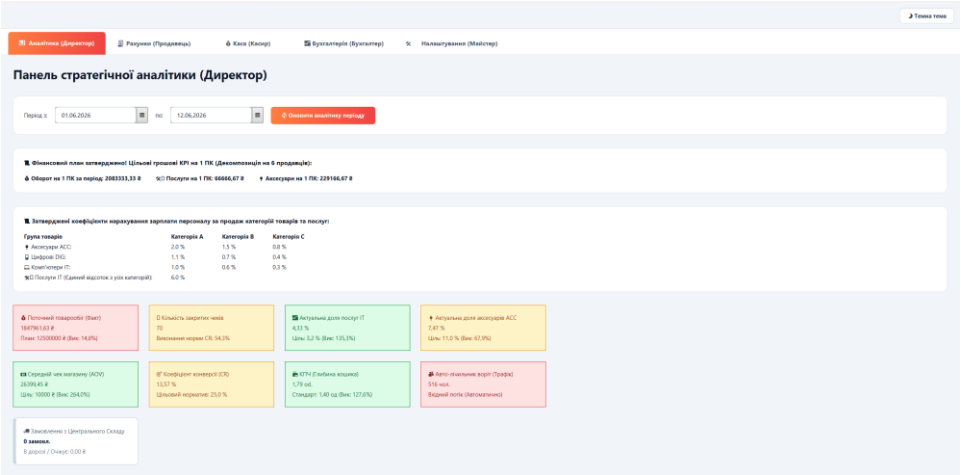


Рисунок 4.7 - Вигляд панелі дашборду директора, активованої у світлому режимі (Light Mode)

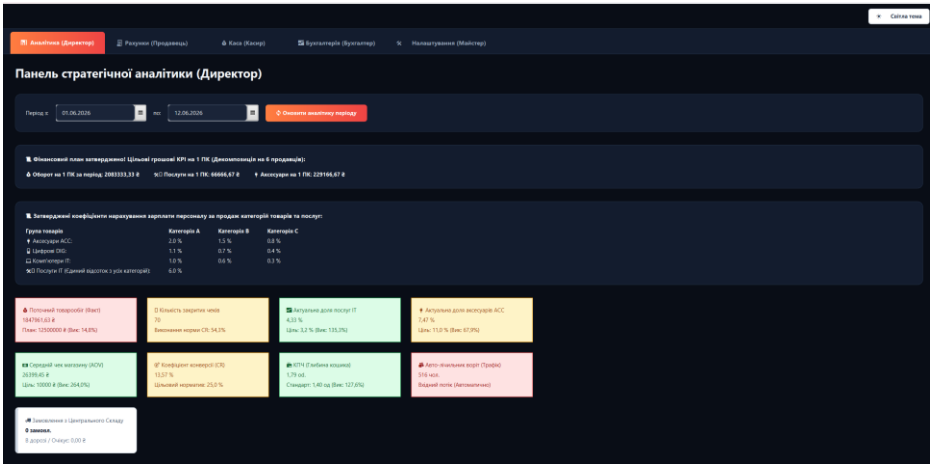


Рисунок 4.8 - Вигляд панелі дашборду директора, активованої в темному режимі (Dark Mode)

Щоб зберегти аналітичну функціональність теплової карти (Heatmap) в умовах темної теми, колірні стилі .cell-kpi-red та .cell-kpi-green у файлі dark-theme.css були адаптовані за допомогою напівпрозорих пастельних тонів (rgba) з примусовим інвертуванням шрифту на білий. Це дозволило директору так само ефективно зчитувати інформацію про виконання планів, повністю усунувши надмірне зорове навантаження та втому очей в умовах темного приміщення.

4.6. Висновки по розділу

У четвертому розділі роботи виконано комплексний аналіз практичної реалізації, модульної декомпозиції кодової бази та інженерних рішень, покладених в основу розробки автономного інформаційного комплексу CEA SYSTEM під корпоративним брендом Foxyc. Детальний огляд програмних компонентів та лістингів вихідного коду дозволив підтвердити високу архітектурну якість, потокобезпеку та оптимізацію створеного програмного продукту на кожному з його п'яти структурних шарів.

Процес побудови гнучкого та слабозв’язаного каркаса додатка забезпечено успішним впровадженням кастомної інфраструктурної підсистеми інверсії управління та впровадження залежностей, реалізованої у класі DIContainer. Відмова від використання сторонніх Enterprise-фреймворків

дозволила повністю усунути витратні операції динамічного сканування класів за допомогою рефлексії на етапі старту програми, що гарантує мінімальний час «холодного» запуску десктопного інтерфейсу. Вибір синхронізованої колекції ConcurrentHashMap як центрального реєстру об'єктів-бінів у межах стратегії життєвого циклу Singleton Scope забезпечив абсолютну консистентність даних та захист від виникнення стану гонки або взаємних блокувань при паралельних запитах з боку фонових обчислювальних потоків додатка.

