ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»

Факультет комп'ютерно-інформаційних технологій

Кафедра комп'ютерних інформаційних систем і технологій

Родченко Ілля Сергійович

КВАЛІФІКАЦІЙНА РОБОТА

на тему: **«Розробка програмного забезпечення для розрахунку успішності студентів»**

Група: ІК-12-21-Б1ІПЗ(4,6з)

Спеціальність: Інженерія Програмного забезпечення

Рівень вищої освіти: перший (бакалаврський)

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

_____ Родченко Ілля Сергійович
(підпис)

Науковий керівник

_____ Піскунов О.Г., к. ф-м. н., доц.
(підпис)

Допущено до захисту ЕК

Завідувач кафедри _____ Сергій КАВУН , д.е.н., професор
(підпис)

Київ 2026

РЕФЕРАТ

Пояснювальна записка до атестаційної роботи: 76 с., 22 рис., 13 додатків, 22 джерела.

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРАХУНКУ УСПІШНОСТІ СТУДЕНТІВ.

Об’єктом дослідження є розробка програмного забезпечення за допомогою системи управління базами даних SQLite і мови програмування C#.

Метою роботи є розробка програмного забезпечення для розрахунку успішності студентів за допомогою системи управління базами даних SQLite і мови програмування C#. Програма має розраховувати бали студентів за семестр, зберігати їх у зручній формі і мати можливість редагувати і експортувати дані.

Методи розробки базуються на мові C# та з використанням системи управління базами даних SQLite.

Результатом роботи є розробка програмного забезпечення для розрахунку успішності студентів. Розроблена програма може використовуватися в навчальному процесі студентів.

ЗМІСТ

ВСТУП

Актуальність

У сучасних закладах освіти важливу роль відіграє автоматизація процесів обробки та збереження інформації. Одним із таких процесів є облік та розрахунок успішності студентів. Традиційні способи ведення журналів успішності, які базуються на паперовій документації або використанні електронних таблиць, потребують значних витрат часу, можуть містити помилки та не забезпечують достатнього рівня зручності при роботі з великими обсягами даних.

Розвиток інформаційних технологій та систем управління базами даних дає можливість створювати програмні засоби, що дозволяють автоматизувати обчислення, зберігати результати навчання у структурованому вигляді та швидко отримувати необхідну інформацію. Особливо актуальним є використання легких і швидких систем управління базами даних, таких як SQLite, у поєднанні з мовою програмування C#, яка забезпечує створення зручного та функціонального програмного забезпечення.

Розробка програмного забезпечення для розрахунку успішності студентів дозволяє підвищити ефективність роботи викладачів і студентів, спростити процес обробки даних, мінімізувати кількість помилок та забезпечити надійне збереження інформації. Саме тому тема дипломної роботи є актуальною та має практичне значення для сучасного освітнього процесу.

Метою роботи є розробка програмного забезпечення для розрахунку успішності студентів із використанням системи управління базами даних SQLite та мови програмування C#.

Для досягнення поставленої мети були поставлені такі завдання:

1. проаналізувати сучасні засоби розробки програмного забезпечення та системи управління базами даних;
2. дослідити можливості SQLite для збереження та обробки інформації;
3. обґрунтувати вибір мови програмування C# для реалізації проєкту;
4. розробити структуру бази даних для зберігання інформації про студентів та їх успішність;

5. створити програму для введення, редагування, збереження та експорту даних;
6. реалізувати функції автоматичного розрахунку балів студентів за семестр;
7. провести тестування програмного забезпечення та оцінити результати його роботи.

Наукова новизна та практична цінність роботи полягає у поєднанні можливостей мови програмування C# та системи управління базами даних SQLite для створення компактного та ефективного програмного забезпечення, призначеного для автоматизації процесу розрахунку успішності студентів.

Практична цінність роботи полягає у створенні готового програмного продукту, який може бути використаний у навчальних закладах для ведення обліку успішності студентів, автоматичного розрахунку балів, редагування та збереження інформації. Розроблена програма забезпечує зручність роботи користувача, скорочує час на обробку даних та зменшує ймовірність виникнення помилок при виконанні розрахунків.

Отримані результати можуть бути використані в освітньому процесі, а також як основа для подальшого вдосконалення та розширення функціональних можливостей програмного забезпечення.

1. ТЕОРЕТИЧНА ЧАСТИНА

1.1 Загальні відомості

1.1.1 Поняття про базу даних

База даних (англ. database) - це впорядкована сукупність взаємопов’язаних даних, які зберігаються відповідно до певної структури та призначені для використання в конкретній предметній області. Вона забезпечує збереження, обробку та пошук інформації, а також підтримує зв’язки між окремими елементами даних. Згідно зі стандартом ISO/IEC 2382:2015, база даних являє собою організований набір інформації, характеристики та взаємозв’язки якої описуються певною концепцією.

Сучасні бази даних можуть містити таблиці, схеми, представлення, збережені процедури та інші об’єкти. Окрім самих даних, вони включають засоби опису структури інформації та інструменти для її обробки. Організація даних здійснюється відповідно до визначеної моделі, що забезпечує зручність роботи та швидкий доступ до необхідної інформації.

У базах даних інформація про об’єкти може бути представлена у вигляді таблиць, документів, графів, колекцій та інших структур залежно від типу системи. Дані являють собою окремі одиниці інформації, які можуть бути подані у текстовому, числовому, графічному чи іншому форматі. Вони використовуються для зберігання, передачі та подальшої обробки інформації.

Однією з важливих особливостей сучасних баз даних є можливість їх вбудовування у програмні застосунки. У цьому випадку система управління базами даних працює як складова частина прикладної програми, а користувач не помічає її окремого функціонування. Такий підхід спрощує адміністрування, усуває необхідність складних мережевих налаштувань та підвищує зручність використання програмного забезпечення.

1.1.2 Типи баз даних

Існує велика кількість різновидів баз даних, кожен із яких має власну структуру, принципи організації та сферу застосування.

Розглянемо основні типи баз даних:

Реляційні бази даних

Реляційні бази даних є одними з найпоширеніших і найбільш розвинених типів БД. У таких системах інформація подається у вигляді взаємопов'язаних таблиць. Кожен запис у таблиці має унікальний ідентифікатор, а зв'язки між таблицями реалізуються за допомогою первинних і зовнішніх ключів.

Таблиця складається з рядків та стовпців:

- стовпці визначають характеристики об'єктів та містять дані одного типу;
- рядки представляють окремі записи;
- на перетині рядка і стовпця знаходиться конкретне значення.

Така структура забезпечує зручне зберігання інформації, швидкий пошук даних та підтримку зв'язків між різними таблицями.

Ієрархічні бази даних

Ієрархічні бази даних організовують інформацію у вигляді дерева. Дані розміщуються за принципом «батьківський - дочірній» елемент, де один запис може мати декілька підлеглих записів. Подібна структура нагадує дерево каталогів.

Недоліком ієрархічної моделі є її обмежена гнучкість, оскільки дочірній елемент може мати лише одного батьківського. Незважаючи на це, такі бази даних використовуються у системах, де потрібна висока швидкодія та надійність.

Мережеві бази даних

Мережеві бази даних є розвитком ієрархічної моделі. На відміну від ієрархічних БД, один запис може бути пов'язаний одразу з кількома батьківськими елементами. Завдяки цьому можна створювати складніші структури взаємозв'язків між даними.

У такій моделі інформація подається у вигляді графа, що забезпечує більш гнучку організацію даних.

Об'єктно-орієнтовані бази даних

В об'єктно-орієнтованих базах даних інформація зберігається у формі об'єктів. Кожен об'єкт містить як дані, так і методи для роботи з ними. Такий

підхід добре поєднується з об’єктно-орієнтованими мовами програмування та дозволяє ефективніше працювати зі складними структурами даних.

Бази даних типу «ключ–значення»

У базах даних цього типу інформація зберігається у вигляді пар «ключ–значення». Ключ використовується для швидкого пошуку відповідного значення. Такі системи застосовуються у кешуванні, високошвидкісних сервісах та інших задачах, де важлива швидкість доступу до даних.

Бази даних широко використовуються у різних сферах діяльності: освіті, медицині, банківській справі, торгівлі, промисловості та багатьох інших. Вони дозволяють зберігати значні обсяги інформації, забезпечують швидкий доступ до даних та допомагають автоматизувати різноманітні процеси.

1.1.3 Системи керування базами даних

Система керування базами даних (СКБД) - це програмне забезпечення, призначене для створення, зберігання, обробки та керування даними в базі даних. СКБД забезпечує взаємодію користувача або прикладної програми з даними та визначає структуру їх зберігання.

До основних функцій систем керування базами даних належать:

- створення та редагування структури бази даних;
- додавання, зміна та видалення даних;
- забезпечення цілісності інформації;
- контроль доступу користувачів;
- захист даних від втрати чи несанкціонованого доступу.

Одним із головних завдань сучасних СКБД є підтримка цілісності даних. Для цього використовуються різноманітні обмеження та правила, що дозволяють уникнути дублювання інформації та забезпечують її узгодженість.

СКБД також реалізують механізми авторизації та розмежування прав доступу, що дозволяє контролювати роботу користувачів із базою даних.

Основними компонентами системи керування базами даних є:

- **механізм зберігання даних** - відповідає за фізичне розміщення інформації, її захист та оптимізацію доступу;

- **механізм обробки запитів** - забезпечує читання, запис та оновлення даних;
- **система керування схемами** - визначає структуру бази даних і контролює відповідність даних заданим правилам.

Використання СКБД значно спрощує роботу з інформацією, підвищує швидкість обробки даних, забезпечує безпечне зберігання та надійний доступ до них. Завдяки цьому системи керування базами даних є важливою складовою сучасних інформаційних систем.

Серед популярних реалізацій СКБД можна виділити як комерційні, так і відкриті рішення.

До комерційних СКБД належать:	До систем із відкритим кодом належать:
Microsoft SQL Server	MySQL
Oracle	PostgreSQL
DB2	SQLite
Informix	Firebird
Sybase	Apache Derby
InterBase	

1.2 Означення виняткових ситуацій

Виняткова ситуація (exceptions) – це неочікуваний аварійний стан комп’ютера при якому він не може продовжувати обчислення.

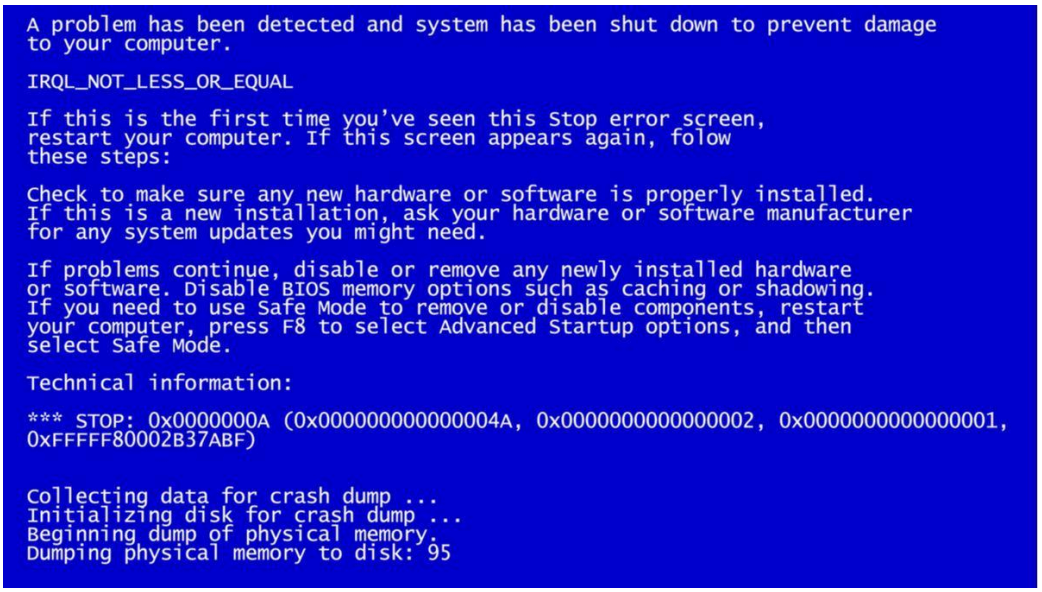


Рис. 1.2.1 Приклад аварійного стану комп’ютера з встановленою операційною системою сімейства Windows.

