Зміст

[ВСТУП 7](#_Toc483754573)

[1 ЗАГАЛЬНА ЧАСТИНА 9](#_Toc483754574)

[1.1 Опис завдання 9](#_Toc483754575)

[1.2 Предметна область і напрямки дослідження 10](#_Toc483754576)

[1.3 Обґрунтування оптимального варіанта реалізації мети бакалаврської роботи 14](#_Toc483754577)

[1.3.1 «Finance.ua» — Перший фінансовий портал України 14](#_Toc483754578)

[1.3.2 Програма " Profit " 15](#_Toc483754579)

[1.4 Опис алгоритму та використаного програмного забезпечення 17](#_Toc483754580)

[1.4.1 Середовище розробки програмного продукту Visual Studio 2013 17](#_Toc483754581)

[1.4.2 Мова програмування додатку 17](#_Toc483754582)

[1.4.3 Мова розробки бази даних для автоматизованої інформаційної системи 18](#_Toc483754583)

[1.4.4 Сервер бази даних 21](#_Toc483754584)

[1.4.5 Бібліотеки використані для взаємодії 23](#_Toc483754585)

[1.4.6 Технологія WCF для абстрагування від технології додатку-клієнта 25](#_Toc483754586)

[1.4.7 Структура базового проекту WCF 27](#_Toc483754587)

[1.5 Основні рішення з реалізації системи в цілому і її компонентів 29](#_Toc483754588)

[1.6 Система контролю версій Git 30](#_Toc483754589)

[1.7 Веб сервіс GitHub 32](#_Toc483754590)

[ВИСНОВОК ДО РОЗДІЛУ 1 33](#_Toc483754591)

[2 СПЕЦІАЛЬНА ЧАСТИНА 34](#_Toc483754592)

[2.1 Постановка задачі 34](#_Toc483754593)

[2.2 Коротка характеристика засобів проектування 34](#_Toc483754594)

[2.3 Модель досліджуваної системи 36](#_Toc483754595)

[2.3.1 Опис бізнес–варіантів використання 36](#_Toc483754596)

[2.3.2 Діаграми класів системи 37](#_Toc483754597)

[2.4 Інструкція роботи користувача з системою 38](#_Toc483754598)

[2.4.1 Інструкція роботи адміністратора з додатком. 38](#_Toc483754599)

[2.4.2 Інструкція роботи клієнта банку із додатком. 45](#_Toc483754600)

[ВИСНОВОК ДО РОЗДІЛУ 2 47](#_Toc483754601)

[ВИСНОВОК 48](#_Toc483754602)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 50](#_Toc483754603)

[Додаток А 52](#_Toc483754604)

[Додаток Б 53](#_Toc483754605)

[Додаток В. Сервіс 54](#_Toc483754606)

[Додаток Г. Клієнт 55](#_Toc483754607)

[Додаток Д. Лістинг програми 56](#_Toc483754608)

# **ВСТУП**

В останні роки банківська система нашої країни переживає бурхливий розвиток. Не зважаючи на існуючі недоліки українського законодавства, що регулює діяльність банків, ситуація неухильно змінюється на краще. Пройшли часи, коли можна було легко заробляти на спекулятивних операціях з валютою. Сьогодні все більше банків роблять ставку на професійність своїх співробітників і нові технології.

Важко уявити собі більш сприятливий ґрунт для запровадження комп’ютерних технологій, ніж банківська діяльність. Майже всі завдання, які виникають у процесі роботи банку, піддаються автоматизації. Швидка і безперебійна обробка значних потоків інформації є одним із головних завдань будь-якої великої фінансової організації. Тому необхідна наявність обчислювальної мережі, яка дозволяє обробляти інформаційні потоки, що все збільшуються і збільшуються. Крім того, банки мають достатні фінансові можливості для використання найсучаснішої техніки. Однак не слід вважати, що середній банк готовий витрачати величезні суми на комп’ютеризацію. Банк є, насамперед, фінансовою організацією, яка призначена для отримання прибутку, тому затрати на модернізацію повинні бути співставленими з очікуваною користю від її проведення. Відповідно до загальносвітової практики, в середньому затрати банку на комп’ютеризацію складають не менше 17 % від загальної суми річних витрат.

Інтерес до розвитку комп’ютеризованих банківських систем визначається, головним чином, стратегічними інтересами. Як показує практика, інвестиції в такі проекти починають приносити прибуток лише через певний період часу, необхідний для навчання персоналу та адаптації системи до конкретних умов. Вкладаючи кошти в програмне забезпечення, комп’ютерне і телекомунікаційне обладнання та створення бази для переходу до нових обчислювальних платформ, банки, в першу чергу, прагнуть до здешевлення і прискорення рутинної роботи та перемоги в конкурентній боротьбі.

Нові технології допомагають банкам, інвестиційним фірмам та страховим компаніям змінити відносини з клієнтами і знайти нові засоби для отримання прибутку. Аналітики сходяться на думці, що нові технології найбільш активно впроваджують інвестиційні фірми, потім йдуть банки, а останніми їх приймають на озброєння страхові компанії.

Завдання, що стоїть перед фінансовими організаціями, однакове. Це інтеграція нових систем у розподілену архітектуру локальних та глобальних мереж.

Банківські комп’ютерні системи на сьогоднішній день набули найбільшого розвитку серед галузей прикладного мережевого програмного забезпечення (ПЗ). Потрібно відзначити, що БС - дуже вигідний ринок для будь-якого виробника комп’ютерів та ПЗ.

Як приклад передових технологій, що використовуються в банківській діяльності, можна назвати бази даних на основі моделі “клієнт-сервер” (характерним є використання ОС Unіx та БД Oracle); засоби міжмережевої взаємодії для міжбанківських розрахунків; служби розрахунків, цілком орієнтовані на Іnternet, і багато іншого.

# **ЗАГАЛЬНА ЧАСТИНА**

# **1.1 Опис завдання**

Розвиток комп’ютерних технологій обумовив їх впровадження в усі сфери життя та роботи людини. Можливість нових технологій накопичувати та оброблювати інформацію, вести облік, автоматизувати частину робочого процесу, використовувати їх в управління тією чи іншою діяльністю допомагає працівникам з їх повсякденними робочими справами. Підприємства, фірми, державні установи, різноманітні заклади якомога більше намагаються автоматизувати певні ділянки роботи, аби йти в ногу з часом та підвищувати ефективність.

В Україні автоматизація банківських технологій в нових ринкових умовах почала проводитися з початку 1990-х років, коли з’явилися перші комерційні банки. Відразу ж на ринку програмного забезпечення для банківської справи засобів з’явилося ряд зарубіжних та вітчизняних фірм-розробників, що його заповнили. Розвиток процесу автоматизації обумовив появу різноманітних автоматизованих банківських систем, що пояснюється не тільки великою кількістю фірм-розробників, але і широким спектром функцій, які банки виконують, різноманітністю структур і розмірів банківської справи.

Розроблений додаток повинен забезпечити функціонал, що зможе задовольнити потреби клієнта комерційного банку, тому програма має містити:

* Необхідні форми та інтерфейси для користувача.
* Зручні форми з точки зору користування не професіоналів в області комп’ютерних наук.
* Простий, в той же час багатофункціональний та зрозуміли дизайн.
* Системи автоматизованої взаємодії з клієнтами (так звані системи “банк-клієнт”).
* Аналітичні системи. Аналіз всієї діяльності банку і системи вибору оптимальних в даній ситуації рішень.
* Системи міжбанківських розрахунків.
* Інформаційні системи. Можливість миттєвого отримання необхідної інформації, що впливає на фінансову ситуацію.

Отже, автоматизована інформаційна система, яка розробляється має бути надійною, зрозумілою та придатною для щоденного користування представникам споживчого товариства.

# **1.2 Предметна область і напрямки дослідження**

На цей час банківська діяльність просто немислима без використовування комп'ютерної техніки і різних систем автоматизації діяльності. Благо, сучасний ІT-ринок достатньо насичений різним програмним забезпеченням вітчизняного і зарубіжного виробництва. Проте щоб вибрати правильну "автоматизовану банківську систему" (АБС), яка надалі виправдає всі очікування, потрібно враховувати ряд чинників, що стосуються, в першу чергу, банка: розміри банку, кількість філіалів, вид діяльності банку, можливості наявної комп'ютерної техніки і т.п. Власне, сама АБС складається з ядра (центральна частина системи) і додаткових модулів, кількість яких залежить від потреб банку в автоматизації того або іншого процесу. Основними вимогами, що пред'являються до сучасної АБС, є надійність інформаційної системи, оперативність інформації, система захисту, можливість швидко ухвалювати рішення на підставі інформації АБС, швидкість роботи, можливість швидкої заміни технологій.

Класифікація АБС:

І покоління

Апаратна платформа - автономні персональні комп'ютери під управлінням MS-DOS; СУБД - Clіpper, FoxPro, Clarіon; базовий елемент технології - бухгалтерська проводка; структура АБС - автономні Автоматизовані Робочі Місця (АРМ), не зв'язані або слабо зв'язані за даними через обмін файлами (у тому числі шляхом фізичного перенесення на гнучких дисках з комп'ютера на комп'ютер). Зараз практично не зустрічається.

ІІ покоління

Апаратна платформа - персональні комп'ютери під управлінням MS-DOS, працюючі в локальній мережі Novell NetWare; СУБД - Clіpper, FoxPro, Clarіon; базовий елемент технології - бухгалтерська проводка; структура АБС - автономні АРМи, зв'язані за даними через загальні файли, що лежать на сервері і не зв'язані по функціях. Широко поширено дотепер, особливо в невеликих банках і в провінції.

ІІІ покоління

Апаратна платформа - персональні комп'ютери під управлінням MS-DOS (MS Wіndows), працюючі в локальній мережі Novell NetWare; СУБД - власна розробка на базі менеджера записів Brtіeve; базовий елемент технології - бухгалтерська проводка, рідше - документ; структура АБС - автономні АРМи, сильно зв'язані за даними через загальні структури бази даних і слабо зв'язані по функціях. Технологія, перехідна від “файл-сервер до “клієнт-сервер. Широко поширено, у тому числі у ряді крупних банків.

ІV покоління

Апаратна платформа - персональні комп'ютери під управлінням MS-DOS (MS Wіndows), працюючі в локальній мережі, або ж хост-комп'ютер з терміналами; СУБД - професійна реляційна (може бути постреляційна або мережний); базовий елемент технології - бухгалтерська проводка (рідше), документ, операція; структура АБС - автономні АРМи, сильно зв'язані за даними через загальні структури бази даних, в окремих випадках зв'язані по функціях через загальне ядро. Технологія “хост-термінал або дворівнева “клієнт-сервер. Досить поширено, але не у нас.

V покоління

Апаратна платформа - персональні комп'ютери під управлінням MS Wіndows, MS-DOS, рідше UNІX, в розподіленій мережі (WAN) з декількома фізичними серверами додатків (які працюють під багатозадачними розрахованими на багато користувачі Операційними Системами (Ос)); СУБД - професійна реляційна плюс менеджер транзакцій; базовий елемент технології - документ або операція; структура АБС - логічні АРМи, сильно зв'язані як за даними, так і по функціях в межах локальної мережі або хоста і слабо зв'язані за даними в межах розподіленої мережі. Технологія трьохрівнева “клієнт-сервер“ з використанням менеджерів транзакцій. Одиничні розробки.

VІ покоління

Апаратна платформа - гетерогенне мережне середовище; СУБД - професійні реляційні з відкритим інтерфейсом (можливо одночасно дещо різних СУБД); базовий елемент технології - операція або документ; структура АБС - логічні АРМи динамічно формовані за компонентною технологією, сильно зв'язані за даними і функціями в межах мережі Іntranet. Перспективна технологія, що з'явилася трохи більше року назад. Одиничні розробки, але за ними майбутнє.

Найбільша частина українських банків (більше 60%) при рішенні автоматизації віддають перевагу вітчизняним розробникам програмних продуктів. Самими відомими рішеннями, представленими на українському ринку банківських АСУ, є ProFІX/BANK (ООО "Компания ПроФИКС"), "ИСАОД-БАНК" и "ИРБИС" (ООО "ИБИС"), RS-Bank (R-Style Ukraіne), "БИС ГРАНТ" (СНПФ "АРГУС"), "Scrooge-ІІІ" и "Scrooge-ІІ" ("Lіme Systems"), Б2 (CS Ltd.), SftBank (ООО СП "40.5 Plus"), "UnіCorn" (ООО "Юникорн") та ін.