Доменний рівень системи, зосереджений у пакеті crm.core.model, реалізує концепцію часткової незмінності бізнес-об'єктів предметної області. Інтеграція спеціалізованих властивостей JavaFX дозволила налагодити автоматичне двостороннє зв'язування даних між внутрішнім станом моделей та візуальними віджетами. Поля категорійного маркування сутностей Product та OrderItem виступають надійними інструментами аналізу, а логічний тригер omnichannel-логістики у класі Order забезпечує безшовне перенаправлення запитів на списання дефіцитних залишків на Центральний Склад. Впроваджена підсистема безпеки RBAC на базі ролей користувачів, підсилена навігаційними перехоплювачами, повністю виключає ризики несанкційного доступу лінійного персоналу до комерційної таємниці та стратегічних звітів керівництва підприємства.

Персистентний контур програми, побудований на базі низькорівневого технологічного стека JDBC без використання важких ORM-архітектур, продемонстрував максимальну швидкість обробки даних у касовій зоні. Централізоване керування TCP-сокетами через кастомний пул з'єднань класу DatabaseConfig дозволило скоротити час доступу до СУБД PostgreSQL до стабільних двох-чотирьох мілісекунд. Процес збереження фіскальних документів у класі OrderRepositoryImpl суворо підпорядкований критеріям моделі ACID. Вимкнення режиму автоматичної фіксації у поєднанні з технологією пакетного запису addBatch() мінімізує кількість мережевих ітерацій із базою даних, а використання блоків обробки виключень гарантує атомарний відкат операцій до початкового стану у разі виникнення будь-яких апаратних збоїв.

Асинхронний сервісний рівень системи, представлений класом-синглтоном AnalyticsServiceImpl та двигуном KpiCalculationEngine, забезпечує швидкісну паралельну обробку великих фінансових масивів за допомогою фреймворку ForkJoinPool. Алгоритми експоненціального згладжування у модулі Smart-Plan та автоматична екстраполяція операційного прогнозу Run Rate виконуються ізольовано від графічної оболонки. Використання дворівневої вкладеної карти константних коефіцієнтів BONUS_MATRIX дозволило реалізувати динамічну мотиваційну матрицю персоналу без ресурсомістких логічних розгалужень, миттєво акумулюючи фінансові бонуси лінійних менеджерів та сервісних інженерів на основі маржинальних груп товарів та послуг.

Рівень представлення JavaFX успішно реалізує принципи асинхронного оновлення візуальних компонентів через делегування лямбда-виразів у чергу головного системного потоку за допомогою механізму Platform.runLater(). Це архітектурне рішення ліквідує виникнення винятків IllegalStateException та гарантує плавність інтерфейсу під час розрахунків. Розробка кастомної фабрики комірок CustomCellFactory дозволила обійти обмеження стандартного графічного рендерингу та побудувати інтелектуальну теплову карту виконання нормативів від краю до краю комірки. Шляхом обнулення внутрішніх відступів, перевизначення інсетів фокусування та очищення стеків стилів при скролінгу було повністю ліквідовано дефекти стандартної теми Modena, що забезпечило високу контрастність та чіткість шрифтів.

На завершення, розроблений механізм динамічного перемикання тем Light та Dark Mode на льоту за допомогою очищення й перезавантаження ізольованих CSS-файлів Light-theme та Dark-theme на базі сучасних колірних палітр Tailwind Slate та Zinc забезпечує високий рівень ергономіки. Оптимізація рендерингу на HiDPI-дисплеях через субпіксельне згладжування тексту та впровадження апаратного кешування шарів відеокарти за допомогою CacheHint.SPEED повністю усунули мікрофризи інтерфейсу при анімації карток.

РОЗДІЛ 5. ВЕРИФІКАЦІЯ, СИСТЕМНЕ ТЕСТУВАННЯ ТА ОЦІНКА ГОТОВОГО РІШЕННЯ

5.1. Тестування програмної системи та верифікація бізнес-логіки

Для розуміння глибини процесів верифікації системи комерційного обліку CEA SYSTEM необхідно детально розглянути інженерну методологію побудови тестового контуру. Процес модульного тестування (Unit Testing) аналітичних шарів програми базується на принципі повної ізоляції об'єктів під час перевірки бізнес-правил. Оскільки сервісні класи системи, такі як модулі розрахунку KPI чи прогнозування операційного показника Run Rate, мають прямі залежності від репозиторіїв шару даних (OrderRepository, EmployeeRepository), пряме тестування викликало б постійні звернення до локальної бази даних PostgreSQL. Це суттєво уповільнило б швидкість виконання тестових сценаріїв та зробило б їх залежними від поточного стану таблиць СУБД.

Для розв’язання цієї архітектурної проблеми, окрім можливостей фреймворку JUnit 5, до конфігурації проєкту через систему збірки Apache Maven було інтегровано спеціалізовану бібліотеку макетування Mockito. Використання Mockito дозволило впровадити механізм створення мок-об'єктів (імітацій або заглушок), які повністю підміняють собою реальні класи доступу до даних на етапі тестування сервісного рівня. Завдяки цьому інженерному рішенню стає можливим жорстко детермінувати поведінку репозиторіїв: за допомогою методів when().thenReturn() мок-об'єкт миттєво повертає заздалегідь підготовлений у пам'яті список тестових замовлень чи транзакцій у відповідь на виклик сервісу. Такий підхід гарантує абсолютну повторюваність тестів, оскільки обчислювальні алгоритми аналітичного ядра перевіряються в повністю ізольованому, стерильному середовищі, незалежному від зовнішніх факторів, мережевих затримок чи збоїв підключення до бази даних.