1.2.1 Теорія типів Рассела і теорема Кантора

Однією з важливих проблем у розвитку математичної логіки та теорії множин стали логічні суперечності, які виникали при побудові універсальних множин. Для усунення таких парадоксів були створені спеціальні підходи, серед яких важливе місце займають теорія типів Бертрана Рассела та теорема Георга Кантора.

Теорія типів була запропонована англійським математиком і філософом Bertrand Russell з метою уникнення логічних парадоксів у теорії множин. Основною причиною створення цієї теорії став так званий парадокс Рассела.

Парадокс полягає у розгляді множини всіх множин, які не містять самі себе як елемент. Якщо така множина містить себе, тоді за означенням вона не повинна містити себе. Якщо ж вона не містить себе, тоді відповідно до означення повинна містити себе. Таким чином виникає суперечність.

Для розв’язання цієї проблеми Рассел запропонував поділити всі об’єкти на різні типи. Об’єкти одного типу можуть містити лише елементи нижчого типу, але не можуть містити самі себе. Завдяки такому обмеженню усувається можливість утворення суперечливих множин.

У теорії типів:

- об’єкти нульового типу є базовими елементами;
- множини елементів нульового типу належать до першого типу;
- множини множин першого типу належать до другого типу і так далі.

Теорія типів стала важливим етапом розвитку математичної логіки та вплинула на подальший розвиток інформатики, мов програмування та систем типізації даних.

Теорема Кантора

Теорема Кантора була сформульована німецьким математиком Georg Cantor та є однією з основних теорем теорії множин.

Суть теореми полягає в тому, що для будь-якої множини кількість усіх її підмножин завжди більша за кількість елементів самої множини. Множина всіх підмножин називається булеаном або степеневою множиною.

Теорему можна подати у вигляді:

$|P(A)| > |A|$

де:

A - деяка множина;

P(A) - множина всіх підмножин множини A;

|A| - потужність множини.

Кантор довів, що не існує взаємно однозначної відповідності між множиною та множиною всіх її підмножин. Це означає, що потужність булеану завжди більша за потужність вихідної множини.

Теорема Кантора має велике значення у математиці та інформатиці, оскільки вона показує існування різних рівнів нескінченності та використовується при дослідженні логічних систем, алгоритмів і структур даних.

Теорія типів Рассела та теорема Кантора стали важливими елементами розвитку сучасної математичної логіки. Вони вплинули на створення формальних систем, мов програмування та методів організації даних, які використовуються у сучасних інформаційних технологіях.

Загалом, для вирішення парадоксів теорії множин Кантора Бертран Рассел (Портрети в додатках 1, 2) побудував теорію типів, котра дозволила використовувати у програмуванні теорію множин. Наразі теорія типів використовується і в сучасності.

1.2.2 Мова PL/I

Мова PL/I (Programming language one), бувши розробленою компанією IBM (International Business Machines) в 1960-му році, стала першою мовою програмування, в якій було передбачено управління виключними станами на високому рівні.

Приклад обробки помилки Overflow мовою PL/I:

```
ON OVERFLOW
   GOTO error;
...
error:
   PRINT "something bad happened"
```

1.3 Сучасні методи обробки виняткових ситуацій

1.3.1 Піраміда тестування програм

Щоб описати загальний обсяг тестів, можна скористуватись діаграмою.

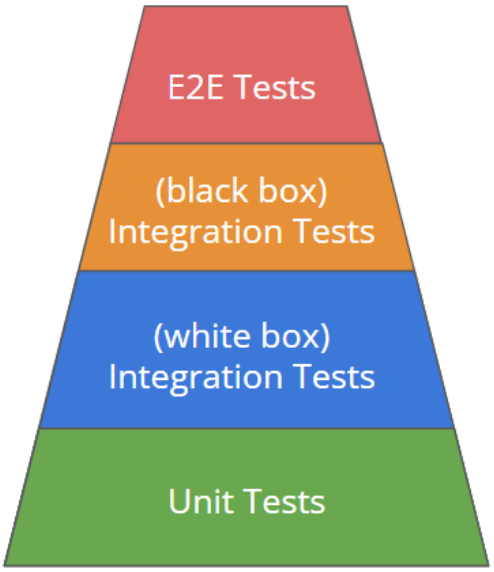


Рис. 1.3.1 Піраміда тестування.

Окремі функції і класи перевіряються юніт-тестами. Далі ці тести поєднуються між собою інтеграційними тестами по частинам і, врешті решт, запускаються E2E тести, котрі перевіряють всю систему зверху донизу як одне ціле, як готову програму.

1.3.2 Ручне тестування

Ручне тестування – метод тестування, при якому тести виконуються розробниками, а не кодом. Ручне тестування необхідне бути здійснене перед автоматизацією тестування, щоб оцінити можливість її імплементації.

Характеристики ручного тестування:

1. Займає багато часу.
2. Велика вірогідність появи помилок через людський фактор.
3. Можливість дослідницького тестування.
4. Для ручного тестування не обов’язкове знання мов програмування.
5. Для ручного тестування не використовуються бібліотеки.
6. Менша точність.

Алгоритм дій для ручного тестування:

1. **Аналіз вимог:** Дослідити документацію проєкту, посібники, і застосунок, що перевіряється. Проаналізуввати вимоги від SRS (специфікацій вимог до програмного забезпечення).
2. **Створення плану тестування:** Створити план тестування, який покриває усі вимоги.
3. **Створення юніт тестів:** Написати юніт-тести, що покриваються усі вимоги, прописані в документації.
4. **Запуск юніт-тестів:** Оглянути і прокоментувати юніт-тести з лідером команди і клієнтом. Запустити юніт-тести для програмі, що перевіряється.
5. **Реєстрація дефектів:** Знайти помилки, зареєструвати і передати їх розробникам.
6. **Вирішення помилок і повторна перевірка:** Коли помилки вирішені, знову запустити юніт-тести, щоб перевірити, що вони не видають нових помилок.

1.3.3 Unit-тести

Тестування юніт-тестами – тестування найменших одиниць програми: функцій, класів.

Приклади юніт-тестів:

Для Android

див. Додаток 3, 4

Для Angular

див. Додаток 5, 6, 7, 8

Для NodeJS

див. Додаток 9, 10

Для React Native

див. Додаток 11

Black box тестування

Тестування, яке характеризується тим, що розробники не мають можливості побачити внутрішню структуру або код застосунку.

Це дає змогу тестувальнику перевірити програму в якості кінцевого користувача й на своєю досвіді удостовіритись, як відбувається взаємодія з нею. Таке тестування підходить для перевірки аспектів UX (user experience) дизайну, як от: зручність використання, доступність і функціональність.

White box тестування

Тестування, яке характеризується тим, що розробники повністю ознайомлені зі структурою застосунку, кодом і документацією. Такий підхід дозволяє командам розробки зфокусуватися на перевірці внутрішніх процесів, наприклад: удостовіритись, що рядки коду працюють, як належно, реєстрація помилок або проріх у безпеці і перевірка функціональності тестових випадків.

Gray box тестування

Комбінація black box і white box тестувань, яка характеризується тим, що розробники мають часткове знання внутрішньої структури застосунку. Наприклад, вони можуть знати лише частину коду, або мати доступ лише до документації. Таким чином, розробники можуть оцінити функціональність і безпеку продукту не поглиблюючись у подробиці, як при black box тестуванні.

1.3.4 Інтеграційне тестування

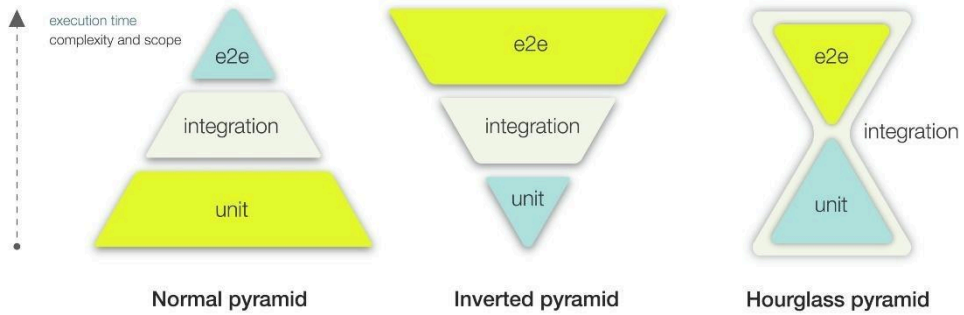


Рис. 1.3.2 Піраміди тестування

Після написання юніт-тестів окремі частини програми тестуються спільно, щоб перевірити, як вони взаємодіють і чи нема проблем з сумісною роботою модулів, запевнитись у її надійності.

Важливість інгтеграційних тестів полягає в тому, що їй наявність і якість дозволяє зменшити витрати на найскладніші тести – E2E.

Підходів до інтеграційного тестування існує декілька:

Інтеграція великого вибуху (Big-Bang Integration)

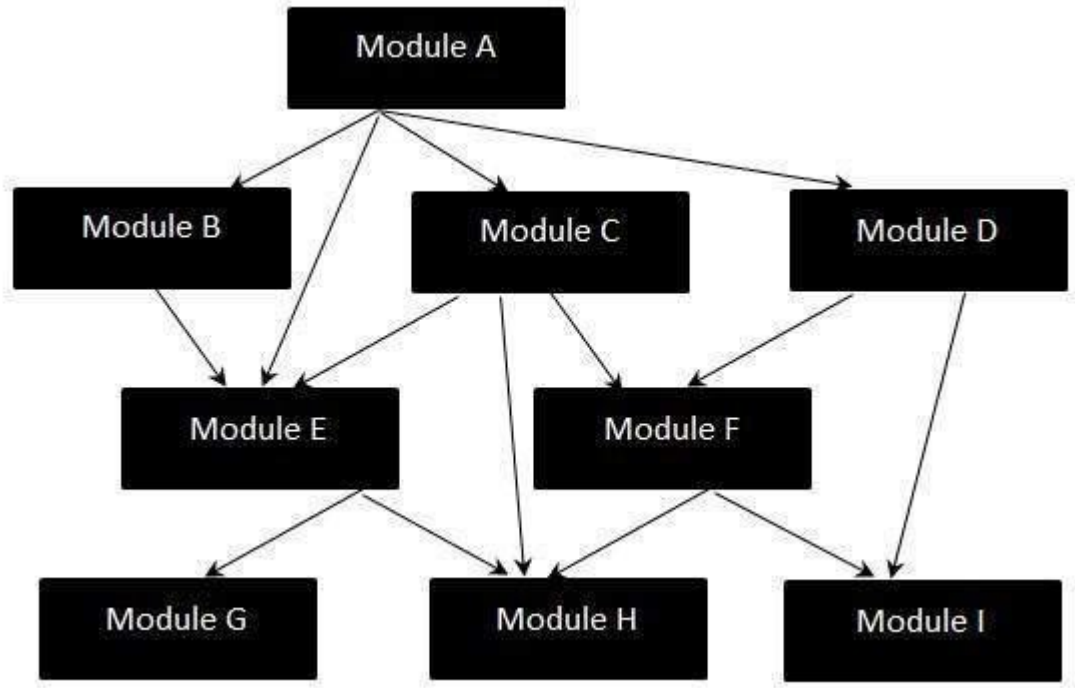


Рис. 1.3.3 Інтеграція великого вибуху

Усі модулі поєднуються разом і тестуються як одне ціле.

