МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **"ВИКОРИСТАННЯ ПЛАТФОРМ ІЗ ЛОУ КОДОМ ДЛЯ МОДЕЛЮВАННЯ ФІНАНСОВИХ ДАНИХ: ПРИКЛАД З MENDIX"**

Виконав: студент 2 курсу, групи 8.1224-з
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
М.М. Романішин
(ініціали та прізвище)
Керівник старший викладач кафедри комп'ютерних наук, PhD, Лебедєва-Дичко А.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" ____ " _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Романішину Микиті Миколайовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Використання платформ із лоу кодом для моделювання фінансових даних: приклад із Mendix

керівник роботи Анастасія Сергіївна Лебедєва-Дичко, PhD
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від " 28 " лютого 2025 року № 327-с

2. Строк подання студентом роботи 25.11.2025

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

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

1. Постановка задачі.
2. Основні теоретичні відомості.
3. Аналіз предметної області та вимог до low-code систем моделювання даних.
4. Проєктування архітектури системи та реалізація програмного забезпечення.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 03.03.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	03.03.2025	
2.	Збір вихідних даних.	07.04.2025	
3.	Обробка методичних та теоретичних	03.06.2025	
	джерел.		
4.	Розробка першого та другого розділу.	02.09.2025	
5.	Розробка третього розділу.	06.10.2025	
6.	Оформлення та нормоконтроль	10.11.2025	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	10.12.2025	

Студент _____ М.М. Романішин
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.С. Лебедєва-Дичко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра “Використання платформ із лоу кодом для моделювання фінансових даних: приклад із Mendix”: 70 с., 6 рис., 4 табл., 49 джерел.

АУДИТНІСТЬ, БІЗНЕС-ЛОГІКА, ДОМЕННА МОДЕЛЬ, ІНТЕГРАЦІЇ, ЛОУ-КОД, МОДЕЛЮВАННЯ ФІНАНСОВИХ ДАНИХ, СЦЕНАРНИЙ АНАЛІЗ, ТРАСОВАНІСТЬ, MENDIX.

Об’єкт дослідження – процес моделювання фінансових даних у діяльності фінансових інституцій розвитку (DFIs).

Предмет дослідження – методи та засоби побудови low-code системи моделювання фінансових даних на платформі Mendix.

Мета роботи – розробити концепцію та архітектуру low-code системи, що підтримує формування грошових потоків, сценарний аналіз, контрольовану бізнес-логіку та інтегроване управління даними.

Методологія охоплює системний аналіз, аналіз вимог, об’єктно-орієнтоване моделювання та проєктування доменної моделі. Наукова новизна – обґрунтування застосування low-code для моделювання фінансових даних у DFIs з акцентом на аудитність (audit trail), трасованість (traceability), відтворюваність розрахунків, управління ризиками та сценарне моделювання. Отримані результати включають визначення функціональних і нефункціональних вимог (генерація cash flow, сценарії, версіонування, інтеграції, прозорість змін), розробку архітектури та прототипу на Mendix, що реалізує доменну модель, ключові мікрофлоу й контроль розрахунків. Підхід узгоджується з практиками DFIs і вимогами комплаєнсу та може стати основою промислового рішення для автоматизації грошових потоків, stress-testing і регуляторної звітності, знижуючи фінансові й репутаційні ризики та демонструючи ефективність low-code для складних моделей.

SUMMARY

Master's Qualifying Theses: "Leveraging Low-Code Platforms for Financial Data Modeling: a Case Study with Mendix": 70 pages, 6 figures, 4 tables, 49 references.

AUDITABILITY, BUSINESS LOGIC, DOMAIN MODEL, INTEGRATIONS, LOW-CODE, FINANCIAL DATA MODELING, SCENARIO ANALYSIS, TRACEABILITY, MENDIX.

Object of research – the process of financial data modeling within the operational activities of development finance institutions (DFIs).

Subject of research – methods and tools for building a low-code financial data modeling system on the Mendix platform.

Purpose of the thesis – to develop the concept and architecture of a low-code system that supports cash-flow generation, scenario analysis, governed business logic, and integrated data management.

The methodology includes systems analysis, requirements analysis, object-oriented modeling, and domain model design. The scientific novelty lies in substantiating the use of low-code for financial data modeling in DFIs, with an emphasis on auditability (audit trail), traceability, reproducibility of calculations, risk management, and scenario modeling – areas that have been insufficiently explored in the low-code context. The results include defining functional and non-functional requirements (cash-flow generation, scenarios, versioning, integrations, and change transparency), developing the architecture, and creating a Mendix-based prototype that implements the domain model, key microflows, and calculation control mechanisms. The proposed approach aligns with DFI practices and compliance requirements and can serve as a foundation for an industrial-grade solution to automate cash-flow modeling, stress testing, and regulatory reporting, reducing financial and reputational risks while demonstrating the effectiveness of low-code for complex models.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ 8
1 Теоретичні основи використання low-code платформ для моделювання фінансових даних 10
1.1 Платформи з лоу-кодом: визначення, еволюція, класифікація 10
1.1.1 Історичні передумови та становлення лоу-код підходу 12
1.1.2 Переваги та виклики використання платформ з лоу-кодом 13
1.2 Принципи проєктування систем на основі low-code. Технічні аспекти проєктування low-code систем 15
1.3 Платформа Mendix як приклад low-code рішення 16
1.4 Особливості моделювання фінансових даних у банках та DFIs 18
1.4.1 Застосування моделювання у DFIs 18
1.4.2 Виклики та обмеження фінансового моделювання 19
1.5 Порівняння low-code та традиційної розробки у фінансовому секторі. 21
2 Аналіз предметної області та вимог до low-code системи моделювання фінансових даних 22
2.1 Особливості процесів моделювання фінансових даних у DFIs 22
2.1.1 Роль фінансової моделі в інвестиційному циклі проєкту 23
2.1.2 Ключові процеси, що ґрунтуються на фінансовому моделюванні 24
2.1.3 Основні проблемні точки поточного підходу до моделювання 25
2.2 Поточні технологічні виклики та потреби у цифровізації процесів моделювання 27
2.2.1 Основні проблемні точки поточного підходу до моделювання 29
2.2.2 Основні проблемні точки поточного підходу до моделювання 30

2.3 Функціональні вимоги до low-code системи моделювання фінансових даних 30
2.3.1 Формалізація введення припущень і параметрів фінансової моделі .. 31
2.3.2 Автоматизована генерація грошових потоків та фінансових показників 32
2.3.3 Підтримка сценарного аналізу та моделювання ризиків 33
2.3.4 Версіонування моделей і контроль доступу 33
2.4 Нефункціональні вимоги та вимоги до безпеки й комплаєнсу 34
2.4.1 Продуктивність та обробка великих фінансових масивів 34
2.4.2 Масштабованість і підтримка зростання портфеля 35
2.4.3 Досвід користувача та підтримка бізнес-процесів 36
2.5 Обґрунтування вибору Mendix як платформи для реалізації цільової системи 37
2.5.1 Відповідність функціональним вимогам системи 37
2.5.2 Відповідність нефункціональним вимогам та вимогам до безпеки ... 38
3 Проєктування low-code системи моделювання фінансових даних 39
3.1 Загальна архітектура системи 39
3.1.1 Логічна схема архітектури 45
3.1.2 Ролі основних компонентів 49
3.2 Доменна модель системи 52
3.2.1 Основні сутності доменної моделі 52
3.2.2 Узагальнена діаграма доменної моделі 55
3.2.3 Особливості реалізації доменної моделі в Mendix 59
3.3 Реалізація логіки моделювання фінансових даних у Mendix 59
3.3.1 Обробка припущень та параметрів моделі 61
3.3.2 Генерація грошового потоку (Cash Flow Engine) 61
3.3.3 Реалізація сценарного аналізу 62
3.3.4 Версіонування та аудит змін 63
Висновки 65
Перелік посилань 67

ВСТУП

Сучасні фінансові інституції розвитку функціонують у складному та динамічному середовищі, у якому якість, прозорість і відтворюваність фінансових моделей безпосередньо впливають на прийняття інвестиційних рішень, оцінку ризиків та ефективність управління портфелем. Зростання обсягів даних, ускладнення регуляторних вимог, необхідність оперативного проведення сценарного аналізу та забезпечення аудиту всіх етапів моделювання формують потребу у технологічних рішеннях нового покоління. Традиційні інструменти, зокрема табличні процесори, залишаються фундаментальними для фінансового аналізу, проте їхні обмеження у сфері узгодженості даних, спільної роботи, контролю доступів, версіонування та комплаєнсу стимулюють пошук альтернативних підходів.

У цьому контексті технології лоу-код набули особливої актуальності. Вони пропонують можливість швидкої розробки бізнес-орієнтованих систем, забезпечуючи гнучкість, масштабованість і прозорість обробки даних без необхідності повної залежності від традиційних циклів програмної розробки. Платформа Mendix, як один із провідних представників лоу-код рішень, поєднує модельне проєктування, вбудовані механізми підтримки бізнес-логіки, розширені можливості інтеграції та засоби забезпечення аудиту. Це робить її перспективною основою для створення сучасної системи моделювання фінансових даних у фінансових інституціях розвитку.

Актуальність обраної теми зумовлена необхідністю переходу від фрагментованих, ручних та малокерованих процесів фінансового моделювання до стандартизованих рішень із чіткими правилами, централізованим управлінням даними та відтворюваними розрахунковими процедурами. Особливої ваги це набуває у діяльності таких інституцій, як Нідерландський банк розвитку (FMO), де інвестиційні рішення базуються на комплексних

фінансових оцінках, що повинні відповідати високим вимогам до точності, комплаєнсу, документальності та прозорості.

Метою кваліфікаційної роботи є формування методичних і технологічних засад проєктування лоу-код системи моделювання фінансових даних на платформі Mendix, а також розроблення архітектури та прототипу рішення, здатного підтримувати повний цикл обробки фінансових показників: від введення припущень і генерації грошових потоків до сценарного аналізу, збереження історії змін та забезпечення аудиту відповідно до внутрішніх і зовнішніх стандартів.

Для досягнення поставленої мети необхідно:

– проаналізувати предметну область моделювання фінансових даних у міжнарожних банках розвитку (DFIs);
– визначити ключові технологічні та організаційні виклики; сформувати функціональні та нефункціональні вимоги до цільової системи;
– обґрунтувати вибір лоу-код платформи; розробити архітектуру рішення та оцінити його здатність підвищувати керованість і ефективність фінансових моделей.

Структура роботи включає теоретичний огляд лоу-код підходів і платформи Mendix, аналіз вимог до системи моделювання фінансових даних, проєктування компонентів цільової системи та узагальнення результатів дослідження.

1 ТЕОРЕТИЧНІ ОСНОВИ ВИКОРИСТАННЯ LOW-CODE ПЛАТФОРМ ДЛЯ МОДЕЛЮВАННЯ ФІНАНСОВИХ ДАНИХ

1.1 Платформи з лоу-кодом: визначення, еволюція, класифікація

Платформи з лоу-кодом (low-code platforms) – це інструменти розробки програмного забезпечення, призначені для створення застосунків з мінімальним обсягом ручного кодування. Вони орієнтовані як на професійних розробників, так і на так званих citizen developers – користувачів без глибокої технічної підготовки, які можуть будувати рішення на основі візуальних моделей та готових компонентів [1]. Поява та швидкий розвиток таких платформ зумовлені потребою у прискореній доставці цифрових продуктів у умовах дефіциту кваліфікованих розробників і постійного зростання вимог до цифрової трансформації бізнесу [2], [7].

Термін low-code був уперше запропонований аналітичною компанією Forrester Research у 2011 році, що фактично означало формальне визнання нового підходу до спрощення процесів розробки застосунків [1]. Відтоді платформи з лоу-кодом стали важливим елементом сучасних практик розробки, сприяючи демократизації створення ПЗ та ширшому залученню бізнес-користувачів до проєктування й еволюції інформаційних систем. Для подальшого аналізу доцільно розрізняти типи low-code рішень за цільовою аудиторією та призначенням, оскільки різні класи платформ відрізняються рівнем “enterprise”-можливостей, інструментами керування життєвим циклом і фокусом (розробка застосунків, автоматизація процесів, інтеграції тощо). Узагальнену класифікацію low-code/no-code платформ наведено на таблиці. 1.1. [4], [9].

Таблиця 1.1 – Узагальнена класифікація low-code/no-code платформ

Клас платформи	**Призначення**	**Основні користувачі**	**Типові можливості / акценти**	**Приклади / примітки**
1) No-code	Швидке створення простих застосунків і автоматизацій без програмування	Citizen developers, бізнес-користувачі	Конструктор форм і процесів, готові шаблони, мінімум налаштувань	Найнижчий поріг входу; обмежена гнучкість
2) Low-code (загальний клас)	Розробка застосунків із мінімальним ручним кодуванням	Проф. розробники + бізнес-користувачі	Візуальне моделювання UI/логіки/даних, розширення кодом за потреби	Швидкість + можливість кастомізації
3) Enterprise LCAP	Побудова та підтримка корпоративних (mission-critical) застосунків	IT-команди, професійні розробники, архітектори	Повний стек: доменна модель, бізнес-логіка, DevOps, governance, масштабування, моніторинг, інтеграції	Підходить для складних enterprise-сценаріїв (напр. Mendix)
4) CADP (Citizen Application Development Platform)	Self-service розробка для типових бізнес-задач	Citizen developers під наглядом IT	Спрощені конструктори, контрольовані компоненти, обмеження для безпеки та відповідності	Менша складність, ніж LCAP; важливе governance
5) BPM / BPA (суміжний клас)	Оркестрація та керування бізнес-процесами end-to-end	Бізнес-аналітики, process owners, IT	Моделювання процесів, правила, виконання, моніторинг, аудит	Суміжна технологія; може доповнювати low-code
6) RPA (суміжний клас)	Автоматизація повторюваних UI-дій у наявних системах	Операційні команди, аналітики автоматизації	Боти, запис дій, інтеграції через UI/API, швидке впровадження	Добре для ‘переклеювання’ систем; не замінює LCAP
7) iPaaS (суміжний клас)	Інтеграція систем і даних (API, конектори,	Інтеграційні/платформені команди	Конектори, маршрутизація, трансформації, API-менеджмент	Фокусується на інтеграціях; використовується разом із LCAP

	обмін повідомленнями)			

Надалі в роботі під low-code платформою розуміється передусім клас enterprise LCAP, оскільки він забезпечує повний стек засобів для побудови прикладних систем (моделювання даних і логіки, розгортання, керування та інтеграції), що важливо для корпоративних фінансових сценаріїв. Водночас BPM/RPA/iPaaS розглядаються як суміжні технології автоматизації, які можуть доповнювати low-code рішення залежно від задачі.