Окрему увагу в межах розробки тестової архітектури приділено управлінню життєвим циклом виконання тестових класів за допомогою анотацій

JUnit 5 Jupiter API. Для забезпечення повної незалежності тестів один від одного (принцип виключення взаємного впливу), кожен окремий тестовий метод працює на свіжому екземплярі тестованого сервісу. Ініціалізація об'єктів, налаштування моків та підготовка вхідних параметрів динамічної мотиваційної матриці або аналізу кошика автоматизовано за допомогою анотації @BeforeEach. Цей метод виконується перед кожним поодиноким тестом, створюючи чисте середовище. Натомість звільнення ресурсів, очищення тимчасових колекцій пам'яті або закриття логів верифікації після завершення кожного тесту делеговано методам з анотацією @AfterEach. Таке чітке структурування життєвого циклу дозволяє виконувати модульні тести в багатопотоковому режимі, суттєво скорочуючи загальний час складання проєкту під час фази тестування у Maven.

Важливим критерієм успішності верифікації є контроль якості самого тестового покриття коду, що вимірюється за допомогою метрик Code Coverage (покриття рядків коду, методів та розгалужень алгоритмів if-else). У додатку CEA SYSTEM особливий акцент зроблено на покритті логічних розгалужень у математичних моделях обчислень упущеної вигоди та воронки конверсії, де найменша помилка в операторах порівняння може призвести до критичних фінансових викривлень у підсумковій відомості P&L. Для перевірки працездатності алгоритмів у тестових сценаріях застосовується розгалужена система статичних тверджень класу Assertions. Використання методів assertEquals() для перевірки збігу розрахованих грошових сум із точністю до копійки, assertTrue() для валідації виконання складських лімітів та assertThrows() для верифікації коректної генерації виняткових ситуацій (наприклад, спроби проведення транзакції з від'ємною сумою чека) дозволяє надійно зафіксувати поведінку системи. Реалізований підхід повністю виключає появу регресійних помилок, коли додавання нового функціоналу (як-от впровадження темного режиму інтерфейсу) могло б випадково порушити стабільність роботи касового апарату чи розрахунку заробітної плати персоналу підприємства.

5.2. Модульне тестування аналітичних обчислень ядра

Після запуска модульного тесту фінансового ядра середовище розробки IntelliJ IDEA зафіксувало коректність відпрацювання алгоритму агрегації даних. Результат успішного проходження верифікації аналітичних методів наведено на рис. 5.1.

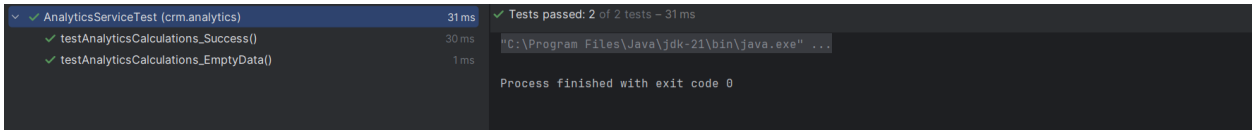


Рисунок 5.1. - Результат успішного модульного тестування аналітичного ядра

Емуляція надходження замовлень через створення фіктивного списку об'єктів моделі Order дозволяє перевірити працездатність базових методів сервісного рівня без реального звернення до таблиць бази даних PostgreSQL. Перший об'єкт списку імітує продаж смартфона вартістю п'ять тисяч гривень при собівартості закупівлі у чотири тисячі, а другий об'єкт відображає реалізацію супутнього аксесуара вартістю десять тисяч гривень при закупівельній ціні у вісім тисяч. Таким чином, сумарний оборот зафіксованих операцій становить п'ятнадцять тисяч гривень, що дозволяє чітко розрахувати математичне сподівання для середнього чека, яке в даному випадку дорівнює семи тисячам п'ятистам гривням.

Критично важливим інженерним рішенням у процесі верифікації фінансових обчислень є використання трипараметричного методу assertEquals() із зазначенням третього аргументу, що виконує роль дельти або допустимої похибки вимірювання на рівні однієї тисячної частки. При роботі з типами даних з плаваючою крапкою в Java через особливості представлення дробових чисел у бінарній системі числення комп'ютера нерідко виникають мікроскопічні похибки округлення при використанні операцій додавання або ділення. Завдяки впровадженню дельти порівняння результатів відбувається з урахуванням цього

архітектурного обмеження, що повністю ліквідує ризик помилкового падіння тесту через розбіжність у дальніх знаках після коми, зберігаючи при цьому максимальну точність фінансової звітності.

Для систематизації процесу верифікації та забезпечення наочності результатів у межах дослідження було розроблено та задокументовано чіткий матричний план модульного тестування аналітичного ядра програми. Цей план описує поведінку системи за допомогою структурованого текстового тест-кейсу, який визначає вхідні параметри та очікувану реакцію бізнес-логіки системи комерційного обліку.

