НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук

_____/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення для наукових та інженерних розрахунків»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми

_____к.т.н., доцент_____ / Вайганг Г.О./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи :_____ / Бородкін Г.О./
підпис ПІБ

Виконав:_____ / Мельник Б.І./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук
_____ / Голуб Б.Л., доцент, к.т.н /
підпис
" " 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи студенту
Мельнику Богдану Івановичу
Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: «Програмний додаток для наукових та інженерних розрахунків»

Затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . ___. ____
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення, навчальна та методична література, державні стандарти

4. Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Керівник бакалаврської кваліфікаційної роботи_____ / Бородкін Г.О. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____ / Мельник Б.І./
підпис ініціали та прізвище

Дата отримання завдання 2025 .__.____
рік, місяць, число

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SQL - Декларативна мова запитів для реляційних СУБД

ПЗ - Програмне забезпечення

БД - База даних

ОВіК - Опалення, вентиляція і кондиціювання повітря

SQLite - Lightweight SQL Database Engine (легка реляційна СУБД)

CFD - Computational Fluid Dynamics (обчислювальна гідродинаміка)

Qt - Бібліотека інтерфейсів для кросплатформенного ПЗ

PyQt5 - Python bindings for Qt5 (обгортка Qt5 для Python)

UML - Уніфікована мова моделювання

ВСТУП

У сучасних умовах підвищених вимог до енергоефективності будівель, комфорту середовища перебування та безпеки людини все більшого значення набувають інженерні розрахунки, пов’язані з рухом повітря у приміщеннях. Оптимальне проектування вентиляційних систем, аналіз циркуляції повітря, виявлення зон застою потребують точних розрахунків та ефективних засобів моделювання.

Застосування спеціалізованого програмного додатку для проведення наукових та інженерних розрахунків дозволяє значно скоротити час і ресурси, необхідні для проектування систем вентиляції, а також підвищити точність і достовірність результатів. Особливо актуальним є створення простих у використанні, але функціонально потужних інструментів, які можуть застосовуватись як фахівцями з ОВіК так і приватними забудовниками або дослідниками.

Об'єктом дослідження в даній роботі є процес розрахунку моделювання повітряних потоків у замкнених приміщеннях. Предметом дослідження — методи та засоби створення програмного додатку для спрощеного аналізу циркуляції повітря на основі геометричних даних про приміщення.

Метою цієї бакалаврської кваліфікаційної роботи є розробка інформаційної системи, що дозволяє користувачам створювати модель приміщення, задавати параметри стін, вікон, отворів і зовнішніх умов, а також візуалізувати напрямки та інтенсивність повітряних потоків у спрощеному вигляді.

Для досягнення поставленої мети в роботі вирішуються наступні завдання:

- проаналізувати предметну область, актуальні потреби користувачів і наявні програмні рішення;

- визначити вимоги до функціональності, інтерфейсу та структури майбутнього ПЗ;
- розробити архітектуру програми, що включає графічний інтерфейс користувача, модель геометрії приміщення та алгоритми розрахунків;
- реалізувати програмне забезпечення з використанням мови Python, бібліотеки PyQt5 та засобів збереження даних (SQLite);
- провести тестування основного функціоналу та оцінити ефективність системи;
- надати рекомендації щодо подальшого розвитку проекту.

Практичне значення отриманих результатів полягає у створенні програмного інструменту, який може використовуватись для попередньої оцінки повітрообміну у приміщеннях на ранніх етапах проектування, у навчальному процесі або для демонстраційних цілей. Система забезпечує базовий рівень моделювання без потреби у складному обладнанні чи глибоких знаннях з аеродинаміки, що розширює коло її потенційних користувачів.

1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Об'єктом дослідження в цьому контексті є програмний комплекс, призначений для наукових та інженерних розрахунків, з функціоналом моделювання циркуляції повітря в приміщенні , яке створює користувач за потрібними йому параметрами та з можливість візуалізації в трьох вимірній графіці тобто в 2.5D. Ця система дозволяє користувачам моделювати та аналізувати потоки повітря в заданій конфігурації приміщення з урахуванням різних архітектурних та екологічних параметрів. Кінцевою метою є створення інструменту, який допоможе інженерам, архітекторам та будівельникам у розробці ефективних вентиляційних рішень, що відповідають нормативним стандартам.

1.2 Аналіз вимог до програмної системи

1.2.1 Функціональні вимоги

Функціональні вимоги - це певний набір можливостей, що характеризують те, що повинна вирішувати програмна система, що вона повинна робити та як себе поводити. Це сформульовані дії, які користувач може виконувати у системі, а вона в свою чергу повинна правильно на них реагувати. Також функціональні вимоги визначають те, як система повинна взаємодіяти з іншими системами, підсистемами або модулями. Функціональні вимоги також включають вимоги до основних бізнес-процесів, алгоритмів, логіки обробки даних та механізму взаємодії з іншими програмними компонентами.

Функціональні вимоги не лише вказують на унікальні потреби користувачів до програмної системи, але й на найменші деталі її роботи, що є дуже важливо для досягнення поставлених цілей. Вони описують, як система повинна відреагувати на певні дії користувача, які функції мають бути доступні для виконання тих чи інших завдань, які вимоги до взаємодії між різними модулями, частинами системи та як платформа має поводити себе в різних

ситуаціях. Це дозволить створити представлення того, як система буде функціонувати, як буде відбуватися взаємодія з користувачами і як вона має взаємодіяти з іншими частинами, модулями або підсистемами.

Функціональні вимоги є фундаментом для розробки будь-якої програмної системи, оскільки саме вони визначають основну логіку обробки даних, архітектуру програмного забезпечення, визначаючи його основні функціональні можливості та поведінку. Вони є ключовими при створенні документації, проектуванні архітектури майбутньої системи, розробці алгоритмів та написанні програмного коду і тестуванні вже готового продукту. Правильно визначенні функціональні вимоги дозволяють створити програму, що відповідає вимогам замовника і очікуванням кінцевих користувачів, а також забезпечують ефективну взаємодію між усіма елементами та користувачами системи.

Функціональні вимоги поділяються на категорії:

- Основні функції системи: ключові можливості, що власне і визначають її призначення (наприклад, можливість реєстрації, оформлення замовлень тощо).
- Обробка даних: описує, як система працює з інформацією, зокрема як відбувається збереження, оновлення чи видалення.
- Інтерфейс користувача: визначає, як саме користувачі будуть взаємодіяти із системою (графічний інтерфейс, наявність мобільної версії).
- Обмеження та правила роботи: встановлює права доступу до певних дій користувачів залежно від ролі або того, чи користувач авторизований.

Функціональні вимоги до програмного додатку для розрахунку моделювання циркуляції повітря в приміщенні охоплюють кілька ключових аспектів, необхідних для створення точних, надійних і зручних для користувача симуляцій. Перш за все, система повинна дозволяти користувачам створювати і змінювати плани приміщень, додаючи, редагуючи або видаляючи архітектурні елементи, такі як стіни, вікна і двері. Ця функціональність гарантує, що користувачі зможуть точно представити фізичне середовище, яке вони хочуть

проаналізувати. Крім того, система повинна дозволяти користувачам вказувати параметри зовнішнього середовища, включаючи температуру, швидкість і напрямок вітру, які впливають на структуру повітряних потоків у приміщенні.

Програмний додаток також повинно підтримувати збереження і відкриття проектів, дозволяючи користувачам зберігати конфігурації приміщень і налаштування моделювання для подальшого використання або подальшого аналізу. Ядром системи є можливість виконувати розрахунки повітряних потоків на основі архітектурного плану та заданих зовнішніх умов. Цей процес розрахунку повинен забезпечувати реалістичні та точні моделі повітряних потоків, які відображають реальну поведінку. Крім того, система повинна візуалізувати результати розрахунку повітряних потоків у чіткому і зрозумілому 2.5D форматі, відображаючи напрямки і швидкості потоків для полегшення аналізу і прийняття рішень.

Кожен компонент - наприклад, редагування плану приміщення, налаштування параметрів навколишнього середовища, розрахунковий механізм і модуль візуалізації - повинен відповідати цим функціональним вимогам, щоб забезпечити ефективну, точну і надійну роботу всієї системи.

В табл. 1.1 наведено специфікацію функціональних вимог до системи наукових та інженерних розрахунків.

Специфікація функціональних вимог

Таблиця 1.1

№	Вимоги	Призначення та використання
1	Створення моделі приміщення	Система повинна дозволяти користувачам задавати архітектурну конфігурацію приміщення (стін, вікон, дверей тощо).
2	Введення параметрів середовища	Система повинна надавати можливість задання зовнішніх та внутрішніх факторів, таких як

№	Вимоги	Призначення та використання
		температура, вологість і швидкість вітру.
3	Візуалізація потоків повітря	Система повинна забезпечувати наочну 2.5D візуалізацію потоків повітря для аналізу.
4	Аналіз результатів	Користувачі повинні мати доступ до інструментів аналізу, таких як графіки потоків.
5	Збереження та завантаження проєктів	Система повинна дозволяти зберігати роботу та завантажувати збережені проєкти для переробки.
6	Розширюваність функцій	Система повинна підтримувати інтеграцію з новими модулями або алгоритмами для вдосконалення моделювання.
7	Генерація звітів	Система повинна автоматично створювати звіти зі сценаріями потоків, висновками та рекомендаціями.

1.2.2 Нефункціональні вимоги

Нефункціональні вимоги визначають фундаментальні характеристики і критерії та функціональність застосунку, такі як ефективність, надійність, безпека, функціональність та інші важливі аспекти. Вони не описують детально конкретні функції системи так як функціональні вимоги, але згадують критерії якості, продуктивності та зовнішніх характеристик, які є дуже важливими для реальних застосувань системи, тобто для її комерційного використання.

Нефункціональні потреби є дуже важливими, для забезпечення ефективності використання програми. Їх відмінність є в тому, що зосереджуються саме на характеристиках та принципах роботи системи, а не на конкретних функціях, які програма повинна виконувати. Це охоплює фактори, що впливають на якість системи загалом та її здатність задовольняти потреби кінцевих користувачів та організацій, які її використовують.

До нефункціональних вимог належать такі, як продуктивність, надійність, масштабованість, безпека, підтримка та сумісність, окрім системної інтеграції. Продуктивність визначає, як швидко система має виконувати операції, особливо коли йдеться про високі навантаження або великі обсяги даних. Надійність означає, що система завжди працює без проблем, що виникає невелика кількість збоїв, які можна за короткий час виправити, що важливо для власників систем та їх бізнесу.

Безпека визначає те, як саме система повинна захищати дані користувачів від незаконного доступу або певних атак. Зручність використання стосується інтерфейсу та зручності взаємодії користувача з програмою, що має безпосередній вплив на ефективність роботи з системою. Масштабованість означає здатність системи обробляти великий обсяг даних, можливість одночасного перебування на сайті великої кількості користувачів без зниження продуктивності.

Вищезгадані вимоги є важливими для створення системи, яка не лише працює, а й ефективно виконує свої функції за різних умов використання, в тому числі в умовах змінюваних навантажень або нестабільних мережевих з'єднань. Тому нефункціональні вимоги є складовою частиною архітектури та дизайну будь-яких програмних додатків.

Нефункціональні вимоги (табл. 1.2) являють собою важливі параметри, які забезпечують якість, продуктивність та безпеку системи. Вони включають такі аспекти, як швидкість обробки даних, сумісність із широким спектром операційних систем і простоту інтерфейсу.

Специфікація нефункціональних вимог

Таблиця 1.2

№	Вимоги	Призначення та використання
1	Швидкість роботи системи	Система повинна обробляти запити користувачів у середньому за час не більше 2 секунд при стандартному навантаженні.
2	Сумістність	Програмний продукт повинен бути сумісним із основними версіями сучасних операційних систем, включаючи Windows, macOS та Linux.
3	Надійнність	Система повинна забезпечувати безперервну роботу з показником доступності на рівні 99,95% упродовж календарного року.
4	Простота у використанні	Інтерфейс користувача має бути інтуїтивно зрозумілим, з чітко структурованими елементами для зручної навігації.
5	Маштабованість	Система повинна підтримувати можливість масштабування для обробки збільшеного обсягу даних без зменшення продуктивності.

Однією з ключових нефункціональних вимог є простота інтерфейсу. Система повинна мати зрозумілий, інтуїтивно зрозумілий інтерфейс, який дозволить користувачам швидко виконувати необхідні дії без потреби в складних налаштуваннях чи додатковому навчанні. Це особливо важливо для уникнення помилок та підвищення ефективності роботи співробітників.

Ще одним важливим аспектом є продуктивність системи. Вона повинна гарантувати швидку обробку запитів користувачів та мінімізувати час відповіді. Це включає оптимізацію обробки запитів до бази даних та ефективну роботу з великими обсягами даних без будь-яких значних затримок. Продуктивність

системи прямо впливає на рівень задоволення користувачів і загальну ефективність процесів.

Сумісність з операційними системами також є важливою вимогою системи. Зокрема, система повинна коректно працювати на операційних системах Windows 10 і новіших версіях. Це гарантує її широке впровадження та використання на існуючому обладнанні, що робить систему зручною та практичною в реальних умовах.

Усі ці нефункціональні вимоги — від простоти інтерфейсу до продуктивності та сумісності — відіграють вирішальну роль у забезпеченні стабільності, надійності та зручності системи. Вони створюють основу для якісної роботи системи в будь-якій сфері, забезпечуючи її відповідність сучасним стандартам безпеки, доступності й ефективності.

1.3 Моделювання предметної області

Моделювання предметної області є наступним не менш важливим етапом у процесі розробки програмного забезпечення після визначення вимог функціональних та нефункціональних, оскільки воно дозволяє задати чіткі межі знань про систему, забезпечуючи уявлення про її структуру та функціональні можливості. У межах даного етапу визначаються основні сутності, їхні характеристики, атрибути, методи та взаємозв’язки між ними, це дозволить створити цілісну і логічну модель системи.

Моделювання відіграє велику роль у забезпеченні аналізу вимог, дозволяючи розробникам, аналітикам та усім хто буде брати участь в створенні системи краще зрозуміти потреби кінцевих користувачів та врахувати всі можливі сценарії використання платформи. Завдяки даному етапу можна оптимізувати процес розробки, і також мінімізувати ризики помилок в майбутньому, оскільки це можна виявити ще на ранніх стадіях проекту.

Моделювання предметної області забезпечує:

- Чітку структуру даних, дозволяючи вибрати та спроектувати архітектуру системи, яка буде ефективно працювати з інформацією.
- Опис бізнес-процесів, допомагаючи уникнути помилок при розробці та впровадженні програми, оскільки всі процеси детально описані та узгоджені між собою.
- Комунікацію між учасниками проєкту, що в свою чергу сприяє спільному розумінню системи серед розробників, тестувальників і замовників, це значно зменшує ризик виникнення помилок.
- Можливість масштабувати та розширювати програмну систему, для цього враховує можливі майбутні зміни в бізнес-процесах і мінімізує ризики, які можуть виникнути при масштабуванні чи розширенні програми.

При моделюванні предметної області часто використовують уніфіковану мову моделювання. Раніше існувало декілька підходів до побудови моделей, це, у свою чергу, спричиняло певні труднощі у взаєморозумінні між учасниками команди розробників та замовниками. Щоб цей процес став стандартизованим, було створено UML (уніфіковану мову моделювання). Така мова моделювання використовується в усьому світі. Вона містить необхідний набір діаграм і графічних елементів та позначень, що дозволяє схематично зобразити структуру та поведінку системи загалом, або її окремих модулів, зробити її зрозумілою для всіх учасників проєкту, що позитивно сприятиме на процес розробки.[3]

Моделювання предметної області - це складний, але дуже важливий процес, що вимагає системного підходу та дотримання низки фундаментальних принципів. Вони допомагають забезпечити точність, логічність та узгодженість моделі, що, у свою чергу, сприяє її ефективному використанню для виконання поставлених завдань.

Одним із ключових принципів є принцип абстрагування, який дозволяє зосередитися лише на основних аспектах об’єкта чи процесу, що мають важливе значення для вирішення конкретної задачі, відкидаючи другорядні, менш важливі

деталі. Такий принцип спрощує аналіз та сприйняття системи. Наприклад, у бізнес-процесі обробки замовлення важливими аспектами є: статус, взаємодія з клієнтом і терміни виконання, а ось розташування складу може бути несуттєвим. Модульність – ще один важливий принцип, що передбачає поділ системи на незалежні компоненти, кожен із яких виконує чітко визначену свою роль. Завдяки такому поділу можна робити оновлення або зміни в окремих модулях без порушення роботи всієї системи.

Також не менш важливу роль відіграє принцип ієрархічності, який організовує структуру системи у багаторівневому вигляді. Це допоможе створити більш впорядковану взаємодію між елементами, а також розподілити відповідальність. Формалізація - ще один важливий принцип, який ґрунтується на використанні строгих правил, певних нотацій та методів для моделювання.