1.1.1 Історичні передумови та становлення лоу-код підходу

Історія лоу-код підходів бере свій початок у концепціях візуального програмування та четвертого покоління мов програмування (4GL), які активно розвивалися ще в 1980–1990-х роках [2], [3]. Інструменти на кшталт Visual Basic, представлений Microsoft у 1991 році, стали ранніми прикладами платформ, що дозволяли будувати інтерфейси та бізнес-логіку за допомогою візуальних конструкторів, істотно знижуючи поріг входу в розробку застосунків [3]. Паралельно розвивалися методології швидкої розробки (Rapid Application Development, RAD), які наголошували на швидкому прототипуванні, ітеративності та повторному використанні компонентів [2]. Значний внесок у популяризацію RAD зробив Джеймс Мартін, який у 1991 році обґрунтував перехід від жорстких каскадних моделей до гнучких, ітеративних процесів, що безпосередньо вплинуло на подальший розвиток low-code інструментів [2].

У 2010-х роках поняття low-code виходить на передній план в бізнес-середовищі: компанії Appian, OutSystems, Mendix та інші починають пропонувати промислові платформи, орієнтовані як на внутрішні бізнес-процеси, так і на клієнтські веб- та мобільні застосунки [2], [3], [4]. Саме в цей період low-code перестає сприйматися лише як інструмент для “внутрішньої

автоматизації" й починає розглядатися як стратегічний підхід до побудови повноцінних корпоративних систем.

Кінець 2010-х та початок 2020-х років характеризуються стрімким зростанням масового використання low-code платформ: за даними досліджень, майже кожна велика організація застосовує певні інструменти low-code/no-code для розбудови цифрових сервісів, оптимізації процесів і скорочення часу виходу продуктів на ринок [8], [9], [11]. Цю динаміку підсилює дефіцит розробників та загальна потреба у масштабуванні цифрових ініціатив у відповідь на глобальні виклики та прискорену цифрову трансформацію [7].

1.1.2 Переваги та виклики використання платформ з лоу-кодом

Аналіз сучасних підходів до прискорення цифрової трансформації показує, що low-code платформи посідають усе важливіше місце в розробці корпоративних інформаційних систем. Їх популярність зумовлена потребою організацій швидко реагувати на зміну бізнес-вимог, скорочувати витрати на створення програмних рішень та мінімізувати залежність від дефіцитних IT-кадрів. Low-code технології пропонують альтернативу традиційній розробці, спрощуючи процес побудови застосунків, забезпечуючи вищу швидкість розробки та доступність для ширшого кола користувачів.

До основних переваг low-code платформ відносять:

- зниження вартості розробки за рахунок скорочення обсягу ручного коду, повторного використання компонентів і меншої залежності від високоспеціалізованих розробників [10], [20];
- прискорення розробки – завдяки візуальному моделюванню та готовим шаблонам час створення прототипів і промислових рішень може скорочуватися в рази порівняно з традиційним підходом [9], [21];

- демократизація розробки – залучення citizen developers та експертів предметної області без глибокої технічної підготовки, що дозволяє швидше перетворювати бізнес-вимоги на робочі рішення [9], [11];
- полегшена інтеграція та супровід – стандартні інтерфейси інтеграції та централізоване керування моделями спрощують внесення змін і підтримку життєвого циклу застосунків [10], [11], [22];
- подолання дефіциту IT-кадрів – можливість частково компенсувати нестачу розробників за рахунок залучення фахівців з бізнес-підрозділів [1], [7], [9].

Водночас використання лоу-код платформ супроводжується низкою викликів:

- питання безпеки та відповідності регуляціям – застосунки, створені некваліфікованими користувачами, можуть містити вразливості, а неконтрольоване розгортання рішень призводить до явища “shadow IT” [5], [6], [23];
- обмеження продуктивності та масштабованості – системи, спочатку спроєктовані як невеликі, з часом можуть ставати критично важливими та вимагати оптимізації, яку не завжди легко реалізувати в межах платформи [5];
- обмежена гнучкість та кастомізація – у деяких випадках платформа накладає суттєві обмеження на реалізацію специфічних вимог, що потребує додаткових обхідних рішень або інтеграції зі сторонніми системами [5], [6].

Усе це вимагає від організацій впровадження зрозумілих політик, процесів рев’ю та контролю, а також системної роботи із забезпечення архітектурної цілісності портфеля low-code рішень.

1.2 Принципи проєктування систем на основі low-code. Технічні аспекти проєктування low-code систем

Принципи проєктування low-code систем включають орієнтацію на користувача, послідовність дизайну, масштабованість, безперервне вдосконалення та співтворення між учасниками процесу. У сукупності ці принципи дозволяють створювати застосунки, які одночасно відповідають вимогам бізнесу, забезпечують позитивний користувацький досвід і мають потенціал до подальшого розширення. Проте впровадження low-code підходів пов’язане й з низкою ризиків – зокрема, обмеженнями масштабованості, залежністю від конкретного постачальника платформи, викликами інтеграції та безпеки [35], [11]. Тому підприємства мають зважено підходити до застосування low-code технологій, поєднуючи їхні переваги з ефективними інструментами управління, контролю та планування.

Незважаючи на концептуальну простоту, low-code системи мають низку суттєвих технічних вимог. Їх ігнорування може призвести до проблем продуктивності, надійності або безпеки:

– більшість low-code платформ мають вбудовані засоби аутентифікації, авторизації, шифрування та журналювання. Однак зростає ризик нехтування глибоким аудитом безпеки, особливо коли розробка здійснюється нетехнічними користувачами [7], [22]. У фінансових установах важливо дотримуватися регуляторних вимог, включаючи політики доступу, контроль змін і перевірку відповідності [20];
– продуктивність low-code застосунків залежить від правильної організації логіки, оптимальних моделей даних та уникнення надмірно ресурсомістких операцій. Великі набори даних, складні фільтри та неоптимальні інтеграційні сценарії можуть створити вузькі місця [34], [35], [36];
– ефективність low-code рішень значною мірою визначається можливість інтеграції з ERP, CRM, core-banking, DWH та іншими критичними

системами. Різні платформи мають різний рівень підтримки API, конекторів та стандартів інтеграції [36], [39]. Вибір платформи повинен базуватися на її здатності інтегруватися з теперішніми і майбутніми системами організації [9], [10], [15];

– сучасні low-code платформи здебільшого є cloud-native і підтримують автоматичне горизонтальне та вертикальне масштабування на основі контейнеризації та оркестрації (Docker, Kubernetes) [7], [11];
– попри наявність візуальних редакторів та готових UI-компонентів, які спрощують створення інтерфейсів, розробники мають враховувати базові принципи UX: доступність, послідовність, інтуїтивність та мінімізацію когнітивного навантаження [8], [12], [18].

Отже, ефективне проєктування low-code систем потребує врахування низки взаємопов'язаних технічних факторів – безпеки та контролю доступу, продуктивності, інтеграційної сумісності з критичними системами, масштабованості та якості UX/UI. Незважаючи на візуальний характер розробки, low-code рішення залишаються повноцінними корпоративними застосунками, для яких обов'язковими є вимоги до надійності, керованості та комплаєнсу. Ігнорування зазначених аспектів може призвести до виникнення "вузьких місць" у продуктивності, підвищення операційних і безпекових ризиків або до складнощів інтеграції в IT-ландшафт організації. Відповідно, систематичний облік цих вимог на етапі проєктування є необхідною умовою реалізації потенційних переваг low-code підходу, особливо у фінансових та інших критично важливих доменах.

1.3 Платформа Mendix як приклад low-code рішення

Mendix є однією з провідних low-code платформ для розробки корпоративних застосунків. Платформа була заснована у 2005 році як відповідь на потребу прискорити створення програмного забезпечення та зменшити

залежність від традиційних підходів до кодування [4], [5]. Основна ідея Mendix полягає в тому, щоб надати можливість проєктувати та реалізовувати бізнес-застосунки з мінімальним обсягом ручного коду, використовуючи модельно-орієнтовану (model-driven) розробку.

Значним етапом у розвитку платформи стало її придбання компанією Siemens у 2018 році. Це дало змогу інтегрувати Mendix в екосистему промислового Інтернету речей (Industrial IoT), розширити глобальну присутність та посилити фокус на індустріальних і корпоративних сценаріях використання [3], [5]. На тлі загального росту ринку low-code рішень, що прискорився після пандемії COVID-19, Mendix посів стійку позицію серед лідерів завдяки поєднанню зручних інструментів для citizen developers та розширених можливостей для професійних розробників [2], [24], [16].

Від самого початку Mendix позиціонувалася як інструмент для швидкого створення бізнес-застосунків, здатний скоротити часові та ресурсні витрати традиційних IT-проєктів. Засновники платформи побачили обмеження класичного підходу до розробки – довгі цикли погодження, дефіцит розробників, складність змін – і запропонували модель, у якій бізнес-експерти та розробники працюють із єдиною візуальною моделлю застосунку [49], [12], [15].

Після придбання Siemens платформа почала активніше інтегруватися у промислові сценарії: MES-системи, моніторинг виробничих ліній, рішення для smart factory, IoT-аналітика тощо [5], [7]. Це дозволило позиціонувати Mendix не лише як «класичну» low-code платформу для бізнес-додатків, але й як інструмент для створення критично важливих промислових рішень.

Останні роки Mendix розвиває стратегію, орієнтовану на поєднання low-code з AI/ML, розширення маркетплейсу модулів та поглиблення інтеграції з хмарними платформами та сервісами великих постачальників (AWS, Azure тощо) [5], [8], [9].

1.4 Особливості моделювання фінансових даних у банках та DFIs

Моделювання фінансових даних є ключовим інструментом банків та інститутів розвитку (development finance institutions, DFI) для аналізу поточного стану, прогнозування майбутніх показників і підтримки стратегічного ухвалення рішень. Фінансові моделі дозволяють кількісно оцінювати фінансову стійкість, рентабельність, структуру ризиків та вплив управлінських рішень на капітал і ліквідність. У випадку DFI, фінансове моделювання також пов’язане з оцінкою впливу на розвиток, стійкість портфеля в складних юрисдикціях та відповідність мандату установи [11], [13], [25].

Ефективність фінансового моделювання безпосередньо впливає не лише на окрему установу, але й на стабільність фінансової системи в цілому: від якості моделей залежать рішення щодо кредитування, інвестицій, реструктуризації, управління ризиками та виконання регуляторних вимог [3], [10]. Водночас моделювання стикається з викликами – фрагментованістю даних, ручними процесами, складними вимогами до управління моделями та контролю якості даних [1], [6], [17]. Це підштовхує організації до пошуку технологічних підходів, які поєднують гнучкість моделей, високу якість даних та прозорість розрахунків.

1.4.1 Застосування моделювання у DFIs

Для DFI роль фінансового моделювання має додатковий вимір: поряд з фінансовою стійкістю оцінюється розвитковий вплив, стійкість проєктів у високоризикових середовищах та відповідність мандату установи [9], [17].

Фінансові моделі використовуються у процесі підготовки інвестиційних пропозицій (CIP/FP) для аналізу фінансової стійкості позичальників, структур проєктного фінансування, структури cash flow та покриття боргу при різних сценаріях (base, downside) [9], [8]. Кредитні департаменти перевіряють моделі

на предмет стійкості до шоків, якості припущень та відповідності ризик-апетиту [12].

DFI фінансують середньо- та довгострокові проєкти (енергетика, інфраструктура, фінансові інтермедіари тощо), тому моделі зазвичай охоплюють тривалий часовий горизонт із детальним описом операційних та фінансових припущень [3], [8], [9]. Важливе значення має чутливість до ключових драйверів (тарифи, завантаження, курсові коливання, субсидії, податковий режим).

DFI активно застосовують сценарні моделі для аналізу впливу макрошоків, змін регуляторного середовища, геополітичних ризиків та кліматичних факторів на портфель і капітальну позицію [11], [12]. Це дозволяє приймати рішення щодо диверсифікації портфеля, country limits, структури продуктів та ролі гарантійних інструментів.

Моделі відіграють ключову роль у підготовці звітності для акціонерів, донорів, фондів (наприклад, blended finance структур) та регуляторів. Вони мають відповідати стандартам фінансового та регуляторного обліку, що ставить високі вимоги до прозорості, відтворюваності та документованості розрахунків [1], [2], [5].

1.4.2 Виклики та обмеження фінансового моделювання

Фінансове моделювання відіграє ключову роль у діяльності банків і фінансових інституцій розвитку (DFIs), оскільки результати моделей безпосередньо впливають на оцінку ризиків, прогнозування ліквідності й капіталу, підготовку інвестиційних рішень та виконання вимог комплаєнсу. Висока якість моделей і даних є критичною не лише для внутрішнього управління, а й для стабільності фінансового профілю установи в умовах стресу, коли потрібні точні, повні й своєчасні ризик-звіти. Однак на практиці процеси створення та підтримки моделей часто залишаються недостатньо

формалізованими й технологічно фрагментованими, що підвищує операційні ризики та знижує відтворюваність розрахунків.

Попри критичну важливість фінансових моделей, на практиці існує низка системних проблем:

- фрагментованість даних та ручні процеси. Дані часто зберігаються у різних системах (ERP, core-banking, CRM, проектні системи) без повноцінної інтеграції. Це змушує команди FP&A, ризик-менеджменту та фронт-офісу опиратися на ручні експорти й консолідацію в електронних таблицях, що підвищує ризик помилок і затримок [15], [16], [17];
- низька якість даних та регуляторні вимоги. Неповні, несвоєчасні або неконсистентні дані можуть маскувати критичні ризики (ліквідності, кредитні, операційні). Регуляторні стандарти на кшталт BCBS 239 та вимоги щодо захисту даних (GDPR) підсилюють потребу в системному підході до якості й управління даними [1], [10], [17];
- недостатнє управління моделями (model risk). За відсутності чіткої політики управління моделями виникають ризики: дублювання моделей, відсутність валідації, некеровані зміни формул та макросів, втрата знань при зміні персоналу [9], [13]. Це особливо небезпечно для моделей, що впливають на капітал, резерви та регуляторні показники;
- обмеження традиційних інструментів. Звичайні електронні таблиці, попри свою гнучкість, погано масштабуються в умовах великих масивів даних та складних сценаріїв stress testing. Вони також обмежені у можливостях аудиту, контролю доступу та централізованого управління версіями [8], [15].

Ці виклики створюють попит на більш структуровані, керовані та інтегровані підходи до побудови фінансових моделей – зокрема із застосуванням спеціалізованих платформ та low-code рішень.

1.5 Порівняння low-code та традиційної розробки у фінансовому секторі

Порівняння low-code розробки з традиційною (pro-code) розробкою у фінансовому секторі відображає глибоку трансформацію підходів фінансових установ до створення та впровадження програмних рішень. Low-code платформи (Low-Code Development Platforms, LCDP) пропонують візуальне моделювання, готові компоненти та високу швидкість розробки, залучаючи не лише професійних розробників, але й бізнес-користувачів – так званих citizen developers [18], [19]. Це особливо актуально для банків і фінансових інститутів розвитку, де поєднуються високі регуляторні вимоги, потреба в швидких змінах та складні внутрішні процеси [12], [21].

За даними низки досліджень, використання low-code рішень може скорочувати час розробки до 70–90 % та зменшувати витрати на створення застосунків до 70 % порівняно з традиційними підходами [22], [23], [14]. Водночас традиційна розробка зберігає вирішальне значення там, де потрібна максимальна гнучкість, глибока кастомізація, контроль над вихідним кодом і реалізація складних нефункціональних вимог (безпека, продуктивність, регуляторна відповідність) [25], [26]. Отже, для фінансового сектору питання вже не стільки в «протиставленні» підходів, скільки в пошуку оптимального балансу та сценаріїв їх поєднання.