Переваги:

- Краще всього працює з маленькими системами.
- Швидко видає результат.

Недоліки:

- Складніше знайти помилки.
- Можливість пропустити компонент при перевірці.
- Усі модулі мають бути завершеними, тож тестування не може початися раніше.
- Особливо важливі модулі не перевіряються більш ретельно, ніж інші.

Інтеграція згори вниз (Top Down Integration)

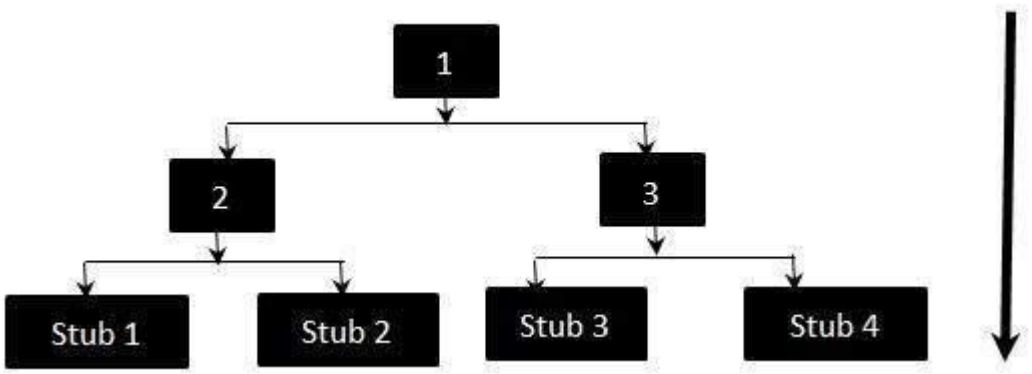


Рис. 1.3.4 Інтеграція згори вниз

При такому підході спочатку тестуються і поєднуються модулі найвищого рівня, потім тестуються модулі нижчого рівня і приєднуються, коли їх функціональність підтверджується.

Переваги:

- Легкість знаходження і виправлення помилок.
- Можливість створення раннього прототипу програми.
- Тести, які виконуються над пріоритетними модулями можуть розкрити значущі помилки, які можна виправити до повної інтеграції.

Недоліки:

- Мають бути написані заглушки замість відсутніх модулів.
- Модулі низького рівня часто тестуються недостатньо.

Інтеграція знизу вгору (Bottom Up Integration)

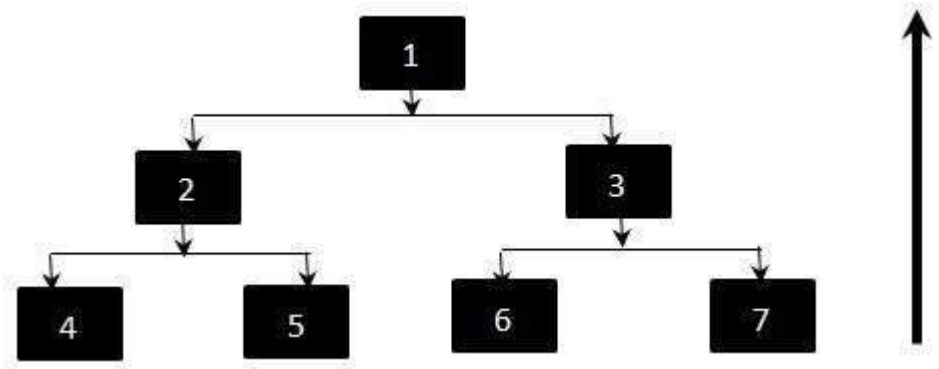


Рис. 1.3.5 Інтеграція знизу вгору

На відміну від інтеграції згори вниз, у інтеграції знизу вгору першими тестуються низькорівневі модулі. Результати будують фундамент для тестування вищих рівнів.

Переваги:

- Легкість знаходження й виправлення помилок.
- Нема необхідності чекати написання усіх модулів перед початком тестування.

Недоліки:

- Критичні, високорівневі компоненти тестуються останніми.

- Неможливо зробити ранній прототип.

Гібридна інтеграція (Hybrid Integration)

Рис. 1.3.6 Гібридна інтеграція

Інтегроване тестування відбувається одночасно з обох кінців.

Переваги:

- Забезпечує всебічний погляд на систему.
- Дозволяє знаходити помилки в процесі розробки.

Недоліки:

- Складність налаштування й підтримання середовища для тестування.
- Займає багато часу й ресурсів порівняно з іншими підходами.

1.3.5 End-to-end (E2E) Тестування

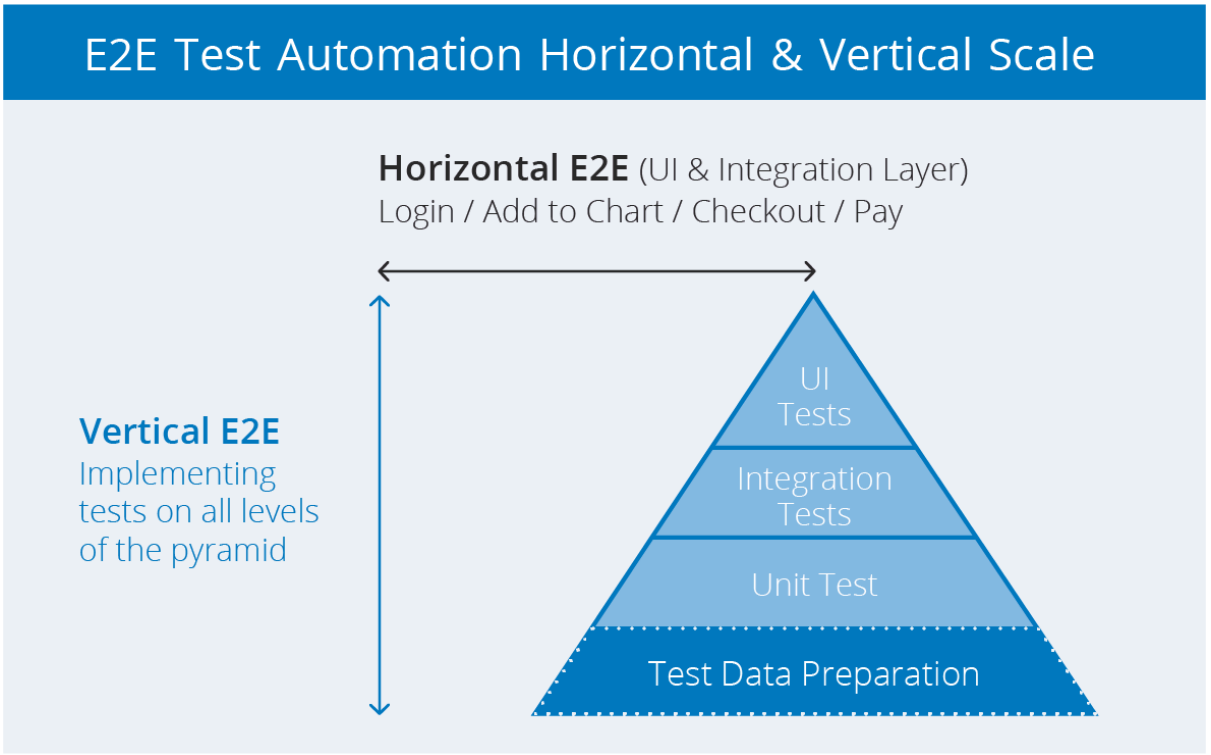


Рис. 1.3.7 Піраміда тестування

По складності, об’ємом роботи й затратності за часом після інтеграційних тестів слідують E2E тести, які перевіряють, що вся система працює правильно.

Перевіряються, зокрема, кнопки, форми, зміни, посилання, зовнішні процеси. Загалом, увесь процес роботи програми.

Приклад E2E тесту:

Див. Додаток 12.

Пояснення до коду вище:

1. Відвідати https://example.cypress.io .
2. Знайти елемент зі змістом: type.
3. Натиснути на нього.
4. Взяти URL.
5. Впевнитись, що він містить: /commands/actions
6. Взяти вхід для даних з data-testid: action-email
7. Написати fake@email.com у вхід для даних.
8. Впевнитись, що вхід для даних має нове значення.

Горизонтальне E2E тестування

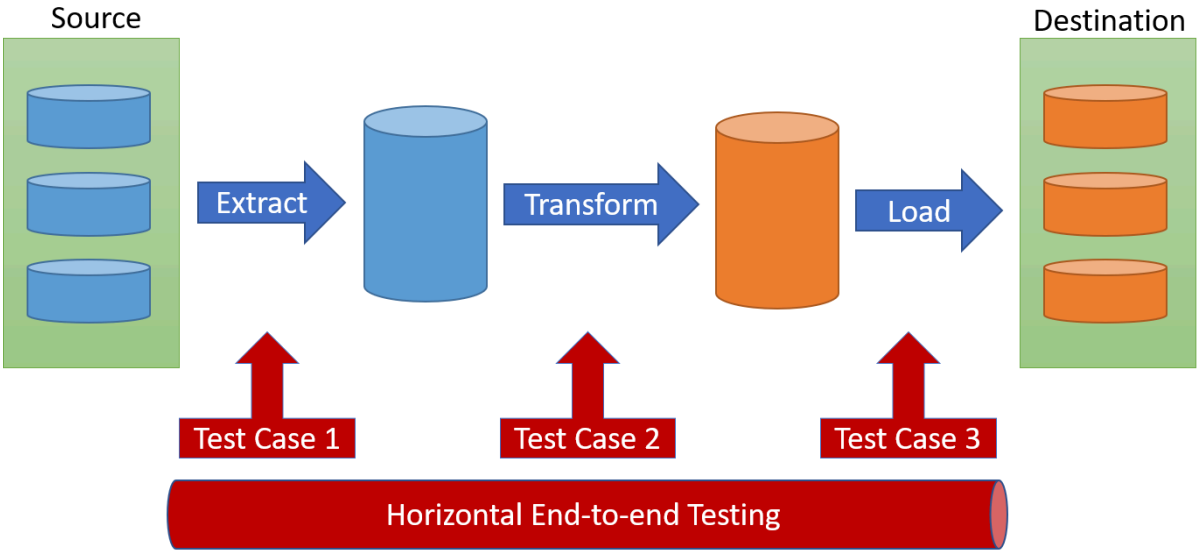


Рис. 1.3.8 Горизонтальне E2E тестування

Горизонтальне тестування зосереджується на перевірці всього процесу використання користувачем програмного забезпечення. Тобто, тести що виконуються від точки зору клієнта. Імітується поведінка користувача і програма перевіряється на помилки під час цього процесу.