Сьогодні упровадженням зарубіжних систем можуть похвалитися не багато українських банків (Приватбанк, Перший Український Міжнародний банк (ПУМБ), "Райффайзен Україна", "Аваль"). Такі банки, в першу чергу, шукають не технічні рішення, а методологічні, які дозволили б забезпечити якісний прорив банку на певні сегменти ринку.

Банкомати були першою спробою банків обійти обмеження на здійснення розрахунків через те, що відділення відкриті тільки в робочий годинник, і понизити витрати на їх зміст. Потім з'явилися послуги по телефону. Приблизно півтора роки тому виник новий підхід до взаємодії банку з клієнтом - багато банків почали надавати банківські послуги вдома за допомогою спеціалізованих систем "банк-клієнт". Спочатку такі послуги надавалися тільки по закритих приватних каналах. В даний час ситуація міняється у бік використовування Іnternet. Зараз найбільш популярні змішані рішення.

Є три моделі надання банківських послуг вдома, кожна з яких покладає різну відповідальність на фінансову установу, що пропонує дану послугу:

* Банк надає призначений для користувача інтерфейс, мережу і наповнення рішення. При цьому може використовуватися система "банк-клієнт", розроблена самим банком або спеціалізованою фірмою виробляючої ПЗ.
* Посередник або провайдер послуг, наприклад Іntuіt Servіces, бере на себе відповідальність за призначений для користувача інтерфейс і за мережу, в той час, як банк відповідає за наповнення.
* Надання послуг вдома за допомогою Іnternet. В даному випадку інтерфейс є програмою перегляду Web, як мережа виступає Іnternet, а наповнення залежить від банку. Взагалі кажучи, через вузол Web фінансові інститути можуть запропонувати широкий спектр послуг.

Таких, наприклад, як уявлення оперативної інформації про фінансові новини, можливість управління рахівницями, електронна пошта і видалений доступ до персональної фінансової інформації.

Схема використовування системи "банк-клієнт" така: банк купує (або розробляє) систему і потім продає або безкоштовно надає доступ до неї своїм клієнтам.

З погляду реалізації фінансових послуг для банків система "банк-клієнт" не є нічим принципово новим, основні зміни торкаються організаційної сфери діяльності. Система "банк-клієнт" дозволяє всього лише виключити з технологічного ланцюжка обробки фінансового документа процедуру передачі паперового оригіналу з рук клієнта в руки операціоніста і перекладу його в електронну форму. Супутні цьому процесу операції ідентифікації і аутентифікації документа теж виконуються автоматично. Надалі документ в електронному вигляді проходить абсолютно ті ж етапи обробки, передбачені існуючою банківською технологією, що і паперовий документ.

Обмеження обумовлені лише готовністю банку довірити виконання фінансових операцій автоматизованій системі.

# **1.3 Обґрунтування оптимального варіанта реалізації мети бакалаврської роботи**

Актуальність теми дипломної роботи полягає в в необхідності розробки автоматизованої системи обробки банківської інформації, оскільки на сьогоднішній день наступає новий етап автоматичних систем банківського самобслуговування, в якому персонал банку все менше приймає участь у банківських транзакціях, а паперова інформація банківських платіжних документів все більше заміщується системою кодованих комп’ютерних файлів, що є єдиними носіями інформації про здійснені банківські операції.

## **1.3.1 «Finance.ua» — Перший фінансовий портал України**

Основні напрямки діяльності порталу:

* агрегація економічних і фінансових новин
* агрегація курсів валют
* сервіс обміну валют «Міняйло»
* сервіс онлайн бухгалтерії «Домашні фінанси»
* великий фінансовий форум

До недоліків даного порталу можна віднести неможливість проведення розрахунку депозитів та кредитів комерційних банків, які мають спеціальні умови для надання даних послуг.

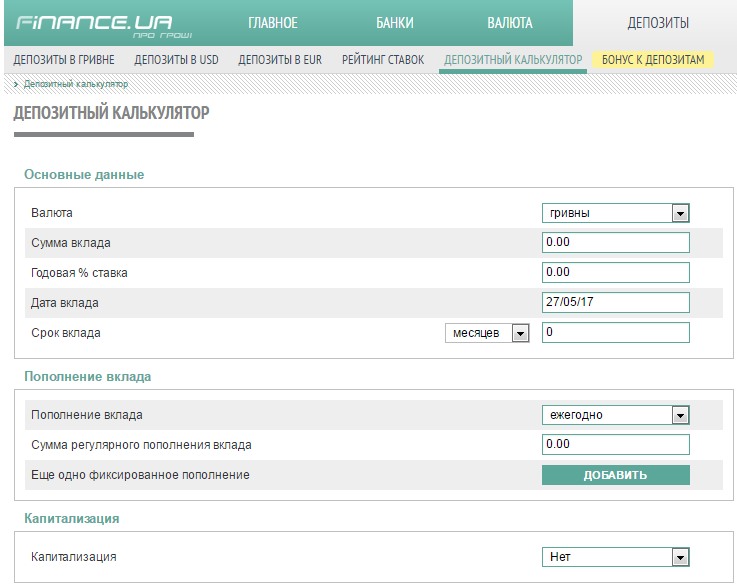


Рис. 1.1 Інтерфейс порталу «Finance.ua»

## **1.3.2 Програма " Profit "**

Депозитний калькулятор "Profit" - програма призначена для обчислення результату розміщення грошових коштів на депозитному рахунку (внесок в банку), формування виписки з рахунку. Отримані дані можуть використовуватися з метою:

* оцінки ефективності розміщення грошових коштів у внеску;
* порівняння і вибору найкращої пропозиції серед вкладів;
* складання накопичувальної програми, особистого фінансового плану.

Дана програма може бути корисна особам, які беруть рішення відкрити вклад в банку, або передати грошові кошти в борг під відсотки, фінансовим працівникам.

Програма призначена для обчислення результату розміщення грошових коштів на депозитному рахунку банку із зазначенням наступних умов:

* валюта вкладу (рублі, долари, євро);
* початкова сума вкладу;
* період і розмір поповнень вкладу;
* величина ставки за вкладом;
* період нарахування відсотків;
* можливість капіталізації нарахованих відсотків.

Для заданих умов вкладу програма обчислює:

* сума нарахованих відсотків;
* сума податку;
* чистий прибуток;
* підсумкова сума заощаджень.

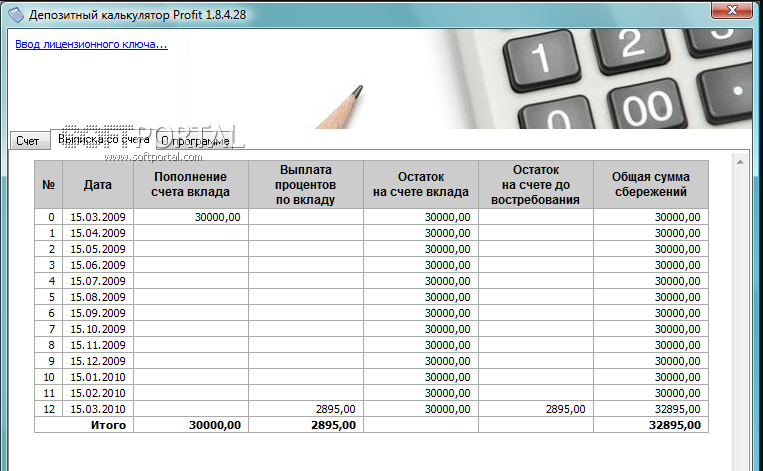


Рис. 1.2 Депозитний калькулятор "Profit"

До недоліків даного програмного забезпечення можна віднести неможливість проведення розрахунку виплат по кредиту.

## **1.4 Опис алгоритму та використаного програмного забезпечення**

## **1.4.1 Середовище розробки програмного продукту Visual Studio 2017**

## Microsoft Visual Studio - це набір інструментів для створення програмного забезпечення: від планування до розробки призначеного для користувача інтерфейсу, написання коду, тестування, налагодження, аналізу якості коду і продуктивності, розгортання в середовищах клієнтів і збору даних телеметрії по використанню. Ці інструменти призначені для максимально ефективної спільної роботи; всі вони доступні в інтегрованому середовищі розробки (IDE) Visual Studio.

## Visual Studio можна використовувати для створення різних типів додатків, від простих додатків для магазину та ігор для мобільних клієнтів до великих і складних систем, які обслуговують підприємства та центри обробки даних. Ви можете створювати:

* додатки та ігри, які виконуються не тільки на платформі Windows, але і на Android і iOS;
* веб-сайти і веб-служби на основі ASP.NET, JQuery, AngularJS і інших популярних платформ;
* додатки для самих різних платформ і пристроїв, включаючи, але не обмежуючись: Office, Sharepoint, Hololens, Kinect і "Інтернету речей";
* ігри і графічні додатки для різних пристроїв Windows, включаючи Xbox, з підтримкою DirectX.

За замовчуванням Visual Studio забезпечує підтримку C #, C і C ++, JavaScript, F # і Visual Basic. Visual Studio добре працює і інтегрується зі сторонніми додатками, наприклад Unity і Apache Cordova, за допомогою розширень і інструментів Visual Studio для Apache Cordova відповідно. Ви також можете самостійно розширити Visual Studio, створивши власні інструменти для виконання спеціалізованих завдань.

Інструменти моделювання та архітектури Visual Studio можна використовувати для розробки і моделювання додатки. Вони допомагають візуалізувати структуру, поведінку і зв'язку коду. В процесі розробки можна створювати моделі з різним рівнем деталізації протягом життєвого циклу програми. Можна відстежувати вимоги, завдання, тестові випадки, помилки або інші типи робіт, пов'язані з моделями, зв'язавши елементи моделі з робочими елементами Team Foundation Server і планом розробки.

## **1.4.2 Мова програмування додатку**

C# — [об'єктно–орієнтована](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%E2%80%99%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [мова програмування](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) з безпечною [системою типізації](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%97) для платформи[.NET](https://uk.wikipedia.org/wiki/.NET).

На сьогоднішній момент мова програмування C# одна з найпотужніших, що швидко розвиваються і затребуваних в ІТ-галузі. На даний момент на ньому пишуться найрізноманітніші програми: від невеликих десктопних програмок до великих веб-порталів і веб-сервісів, які обслуговують щодня мільйони користувачів.

У порівнянні з іншими мовами C# досить молодий, але в той же час він вже пройшов великий шлях. Перша версія мови вийшла разом з релізом Microsoft Visual Studio .NET в лютому 2002 року. Поточною версією мови є версія C# 7.0, яка вийшла в 7 березня 2017 року разом з Visual Studio 2017.

C# є мовою з Сі-подібним синтаксисом і близький в цьому відношенні до C ++ і Java. Тому, якщо ви знайомі з одним з цих мов, то опанувати C# буде легше.

C# є об'єктно-орієнтованим і в цьому плані багато перейняв у Java і С ++. Наприклад, C# підтримує поліморфізм, успадкування, перевантаження операторів, статичну типізацію. Об'єктно-орієнтований підхід дозволяє вирішити завдання з побудови великих, але в той же час гнучких, масштабованих і розширюваних додатків. І C# продовжує активно розвиватися, і з кожною новою версією з'являється все більше цікавих функциональностей, як, наприклад, лямбда, динамічне зв'язування, асинхронні методи і т.д.

Коли говорять C#, нерідко мають на увазі технології платформи .NET (WPF, ASP.NET). І, навпаки, коли говорять .NET, нерідко мають на увазі C#. Однак, хоча ці поняття пов'язані, ототожнювати їх невірно. Мова C# був створений спеціально для роботи з фреймворком .NET, проте саме поняття .NET дещо ширше.