Уніфікація - принцип моделювання предметної області. Уніфікація забезпечує узгодженість між собою моделей завдяки використанню загальновизнаних, стандартних підходів. Це дуже важливо при поєднанні різних систем, так як усім відомі стандарти спрощують їхню взаємодію. Для прикладу, при веб-розробці використання HTML, CSS і JavaScript гарантує коректну роботу сайту у будь-яких браузерах та на різних пристроях.

Застосування вище згаданих принципів моделювання предметної області дозволяє створювати моделі, які будуть зручними для використання, а також для розширення та підтримки в майбутньому.

На основі вищезгаданого можна зробити висновок, що UML є універсальною мовою для моделювання програмних систем при аналізі предметної області. Вона забезпечує найбільш стандартизований підхід до аналізу, дозволяючи учасникам проекту легко взаємодіяти між собою та документувати всі необхідні аспекти розроблюваної системи.

У версії UML 1.5 визначено дванадцять типів діаграм, які розподіленні на три основні групи залежно від їхнього призначення:

- Діаграми, що зображують статичну структуру додатка. Вони описують будову системи, її основні компоненти та їхню взаємодію між собою.

- Діаграми, що зображують поведінку системи. Вони відображають динамічні аспекти додатка, моделюючи процеси та сценарії взаємодії. Такі діаграми допомагають зрозуміти, як саме система повинна поводитися в різних ситуаціях.

- Діаграми, що відображають фізичні аспекти функціонування системи. Вони демонструють організацію системи на архітектурному рівні та розміщення компонентів на фізичному рівні, тобто на рівні обладнання.

У більшості випадків немає потреби створювати всі ці діаграми під час проектування, аналізу чи розробки системи - будуються лише ті, що справді необхідні. У процесі моделювання предметної області для програм найбільш важливими є такі діаграми:

- Діаграма варіантів використання - показує що може робити той чи інший користувач системи.

- Діаграма послідовності - показує як саме взаємодіють об’єкти між собою.

- Діаграма діяльності - описує як саме виконуються бізнес-процеси та алгоритми.

Далі детальніше розглянемо діаграми, які використовувалися під час моделювання предметної області для системи наукових та інженерних розраххунків . Ми проаналізуємо, як вони допоможуть зрозуміти структуру та логіку роботи системи, які бізнес-процеси відображають і як сприяють ефективній розробці. Це продемонструє, як користувачі взаємодіють з програмною системою, як обробляються запити від них під час виконання бізнес-процесів.

1.3.1 Діаграма прецедентів

Для уточнення структури та логіки роботи програми для наукових та інженерних розрахунків була побудована діаграма прецедентів (рис.1.1), яка ілюструє ключових учасників процесу та їх взаємодію із системою. Центральним елементом цієї взаємодії є специфічна програмна система, що підтримує кілька сценаріїв роботи для різних категорій користувачів, таких як інженери або будівництва користувачі

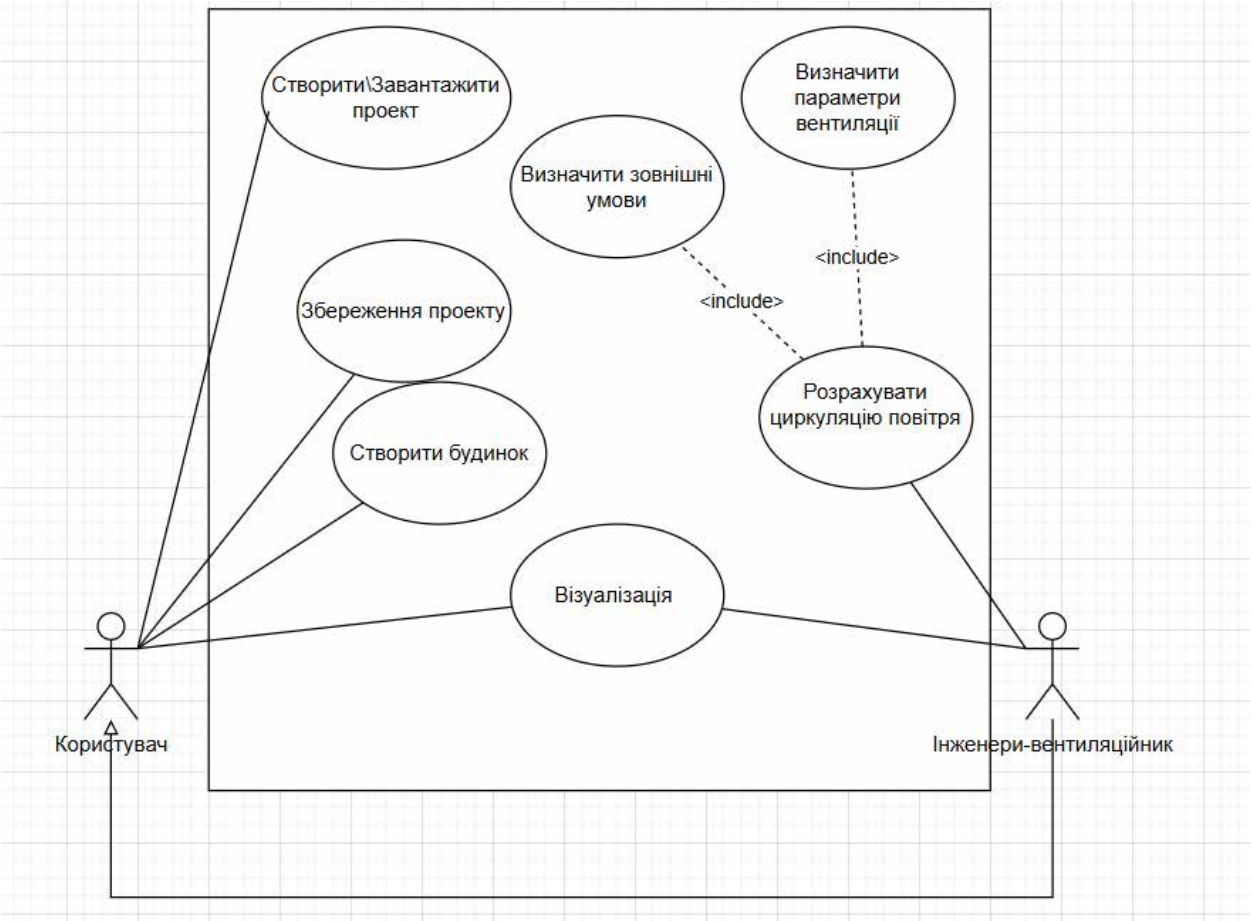

Рис.1.1 Діаграма прецедентів

Користувач виконує основний обсяг щоденних операцій. До його функціоналу належать:

- Створення будинку.
- Збереження проекту
- Створення або завантаження існуючого проекту.
- Візуалізація в 2.5D.

Інженер-вентиляційник має більшу роль , він має такий ж функціонал як і користувач але також має додаткові функції:

- Розрахувати циркуляцію повітря , що включає визначення зовнішніх умов , адже без них коректно визначити циркуляції неможливо , також визначення параметрів вентиляції, без інформації про їх розташування не є можливим правильне розрахування повітряних потоків.

Таким чином, структура ролей та функціоналу користувачів і інженерів-вентиляційників чітко демонструє поділ відповідальностей і визначення ключових завдань для створення ефективного програмного рішення. Користувач виконує основний обсяг щоденних операцій, зосереджених на створенні й візуалізації проектів, забезпечуючи базовий рівень функціоналу. З іншого боку, функціонал інженера-вентиляційника не лише включає всі можливості користувача, але й розширений додатковими спеціалізованими завданнями, такими як розрахунок циркуляції повітря та визначення параметрів вентиляції. Цей детальний розподіл задач дозволяє сформувати інформаційну систему, яка забезпечить автоматизацію процесів від базового моделювання до складних інженерних розрахунків. В результаті, така система сприятиме підвищенню продуктивності, точності та інтеграції робочих процесів.

1.3.2 Діаграма послідовності

Для уточнення структури та логіки роботи програми була побудована діаграма послідовності (рис.1.2), яка візуалізує взаємодію між основними учасниками процесу створення будинку ,візуалізації та аналізу циркуляції повітря.

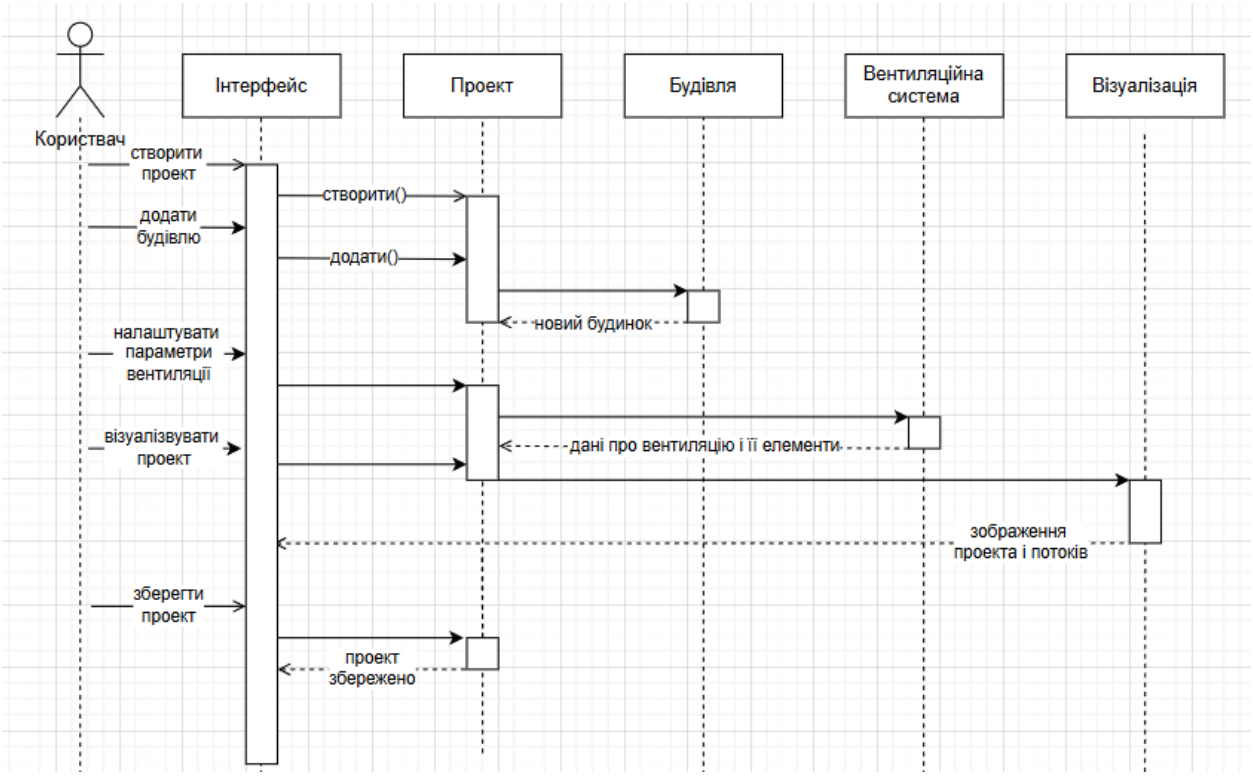

Рис.1.2 Діаграма послідовності

Діаграма послідовності ілюструє структуру інформаційної системи, зосередженої на автоматизації процесів інженера-вентиляційника. Вона демонструє взаємозв'язок між базовими функціями користувача та розширеними завданнями, що виконуються фахівцем. Ключові елементи діаграми включають моделювання вентиляційної системи, розрахунок параметрів циркуляції повітря, а також інтеграцію цих даних у загальний робочий процес. Завдяки ієрархічному підходу, відображеному на діаграмі, система забезпечує послідовність і злагодженість усіх етапів роботи, дозволяючи оптимізувати ефективність і зменшити людський фактор у складних розрахунках.

1.3.3 Діаграма активності

Для візуалізації бізнес-процесу створення будинку ,візуалізації та аналізу циркуляції повітря була побудована діаграма активності (рис.1.3).

Рис.1.3 Діаграма активності

Дана діаграма активності дозволяє побачити основний сценарій роботи з програмою, також дає побачити деякі розгалуження в різних функціях. Така модель є необхідною для аналізу та оптимізації внутрішніх логістичних процесів, зменшення часу простою та забезпечення безперебійної роботи програми.

1.4 Огляд інформаційних джерел та існуючих рішень

У ході дослідження були проаналізовані наступні програмні засоби, які частково або повністю реалізують потрібний функціонал для інженерного моделювання. Кожен із них має свої сильні та слабкі сторони, що дозволяє оцінити їх придатність для різних задач.

Autodesk CFD

Autodesk CFD є професійним рішенням для інженерного моделювання потоків рідин і газів. Ця програма призначена для високоточних розрахунків і має гнучкі можливості інтеграції з іншими інструментами, такими як AutoCAD.

Переваги:

- Висока точність розрахунків;
- Гнучкість у налаштуванні параметрів;
- Зручна інтеграція з іншими продуктами Autodesk.

Недоліки:

- Складність у використанні, потреба в кваліфікованих спеціалістах;
- Надмірна багатофункціональність для простих завдань;
- Висока вартість ліцензії.

COMSOL Multiphysics

COMSOL Multiphysics – це мультифізична платформа для моделювання складних теплових, механічних або потокових задач. Вона дозволяє враховувати багатофакторні взаємодії між об’єктами.

Переваги:

- Надзвичайно точний фізичний аналіз;
- Ефективне багатофакторне моделювання.

Недоліки:

- Потребує глибоких знань фізики та математики для налаштування;
- Освоєння функціоналу займає значний час.

EnergyPlus + OpenStudio

EnergyPlus разом із OpenStudio є пакетами для енергетичного аналізу будівель. Програми відомі своєю відповідністю сучасним стандартам енергозбереження.

Переваги:

- Відповідність енергетичним стандартам;
- Підтримка активного співтовариства розробників і користувачів.

Недоліки:

- Велика кількість вхідних даних ускладнює роботу;
- Слабка візуалізація результатів розрахунків.

CoolVent (MIT)

CoolVent є науковим продуктом, що розроблений для моделювання природної вентиляції в будівлях. Його головним акцентом є практична реалізація концепцій вентиляції.

Переваги:

- Фокус на практичних рішеннях;
- Програма доступна з відкритим вихідним кодом.

Недоліки:

- Обмежена вузькою спеціалізацією;
- Відсутність підтримки і регулярних оновлень.

Далі наведено (табл. 1.3) аналіз існуючих рішень

Таблиця аналізу існуючих рішень

Таблиця 1.3

Програма	**Точність**	**Простота**	**Графіка**	**Ціна**	**Офлайн робота**
Autodesk CFD	Висока	Низька	Висока	$	Так
COMSOL	Висока	Низька	Середня	$	Так
OpenStudio	Середня	Середня	Низька	Безкоштовна	Ні
CoolVent	Середня	Висока	Низька	Безкоштовна	Так
Розробка (ця)	Середня	Висока	Середня	Безкоштовна	Так

Підсумок

Аналіз існуючих рішень показав, що на ринку вже присутні програми з широкими функціональними можливостями, такими як Autodesk CFD і COMSOL Multiphysics, які забезпечують високу точність розрахунків, але мають високу вартість та складність у використанні. Водночас більш доступні рішення, такі як OpenStudio або CoolVent, покривають лише окремі задачі і мають свої обмеження.

З огляду на потребу у створенні легкого, зрозумілого і водночас функціонального інструменту, нова розробка може зайняти нішу, орієнтовану на інженерів та проектувальників із середніми вимогами до ресурсів і простоти у використанні.

Важливо зазначити, що подальша розробка даного інструменту повинна враховувати кращі практики аналізованих програм, забезпечуючи баланс між зручністю використання, ефективністю та доступністю функціоналу.

1.5 Постановка завдання

Програмний додаток, що розробляється, є настільним застосунком (десктопним додатком), призначеним для виконання наукових та інженерних розрахунків, пов’язаних з моделюванням циркуляції повітря у приміщеннях. Основною метою є створення зручного інструменту для інженерів, архітекторів, проектувальників систем вентиляції та дослідників, що дозволяє проводити попередній аналіз повітряного обміну в будівлях на етапах проектування та реконструкції.

Система призначена для використання в сфері будівництва, кліматичного моделювання, енергетичного аналізу, а також для навчальних і дослідницьких цілей. Вона дозволяє користувачу створювати віртуальні моделі будівель, вносити зміни до архітектурної конфігурації (додавання, редагування, видалення стін, вікон, дверей), задавати зовнішні параметри середовища (температура, тиск, швидкість вітру) і виконувати розрахунок циркуляції повітря з подальшою візуалізацією результатів.

Програмний додаток функціонує в середовищі операційної системи Windows. Воно реалізує повний цикл проектування: від створення моделі приміщення до отримання результатів розрахунку з візуальним відображенням повітряних потоків у 2.5D-форматі. Для кожної моделі передбачено можливість збереження проекту у форматі, придатному для подальшого редагування або повторного використання.