Вертикальне E2E тестування

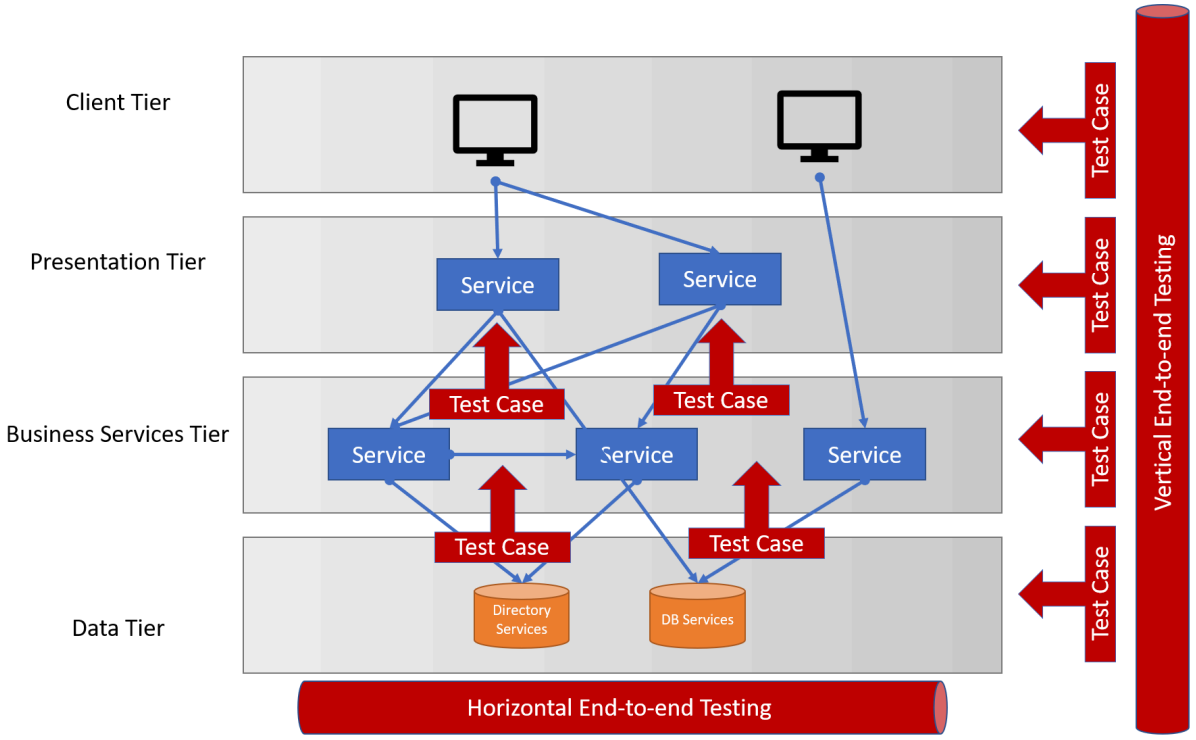


Рис. 1.3.9 Вертикальне E2E тестування

Вертикальне тестування – перевірка процесів, що відбуваються поза оком користувача. Себто, перевірка логіки програми, удостовірення, що потрібні дані передаються в належні їм місця, в належний час, щоб усі процеси на фоні проходили швидко і без проблем.

В даній роботі, здебільшого використовувалися Smoke testing (ручне тестування) і тестування знизу вверх.

1.4 Порівняння мов програмування та систем керування базами даних

У сучасному програмуванні існує велика кількість мов програмування та систем керування базами даних, які використовуються для створення програмного забезпечення різного призначення. Вибір відповідної мови програмування та СКБД є важливим етапом розробки будь-якого програмного продукту, оскільки від цього залежить продуктивність, швидкодія, зручність підтримки та подальший розвиток системи.

Для розробки програмного забезпечення розрахунку успішності студентів необхідно обрати інструменти, які забезпечують:

- зручність створення графічного інтерфейсу;

- простоту роботи з базами даних;
- високу швидкодію;
- стабільність роботи;
- можливість подальшого розширення функціоналу;
- простоту встановлення та використання.

Порівняння мов програмування

Серед найбільш популярних мов програмування для створення прикладного програмного забезпечення можна виділити:

- C++;
- Java;
- Python;
- JavaScript;
- C#.

Мова програмування C++

C++ є однією з найпотужніших мов програмування системного та прикладного рівня. Вона забезпечує високу продуктивність і дозволяє ефективно працювати з пам’яттю та ресурсами комп’ютера.

Переваги C++:

- висока швидкість виконання програм;
- можливість роботи на низькому рівні;
- підтримка об’єктно-орієнтованого програмування;
- велика кількість бібліотек.

Недоліки:

- складність синтаксису;
- складніший процес розробки;
- вища ймовірність помилок при роботі з пам’яттю;
- потребує більше часу на створення програмного забезпечення.

Для невеликих інформаційних систем використання C++ не завжди є доцільним через складність розробки.

Мова програмування Java

Java є популярною об’єктно-орієнтованою мовою програмування, яка широко використовується для створення корпоративних систем та кросплатформних застосунків.

Переваги Java:

- незалежність від операційної системи;
- високий рівень безпеки;
- велика кількість готових бібліотек;
- автоматичне керування пам’яттю.

Недоліки:

- нижча швидкодія порівняно з C++;
- значне споживання оперативної пам’яті;
- складніший процес створення настільних інтерфейсів.

Java добре підходить для великих корпоративних систем, проте для компактних локальних програм її використання може бути надмірним.

Мова програмування Python

Python є мовою програмування високого рівня, яка відзначається простотою синтаксису та швидкістю розробки.

Переваги Python:

- простий та зрозумілий синтаксис;
- швидка розробка програм;
- велика кількість бібліотек;
- підтримка різних платформ.

Недоліки:

- нижча продуктивність;
- менша швидкість виконання програм;
- не завжди зручний для створення складних настільних програм.

Python часто використовується у веброзробці, аналізі даних та автоматизації, однак для створення класичних настільних програм може поступатися іншим мовам.

Мова програмування JavaScript

JavaScript переважно використовується для створення вебзастосунків та інтерактивних вебінтерфейсів.

Переваги:

- підтримка всіма сучасними браузерами;
- велика кількість фреймворків;
- можливість створення вебсервісів.

Недоліки:

- орієнтація переважно на веброзробку;
- нижчий рівень безпеки у клієнтських застосунках;
- складність створення локальних настільних програм без додаткових технологій.

Для створення локального програмного забезпечення обліку успішності JavaScript використовується рідше.

Мова програмування C#

C# є сучасною об’єктно-орієнтованою мовою програмування, розробленою компанією Microsoft. Вона широко використовується для створення настільних, веб- та корпоративних застосунків.

Переваги C#:

- зрозумілий синтаксис;
- підтримка об’єктно-орієнтованого програмування;
- зручні засоби створення графічного інтерфейсу;
- інтеграція з платформою .NET;
- простота роботи з базами даних;
- автоматичне керування пам’яттю;
- висока швидкість розробки.

Недоліки:

- залежність від платформи .NET;
- дещо більше використання ресурсів порівняно з C++.

Для розробки програмного забезпечення розрахунку успішності студентів мова C# є оптимальним вибором, оскільки дозволяє швидко створити зручний

інтерфейс користувача, забезпечує просту інтеграцію з базами даних та підтримує сучасні методи програмування.

Порівняння систем керування базами даних

Для зберігання інформації у програмному забезпеченні використовуються системи керування базами даних. Найбільш поширеними СКБД є:

- MySQL;
- PostgreSQL;
- Microsoft SQL Server;
- Oracle;
- SQLite.

MySQL

MySQL є однією з найвідоміших реляційних систем керування базами даних із відкритим кодом.

Переваги:

- висока популярність;
- простота використання;
- підтримка багатьох платформ;
- хороша продуктивність.

Недоліки:

- обмежені можливості у складних корпоративних системах;
- менша функціональність порівняно з PostgreSQL.

MySQL часто використовується у веброзробці та невеликих інформаційних системах.

PostgreSQL

PostgreSQL - потужна об’єктно-реляційна система керування базами даних.

Переваги:

- висока надійність;
- підтримка складних запитів;
- розширені можливості роботи з даними;
- відповідність стандартам SQL.

Недоліки:

- складніше адміністрування;
- вищі вимоги до ресурсів.

PostgreSQL добре підходить для великих систем із великим навантаженням.

Microsoft SQL Server

Microsoft SQL Server є комерційною СКБД компанії Microsoft.

Переваги:

- висока продуктивність;
- потужні засоби адміністрування;
- інтеграція з продуктами Microsoft;
- високий рівень безпеки.

Недоліки:

- платна ліцензія;
- значні системні вимоги.

Ця система використовується переважно у великих корпоративних проєктах.

Oracle Database

Oracle Corporation Oracle Database є однією з найпотужніших комерційних СКБД.

Переваги:

- дуже висока надійність;
- підтримка великих обсягів даних;
- розвинені механізми безпеки.

Недоліки:

- висока вартість;
- складність адміністрування;
- значні вимоги до обладнання.

Oracle використовується переважно у великих корпоративних інформаційних системах.

SQLite

SQLite є легкою вбудованою системою керування базами даних, яка не потребує встановлення окремого серверного програмного забезпечення.

Переваги SQLite:

- компактність;
- простота використання;
- висока швидкість роботи;
- відсутність необхідності у сервері;
- збереження всієї бази даних в одному файлі;
- мінімальні системні вимоги;
- безкоштовне використання.

Недоліки:

- обмеження при великій кількості одночасних користувачів;
- менша масштабованість порівняно з серверними СКБД.

SQLite добре підходить для невеликих локальних програм, настільних застосунків та навчальних проєктів.

1.5 SQLite, система управління базами даних

SQLite - це реляційна система керування базами даних, яка призначена для зберігання, обробки та керування інформацією у компактному форматі. Вона є однією з найпоширеніших вбудованих систем керування базами даних і використовується у великій кількості програмних продуктів, мобільних застосунків та настільних програм.

Особливістю SQLite є те, що вона не потребує встановлення окремого серверного програмного забезпечення. На відміну від серверних СКБД, таких як MySQL або PostgreSQL, SQLite працює без окремого процесу сервера. Уся база даних зберігається у вигляді одного файлу, що значно спрощує використання та адміністрування системи.

SQLite була створена у 2000 році програмістом D. Richard Hipp. Основною метою розробки було створення легкої та швидкої системи керування базами даних, яка могла б використовуватись без складного налаштування. Завдяки своїй

простоті та надійності SQLite швидко стала популярною серед розробників програмного забезпечення.

SQLite підтримує стандартну мову SQL, яка використовується для створення таблиць, додавання, редагування, видалення та пошуку даних. За допомогою SQL-запитів можна виконувати різноманітні операції з інформацією, забезпечуючи швидкий доступ до необхідних даних.

Однією з найважливіших особливостей SQLite є її вбудованість у програму. База даних функціонує як частина застосунку, а не як окрема служба. Це дозволяє значно спростити розробку локальних програм та уникнути складного адміністрування серверів баз даних.

SQLite широко використовується у мобільних операційних системах, зокрема Android та iOS, а також у браузерах, вбудованих системах, навчальних проєктах і настільних застосунках. Завдяки невеликому розміру та мінімальному споживанню ресурсів вона добре підходить для програм, які працюють на персональних комп’ютерах або мобільних пристроях.

У SQLite дані зберігаються у таблицях, які складаються з рядків і стовпців. Кожна таблиця містить записи певного типу, а кожен стовпець має власний тип даних. Для організації зв’язків між таблицями використовуються первинні та зовнішні ключі.

SQLite підтримує основні типи даних:

INTEGER - цілі числа;

REAL - числа з плаваючою комою;

TEXT - текстові значення;

BLOB - двійкові дані;

NULL - порожні значення.

Для роботи з базою даних SQLite використовуються SQL-запити. Основними командами є:

CREATE TABLE - створення таблиці;

INSERT INTO - додавання даних;

SELECT - вибірка інформації;

UPDATE - оновлення записів;

DELETE - видалення даних.

```
CREATE TABLE Students (
    Id INTEGER PRIMARY KEY,
    Name TEXT,
    GroupName TEXT,
    AverageScore REAL
);
```

Рис. 1.5.1 Приклад створення таблиці студентів у SQLite

У наведеному прикладі створюється таблиця Students, яка містить інформацію про студентів: ідентифікатор, ім’я, групу та середній бал.