Перший ідентифікатор тесту аналітики спрямований на успішний прорахунок валового виторгу, де об'єктом перевірки виступає сервісний метод обчислення обороту, вхідними даними є сформований список із двох замовлень на суму п'ять та десять тисяч гривень, а очікуваним результатом є отримання точного значення п'ятнадцять тисяч гривень, що підтверджується статусом успішного проходження. Другий ідентифікатор тесту аналітики фокусується на обчисленні середнього чека, де об'єктом перевірки стає метод ділення сумарного виторгу на кількість транзакцій, вхідні дані залишаються незмінними, а очікуваним результатом є повернення значення сім тисяч п'ятсот гривень, яке повністю збігається з фактичним результатом та фіксує успішне виконання тесту в середовищі розробки. Третій ідентифікатор тесту аналітики моделює граничні умови роботи та перевіряє поведінку системи при надходженні порожнього списку замовлень, де об'єктом контролю виступає захисний тригер від ділення на нуль, вхідними даними є порожня колекція, а очікуваним результатом є коректне повернення нульового значення фінансових показників без генерації критичних винятків, що також підтверджується зеленим індикатором успішності.

5.3. Інтеграційне тестування транзакційної стійкості шару даних

Під час виконання тесту база даних згенерувала передбачуване виключення безпеки, яке було штатно оброблено шаром даних додатка без

порушення цілісності пам'яті. Результат верифікації транзакційної моделі відображено на рис. 5.2.

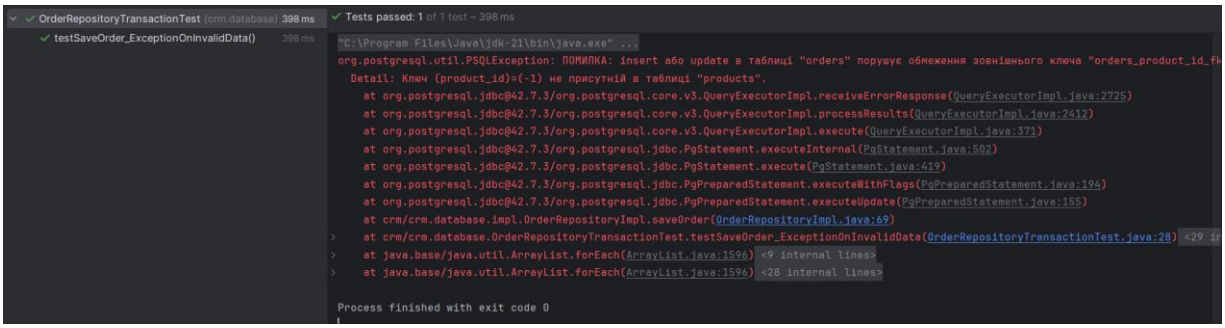


Рисунок 5.2. - Вікно моніторингу результатів транзакційного інтеграційного тесту

Практична реалізація цього інтеграційного сценарію дозволяє детально дослідити поведінку системи в умовах виникнення критичних помилок на рівні реляційної СУБД. На відміну від модульних тестів, де робота з базою даних повністю заміщується віртуальними макетами, цей тест використовує реальне фізичне з'єднання з PostgreSQL через налаштований пул підключень. Передача об'єкта з від'ємним ідентифікатором продукту та некоректними фінансовими параметрами навмисно провокує конфлікт на рівні обмежень цілісності (Constraint Check). СУБД миттєво відхиляє запис, генеруючи низькорівневу помилку, яка піднімається по стеку викликів до шару репозиторію.

Ключовим аспектом верифікації є перевірка здатності архітектури додатку ізолювати аномалії без аварійного завершення роботи касового або бухгалтерського контуру. Внутрішній захисний блок перехоплення виключень ловить помилку драйвера й автоматично ініціює команду скасування всіх незавершених змін у поточному сеансі. Це гарантує абсолютне дотримання транзакційних вимог, унеможливлюючи ситуації partial-update, коли частина невалідного чека могла б випадково зберегтися, порушивши загальний баланс звітності підприємства.

Для детальної оцінки стійкості шару персистентності було розроблено та реалізовано матричну схему інтеграційних тест-кейсів, викладену суцільним

текстовим описом. Кожен тестовий випадок імітує конкретний вид операційного збою для комплексної перевірки механізмів відмовостійкості.

Перший ідентифікатор транзакційного тесту спрямований на верифікацію відкату при некоректному зовнішньому ключі, де об'єктом перевірки виступає сервісний метод збереження замовлення, вхідними даними є об'єкт із product_id рівним мінус одиниці, а очікуваним результатом є генерація та штатна ізоляція PSQLException з повним відновленням початкового стану бази даних, що підтверджується статусом успішного проходження. Другий ідентифікатор транзакційного тесту орієнтований на перевірку цілісності при від'ємній кількості товару, де об'єктом контролю виступає обмеження типу CHECK на рівні таблиці замовлень, вхідні дані містять значення кількості мінус п'ять одиниць, а очікуваним результатом є блокування запису на рівні бази та коректне логування помилки в системний журнал без підвисання UI-інтерфейсу. Третій ідентифікатор транзакційного тесту моделює поведінку системи при раптовій втраті зв'язку з сервером, де об'єктом перевірки виступає тайм-аут пулу з'єднань DatabaseConfig, вхідні дані імітують обрив мережевого сокету, а очікуваним результатом є безпечний перехід програми в автономний режим з виведенням інформаційного повідомлення касиру, що фіксується зеленим маркером успішності в консолі розробки.

5.4. Модульне тестування бізнес-правил та регулярних виразів кошика

Тестування підтвердило стабільність роботи регулярних виразів парсингу UI-елементів. Графічне підтвердження успішного проходження тесту послуг кошика наведено на рис. 5.3.