2 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО LOW-CODE СИСТЕМИ МОДЕЛЮВАННЯ ФІНАНСОВИХ ДАНИХ

У цьому розділі розглядаються особливості процесів моделювання фінансових даних у міжнародних фінансових інституціях розвитку (development finance institutions), зокрема в Нідерландському банку розвитку (FMO), а також формуються вимоги до цільової low-code системи моделювання. На відміну від абстрактних теоретичних підходів, що часто описуються в літературі з фінансового моделювання, моделі у таких інституціях є практичним інструментом для ухвалення інвестиційних рішень, управління ризиками та моніторингу портфеля протягом усього життєвого циклу проєкту [1], [23].

Сучасні дослідження підкреслюють, що ефективність фінансових моделей значною мірою залежить від якості даних, чіткості методології та можливості відтворити розрахунки в часі [23], [25]. На практиці у багатьох фінансових організаціях, включно з інституціями розвитку, ключові моделі будуються в електронних таблицях (Microsoft Excel). Такий підхід дає гнучкість, але ускладнює централізоване управління, контроль версій, аудит змін та інтеграцію з іншими інформаційними системами [24], [26]. Це особливо критично для установ, які працюють із довгостроковими проєктами в країнах з підвищеним рівнем ризику, де помилки в моделюванні можуть мати суттєві наслідки для фінансової стійкості та репутації організації [12].

2.1 Особливості процесів моделювання фінансових даних у DFIs

Фінансова модель у діяльності DFIs виконує роль центрального аналітичного артефакту, навколо якого будується значна частина інвестиційного процесу. Вона використовується не лише як інструмент оцінки прибутковості проєкту, але й як засіб для аналізу ризиків, перевірки

відповідності фінансовим ковенантам та обґрунтування структурних рішень щодо фінансування. Для інституцій розвитку, які працюють у складних юрисдикціях та секторах з підвищеною волатильністю, вимоги до якості та прозорості моделей є особливо високими [12], [18].

Типовий інвестиційний проєкт проходить низку етапів: від початкового аналізу та структуризації до довгострокового моніторингу та, за потреби, реструктуризації. На кожному з цих етапів фінансова модель розвивається: змінюється рівень деталізації, джерела даних, сценарії аналізу та коло користувачів. При цьому базова логіка – побудова прогнозних грошових потоків, розрахунок показників ефективності (таких як внутрішня норма дохідності, рівень покриття боргу, коефіцієнти ліквідності) та аналіз чутливості до ключових припущень – залишається незмінною [1], [9].

2.1.1 Роль фінансової моделі в інвестиційному циклі проєкту

Життєвий цикл інвестиційного проєкту у DFIs можна умовно поділити на кілька основних фаз: первинний аналіз, остаточне погодження структури фінансування, періодичний портфельний моніторинг та, за необхідності, реструктуризація. На кожній з цих фаз фінансова модель виконує специфічну функцію, але в усіх випадках залишається головним інструментом для кількісної оцінки проєкту.

На етапі попереднього аналізу готується початкова версія моделі, яка відображає базові припущення щодо обсягу інвестицій, структури фінансування, прогнозних доходів та витрат. Мета цієї версії – відповісти на питання, чи є проєкт у принципі життєздатним: чи здатні прогнозні грошові потоки забезпечити обслуговування боргу, який рівень волатильності може витримати проєкт, яка приблизна дохідність для інвестора. Часто на цьому етапі модель ще не є формальною «кредитною моделлю», але закладає основу для подальшої структуризації.

На стадії підготовки фінального рішення щодо фінансування модель суттєво ускладнюється. Вона інтегрує результати фінансового, технічного, юридичного, податкового та екологічного аналізів, відображає реальну структуру угоди, умови кредитних договорів, механізми індексації та можливі резервні сценарії [12]. Ця версія моделі використовується інвестиційним комітетом для ухвалення рішення і розглядається як офіційний документ, тому вимоги до її якості, пояснюваності та можливості аудиту є надзвичайно високими.

Після надання фінансування модель переходить у режим «живого інструменту» портфельного менеджменту. Вона регулярно оновлюється фактичними даними позичальника, використовується для перевірки виконання фінансових ковенантів, оцінки потреби в коригувальних заходах та підготовки внутрішніх звітів. У разі погіршення фінансових показників та виникнення ризику дефолту модель слугує основою для аналізу альтернативних сценаріїв – зміни графіку погашення, введення пільгових періодів, часткової капіталізації відсотків тощо.

Отже, одна й та сама модель упродовж років підтримує різні фази інвестиційного циклу, а отже, потребує чіткої структури, історії змін та механізмів контролю доступу, що і є важливим аргументом на користь переходу до спеціалізованої системи на базі low-code платформи [7], [11].

2.1.2 Ключові процеси, що ґрунтуються на фінансовому моделюванні

Фінансова модель використовується не ізольовано, а в рамках ширшого набору процесів, що охоплюють як попередній аналіз проєкту, так і його подальший супровід. Для міжнародної інституції розвитку, такої як FMO, можна виділити кілька груп процесів, де модель є центральним елементом.

По-перше, це процеси підготовки інвестиційних рішень. На ранніх етапах модель допомагає визначити доцільність подальшої розробки проєкту, на подальших – виступає основою для фінального кредитного рішення. Саме через

модель у рішення інтегруються результати технічних, ринкових та юридичних аналізів, а також висновки з оцінки екологічних та соціальних ризиків.

По-друге, це процеси портфельного моніторингу. У цьому контексті модель використовується для регулярного оновлення прогнозів з урахуванням фактичних результатів, перевірки виконання фінансових ковенантів, аналізу причин відхилень від базового сценарію. Результати моделі потім потрапляють до внутрішніх систем звітності та використовуються як на рівні окремих проєктів, так і на агрегованому портфельному рівні [12], [18].

По-третє, важливу роль відіграють сценарний аналіз та стрес-тестування. Міжнародні рекомендації з управління ризиками підкреслюють необхідність оцінки поведінки портфеля у різних макроекономічних та ринкових сценаріях, включаючи різке падіння виручки, зміну відсоткових ставок, девальвацію валют та інші шоки[5], [20]. У практиці FMO це реалізується через побудову базового, песимістичного та стресового сценаріїв у моделі, порівняння результатів і подальше відображення цих висновків у внутрішніх ризикових звітах.

Нарешті, окремою групою процесів є реструктуризації та workout-кейси. У таких ситуаціях модель використовується для розрахунку різних варіантів реструктуризації – змін графіку погашення, введення пільгових періодів, зміни відсоткової ставки, часткового списання або конвертації боргу. Без якісної моделі складно обґрунтувати, який варіант дозволяє досягти балансу між фінансовою стійкістю проєкту та захистом інтересів інвестора.

Усі ці процеси мають спільну рису: вони спираються на одну або кілька версій моделі, створеної переважно в електронних таблицях, з частковим дублюванням логіки та ручними кроками. Це створює передумови для численних операційних ризиків, які розглядаються в підрозділі 2.1.3 [23], [24].

2.1.3 Основні проблемні точки поточного підходу до моделювання

Попри те, що електронні таблиці (Microsoft Excel) залишаються де-факто стандартом фінансового моделювання, їх використання у великих інституціях розвитку супроводжується низкою системних проблем. У науковій та прикладній літературі ці проблеми описуються через призму якості даних, управління моделями та операційних ризиків [23], [25], [26].

Однією з ключових проблем є фрагментованість даних. Фінансова модель зазвичай отримує вхідні дані з декількох джерел: систем управління клієнтами, сховищ даних, систем обліку, інструментів управління ризиками, а також із зовнішніх джерел (наприклад, макроекономічні припущення). За відсутності інтегрованої платформи значна частина перенесення даних відбувається вручну через імпорт та експорт файлів, копіювання та вставлення, що підвищує ризик помилок, затримок і непослідовності у використанні вихідної інформації [24], [26].

Друга група проблем стосується версіонування та аудиту. Зазвичай фінансова модель зберігається у вигляді файлу, який надсилається електронною поштою або завантажується в спільні папки. Кожна нова версія створюється шляхом копіювання попередньої, при цьому виникають «паралельні» версії з відмінними припущеннями і змінами. Відстежити, яка саме версія була використана для конкретного рішення інвестиційного комітету, хто і коли вніс зміни в ключові формули, часто буває складно або неможливо без додаткових ручних процедур документування [23], [25].

Третя проблема пов’язана з масштабованістю та підтримуваністю моделей. У міру ускладнення структури проєктів та росту портфеля окремі моделі стають настільки громіздкими, що їх супровід перетворюється на окрему спеціалізацію. Будь-яка зміна логіки, наприклад впровадження нового підходу до стрес-тестування чи зміна вимог до внутрішньої звітності, вимагає ручного оновлення великої кількості файлів, що збільшує ризик непослідовної імплементації [9], [11], [23].

Окремо слід зазначити обмеженість можливостей для централізованого контролю доступу, трасування дій користувачів та автоматизованого

впровадження політик комплаєнсу. У контексті регуляторних вимог та внутрішніх контролів це створює додатковий ризик невідповідності встановленим стандартам[25], [27]. Саме ці проблемні точки формують запит на перехід від моделей у вигляді ізольованих файлів до інтегрованої low-code системи, яка забезпечує єдине джерело правди (single source of truth), прозору історію змін і можливість масштабувати логіку моделювання на рівень всієї інституції[5], [7], [11], [23].

2.2 Поточні технологічні виклики та потреби у цифровізації процесів моделювання

Поточний підхід до моделювання фінансових даних у міжнародних інституціях розвитку, включно з FMO, історично сформувався як комбінація електронних таблиць, окремих файлів, фрагментованих баз даних та локальних інструментів аналізу. Така "гібридна" екосистема виросла органічно навколо потреб швидкого ухвалення рішень у фронт-офісі та ризик-функціях: Excel-моделі легко копіювати, адаптувати під конкретний проєкт і швидко будувати сценарії. Водночас із ростом портфеля, збільшенням кількості джерел даних та появою додаткових вимог до прозорості й контролів (в т.ч. внутрішніх і регуляторних) цей підхід проявив системні обмеження: низька інтегрованість, слабка керованість версій/доступів, складність відтворення розрахунків та контрольованого комплаєнсу.

У більшості випадків процес моделювання будується як послідовність напівручних кроків. По-перше, дані для моделі збираються з кількох операційних і довідкових систем (клієнтські/портфельні реєстри, фінансові системи, зовнішні ринкові джерела), а частина припущень зберігається в окремих "історичних" файлах або локальних таблицях. Далі ці дані експортуються, очищуються та узгоджуються вручну, після чого

завантажуються у фінансову модель (часто – у вигляді шаблонів, які копіюються й модифікуються під конкретний кейс). У разі відсутності центральної платформи користувачі змушені постійно переносити дані між системами, імпортувати/експортувати таблиці та узгоджувати показники – що саме по собі є джерелом помилок і втрат часу.

По-друге, сценарний аналіз і стрес-тестування в такій моделі часто реалізуються через “гілки” розрахунків у файлі або набір копій файлів (base/downside тощо). Це забезпечує гнучкість, але ускладнює відстеження того, які саме припущення були змінені, ким, коли, та як це вплинуло на результат. На рівні організації це перетворюється на ризик “розходження істин”: різні команди можуть працювати з різними версіями моделі або різними інтерпретаціями одних і тих же вхідних даних, а перевірка стає трудомісткою. Саме тому в документах щодо управління моделями та End-User Applications (EUA, зокрема Excel) підкреслюється, що помилки або некоректне використання моделей створюють фінансові й репутаційні ризики, а контроль має будуватись через інвентаризацію, документацію, періодичний перегляд і валідацію.

По-третє, типові “операційні механізми” роботи з моделями (погодження, рев’ю, обмін файлами, внесення правок) часто відбуваються через пошту/папки/спільні сховища. У такій схемі контроль доступу, аудит змін та централізоване версіонування є обмеженими або залежать від дисципліни користувачів, що підвищує ризик неконтрольованих змін і складність відновлення історії рішень. У внутрішніх матеріалах це прямо пов’язується з Model/EUA ризиком: організації потрібні реєстри (inventory), стандарти розробки, регулярні review/validation та мінімальні стандарти для критичних EUAs [23], [24].

Зі зростанням масштабу діяльності “файлова” модель дедалі гірше відповідає вимогам до керованості та контролю, які очікуються від фінансової

організації. У FMO-контексті це додатково підсилюється фокусом на цифровізації процесів у відповідь на регуляторні вимоги: внутрішні програми прямо фіксують, що цифровізація портфельних процесів стартувала через регуляторний тиск і надалі поширюється на ранні стадії інвестиційного процесу (lead management, onboarding, управління контактами тощо). Це демонструє загальний тренд: зростає потреба не лише в "інструменті моделювання", а в наскрізному цифровому процесі, де дані, контроль та відстежуваність є вбудованими.

2.2.1 Основні проблемні точки поточного підходу до моделювання

Один із найбільш системних викликів – слабка інтегрованість даних між основними інформаційними системами FMO. Інвестиційні та ризикові команди використовують кілька окремих джерел: систему управління клієнтами, окремі портфельні сховища, зовнішні ринкові бази даних, а також ручні файли з історичними припущеннями. Сучасні дослідження підкреслюють, що саме фрагментованість даних є ключовим фактором, який знижує якість фінансових моделей і створює ризики несумісності між вихідними джерелами та проміжними розрахунками [23], [26].

У відсутності центральної платформи користувачі змушені вручну переносити дані між системами, імпортувати та експортувати таблиці, очищувати інформацію та узгоджувати показники. Такі дії збільшують ризик помилок, створюють дублікати, а також ускладнюють аудит того, які саме дані були використані у конкретній версії моделі.

Low-code платформи, як описано у літературі [5], [7], передбачають можливість створення інтеграційних модулів, що можуть уніфікувати джерела даних через API або спеціалізовані конектори, усуваючи потребу в ручних операціях. У цьому контексті цифровізація стає не лише питанням зручності, але й необхідним елементом для підвищення надійності та достовірності фінансового аналізу.

2.2.2 Основні проблемні точки поточного підходу до моделювання

У діяльності міжнародних інституцій розвитку дедалі більшу роль відіграють вимоги прозорості, звітності та ретельного документування рішень. Зокрема, внутрішні аудити та зовнішні партнери (наприклад, державні агенції та міжнародні фінансові організації) часто вимагають документального підтвердження фінансових прогнозів, припущень і методологій.

У контексті регуляторних норм (включно з GDPR для персональних даних, внутрішніми політиками кредитного ризику та рекомендаціями Basel Committee) установи повинні забезпечити контроль доступу, фіксацію змін і можливість швидкого відтворення результатів [27].

На практиці ж досягти цього засобами електронних таблиць складно. Відсутність формального audit trail та неможливість централізовано контролювати доступ користувачів створюють додаткові комплаєнс-ризики. Дослідження вказують на те, що low-code платформи завдяки вбудованим механізмам контролю доступу, журналам подій та стандартизованим моделям суттєво знижують ці ризики та спрощують регуляторну відповідність [7], [11].