SQLite підтримує механізми забезпечення цілісності даних, що дозволяє уникати помилок та дублювання інформації. Для цього використовуються:

- первинні ключі;
- зовнішні ключі;
- обмеження унікальності;
- перевірки правильності введених даних.

Ще однією важливою перевагою SQLite є підтримка транзакцій. Транзакція - це набір операцій, які виконуються як єдине ціле. Якщо під час виконання виникає помилка, зміни можуть бути скасовані, що забезпечує надійність роботи з даними.

SQLite підтримує такі властивості транзакцій:

- атомарність;
- узгодженість;
- ізольованість;
- довговічність.

Ці властивості гарантують правильність і безпечність роботи бази даних навіть у випадку збоїв або помилок.

Для роботи з SQLite у мові програмування C# використовуються спеціальні бібліотеки, наприклад System.Data.SQLite або Microsoft.Data.Sqlite. Вони дозволяють підключатися до бази даних, виконувати SQL-запити та обробляти результати.

Перевагою використання SQLite разом із C# є простота інтеграції та швидкість розробки програмного забезпечення. База даних легко підключається до проєкту, а всі необхідні дані можуть зберігатися у локальному файлі.

Незважаючи на значну кількість переваг, SQLite має певні обмеження. Вона не призначена для великих багатокористувацьких систем із високим навантаженням. При одночасній роботі великої кількості користувачів продуктивність системи може знижуватись. Також SQLite має менше можливостей адміністрування порівняно з серверними СКБД.

Однак для локальних настільних програм, навчальних проєктів та невеликих інформаційних систем SQLite є одним із найкращих рішень. Вона забезпечує швидке зберігання та обробку інформації, не потребує складного налаштування та дозволяє створювати компактні й ефективні програмні продукти.

У даній дипломній роботі SQLite використовується для зберігання інформації про студентів, навчальні дисципліни, оцінки та результати розрахунку успішності. Використання цієї системи керування базами даних дозволяє забезпечити надійне збереження інформації, швидкий доступ до даних та простоту роботи програмного забезпечення.

1.6 Висновки до теоретичної частини

У теоретичній частині дипломної роботи було розглянуто основні поняття, пов’язані з базами даних, системами керування базами даних та сучасними засобами розробки програмного забезпечення. Проведений аналіз показав, що використання баз даних є важливою складовою сучасних інформаційних систем, оскільки вони забезпечують ефективне зберігання, обробку та пошук інформації.

Було досліджено основні типи баз даних, їх структуру та особливості використання. Особливу увагу приділено реляційним базам даних, які є найбільш поширеним типом організації даних у сучасних програмних системах. Також було розглянуто принципи роботи систем керування базами даних, їх функції, переваги та роль у забезпеченні цілісності, безпеки та надійності інформації.

У процесі виконання теоретичного аналізу було проведено порівняння сучасних мов програмування та систем керування базами даних. За результатами порівняння для реалізації програмного забезпечення було обрано мову програмування C# та систему керування базами даних SQLite. Вибір C# обумовлений її зручністю, підтримкою об’єктно-орієнтованого програмування, можливістю швидкої розробки графічного інтерфейсу та простою інтеграцією з базами даних. SQLite, у свою чергу, є компактною та легкою СКБД, яка не потребує встановлення окремого серверного програмного забезпечення та забезпечує швидке зберігання і обробку даних.

Також було розглянуто особливості SQLite, її архітектуру, основні можливості та переваги використання у локальних настільних застосунках. Аналіз показав, що SQLite є ефективним рішенням для створення невеликих інформаційних систем та навчальних проєктів.

Результати теоретичного дослідження підтвердили доцільність використання мови програмування C# та системи керування базами даних SQLite для розробки програмного забезпечення розрахунку успішності студентів. Отримані теоретичні відомості стали основою для виконання практичної частини дипломної роботи та подальшої реалізації програмного продукту.

2. ПРАКТИЧНА ЧАСТИНА

2.1 Вибір інструментів для побудови проєкту

Обґрунтування вибору C# та SQLite

Вибір мови програмування C# для розробки програмного забезпечення обумовлений її сучасністю, підтримкою об’єктно-орієнтованого програмування, зручними засобами створення графічного інтерфейсу та простою інтеграцією з базами даних. Крім того, платформа .NET надає велику кількість бібліотек і

готових компонентів, що значно спрощує процес розробки програмного забезпечення. У якості системи керування базами даних було обрано SQLite, оскільки вона є легкою, компактною та не потребує встановлення окремого серверного програмного забезпечення. SQLite забезпечує швидке збереження й обробку даних, підтримує зберігання всієї бази даних в одному файлі та є зручною для використання у локальних настільних застосунках. Поєднання C# та SQLite дозволяє створити ефективне, надійне та зручне програмне забезпечення для автоматизації розрахунку успішності студентів.

Середу розробки (IDE) було обрано Visual Studio через зручність і вбудований функціонал Windows forms.

2.2 Побудова програми

Програма побудована як консольний застосунок на мові C# з графічним інтерфейсом на базі Windows Forms. Точкою входу є клас Program у файлі CalculatorUspishnosti.cs, який обробляє аргументи командного рядка та запускає відповідні операції або головне вікно.

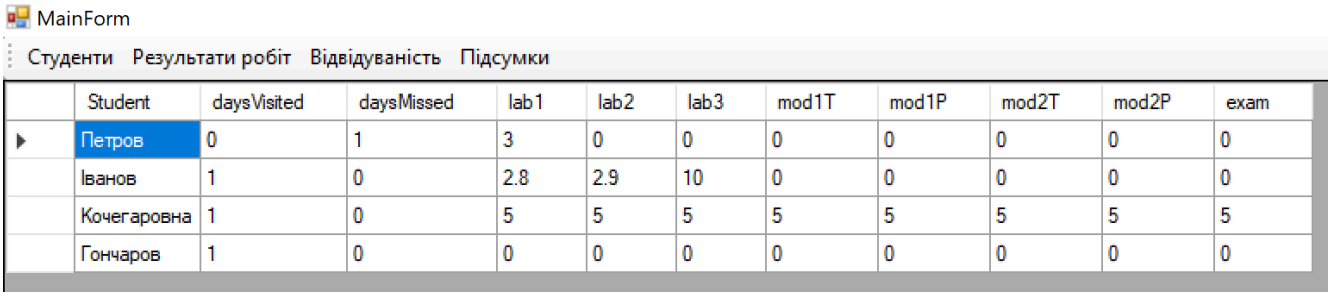
Структура бази даних. База даних SQLite складається з чотирьох основних таблиць: Student (студенти), Work (роботи/завдання), Result (журнал оцінок), Attendance (відвідуваність), а також службової таблиці Settings для зберігання налаштувань програми. Ініціалізація бази даних виконується командою -init, імпорт даних з CSV-файлів - командою -insert.

```
C:\Users\asada\Desktop\DyplomRodchenko>compile
**** local is here

C:\Users\asada\Desktop\DyplomRodchenko>csc /nologo /platform:x86   /target:exe /out:a.exe /unsafe /r:logger.cs.dll /r:ar
gs.dll /r:System.Data.SQLite.dll /r:System.Windows.Forms.dll /r:System.Drawing.dll  *.cs    /langversion:5

C:\Users\asada\Desktop\DyplomRodchenko>a database.db -init

C:\Users\asada\Desktop\DyplomRodchenko>a database.db -insert
lab1,3,y,n,n
lab2,3,y,n,n
lab3,3,y,n,n
mod1T,5,y,y,y
mod1P,5,y,y,y
mod2T,2,y,n,y
mod2P,2,y,n,y
exam,5,y,n,y
```

Рис. 2.2.1 Компіляція програми, створення і заповнення таблиць

Представлення **Summary**. Зведена таблиця успішності реалізована як динамічне SQL-представлення (VIEW), яке будується програмно методом

CreateView. Оскільки кількість робіт є змінною і залежить від вмісту таблиці Work, представлення перебудовується щоразу при запуску: для кожної роботи генерується окремий стовпець із підзапитом IFNULL(CAST(SUM(grade) AS REAL), 0). Стовпці daysVisited та daysMissed обчислюються автоматично з таблиці Attendance.



Student	daysVisited	daysMissed	lab1	lab2	lab3	mod1T	mod1P	mod2T	mod2P	exam
Петров	0	1	3	0	0	0	0	0	0	0
Іванов	1	0	2.8	2.9	10	0	0	0	0	0
Кочегаровна	1	0	5	5	5	5	5	5	5	5
Гончаров	1	0	0	0	0	0	0	0	0	0

Рис. 2.2.2 Таблиця Підсумки

Механізм редагування оцінок. Редагування реалізоване через тригер INSTEAD OF UPDATE ON Summary. При подвійному кліці на комірці зведеної таблиці відкривається діалогове вікно для введення нового значення. Програма обчислює дельту (різницю між новим і поточним значенням) і виконує UPDATE Summary, який перехоплюється тригером, що записує новий запис у таблицю Result. Таким чином зберігається повна історія всіх змін оцінок.

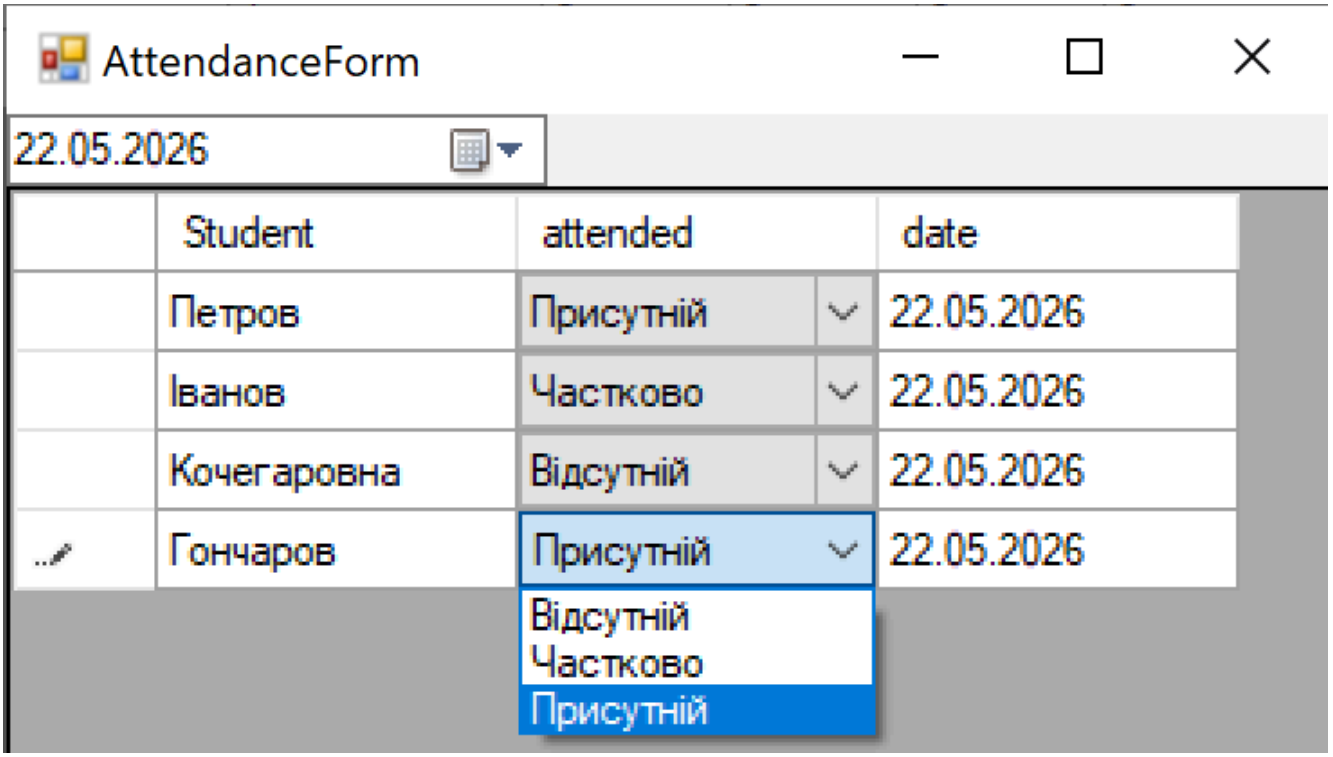
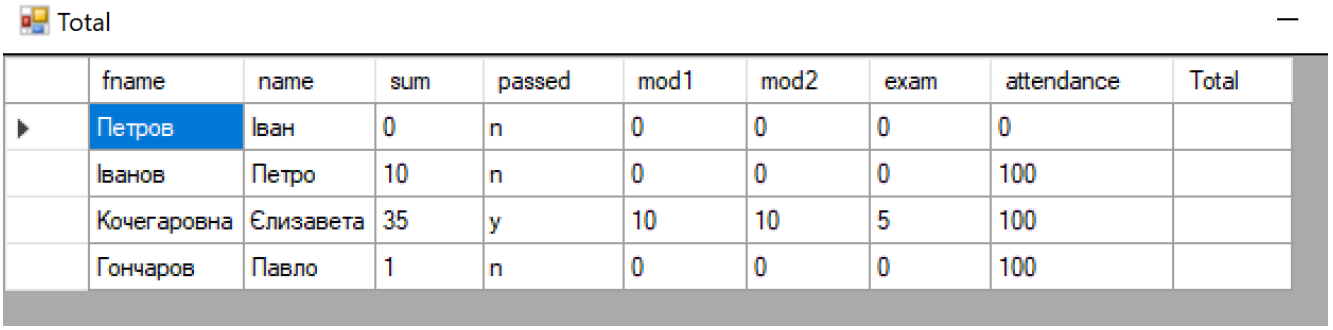