З метою захисту даних користувачів реалізовано механізм автоматичного збереження та резервного копіювання проектів. Копії створюються автоматично з інтервалом, заданим у налаштуваннях, і зберігаються у вказаному каталозі, що забезпечує збереження результатів у разі збою або раптового завершення роботи програми.

Інтерфейс користувача побудований відповідно до принципів зручності, простоти й інтуїтивної зрозумілості. Усі основні функції згруповані за вкладками відповідно до етапів роботи: проектування, параметризація, розрахунок, візуалізація. Це дозволяє зменшити час навчання нових користувачів та забезпечує ефективну навігацію по функціоналу.

У системі передбачено дві основні ролі користувачів:

- **Інженер-проєктувальник** —задає вхідні параметри зовнішнього середовища, запускає розрахунок, переглядає результати у візуальному форматі.
- **Користувач** — виконує створення та редагування архітектурної моделі приміщення та візуалізує її.

Розробка програмного додатку передбачає модульну архітектуру, що включає модулі побудови приміщення, розрахунків, візуалізації та управління даними. Це дозволяє забезпечити масштабованість системи, можливість подальшого розширення функціоналу та адаптацію під специфіку різних галузей застосування.

Висновок до розділу

Системний аналіз предметної області дозволив визначити основні задачі, які стоять перед програмним забезпеченням для наукових та інженерних розрахунків, а також сформулювати функціональні та нефункціональні вимоги до його реалізації. Проведений огляд інформаційних джерел і існуючих рішень показав, що на ринку присутні потужні інструменти, проте їм часто бракує зручності використання, доступності чи адаптованості до потреб спеціалістів середньої кваліфікації. Це створює значний простір для розробки нового інструменту, який поєднує простоту, функціональність і ефективність.

Ключовою метою є створення системи, яка задовольнить потреби інженерів та будівельників, забезпечуючи точні розрахунки разом із інтуїтивно зрозумілим інтерфейсом. Використання модульної архітектури дозволить забезпечити гнучкість і масштабованість програмного забезпечення, що є критично важливим для адаптації до майбутніх потреб і вдосконалень. Запропонована система не тільки сприяє автоматизації складних розрахунків, але й створює інтуїтивний та візуально зрозумілий інструмент для аналізу, що підвищує ефективність і оптимізацію процесів у будівництві та проектуванні.

2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Модель «сутність-зв'язок» (ER-модель) – модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. Існує ряд моделей для представлення знань, але одним з найзручніших інструментів уніфікованого представлення даних, незалежного від програмного забезпечення що його реалізує, є модель «сутність-зв'язок». Важливим є той факт, що з моделі «сутність-зв'язок» можуть бути породжені всі існуючі моделі даних (ієрархічна, мережева, реляційна, об'єктна), тому вона є найзагальнішою.

Модель сутність-зв'язок є результатом систематичного процесу, який описує та визначає деяку предметну область. Вона не визначає сам процес, а лише візуалізує його. Дані представлені у вигляді компонентів (сутностей), які пов'язані між собою певними зв'язками, які виражають залежності і вимоги між ними, такі як: одна будівля може бути розділена на нуль або більше квартир, але одна квартира може бути розташована лише в одній будівлі. Сутності можуть мати різні властивості (атрибути), які характеризують їх. Діаграми, створені для представлення цих сутностей, атрибутів і зв'язків графічно, називають сутність-зв'язок діаграмами.

ER-модель зазвичай реалізується в вигляді баз даних. У разі реляційної бази даних, в якій зберігаються дані в таблицях, кожен рядок кожної таблиці являє собою один екземпляр сутності. Деякі поля даних в цих таблицях вказують на індекси в інших таблицях. Такі поля є покажчиками фізичної реалізації зв'язків між сутностями.

Коли ми говоримо про сутність, ми зазвичай говоримо про деякий аспект реального світу, який можна виділити поміж інших аспектів. Сутність – це збірне поняття, деяка абстракція реально існуючого об'єкта, процесу, явища чи деякого уявлення про об'єкт. Хоча термін сутність найбільш вживаний, потрібно розрізняти поняття типу сутності та екземпляру сутності. Поняття тип сутності

відноситься до набору однорідних особистостей, предметів, подій або ідей, виступаючих як ціле. Екземпляр сутності відноситься до конкретної речі в наборі. Наприклад, типом сутності може бути МІСТО, а екземпляром – Київ, Львів і т. д.

На етапі проектування інформаційного забезпечення інформаційної системи було створено логічну модель даних у вигляді ER-діаграми (Entity-Relationship diagram), що представлена на рисунку (див. рис. 2.1). Побудована модель є фундаментом для створення реляційної бази даних та описує структуру інформаційних об'єктів системи, їхні властивості (атрибути), а також логічні зв’язки між ними

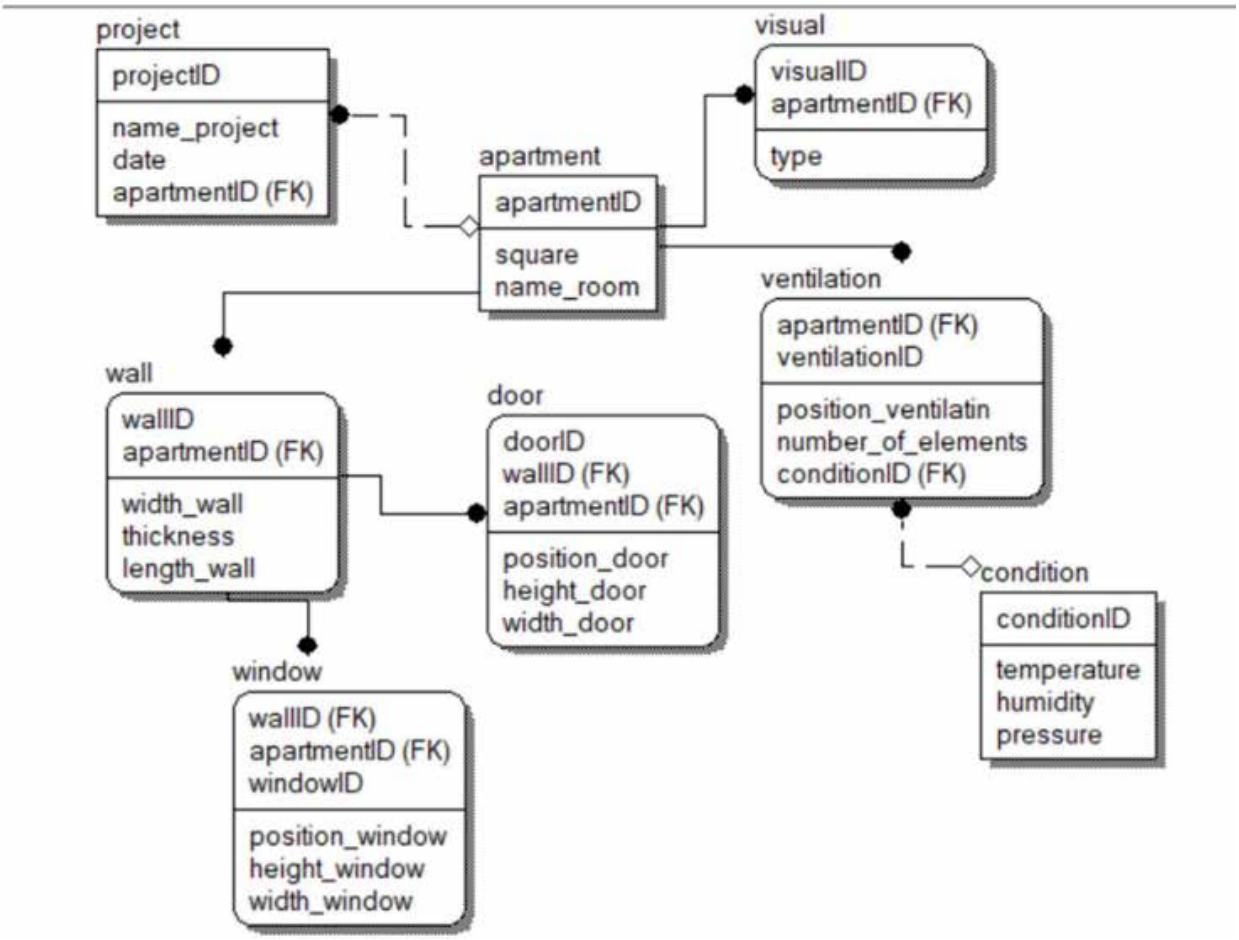

Рис. 2.1 ER-діаграма

Основні сутності моделі:

1. **project** – відображає проєкт, який розробляється користувачем. Містить атрибути:
    - projectID – первинний ключ;
    - name_project – назва проєкту;
    - date – дата створення;
    - apartmentID (FK) – зовнішній ключ, що посилається на сутність apartment.
2. **apartment** – описує окреме приміщення (кімнату або блок). Атрибути:
    - apartmentID – первинний ключ;
    - square – площа приміщення;
    - name_room – назва або тип кімнати.
3. **wall** – містить інформацію про стіни, що належать конкретному приміщенню. Атрибути:
    - wallID – первинний ключ;
    - apartmentID (FK) – зовнішній ключ;
    - width_wall, thickness, length_wall – відповідно ширина, товщина та довжина стіни.
4. **window** – описує вікна, прив'язані до певної стіни та приміщення:
    - windowID – первинний ключ;
    - wallID (FK), apartmentID (FK) – зовнішні ключі;
    - position_window, height_window, width_window – розміщення та розміри вікна.
5. **door** – модель для опису дверей:
    - doorID – первинний ключ;
    - wallID (FK), apartmentID (FK) – зовнішні ключі;
    - position_door, height_door, width_door – координати та розміри дверей.
6. **visual** – відповідає за збереження інформації про візуалізацію об'єкта:
    - visualID – первинний ключ;

- apartmentID (FK) – зовнішній ключ;
- type – тип візуалізації (наприклад, 2D, 3D тощо).

7. **ventilation** – описує елементи вентиляційної системи:
   - ventilationID – первинний ключ;
   - apartmentID (FK) – зовнішній ключ;
   - position_ventilation, number_of_elements, conditionID (FK) – розміщення вентиляції, кількість елементів та зовнішній ключ до умов.
8. **condition** – зберігає параметри зовнішніх умов, що впливають на вентиляцію:
   - conditionID – первинний ключ;
   - temperature, humidity, pressure – відповідно температура, вологість, тиск.

Обґрунтування відповідності 3НФ:

Нормалізація та відповідність 3НФ Щоб схема була в третій нормальній формі (3НФ), вона повинна відповідати наступним вимогам:

1. Перша нормальна форма (1НФ)

- Всі атрибути мають атомарні значення (немає повторюваних груп чи множинних значень).
- Всі таблиці мають унікальні ідентифікатори (Primary Key).

Відповідає – кожна таблиця містить унікальні атрибути (наприклад, apartmentID, wallID, ventilationID тощо).

2. Друга нормальна форма (2НФ)
   - Вона повинна бути в 1НФ.
   - Всі атрибути, які не є первинними ключами, залежать повністю від первинного ключа (тобто не повинно бути часткових залежностей у таблицях зі складеними ключами).

Відповідає – всі неключові атрибути в кожній таблиці залежать лише від свого унікального ідентифікатора (наприклад, width, length, thickness у walls залежать тільки від wallID).

3. Третя нормальна форма (3НФ)
    - Вона повинна бути в 2НФ.
    - Усі атрибути, що не є первинними ключами, не повинні мати транзитивних залежностей (атрибути залежать тільки від первинного ключа, а не від інших неключових атрибутів).

    Відповідає – немає транзитивних залежностей. Наприклад, у таблиці ventilation поле conditionID є зовнішнім ключем (FK), але воно логічно зв’язане через окрему сутність condition, що усуває транзитивну залежність.

2.2 Діаграма класів та кооперації

2.2.1 Діаграма класів

Діагра́ма кла́сів — статичне представлення структури моделі в UML. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів 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 служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини (зв'язки).

На діаграмі класи представлені у вигляді прямокутників, які містять три відділення:

Відображення класу з трьома відділеннями.

- Верхня частина містить **назву класу**. Вона надрукована напівжирним шрифтом і вирівняна по центру, починається з великої літери.
- Середнє відділення містить **атрибути класу**. Вони вирівняні по лівому краю і перша літера мала.
- Нижній відсік містить **операції, які може виконувати клас**. Вони також вирівняні по лівому краю і перша літера мала.

Можливо додання четветого відділення, яке містить коментарі.

Не менш важливими є зв’язки між класами. Всього існує п’ять типів зв’язків: асоціація, узагальнення, залежність, агрегація і композиція.

Асоціація показує, що об'єкти однієї сутності (класу) пов'язані з об'єктами іншої сутності. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Узагальнення (наслідування) вказує на ієрархічну структуру класів, де один клас (дочірній) успадковує атрибути та методи іншого класу (батьківський). Узагальнення означає, що об'єкти класу-нащадка можуть використовуватися скрізь, де зустрічаються об'єкти класу-батька, але не навпаки. Іншими словами, нащадок може бути підставлений замість батька. При цьому він успадковує властивості батьків, зокрема його атрибути і операції. Часто, хоча і не завжди, у нащадків є і свої власні атрибути і операції, крім тих, що існують у батька.

Залежність показує, як один клас використовує інший. Зміна в описі одного класу може вплинути на інший клас. Наприклад, клас "Замовлення" може залежати від класу "Клієнт", якщо він має посилання на клієнта, який оформив замовлення.

Агрегація є вкладенням одного класу в інший, але клас обгортки не контролює термін служби вкладеного об'єкта (який представлений посиланням на об'єкт, що використовується).

Композиція подібний до агрегації, але має більшу залежність. Об'єкти класу, що використовується в композиції, створюються разом з об'єктом іншого класу і повністю контролюються ним. Наприклад, клас "Автомобіль" може

містити клас "Двигун", при цьому двигун існує тільки як частина автомобіля і не може існувати окремо.

На діаграмі класів (рис. 2.2) представлено основні сутності програмного додатку для наукових та інженерних розрахунків.

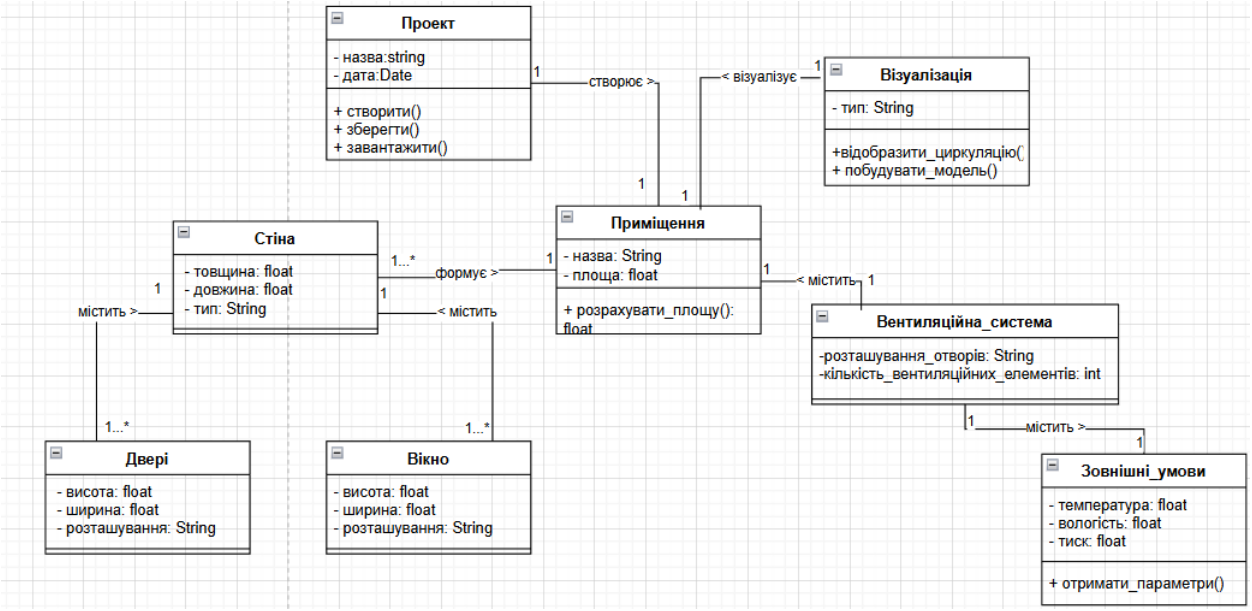

Рис. 2.2 Діаграма класів

Діаграма включає класи, які представляють ключові елементи системи, такі як: проект, приміщення, візуалізація, стіна, двері, вікно, вентиляційні системи, зовнішні умови. В проекті створюються приміщення. Візуалізація надає нам 2.5D зображення приміщення. Стіни формують приміщення. Двері та вікна містяться на стіни. Приміщення містить вентиляційні системи. Вентиляційні системи містять зовнішні умови завдяки яким можливий прорахунок циркуляції повітря.

Опис структури класів наведений в табл. 2.1.