2.3 Функціональні вимоги до low-code системи моделювання фінансових даних

Функціональні вимоги до системи моделювання фінансових даних, що розробляється на основі платформи low-code, визначаються як особливостями діяльності FMO та інших інституцій розвитку, так і загальними принципами сучасного фінансового аналізу. Ефективна система має не лише реплікувати можливості традиційних електронних таблиць, але й вирішувати їх фундаментальні недоліки – забезпечувати прозорість, відтворюваність, контрольованість та безперервну інтеграцію з бізнес-процесами [23], [24].

Отже, наведені функціональні вимоги формують цілісну рамку для створення low-code системи фінансового моделювання, яка має забезпечити не лише зручність роботи користувачів, а й керованість процесу розрахунків на рівні організації. На відміну від традиційних електронних таблиць, така система повинна гарантувати стандартизоване введення припущень, централізовану реалізацію розрахункової логіки та контрольовані механізми сценарного аналізу без ризику “прихованих” змін у формулах. Ключовим наслідком виконання цих вимог є можливість отримувати результати, які можна однозначно відтворити та перевірити, маючи прозору історію змін, контроль доступів і відповідність внутрішнім політикам та регуляторним очікуванням. Отже, функціональні вимоги виступають не переліком опцій, а набором умов, необхідних для перетворення фінансового моделювання на контрольований, масштабований і надійний цифровий процес.

2.3.1 Формалізація введення припущень і параметрів фінансової моделі

Однією з основних функцій майбутньої системи є можливість структурованого введення припущень (assumptions) та параметрів, що формують грошові потоки, операційні витрати, вартість фінансування та макроекономічні індикатори. У традиційній моделі на основі електронних таблиць ці дані зберігаються у різних частинах файлу, що ускладнює аудит та відтворення моделі [25].

Low-code підхід дозволяє створити централізовану форму введення припущень, прив’язану до доменної моделі даних. Це означає, що кожне припущення матиме чітко визначений тип, механізм валідації, джерело, відповідального користувача та часову позначку. Дослідження показують, що саме формалізація припущень є ключовим фактором підвищення якості фінансових моделей та зменшення ймовірності людських помилок [26], [27].

Крім того, у межах платформи повинна бути реалізована підтримка різних категорій припущень:

– статичні параметри (наприклад, ставка кредиту, інфляція, індексні коефіцієнти);
– динамічні припущення, які можуть змінюватися в часі;
– зовнішні ринкові дані, імпортовані через інтеграції (API).

Отже, введення припущень стає не спорадичною операцією, а частиною формалізованого бізнес-процесу, що підсилює дисципліну моделювання в організації.

2.3.2 Автоматизована генерація грошових потоків та фінансових показників

Ключова цінність фінансової моделі полягає у здатності створювати прогнозні грошові потоки (cash flow) та низку похідних фінансових показників, включаючи чисту приведену вартість, коефіцієнти обслуговування боргу, рентабельність тощо. У традиційних моделях такі розрахунки виконуються за допомогою формул на рівні клітинок, що робить їх вразливими до неочевидних змін та випадкового редагування[23].

У low-code системі логіка розрахунків реалізується через мікропроцеси (microflows) або бізнес-правила, що забезпечує:

– повну прозорість алгоритмів;
– уніфікацію логіки для всіх проєктів;
– захист від несанкціонованих змін;
– можливість централізованого оновлення формул у разі зміни методології.

Фахові публікації підкреслюють, що автоматизація розрахунків дозволяє значно знизити ризики, пов’язані з ручними моделями, а також підвищить відтворюваність та точність прогнозів [24], [27].

Окрім розрахунків у базовому сценарії, система повинна підтримувати побудову альтернативних сценаріїв із різними наборами припущень, що є критично важливим для оцінки ризиків у діяльності інституцій розвитку [21].

2.3.3 Підтримка сценарного аналізу та моделювання ризиків

Сценарний аналіз (scenario analysis) – одна з базових вимог для фінансових моделей у міжнародних фінансових інституціях. FMO застосовує кілька категорій сценаріїв: базовий, песимістичний (downside), стресовий, а у деяких випадках – політико-економічні комбіновані сценарії.

У системі low-code сценарії повинні існувати як окремі об’єкти з власними наборами припущень та логікою застосування. Такий підхід дозволяє:

- швидко порівнювати результати між сценаріями;
- забезпечувати повну відтворюваність кожного сценарію в історичній перспективі;
- створювати audit trail рішень та змін параметрів.

Аналітичні джерела підкреслюють, що саме можливість автоматизованої побудови сценаріїв значно підвищує точність оцінки ризиків та якість управлінських рішень[21], [24].

Крім того, система повинна забезпечувати інтеграцію з підрозділами ризиків для використання єдиної методології розрахунків, що є обов’язковою вимогою відповідно до внутрішніх стандартів FMO.

2.3.4 Версіонування моделей і контроль доступу

Одна з найбільш критичних вимог – можливість автоматичного версіонування фінансової моделі. У традиційних підходах до моделювання версіонування виконується вручну, що створює численні операційні та комплаєнс-ризики[23].

Low-code підхід передбачає централізоване зберігання моделі у репозиторії, де кожне оновлення логіки, припущень або параметрів фіксується системою. Це дає можливість:

- відтворити модель на будь-який момент історії;
- зафіксувати автора та суть змін;
- контролювати, які користувачі мають право змінювати певні частини моделі;
- мінімізувати ризик ручних помилок та випадкового видалення формул.

Наукові джерела вказують, що саме контрольованість історії змін є ключовою вимогою для впровадження професійних систем моделювання у корпоративних середовищах[25], [27].

2.4 Нефункціональні вимоги та вимоги до безпеки й комплаєнсу

Нефункціональні вимоги відіграють ключову роль у визначенні того, наскільки ефективно low-code система здатна підтримувати фінансове моделювання в установах розвитку. Якщо функціональні вимоги описують «що має робити система», то нефункціональні – «як саме вона має це робити», враховуючи продуктивність, надійність, безпеку, масштабованість та відповідність внутрішнім і зовнішнім регуляторним стандартам.

Для організацій на кшталт FMO якість нефункціональних характеристик часто є навіть важливішою за набір функцій, оскільки системи працюють з конфіденційними фінансовими, правовими та корпоративними даними, що вимагає високого рівня захисту, прозорості та операційної стійкості [10], [11], [21]. У цьому підрозділі розглянуто ключові аспекти, що формують нефункціональний профіль цільової low-code системи моделювання.

2.4.1 Продуктивність та обробка великих фінансових масивів

Моделі фінансових потоків для проєктів розвитку часто містять великі таблиці з помісячними розрахунками на 15–25 років. Обсяг даних у таких моделях може сягати десятків тисяч рядків. Ефективність системи визначається здатністю швидко виконувати обчислення, формувати сценарії та візуалізувати результати без затримок, оскільки ці операції є частиною щоденної роботи інвестиційних та ризикових аналітиків.

Традиційні інструменти на кшталт електронних таблиць погано масштабуються при збільшенні обсягів даних, що призводить до значних затримок і помилок у процесах обробки [23], [24]. Low-code система натомість повинна виконувати складні циклічні розрахунки через оптимізовані серверні процеси, забезпечуючи стабільний час відповіді навіть при великому навантаженні. У публікаціях відзначається, що продуктивність є критичною нефункціональною характеристикою систем фінансового аналізу, оскільки повільне виконання розрахунків знижує оперативність ухвалення рішень [27].

2.4.2 Масштабованість і підтримка зростання портфеля

Оскільки портфель FMO зростає за кількістю операцій та складністю структур фінансування, система моделювання повинна легко масштабуватися як у вимірі даних, так і в кількості користувачів. Low-code платформи переважно підтримують горизонтальне масштабування за рахунок хмарної інфраструктури, що дозволяє системі адаптуватися до збільшеного навантаження без радикальної зміни архітектури [21].

Особливо важливою є масштабованість у контексті сценарного аналізу: для великих портфелів одночасне генерування кількох альтернативних сценаріїв може значно збільшити обчислювальні потреби. У дослідженнях підкреслюється, що сучасні low-code платформи здатні забезпечувати ефективну підтримку таких процесів у середовищах зі змінним навантаженням, що робить їх прийнятними для фінансових установ [24].

2.4.3 Досвід користувача та підтримка бізнес-процесів

Оскільки основними користувачами системи є не технічні спеціалісти, а інвестиційні менеджери, аналітики, фахівці з ризиків та співробітники комплаєнс-підрозділів, система повинна мати інтерфейс, який дозволяє виконувати складні фінансові операції без потреби у спеціальних технічних навичках.

Дослідження свідчать, що зручність інтерфейсу та інтуїтивність робочих процесів прямо впливають на якість моделювання та швидкість прийняття рішень [17], [21].

Особлива увага приділяється:

– чіткій візуалізації складних фінансових структур;
– зручному введенню припущень;
– гнучким фільтрам і параметрам аналізу;
– можливості швидко порівнювати сценарії та результати.

Отже, нефункціональна вимога до користувацького досвіду є не вторинною, а фундаментальною, оскільки вона визначає прийнятність системи для операційної діяльності інституції.

Вигляд інтерфейсу наводиться на рисунку 2.1.

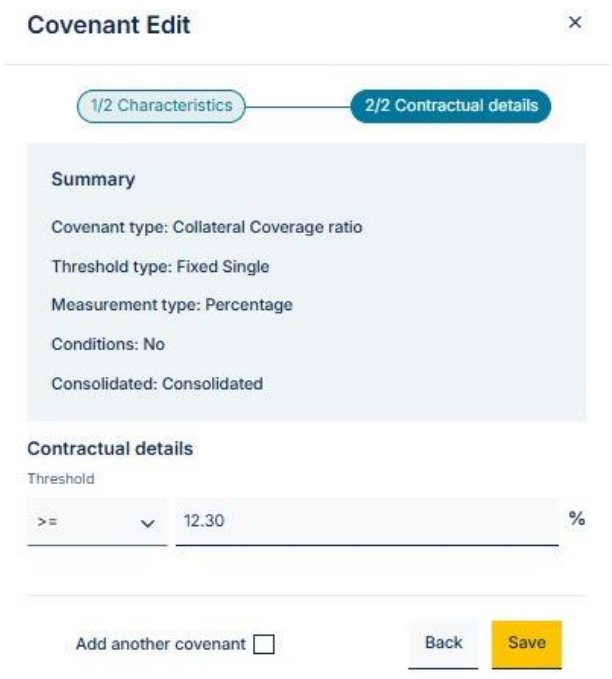


Рисунок 2.1 – Вигляд інтерфейсу

2.5 Обґрунтування вибору Mendix як платформи для реалізації цільової системи

Вибір технологічної платформи для побудови системи моделювання фінансових даних є стратегічним рішенням, що визначає її довговічність, масштабованість та відповідність операційним потребам інституції розвитку. У попередніх підрозділах було виокремлено низку функціональних і нефункціональних вимог, серед яких – обробка великих масивів фінансових даних, підтримка сценарного аналізу, автоматизовані механізми аудиту, інтеграція з внутрішніми системами, прозорість змін, висока доступність та відповідність регуляторним вимогам. У цьому контексті платформа Mendix демонструє відповідність ключовим критеріям і забезпечує поєднання гнучкості low-code підходу з можливостями корпоративного класу.

2.5.1 Відповідність функціональним вимогам системи

Архітектура Mendix побудована на використанні доменної моделі, мікропроцесів (microflows) та інтеграційних компонентів, що дозволяє формувати складні логічні конструкції без надмірної залежності від ручного програмування. Ці можливості є критичними для реалізації сценарного аналізу, моделювання грошових потоків, автоматичного розрахунку показників та підтримки багатоваріантності моделей, що є стандартною практикою у фінансових організаціях [5], [6].

Доменна модель Mendix забезпечує строгість у роботі з фінансовими сутностями, дозволяючи чітко визначати структури даних, зв’язки та обмеження. Це сприяє підвищенню якості моделей і дозволяє уникати розрізненості даних, яка властива інструментам на основі електронних таблиць [23].

Microflows, у свою чергу, дозволяють формувати правила обробки, перевірки узгодженості та розрахункові алгоритми, що робить платформу здатною підтримувати процеси створення фінансових моделей з високим ступенем автоматизації.

2.5.2 Відповідність нефункціональним вимогам та вимогам до безпеки

Mendix підтримує хмарну інфраструктуру з можливістю горизонтального масштабування, що є необхідною умовою для роботи з великими таблицями грошових потоків, обчисленнями на тривалі часові горизонти та паралельним виконанням кількох сценаріїв аналізу [21], [24].

З погляду безпеки Mendix забезпечує:

– контроль доступу на рівні ролей;
– audit trail для кожної транзакції та зміни;
– шифрування даних;
– можливість ізоляції середовищ для розробки, тестування та продуктивної експлуатації.

У дослідженнях підкреслюється, що low-code платформи корпоративного рівня здатні забезпечувати відповідність внутрішнім контролям установ та зовнішнім регуляторним стандартам, включно із вимогами до прозорості, захисту даних і комплаєнсу[10], [27]. Це робить Mendix технологічно сумісним з вимогами FMO щодо захисту конфіденційної інформації та забезпечення операційної стійкості.

3 ПРОЄКТУВАННЯ LOW-CODE СИСТЕМИ МОДЕЛЮВАННЯ ФІНАНСОВИХ ДАНИХ

Проєктування системи моделювання фінансових даних на платформі Mendix ґрунтується на поєднанні (i) принципів предметно-орієнтованого (domain-driven) моделювання, (ii) функціональних і нефункціональних вимог, сформованих у розділі 2, та (iii) архітектурних патернів корпоративної low-code розробки. У межах цього підходу бізнес-об’єкти фінансового моделювання (проєкт/угода, версія моделі, припущення, сценарії, грошові потоки, показники та журнали контролю) формалізуються у доменній моделі, а обчислювальна та керуюча логіка реалізуються у вигляді керованих візуальних процесів.

На відміну від традиційної розробки, де значна частина інженерної роботи виконується вручну й потребує окремих шарів коду для інтеграцій, контролю доступу та журналювання, Mendix дозволяє моделювати поведінку застосунку на рівні мікропроцесів (microflows), бізнес-правил і композиції об’єктів, забезпечуючи узгодженість логіки, повторне використання компонентів і кращу керованість змін [5], [11]. Відтак проєктне рішення орієнтоване не лише на автоматизацію розрахунків, а й на підвищення прозорості, відтворюваності та аудитності моделювання – критичних характеристик для фінансових інституцій розвитку.

3.1 Загальна архітектура системи

Архітектура цільової системи моделювання фінансових даних побудована за принципами багаторівневого корпоративного застосунку (multi-tier enterprise application) із розділенням відповідальностей між рівнями даних, бізнес-логіки, інтеграцій та представлення. Така декомпозиція зменшує зв’язність компонентів, підвищує модульність і спрощує супровід, що є особливо

важливим для фінансового моделювання, де зміни методологій, вимог до звітності або джерел даних мають бути контрольованими та відстежуваними [10], [21].