Рис. 2.2.3 Таблиця Відвідуваність

Облік відвідуваності. Форма AttendanceForm дозволяє переглядати і редагувати відвідуваність по датах через DateTimePicker. Кожен рядок містить спадний список зі значеннями «Присутній» (10), «Частково» (1) і «Відсутній» (0). При виборі дати автоматично створюються записи для всіх студентів, яких ще немає в таблиці на цю дату, зі значенням «Відсутній» за замовчуванням.



	fname	name	sum	passed	mod1	mod2	exam	attendance	Total
▶	Петров	Іван	0	n	0	0	0	0	
	Іванов	Петро	10	n	0	0	0	100	
	Кочегаровна	Єлизавета	35	y	10	10	5	100	
	Гончаров	Павло	1	n	0	0	0	100	

Рис. 2.2.4 Таблиця Підсумки

Розрахунок підсумків. Метод CreateTotal формує таблицю Total з підсумковими показниками для кожного студента: сума балів за роботи з позначкою includeInReport = 'y', бонус за відвідуваність із коефіцієнтом add (формула: bonus + bonus * add), відсоток відвідуваності, бали за модульні контролі та іспит, а також ознака допуску passed. Коефіцієнт add зберігається у таблиці Settings і задається через аргумент командного рядка -add.

```
C:\Users\asada\Desktop\DyplomRodchenko>a.exe .db -export
Exported to C:\Users\asada\Desktop\DyplomRodchenko\exported
```

Рис. 2.2.5 Команда експорту таблиць

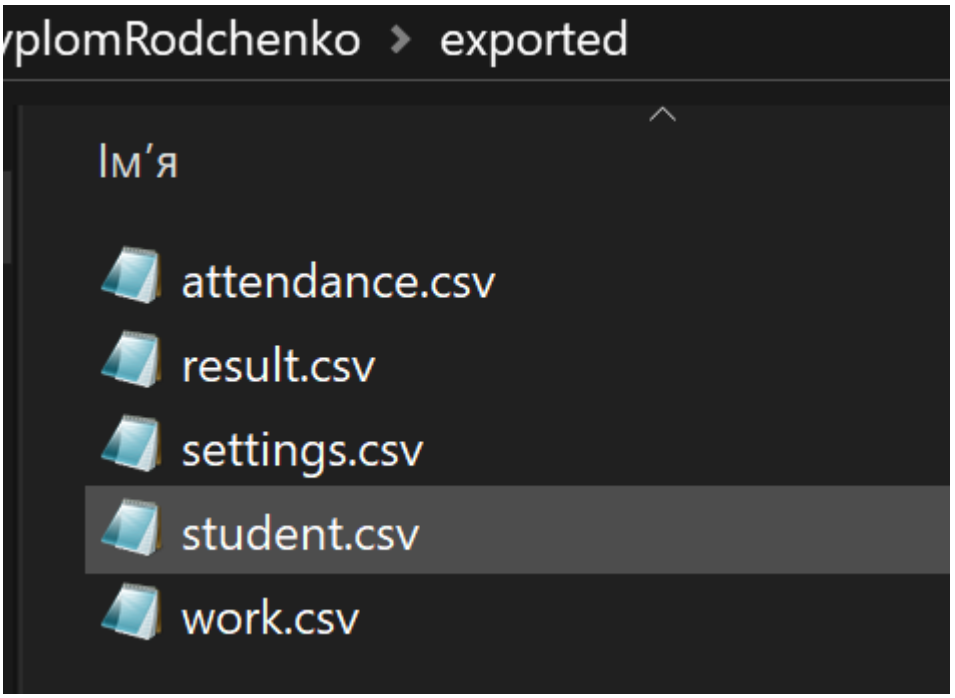


Рис. 2.2.6 Експортовані таблиці

```
C:\Users\asada\Desktop\DyplomRodchenko>a.exe .db -total
Exported to C:\Users\asada\Desktop\DyplomRodchenko\total.txt
```

Рис. 2.2.7 Команда для експорту підсумків

total.txt: Блокнот

Файл Редагування Формат Вигляд Довідка

fname	name	sum	passed	mod1	mod2	exam	attendance	Total
Петров	Іван	0	n	0	0	0	0	
Іванов	Петро	10	n	0	0	0	100	
Кочегаровна	Єлизавета	35	y	10	10	5	100	
Гончаров	Павло	1	n	0	0	0	100	

Рис. 2.2.8 Експортовані підсумки

Експорт даних. Програма підтримує експорт усіх таблиць у формат CSV командою -export. Дані відвідуваності при експорті перетворюються: замість числового studentId зберігається ім'я студента (fname), що забезпечує зручність повторного імпорту. Папка для експорту автоматично отримує унікальне ім'я (exported, exported_1, тощо) щоб уникнути перезапису попередніх даних.

Імпорт даних. Також, програма підтримує імпорт командою -insert [папка].

```
C:\Users\asada\Desktop\DyplomRodchenko>a .db -insert exported
lab1,3,y,n,n
lab2,3,y,n,n
lab3,3,y,n,n
mod1T,5,y,y,y
mod1P,5,y,y,y
mod2T,2,y,n,y
mod2P,2,y,n,y
exam,5,y,n,y
```

Рис. 2.2.9 Команда для імпорту даних

2.3 SchemaSpy, схема реляційних зв'язків

Для наочного відображення структури бази даних та реляційних зв'язків між таблицями було використано інструмент SchemaSpy - програмне забезпечення з відкритим кодом, що автоматично аналізує схему бази даних і генерує HTML-документацію з діаграмами зв'язків.

SchemaSpy підключається до бази даних через JDBC-драйвер і зчитує метадані таблиць, стовпців, первинних та зовнішніх ключів. Для роботи з SQLite використовується драйвер sqlite-jdbc, а для побудови графічних діаграм - система Graphviz.

Для генерації документації використовувалась наступна команда:

java -jar schemaspy-app.jar -t sqlite.properties

-dp sqlite-jdbc-3.53.1.0.jar

-db asd.db -o schemaspy_output

-u dummy -cat % -s % -vizjs -noads

де параметр -vizjs вказує на використання вбудованого JavaScript-рушія Graphviz замість зовнішнього виклику dot, що забезпечує коректну генерацію діаграм незалежно від налаштувань операційної системи.

На рисунку 21 зображена схема реляційних зв'язків між таблицями бази даних.

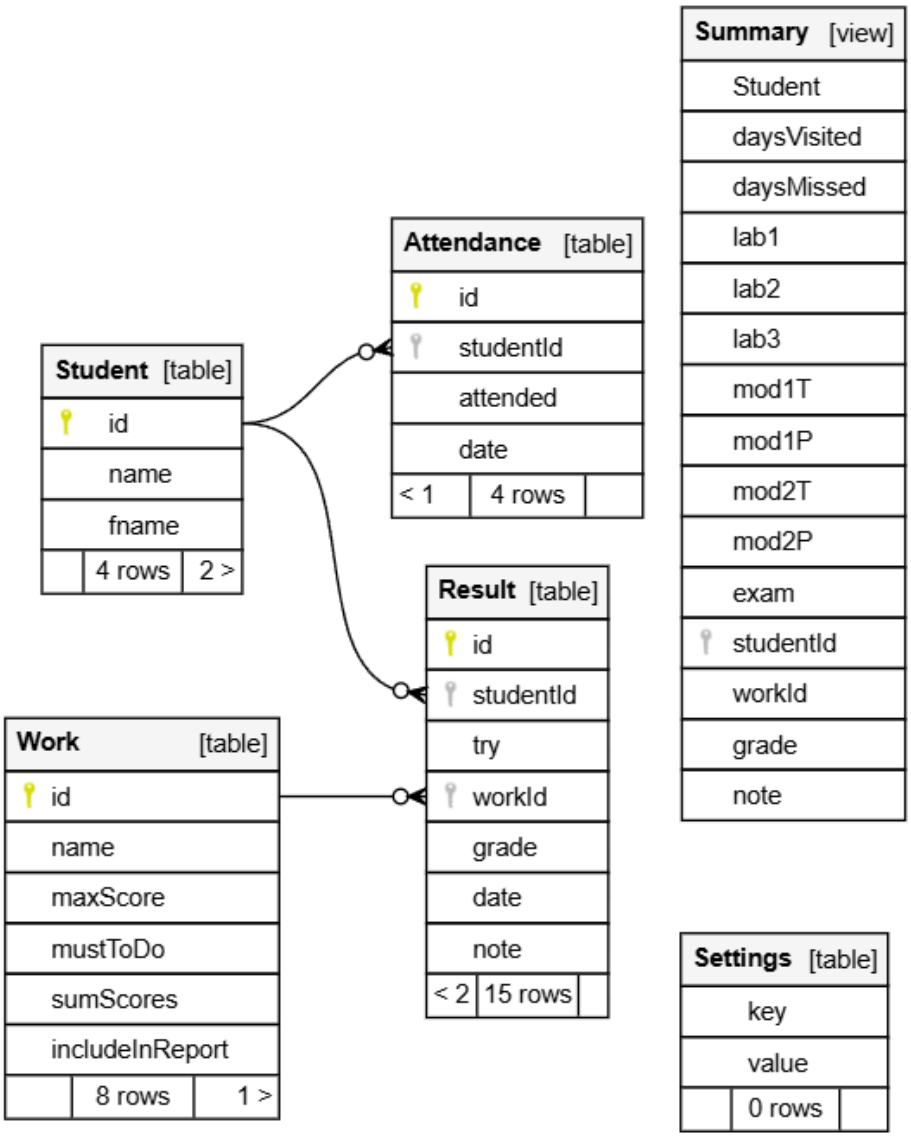


Рис. 2.3.1 Схема реляційних зв'язків бази даних

Як видно зі схеми, центральною сутністю є таблиця Student, з якою пов'язані таблиці Result та Attendance через зовнішній ключ studentId. Таблиця Result також має зовнішній ключ workId, що посилається на таблицю Work. Таблиця Settings є незалежною і зберігає службові налаштування програми у форматі «ключ–значення». Представлення Summary побудоване на основі таблиць Student, Work та Result і не має власних зовнішніх ключів, оскільки є обчислюваним об'єктом.