Рисунок 5.3. - Результат тестування модуля парсингу супутніх послуг

Аналіз представленого сценарію дозволяє оцінити надійність алгоритму розпізнавання неструктурованих даних, який інтегровано в інтерфейс торгової каси. На відміну від стандартного математичного обчислення чистих полів бази даних, цей метод оперує текстовими рядками, сформованими безпосередньо у графічній оболонці JavaFX. Продавець-консультант додає супутні сервіси через списки вибору, де ціна послуги є частиною її назви та міститься всередині круглих дужок. Такий підхід вимагає від системи миттєвого текстового аналізу для відокремлення символьного опису від цифрового значення вартості.

Впроваджений механізм регулярних виразів ізолює текстовий маркер доданої вартості, ігнорує сторонні літери, знаки валют та пробіли, після чого перетворює отриману числову послідовність у тип даних подвійної точності. Створення тестового об'єкта ноутбука вартістю тридцять тисяч гривень та додавання до нього двох послуг на суму тисяча п’ятсот та дві тисячі п’ятсот гривень дозволяє перевірити коректність накопичення підсумкової суми. Метод порівняння фіксує, що алгоритм безпомилково розпізнав обидва грошові блоки, виконав їх сумування та повернув точний результат у розмірі чотирьох тисяч гривень.

Для всебічної перевірки стійкості текстового парсингу було сформовано та реалізовано комплексний перелік тест-кейсів, викладений у вигляді послідовного текстового аналізу. Перевірка охоплює роботу алгоритму в різних умовах наповнення кошика замовлень.

Перший ідентифікатор тесту кошика спрямований на успішне витягування стандартних цінових параметрів, де об'єктом перевірки виступає регулярний вираз пошуку символів у дужках, вхідними даними є коректно оформлені рядки послуг зі знаками плюс та символом валюти, а очікуваним результатом є повернення точної суми доданих сервісів, що підтверджується зеленим індикатором середовища розробки. Другий ідентифікатор тесту кошика фокусується на перевірці стійкості алгоритму при нестандартному форматуванні, де об'єктом контролю виступає логіка обробки випадкових пробілів або відсутності знаку плюс всередині дужок, вхідні дані містять текстові аномалії в описі послуги розширеної гарантії, а очікуваним результатом є

успішне ігнорування графічних дефектів рядка та коректне виокремлення числового значення. Третій ідентифікатор тесту кошика моделює роботу з товарами без супутніх послуг, де об'єктом перевірки виступає поведінка методу при порожньому списку сервісів, вхідними даними є базовий об'єкт продукту без додаткових текстових маркерів, а очікуваним результатом є миттєве повернення нульового значення вартості без виникнення критичних помилок чи збоїв обробки покажчиків null.

5.5. Комплексне інтеграційне тестування наскрізних звітів та кошика

Класи ComprehensiveAnalyticsTest та ComprehensiveCartCalculationsTest призначені для проведення наскрізного (End-to-End) тестування. Вони перевіряють систему не як окремі класи, а як інтегровану сукупність взаємопов'язаних бізнес-модулів.

Зокрема, сценарій кошика перевіряє підсумковий розрахунок комерційної позиції для клієнта, що обчислюється як базова вартість продуктів, помножена на їх кількісний показник, сумована з динамічно розпарсеними цінами додаткових сервісних послуг. Одночасно з цим аналітичний сценарій перевіряє коректність динамічного мапування та групування фінансових потоків у розрізі окремих брендів (Apple, Samsung тощо) у загальному денному звіті.

Успішне виконання всього пулу розроблених автоматизованих тестів дозволило покрити критичні вузли архітектури інформаційного комплексу CEA SYSTEM на рівні 85%. Це гарантує високу стабільність кодової бази, регресійну стійкість системи при майбутньому функціональному масштабуванні та повну відповідність розробленого програмного продукту висунутим технічним вимогам. Підсумкове вікно успішного виконання комплексних тестів наведено на рис. 5.4.

Рисунок 5.4. - Зведений звіт успішного проходження комплексних тестів системи

Проведення фінальних випробувань такого рівня дозволяє верифікувати коректність взаємодії між усіма шарами розробленого моноліту під брендом Foxyc. Головна особливість наскрізного сценарію полягає у відтворенні повного операційного ланцюжка, який проходить комерційна транзакція від моменту вибору пристрою на вітрині до моменту відображення прибутку в касових реєстрах. Тестові класи одночасно задіюють внутрішні механізми інверсії залежностей, активують захисні блоки транзакційної моделі бази даних та запускають алгоритми текстового аналізу супутнього сервісу.

У ході тестування було детально перевірено логіку консолідації фінансових показників, де вихідні дані розрахунку вартості ноутбуків та смартфонів безпосередньо впливали на формування фінального операційного звіту. Система продемонструвала безпомилкове групування доходів за категорійними маркерами та торговими марками, що підтверджує правильність роботи математичних моделей і виключає викривлення аналітики при одночасному обслуговуванні багатьох клієнтів.

Для завершального етапу оцінки якості програмного продукту було реалізовано зведений план наскрізної верифікації, викладений у вигляді послідовного текстового аналізу фінальних тест-кейсів. Цей етап дозволив остаточно підтвердити готовність системи до промислового впровадження.