Кожен рівень виконує визначену роль, а взаємодія між ними здійснюється через стандартизовані механізми платформи Mendix (керування сесіями та ролями, транзакційне збереження сутностей, виклики мікропроцесів, публікація/споживання сервісів), що забезпечує цілісність системи та знижує кількість технічних залежностей.

Архітектура low-code системи моделювання фінансових даних на платформі Mendix побудована за багаторівневим принципом із чітким розподілом відповідальностей між шарами. Такий підхід забезпечує керованість рішення, знижує зв’язність компонентів, спрощує супровід і масштабування, а також створює передумови для виконання ключових нефункціональних вимог: відтворюваності розрахунків, прозорості змін (auditability), безпеки та продуктивності. У структурі системи виділяються рівень даних, рівень бізнес-логіки, інтеграційний рівень, рівень доступу користувачів і наскрізні компоненти управління та контролю:

– рівень даних (Data layer). Рівень даних виконує роль «єдиного джерела правди» для всієї системи та зберігає повний набір фінансових сутностей, припущень, сценаріїв і результатів. Основою цього шару є доменна модель, яка формалізує структуру предметної області через сутності та їхні зв’язки. У межах моделювання фінансових даних доцільно виділяти сутності на кшталт Project/Deal, Model, Assumption, Scenario, CashFlow, Indicator, а також службові сутності для контролів і трасування змін (наприклад, ValidationLog, AuditEvent). Реалізація в Mendix здійснюється засобами Domain Model (Entities, Associations), де персистентні сутності зберігаються у вбудованій базі даних Mendix (зокрема PostgreSQL) або у керованому сховищі відповідно до обраної інфраструктури. Для уніфікації довідників та бізнес-словника застосовуються enumerations

(наприклад, валюти, частоти, типи припущень), а також правила валідації на рівні сутностей (обов’язковість полів, перевірки діапазонів тощо). На вхід рівня даних надходять дані з корпоративних систем (ERP/CRM/DWH), довідкова інформація, припущення користувачів і метадані версій. На виході формується нормалізований набір даних для розрахунків, збережені результати моделювання та журнали змін. Для забезпечення якості даних використовуються механізми цілісності: референтна цілісність через associations, обмеження (constraints), типізація атрибутів, а також політики версійності, де версії моделі або сценарію фіксуються як окремі об’єкти. Підготовка до аудиту підтримується збереженням авторства змін, часових міток і походження даних. Масштабованість посилюється індексуванням ключових атрибутів (наприклад, ProjectId, ModelVersion, Date), що зменшує час вибірок для сценарних розрахунків і аналітики;

– рівень бізнес-логіки (Business logic). Рівень бізнес-логіки реалізує алгоритмічну частину фінансової моделі та правила предметної області. Саме тут виконуються обчислення, що перетворюють структуровані дані та припущення на прогнозні грошові потоки та фінансові показники. До типових задач належать генерація cash flow, розрахунок метрик на кшталт NPV, DSCR, а також перевірки узгодженості й валідація припущень. Окремий фокус становить оркестрація сценаріїв (наприклад, базовий, негативний/downside, стрес-тести) та аналіз чутливості, що дозволяє оцінювати ризики та стійкість проєктів/угод до змін ключових параметрів. Також цей рівень відповідає за керування життєвим циклом моделі: створення, редагування, рецензування, затвердження та запуск розрахунків. У Mendix бізнес-логіка здебільшого реалізується через microflows і nanoflows для процесів, трансформацій і розрахунків, а формалізація правил може бути додатково підтримана decision flows або decision

tables. Для пакетних операцій (наприклад, нічних перерахунків чи масових прогонів сценаріїв) застосовуються scheduled events. Використання custom Java actions доцільне лише в разі специфічних оптимізацій або нестандартних інтеграцій, щоб не втрачати переваги low-code підтримуваності. Вхідними даними для бізнес-логіки є нормалізовані сутності доменної моделі, параметри обраного сценарію, а також тригери від інтерфейсу користувача або інтеграційних викликів. Вихідними результатами виступають згенеровані ряди грошових потоків, таблиці показників, статуси валідації та повідомлення для користувача. Відтворюваність забезпечується фіксацією версії правил/алгоритму (rule set version) разом із набором вхідних припущень, що дозволяє повторити розрахунок у незмінному вигляді. Прозорість реалізації посилюється декомпозицією на зрозумілі кроки (inputs → calc → outputs). Продуктивність досягається оптимізацією доступу до даних (уникнення N+1 запитів, агрегування, пагінація), а також застосуванням асинхронних задач для важких обчислень. Тестованість підтримується модульною структурою microflows і використанням контрольних тест-сценаріїв та тест-даних;

– інтеграційний рівень (Integration layer). Інтеграційний рівень забезпечує зв’язок системи з внутрішніми та зовнішніми джерелами даних і сервісами. Його функції охоплюють підключення до ERP/CRM/Core-систем, сховищ даних (DWH), джерел ринкових показників (market data), а також імпорт/експорт у поширених форматах (CSV, Excel, PDF) і синхронізацію довідників. Крім обміну даними, інтеграційний рівень виконує роль API-шлюзу, що дозволяє зовнішнім сервісам викликати функції системи або отримувати дані для подальшої обробки. Типова реалізація в Mendix включає опубліковані REST API endpoints, використання OData для аналітичних потреб і підключення BI, а також конектори Mendix,

виклики REST до сторонніх сервісів або черги повідомлень за необхідності. Ключовим механізмом інтеграцій є mapping microflows, які реалізують конвеєр transform → validate → persist: перетворення вхідних структур, перевірка якості та запис у доменну модель. Вхідними сигналами інтеграційного рівня є запити API, вебхуки, пакетні файли, OData-потоки й маркери автентифікації/авторизації (SSO). Вихідними результатами є відповіді API, синхронізовані сутності, експортні файли та повідомлення, передані у зовнішні системи. Безпека забезпечується механізмами OAuth2/SAML/SSO, політиками доступу, allow-lists і обмеженням частоти запитів (API throttling). Якість даних підтримується перевірками структури (schema checks) і процедурами звірки (reconciliation). Надійність посилюється повторними спробами (retry), ідемпотентністю імпортів та контролем дублювання. Спостережуваність досягається логуванням інтеграційних викликів, використанням correlation IDs і структурованими журналами для аналізу інцидентів;

– рівень доступу користувачів (Presentation / Access layer). Рівень доступу користувачів забезпечує веб-інтерфейс для введення припущень, керування моделями, запуску сценаріїв та перегляду результатів у зручній формі. На цьому рівні реалізуються рольові сценарії роботи для різних груп користувачів (наприклад, інвестиційний аналіз, ризик-менеджмент, портфельний супровід, контроль, адміністрування). Важливою складовою є підтримка процесів review/approval, коментування, формування задач і сповіщень, що підсилює контрольованість і відповідність внутрішнім процедурам. У Mendix інтерфейс реалізується через Mendix Pages, widgets і layouts, а керування доступом — через рольову навігацію та security-механізми на рівні сутностей і атрибутів. Для зниження ризику помилок введення використовуються UI-валидації, керовані форми та покрокові майстри (wizards), які стандартизують процес

внесення припущень. За потреби може бути додано мобільний доступ для швидкого погодження або перегляду ключових результатів. Вхідні дані цього рівня включають введення користувачів, завантаження файлів, події UI та параметри пошуку/фільтрації. Вихідними артефактами є екранні звіти й дашборди, експорт у Excel/PDF, системні сповіщення та задачі. Контроль доступу має відповідати принципу найменших привілеїв (least privilege) і забезпечувати розмежування прав на деталізованому рівні. Для auditability важливо відображати історію змін («хто/коли/що змінив») та забезпечувати доступ до журналів дій у межах інтерфейсу. Відмовостійкість підтримується коректною обробкою помилок (graceful errors), збереженням чернеток і застосуванням optimistic locking для уникнення конфліктів при паралельному редагуванні; та

– наскрізні компоненти (Cross-cutting). Наскрізні компоненти охоплюють управління, контроль якості та експлуатаційні аспекти системи. Вони включають governance-механізми (інвентаризація моделей, визначення статусів «fit-for-purpose», дотримання комплаєнсу), ведення журналів і моніторинг, підтримку SLA та сповіщень, а також CI/CD-процеси й керування середовищами (DEV/TEST/ACC/PROD) та конфігураціями. Ці елементи забезпечують керованість змін і стабільність роботи в умовах регулярних оновлень. У Mendix такі вимоги реалізуються через сутності для audit/event logging, засоби журналювання (Logbook), конфігурації керування користувачами й SSO, механізми Mendix pipelines або інтеграцію з зовнішніми DevOps-інструментами, а також інструменти моніторингу (Mendix APM, alerts). На вхід надходять події виконання microflows, зміни даних, деплої та інциденти; на виході формуються аудиторські журнали, звіти контролів, метрики продуктивності та статуси релізів. Відповідність політикам безпеки й зберігання (retention) забезпечується стандартами доступу та

журналюванням, автоматизованими перевірками при релізах і версіонуванням бізнес-правил, що дозволяє керовано еволюціонувати систему без втрати прозорості та відтворюваності.

3.1.1 Логічна схема архітектури

Основу архітектури становить доменно-орієнтована модель даних із явним механізмом версіонування, де ModelVersion реалізує роль агрегата, що задає межі узгодженості (consistency boundary) для фінансового розрахунку. ModelVersion пов’язує бізнес-контекст InvestmentProject із повним набором артефактів моделювання – параметрами (AssumptionSet), сценарними модифікаціями (Scenario) та зафіксованими результатами, що створює підґрунтя для відтворюваності й формального порівняння варіантів. Такий підхід мінімізує типові ризики “файлових” моделей (неявні правки, дублювання, неможливість однозначного відновлення входів) та підтримує керований життєвий цикл моделі (чернетка → рев’ю → затвердження). Компонент PortfolioLink виконує функцію інтеграційного “якоря” – забезпечує ідентифікацію та узгодження даних з корпоративними джерелами, зменшуючи залежність від ручних перенесень. Журнал AuditTrailEntry формалізує аудит як частину доменної моделі: він зберігає події зміни, автора та часову позначку, що дає змогу ретроспективної перевірки розрахунків і підтвердження їх коректності в межах контрольного середовища.

Опис блоків логічної схеми:

– investment project (основна інвестиційна сутність). Агрегує бізнес-контекст моделювання для конкретної угоди/проєкту: ідентифікатори, тип інструмента, контрагент, базові параметри структури, а також метадані для навігації та контролю доступу. Усі версії моделей, сценарії та результати належать до одного

InvestmentProject, що забезпечує організацію моделювання на рівні портфеля;

– model version (версія моделі: припущення, результати, час). Центральний вузол архітектури – «знімок» моделі в конкретний момент часу. Містить посилання на застосований AssumptionSet, пов’язаний(і) Scenario, результати розрахунків (cash flow, показники), статус життєвого циклу (draft/reviewed/approved) та часову позначку. Саме ModelVersion забезпечує відтворюваність: один і той самий набір входів дає той самий результат і може бути перевірений ретроспективно;
– assumption set (набір припущень). Структурований контейнер параметрів моделі (ставки, FX-курси, CAPEX/OPEX, індекси, дисконтні ставки тощо). Параметри типізуються та валідовуються, що зменшує ризик помилок порівняно з вільним редагуванням у таблицях. AssumptionSet прив’язується до ModelVersion, тому для кожного розрахунку відомо, які саме припущення були використані;
– scenario (base/downside/stress). Визначає сценарні відхилення або стрес-шоки до базових припущень (наприклад, зміна ставки, девальвація, зниження виручки). Сценарій може містити набір «override»-значень і правила їх застосування у часі. Прив’язка Scenario до ModelVersion дозволяє порівнювати результати між сценаріями за незмінного базового контексту;
– portfolio link (зв’язок з портфельними даними). Інтеграційний міст між проєктом/версією моделі та зовнішніми джерелами (CRM/DWH/портфельні системи). Зберігає ключі зіставлення (IDs), правила мапінгу та (за потреби) «відбиток» використаних даних. Це дозволяє фіксувати, які саме зовнішні дані підживлювали модель у конкретній версії;

– audit trail entry (аудит змін). Журнал подій і змін: хто, що і коли змінив (author, timestamp, об’єкт зміни, значення до/після). Записи аудиту пов’язуються з ModelVersion, щоб забезпечити доказовість і контрольованість моделювання, включно з підтримкою review/approval та комплаєнс-вимог;

– логіка зв’язків (кардинальності). Наприклад, investment project 1 → * model version: один проєкт має багато версій моделі (еволюція моделі у часі). model version 1 → 1 assumption set: кожна версія використовує визначений набір припущень (або посилання на нього). model version 1 → * scenario: у межах однієї версії можна запускати кілька сценаріїв (base/downside/stress). model version 0.1 → 1 portfolio link: версія може бути пов’язана з портфельними даними (за наявності інтеграції). model version 1 → * audit trail entry: усі значущі дії та зміни логуються для аудиту.

На рисунку 3.1 наведено узагальнену ASCII-діаграму архітектури системи, виконану у стилі технічної документації, характерної для інженерних досліджень.

Ця архітектура поєднує структурну строгість із гнучкістю, властивою low-code платформам, дозволяючи легко розширювати систему новими компонентами без порушення її базових механізмів[14], [21].

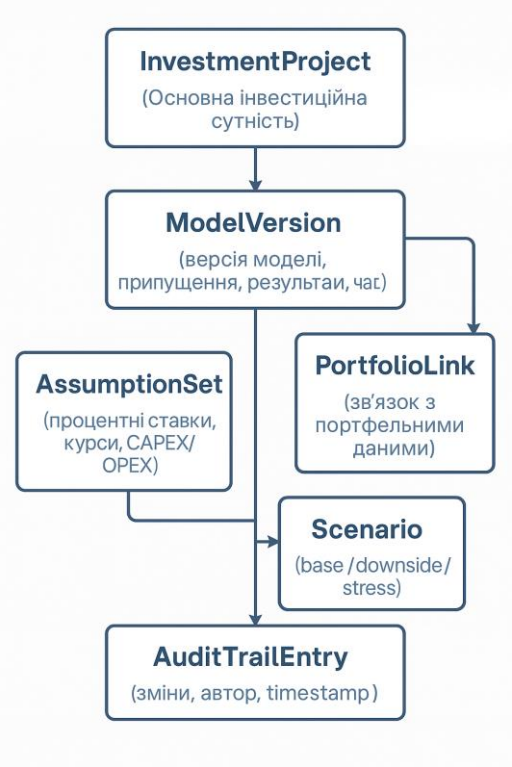


Рисунок 3.1 – Узагальнена ASCII-діаграма архітектури системи

Схема підкреслює, що керованість моделювання досягається не лише обчислювальною логікою, а й правильним структуруванням доменних сутностей: версіонування концентрується в model version, сценарії локалізують зміни припущень, інтеграції винесені у portfolio link, а контрольованість і відповідність вимогам аудиту забезпечуються audit trail entry.

З метою стислого підсумування логічної моделі та взаємозв'язків між її компонентами таблиця 3.2 узагальнює ролі доменних сутностей, характерні атрибути та основні асоціації, показані на рисунку 3.1.