Така структура забезпечує нормалізацію даних, уникнення дублювання інформації та цілісність записів завдяки обмеженням зовнішніх ключів (REFERENCES).

2.4 Doxygen, схема наслідування класів

Для документування вихідного коду програми та побудови схеми наслідування класів було використано інструмент Doxygen - широко розповсюджений генератор документації, що підтримує мови програмування C#, C++, Java та інші. Doxygen аналізує вихідний код, зчитує коментарі у спеціальному форматі та автоматично генерує HTML або PDF документацію з діаграмами класів, їх ієрархії та взаємозв'язків.

Для побудови графічних діаграм Doxygen використовує систему Graphviz, яка генерує схеми наслідування на основі зібраної інформації про класи та їх зв'язки.

Програма складається з таких основних класів:

Program - головний клас застосунку, що містить точку входу Main, обробляє аргументи командного рядка та реалізує методи роботи з базою даних: CreateView, CreateTotal, Export, GetCoefficient, SetCoefficient та інші;

MainForm - головна форма застосунку, що відображає зведену таблицю успішності Summary і надає доступ до інших вікон через панель інструментів;

ResultForm - форма перегляду таблиці Result з повною історією всіх оцінок;

StudentForm - форма перегляду списку студентів з таблиці Student;

AttendanceForm - форма обліку відвідуваності з вибором дати через DateTimePicker та редагуванням значень через спадний список;

TotalForm - форма відображення підсумкової таблиці Total з можливістю ручного редагування стовпця Total.

Усі форми є частковими класами (partial class) і наслідують базовий клас Form із бібліотеки System.Windows.Forms.

На рисунку 22 зображена схема наслідування одного з класів програми.

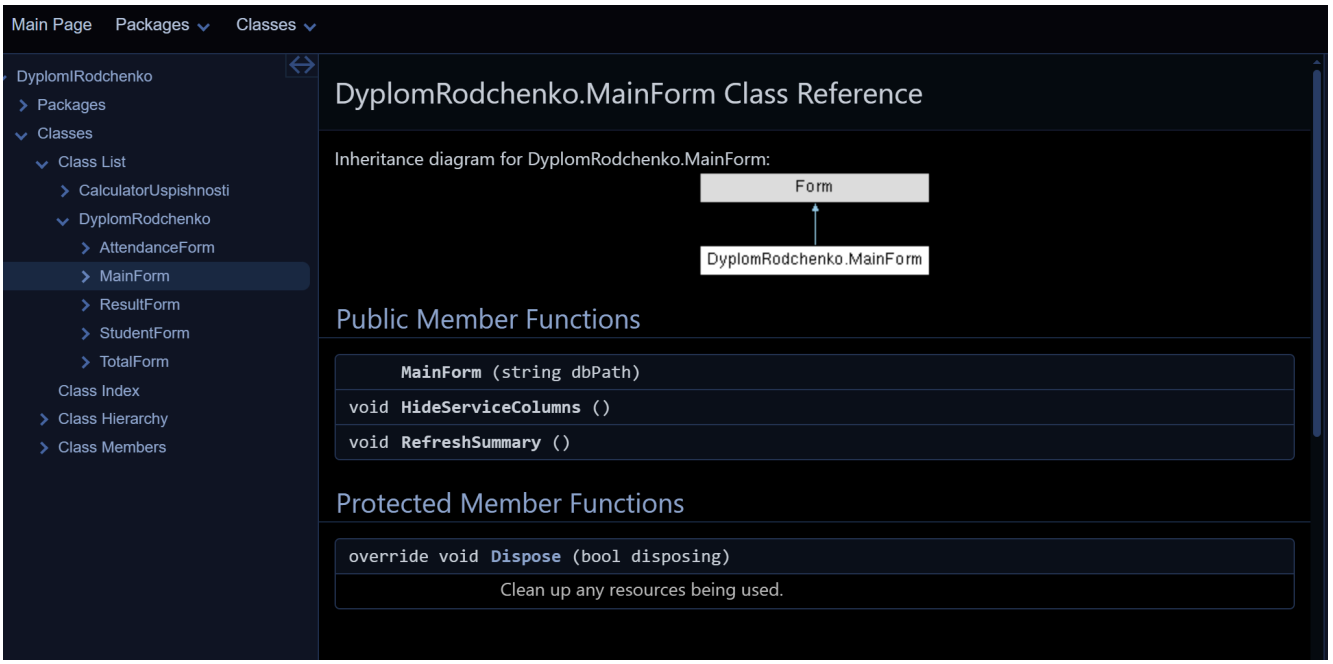


Рис. 2.4.1 Схема наслідування одного з класів програми

Як видно зі схеми, всі форми застосунку безпосередньо наслідують клас Form, що є стандартною архітектурою для Windows Forms застосунків. Клас Program є незалежним і не входить до ієрархії форм, виконуючи роль контролера між інтерфейсом користувача та базою даних.

Більш докладна документація доступна в папці з програмою.

2.5 Висновки до практичної частини

У практичній частині роботи було розроблено програмне забезпечення для розрахунку успішності студентів із використанням мови програмування C# та системи управління базами даних SQLite.

В результаті виконання практичної частини було досягнуто таких результатів:

Спроектовано та реалізовано структуру бази даних, що складається з таблиць Student, Work, Result, Attendance та Settings. Структура забезпечує нормалізацію даних, цілісність записів через зовнішні ключі та гнучкість при розширенні функціональності.

Реалізовано динамічне SQL-представлення Summary, що автоматично формує зведену таблицю успішності залежно від поточного складу робіт у базі

даних. Представлення перебудовується при кожному запуску програми, що дозволяє коректно відображати нові роботи без змін у коді.

Розроблено механізм редагування оцінок на основі тригера INSTEAD OF UPDATE, який перехоплює зміни у представленні Summary і автоматично записує дельту оцінки до таблиці Result. Завдяки цьому зберігається повна історія всіх змін із зазначенням дати та коментаря.

Реалізовано систему обліку відвідуваності з можливістю вибору дати та редагування статусу кожного студента через зручний спадний список. Програма автоматично заповнює записи для всіх студентів при виборі нової дати, встановлюючи значення «Відсутній» за замовчуванням.

Розроблено модуль розрахунку підсумкових показників, який формує таблицю Total із сумою балів за семестр, бонусом за відвідуваність із налаштовуваним коефіцієнтом, відсотком відвідуваності, балами за модульні контролі та іспит, а також ознакою допуску студента до заліку.

Реалізовано функціонал імпорту та експорту даних у форматі CSV, що забезпечує зручне перенесення інформації між різними екземплярами програми та можливість резервного копіювання даних.

Для документування структури бази даних використано інструмент SchemaSpy, який автоматично генерує HTML-документацію зі схемою реляційних зв'язків. Для документування вихідного коду використано Doxygen, що дозволяє наочно відобразити ієрархію класів програми.

Розроблена програма відповідає поставленим вимогам та може бути використана у навчальному процесі для автоматизації обліку успішності студентів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено програмне забезпечення для розрахунку успішності студентів із використанням мови програмування C# та системи управління базами даних SQLite. Поставлена мета роботи була досягнута в повному обсязі.

У теоретичній частині було проаналізовано сучасні підходи до розробки програмного забезпечення, розглянуто типи баз даних та системи керування базами даних, досліджено можливості SQLite як вбудованої реляційної СКБД. Також було розглянуто методи обробки виняткових ситуацій та підходи до тестування програмного забезпечення, що дозволило обґрунтувати вибір інструментів для реалізації проєкту.

У практичній частині було виконано такі завдання:

- розроблено реляційну структуру бази даних, що включає таблиці для зберігання інформації про студентів, роботи, результати та відвідуваність;
- реалізовано динамічне представлення зведеної таблиці успішності, що автоматично адаптується до змін у складі навчальних робіт;
- розроблено механізм журналювання змін оцінок на основі тригера INSTEAD OF UPDATE, що забезпечує збереження повної історії коригувань;
- створено зручний графічний інтерфейс користувача з кількома вікнами для перегляду та редагування даних;
- реалізовано систему обліку відвідуваності з автоматичним заповненням записів та зручним редагуванням через спадний список;
- розроблено модуль розрахунку підсумкових показників успішності з урахуванням бонусу за відвідуваність та налаштовуваного коефіцієнту;
- реалізовано функціонал імпорту та експорту даних у форматі CSV для забезпечення зручного перенесення та резервного копіювання інформації;
- задокументовано структуру бази даних за допомогою SchemaSpy та ієрархію класів за допомогою Doxygen.

Практична цінність розробленого програмного забезпечення полягає в тому, що воно може бути безпосередньо використане у навчальних закладах для

автоматизації процесу обліку успішності студентів. Програма скорочує час на обробку даних, мінімізує ймовірність помилок при розрахунках та забезпечує надійне зберігання інформації.

Отримані результати підтверджують ефективність використання поєднання мови програмування C# та системи управління базами даних SQLite для створення компактного, функціонального та зручного у використанні програмного забезпечення. Розроблена програма може слугувати основою для подальшого вдосконалення та розширення функціональних можливостей, зокрема додавання веб-інтерфейсу, розширеної системи звітності або інтеграції з іншими інформаційними системами навчального закладу.