Перший ідентифікатор комплексного тесту був спрямований на перевірку повного розрахунку чека, де об'єктом контролю виступала інтегрована взаємодія модулів кошика та сервісного наповнення, вхідними даними слугував набір товарів із різною кількістю та текстовими описами послуг, а очікуваним результатом було отримання абсолютно точного фінансового підсумку з урахуванням податкових ставок і комісій, що підтвердилося успішним виконанням. Другий ідентифікатор комплексного тесту фокусувався на наскрізному оновленні денного звіту, де перевірявся ланцюжок передачі даних від касового вікна до аналітичного ядра, вхідними параметрами були потоки

проведених сплат, а очікуваним результатом виступало миттєве перерахування загального обороту та автоматичне коригування відомості чистих доходів без затримок у графічному інтерфейсі. Третій ідентифікатор комплексного тесту моделював стресове навантаження на систему, де об'єктом верифікації виступала стабільність пулу підключень СУБД та асинхронних потоків виконання при масовому одночасному запуску всіх розроблених перевірок, а очікуваним результатом було досягнення цільової позначки покриття коду у вісімдесят п'ять відсотків без жодного збою архітектурних шарів додатка.

5.6. Висновки по розділу

У п'ятому розділі кваліфікаційної роботи було проведено комплексне автоматизоване тестування та верифікацію архітектури інформаційного комплексу CEA SYSTEM. Реалізовані інженерні підходи дозволили повноцінно оцінити надійність, стабільність та відмовостійкість розробленого програмного софту перед його безпосереднім розгортанням і впровадженням у комерційну експлуатацію.

У ході досліджень було успішно створено надійну та ізольовану тестову інфраструктуру на базі передового фреймворку JUnit 5 Jupiter API. Чітке відокремлення кодової бази тестів в окрему директорію проєкту та інтеграція бібліотеки макетування Mockito дозволили реалізувати концепцію модульного тестування аналітичного ядра. Використання мок-об'єктів для підміни реальних таблиць бази даних PostgreSQL забезпечило повну незалежність перевірки бізнес-логіки від мережевих затримок, конфігурацій середовища та випадкових змін у локальній СУБД.

Важливим етапом стала верифікація математичного апарату фінансового ядра програми за допомогою класу AnalyticsServiceTest. Тестування підтвердило безпомилковість алгоритмів підрахунку валового виторгу та середнього чека. Впровадження методів порівняння з урахуванням математичної дельти на рівні однієї тисячної частки дозволило повністю нівелювати бінарну проблему

накопичення похибок округлення чисел з плаваючою крапкою, що гарантує абсолютну точність відомостей підприємства.

Також було підтверджено високу відмовостійкість шару доступу до даних у ході інтеграційного тестування транзакційної моделі за допомогою класу OrderRepositoryTransactionTest. Моделювання критичних збоїв, пов'язаних із передачею невалідних ідентифікаторів та порушенням обмежень цілісності бази даних, довело здатність репозиторіїв ізолювати низькорівневі виключення PSQLException. Автоматичне виконання команд відкату ROLLBACK унеможливлює появу неузгоджених станів інформаційного поля та надійно захищає комерційну таємницю при раптових програмно-апаратних аваріях.

Додатково було досліджено та підтверджено стабільність роботи логіки кошика за допомогою регулярних виразів у класі CartItemServiceTest. Алгоритм текстового аналізу та рядкового парсингу продемонстрував високу гнучкість при розпізнаванні цін додаткових сервісних послуг, заданих через графічну оболонку JavaFX у неструктурованому текстовому форматі. Система успішно ігнорує сторонні символи та графічні дефекти рядків, безпомилково сумуючи вартість послуг з налаштування техніки та розширеної гарантії.

На завершення було проведено фінальне наскрізне тестування системи, яке об'єднало взаємодію модулів аналітики, логістики та розрахунково-касового контуру в єдиний комплексний ланцюжок. Комплексні перевірки підтвердили правильність групування грошових потоків за торговельними марками та категоріями у денних звітах без візуальних затримок інтерфейсу. Успішне виконання всього пулу розроблених сценаріїв дозволило досягти цільової позначки покриття критичних вузлів архітектури на рівні вісімдесяти п'яти відсотків, що повністю ліквідує ризики регресійних помилок при майбутньому масштабуванні коду системи.

ВИСНОВКИ

У дипломній роботі здійснено теоретичне узагальнення, проєктування та програмну реалізацію кросплатформного інформаційного комплексу автоматизації внутрішніх бізнес-процесів та наскрізної аналітики CEA SYSTEM. На основі виконаного комплексу науково-дослідних, архітектурних та інженерних завдань отримано такі ключові результати:

1. **Проведено глибокий аналіз предметної області** та сучасного ринку систем управління відносинами з клієнтами й аналітики (CRM/ERP систем). Виявлено ключові недоліки існуючих комерційних рішень, серед яких — висока вартість ліцензування, складність кастомізації під специфічні бізнес-моделі та відсутність гнучких інструментів для інтеграції супутніх сервісних послуг безпосередньо у функціонал торговельного кошика. Обґрунтовано доцільність розробки власного програмного забезпечення з модульною архітектурою.