Таблиця 3.2 – Узагальнення ролей доменних сутностей у логічній схемі

Сутність	Роль у схемі	Типові атрибути	Ключові зв'язки
InvestmentProject	Контекст інвестиції/угоди	ProjectId, Customer, InstrumentType, Dates	1→* ModelVersion
ModelVersion	Версійований знімок моделі	VersionNo, Status, CreatedOn, CalculatedOn	→ AssumptionSet; →* Scenario; →* AuditTrailEntry
AssumptionSet	Параметри та припущення	InterestRate, FX, CAPEX/OPEX,	1→1 ModelVersion

Сутність	Роль у схемі	Типові атрибути	Ключові зв'язки
		Indexes	
Scenario	Варіант розрахунку	ScenarioType, Overrides, ShockParams	*→1 ModelVersion
PortfolioLink	Мапінг до портфельних даних	ExternalIds, Mapping, SnapshotRef	0..1→1 ModelVersion
AuditTrailEntry	Аудит і трасування змін	Actor, Timestamp, Entity, Before/After	*→1 ModelVersion

3.1.2 Ролі основних компонентів

Для забезпечення керованості, масштабованості та відтворюваності фінансового моделювання архітектуру системи доцільно розглядати як сукупність взаємопов'язаних рівнів, кожен із яких виконує окрему роль у загальному циклі роботи моделі – від збереження структурованих даних і формалізації припущень до виконання розрахунків, інтеграції з корпоративними джерелами та взаємодії з користувачами. Такий поділ відповідає підходам до побудови корпоративних застосунків і дозволяє чітко розмежувати відповідальності: дані відповідають за консистентність та «єдине джерело правди», бізнес-логіка – за виконання алгоритмів і контроль правил, інтеграції – за синхронізацію із зовнішніми системами, а рівень доступу – за зручну та контрольовану роботу користувачів у межах ролей і прав. У подальшому наведено характеристику кожного рівня та його внесок у реалізацію вимог до low-code системи моделювання:

– рівень даних. Цей рівень відповідає за формування єдиної доменної моделі фінансових даних, включно з структурою інвестиційних проєктів, таблицями грошових потоків, параметрами ризику, сценарними наборами та припущеннями та користувацькими версіями моделей. Така формалізація дозволяє уникати розрізнених Excel-файлів і створює централізоване «єдине джерело правди», що було визначено серед ключових потреб у розділі 2.2 [23];

- рівень бізнес-логіки. Mendix використовує microflows як основний механізм реалізації бізнес-процесів та розрахунків. У системі вони відповідають за генерування грошових потоків, перерахунок моделі у разі зміни параметрів, застосування стресових сценаріїв, аудит дій користувачів та контроль доступу та прав редагування. Microflows дозволяють створювати складні алгоритми без суцільного програмування, що прискорює розробку й підвищує прозорість логіки [5], [6];
- інтеграційний рівень. Враховуючи те, що FMO працює з даними з різних джерел, система повинна забезпечувати синхронізацію з CRM, сховищами даних, Excel-імпортами та ризиковими інструментами. Mendix підтримує сучасні методи інтеграції, включно з REST, SOAP і OData, що дозволяє системі виконувати роль консолідованої моделювальної платформи [14];
- рівень доступу. Користувацький інтерфейс побудовано з урахуванням нефункціональних вимог до зручності та простоти для бізнес-користувачів, які не володіють технічними навичками. Для ілюстрації реалізації рівня доступу та принципів взаємодії користувача із системою на рисунку 3.2 наведено приклад головного інтерфейсу застосунку. Представлений екран демонструє ключові елементи навігації та функціональні блоки, через які користувачі виконують основні операції: введення припущень, запуск/перерахунок моделі, роботу з результатами (таблиці, графіки), перегляд сценаріїв та зіставлення версій. Отже, інтерфейс виступає не лише «візуальною оболонкою», а інструментом, що забезпечує мінімізацію ручних операцій, стандартизацію дій та зниження ризику помилок під час моделювання.

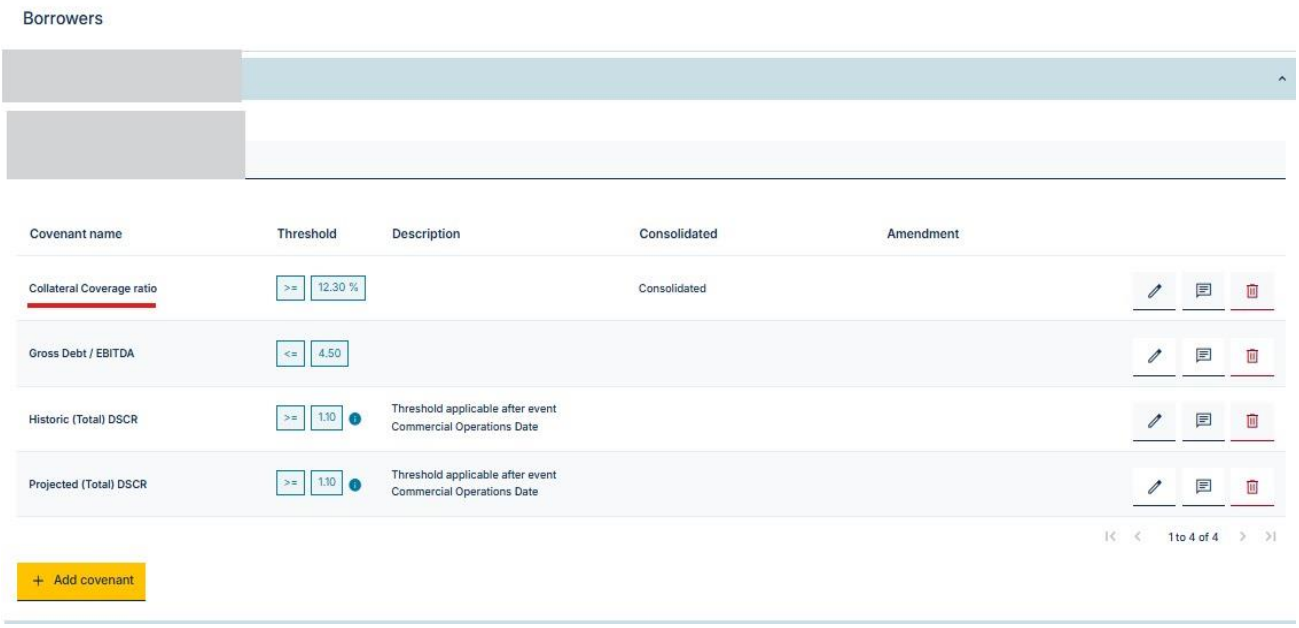


Рисунок 3.2 – Вигляд інтерфейсу програми

Інтерфейс підтримує:

– введення припущень;
– роботу з таблицями та графіками;
– перегляд сценаріїв;
– порівняння версій моделей.

Дослідження підкреслюють, що саме простота взаємодії є визначальним фактором успіху low-code систем у фінансовому секторі [17], [21].

Отже, представлений інтерфейс не лише забезпечує зручність роботи користувача, але й виконує роль механізму стандартизації процесу моделювання: вводить припущення у формалізованому вигляді, підтримує керовану роботу зі сценаріями та зменшує ризик несанкціонованих або випадкових змін. Візуальна структура екранів і типові операції (додавання/редагування/видалення, навігація по об’єктах) сприяють зниженню когнітивного навантаження та пришвидшують виконання повторюваних дій. Як наслідок, інтерфейс підсилює відтворюваність і контрольованість результатів, оскільки всі ключові зміни здійснюються у межах визначених сценаріїв взаємодії та можуть бути пов’язані з конкретними версіями моделі.

3.2 Доменна модель системи

Доменна модель є центральним елементом застосунків, розроблених на платформі Mendix, оскільки вона визначає структуру даних, правила зв’язків між ними та принципи зберігання у персистентному сховищі. Для системи моделювання фінансових даних у фінансовій інституції розвитку доменна модель має відображати не лише об’єкти фінансової діяльності, а й забезпечувати підвищені вимоги до аудитності, трасованості та відтворюваності розрахунків у будь-який момент часу. Це означає, що ключові артефакти моделювання (вхідні припущення, сценарні налаштування, результати та метадані виконання) повинні бути формалізовані як окремі сутності з однозначними зв’язками і контрольованим життєвим циклом.

Розроблена в межах дослідження доменна модель відображає основні етапи інвестиційного циклу – від первинних оцінок та підготовки інвестиційних матеріалів до фаз моніторингу, перегляду припущень, реструктуризаційних розрахунків і аналізу стрес-сценаріїв. Її структура побудована за принципами предметно-орієнтованого моделювання (Domain-Driven Design), що дозволяє відобразити сутності реального бізнес-процесу у вигляді логічно узгоджених компонентів та зменшити розрив між бізнес-термінами і реалізацією в системі [5].

3.2.1 Основні сутності доменної моделі

Ключові сутності доменної моделі визначено з урахуванням типового життєвого циклу фінансової моделі, вимог до сценарного аналізу та необхідності повного аудиту змін. Відбір сутностей здійснювався за

критеріями: (i) бізнес-значущість для інвестиційного рішення та ризик-оцінки; (ii) необхідність версіонування та порівняння результатів; (iii) потреба інтеграції з портфельними/корпоративними даними; (iv) підтримка трасування дій користувачів і доказовості розрахунків.

Узагальнена структура доменної моделі включає такі ключові сутності:

– investment project (інвестиційний проєкт). Центральна сутність, що містить загальну інформацію про фінансування, вид продукту, внутрішні параметри ризику, країну, сектор, валюту та ключові дати. Основна роль – виступати «якорем» для всіх інших даних моделі;
– model version (версія моделі). Окрема версія моделі для певного проєкту, що містить набір припущень, результати розрахунків, сформовані таблиці грошових потоків, сценарії та інформацію про автора й час створення. Це забезпечує повне версіонування та відтворюваність моделювання – вимога, визначена у розділі 2.3 та критична для фінансових установ [21];
– assumption set (набір припущень). Сутність, яка містить параметри, що впливають на модель: процентні ставки, курсові припущення, темпи зростання, графіки погашення, операційні та капітальні витрати, а також параметри кредитного ризику. Це дозволяє повторно використовувати та порівнювати припущення в різних сценаріях;
– cash flow item (елемент грошового потоку). Одна з найважливіших сутностей системи, оскільки фінансове моделювання значною мірою базується на коректній генерації потоків платежів. Сутність включає: дату, валюту, категорію платежу, суму та сценарну належність. Особливу увагу приділено оптимізації роботи з великими таблицями, що є вимогою нефункціональних характеристик (розділ 2.4) [10];
– scenario (сценарій). Містить параметри для аналізу base, downside, stress-cases. Завдяки окремому сценарному компоненту система дозволяє: порівнювати сценарії, зберігати їх історію, икористовувати параметри чутливості та здійснювати стрес-тестування;

– portfolio link (зв’язок з портфелем). Дає змогу інтегрувати модель із системами моніторингу портфеля, зокрема з інструментами оцінки ризику та фінансовими сховищами даних.У доменній моделі зв’язки реалізовано через асоціації 1–* або – залежно від структури портфеля;
– audit trail entry (запис аудиту). Системна сутність, що фіксує зміни у будь-якій частині моделі, включно з: зміною припущень, зміною користувацьких прав, зміною структура моделі та запуском розрахунків.

Це реалізує вимоги до трасованості та відповідності комплаєнсу, визначені в розділі 2.4, і є критично важливою функціональністю фінансових застосунків [10], [21].

Для коректної реалізації доменної моделі в Mendix важливо не лише перелічити сутності, а й чітко зафіксувати логіку їх взаємозв’язків: які об’єкти агрегують інші, де виникає множинність, а де потрібна прив’язка до конкретної версії моделі. З цією метою нижче наведено табличне подання асоціацій між ключовими сутностями (Таблиця 3.3 нижче), яке доповнює текстовий опис підрозділу 3.2.1. Така форма подання забезпечує однозначне трактування кардинальностей та спрощує перевірку вимог до версіонування, сценарного аналізу й аудитності під час моделювання структури даних у Mendix.

Таблиця 3.3 – Матриця зв’язків між ключовими сутностями доменної моделі

Сутність-джерело	**Зв’язана сутність**	**Кардинальність**	**Семантика зв’язку (пояснення)**
InvestmentProject	ModelVersion	1 → *	Один інвестиційний проєкт має множину версій моделі (еволюція моделювання в часі: draft/review/approved).
ModelVersion	AssumptionSet	1 → 1	Кожна версія моделі використовує визначений набір припущень; забезпечує відтворюваність входів для конкретного розрахунку.
ModelVersion	Scenario	1 → *	У межах однієї версії можуть виконуватися кілька сценаріїв (base/downside/stress) для порівняння результатів.

Сутність-джерело	Зв'язана сутність	Кардинальність	Семантика зв'язку (пояснення)
ModelVersion	CashFlowItem	1 → *	Версія моделі породжує детальні ряди грошових потоків як результат розрахунків (вихідні дані моделі).
Scenario	CashFlowItem	1 → *	Кожен сценарій формує власний набір рядків cash flow (прив'язка результатів до сценарного контексту).
ModelVersion	PortfolioLink	0..1 → 1	Версія моделі може мати (або не мати) інтеграційний зв'язок із портфельними/корпоративними джерелами даних; фіксує ідентифікатори та мапінг.
ModelVersion	AuditTrailEntry	1 → *	Усі значущі дії та зміни в межах версії (редагування, запуск розрахунку, погодження) журналюються для аудиту.

3.2.2 Узагальнена діаграма доменної моделі

Узагальнене подання доменної моделі доцільне для того, щоб зафіксувати базову логіку фінансового моделювання на рівні предметної області та одночасно показати, як ця логіка відображається у типових механізмах Mendix. У спрощеному вигляді модель можна звести до послідовності «Проєкт → Версія → Припущення → Потоки» (як зображено на малюнку 3.3 нижче), де проєкт задає бізнес-контекст, версія моделі фіксує стан розрахунку у часі, припущення формують вхідні параметри, а грошові потоки є основним результатом, на основі якого обчислюються похідні показники. Додаткові компоненти (сценарії, інтеграційні посилання та аудит) забезпечують порівнянність варіантів, відтворюваність та контрольованість змін.

На рисунку 3.3 показано, як доменний контур моделювання реалізується через базові шари платформи Mendix. Кожний блок діаграми відповідає окремому аспекту системи та фіксує межі відповідальності між введенням даних, виконанням розрахунків, зберіганням сутностей і зовнішніми інтеграціями:

– інтерфейс користувача (UI) – забезпечує кероване введення припущень та параметрів (форми/майстри), запуск перерахунку

моделі й перегляд результатів. UI мінімізує ручні операції та стандартизує послідовність дій користувача;

– microflows (бізнес-логіка) – реалізують правила й алгоритми: валідацію даних, створення/оновлення версій моделі, генерацію грошових потоків, застосування сценарних відхилень, а також ініціюють логування подій аудиту;
– доменна модель (дані) – містить сутності та зв'язки (Project, ModelVersion, AssumptionSet, Scenario, CashFlowItem, AuditTrailEntry, PortfolioLink) і забезпечує узгоджене зберігання стану моделі та результатів розрахунку;
– зовнішні інтеграції (external integrations) – використовуються для підвантаження портфельних/корпоративних даних і ринкових параметрів (FX, ставки), а також для експорту результатів у суміжні системи через REST/Odata;
– база даних Mendix (Mendix Database) – забезпечує персистентне зберігання доменних сутностей і транзакційність операцій, що є передумовою надійності та відтворюваності розрахунків.