Фреймворк .NET представляє потужну платформу для створення додатків. Можна виділити наступні її основні риси:

* Підтримка декількох мов. Основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), завдяки чому .NET підтримує кілька мов: поряд з C# це також VB.NET, C ++, F #, а також різні діалекти інших мов, прив'язані до .NET, наприклад, Delphi. NET. При компіляції код на будь-якому з цих мов компілюється в збірку спільною мовою CIL (Common Intermediate Language) - свого роду асемблер платформи .NET. Тому ми можемо зробити окремі модулі однієї програми на окремих мовах.
* Кросплатформеність. Остання версія платформи на даний момент підтримується на більшості сучасних ОС Windows (Windows 10 / 8.1 / 8/7 / Vista). А завдяки проекту Mono можна створювати додатки, які будуть працювати і на інших ОС сімейства Linux, в тому числі на мобільних платформах Android і iOS.
* Потужна бібліотека класів. .NET представляє єдину для всіх підтримуваних мов бібліотеку класів. І яке б додаток ми не збиралися писати на C# - текстовий редактор, чат або складний веб-сайт - так чи інакше ми задіємо бібліотеку класів .NET.
* Різноманітність технологій. Загальномовне середовище виконання CLR і базова бібліотека класів є основою для цілого стека технологій, які розробники можуть задіяти при побудові тих чи інших додатків. Наприклад, для роботи з базами даних в цьому стеку технологій призначена технологія ADO.NET. Для побудови графічних додатків з багатим насиченим інтерфейсом - технологія WPF. Для створення веб-сайтів - ASP.NET і т.д.
* Також ще слід відзначити таку особливість мови C# і фреймворка .NET, як автоматичне прибирання сміття. А це означає, що нам в більшості випадків не доведеться, на відміну від С ++, піклуватися про звільнення пам'яті. Вищезазначена загальномовного середовища CLR сама викличе збирач сміття і очистить пам'ять.

**1.4.3 Мова розробки бази даних для автоматизованої інформаційної системи**

Для створення бази даних для обліку товарів споживчого товариства було використано мову T–SQL (Transact–SQL) – процедурне розширення мови SQL, створене компанією Microsoft (для Microsoft SQL Server) і Sybase (для Sybase ASE).

SQL був розширений такими додатковими можливостями як:

* керуючі оператори,
* локальні і глобальні змінні,
* різні додаткові функції для обробки рядків, дат, математики і т. п.,
* підтримка аутентифікації Microsoft Windows.

Мова Transact–SQL є ключем до використання MS SQL Server. Всі додатки, які взаємодіють з екземпляром MS SQL Server, незалежно від їх реалізації і призначеного для користувача інтерфейсу, відправляють сервера інструкції Transact–SQL.

У Transact–SQL існують спеціальні команди, які дозволяють керувати потоком виконання сценарію, перериваючи його або направляючи в потрібну логіку.

Блок угруповання – структура, яка об'єднує список виразів в один логічний блок (BEGIN ... END).

Блок умови – структура, яка перевіряє виконання певної умови (IF ... ELSE).

Блок циклу – структура, що організує повторення виконання логічного блоку (WHILE ... BREAK ... CONTINUE).

Перехід – команда, яка виконує перехід потоку виконання сценарію на вказану мітку (GOTO).

Затримка – команда, що затримує виконання сценарію (WAITFOR)

Виклик помилки – команда, яка генерує помилку виконання сценарію (RAISERROR).

## **1.4.4 Сервер бази даних**

Microsoft SQL Server — комерційна [система керування базами даних](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), що розповсюджується корпорацією [Microsoft](https://uk.wikipedia.org/wiki/Microsoft).

Використовується як для невеликих і середніх за розміром баз даних, так і для великих баз даних масштабу підприємства. Багато років вдало конкурує з іншими системами керування базами даних.

Microsoft SQL Server як мову запитів використовує версію SQL, що отримала назву TRANSACT–SQL (скорочено T–SQL), яка є реалізацією SQL–92 (стандарт ISO для SQL) з багатьма розширеннями. T–SQL дозволяє використовувати додатковий синтаксис процедур, що зберігаються і забезпечує підтримку транзакцій (взаємодія бази даних з керуючим застосунком). Microsoft SQL Server та Sybase ASE для взаємодії з мережею використовують протокол рівня застосунка під назвою Tabular Data Stream (TDS, протокол передачі табличних даних).

Microsoft SQL Server також підтримує Open Database Connectivity (ODBC) — інтерфейс взаємодії застосунків з СУБД. Версія SQL Server 2005 надає можливість підключення користувачів через веб–сервер–сервіси, що використовують протокол SOAP. Це дозволяє клієнтським програмам, не призначеним для Windows, кроссплатформенно з'єднуватися з SQL Server. Microsoft також випустила сертифікований драйвер JDBC, що дозволяє застосункам під керування Java (таким як BEA і IBM Websphere) з'єднуватися з Microsoft SQL Server 2000 і 2005.

SQL Server підтримує дзеркалювання та кластеризацію баз даних. Кластер серверу SQL — це сукупність однаково конфігурованих серверів; така схема допомагає розподілити робоче навантаження між декількома серверами. Усі сервери мають одне віртуальне ім'я, а дані розподіляються за IP–адресами машин кластеру протягом робочого циклу. Також у разі відмови або збою на одному з серверів кластеру доступне автоматичне перенесення навантаження на інший сервер.

SQL Server підтримує надлишкове дублювання даних за трьома сценаріями:

* Знімок: виконується «знімок» бази даних, який сервер відправляє одержувачам.
* Історія змін: всі зміни бази даних безперервно передаються користувачам.
* Синхронізація з іншими серверами: бази даних декількох серверів синхронізуються між собою. Зміни усіх баз даних відбуваються незалежно на кожному сервері, а під час синхронізації відбувається звірка даних. Дублювання такого типу передбачає можливість вирішення протиріч між базами даних.

SQL Server 2005 має вбудовану підтримку .NET Framework. Завдяки цьому, процедури бази даних, що зберігаються, можуть бути написані на будь–якій мові платформи .NET з використанням повного набору бібліотек, доступних для .NET Framework. На відміну від інших процесів, .NET Framework виділяє додаткову пам'ять і будує засоби керування SQL Server, не використовуючи вбудовані засоби Windows. Це підвищує продуктивність порівняно із загальними алгоритмами Windows, оскільки алгоритми розподілу ресурсів спеціально налагоджені для використання у структурах SQL Server.

## **1.4.5 Бібліотеки використані для взаємодії**

Entity Framework являється спеціальною об'єктно-орієнтованою технологією на базі фреймворка .NET для роботи з даними. Якщо традиційні засоби ADO.NET дозволяють створювати підключення, команди та інші об'єкти для взаємодії з базами даних, то Entity Framework являє собою більш високий рівень абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працює з об'єктами.

Entity Framework передбачає три можливі способи взаємодії з базою даних:

* Database first: Entity Framework створює набір класів, які відображають модель конкретної бази даних.
* Model first: спочатку розробник створює модель бази даних, по якій потім Entity Framework створює реальну базу даних на сервері.
* Code first: розробник створює клас моделі даних, які будуть зберігатися в бд, а потім Entity Framework за цією моделлю генерує базу даних і її таблиці.

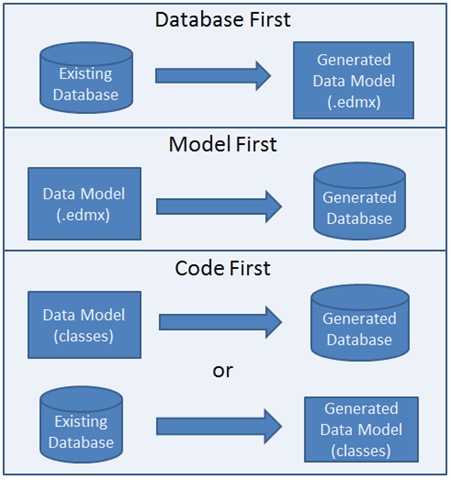


Рис. 1.3 Підхід Code First

Entity Framework забезпечує просте і зручне управління об'єктами з бази даних. В даному випадку нам не треба навіть створювати базу даних і визначати в ній таблиці. Entity Framework все зробить за нас на основі визначення класу контексту даних і класів моделей. І якщо база даних вже є, то EF не буде повторно створювати її.

Наше завдання - тільки визначити модель, яка буде зберігатися в базі даних, і клас контексту. Тому даний підхід називається Code First - спочатку пишеться код, а потім по ньому створюється база даних і її таблиці.

Насамперед створюється новий клас, який буде описувати дані (рис.1.4). Це звичайний клас, який містить певну кількість властивостей. Кожна властивість буде зіставлятися з окремим стовпцем в таблиці з БД (рис.1.5).

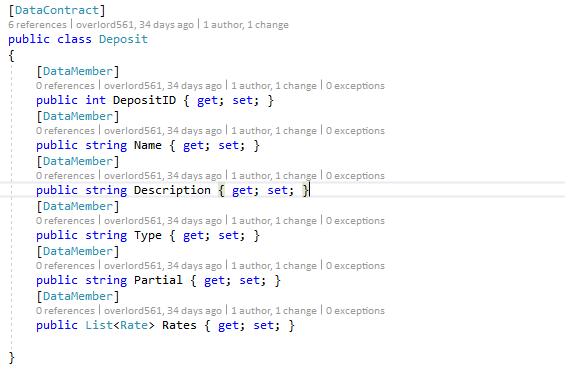


Рис. 1.4 Клас, що описує данні

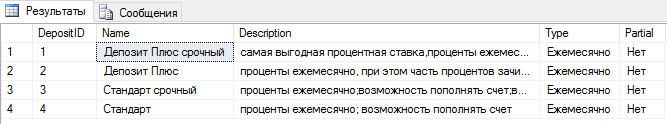


Рис. 1.5 Згенерована та заповнена таблиця в БД

Треба відзначити, що Entity Framework при роботі з Code First вимагає визначення ключа елемента для створення первинного ключа в таблиці в БД. За замовчуванням при генерації БД EF в якості первинних ключів розглядатиме властивості з іменами \*Id.

## **1.4.6 Технологія WCF для абстрагування від технології додатку-клієнта**

WCF (Windows Communication Foundation) - це черговий фреймворк для побудови розподілених додатків і взаємодії між процесами, який є логічним розвитком попередніх подібних технологій компанії Майкрософт, зокрема Веб-сервісів, .Net Remoting і DCOM. І якщо попередники були заточені на виконання якогось конкретного кола завдань, то WCF - технологія, що увібрала в себе все краще від своїх попередників, додавши при цьому, звичайно ж, деяких власних проблем.

Істотною відмінністю WCF від .Net Remoting є те, що WCF - це, перш за все, технологія для побудови сервіс-орієнтованої архітектури додатків (SOA - Service-Oriented Architecture), що дозволяє абстрагуватися від конкретної технологи, на якій цей сервіс реалізований і користуватися ним з інших додатків, написаних на будь-який інший платформі, мовою, технології; головне, щоб реалізація клієнта відповідала певним правилам. Крім того, логіка самого сервісу і його реалізація повністю відокремлена від комунікаційної складової, і ми можемо декларативно змінювати спосіб взаємодії з сервісом шляхом зміни конфігураційного файлу. Ми можемо змінити протокол взаємодії, адреса, налаштувати максимальну кількість підключень, обмежити розмір пакетів і тайм-аут підключення до сервісу, тайм-аут виконання операції, надійність доставки і багато іншого.

«Рівнева архітектура WCF».

Більшість подібних технологій для побудови розподілених додатків (такі як WCF або .Net Remoting) будуються по рівнях, коли кожен шар (рівень) відповідає за свій конкретний рівень абстракції і не знає нічого про нижчі рівні. Якщо говорити коротко, то вся інфраструктура WCF складається з двох головних рівнів: (1) Service Model Layer і (2) Channel Layer (рис. 1.6).

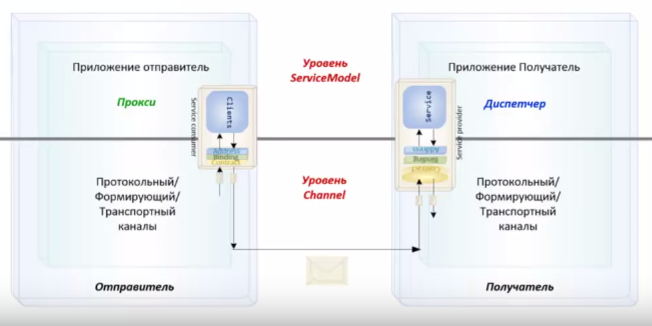


Рис. 1.6 Архітектурні рівні WCF

Перший рівень ближче відноситься до самого сервісу і клієнта і відповідає за перетворення методу і його параметрів в повідомлення для передачі більш низькому канального рівня. Канальний рівень (Channel Layer) інкапсулює в собі канал передачі даних, яких може бути безліч: канали, які використовують в якості транспорту TCP, Http, Named Pipes і т.д. У даному проекті використовується Http канал. Кожен з цих рівнів містить кілька підрівнів, і ви можете вклинитися в кожен з них для якихось власних потреб.