Таблиця 2.1

Опис структури класів додатку

№	**Назва класу**	**Атрибути**	**Методи**
1	Проект	Назва, Дата	Створити, Зберегти, Завантажити
2	Приміщення	Назва,Площа	Розрахувати площу
3	Візуалізація	Тип	Відобразити циркуляцію, Побудувати модель
4	Стіна	Товщина, Довжина ,Тип	
5	Двері	Висота, Ширина, Розташування	
6	Вікна	Висота, Ширина, Розташування	
7	Вентиляційна система	Розташування отворів, Кількість вентиляційних отворів	
8	Зовнішні умови	Вологість, Тиск , Вологість	Отримати параметри

2.2.2 Прості кооперації

Прості кооперації – взаємодія між кількома об'єктами з метою виконання певної функції або досягнення спільної цілі. Кооперації зручно відображати як

діаграми класів за допомогою UML. Проста кооперація являється частиною діаграми класів, яка уточняється для відображення взаємодії між групою класів. Далі будуть наведені приклади простих кооперацій, побудованих на основі діаграми класів. (див. рис. 2.2)

Кооперація будування приміщення (рис 2.3) демонструє створення моделі приміщення . Спочатку малюються стіни потрібних налаштувань товщини, які і формують приміщення , далі потрібні ділянки стіни можна замінити на двері , з налаштуванням їх розміру , або вікна , з налаштуванням їх розміру та відстанню до підлоги.

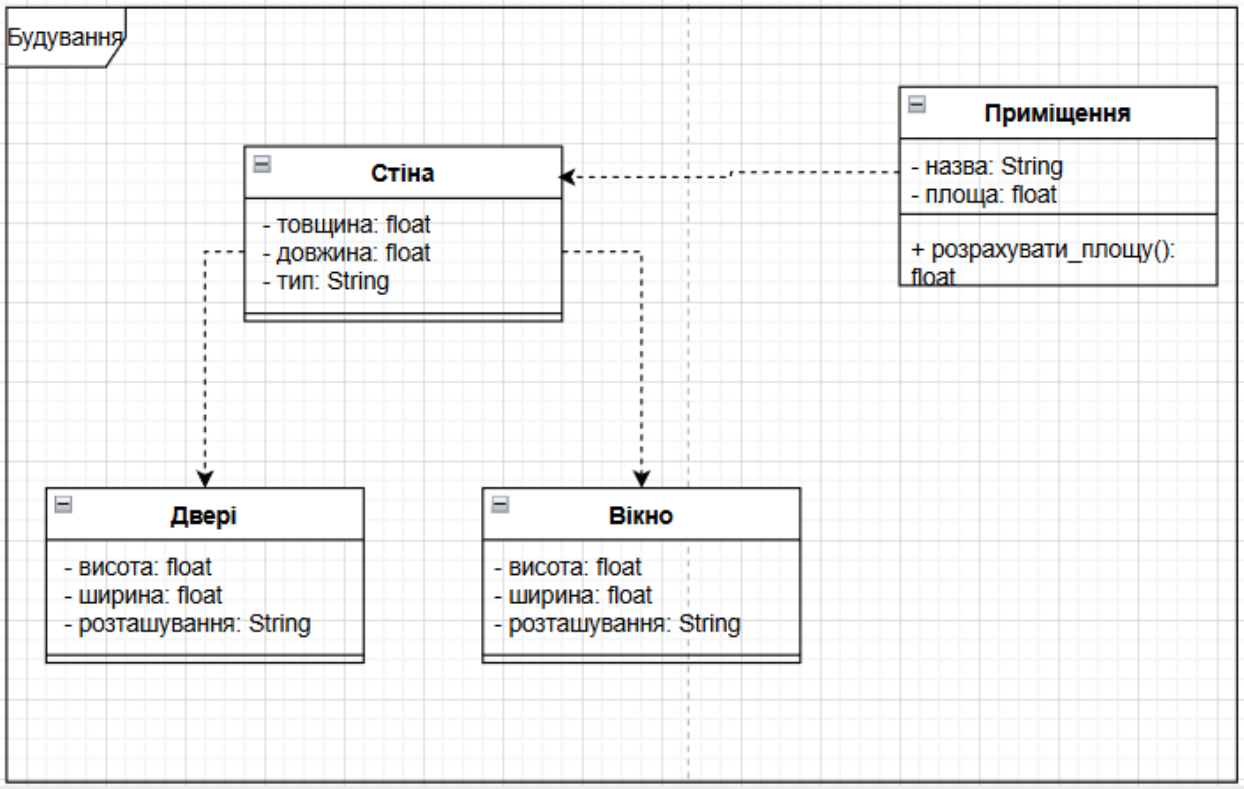

Рис. 2.3 Будування приміщення

Кооперація вентиляція (рис 2.4) демонструє роботу вентиляційної системи. Кожне приміщення має вентиляційні канали , отвори по яким повітря буд циркулювати , а для правильного прорахунку як буде себе поводити повітря в приміщені потрібно задати зовнішні умови, такі як тиск , вологість , температура.

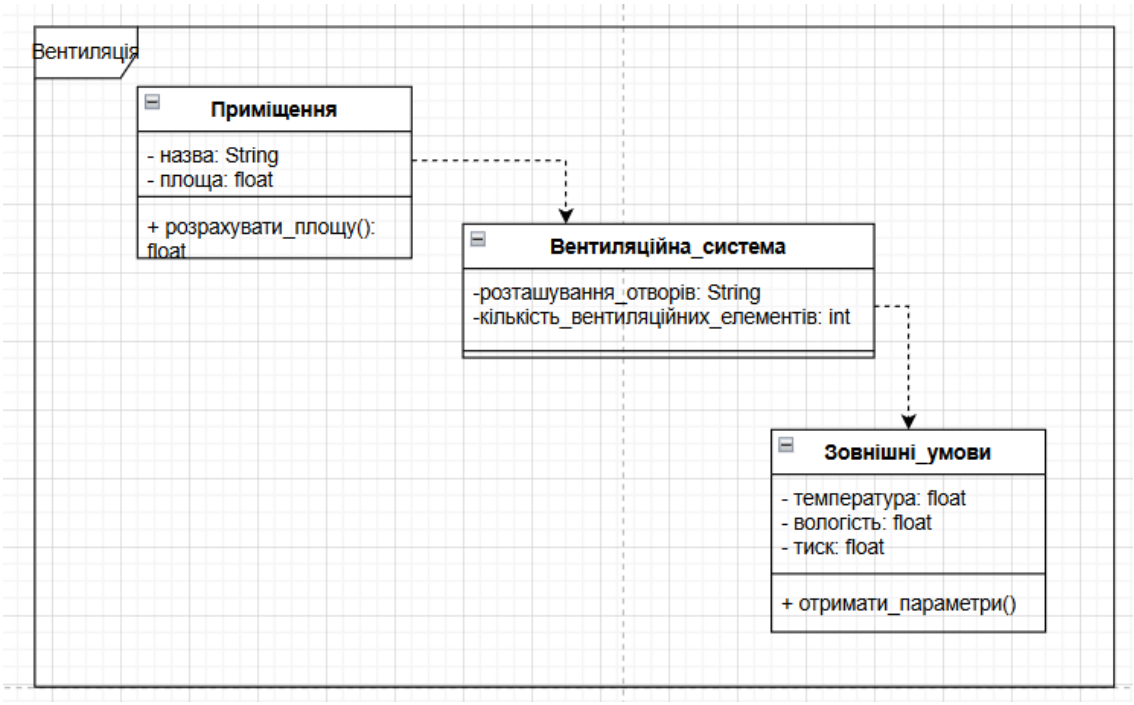

Рис. 2.4 Вентиляція

Кооперація візуалізація проекту (рис 2.5) демонструє процес представлення проекту в трьох вимірній графіці. Також процес представлення повітряних потоків , як вони циркулюють по приміщенню.

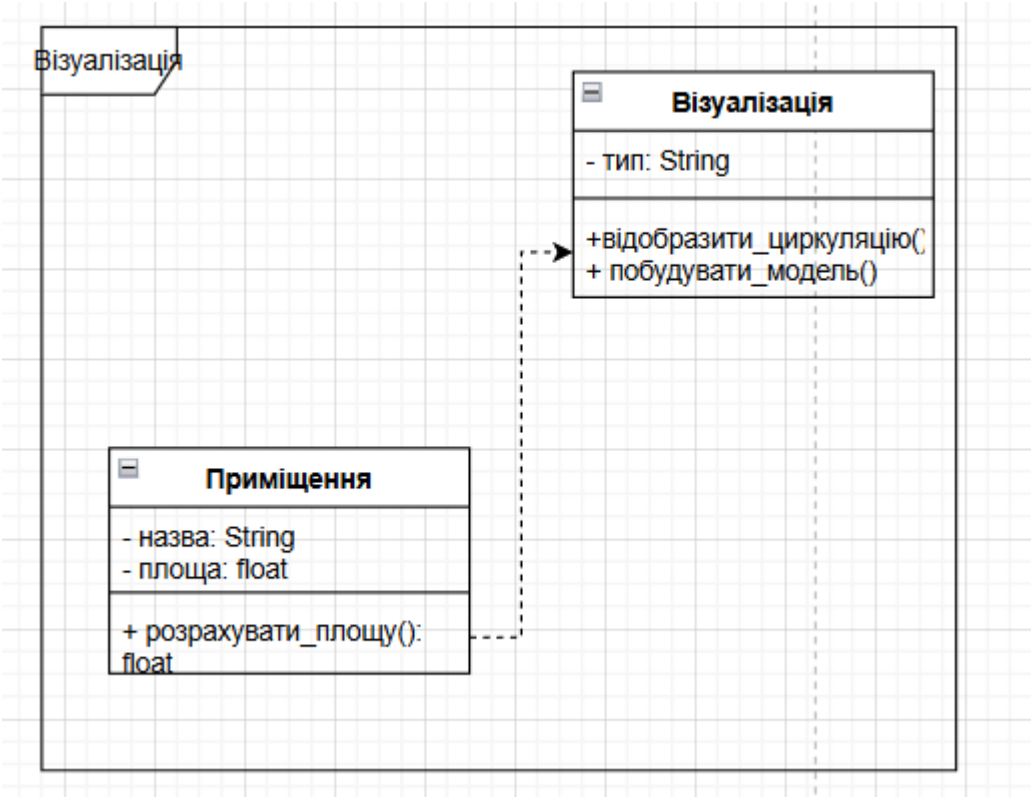

Рис. 2.5 Візуалізація проекту

2.3 Діаграма пакетів

Діаграма пакетів – це різновид структурної діаграми UML, який дозволяє відображати архітектуру програмної системи на високому рівні абстракції. Основним елементом такої діаграми є пакет, що являє собою логічне об'єднання класів, підсистем або інших структурних одиниць. Такі діаграми використовується для візуалізації модульної організації системи, а також для показу залежностей між її елементами.

Діаграми пакетів корисні на етапі проектування програмного забезпечення, коли важливо зрозуміти загальну структуру системи, розділити її на підсистеми та встановити взаємодію між ними. Це допомагає створити гнучку архітектуру та спрощує супровід системи в майбутньому.

Діаграма пакетів (рис. 2.6) відображає архітектуру програмного забезпечення, побудовану за принципами багаторівневої (шарової) моделі, що забезпечує модульність, зручність підтримки та можливість масштабування системи.

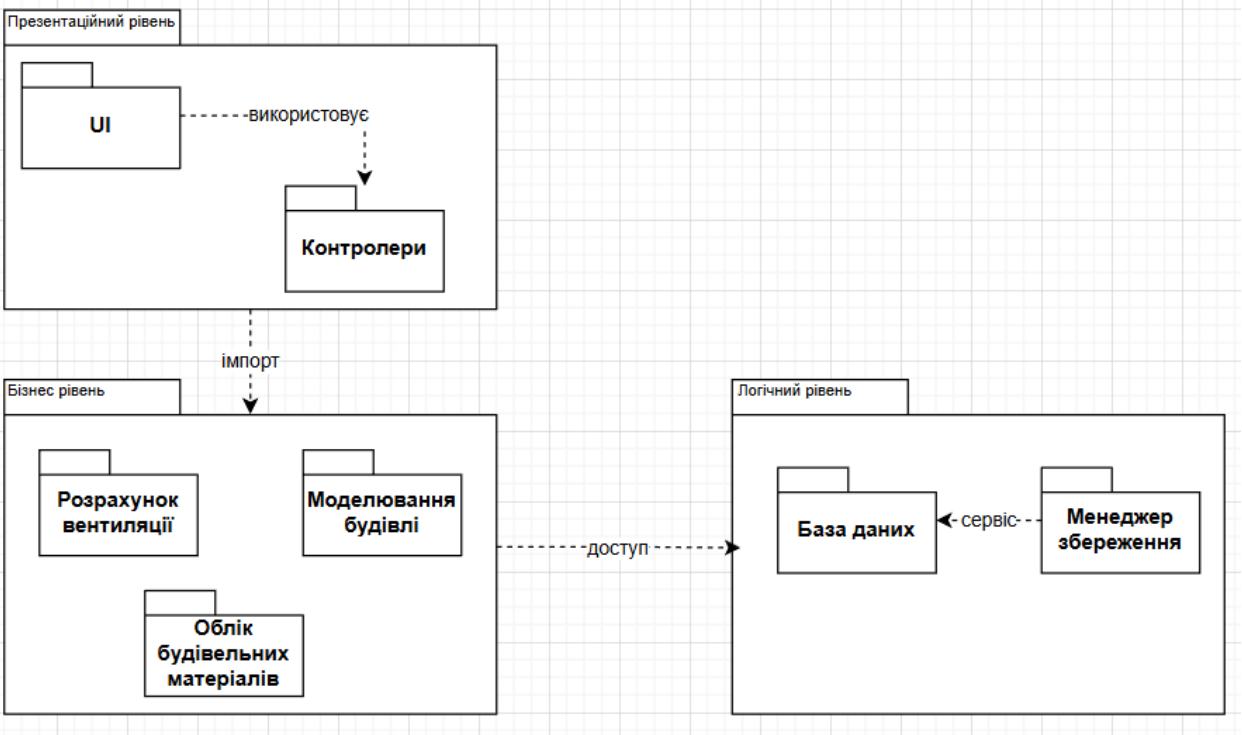

Рис. 2.6 Діаграма пакетів

На верхньому рівні розташовано **презентаційний шар**, який відповідає за взаємодію з користувачем. До його складу входять компоненти інтерфейсу, за допомогою яких користувач створює або редагує проєкт, задає параметри приміщень, стін, вентиляційних отворів і зовнішніх умов. Презентаційний шар реалізовано за допомогою бібліотеки PyQt, що забезпечує графічний інтерфейс програми.

Презентаційний шар взаємодіє з **шаром логіки додатку**, який містить основні модулі, що відповідають за обробку введених даних, керування проектами, побудову структур приміщень, моделювання повітряних потоків і проведення базових розрахунків. Тут зосереджено бізнес-логіку додатку: перевірка даних, обробка змін, формування візуалізаційних моделей та взаємодія з об'єктами приміщень.

Шар логіки звертається до **шару роботи з даними**, який реалізує збереження, завантаження та оновлення інформації про проєкти. Він забезпечує абстрагування від конкретної реалізації зберігання — файлів, бази даних чи іншого формату. Таким чином, додаток може гнучко адаптуватися до змін способу зберігання без втручання в логіку чи інтерфейс користувача.

На нижньому рівні знаходиться **шар зберігання даних**, де безпосередньо зберігається інформація про будівлі, стіни, вентиляційні елементи, зовнішні умови та результати моделювання. У поточній реалізації зберігання здійснюється у вигляді локальних файлів, проте архітектура дозволяє легко розширити її до використання повноцінної бази даних.

Запропонована архітектура забезпечує **чітке розділення відповідальностей** між компонентами системи, полегшує її модифікацію, тестування та супровід. Кожен пакет ізольовано виконує свою функцію і взаємодіє лише з суміжними шарами, що відповідає принципам **low coupling / high cohesion**.

High Cohesion (Висока згуртованість)

Це означає, що усі елементи (методи, змінні) всередині одного модуля або класу мають тісно пов'язану функціональність, тобто працюють над спільним завданням або логічно належать до однієї сутності.

Low Coupling (Низьке зв’язування)

Це означає, що клас або модуль має мінімальні залежності від інших класів або модулів. Він може працювати самостійно або з невеликим впливом зовнішніх змін.

2.3 Діаграма компонентів

Діаграма компонентів UML надає концептуальну картину взаємодії між різними системами. Можуть бути присутні як аспекти логічного, так і фізичного моделювання. Крім того, компоненти автономні. Це модульний системний елемент в UML, який можна замінити на альтернативні. Вони містять конструкції будь-якої складності та є самодостатніми. Лише через інтерфейси вкладені частини взаємодіють з іншими компонентами. Крім того, компоненти мають свої інтерфейси, але вони також можуть отримати доступ до операцій і служб інших компонентів за допомогою їхніх інтерфейсів. На діаграмі компонентів інтерфейси також показують зв’язки та залежності в архітектурі програмного забезпечення.