У спрощеному вигляді доменна модель виглядає як зображено на рис. 3.3.

Отже, діаграма підкреслює, що «Проєкт → Версія → Припущення → Потоки» є не лише концепцією предметної області, а й організаційним принципом реалізації: всі ключові операції (введення, перерахунок, сценарії, аудит) прив'язуються до конкретної версії моделі, що забезпечує порівнянність та відтворюваність результатів.

Доменна структура відображається і на рівні користувацького інтерфейсу: екрани системи організовані навколо об'єктів проєкту та його версій, а введення параметрів виконується через керовані форми з валідаціями. На рис. 3.4–3.5 нижче наведено приклади екранних форм, які демонструють підхід до стандартизованого введення параметрів та роботи з переліками даних.

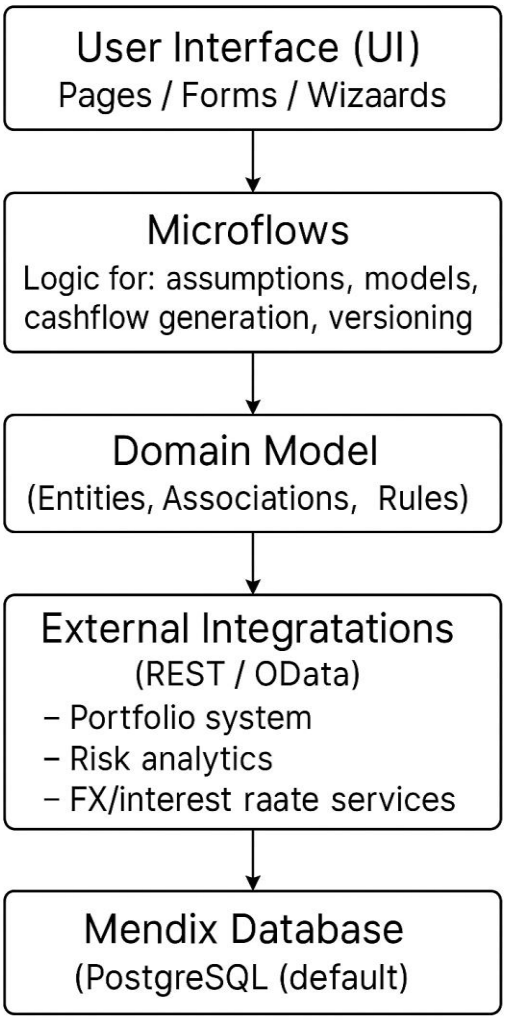


Рисунок 3.3 – Узагальнена схема реалізації доменної моделі в Mendix

Рисунок 3.4 ілюструє роботу з переліками: користувач бачить стандартизовану таблицю параметрів/об’єктів, може виконувати типові дії (перегляд, редагування, підтвердження/скасування), а також контролювати значення ключових полів без переходу до зовнішніх файлів. Такий формат підтримує порівняння та перевірку даних перед перерахунком моделі.

Рисунок 3.5 демонструє керовану форму (wizard) для введення або редагування параметрів. Розбиття на кроки та наявність випадаючих списків/типізованих полів знижує ризик некоректного вводу, а валідації на рівні UI і microflows забезпечують узгодженість даних із доменною моделлю. У результаті введення параметрів стає частиною формалізованого процесу, що напряму підтримує вимоги до контрольованості та аудитності моделювання.

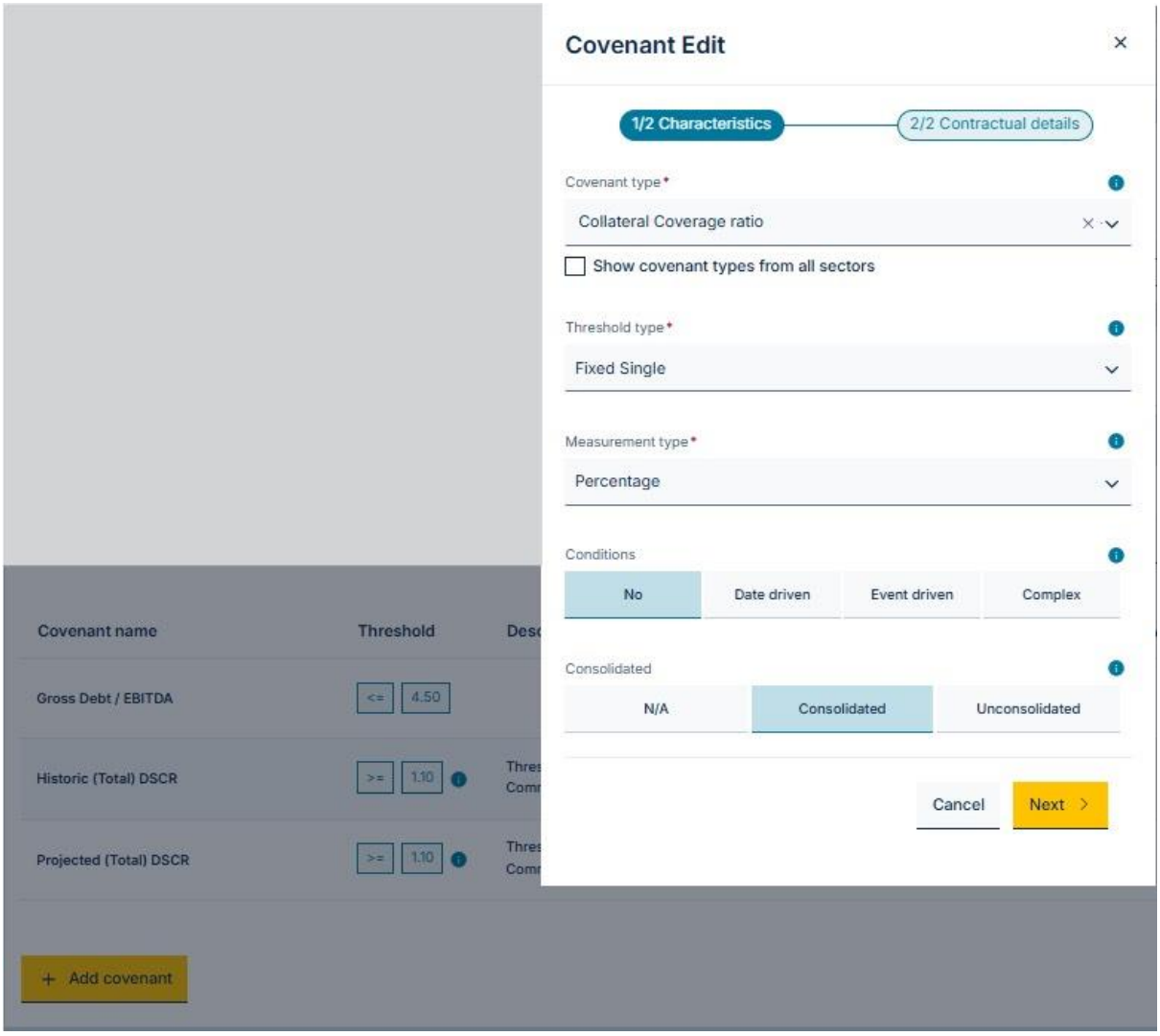


Рисунок 3.4 – Вигляд інтерфейсу (екран переліку/табличного представлення даних)

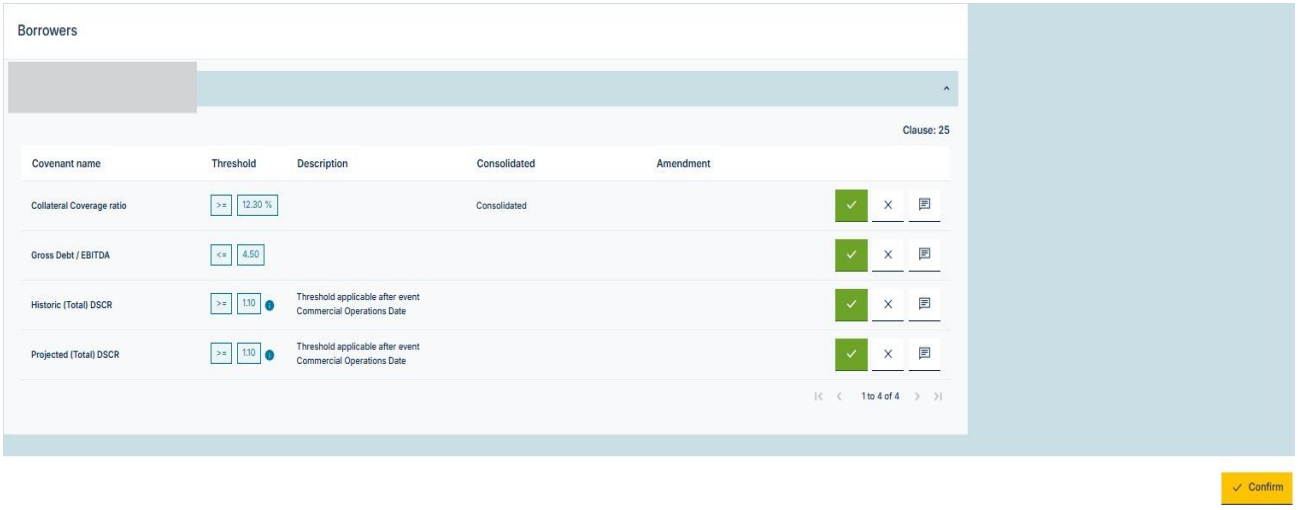


Рисунок 3.5 – Вигляд інтерфейсу (майстер введення/редагування параметрів)

3.2.3 Особливості реалізації доменної моделі в Mendix

Структура доменної моделі враховує як специфічні вимоги FMO, так і обмеження low-code архітектури:

– оптимізація великих таблиць CashFlowItem. Mendix дозволяє працювати з великими наборами даних, однак потребує оптимізації запитів, пагінації та вибіркового завантаження. Це відповідає рекомендаціям щодо продуктивності low-code систем [8], [14];
– вбудована логіка прав доступу (entity access rules). Завдяки Mendix можна на рівні доменної моделі визначати правила: хто може редагувати припущення, хто може створювати версії та хто має право запускати сценарні розрахунки. Це є ключовою вимогою комплаєнсу (GDPR та внутрішніх політик), розглянутою у розділі 2.4;
– підтримка версіонування через ModelVersion. На відміну від Excel, де контроль версій є фрагментованим, Mendix дозволяє зберігати: повну історію, параметри, використані у кожній версії, сценарні набори та результати розрахунків. Це відповідає кращим практикам моделювання фінансових даних [29].
– аудитність через AuditTrailEntry Впровадження audit trail – один з головних аргументів на користь low-code у фінансовому секторі, адже автоматичний журнал змін зменшує операційний ризик і забезпечує відтворюваність рішень [10].

3.3 Реалізація логіки моделювання фінансових даних у Mendix

Отже, реалізація логіки моделювання фінансових даних у Mendix формує цілісне середовище, у якому структура даних (доменна модель) і процедури обробки (microflows) взаємодіють як єдиний механізм розрахунку та управління моделлю. У такій архітектурі фінансова модель описується не як

набір розрізнених формул чи скриптів, а як формалізований об’єкт із визначеними входами, правилами трансформації та зафіксованими виходами, що зменшує випадковість виконання розрахунків і підсилює технологічну дисципліну моделювання. Ключовим наслідком є те, що логіка обчислень стає явною і структурованою: кроки перерахунку, перевірки даних, побудова грошових потоків і застосування сценарних модифікацій оформлюються як керовані процеси, які можуть бути повторені, проаналізовані та підтримані в єдиному стандарті для всіх проєктів.

У порівнянні з підходами, де обчислювальна частина існує як “локальна” реалізація (наприклад, у файлах або персональних скриптах), Mendix-реалізація робить модель організаційно керованою: правила доступу, валідації та зміни параметрів вбудовуються у механіку виконання, а версіонування перетворює розрахунок на контрольований стан системи. Саме зв’язка доменної моделі з microflows забезпечує природне розмежування відповідальностей: дані зберігаються як узгоджені сутності, сценарії – як контрольовані варіанти вхідних умов, а результати – як структуровані вихідні набори (зокрема, cash flow), що дозволяє уникати “роз’їзду” методології між різними моделями та підходами до розрахунку. Крім того, інтеграційні компоненти вбудовують у контур моделювання зовнішні джерела (портфельні, ризикові та ринкові параметри), завдяки чому модель працює в системному контексті даних організації, а не в ізольованому обчислювальному середовищі.

Отже, реалізація логіки моделювання в Mendix поєднує обчислювальну коректність із вимогами корпоративного контролю, характерними для DFIs: модель підтримує керовані зміни, узгодженість входів і виходів, сценарне порівняння та відстежуваність ключових дій. У результаті фінансове моделювання переходить від “інструмента розрахунку” до платформенного процесу, де методологія, дані й результати організовані як керована система, здатна масштабуватися на рівень портфеля та забезпечувати стабільність і прозорість упродовж усього життєвого циклу проєктів.

3.3.1 Обробка припущень та параметрів моделі

Фінансове моделювання неможливе без чіткої роботи з припущеннями (Assumptions), які визначають ключові фінансові параметри: ставки дисконту, FX-курси, графіки погашення, операційні надходження, очікувані збитки, кредитні маржі тощо. У Mendix ця логіка реалізується через окремий модуль Assumption Processing, який відтворює поведінку традиційних «Input Sheets» у Excel, але з вбудованими механізмами валідації, контролю цілісності та аудиту.

Мікрофлоу обробки припущень виконує:

- перевірку коректності параметрів (діапазони значень, формат дат, відповідність фінансовому календарю);
- валідацію логічної узгодженості (наприклад, ставка дисконту не може бути негативною, cash flow не може мати порожні періоди);
- створення зв’язку між припущеннями та конкретною версією моделі (ModelVersion), що гарантує історичність і повторюваність розрахунків;
- формування підготовленого набору вхідних значень, який далі використовується у генерації грошового потоку[9].

Таке структуроване введення припущень значно знижує ризики, пов’язані з помилками ручного вводу, які притаманні Excel-моделям, та забезпечує відповідність вимогам внутрішнього аудиту DFIs[5, 11].

3.3.2 Генерація грошового потоку (Cash Flow Engine)

Грошовий потік проєкту (cash flow) є центральним вихідним результатом фінансового моделювання. У Mendix ця логіка реалізована у вигляді низки мікрофлоу, що виконуються послідовно та формують деталізовані елементи CashFlowItem для кожного періоду моделі.

Алгоритм генерації cash flow включає:

– створення часової шкали (timeline) відповідно до частоти моделі (квартальна, річна, місячна);
– розрахунок операційних потоків, включаючи надходження та витрати;
– моделювання боргових потоків: амортизація, проценти, комісії;
– обробка валютних конверсій, якщо проєкт мультивалютний;
– формування сукупних показників: чистий cash flow, DSCR, NPV, IRR.