Наприклад, ви можете валідувати параметри методів узагальненим способом: припустимо, вам завжди потрібно, щоб об'єкти певного типу, що використовуються в десятці методів сервісу, володіли певною характеристикою, замість виклику методу валідації з усіх методів сервісу, ви можете написати відповідний «перехоплювач» і обробити це один раз. Ви можете додавати узагальнені канали зв'язку для підтримки, наприклад, UDP, оскільки такого каналу немає в WCF «з коробки» і т.д.

## **1.4.7 Структура базового проекту WCF**

Відкриємо Visual Studio і в меню File (Файл) виберемо пункт New –> Project... . Перед нами відкриється діалогове вікно створення проекту, в якому ми виберемо шаблон WCF Service Application (рис. 1.7):

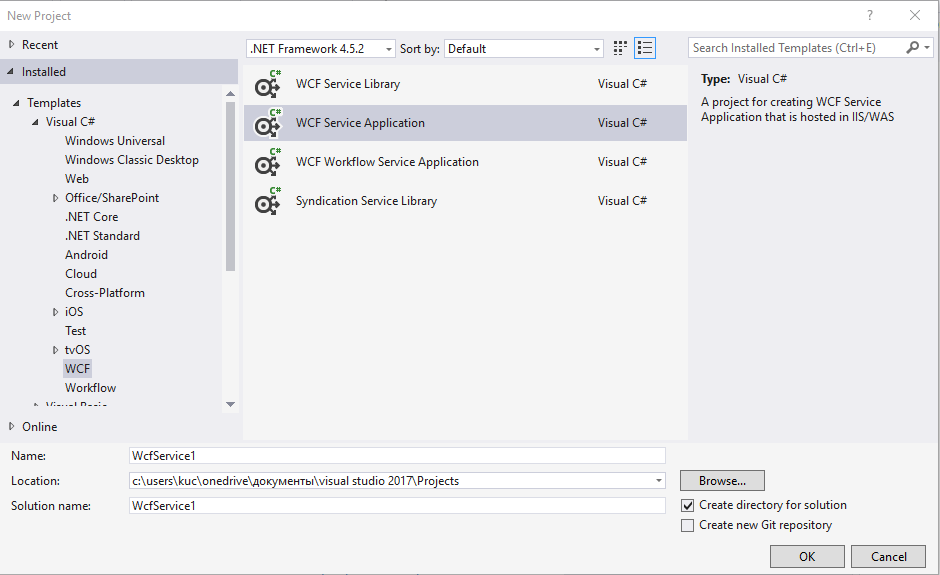


Рис. 1.7 Вибір шаблону розробки

Призначимо проекту яке–небудь ім'я і натиснемо кнопку OK. І Visual Studio створить нам новий проект (рис. 1.8).

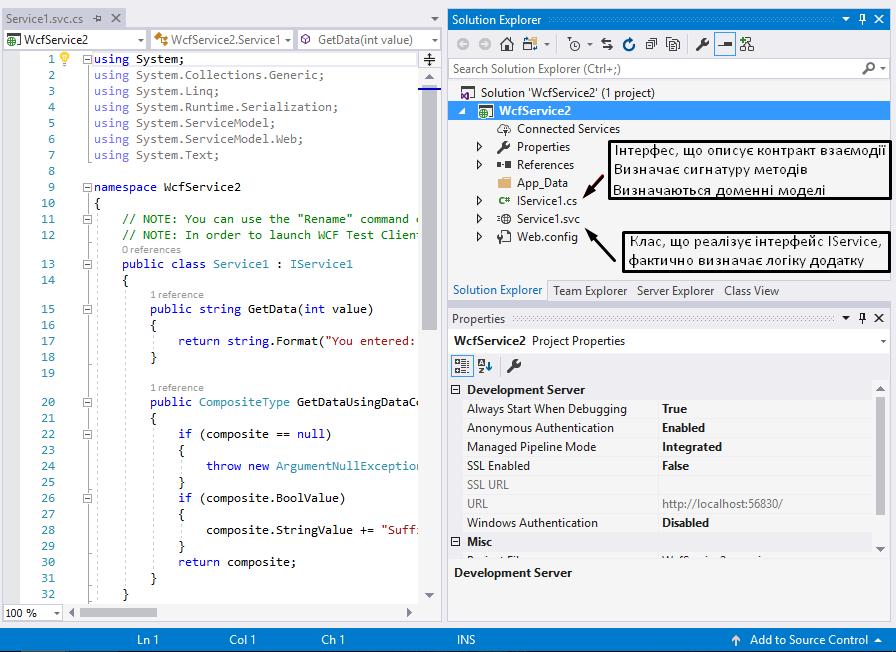


Рис. 1.8 Структура проекту WCF

За замовчуванням студія створює і відкриває нам три основні файли: інтерфейс IService, клас Service, файл конфігурації WebConfig. IService описує контракт взаємодії із додатком-клієнтом. В даному файлі визначена сигнатура методів, описано доменні моделі на основі яких пізніше буде згенеровано сутності та саму базу даних. Service реалізовує інтерфейс ISevice, фактично визначає логіку додатку. WebConfig – файл конфігурації, що визначає підключення до БД, налаштування канального рівня.

Для того, щоб протестувати логіку додатку можна скористатися спеціальною програмою – WCF Test Client (рис. 1.9).

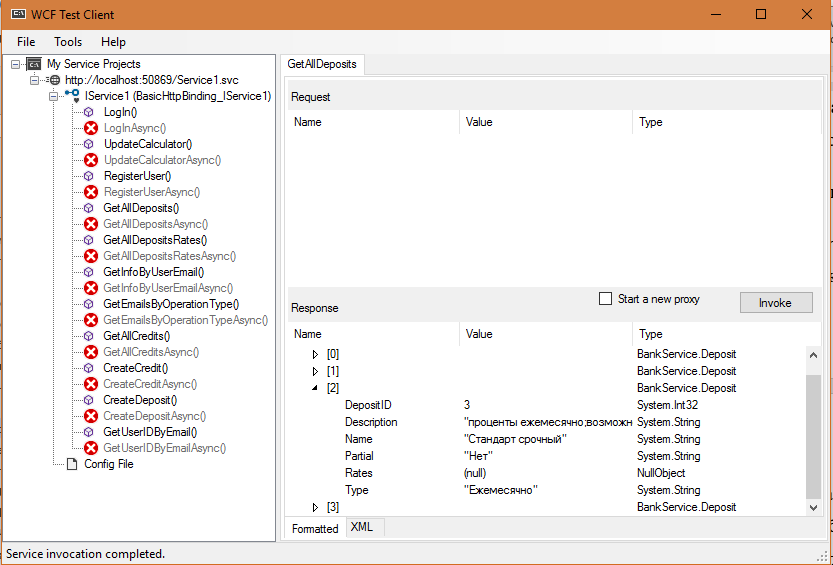


Рис. 1.9 Засіб тестування WCF cлужби

# **1.5 Основні рішення з реалізації системи в цілому і її компонентів**

Реалізація автоматизованої інформаційної системи не можлива без поділу обов’язків та привілегій між користувачами додатку. Тому, автоматизована інформаційна підтримує роботу у двох режимах. Для визначення ролі користувачів їм пропонується пройти авторизацію.

Перший режим – режим адміністратора (працівник банку) бази даних та реалізованого додатку.

До можливостей адміністратора входять:

* Здійснення CRUD операцій щодо депозитів та кредитів – створення, отримання даних, оновлення та видалення.
* Здійснення оповіщення користувачів у разі їх зацікавленості в проведенні депозитних чи кредитних операцій.
* Збір статистики по активності користувачів та передача у відділ роботи із клієнтами.

Другий режим – режим клієнта комерційного банку.

До можливостей клієнта входять:

* Перегляд інформації по депозитах.
* Перегляд інформації по кредитах.
* Прорахунок потенційного прибутку від депозиних операції.
* Визначення умов кредитування.
* Замовлення консультацій по операціях комерційного банку.

Для успішної роботи та хорошої репутації комерційного банку, його продукти повинні відповідати багатьом вимогам, найвижлівішою із яких є безпека персональних даних користувачів. Тому в реалізованому програмному продукті здійснюється шифрування з допомогою алгоритму Rijndael. Платформа .Net надає готовий функціонал по роботі аз даним алгоритмом через клас RijndaelManaged. Дані, що були зашифровані даним алгоритмом представлені на рисунку 1.10.



Рис. 1.10 Зашифровані дані

# **1.6 Система контролю версій Git**

Система контролю версій (CКВ ) – це система, яка реєструє зміни в одному або декількох файлах для того, щоб надалі була можливість повернутися до певних попередніх версій цих файлів.

В Git'і файли можуть знаходитись в одному із трьох станів:

* зафіксованому,
* зміненому,
* підготовленому.

Зафіксований означає, що файл вже збережено у вашій локальній базі. До змінених відносяться файли, які змінилися, але ще не були зафіксовані. Підготовлені файли – це змінені файли, відмічені для включення в наступний коміт.

Таким чином, в проектах, використовучих Git, є три частини: каталог Git'а (Git directory), робочий каталог (working directory) и область підготовлених файлів (staging area) (рис. 1.11).



Рис 1.11 Робочий каталог, область підготовлених файлів, каталог Git'а

Стандартний робочий процес з використанням Git'а виглядає приблизно так:

* Внести зміни в файли в своєму робочому каталозі.
* Підготувати файли, додаючи їх зліпки в область підготовлених файлів за допомогою команди.
* Зробити коміт, який бере підготовлені файли з індексу і поміщає їх в каталог Git'а на постійне зберігання.

Якщо робоча версія файлу збігається з версією в каталозі Git'а, файл вважається зафіксованим. Якщо файл змінений, але доданий в область підготовлених даних, він підготовлений. Якщо ж файл змінився після вивантаження з БД, але не був підготовлений, то він вважається зміненим.

Основні команди Git–a:

* Git add – додає вміст робочої директорії в індекс (staging area ) для подальшого коміта.
* Git status – показує стану файлів в робочій директорії і індексі: які файли змінені, але не додано в індекс; які очікують коммітов в індексі.
* Git commit – бере всі дані, додані в індекс за допомогою git add, і зберігає їх зліпок у внутрішній базі даних, а потім зрушує покажчик поточної гілки на цей зліпок.
* Git push – зберегти зліпок на зовнішній базі (репозиторії) GitHub.

# **1.7 Веб сервіс GitHub**

GitHub – найбільший веб-сервіс для хостингу IT - проектів і їх спільної розробки. Заснований на системі контролю версій Git. Добавивши проект на веб-сервіс, можна прослідкувати історію комітів і повернутись до будь-якої версії проекту, клонувати проект.

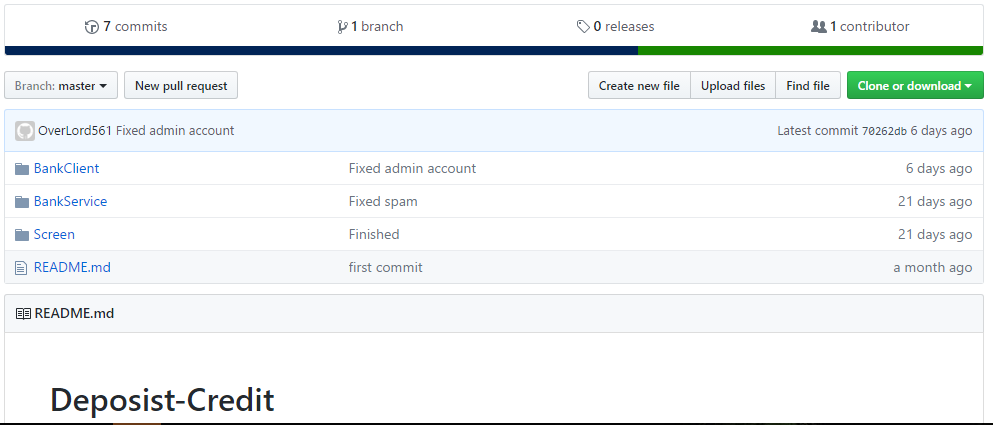


Рис 1.12 Репозиторій GitHub.