Діаграма компонентів (рис. 2.7) демонструє архітектуру програмного додатку для моделювання циркуляції повітря у приміщеннях, побудовану за принципом трирівневої структури, що відповідає концепції розділення обов’язків (Separation of Concerns). Архітектура включає три основні рівні: інтерфейс користувача (Presentation Layer), бізнес-логіку (Domain Logic) та рівень даних (Data Access Layer).

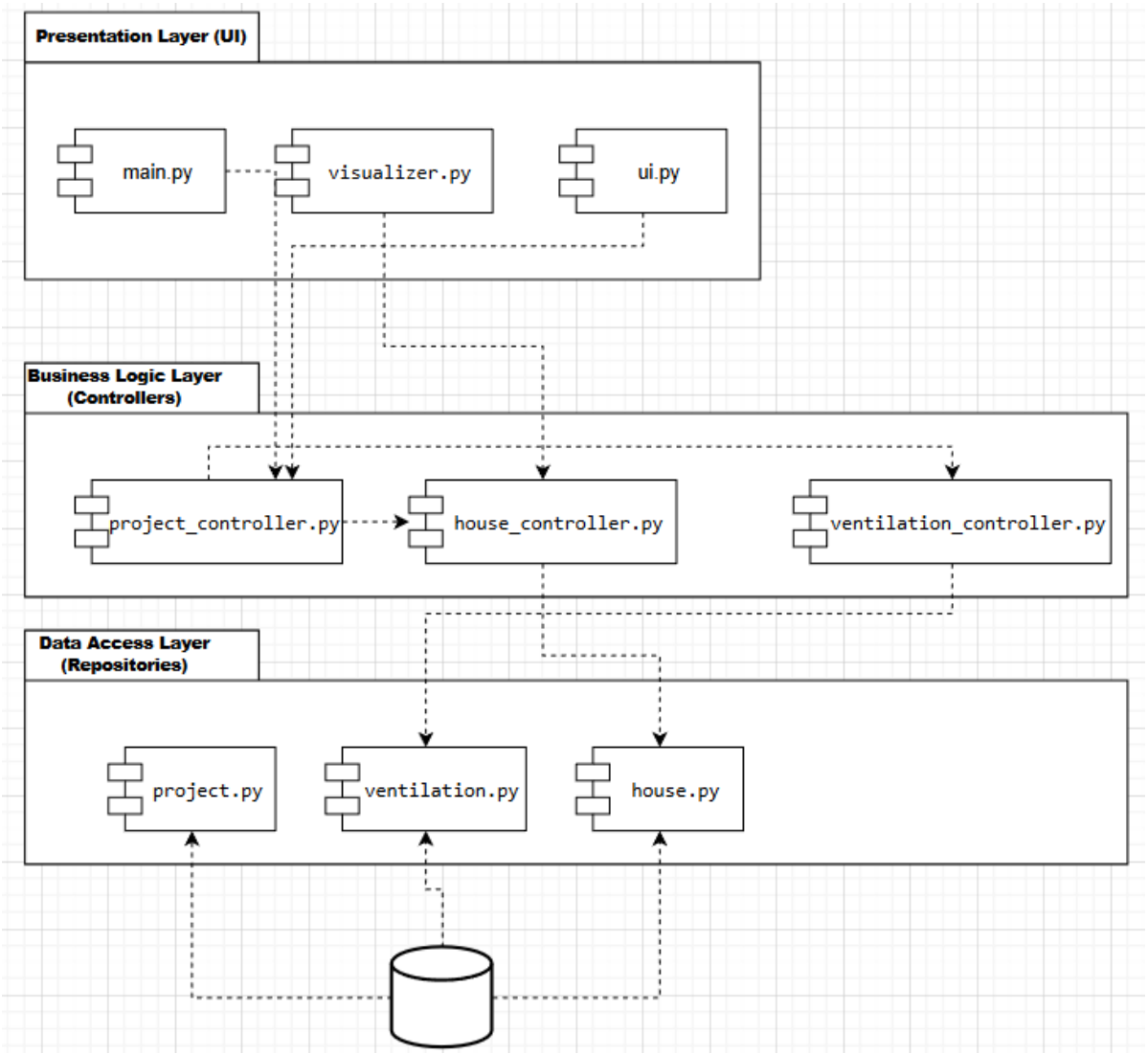

Рис. 2.7 Діаграма компонентів

На рівні інтерфейсу користувача розміщено компоненти, що відповідають за взаємодію з користувачем — створення, відкриття та збереження проєкту, візуалізацію результатів обчислень. Вони реалізовані у вигляді діалогових вікон і головного робочого середовища.

Середній рівень — бізнес-логіка — відповідає за обробку інформації про приміщення, стіни, вікна, двері, вентиляційну систему та зовнішні умови. Цей рівень реалізує основні методи обрахунку, зокрема: площі приміщення, моделювання повітряних потоків, а також структурує зв’язки між об’єктами.

Рівень доступу до даних зосереджений на збереженні та завантаженні інформації про проект, що дає змогу серіалізувати модель і зберігати її у відповідному форматі.

Дана структура дозволяє досягти низького зв’язку (low coupling) між компонентами різних шарів та забезпечує високу згуртованість (high cohesion) всередині кожного модуля. Це значно спрощує масштабування та супровід системи.

Висновок до розділу

У другому розділі детально розглянуто архітектурний підхід до проектування багаторівневих систем, який передбачає чітке розмежування функцій між шарами. Було встановлено, що така структура дозволяє не лише забезпечити високу гнучкість та масштабованість системи, але й підвищити її надійність завдяки зниженню рівня зв’язку між компонентами різних шарів. Аналіз реалізації цього підходу продемонстрував ефективність дотримання принципів низької зв’язаності та високої згуртованості, що є основною передумовою для довгострокового супроводу і модернізації системи. Таким чином, створені підходи та моделі забезпечують стійкий фундамент для розробки програмного додатку, здатного відповідати сучасним вимогам і стандартам.

3. РОЗРБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Система управління базами даних (СУБД) – це спеціалізоване програмне забезпечення, яке забезпечує управління створенням, підтримкою та використанням баз даних. СУБД є посередником між прикладними програмами та даними для структурування збереження інформації та ефективного доступу до них. Основні функції СУБД включають:

- визначення даних через створення та модифікацію структури бази даних;
- маніпулювання даними шляхом додавання, видалення, оновлення та отримання даних;
- забезпечення цілісності даних через контроль за правильністю та несуперечливістю інформації;
- керування транзакціями згідно з принципами ACID;
- управління доступом для забезпечення безпеки та авторизації користувачів;
- оптимізацію продуктивності для ефективного виконання запитів.

СУБД можна класифікувати за різними ознаками. За моделлю даних вони поділяються на реляційні, об'єктно-орієнтовані, ієрархічні та NoSQL. За архітектурою СУБД бувають файл-серверними, клієнт-серверними, вбудованими та розподіленими.

У рамках розробки програмного додатку для наукових та інженерних розрахунків було прийнято рішення використовувати реляційну систему управління базами даних **SQLite**. Цей вибір обумовлений особливостями проекту, який передбачає автономну роботу настільного застосунку, що не потребує централізованого серверного середовища.

Вибір реляційної моделі даних ґрунтується на чіткій структурі інформації, яку обробляє система: проєкти будівель, конфігурації приміщень, стіни, вікна,

двері, вентиляційні елементи та параметри зовнішнього середовища. Усі ці об'єкти мають однозначно визначені атрибути та логічні зв’язки між собою, що зручно моделювати у вигляді таблиць з первинними та зовнішніми ключами.

Реляційна модель також дозволяє забезпечити:

- **цілісність даних** (через обмеження зв’язків і транзакції),
- **гнучкість при виконанні запитів** (наприклад, для аналізу змін конфігурації, порівняння умов, формування звітів про розподіл повітря),
- **масштабованість структури**, що дає змогу розширювати функціональність проекту без порушення логіки зберігання даних.

СУБД **SQLite** є оптимальним варіантом для реалізації даного проекту завдяки своїм технічним характеристикам: вона легка, не вимагає окремої установки або конфігурації серверного ПЗ, інтегрується безпосередньо в застосунок, має низькі вимоги до ресурсів та забезпечує високу швидкість обробки локальних запитів.

Додатковими перевагами SQLite є:

- підтримка **мови SQL** для маніпуляцій з даними;
- **повна сумісність з Python** — основною мовою розробки застосунку;
- інтеграція через стандартну бібліотеку sqlite3, що дозволяє швидко реалізувати механізми збереження, відкриття та редагування проектів користувача;
- зручна підтримка **локального формату файлу проекту**, що підходить для інженерного програмного додатку, орієнтованого на одного користувача або невелику групу.

Таким чином, обраний підхід до управління інформаційною базою відповідає ключовим вимогам до проекту — простота, автономність, стабільність, швидкість та зручність у реалізації на етапі розробки і тестування програмного забезпечення.

3.2 Розробка інформаційної бази

При розробці інформаційної бази програмного додатку для наукових та інженерних розрахунків використовувалась система управління базами даних SQLite через її легкість, вбудованість і достатній функціонал для потреб настільного застосунку. Процес створення бази даних розпочався зі створення логічної структури та формалізації основних об'єктів предметної області.

Було сформовано 8 взаємопов’язаних таблиць, які охоплюють ключові аспекти будівельної моделі, включаючи опис приміщення, його стін, отворів (вікон і дверей), умов середовища, елементів вентиляції, візуальних параметрів та проектів. Приклад SQL-коду створення таблиці wall наведено на рис. 3.1. Повний перелік SQL-інструкцій зі створення таблиць подано в **Додатку А**.

```
CREATE TABLE wall (
    wallID INTEGER PRIMARY KEY,
    apartmentID INTEGER,
    width_wall REAL,
    thickness REAL,
    length_wall REAL,
    FOREIGN KEY (apartmentID) REFERENCES apartment(apartmentID)
);
```

Рис. 3.1 Код створення таблиці стіни

Основу фізичної структури БД становлять такі таблиці:

- apartment — базова таблиця, що описує кімнати (їх площу та назву);
- wall — містить інформацію про стіни, їхні розміри та прив’язку до кімнати;
- door і window — описують відповідні отвори, розташовані в межах стін;
- project — описує окремі проєкти моделювання;
- condition — містить інформацію про параметри середовища (температуру, вологість, тиск);
- ventilation — зберігає конфігурацію вентиляційних елементів;
- visual — зберігає інформацію для візуалізації моделі.

Фізичну модель даних зображено на рис. 3.2.

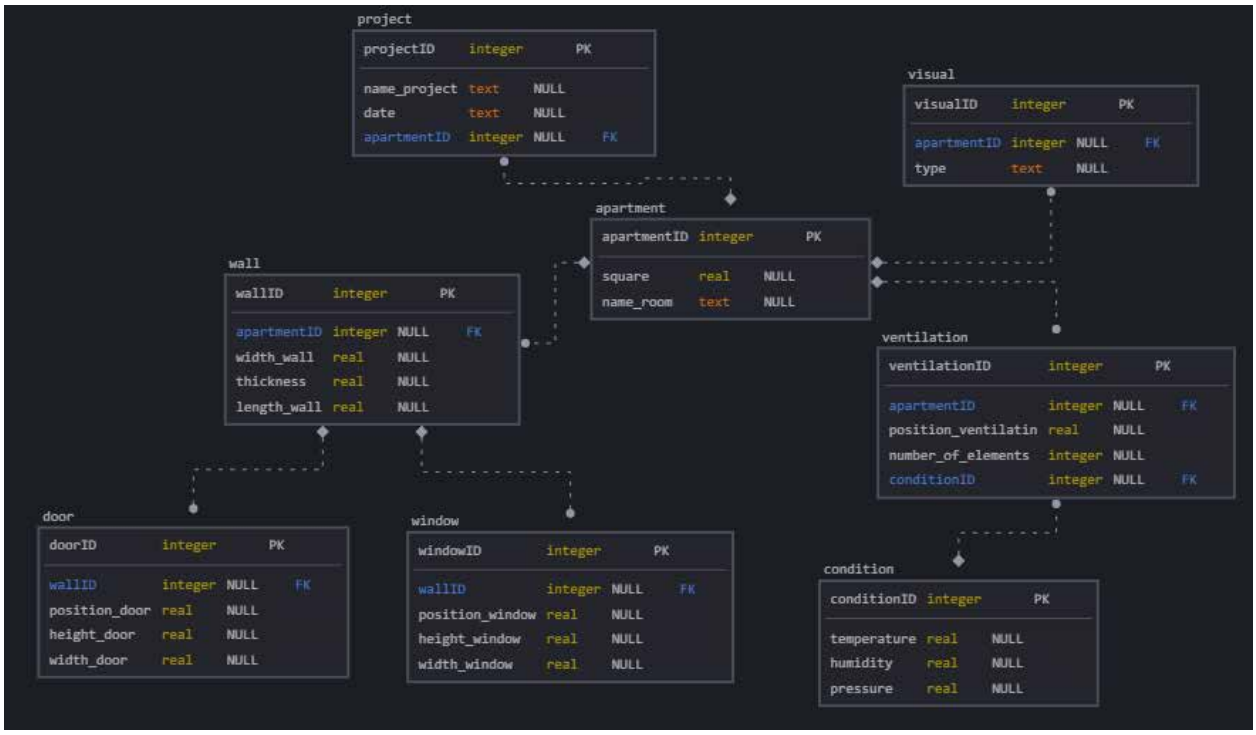

Рис. 3.2 Фізична модель даних

Для реалізації бази даних використовувалися базові типи даних SQLite: INTEGER для ідентифікаторів, REAL для числових параметрів (довжина, ширина, координати), TEXT для зберігання назв приміщень, проектів та типів візуалізації. Особливу увагу приділено забезпеченню коректних зв’язків між таблицями за допомогою зовнішніх ключів (FOREIGN KEY), що забезпечує цілісність даних на фізичному рівні.

Надалі на основі цієї структури реалізовано взаємодію з інтерфейсом користувача, зокрема: створення проекту, редагування конфігурації приміщення, розміщення вентиляційних елементів, та ініціація розрахунків потоків.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

3.3.1 Вимоги проекту та технічні обмеження

Розробка настільного застосунку для операційної системи Windows з локальним розміщенням бази даних безпосередньо на комп’ютері користувача. Основними завданнями системи є створення цифрової моделі приміщення,

розрахунок моделювання повітряних потоків із урахуванням розташування стін, вікон, дверей і вентиляційних елементів, візуалізація напрямків циркуляції повітря у спрощеному 2.5D-режимі, а також збереження проектів для подальшого аналізу.

Серед технічних обмежень варто зазначити:

- Відсутність постійного підключення до інтернету на об’єкті, тому необхідна офлайн-робота з базою даних;
- Обмежені апаратні ресурси (старіші офісні комп’ютери в інженерних організаціях);
- Необхідність простого інтерфейсу для користувачів без глибоких технічних знань.

3.3.2 Мова програмування

Для розробки застосунку було обрано **Python** — мову програмування високого рівня, що активно використовується у сфері наукових та інженерних обчислень. Python має велику екосистему бібліотек для роботи з геометрією, математичним моделюванням та побудовою графічних інтерфейсів, що ідеально підходить для завдань, пов’язаних з проектуванням і візуалізацією.

Перевагами Python у контексті даного проекту є:

- **Широка підтримка інженерних бібліотек** (наприклад, NumPy, SciPy, matplotlib), які можна використовувати для реалізації алгоритмів моделювання повітряних потоків;
- **Використання PyQt** для створення сучасного GUI: забезпечує кросплатформенність і можливість побудови складних діалогів з підтримкою інтерактивної візуалізації;
- **Простота інтеграції з SQLite** через вбудований модуль sqlite3, що дозволяє зберігати локальну інформацію про проєкти без додаткового встановлення СУБД;
- **Гарна читабельність і низький поріг входу**, що спрощує супровід і підтримку системи.

Python також підтримує роботу з файлами, графікою та математичними розрахунками без додаткових витрат на ліцензування або використання сторонніх компонентів, що важливо для розгортання у бюджетних освітніх або проектно-інженерних установах.

3.3.3 Графічний інтерфейс користувача

Для реалізації графічного інтерфейсу обрано бібліотеку **PyQt5** — популярний фреймворк для створення настільних додатків із підтримкою сучасного дизайну. PyQt дозволяє гнучко налаштовувати вигляд елементів управління, реалізовувати багатовіконний режим (головне вікно, вікна налаштувань приміщень, об’єктів, умов середовища), а також вбудовувати зони для інтерактивного креслення та перегляду моделі.

Крім того, бібліотека має підтримку OpenGL та анімації, що в перспективі може бути використано для побудови розширеної візуалізації потоків або інтеграції 3D-елементів.

3.3.4 База даних

Для зберігання інформації про приміщення, їхні властивості, проекти та параметри середовища використовується **SQLite** — легка вбудована СУБД, що не вимагає окремого серверного ПЗ. Це забезпечує простоту розгортання системи на будь-якому комп’ютері під керуванням Windows без додаткових налаштувань.