Розрахункові мікрофлоу виконуються оптимізовано, з мінімізацією доступів до бази даних та використанням тимчасових об’єктів у пам’яті. Це відповідає рекомендаціям щодо продуктивності low-code платформ при роботі з великими структурованими даними[14].

У порівнянні з Excel-моделями, Mendix забезпечує:

– стабільність обчислень незалежно від користувача;
– відсутність ризику випадкового редагування формул;
– структурованість, яка полегшує ревізію моделі [9, 12].

3.3.3 Реалізація сценарного аналізу

Сценарний аналіз (base, downside, stress) відіграє ключову роль у DFIs, які працюють з високоризиковими проєктами та зобов’язані оцінювати можливі наслідки зовнішніх шоків. У Mendix сценарний модуль реалізується як шар додаткових мікрофлоу, що модифікують базові cash flow залежно від заданих припущень.

Основні модифікації включають:

– зміни у виручці (зниження при stress-сценарії);
– збільшення операційних витрат;
– коригування FX-курсів та ставок капіталу;
– застосування кредитних шоків (затримки погашення, списання частини експозиції).

Такі підходи відповідають методологіям stress testing та scenario analysis, описаним у сучасній фінансовій літературі [5, 20].

Цільовий результат – ScenarioResultSnapshot, який фіксується у системі разом із AuditTrailEntry, що забезпечує простежуваність.

3.3.4 Версіонування та аудит змін

У Mendix реалізовано повноцінну модель версіонування, яка враховує вимоги до reproducibility та auditability, критичні для DFIs й банківських організацій [10, 11].

Версіонування включає:

- клонування поточної моделі з усіма припущеннями;
- фіксацію статусів (Draft → Review → Final);
- створення окремих записів AuditTrailEntry для кожної зміни параметрів;
- забезпечення read-only захисту для затверджених моделей.

Ця логіка належить до best practices low-code governance, що дають змогу мінімізувати помилки користувачів і забезпечити регуляторну відповідність [6].

Підсумовуючи, модель версіонування та аудиту змін у Mendix формує керований життєвий цикл фінансової моделі, який одночасно підтримує відтворюваність (reproducibility) та аудитопридатність (auditability) – критичні вимоги для DFIs і банківських організацій. Клонування поточної моделі разом із усіма припущеннями гарантує, що кожен результат можна “відтворити” у тому самому контексті даних і параметрів, а фіксація статусів (Draft → Review → Final) вводить зрозумілі контрольні точки: що ще є робочою версією, що проходить перевірку, а що є затвердженим джерелом істини. Завдяки цьому зникає типова для spreadsheet-підходів проблема “неясно, яка саме версія і чому стала базою для рішення”.

Не менш важливим є те, що створення окремих записів AuditTrailEntry для кожної зміни параметрів забезпечує деталізований слід “хто/коли/що саме змінив”, що підвищує прозорість, дисциплінує процес внесення правок і зменшує операційні ризики (помилки користувачів, несанкціоновані

коригування, втрату історії рішень). Read-only захист для затверджених моделей додатково закріплює принцип контрольованої незмінності: затверджені версії не редагуються напряму, а будь-яка модифікація проходить через створення нової версії та повторне погодження. У підсумку, описана логіка відповідає best practices low-code governance та узгоджується із загальними підходами governance/контролю модельного ризику у фінансовому секторі, спрощуючи підготовку до внутрішніх і зовнішніх аудитів. За потреби більш жорстких вимог (довготривале зберігання, масштабований пошук, незмінність журналів) підхід може посилюватися використанням розширених механізмів аудиту на кшталт Advanced Audit Trail.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досліджено проблематику цифровізації процесів моделювання фінансових даних у контексті діяльності міжнародних фінансових установ розвитку. Проведений аналіз продемонстрував, що традиційні інструменти, зокрема електронні таблиці, не забезпечують належного рівня прозорості, відтворюваності, аудиту та інтегрованості з корпоративними системами, що є критичним для інституцій з високими вимогами до управління ризиками та комплаєнсу. Це обумовлює потребу у впровадженні сучасних технологічних підходів, здатних забезпечити більш структуроване, контрольоване та масштабоване середовище моделювання.

У роботі здійснено комплексний аналіз концепції low-code розробки, її переваг та обмежень для фінансового сектору, а також проведено оцінку придатності платформи Mendix для реалізації системи моделювання фінансових даних. Дослідження показало, що Mendix здатен забезпечити високу швидкість розробки, консистентність бізнес-логіки, автоматичне ведення журналів аудиту, можливість моделювання складних процесів через мікрофлоу та підтримку розширених інтеграційних механізмів. Це робить платформу перспективним інструментом для побудови систем підтримки прийняття рішень у банках та установах розвитку.

У межах роботи було сформовано та обґрунтовано функціональні та нефункціональні вимоги до системи моделювання; спроєктовано доменну модель та архітектуру; описано логіку обробки припущень, генерації грошових потоків та сценарного аналізу; визначено підходи до версіонування та забезпечення аудиту. Окрему увагу приділено інтеграції з портфельними, ризиковими та інформаційними системами, що є ключовою передумовою для створення єдиного джерела достовірних даних (single source of truth).

Практична цінність отриманих результатів полягає у можливості їх безпосереднього застосування у фінансових установах розвитку, зокрема в контексті міжнародних банків розвитку (DFIs). Запропонована концепція low-code системи моделювання дозволяє значно скоротити операційні витрати, мінімізувати ризики, пов’язані з ручним введенням даних, підвищити швидкість підготовки фінансових моделей та забезпечити їх контрольованість відповідно до внутрішніх і зовнішніх вимог комплаєнсу.

Разом із тим дослідження показало, що low-code платформи мають свої обмеження – зокрема у частині продуктивності при опрацюванні великих масивів даних, реалізації надто специфічних фінансових алгоритмів і залежності від архітектурної політики постачальника. Тому для досягнення оптимального результату рекомендовано поєднувати low-code підхід з традиційними методами розробки у тих випадках, коли необхідна максимальна гнучкість або продуктивність.

Отже, результати дослідження підтверджують доцільність використання low-code платформ, зокрема Mendix, для побудови систем фінансового моделювання у DFI-середовищі. Запропонований підхід забезпечує оптимальний баланс між швидкістю розробки, контрольованістю, прозорістю, комплаєнсом та масштабованістю, що робить його ефективним інструментом для підтримки процесів прийняття управлінських та інвестиційних рішень у сучасних фінансових інституціях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Mendix. What Is Low-Code? Mendix Documentation. URL : https://www.mendix.com/low-code/ (дата звернення : 06.05.2025).

2. Gartner. Magic Quadrant for Enterprise Low-Code Application Platforms. Gartner. URL : https://www.gartner.com/document/readmepage3992030 (дата звернення: 06.05.2025).

3. Forrester. The Forrester Wave: Low-Code Development Platforms. Forrester. URL : https://www.forrester.com/report/Low-Code/overview-table/RES147 (дата звернення: 07.05.2025).

4. IBM. Low-Code Development Explained. IBM Documentation. URL : https://www.ibm.com/topics/low-code/ (дата звернення: 07.05.2025).

5. Microsoft. Introduction to Power Apps Low-Code Platform. Microsoft Learn. URL : https://learn.microsoft.com/en-us/power-apps/apps (дата звернення: 07.05.2025).

6. OutSystems. OutSystems Developer Docs: Architecture Overview. URL : https://www.outsystems.com/blog/posts/architecture-overview/5124244qwert (дата звернення: 08.05.2025).

7. Appian. Low-Code Automation Platform Documentation. Appian Docs. URL : https://docs.appian.com/ (дата звернення: 08.05.2025).

8. Okoone. The Inherent Limitations of Low-Code Platforms. Okoone. URL : https://www.okoone.com/blog/low-code-limitations/231314/resources/read (дата звернення: 08.05.2025).

9. Oracle. What Is Low-Code Development? Oracle Resources. URL : https://www.oracle.com/low-code/ (дата звернення: 09.05.2025).

10. Kissflow. Low-Code Statistics and Trends 2025. Kissflow. URL : https://kissflow.com/low-code/statistics/ (дата звернення: 09.05.2025).

11. Meegle. UX Design for Low-Code and No-Code Platforms. Meegle Blog. URL : https://www.meegle.com/blog/ux-low-code/ (дата звернення: 09.05.2025).

12. TechAhead. User-Centered Design Principles. TechAhead. URL : https://www.techaheadcorp.com/blog/user-centered-design/ (дата звернення: 09.05.2025).

13. Microsoft. UX/UI Design Principles in Modern Applications. Microsoft Learn. URL : https://learn.microsoft.com/en-us/windows/apps/design (дата звернення: 10.05.2025).

14. Nielsen Norman Group. Visual Hierarchy in UX. Nielsen Norman Group. URL : https://www.nngroup.com/articles/visual-hierarchy/ (дата звернення: 10.05.2025).

15. OutSystems. Performance Best Practices. OutSystems Documentation. URL : https://www.outsystems.com/blog/posts/performance/ (дата звернення: 10.05.2025).

16. Mendix. Microflows and Logic Design Overview. Mendix Documentation. URL : https://docs.mendix.com/refguide/microflows/ (дата звернення: 10.05.2025).

17. Mendix. Domain Modeling Best Practices. Mendix Documentation. URL : https://docs.mendix.com/refguide/domain-model/ (дата звернення: 11.05.2025).

18. Mendix. Marketplace Module Catalogue. Mendix Marketplace. URL : https://marketplace.mendix.com/ (дата звернення: 11.05.2025).

19. Fujitsu. Why UX Still Matters in Low-Code Development. Fujitsu Benelux. URL : https://www.fujitsu.com/nl/low-code-ux/documentation-cursor-data (дата звернення: 11.05.2025).

20. Oracle. Low-Code Prototyping Methodologies. Oracle Blog. URL : https://blogs.oracle.com/low-code/prototyping (дата звернення: 12.05.2025).

21. Corporate Finance Institute (CFI). Scenario Analysis. CFI. URL : https://corporatefinanceinstitute.com/resources/valuation/scenario-covert (дата звернення: 12.05.2025).

22. Corporate Finance Institute (CFI). 3-Statement Financial Model Guide. CFI. URL : https://corporatefinanceinstitute.com/resources/valuation/3-statemen/ (дата звернення: 12.05.2025).

23. Corporate Finance Institute (CFI). Discounted Cash Flow (DCF) Overview. CFI. URL : https://corporatefinanceinstitute.com/resources/valuation/dcf-overview (дата звернення: 13.05.2025).

24. Mergers & Inquisitions. Financial Modeling Explained. Mergers & Inquisitions. URL : https://www.mergersandinquisitions.com/financial-modeling/ (дата звернення: 13.05.2025).

25. Investopedia. Comparable Company Analysis (Comps). Investopedia. URL : https://www.investopedia.com/terms/c/comps.asp/low-code/analytics/4214324 (дата звернення: 13.05.2025).

26. Investopedia. Precedent Transactions Analysis. Investopedia. URL : https://www.investopedia.com/terms/p/precedent-transaction-analysis.asp/sap-data (дата звернення: 14.05.2025).

27. Investopedia. Leveraged Buyout (LBO). Investopedia. URL : https://www.investopedia.com/terms/l/lbo.asp (дата звернення: 14.05.2025).

28. Corporate Finance Institute (CFI). IPO Financial Modeling Overview. CFI. URL : https://corporatefinanceinstitute.com/resources/valuation/ipo-model/overview (дата звернення: 14.05.2025).

29. Oracle. Financial Modeling Techniques. Oracle Finance Learning. URL : https://www.oracle.com/finance/modeling/financial-modeling/low-code (дата звернення: 15.05.2025).

30. Deloitte. Data Governance for Financial Institutions. Deloitte Insights. URL : https://www.deloitte.com/data-governance/compliance-finance/finance-data (дата звернення: 15.05.2025).

31. Basel Committee on Banking Supervision (BCBS). Basel Committee Principles for Risk Data Aggregation (BCBS 239). Bank for International Settlements. URL : https://www.bis.org/public-data/bcbs239.html (дата звернення: 15.05.2025).

32. Federal Deposit Insurance Corporation (FDIC). Model Governance Guidelines. URL : https://www.fdic.gov/regulations/examinations/model-governance/2357323232 (дата звернення: 16.05.2025).

33. OECD. Development Finance Standards. OECD Library. URL : https://www.oecd.org/development/ (дата звернення: 16.05.2025).

34. World Bank. Development Finance Assessment Principles. World Bank. URL : https://www.worldbank.org/en/topic/financialsector/corefile1 (дата звернення: 16.05.2025).

35. FMO. Annual Report 2024 – Risk & Portfolio Management. FMO. URL : https://www.fmo.nl/annual-report (дата звернення: 16.05.2025).

36. FMO. Mobilizing Finance for Development Overview. FMO. URL : https://www.fmo.nl/mobilizing-finance (дата звернення: 17.05.2025).

37. PwC. Stress Testing in Financial Institutions. PwC Publications. URL : https://www.pwc.com/stress-testing (дата звернення: 17.05.2025).

38. EY. Data Quality Management in Finance. EY Insights. URL : https://www.ey.com/data-quality (дата звернення: 17.05.2025).

39. IBM. Data Integrity and Governance Overview. IBM Knowledge Center. URL : https://www.ibm.com/data-governance/ (дата звернення: 18.05.2025).

40. AWS. Financial Services Data Architecture. AWS Documentation. URL : https://aws.amazon.com/financial-services/readme (дата звернення: 18.05.2025).

41. AWS. Amazon RDS Documentation. AWS Documentation. URL : https://docs.aws.amazon.com/rds/ (дата звернення: 18.05.2025).

42. Google. Data Modeling Best Practices. Google Cloud Architecture. URL : https://cloud.google.com/architecture/data-modeling (дата звернення: 19.05.2025).

43. Microsoft. Enterprise Integration Patterns. Microsoft Learn (Azure Architecture Center). URL : https://learn.microsoft.com/azure/architecture/patterns/ (дата звернення: 19.05.2025).

44. Nielsen Norman Group. UX for Data-Intensive Applications. Nielsen Norman Group. URL : https://www.nngroup.com/articles/data-intensive-ux/rootversion1 (дата звернення: 19.05.2025).

45. Mendix. Performance Optimization for Microflows. Mendix Documentation. URL : https://docs.mendix.com/refguide/performance/ (дата звернення: 20.05.2025).

46. Mendix. Security & Access Management. Mendix Documentation. URL : https://docs.mendix.com/refguide/security/ (дата звернення: 20.05.2025).

47. Mendix. Audit Trail Module. Marketplace Documentation. URL : https://marketplace.mendix.com/link/component/108107 (дата звернення: 20.05.2025).

48. Jabil. How Jabil Scales Manufacturing with Mendix. Jabil Case Study. URL : https://www.jabil.com/mendix-case-study.html (дата звернення: 21.05.2025).

49. Mendix. Lever Style. Low-Code Transformation Success Story. Mendix Case Study. URL : https://www.mendix.com/customers/lever-style/data (дата звернення: 21.05.2025).