# **ВИСНОВОК ДО РОЗДІЛУ 1**

Створення та вибір банківських інформаційних систем базується на плануванні всіх систем, що пов’язані з функціонуванням повної інфраструктури банку, побудованої на інформаційних технологіях. Під інформаційною інфраструктурою автоматизованої банківської системи (АБС) розуміють сукупність, співвідношення і змістовне наповнення окремих складових процесів діяльності банку на основі автоматизації банківських технологій. В інфраструктурі, крім концептуальних складових, виділяють підтримуючі і функціональні системи. До підтримуючих систем відносять інформаційне забезпечення, технічне оснащення, системи зв’язку і комунікацій, програмні засоби, системи безпеки, захисту і надійності. Функціональні системи реалізують банківські послуги, бізнес - процеси та будь-які комплекси задач з сфери змістовної чи предметної спрямованості банківської діяльності. Крім цього створення автоматизованих банківських технологій поряд із загальносистемними (системотехнічними) принципами вимагає врахування особливостей структури, специфіки і обсягів банківської діяльності. Характерною особливістю банківських технологій є різноманітність і складність видів забезпечення автоматизації діяльності банку.

Основні функції реалізованого додатку – аналіз та розрахунок кредитних та депозитних операцій, забезпечення безпеки персональних даних, обробка звернень клієнтів.

В ході виконання роботи було здійснено аналіз продуктів схожих за своїми функціями, але більшість з них є платними та потребують відповідної кваліфікації від користувача.

Метою створення автоматизованої інформаційної системи обробки банківських операцій було забезпечення банківських установ та їх клієнтів захищеної від несанкціонованого доступу, надійною, високошвидкісною і цілодобово працюючою системою для обробки банківських операцій.

# **2 СПЕЦІАЛЬНА ЧАСТИНА**

# **2.1 Постановка задачі**

Метою створення автоматизованої інформаційної системи розрахунку кредитних та депозитних операцій клієнтами комерційного банку є:

* розробка і впровадження комплексної автоматизованої системи;
* створення єдиної бази даних;
* абстрагування від технології додатку-клієнта;
* розробка багатофункціонального, але зручного у користуванні інтерфейсу, зрозумілого для пересічного користувача;
* швидке реагування на звернення клієнтів;
* забезпечення інформування клієнта на високому рівні;
* безпека персональних даних клієнтів шляхом використання алгоритмів шифрування.

Система також призначена для ведення статистики, математичного та графічного відображення операцій комерційного банку.

# **2.2 Коротка характеристика засобів проектування**

При комп'ютерному моделюванні складних систем c успіхом використовуються об'єктно–орієнтовані мови. Володіння об'єктно–орієнтованою мовою програмування і доступ до великої бібліотеки ресурсів становить необхідну, але не достатню умову для створення об'єктної системи. Дуже важливу роль в процесі її розробки грають аналіз і проектування системи з погляду об'єктної методології.

Більшість сучасних методів ООАП базуються на використанні мови UML. Уніфікована мова моделювання UML (Unified Modeling Language) є мовою для визначення, подання, проектування і документування програмних систем, організаційно–економічних систем, технічних систем та інших систем різної природи. UML містить стандартний набір діаграм і нотацій найрізноманітніших видів.

UML (Unified Modeling Language – уніфікована мова моделювання) – мова графічного Опису для об'єктного моделювання в області розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називані UML моделлю. UML був створений для визначення, візуалізації, проектування й документування здебільшого програмних систем.

Сучасні CASE–засоби охоплюють велику область підтримки численних технологій проектування інформаційних систем (ІС): від простих засобів аналізу і документування до повномасштабних засобів автоматизації, що покривають весь життєвий цикл програмного забезпечення (ПЗ). Практично усі [CASE–засоби](https://uk.wikipedia.org/wiki/CASE) (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Під час розробки даної ІС використовувалась програма Rational Rose 2000. Rational Rose представляє собою CASE–засіб проектування і розробки інформаційних систем і програмного забезпечення для управління підприємствами. Як і інші CASE засоби (ARIS, BPwin, ERwin) його можна застосовувати для аналізу і моделювання бізнес процесів.

Принципова відмінність Rational Rose від інших засобів полягає в об'єктно–орієнтованому підході. Графічні моделі, що створюються за допомогою цього засобу, засновані на об'єктно–орієнтованих принципах і мові UML (Unified Modeling Language). Інструменти моделювання Rational Rose дозволяють розробникам створювати цілісну архітектуру процесів підприємства, зберігаючи всі взаємозв'язки та управляючі між різними рівнями ієрархії. Rational Rose на відміну від подібних засобів проектування здатна проектувати системи будь–якої складності, тобто інструментарій програми допускає як високорівневе (абстрактне) представлення (наприклад, схема автоматизації підприємства), так і низькорівневе проектування (інтерфейс програми, схема бази даних, частковий опис класів). Уся міць програми базується усього на 7 діаграмах, що у залежності від ситуації здатні описувати різні дії.

# **2.3 Модель досліджуваної системи**

Для того, щоб найбільш повно зрозуміти процес, необхідно мати велику кількість спостережень та вимірювань. Виділити основне, а потім глибоко дослідити процеси або явища за допомогою об’ємної, не систематизованої інформації достатньо важко.

Формальний опис мови UML ґрунтується на наступній загальній ієрархічній структурі модельних подань, що складається із чотирьох рівнів абстракції:

* позначка–метамодель,
* мета модель,
* модель,
* об'єкти користувача.

Модель – штучна система, що відображає з певною точністю основні властивості об’єкта–оригіналу, що вивчається. Модель знаходиться в певній відповідності із об’єктом вивчення, може замінити його при дослідженні і дозволяє отримати інформацію про цей об’єкт.

# **2.3.1 Опис бізнес–варіантів використання**

Діаграма прецедентів – в [UML](http://uk.wikipedia.org/wiki/UML), [діаграма](http://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0), на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є [графом](http://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

* асоціації (association relationship),
* включення (include relationship),
* розширення (extend relationship),
* узагальнення (generalization relationship).

Інформація відображена на діаграмі варіантів використання описує дійових осіб та їхні дії, а точніше варіанти використання (Див. Додаток А).

На діаграмі варіантів використання описана послідовність дій для дійових осіб, тобто ті варіанти використання, які може здійснювати кожен із акторів.

# **2.3.2 Діаграми класів системи**

Діаграма класів – статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: [класи](http://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)),[типи даних](http://uk.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка (розкривається в інших типах діаграм.

Діаграма класів – кінцевий результат проектування і відправна точка процесу розробки.

Програмний продукт складається із двох компонентів: WCF сервіса, та додатку-клієнта – Windows Forms App. Діаграма класів сервісу наведена в Додатку В, щодо діаграми класів додатку-клієнта, то вона наведена в Додатку Г. Щоб підключитися до бази даних через Entity Framework, нам потрібен контекст даних. Контекст даних являє собою клас, похідний від класу DbContext. Контекст даних містить одне або кілька властивостей типу DbSet <T>, де T представляє тип об'єкта, що зберігається в базі даних. Клас BankContext– самостійно розроблений контекст даних з наступними об’єктами:

* Таблиця користувачів – «User».
* Таблиця збору статистики по банківських операціях, в яких зацікавлений клієнт – «UserCalculator».
* Таблиця депозитів – «Deposit».
* Таблиця кредитів – «Credit».
* Таблиця відсотків по депозитах – «Rate».

Дана діаграма демонструє як класи пов’язані між собою та які типи зв’язків між ними (Додаток Б).

# **2.4 Інструкція роботи користувача з системою**

# **2.4.1 Інструкція роботи адміністратора з додатком.**

Для роботи з програмою необхідно авторизуватись. При відкритті програми з’являється форма введення логіна та пароля (рис.2.1).

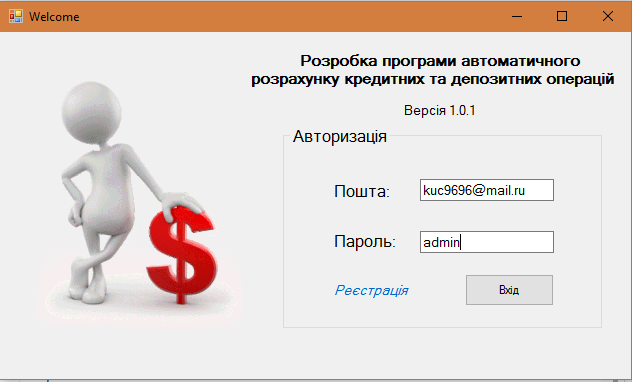


Рис. 2.1. Форма автризації

Після авторизації відкривається головна форма (рис. 2.2).

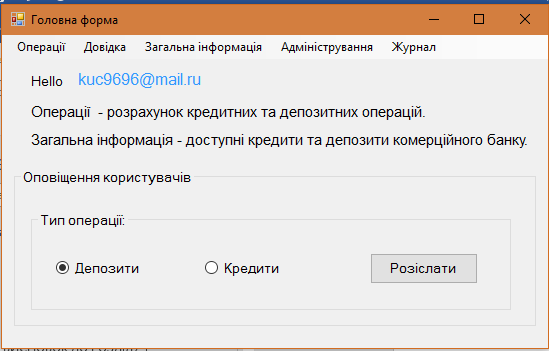


Рис. 2.2. Головна форма додатку

Головна форма містить кнопки «Операції», «Довідка», «Загальна інформація», «Адміністрування», «Журнал» вгорі (рис.2.3).

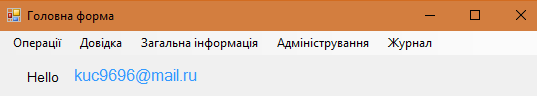


Рис. 2.3. Меню

При виборі пункту меню «Операції» з’являється випадаючий список, який надає можливість скористатить калькулятором депозитів та кредитів (рис.2.4).

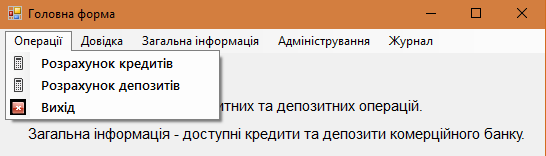


Рис. 2.4. Вибір пункту меню «Операції»

При виборі пункту меню «Довідка» - «Про програму» з’являється нове вікно, яке містить відомості про розробника програмного продукту (рис.2.5).

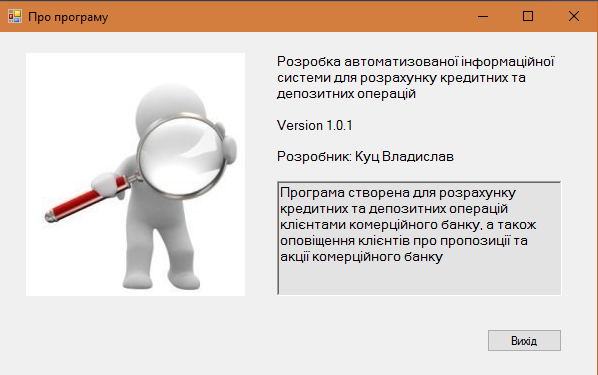


Рис. 2.5. Пункт меню «Довідка» - «Про програму»

При виборі пункту меню «Загальна інформація» - «Кредити» з’являється нове вікно, яке містить відомості про кредити, що доступні клієнтам (рис. 2.6).

При виборі пункту меню «Загальна інформація» - «Депозити» з’являється нове вікно, яке містить відомості про депозити, що доступні клієнтам (рис. 2.7).

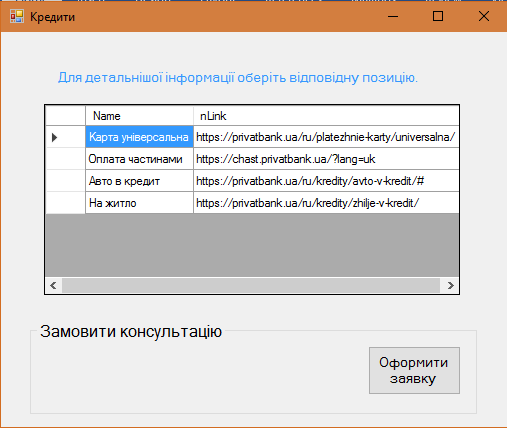


Рис. 2.6 Пункт меню «Загальна інформація» - «Кредити»

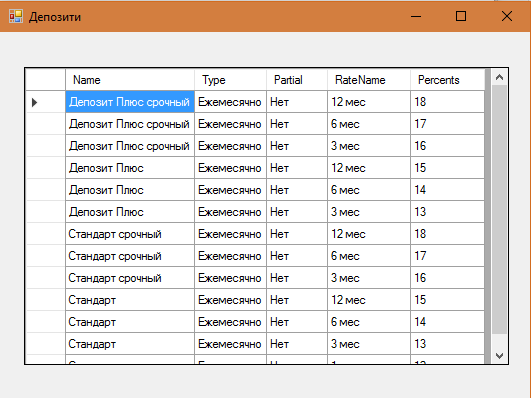


Рис. 2.7 Пункт меню «Загальна інформація» - «Депозити»

Детально переглянути інформацію за кредитами можна шляхом натиску на відповідний запис в таблиці, після чого користувача буде направлено на сторінку комерційного банку, де інформація відображена більш повно (рис. 2.8).