База даних зберігається у вигляді окремого .db-файлу в каталозі користувача. Завдяки використанню зовнішніх ключів та відповідно організованій структурі таблиць забезпечується цілісність даних і можливість зберігання декількох проектів в одному сховищі.

3.4 Алгоритмізація та програмування програмних модулів

В ході розробки програмної системи було реалізовано багато програмних рішень для вирішення поставлених задач. Далі розглянемо код деяких з них. Код розглянутих програмних рішень надано в ДОДАТКУ Б.

3.4.1 Створення стіни

Розглянемо основну , несучу , програмне рішення створення стін приміщення рис. 3.3.

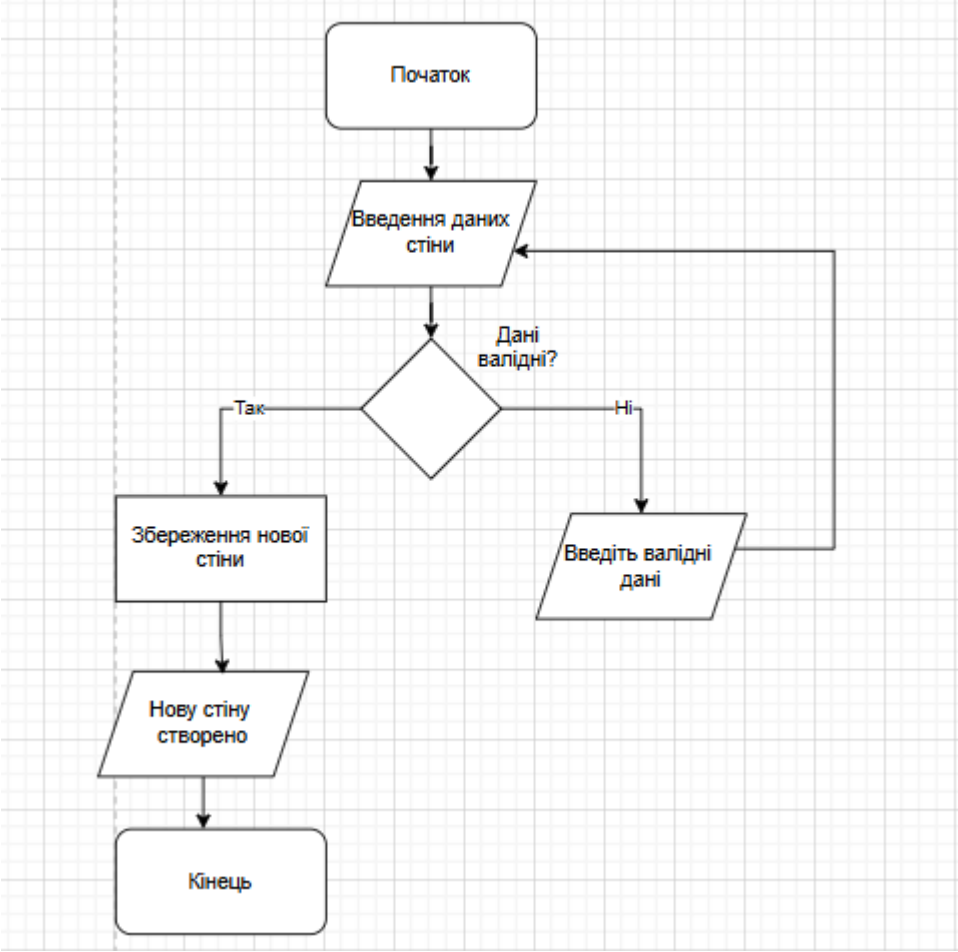

Рис. 3.3 Блок-схема алгоритму створення стіни

У застосунку є спеціальна форма (рис.3.4) для створення стіни . Вона містить кілька полів для введення користувачем : висота , товщина.

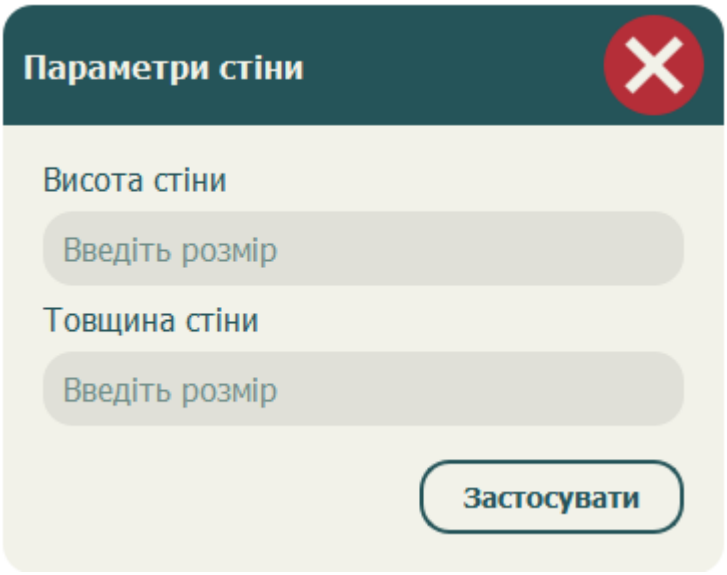

Рис. 3.4 Форма створення стіни

Після заповнення форми користувач натискає кнопку «застосувати» , далі програма перевіряє чи не пусті поля , що обов’язкові для заповнення , якщо вони пусті програма повідомляє про це . Також програма не дозволяє вводити дані в некоректному форматі.

Якщо все гаразд, формується об’єкт , де:

1. генерується унікальний код;
2. усі введені значення зберігаються в модель.

Після успішного збереження форма очищується і користувачу показується повідомлення про успішне створення нової стіни.

3.4.2 Додавання вікна

Для того щоб була можливість створення кімнати , а не тільки складу чи гаражу ,створено програмне рішення додавання вікон рис. 3.5.

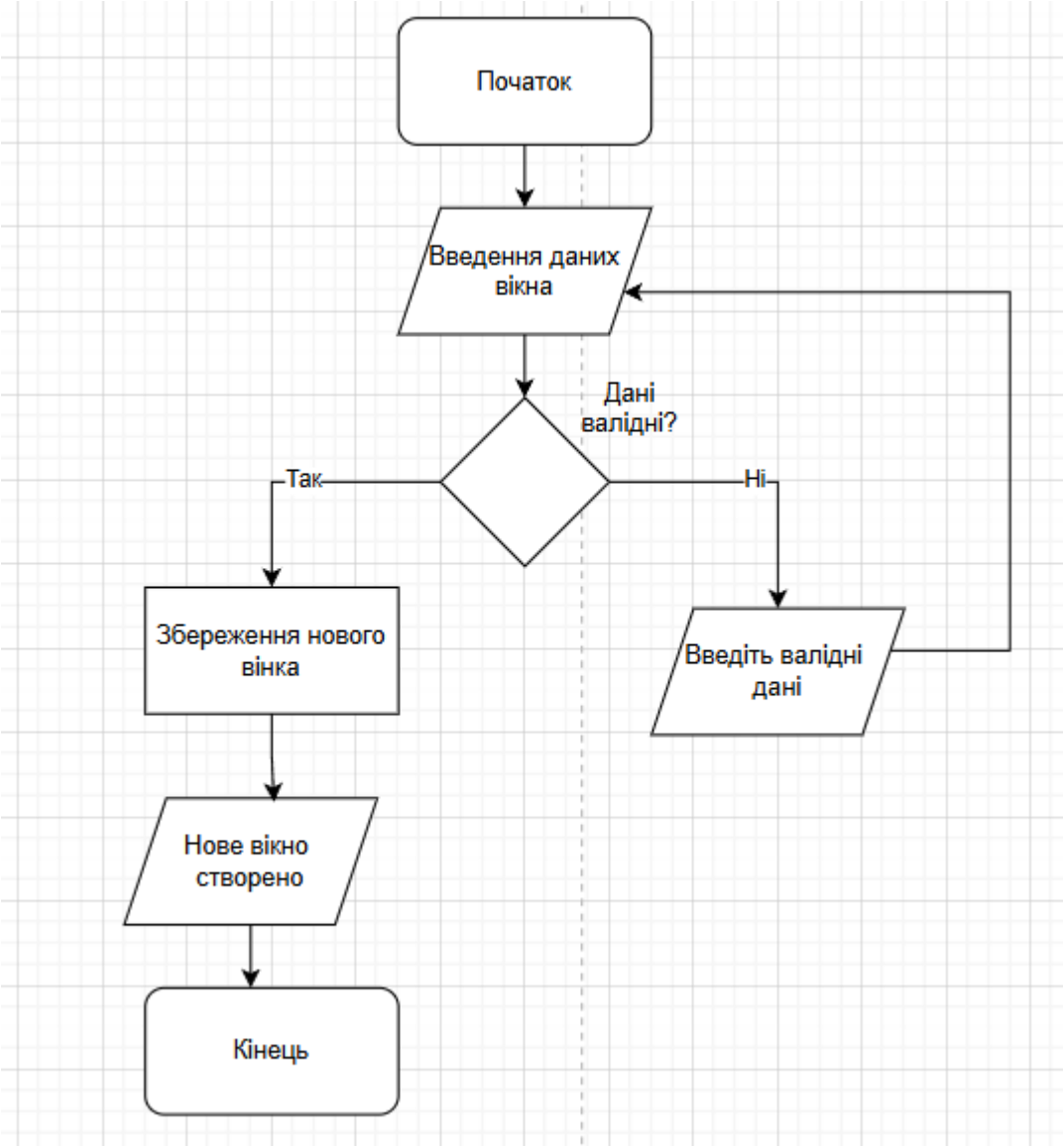

Рис. 3.5 Блок-схема алгоритму створення вікон

У застосунку є спеціальна форма (рис.3.4) для створення стіни . Вона містить кілька полів для введення користувачем : висота , ширина , відстань до підлоги.

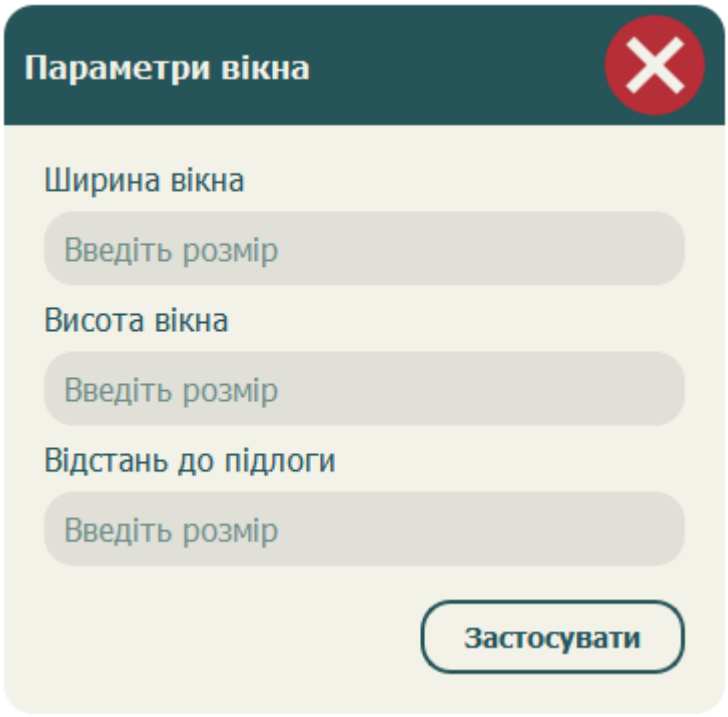

Рис. 3.6 Форма створення вікна

Після заповнення форми користувач натискає кнопку «застосувати» , далі програма перевіряє чи не пусті поля , що обов’язкові для заповнення , якщо вони пусті програма повідомляє про це . Також програма не дозволяє вводити дані в некоректному форматі.

Якщо все гаразд, формується об’єкт , де:

1. генерується унікальний код;
2. усі введені значення зберігаються в модель.

Після успішного збереження форма очищується і користувачу показується повідомлення про успішне створення нового вікна.

3.4.3 Додавання дверей

Жодна кімната не може бути безе дверей , адже без них в неї не можливо було б увійти всередину , отже створено програмне рішення створення дверей рис. 3.7.

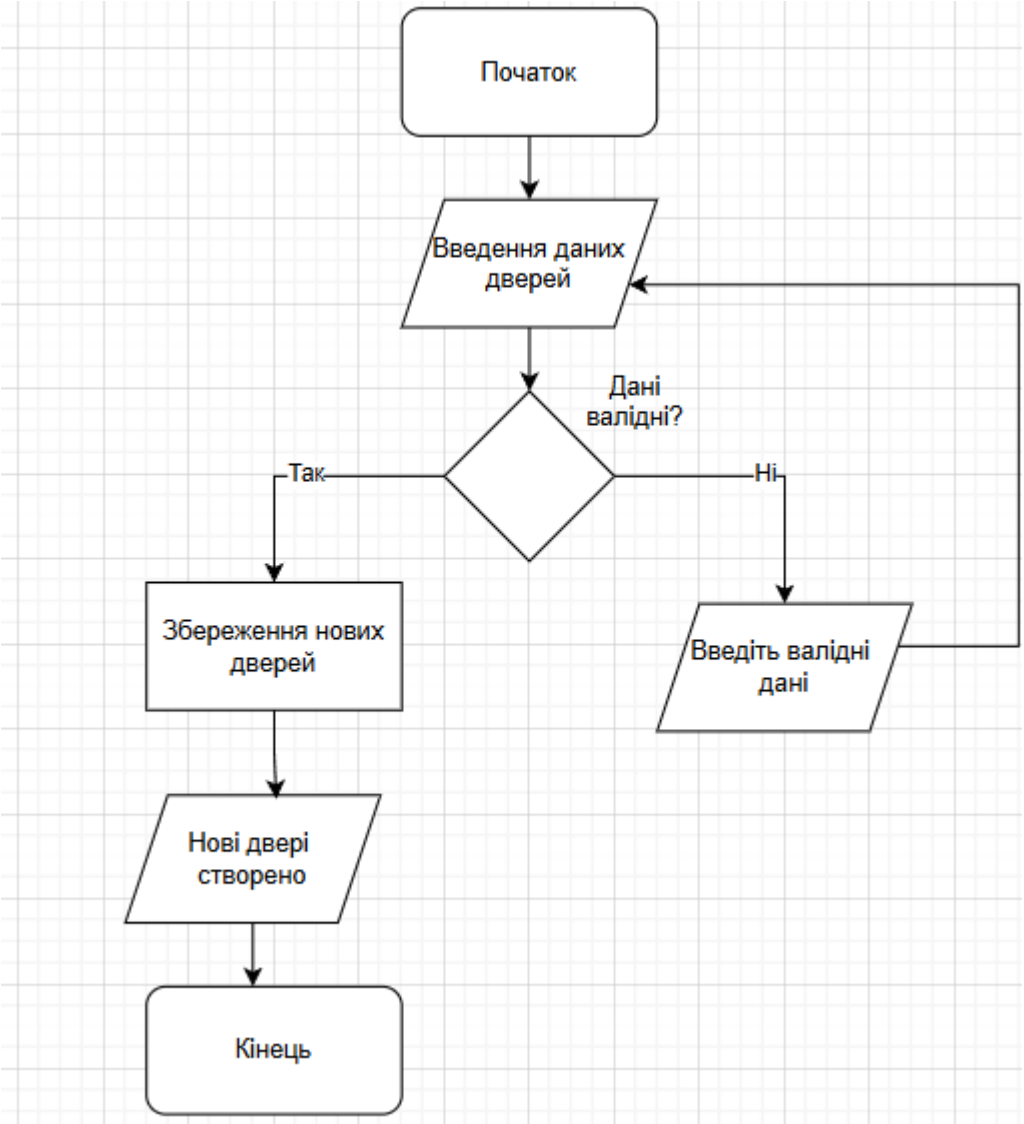

Рис. 3.7 Блок-схема алгоритму створення дверей

У застосунку є спеціальна форма (рис.3.8) для створення дверей . Вона містить кілька полів для введення користувачем : висота , ширина.

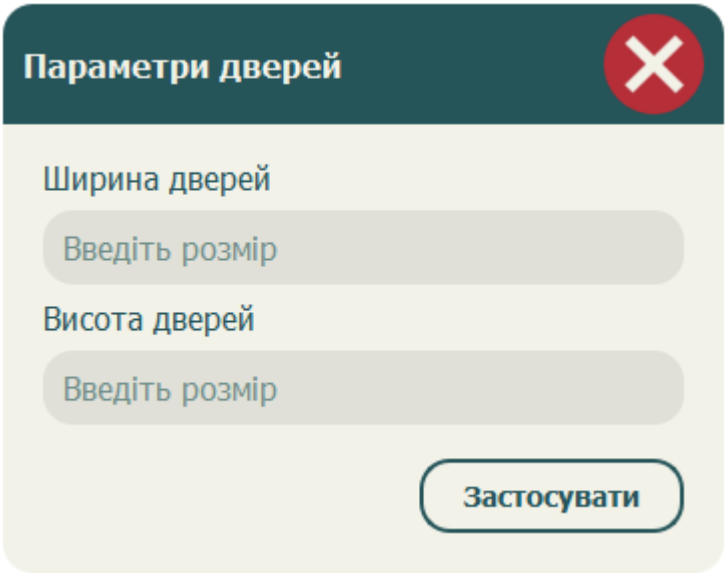

Рис. 3.8 Форма створення дверей

Після заповнення форми користувач натискає кнопку «застосувати» , далі програма перевіряє чи не пусті поля , що обов’язкові для заповнення , якщо вони пусті програма повідомляє про це . Також програма не дозволяє вводити дані в некоректному форматі.