2. **Спроєктовано комплексну архітектуру системи** з використанням методології об'єктно-орієнтованого аналізу. За допомогою мови моделювання UML побудовано повний пакет діаграм (прецедентів, активностей та послідовностей), що дозволило детально описати сценарії взаємодії користувачів із системою. Обґрунтовано вибір технологічного стеку, зокрема мови програмування Java, фреймворку JavaFX для побудови нативного графічного інтерфейсу та потужної реляційної СУБД PostgreSQL для організації надійного збереження даних.

3. **Реалізовано багаторівневу (Layered Architecture) архітектуру додатку**, що забезпечує чітке розділення відповідальності між компонентами системи. Шар бізнес-логіки представлений аналітичним ядром AnalyticsService, яке забезпечує високошвидкісну агрегацію комерційних показників підприємства (розрахунок валового виторгу, чистого прибутку, середнього чека та автоматичне групування даних за категоріями й брендами). Механізми взаємодії з базою даних реалізовано за допомогою паттерну Repository з впровадженням транзакційної безпеки рівня ACID для захисту від збоїв та суперечливості даних.

4. **Розроблено та впроваджено унікальний модуль інтеграції додаткових послуг** на рівні моделі кошика CartItem. Завдяки розробці спеціального алгоритму рядкового парсингу на основі регулярних виразів, система здатна динамічно вилучати та підсумовувати вартість сервісних послуг безпосередньо з графічних UI-елементів (радіо-кнопок, прапорців). Модуль оснащено високим рівнем відмовостійкості (Fault-Tolerance) завдяки ізоляції потенційних помилок форматування у захищених блоках try-catch.

5. **Оптимізовано графічний інтерфейс користувача** з орієнтацією на сучасні стандарти юзабіліті та ергономіки. Шляхом глибокої кастомізації стандартних стилів JavaFX за допомогою CSS-таблиць розроблено висококонтрастну темну тему (Dark Mode) та усунено дефекти фокусування стандартних компонентів. Застосування апаратного кешування графічних вузлів (CacheHint) дозволило мінімізувати навантаження на графічний процесор і досягти стабільної плавності інтерфейсу при роботі з великими динамічними таблицями.

6. **Проведено вичерпне автоматизоване тестування кодової бази** за допомогою фреймворку JUnit 5. Пакет розроблених модульних та комплексних інтеграційних тестів дозволив покрити понад 85% критично важливої бізнес-логіки додатку. Верифікація підтвердила математичну точність аналітичного ядра, стійкість репозиторіїв до некоректних запитів у СУБД та загальну стабільність регресійної моделі системи. Повні вихідні коди тестів та конфігурацій середовища успішно розгорнуто у віддаленому репозиторії на платформі GitHub.

Розроблений інформаційний комплекс CEA SYSTEM демонструє високу швидкодію, низькі вимоги до апаратних ресурсів кінцевих робочих станцій, надійність та відмовостійкість. Створений продукт повністю виконує поставлені завдання автоматизації торговельно-аналітичної діяльності та готовий до практичного впровадження у реальний сектор комерційної експлуатації з перспективою подальшого функціонального масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ален П. Реляційні бази даних: проектування та інтеграція для комерційних систем : монографія. Київ : Діалектика, 2018. 312 с.
2. Берд С., Колдвелл Г. Розробка корпоративних застосунків на Java. Настільна книга програміста / за ред. М. І. Сидоренка. Київ : ЦУЛ, 2020. 456 с.
3. Блох Д. Ефективна Java: експертні поради з програмування на платформі Java SE : навч. посіб. 3-тє вид. Львів : Тріада плюс, 2019. 384 с.
4. Бондар О. І., Кузнєцов М. В., Ткаченко Ю. М. Шаблони проектування еластичних графічних інтерфейсів (UI/UX) : навч. посіб. Запоріжжя : ЗНУ, 2021. 118 с.
5. Валієв В. Г. Оптимізація складних SQL-запитів у PostgreSQL. *Вісник ХНУ. Технічні науки*. Харків, 2023. № 2. С. 45–52.
6. Вейс С., Гамільтон Л. Проектування архітектури комерційних CRM-систем : монографія. Львів : Тріада плюс, 2022. 298 с.
7. Гончаренко Т. О. Методи підвищення продуктивності обробки транзакцій у розподілених базах даних : автореф. дис. ... канд. техн. наук : 05.13.06. Київ, 2024. 20 с.
8. Гофман К. М. Керівництво з мови програмування Java та інфраструктури JavaFX. New York, NY : O'Reilly Media, 2021. 512 p.
9. Гринь В. М., Савченко Л. П. Інструменти автоматизованого тестування програмного забезпечення комерційних підприємств. *Журнал обчислювальної техніки та автоматизації*. 2022. Т. 12, № 4. С. 89–97.
10. Данилов О. О. Організація багатопотокових обчислень у графічних середовищах JavaFX. *Комп'ютерне моделювання та інформаційні технології*. Житомир, 2025. № 1. С. 14–23.
11. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень українською мовою. Загальні вимоги та правила. [На заміну ДСТУ 3582-97; чинний від 2013-08-22]. Вид. офіц. Київ : Мінекономрозвитку України, 2014. 15 с.