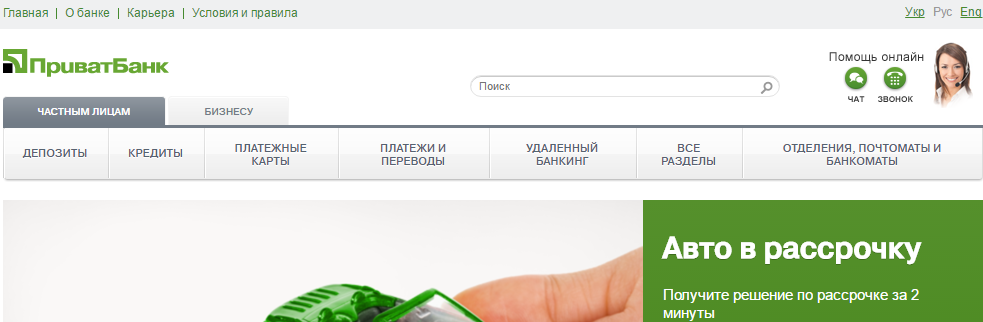


Рис. 2.8 Детальна інформація про кредит

Якщо в клієнта залишаються невирішені питання щодо кредиту, то він може замовити консультацію, натиснувши кнопку «Оформити заявку» (рис. 2.9).

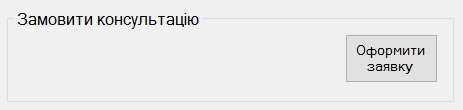


Рис. 2.9 Оформлення заявки

В результаті на поштову адресу відділу по роботі із персоналом буде надіслано листа (рис. 2.10).

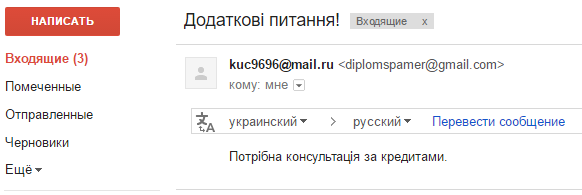


Рис. 2.10 Оформленна заявка за кредитами

Для того, щоб отримати додаткову інформацію по депозиту достатньо натиснути на відповідний запис в таблиці. Після цього буде відкрите нове вікно з детальною інформацією (рис. 2.11).

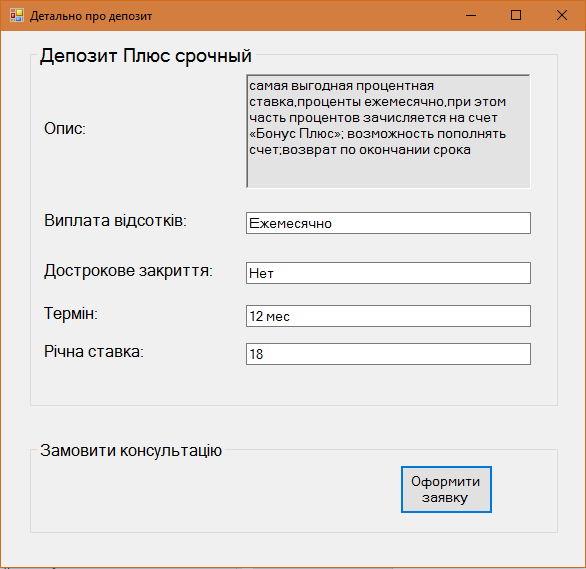


Рис. 2.11 Детальна інформація по депозиту

Після офрмлення заявки на електронну пошту відділу кадрів буде надіслано повідомлення (рис. 2.12).

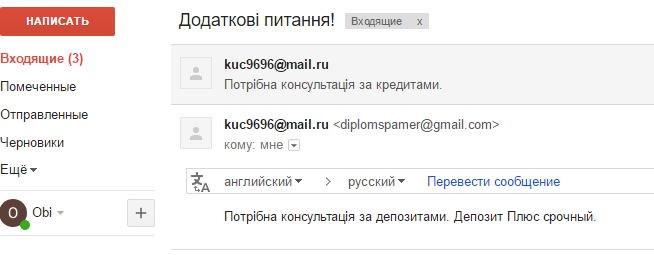


Рис. 2.12 Оформлена заявка за депозитами

Головним завданням адміністратора є ведення бази даних. Відкривши пункт меню «Адміністрування», можна здійснити операцію створення нової інформації по кредиту (рис. 2.13).

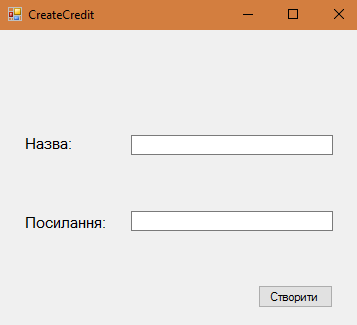


Рис. 2.13 Створення запису про кредит

Відкривши пункт меню «Адміністрування», можна здійснити операцію створення нової інформації по депозиту (рис. 2.14).

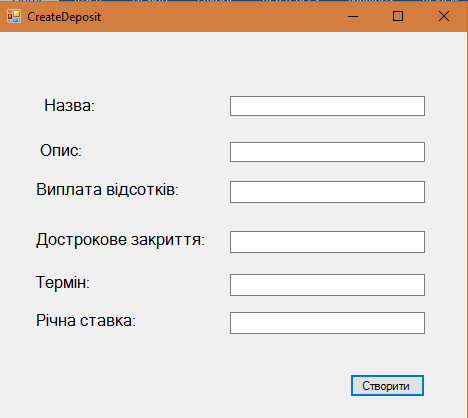


Рис. 2.14 Створення запису про депозит

Наступним завданням адміністатора є оповіщення клієнтів, які зацікавлені в операціях комерційного банку, відкривши пункт меню «Журнал» - «Інформувати про акції» та обравши відповідну категорію операцій (рис. 2.15).

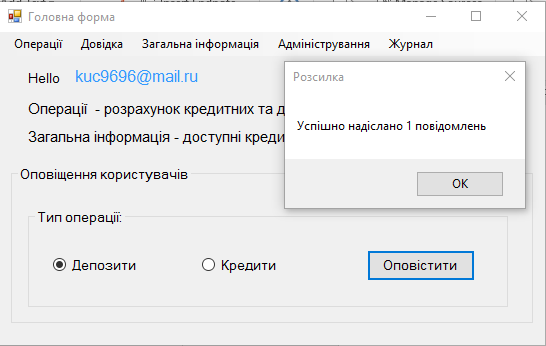


Рис. 2.15 Оповіщення клієнтів

# **2.4.2 Інструкція роботи клієнта банку із додатком.**

Після авторизації клієнту буде відображено головне вікно програми(рис. 2.16)

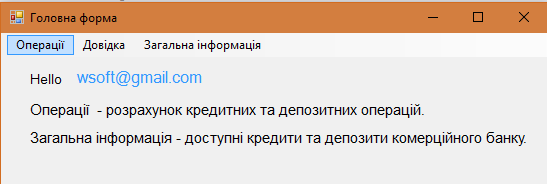


Рис. 2.16 Головне вікно клієнта

Роботу із вкладками «Довідка» та «Загальна інформація» було описано вище.

Обравши пункт меню «Операції» - «Розрахунок кредитів», клієнт має змогу змогу провести розрахунок кредитів відповідно до заданих критеріїв. Результати будуть відображені в табличному та графічному виді (рис. 2.17).

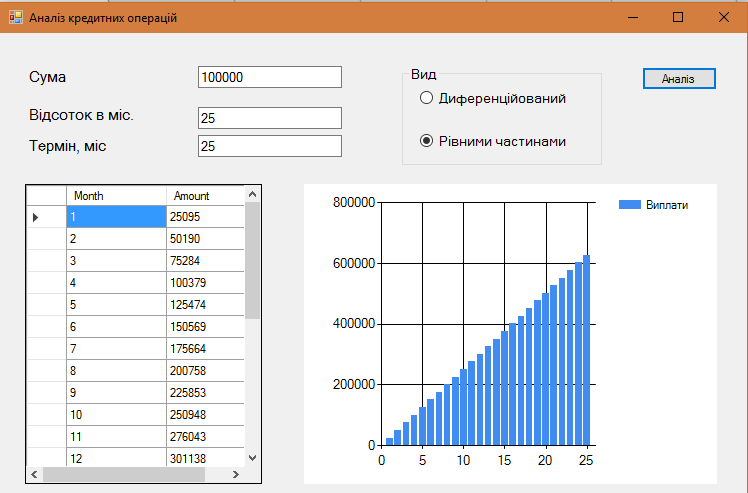


Рис. 2.17 Вікно з результами по розрахунку кредитної операції

Обравши пункт меню «Операції» - «Розрахунок депозитів», клієнт має змогу змогу провести розрахунок депозитів відповідно до заданих критеріїв. Результати будуть відображені в табличному та графічному виді (рис. 2.18).

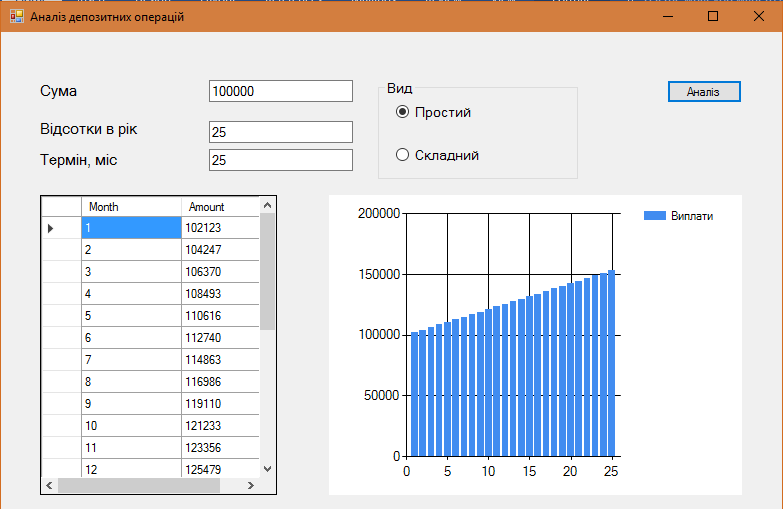


Рис. 2.18 Вікно з результами по розрахункудепозитної операції

# **ВИСНОВОК ДО РОЗДІЛУ 2**

В ході роботи було розроблено автоматизовану інформаційну систему розрахунку кредитних та депозитних операцій, оповіщення користувачів із забезпеченнями відповідного рівня безпеки особистих даних користувачів. Головною розробкою даного програмного продукту є створення бази даних для збереження інформації, створення сервісу WCF, що дозволяє абстрагуватися від технології на якій реалізовано додаток-клієнт, реалізація алгоритму розрахунку виплат за різними типами кредитів та депозитів, реагування на звернення клієнтів комерційного банку.

Розроблена система передбачає користувачів двох рівнів. Вхід в систему проводиться після введення відповідного логіну та паролю, права користувачів не однакові. Можна виділити дві ролі: адміністратор, клієнт комерційного банку. Користувач із правами адміністратора має наступні можливості:

* Здійснення CRUD операцій щодо депозитів – створення, отримання даних, оновлення та видалення.
* Здійснення CRUD операцій щодо кредитів – створення, отримання даних, оновлення та видалення.
* Оповіщення користувачів.

До можливостей клієнта входять:

* Розрахунок кредитів.
* Розрахунок депозитів.
* Перегляд актуальної інформації щодо послуг банку.
* Отримання додаткової інформації за невирішеними питаннями.

# **ВИСНОВОК**

Банківські комп'ютерні системи та технології на сьогоднішній день є одними з тих галузей, які швидко розвиваються в області прикладного мережевого програмного забезпечення. Як приклади передових технологій, використовуваних у банківській діяльності, можна назвати бази даних на основі моделі "клієнт-сервер" засобу міжмережевої взаємодії для міжбанківських розрахунків; служби розрахунків, цілком орієнтованих на Іnternet, або, так звані, віртуальні банки; банківські експертно-аналітичні системи.

В дипломному проекті розрахована надійність програмного продукту, який показує, що програма досить стабільна і може використовуватись банківськими установами для для забезпечення потреб клієнтів .