Якщо все гаразд, формується об’єкт , де:

1. генерується унікальний код;
2. усі введені значення зберігаються в модель.

Після успішного збереження форма очищується і користувачу показується повідомлення про успішне створення нових дверей.

Висновки до розділу

У цьому розділі було розроблено інформаційне та програмне забезпечення для прототипу системи моделювання повітрообміну в приміщеннях. Основну увагу приділено реалізації базового графічного інтерфейсу користувача із використанням PyQt5, який дозволяє створювати нові проекти, відкривати наявні та зберігати дані у базу даних SQLite.

Реалізовано вікно проекту з базовими функціями навігації, збереження та візуального представлення структури об'єкта. Створено первинну структуру бази даних, яка дозволяє зберігати параметри проекту, зокрема загальні

характеристики будівлі та зовнішні умови. Також реалізовано можливість передачі даних між формами програми для подальшого розширення функціоналу.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇПРОГРАМНОЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення – це процес, під час якого проводяться експерименти для виявлення помилок і дефектів у програмі. Воно дає змогу переконатися, що ПЗ працює коректно, відповідає вимогам і очікуванням користувачів, а також працює надійно і безпечно.

Тестування є критично важливим етапом життєвого циклу розробки програмного забезпечення, оскільки дозволяє ідентифікувати та усунути проблеми до того, як система потрапить до кінцевих користувачів.

В індустрії розробки ПЗ існує широкий спектр видів тестування, кожен з яких фокусується на певних аспектах системи.

Функціональне тестування перевіряє, чи відповідає програмне забезпечення заданим функціональним вимогам. Воно фокусується на тестуванні функцій, операцій і поведінки програми, а також включає перевірку вхідних даних, правильності обробки даних, роботи функцій і коректності вихідних результатів.

Нефункціональне тестування оцінює якісні атрибути програмного забезпечення, як-от продуктивність, надійність, безпека, зручність використання та сумісність. Приклади нефункціонального тестування включають навантажувальне тестування, регресійне тестування, тестування безпеки, тестування юзабіліті та інші.

Окрім цих основних категорій, у сучасній практиці застосовуються інтеграційне тестування, яке перевіряє взаємодію між різними модулями системи; системне тестування, що оцінює відповідність системи технічним вимогам; приймальне тестування, яке визначає готовність програмного продукту до передачі замовнику; регресійне тестування, що перевіряє, чи не порушили нові зміни існуючу функціональність; стрес-тестування, яке оцінює поведінку

системи при екстремальних навантаженнях; та юніт-тестування, що фокусується на перевірці окремих компонентів коду.

Для тестування розробленої системи було обрано функціональне тестування як найбільш релевантний підхід для вирішення поставлених задач. Цей вибір обумовлений кількома ключовими факторами. По-перше, розроблена система має чітко визначений набір функціональних вимог, тому функціональне тестування дозволяє безпосередньо перевірити відповідність реалізації цим вимогам. По-друге, функціональне тестування фокусується на перевірці що система виконує саме ті завдання, для яких вона була створена, без заглиблення в деталі реалізації.

Розглянемо приклад роботи з програмою , а саме створення проекту то праці з ним .

Спочатку користувач запускає програму та бачить головне вікно рис. 4.1 , після цього натискає на кнопку «створити проект» і відкривається робоче вікно рис. 4.2.

Рис. 4.1 Головне вікно

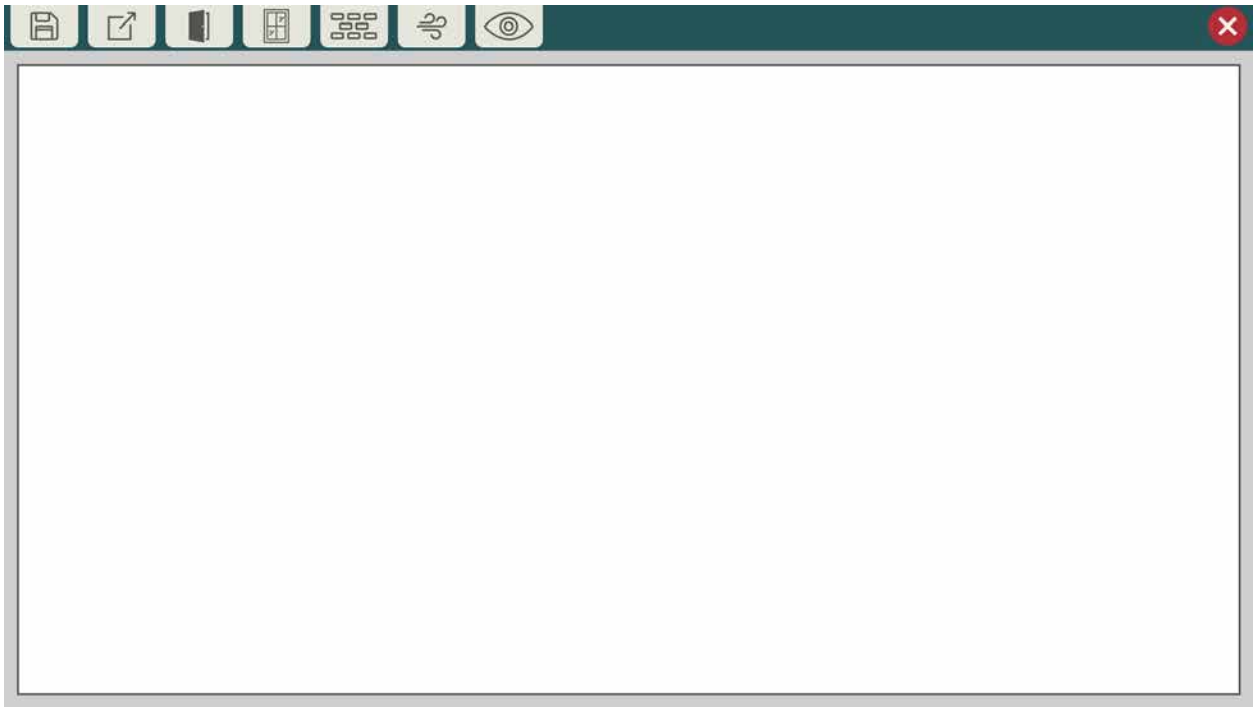

Рис. 4.2 Робоче вікно

Далі щоб створити кімнату , користувач обирає кнопку «стіна» , з’являється діалогове вікно де він вводить необхідні параметри , якщо після кнопки «застосувати» діалогове вікно зникає , це означає що все коректно . Інакше , при помилці буде виведено повідомлення рис. 4.3.

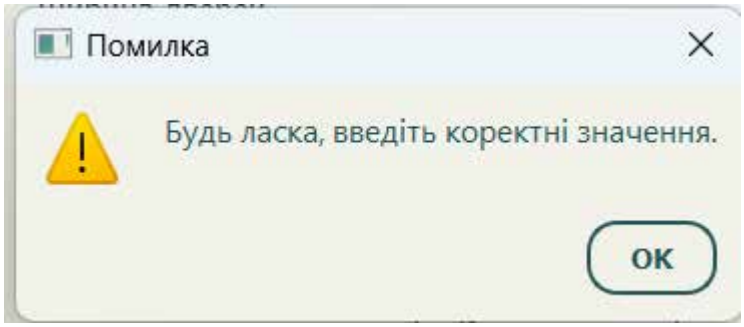

Рис. 4.3 Повідомлення про помилку

Тестування здійснювалося вручну під час етапу розробки. Було перевірено правильність відкриття вікон, валідацію даних при введенні.

Усі перевірені компоненти інтерфейсу працюють згідно з очікуваннями. Функціональність моделювання повітряних потоків наразі не реалізована, тому тестування обмежено графічним інтерфейсом і базовою логікою роботи з об’єктами будівлі.

4.2 Вимоги до апаратного та програмного забезпечення

Опис архітектури

Інформаційна система має локальну архітектуру, в якій усі компоненти розташовані та функціонують на одному пристрої – персональному комп'ютері користувача. Це дозволяє уникнути необхідності підключення до мережі Інтернет або використання віддалених серверів.
На діаграмі розміщення (рис. 4.4) зображено:

- користувач взаємодіє з локальним застосунком через графічний інтерфейс;
- усі обчислення, збереження та обробка даних виконуються на комп'ютері користувача;
- для зберігання даних використовується локальна база даних SQLite, що знаходиться у файловій системі ПК.

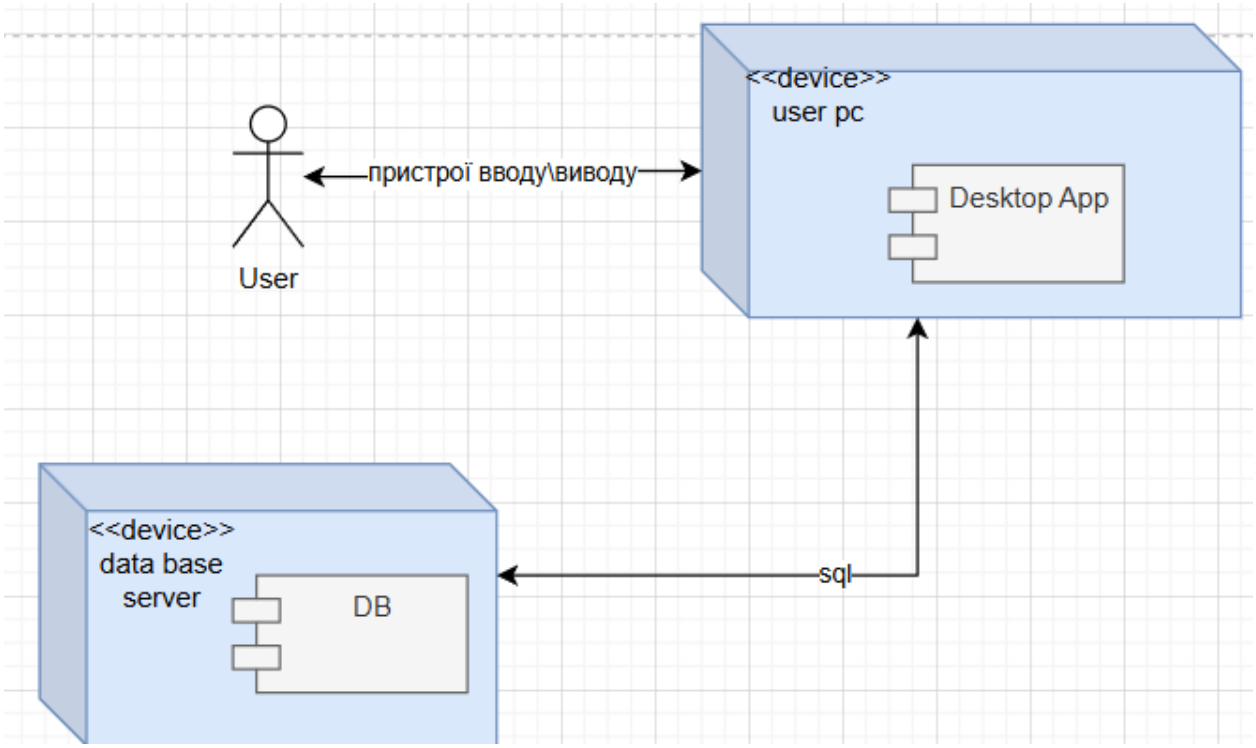

Рис. 4.4 Діаграма розгортання

Апаратні вимоги

Оскільки програмне забезпечення є десктопною системою з незначним навантаженням, апаратні вимоги залишаються мінімальними табл.4.1:

Апаратні вимоги залишаються

Таблиця 4.1

Параметр	**Мінімальна конфігурація**	**Рекомендована конфігурація**
Процесор	Intel Core i3 / AMD Ryzen 3	Intel Core i5+ / Ryzen 5+
Оперативна пам’ять (RAM)	4 ГБ	8 ГБ або більше
Вільне місце на диску	200 МБ	500 МБ (для збереження проєктів)
Дисплей	1280×720	1920×1080 або вище
Інші пристрої	Клавіатура, миша	—

Програмні вимоги

Програмне забезпечення розроблено мовою Python з використанням бібліотеки PyQt5 для створення графічного інтерфейсу, тому для запуску необхідно мати відповідне середовище виконання.

Операційна система:

- Windows 10/11 (x64)
- Linux (Ubuntu 20.04+, Debian, Arch та інші)
- macOS (версії з підтримкою Python 3.10+)

Середовище виконання:

- **Python 3.10 або новіший**

(підтримка бібліотек, типізація, робота з SQLite)

Необхідні бібліотеки Python табл . 4.2:

(Встановлюються через pip або файл requirements.txt)

Необхідні бібліотеки

Таблиця 4.2

Бібліотека	Призначення
PyQt5	Графічний інтерфейс користувача
sqlite3	Вбудована в Python, для роботи з БД
os, sys	Робота з файловою системою та ОС

4.3 Склад інсталяційного пакету

Для успішного впровадження інформаційної системи на стороні користувача необхідно створити інсталяційний пакет, який містить усі компоненти, потрібні для запуску та роботи застосунку без додаткового налаштування середовища.

Склад інсталяційного пакету

Інсталяційний пакет включає такі основні компоненти табл. 4.3:

Основні компоненти

Таблиця 4.3

Компонент	Призначення
main.py	Головний файл запуску застосунку
app/	Модулі інтерфейсу: головне вікно, діалоги, візуалізація
models/	Класи об’єктів: стіни, вікна, двері, будівля
utils/	Допоміжні функції: робота з БД, обробка параметрів
assets/	Іконки, шаблони, файли конфігурації (за потреби)
requirements.txt	Список залежностей для встановлення за допомогою pip

Таблиця 4.3 (закінчення)

README.txt або INSTALL.md	Інструкція щодо встановлення та запуску програми
data/database.db	Порожній або приклад бази даних SQLite (опціонально)
setup.bat / setup.sh	Скрипти автоматизованого встановлення (для Windows/Linux)

Формати інсталяції

Залежно від потреб цільової аудиторії, інсталяційний пакет може бути представлений у таких форматах:

- **Архів .zip або .tar.gz**, що містить всі вказані файли та каталоги.
- **Автономний .exe (Windows)**, зібраний за допомогою PyInstaller або cx_Freeze, який не вимагає попередньо встановленого Python.
- **AppImage або .deb (Linux)** – для пакетування під Linux (за потреби).
- **Portable-режим** – без встановлення, просто розпаковка архіву.

Висновки до розділу

У цьому розділі було детально розглянуто процес підготовки програмної системи до впровадження та її подальшої експлуатації. Здійснено функціональне тестування основних компонентів інтерфейсу користувача, яке підтвердило коректну роботу ключових функцій, зокрема створення проекту, взаємодії з об'єктами та перевірки валідації введених даних.

Також були визначені мінімальні та рекомендовані вимоги до апаратного й програмного забезпечення, що дає змогу впевнено використовувати застосунок на більшості сучасних персональних комп’ютерів без необхідності в додаткових ресурсах. Створено структуру інсталяційного пакету, яка забезпечує простоту

встановлення програми кінцевим користувачем та гарантує її працездатність у середовищах Windows, Linux та macOS.

Таким чином, розроблена система є готовою до локального впровадження у користувача, має зрозумілий механізм інсталяції та не потребує складної підготовки середовища, що значно спрощує її розгортання та подальше використання.

ВИСНОВКИ

У межах виконання бакалаврської кваліфікаційної роботи було розроблено програмний додаток для наукових та інженерних розрахунків, призначене для моделювання повітряних потоків у внутрішніх просторах будівель. Основною метою проекту було створити інструмент, що дозволяє інженерам, проектувальникам та дослідникам ефективно аналізувати циркуляцію повітря з урахуванням конструктивних елементів будівлі. Поставлена мета досягнута повною мірою: система реалізує базовий функціонал для проектування приміщень, обчислення руху повітря та візуалізації результатів у зручному графічному вигляді.

У процесі дослідження було здійснено системний аналіз предметної області, вивчено актуальні методи розрахунку моделювання повітряних потоків, а також проаналізовано наявні інженерні інструменти, що дозволило окреслити основні вимоги до функціоналу та визначити напрямки розвитку власного рішення. На основі цього аналізу було побудовано інформаційну модель, спроектовано архітектуру програмного забезпечення та створено базу даних проектів на основі SQLite.