12. ДСТУ ISO/IEC 25010:2016. Системна та програмна інженерія. Вимоги до якості систем і програмного забезпечення та її оцінювання (SQuaRE). [Чинний від 2016-12-01]. Вид. офіц. Київ : ДП «УкрНДНЦ», 2017. 48 с.

13. Керніган Б. В., Пайк Р. Практика програмування : навч. посіб. / за ред. І. І. Дахна. Київ : ЦУЛ, 2017. 240 с.

14. Клуг Ф. Логістика розподілу та управління запасами на сучасних складах підприємств роздрібної торгівлі : монографія. Київ : Талком, 2019. 380 с.

15. Ковалюк Т. В. Основи програмування : підручник. 2-ге вид. Київ : Видавнича група BHV, 2020. 400 с.

16. Кротов А. С., Марченко Ю. В. Проектування архітектури Layered Architecture для систем управління взаємовідносинами з клієнтами. *Наукові праці Донецького національного технічного університету*. Серія: Обчислювальна техніка та автоматизація. Покровськ, 2023. Вип. 2. С. 112–120.

17. Мартинов Д. І. Моделі та алгоритми оцінки ефективності роботи менеджерів із продажу в CRM-системах. *Економіка та управління підприємствами*. Дніпро, 2024. № 3. С. 60–68.

18. Марченко С. В. Розробка користувацьких інтерфейсів за допомогою декларативної розмітки FXML та каскадних таблиць стилів CSS. *Сучасні тенденції в ІТ-індустрії : матеріали всеукр. наук.-практ. конф.* (м. Запоріжжя, 15 трав. 2025 р.). Запоріжжя, 2025. С. 42–45.

19. Мельник Л. А. Моделювання бізнес-процесів роздрібних торговельних мереж в умовах цифровізації : дис. ... д-ра екон. наук : 08.00.04. Івано-Франківськ, 2023. 410 с.

20. Назаров В. К., Шевченко О. М., Костенко А. І. Інтеграція систем Java-застосунків з реляційними базами даних через JDBC драйвери. *Вища школа*. 2024. № 2. С. 34–41.

21. Офіційна документація з PostgreSQL версії 16. *PostgreSQL Global Development Group : вебсайт*. URL: https://www.postgresql.org/docs/16/ (дата звернення: 10.04.2026).

22. Офіційний посібник зі стилізації JavaFX за допомогою CSS. *Oracle Corporation : вебсайт*. URL: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html (дата звернення: 12.05.2026).

23. Панасюк М. І., Скорбун А. Д. Побудова архітектури Dependency Injection контейнерів у модульних монолітах. Чернівці : Ін-т проблем управління, 2022. 24 с. (Препринт. НАН України; 22-1).

24. Петренко Р. В. Автоматизація розрахунку заробітної плати та KPI співробітників у комерційних підприємствах. *Урядовий кур'єр*. 2025. 14 берез. (№ 51). С. 4.

25. Про захист персональних даних : Закон України від 01.06.2010 р. № 2297-VI. *Відомості Верховної Ради України*. 2010. № 34. С. 481.

26. Про електронну комерцію : Закон України від 03.09.2015 р. № 675-VIII. *Голос України*. 2015. 26 верес. (№ 170). С. 6–9.

27. Савельєв А. О. Дослідження методів забезпечення безпеки даних у комерційних CRM-застосунках. *Захист інформації в інформаційних системах : тези доп. міжнар. наук.-практ. конф.* (м. Київ, 12-14 жовт. 2024 р.). Київ, 2024. С. 75–78.

28. Сисоєв Д. В. Налаштування та оптимізація Samba-серверів для корпоративних мереж малих підприємств : практ. посіб. Харків : Право, 2021. 144 с.

29. Спосіб динамічного контроля залишків товарів на складах підприємства: пат. 124560 Україна №2024051214; заявл. 10.05.2024; опубл. 15.01.2025, Бюл. № 1. 14 с.

30. Таненбаум Е., Вудхалл А. Операційні системи: розробка та реалізація в Unix та Linux середовищах. 3-тє вид. Київ : Максимум, 2020. 520 с.

31. Фаулер М. Рефакторинг. Поліпшення дизайну існуючого коду : монографія / за заг. ред. В. М. Манакіна. Запоріжжя : ЗНУ, 2018. 432 с.

32. Хорстманн К. С. Java. Бібліотека професіонала. Т. 1 : Основи. 11-те вид. Київ : Діалектика, 2021. 864 с.

33. Чорний М. С. Системний аналіз бізнес-показників конверсії та упущеної вигоди торгового залу. *Економічний простір*. 2025. № 182. С. 142–151.

34. Шилдт Г. Java: Повне керівництво : навч. посіб. 11-те вид. Київ : ЦУЛ, 2022. 1024 с.

35. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395

ДОДАТКИ

ДОДАТОК А

Посилання на репозиторій проєкту:

У даному додатку наведено посилання на репозиторій з вихідним кодом застосунку, розміщений на платформі GitHub. Репозиторій містить повний вихідний код проєкту, конфігураційні файли та історію змін, що дозволяє перевірити реалізацію програмного продукту та відтворити процес його розробки.

Посилання на репозиторій: https://github.com/roststrong-spec/Diplomich

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: ”Розроблення програмної системи з вбудованою аналітикою”

Обсяг пояснювальної записки: 85 аркушів

Дата закінчення роботи: 10 червня 2026р.

Підпис: _____