В ході роботи було розроблено зручну для користування автоматизовану інформаційну систему обробки операцій комерційного банку.

Інформацію викладено у доступному для не професіонала вигляді. Тож, відшукати необхідні дані не складе труднощів для основних користувачів автоматизованої інформаційної системи.

Інструкція для користувача системи викладена доступною для кожного мовою. Кожне твердження та хід роботи супроводжується зображенням відповідної форми та реакції програми на дії користувача.

Результати бакалаврської роботи полягають в наступному:

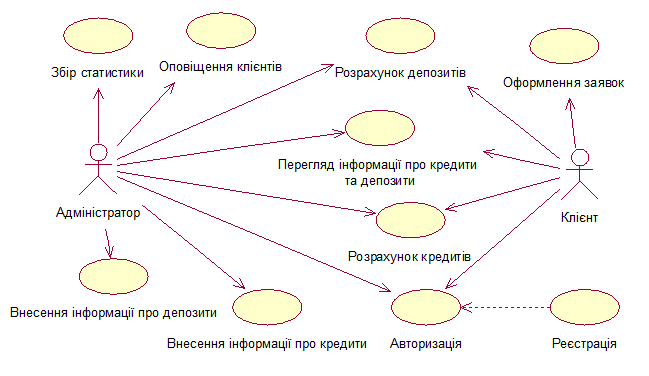
* Проведений аналіз існуючих найпоширеніших програм, виявлення їх переваг та недоліків.
* Збір статистичних данних для подальшого аналізу.
* Оповіщення користувачів за допомогою електронного листування.
* Створено базу даних для обліку клієтів, збереження інформації про кредити та депозити користувача.
* Забезпеченяя відповідного рівня безпеки даних клієнтів.

З урахуванням актуальності розглянутих питань розроблена методика та отримані результати можуть бути використані для подальших досліджень в даній предметній області.

# **СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. CASE–засоби. Особливості функціонування та роботи з ними. Режим доступу: [http://www.unicyb.kiev.ua/~boiko/it/new1/kondrat enko .html](http://www.unicyb.kiev.ua/~boiko/it/new1/kondrat%20enko%20.html)
2. Рамбо Дж., Якобсон А., Буч Г. UML: спеціальний довідник. – СПб: Пітер, 2002 г. Режим доступу: <http://www3.msiu.ru/~belova/compmod/obj.pdf>
3. Рамбо Дж., Якобсон А., Буч Г. Уніфікований процес розробки програмного забезпечення. – СПб .: Питер, 2002 г. Режим доступу: <http://irinavezner.ru/>
4. Ларман. Застосування UML і шаблонів проектування. – М.: «Вільямс», 2002 р.
5. Діаграм класів. Режим доступу: uk.wikipedia.org/wiki/Діаграма\_класів
6. Відносини класів – від UML до коду. Режим доступу: <https://habrahabr.ru/post/150041/>
7. Обмеження цілісності. Режим доступу: http://citforum.ru/database/dbguide/2–5.shtml
8. Діаграма прецедентів. Функції, особливості роботи: uk.wikipedia.org/wiki/Діаграма\_прецедентів
9. Діаграма компонентів. Функції, особливості роботи. Режим доступ: uk.wikipedia.org/wiki/Діаграма\_компонентів
10. Середовище програмування Microsoft Visual Studio. Режим доступу: <https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio>
11. Клименко О.В. «Інформаційні системи і технології в обліку»: навч. посібник / О. В. Клименко. – Київ : Центр учб. л–ри, 2008.
12. Дебет–Кредит. Бухгалтерія і фінанси. [Електронний ресурс]: [Веб–сайт]. Режим доступу <http://dtkt.com.ua/show/4cid0367.html>
13. Стаття про С шарп з Вікіпедії. [Електронний ресурс]: [Веб–сайт]. Режим доступу: <https://uk.wikipedia.org/wiki/C_Sharp>
14. Стаття про Visual Studio з Вікіпедії. [Електронний ресурс]: [Веб–сайт]. Режим доступу: <https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio>
15. Стаття про Linq. [Електронний ресурс]: [Веб–сайт]. Режим доступу: [https://uk.wikipedia.org/wiki/Linq](https://uk.wikipedia.org/wiki/ADO.NET)
16. Введення в Linq. [Електронний ресурс]: [Веб–сайт]. Режим доступу: <https://metanit.com/sharp/linq/1.1.php>
17. Juval Lowy. [Programming WCF Services: Mastering WCF and the Azure AppFabric Service Bus](http://www.amazon.com/Programming-WCF-Services-Mastering-AppFabric/dp/0596805489/ref=sr_1_1?ie=UTF8&qid=1298581118&sr=8-1). O’Reilly Media. 2010
18. Justin Smith. [Inside Windows Communication Foundation (Pro Developer)](http://www.amazon.com/Inside-Windows-Communication-Foundation-Developer/dp/0735623066/ref=sr_1_1?ie=UTF8&qid=1298581149&sr=8-1). Microsoft Press. 2007
19. Основи Git. Режим доступу: [https://git–scm.com/book/ru/v1/Введение–Основы–Git/](https://git-scm.com/book/ru/v1/Введение-Основы-Git/)
20. Веб сервіс GitHub. Режим доступу: [https://ru.wikipedia.org/wiki/ Bitbucket](https://ru.wikipedia.org/wiki/GitHub)
21. Репозиторій. Режим доступу: <https://uk.wikipedia.org/wiki/Репозиторій_програмного_забезпечення>
22. Офіційний сайт Microsoft TechNet – Режим доступу:[https://technet.microsoft.com/ru](https://vk.com/away.php?utf=1&to=https%3A%2F%2Ftechnet.microsoft.com%2Fru)
23. Перша програма із Entity Framework. Підхід Code First. Entity Framework. Режим доступу: <https://metanit.com/sharp/entityframework/1.2.php>
24. Шифрування. Режим доступу: <https://uk.wikipedia.org/wiki/Шифрування>
25. Алгоритм шифрування AES. Режим доступу: <https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard>
26. Операції комерційних банків, їх класифікація та загальна характеристика. Режим доступу: http://pidruchniki.com/10160520/finansi/operatsiyi\_komertsiynih\_bankiv\_klasifikatsiya\_zagalna\_harakteristika

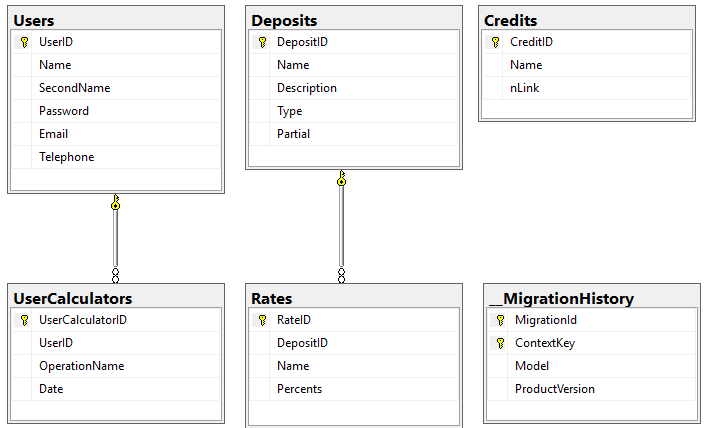
# **Додаток А**



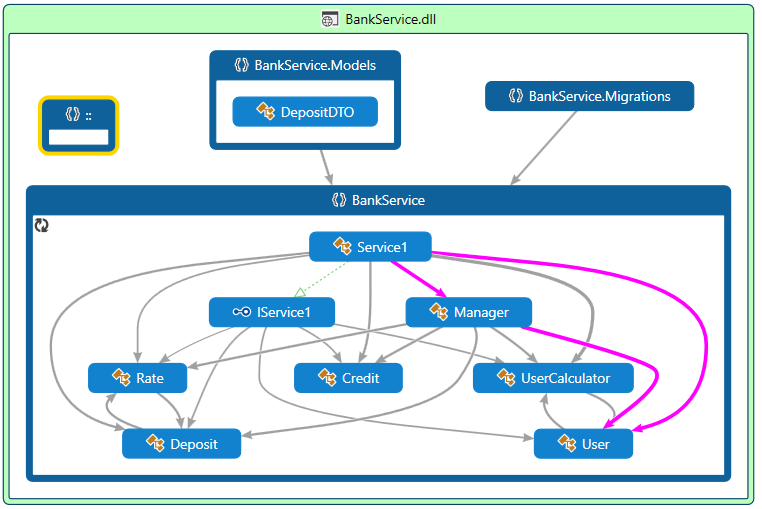
# **Додаток Б**

Діаграма «сутність–зв’язок» призначена для розробки моделей даних і забезпечує стандартний спосіб визначення даних і відносин між ними. Фактично за допомогою ERD здійснюється деталізація сховищ даних проектованої системи, а також документуються сутності системи і способи їх взаємодії, включаючи ідентифікацію об'єктів, важливих для предметної області (сутностей), властивостей цих об'єктів (атрибутів) та їх відносин з іншими об'єктами (зв'язків).

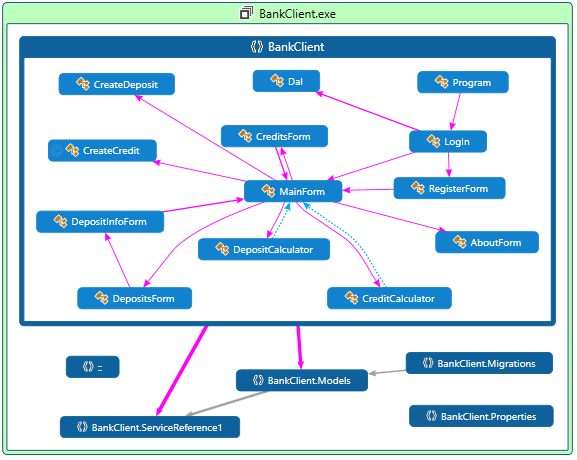
Дана діаграма показує нам зв’язки між сутностями у розробленій базі даних.



# **Додаток В. Сервіс**



# **Додаток Г. Клієнт**



# **Додаток Д. Лістинг програми**

Інтерфейс IService1 – інтерфейс, що описує контракт: описує моделі, сигнатуру методів.

public interface IService1

{

[OperationContract]

User LogIn(string name, string password);

[OperationContract]

void UpdateCalculator(UserCalculator calc);

[OperationContract]

string RegisterUser(User user);

[OperationContract]

List<Deposit> GetAllDeposits();

[OperationContract]

List<Rate> GetAllDepositsRates();

[OperationContract]

List<UserCalculator> GetInfoByUserEmail(string email);

[OperationContract]

List<string> GetEmailsByOperationType(string operationType);

[OperationContract]

List<Credit> GetAllCredits(string operationType);

[OperationContract]

void CreateCredit(Credit cr);

[OperationContract]

void CreateDeposit(Deposit dep);

[OperationContract]

int GetUserIDByEmail(string email);

}

[DataContract]

public class User

{

[DataMember]

public int UserID { get; set; }

[DataMember]

public string Name { get; set; }

[DataMember]

public string SecondName { get; set; }

[DataMember]

public string Password { get; set; }

[DataMember]

public string Email { get; set; }

[DataMember]

public string Telephone { get; set; }

[DataMember]

public List<UserCalculator> UserCalculations {get;set;}

}

[DataContract]

public class Deposit

{

[DataMember]

public int DepositID { get; set; }

[DataMember]

public string Name { get; set; }

[DataMember]

public string Description { get; set; }

[DataMember]

public string Type { get; set; }

[DataMember]

public string Partial { get; set; }

[DataMember]

public List<Rate> Rates { get; set; }

}

[DataContract]

public class Credit

{

[DataMember]

public int CreditID { get; set; }

[DataMember]

public string Name { get; set; }

[DataMember]

public string nLink { get; set; }

}

[DataContract]

public class Rate

{

[DataMember]

public int RateID { get; set; }

[DataMember]

public int DepositID { get; set; }

[DataMember]

public Deposit Deposit { get; set; }

[DataMember]

public string Name { get; set; }

[DataMember]

public int Percents { get; set; }

}

[DataContract]

public class UserCalculator

{

[DataMember]

public int UserCalculatorID { get; set; }

[DataMember]

public int UserID { get; set; }

[DataMember]

public string OperationName { get; set; }

[DataMember]

public DateTime Date { get; set; }

[DataMember]

public User User { get; set; }

}

Клас Service1, що реалізує інтерфейс Iservice.