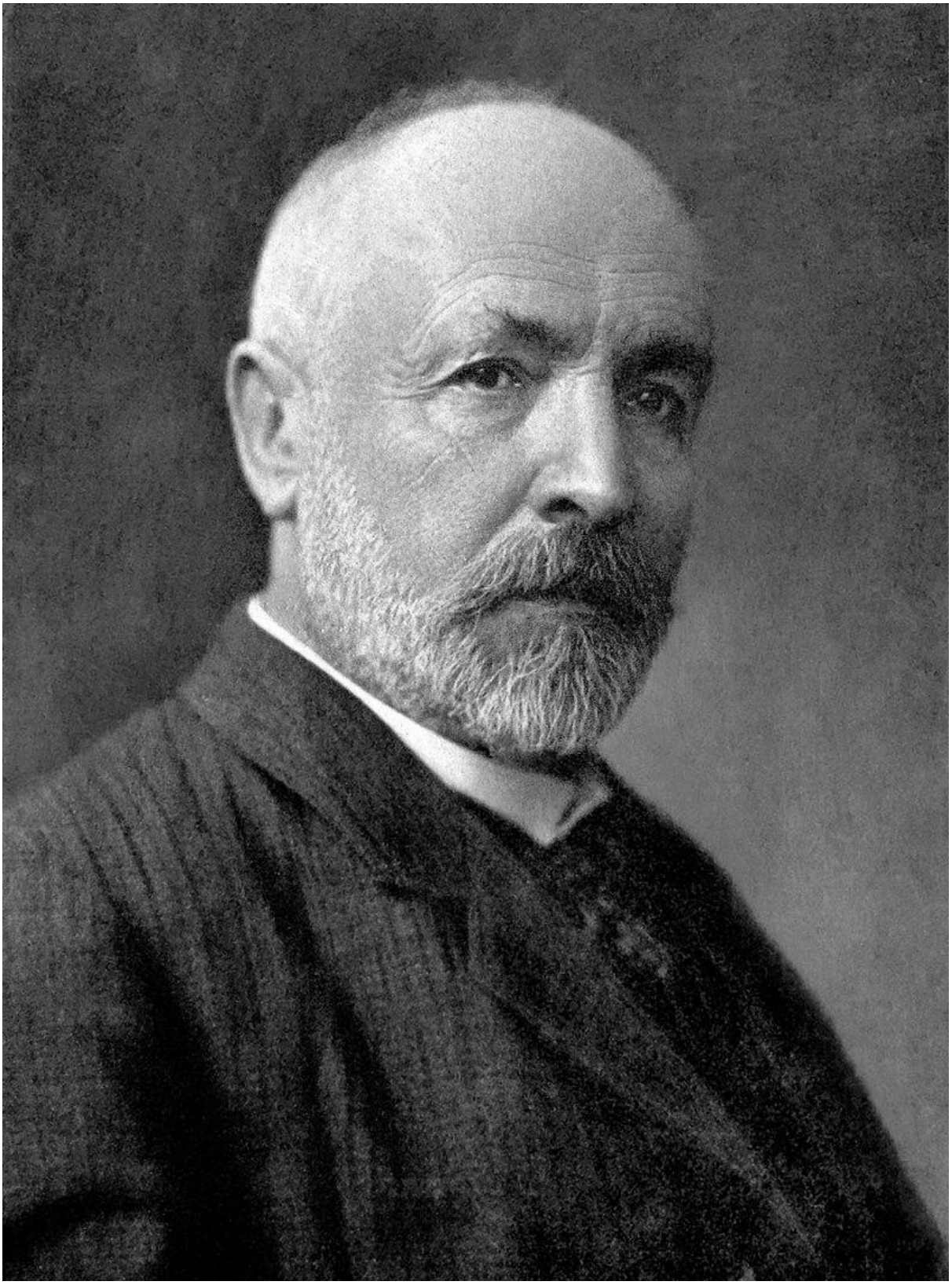
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Розробка клієнт-серверних додатків засобами мови C# та їхнє документування / О.Г.Піскунов, Л.В.Єсик, О.П.Томащук, А.К.Жултинська, Д.М.Яременко // [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/380290412 – Дата доступу: 15.05.2026.
2. Розробка додатків засобами мови програмування C# / О.Г.Піскунов, E.V.Ivohin, M.M.Makhno // К.: Видавничо-поліграфічний центр 'Київський університет // [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/354860614– Дата доступу: 15.05.2026.
3. Troelsen A. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming / A. Troelsen, P. Japikse. – New York.: Apress, 2022. – 1310 p.
4. Price M. C# 10 and .NET 6 – Modern Cross-Platform Development / M. Price. – Birmingham.: Packt Publishing, 2022. – 823 p.
5. MacDonald M. Pro WinForms with C# / M. MacDonald. – New York.: Apress, 2023. – 740 p.
6. Griffiths I. Programming C# 10 / I. Griffiths. – Sebastopol.: O'Reilly Media, 2022. – 760 p.
7. Kreibich J. Using SQLite / J. Kreibich. – Sebastopol.: O'Reilly Media, 2010. – 530 p.
8. Date C. J. An Introduction to Database Systems / C. J. Date. – Boston.: Addison-Wesley Professional, 2003. – 1024 p.
9. Hernandez M. Database Design for Mere Mortals / M. Hernandez. – Boston.: Addison-Wesley Professional, 2020. – 672 p.
10. Документація SQLite. [Електронний ресурс]. – Режим доступу: https://www.sqlite.org/docs.html – Дата доступу: 15.05.2026.
11. Документація мови C# від Microsoft. [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/uk-ua/dotnet/csharp/ – Дата доступу: 15.05.2026.

12. Документація Windows Forms від Microsoft. [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/uk-ua/dotnet/desktop/winforms/ – Дата доступу: 15.05.2026.
13. Документація System.Data.SQLite. [Електронний ресурс]. – Режим доступу: https://system.data.sqlite.org/index.html/doc/trunk/www/index.wiki – Дата доступу: 15.05.2026.
14. Документація SchemaSpy. [Електронний ресурс]. – Режим доступу: https://schemaspy.org/ – Дата доступу: 15.05.2026.
15. Документація Doxygen. [Електронний ресурс]. – Режим доступу: https://www.doxygen.nl/manual/index.html – Дата доступу: 15.05.2026.
16. Документація Graphviz. [Електронний ресурс]. – Режим доступу: https://graphviz.org/documentation/ – Дата доступу: 15.05.2026.
17. SQLite Tutorial. [Електронний ресурс]. – Режим доступу: https://www.sqlitetutorial.net/ – Дата доступу: 15.05.2026.
18. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. – Boston.: Addison-Wesley Professional, 2002. – 560 p.
19. Martin R. Clean Code: A Handbook of Agile Software Craftsmanship / R. Martin. – Boston.: Prentice Hall, 2008. – 464 p.
20. Codd E. F. A Relational Model of Data for Large Shared Data Banks / E. F. Codd. – Communications of the ACM, 1970. – Vol. 13, No. 6. – P. 377–387.
21. Стандарт ISO/IEC 9075:2023 — Мова баз даних SQL. [Електронний ресурс]. – Режим доступу: https://www.iso.org/standard/76583.html – Дата доступу: 15.05.2026.
22. Документація Xerial sqlite-jdbc. [Електронний ресурс]. – Режим доступу: https://github.com/xerial/sqlite-jdbc – Дата доступу: 15.05.2026.

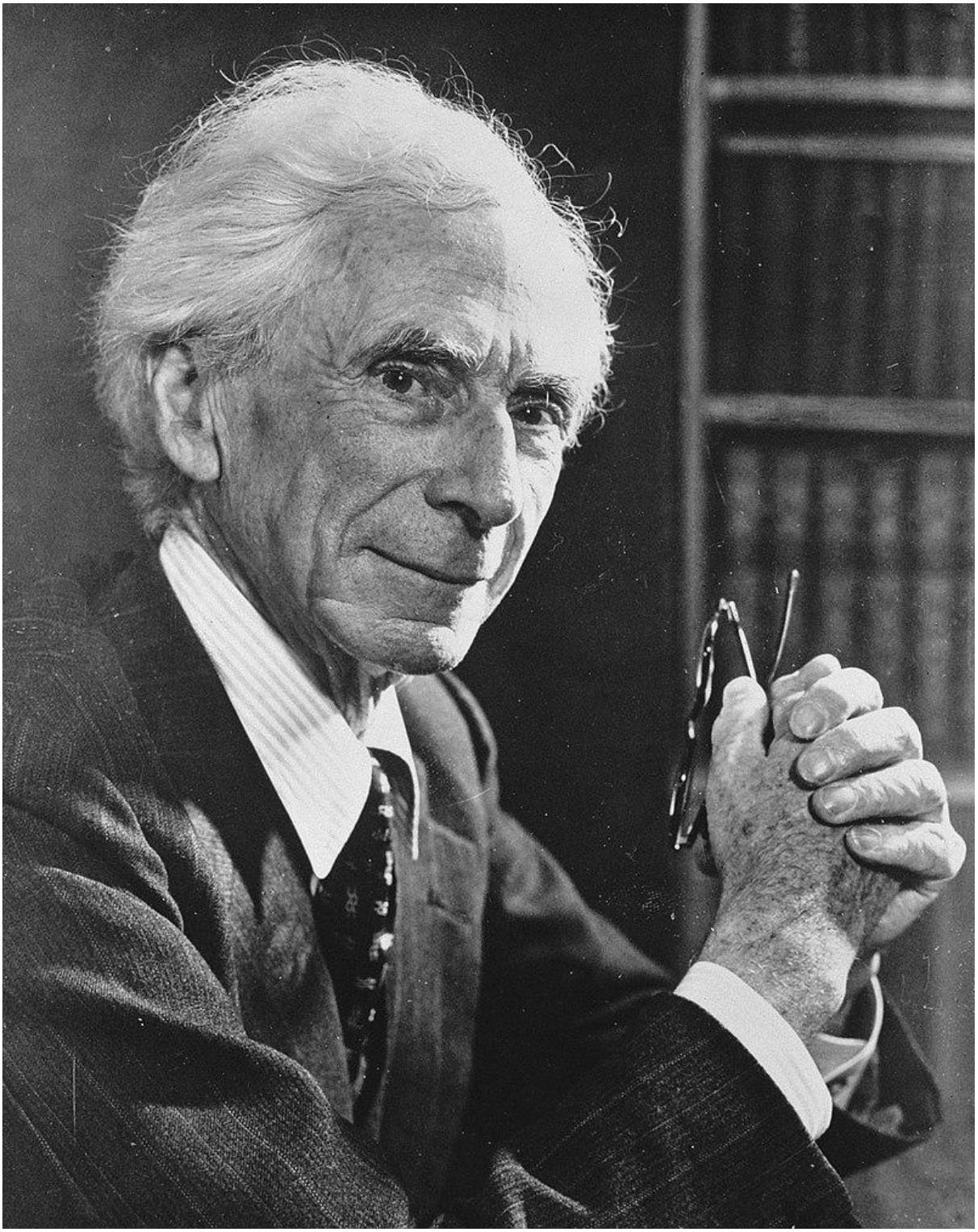
ДОДАТКИ

Додаток 1



Портрет Георга Кантора

Додаток 2



Портрет Бертрана Рассела

Додаток 3

```
      // If the Start button is clicked
onView(withText("Start"))
    .perform(click())
      // Then display the Hello message
onView(withText("Hello"))
    .check(matches(isDisplayed()))
```

Додаток 4

```
      // If given a ViewModel1 instance
val viewModel = ViewModel1(ExampleDataRepository)
      // After loading data
viewModel.loadData()
      // Expose data
assertTrue(viewModel.data != null)
```

Додаток 5

```
      beforeEach(async(() => {
   TestBed.configureTestingModule({
      declarations: [
         AppComponent
      ],
   }).compileComponents();
}));
```

Додаток 6

```
      it('creates the app', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
}));
```

Додаток 7

```
      it(`title should be 'angular-unit-test'`, async(() => {
     const fixture = TestBed.createComponent(AppComponent);
     const app = fixture.debugElement.componentInstance;
```

```
     expect(app.title).toEqual('angular-unit-test');
}));
```

Додаток 8

```
      it('render title in h1 tag', async(() => {
   const fixture = TestBed.createComponent(AppComponent);
   fixture.detectChanges();
   const compiled = fixture.debugElement.nativeElement;
 expect(compiled.querySelector('h1').textContent).toContain('Welcome to
angular-unit-test!');
}));
```

Додаток 9

```
      const {describe} = require('mocha');
const assert = require('assert');

describe('Simple test suite:', function() {
    it('1 === 1 should be true', function() {
        assert(1 === 1);
    });
});
```

Додаток 10

```
      $ cd src/projects/IBM-Developer/Node.js/Course/Unit-9
$ ./node_modules/.bin/mocha test/example1.js
       Simple test suite:
    ✓ 1 === 1 should be true
      1 passing (5ms)
```

Додаток 11

```
      const ExampleSum = require('./ExampleSum');
      test('ExampleSum equals 3', () => {
       expect(ExampleSum(1, 2).toBe(3);
});
```

Додаток 12

```
      describe('My First Test', () => {
  it('Gets, types and asserts', () => {
    cy.visit('https://example.cypress.io')
```

```
    cy.contains('type').click()

    // Should be on a new URL which
    // includes '/commands/actions'
    cy.url().should('include', '/commands/actions')

    // Get an input, type into it
    cy.get('.action-email').type('fake@email.com')

    //  Verify that the value has been updated
    cy.get('.action-email').should('have.value', 'fake@email.com')
  })
})
```