public class Service1 : IService1

{

public BankContext db = new BankContext();

public void CreateCredit(Credit cr)

{

db.Credits.Add(cr);

db.SaveChanges();

}

public void CreateDeposit(Deposit dep)

{

db.Deposits.Add(dep);

db.SaveChanges();

}

public List<Credit> GetAllCredits(string operationType)

{

return db.Credits.ToList();

}

public List<Deposit> GetAllDeposits()

{

return db.Deposits.ToList();

}

public List<Rate> GetAllDepositsRates()

{

return db.Rates.ToList();

}

public List<string> GetEmailsByOperationType(string operationType)

{

List<string> res = new List<string>();

List<UserCalculator> dto = db.UserCalculators.Where(x => x.OperationName.ToLower() == operationType.ToLower())

.ToList();

foreach (UserCalculator operation in dto)

{

User \_user = db.Users.FirstOrDefault(x => x.UserID == operation.UserID);

if (res.IndexOf(\_user.Email) == -1)

{

res.Add(\_user.Email);

}

}

return res;

}

public List<UserCalculator> GetInfoByUserEmail(string email)

{

User \_user = db.Users.FirstOrDefault(x => x.Email.ToUpper() == email.ToUpper());

var res = db.UserCalculators.Where(x => x.UserID == \_user.UserID).ToList();

return res;

}

public int GetUserIDByEmail(string email)

{

User u = db.Users.FirstOrDefault(x => x.Email == email);

return u.UserID;

}

public User LogIn(string email, string password)

{

User \_user\_e = db.Users.FirstOrDefault(x => x.Email == email);

if (\_user\_e == null) return null;

if (Manager.DecryptUser(\_user\_e).Password == password) return \_user\_e;

else return null;

//#region Deposits

//Deposit d1 = new Deposit

//{

// Name = "Депозит Плюс срочный",

// Description = "самая выгодная процентная ставка,проценты ежемесячно,при этом часть процентов зачисляется на счет «Бонус Плюс»; возможность пополнять счет;возврат по окончании срока",

// Partial = "Нет",

// Type = "Ежемесячно"

//};

//Deposit d2 = new Deposit

//{

// Name = "Депозит Плюс",

// Description = "проценты ежемесячно, при этом часть процентов зачисляется на счет «Бонус Плюс»;возможность пополнять счет",

// Partial = "Нет",

// Type = "Ежемесячно"

//};

//Deposit d3 = new Deposit

//{

// Name = "Стандарт срочный",

// Description = "проценты ежемесячно;возможность пополнять счет;возврат по окончании срока",

// Partial = "Нет",

// Type = "Ежемесячно"

//};

//Deposit d4 = new Deposit

//{

// Name = "Стандарт",

// Description = "проценты ежемесячно; возможность пополнять счет",

// Partial = "Нет",

// Type = "Ежемесячно"

//};

//db.Deposits.Add(d1);

//db.Deposits.Add(d2);

//db.Deposits.Add(d3);

//db.Deposits.Add(d4);

//db.SaveChanges();

//#endregion

//#region DepoRates

//Rate r1 = new Rate { DepositID = 1, Name = "12 мес", Percents = 18 };

//Rate r2 = new Rate { DepositID = 1, Name = "6 мес", Percents = 17 };

//Rate r3 = new Rate { DepositID = 1, Name = "3 мес", Percents = 16 };

//Rate r4 = new Rate { DepositID = 2, Name = "12 мес", Percents = 15 };

//Rate r5 = new Rate { DepositID = 2, Name = "6 мес", Percents = 14 };

//Rate r6 = new Rate { DepositID = 2, Name = "3 мес", Percents = 13 };

//Rate r7 = new Rate { DepositID = 3, Name = "12 мес", Percents = 18 };

//Rate r8 = new Rate { DepositID = 3, Name = "6 мес", Percents = 17 };

//Rate r9 = new Rate { DepositID = 3, Name = "3 мес", Percents = 16 };

//Rate r10 = new Rate { DepositID = 4, Name = "12 мес", Percents = 15 };

//Rate r11 = new Rate { DepositID = 4, Name = "6 мес", Percents = 14 };

//Rate r12 = new Rate { DepositID = 4, Name = "3 мес", Percents = 13 };

//Rate r13 = new Rate { DepositID = 4, Name = "1 мес", Percents = 12 };

//db.Rates.Add(r1);

//db.Rates.Add(r2);

//db.Rates.Add(r3);

//db.Rates.Add(r4);

//db.Rates.Add(r5);

//db.Rates.Add(r6);

//db.Rates.Add(r7);

//db.Rates.Add(r8);

//db.Rates.Add(r9);

//db.Rates.Add(r10);

//db.Rates.Add(r11);

//db.Rates.Add(r12);

//db.Rates.Add(r13);

//db.SaveChanges();

//#endregion

//#region CreditOperations

//UserCalculator c = new UserCalculator { UserID = 2, OperationName = "Депозити", Date = DateTime.Now };

//UserCalculator c1 = new UserCalculator { UserID = 2, OperationName = "Депозити", Date = DateTime.Now };

//UserCalculator c2 = new UserCalculator { UserID = 2, OperationName = "Кредити", Date = DateTime.Now };

//db.UserCalculators.Add(c);

//db.UserCalculators.Add(c1);

//db.UserCalculators.Add(c2);

//db.SaveChanges();

//#endregion

//#region Credits

//Credit cr1 = new Credit { Name = "Карта універсальна", nLink = "https://privatbank.ua/ru/platezhnie-karty/universalna/" };

//Credit cr2 = new Credit { Name = "Оплата частинами", nLink = "https://chast.privatbank.ua/?lang=uk" };

//Credit cr3 = new Credit { Name = "Авто в кредит", nLink = "https://privatbank.ua/ru/kredity/avto-v-kredit/#" };

//Credit cr4 = new Credit { Name = "На житло", nLink = "https://privatbank.ua/ru/kredity/zhilje-v-kredit/" };

//db.Credits.Add(cr1);

//db.Credits.Add(cr2);

//db.Credits.Add(cr3);

//db.Credits.Add(cr4);

//db.SaveChanges();

//#endregion

//return null;

}

public string RegisterUser(User user)

{

User regUser = db.Users.FirstOrDefault(x => x.Email.ToUpper() == user.Email.ToUpper());

if (regUser != null)

{

return "";

}

else

{

try

{

user = Manager.EncryptUser(user);

db.Users.Add(user);

db.SaveChanges();

return user.Name;

}

catch

{

return "";

}

}

}

public void UpdateCalculator(UserCalculator calc)

{

db.UserCalculators.Add(calc);

db.SaveChanges();

}

}

Форма CreditCalculator, реалізує логіку калькулятора кредитів.

public partial class CreditCalculator : Form

{

string \_type;

Service1Client \_bankService = new Service1Client();

public CreditCalculator(string userEmail)

{

InitializeComponent();

}

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

\_type = (sender as RadioButton).Text;

}

private void radioButton2\_CheckedChanged(object sender, EventArgs e)

{

\_type = (sender as RadioButton).Text;

}

private void button1\_Click(object sender, EventArgs e)

{

switch (\_type)

{

case "Рівними частинами":An(); break;

case "Диференційований":De(); break;

default: An(); break;

}

}

private void An()

{

double v\_mes = Convert.ToInt32(textBox1.Text) \*

(

Convert.ToInt32(textBox3.Text) \* 0.01 \* Math.Pow(1 + Convert.ToInt32(textBox3.Text) \* 0.01, Convert.ToInt32(textBox2.Text))

/

(Math.Pow(1 + Convert.ToInt32(textBox3.Text) \* 0.01, Convert.ToInt32(textBox2.Text)) - 1));

List<Models.CreditGraph> data = new List<Models.CreditGraph>();

for (int i = 1; i <= Convert.ToInt32(textBox2.Text); i++)

{

Models.CreditGraph g = new Models.CreditGraph { Amount = Convert.ToInt32(v\_mes \* i), Month = i };

data.Add(g);

}

dataGridView1.DataSource = data;

chart1.Series["Виплати"].XValueMember = "Month";

chart1.Series["Виплати"].YValueMembers = "Amount";

chart1.DataSource = data;

chart1.DataBind();

UserCalculator c = new UserCalculator { Date = DateTime.Now, UserID = \_bankService.GetUserIDByEmail(MainForm.UserEmail), OperationName = "Кредити" };

\_bankService.UpdateCalculator(c);

}

private void De()

{

double sum = 0;

List<Models.CreditGraph> data = new List<Models.CreditGraph>();

for (int i = 1; i <= Convert.ToInt32(textBox2.Text); i++)

{

double che = Convert.ToInt32(textBox1.Text) / (Convert.ToInt32(textBox2.Text));

sum = che

+ ((Convert.ToInt32(textBox1.Text) -( che\* i))\*0.1/12);

Models.CreditGraph g = new Models.CreditGraph { Amount = Convert.ToInt32(sum ), Month = i };

data.Add(g);

}

dataGridView1.DataSource = data;

chart1.Series["Виплати"].XValueMember = "Month";

chart1.Series["Виплати"].YValueMembers = "Amount";

chart1.DataSource = data;

chart1.DataBind();

UserCalculator c = new UserCalculator { Date = DateTime.Now, UserID = \_bankService.GetUserIDByEmail(MainForm.UserEmail), OperationName = "Кредити" };

\_bankService.UpdateCalculator(c);

}

}

Форма DepositCalculator реалізує логіку калькулятора депозитів.

public partial class DepositCalculator : Form

{

string \_type;

Service1Client \_bankService = new Service1Client();

public DepositCalculator(string userEmail)

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

switch (\_type)

{

case "Простий": An(); break;

case "Складний": De(); break;

default: An(); break;

}

}

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

\_type = (sender as RadioButton).Text;

}

private void radioButton2\_CheckedChanged(object sender, EventArgs e)

{

\_type = (sender as RadioButton).Text;

}

private void An()

{

//Convert.ToInt32(textBox1.Text) +

double first = Convert.ToInt32(textBox1.Text) \* Convert.ToInt32(textBox3.Text) \* Convert.ToInt32(textBox2.Text) \* 31;

var second = first / 365 \* 0.01;

double res = Convert.ToInt32(textBox1.Text) + second;

List<Models.CreditGraph> data = new List<Models.CreditGraph>();

for (int i = 1; i <= Convert.ToInt32(textBox2.Text); i++)

{

double first1 = Convert.ToInt32(textBox1.Text) \* Convert.ToInt32(textBox3.Text) \* i \* 31;

var second1 = first1 / 365 \* 0.01;

double res1 = Convert.ToInt32(textBox1.Text) + second1;

Models.CreditGraph g = new Models.CreditGraph { Amount = Convert.ToInt32(res1), Month = i };

data.Add(g);

}

dataGridView1.DataSource = data;

chart1.Series["Виплати"].XValueMember = "Month";

chart1.Series["Виплати"].YValueMembers = "Amount";

chart1.DataSource = data;

chart1.DataBind();

UserCalculator c = new UserCalculator { Date = DateTime.Now, UserID = \_bankService.GetUserIDByEmail(MainForm.UserEmail), OperationName = "Депозити" };

\_bankService.UpdateCalculator(c);

}

private void De()

{

double sum = 0;

double first = Convert.ToInt32(textBox1.Text);

var second = 1 + Convert.ToInt32(textBox3.Text) \* 0.01 / Convert.ToInt32(textBox2.Text);

double res = first \* Math.Pow(second, Convert.ToInt32(textBox2.Text));

List<Models.CreditGraph> data = new List<Models.CreditGraph>();

for (int i = 1; i <= Convert.ToInt32(textBox2.Text); i++)

{

double first1 = Convert.ToInt32(textBox1.Text);

var second1 = 1 + Convert.ToInt32(textBox3.Text) \* 0.01 / i;

double res1 = first1 \* Math.Pow(second1, Convert.ToInt32(i));

Models.CreditGraph g = new Models.CreditGraph { Amount = Convert.ToInt32(res1), Month = i };

data.Add(g);

}

dataGridView1.DataSource = data;

chart1.Series["Виплати"].XValueMember = "Month";

chart1.Series["Виплати"].YValueMembers = "Amount";

chart1.DataSource = data;

chart1.DataBind();

UserCalculator c = new UserCalculator { Date = DateTime.Now, UserID = \_bankService.GetUserIDByEmail(MainForm.UserEmail), OperationName = "Депозити" };

\_bankService.UpdateCalculator(c);

}

}