Система реалізована з використанням мови Python та бібліотеки PyQt5 для побудови графічного інтерфейсу користувача. Реалізовано зручний інтерфейс, який дозволяє користувачеві створювати модель приміщення, додавати стіни, вентиляційні отвори, вікна та двері, задавати зовнішні умови та отримувати візуалізацію повітряних

потоків у 2.5D форматі. В основі розрахунків – спрощені алгоритми, що дозволяють зобразити напрямки руху повітря та визначити зони застою на основі заданої конфігурації будівлі.

Було проведено функціональне тестування системи: перевірено відкриття та збереження проектів, правильність взаємодії графічних елементів, валідацію даних, коректність розрахунків та побудову візуалізації. Програмне забезпечення показало стабільну та передбачувану роботу на типових конфігураціях комп’ютерів, що відповідає вимогам до інженерних інструментів.

Попри завершення основного етапу розробки, проект має значний потенціал для подальшого розвитку. Зокрема, можлива інтеграція складніших алгоритмів розрахунку (наприклад, основ CFD-моделювання), підтримка тривимірної візуалізації, розширення інтерфейсу управління проектами та імпорту/експорту даних, а також створення мобільної або веб-версії застосунку.

Загалом, розроблене рішення повністю готове до використання в навчальних і професійних цілях. Воно поєднує інженерну практичність, зручність використання та архітектурну відкритість, що робить його актуальним інструментом для фахівців, які працюють у галузі проектування мікроклімату будівель та аналізу вентиляції приміщень.

Список використаних джерел

1. Холл, К. Алгоритми: побудова та аналіз / Т. Кормен, Ч. Лейзерсон, Р. Рівест, К. Стайн ; пер. з англ. — Київ: Видавнича група BHV, 2021. — 1296 с.
2. Langtangen, H. P. A Primer on Scientific Programming with Python / Hans Petter Langtangen. — 5th ed. — Springer, 2016. — 922 p. — (Texts in Computational Science and Engineering ; Vol. 6). DOI: 10.1007/978-3-662-49887-3
3. Beazley, D. Python Essential Reference / David Beazley. — 4th ed. — Addison-Wesley Professional, 2009. — 717 p.
4. Šmídl, V., & Quinn, A. The Elements of Statistical Learning in Engineering / V. Šmídl, A. Quinn. — Cambridge University Press, 2022. — 380 p.
5. Badrinarayanan, V. et al. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 2017. — Vol. 39, Issue 12. — P. 2481–2495. DOI: 10.1109/TPAMI.2016.2644615
6. Ghia, U., Ghia, K. N., & Shin, C. T. High-Re Solutions for Incompressible Flow Using Navier-Stokes Equations and a Multigrid Method // Journal of Computational Physics. — 1982. — Vol. 48, Issue 3. — P. 387–411. DOI: 10.1016/0021-9991(82)90058-4
7. PyQt5 Documentation. — The Qt Company. — https://doc.qt.io/qtforpython/
8. SQLite Documentation. — SQLite Consortium. — https://www.sqlite.org/docs.html

Додаток А

Код бази даних

```
-- Видалення таблиць у правильному порядку
DROP TABLE IF EXISTS ventilation;
DROP TABLE IF EXISTS visual;
DROP TABLE IF EXISTS window;
DROP TABLE IF EXISTS door;
DROP TABLE IF EXISTS wall;
DROP TABLE IF EXISTS project;
DROP TABLE IF EXISTS condition;
DROP TABLE IF EXISTS apartment;

-- Створення таблиці apartment
CREATE TABLE apartment (
    apartmentID INTEGER PRIMARY KEY,
    square REAL,
    name_room TEXT
);

-- Створення таблиці condition
CREATE TABLE condition (
    conditionID INTEGER PRIMARY KEY,
    temperature REAL,
    humidity REAL,
    pressure REAL
);

-- Створення таблиці project
CREATE TABLE project (
```

```
    projectID INTEGER PRIMARY KEY,
    name_project TEXT,
    date TEXT,
    apartmentID INTEGER,
    FOREIGN KEY (apartmentID) REFERENCES apartment(apartmentID)
);

-- Створення таблиці wall
CREATE TABLE wall (
    wallID INTEGER PRIMARY KEY,
    apartmentID INTEGER,
    width_wall REAL,
    thickness REAL,
    length_wall REAL,
    FOREIGN KEY (apartmentID) REFERENCES apartment(apartmentID)
);

-- Створення таблиці door
CREATE TABLE door (
    doorID INTEGER PRIMARY KEY,
    wallID INTEGER,
    position_door REAL,
    height_door REAL,
    width_door REAL,
    FOREIGN KEY (wallID) REFERENCES wall(wallID)
);

-- Створення таблиці window
CREATE TABLE window (
```

```
    windowID INTEGER PRIMARY KEY,
    wallID INTEGER,
    position_window REAL,
    height_window REAL,
    width_window REAL,
    FOREIGN KEY (wallID) REFERENCES wall(wallID)
);

-- Створення таблиці visual
CREATE TABLE visual (
    visualID INTEGER PRIMARY KEY,
    apartmentID INTEGER,
    type TEXT,
    FOREIGN KEY (apartmentID) REFERENCES apartment(apartmentID)
);

-- Створення таблиці ventilation
CREATE TABLE ventilation (
    ventilationID INTEGER PRIMARY KEY,
    apartmentID INTEGER,
    position_ventilatin REAL,
    number_of_elements INTEGER,
    conditionID INTEGER,
    FOREIGN KEY (apartmentID) REFERENCES apartment(apartmentID),
    FOREIGN KEY (conditionID) REFERENCES condition(conditionID)
);
```

Додаток Б

Код діалогового вікна «Створення стіни»

```
from PyQt5.QtWidgets import (
    QDialog, QVBoxLayout, QLabel, QLineEdit, QPushButton, QMessageBox,
```

```
    QWidget, QHBoxLayout
)
from PyQt5.QtGui import QDoubleValidator, QIcon
from PyQt5.QtCore import Qt, QSize


class WallDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowFlags(Qt.FramelessWindowHint)
        self.setModal(True)
        self.setMinimumWidth(360)

        self.setStyleSheet("""
            QDialog {
                background-color: #F2F2E9;
                border-radius: 15px;
            }
            QLabel {
                font-size: 16px;
                background: transparent;
                color: #255459;
            }
            QLineEdit {
                background-color: #E0E0D8;
                border: none;
                border-radius: 15px;
                padding: 8px;
                font-size: 16px;
                color: #003b3a;
            }
            QPushButton {
                background-color: transparent;
                border: 2px solid #255459;
                color: #255459;
                font-size: 14px;
                font-weight: bold;
                padding: 8px 20px;
                border-radius: 15px;
            }
            QPushButton:hover {
                background-color: #d8eae9;
```

```
            }
        """)

        # ==== Верхній бар ====
        header = QWidget()
        header.setStyleSheet("background-color: #255459; border-top-left-radius:
15px; border-top-right-radius: 15px;")
        header.setFixedHeight(60)

        header_layout = QHBoxLayout(header)
        header_layout.setContentsMargins(10, 5, 10, 5)

        title = QLabel("Параметри стіни")
        title.setStyleSheet("color: #F2F2E9; font-size: 16px; font-weight:
bold;")

        close_btn = QPushButton("")

close_btn.setIcon(QIcon("C:\\Users\\Lenovo\\PycharmProjects\\Dip\\.venv\\app\\im
age\\close.png"))
        close_btn.setFixedWidth(50)
        close_btn.setIconSize(QSize(30, 40))
        close_btn.setStyleSheet("""
            QPushButton {
                background-color: #B52E38;
                color: white;
                border: none;
                font-size: 16px;
                border-radius: 25px;
            }
            QPushButton:hover {
                background-color: #c75c66;
            }
        """)
        close_btn.clicked.connect(self.reject)

        header_layout.addWidget(title)
        header_layout.addStretch()
        header_layout.addWidget(close_btn)

        # ==== Основна частина ====
        main_layout = QVBoxLayout(self)
```

```
        main_layout.setContentsMargins(0, 0, 0, 0)
        main_layout.addWidget(header)

        form_layout = QVBoxLayout()
        form_layout.setContentsMargins(20, 10, 20, 0)

        # Поля вводу
        self.wall_height_input = QLineEdit()
        self.wall_height_input.setValidator(QDoubleValidator(0.0, 10000.0, 2))
        self.wall_height_input.setPlaceholderText("Введіть розмір")

        self.wall_thickness_input = QLineEdit()
        self.wall_thickness_input.setValidator(QDoubleValidator(0.0, 10000.0,
2))
        self.wall_thickness_input.setPlaceholderText("Введіть розмір")

        form_layout.addWidget(QLabel("Висота стіни"))
        form_layout.addWidget(self.wall_height_input)

        form_layout.addWidget(QLabel("Товщина стіни"))
        form_layout.addWidget(self.wall_thickness_input)

        main_layout.addLayout(form_layout)

        # ==== Кнопка "Застосувати" ====
        submit_btn = QPushButton("Застосувати")
        submit_btn.clicked.connect(self.accept_inputs)

        btn_layout = QHBoxLayout()
        btn_layout.setContentsMargins(10, 10, 20, 20)
        btn_layout.addStretch()
        btn_layout.addWidget(submit_btn)

        main_layout.addLayout(btn_layout)

        self.adjustSize()

    def accept_inputs(self):
        try:
            wall_height = float(self.wall_height_input.text())
            wall_thickness = float(self.wall_thickness_input.text())
            # Тут можна передати ці значення назад до головної форми
```

```
            self.accept()
        except ValueError:
            QMessageBox.warning(self, "Помилка", "Будь ласка, введіть коректні значення.")
```

Код діалогового вікна «Створення вікна»

```
from PyQt5.QtWidgets import (
    QDialog, QVBoxLayout, QLabel, QLineEdit, QPushButton, QMessageBox,
    QWidget, QHBoxLayout
)
from PyQt5.QtGui import QDoubleValidator, QIcon
from PyQt5.QtCore import Qt, QSize
class WindowDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowFlags(Qt.FramelessWindowHint)
        self.setModal(True)
        self.setMinimumWidth(360)

        self.setStyleSheet("""
            QDialog {
                background-color: #F2F2E9;
                border-radius: 15px;
            }
            QLabel {
                font-size: 16px;
                background: transparent;
                color: #255459;
            }
            QLineEdit {
                background-color: #E0E0D8;
                border: none;
                border-radius: 15px;
                padding: 8px;
                font-size: 16px;
                color: #003b3a;
            }
            QPushButton {
                background-color: transparent;
                border: 2px solid #255459;
                color: #255459;
                font-size: 14px;
```

```
            font-weight:                                    bold;
            padding:                    8px                 20px;
            border-radius:                                  15px;
        }
        QPushButton:hover                                       {
            background-color:                               #d8eae9;
        }
    """)

    # Верхній бар
    header = QWidget()
    header.setStyleSheet("background-color: #255459; border-top-left-radius:
15px; border-top-right-radius: 15px;")
    header.setFixedHeight(60)

    header_layout = QHBoxLayout(header)
    header_layout.setContentsMargins(10, 5, 10, 5)

    title = QLabel("Параметри вікна")
    title.setStyleSheet("color: #F2F2E9; font-size: 16px; font-weight: bold;")

    close_btn = QPushButton("")

close_btn.setIcon(QIcon("C:\\Users\\Lenovo\\PycharmProjects\\Dip\\.venv\\app\\im
age\\close.png"))
    close_btn.setFixedWidth(50)
    close_btn.setIconSize(QSize(30,                             40))
    close_btn.setStyleSheet("""
        QPushButton                                             {
            background-color:                               #B52E38;
            color:                                            white;
            border:                                           none;
            font-size:                                        16px;
            border-radius:                                    25px;
        }
        QPushButton:hover                                       {
            background-color:                               #c75c66;
        }
    """)
    close_btn.clicked.connect(self.reject)

    header_layout.addWidget(title)
```

```
        header_layout.addStretch()
        header_layout.addWidget(close_btn)

        # Основна частина
        main_layout = QVBoxLayout(self)
        main_layout.setContentsMargins(0, 0, 0, 0)
        main_layout.addWidget(header)

        form_layout = QVBoxLayout()
        form_layout.setContentsMargins(20, 10, 20, 0)

        # Поля для вікна
        self.window_width_input = QLineEdit()
        self.window_width_input.setValidator(QDoubleValidator(0.0, 10000.0, 2))
        self.window_width_input.setPlaceholderText("Введіть розмір")

        self.window_height_input = QLineEdit()
        self.window_height_input.setValidator(QDoubleValidator(0.0, 10000.0, 2))
        self.window_height_input.setPlaceholderText("Введіть розмір")

        self.window_distance_input = QLineEdit()
        self.window_distance_input.setValidator(QDoubleValidator(0.0, 10000.0,
2))
        self.window_distance_input.setPlaceholderText("Введіть розмір")

        form_layout.addWidget(QLabel("Ширина вікна"))
        form_layout.addWidget(self.window_width_input)

        form_layout.addWidget(QLabel("Висота вікна"))
        form_layout.addWidget(self.window_height_input)

        form_layout.addWidget(QLabel("Відстань до підлоги"))
        form_layout.addWidget(self.window_distance_input)

        main_layout.addLayout(form_layout)

        # Кнопка "Застосувати"
        submit_btn = QPushButton("Застосувати")
        submit_btn.clicked.connect(self.accept_inputs)

        btn_layout = QHBoxLayout()
        btn_layout.setContentsMargins(10, 10, 20, 20)
```

```
        btn_layout.addStretch()
        btn_layout.addWidget(submit_btn)

        main_layout.addLayout(btn_layout)

        self.adjustSize()                  # Автоматична висота

    def accept_inputs(self):
        try:
            window_width = float(self.window_width_input.text())
            window_height = float(self.window_height_input.text())
            window_distance = float(self.window_distance_input.text())

            self.accept()

        except ValueError:
            QMessageBox.warning(self, "Помилка", "Будь ласка, введіть коректні значення.")
```

Код діалогового вікна «Створення дверей»

```
from PyQt5.QtWidgets import (
    QDialog, QVBoxLayout, QLabel, QLineEdit, QPushButton, QMessageBox,
    QWidget, QHBoxLayout
)
from PyQt5.QtGui import QDoubleValidator, QIcon
from PyQt5.QtCore import Qt, QSize
class DoorDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowFlags(Qt.FramelessWindowHint)
        self.setModal(True)
        self.setMinimumWidth(360)

        self.setStyleSheet("""
            QDialog {
                background-color: #F2F2E9;
                border-radius: 15px;
            }
            QLabel {
                font-size: 16px;
                background: transparent;
                color: #255459;
```

```
            }
            QLineEdit                                                    {
                background-color:                                    #E0E0D8;
                border:                                                 none;
                border-radius:                                          15px;
                padding:                                                 8px;
                font-size:                                              16px;
                color:                                               #003b3a;
            }
            QPushButton                                                  {
                background-color:                                transparent;
                border:              2px              solid              #255459;
                color:                                               #255459;
                font-size:                                              14px;
                font-weight:                                            bold;
                padding:                         8px                    20px;
                border-radius:                                          15px;
            }
            QPushButton:hover                                            {
                background-color:                                    #d8eae9;
            }
        """)

        #                                Верхній                              бар
        header                                =                          QWidget()
        header.setStyleSheet("background-color: #255459; border-top-left-radius:
15px;                      border-top-right-radius:                      15px;")
        header.setFixedHeight(60)

        header_layout                    =                   QHBoxLayout(header)
        header_layout.setContentsMargins(10,          5,          10,          5)

        title              =              QLabel("Параметри              дверей")
        title.setStyleSheet("color: #F2F2E9; font-size: 16px; font-weight: bold;")

        close_btn                        =                        QPushButton("")

close_btn.setIcon(QIcon("C:\\Users\\Lenovo\\PycharmProjects\\Dip\\.venv\\app\\im
age\\close.png"))
        close_btn.setFixedWidth(50)
        close_btn.setIconSize(QSize(30,                                  40))
        close_btn.setStyleSheet("""
```

```
    QPushButton                                                    {
        background-color:                                      #B52E38;
        color:                                                   white;
        border:                                                   none;
        font-size:                                                16px;
        border-radius:                                            25px;
    }
    QPushButton:hover                                              {
        background-color:                                      #c75c66;
    }
""")
close_btn.clicked.connect(self.reject)

header_layout.addWidget(title)
header_layout.addStretch()
header_layout.addWidget(close_btn)

#                              Основна                              частина
main_layout                        =                      QVBoxLayout(self)
main_layout.setContentsMargins(0,           0,           0,           0)
main_layout.addWidget(header)

form_layout                          =                        QVBoxLayout()
form_layout.setContentsMargins(20,          10,          20,          0)

#                    Поля                    для                    дверей
self.door1_width_input                    =                    QLineEdit()
self.door1_width_input.setValidator(QDoubleValidator(0.0,  10000.0,  2))
self.door1_width_input.setPlaceholderText("Введіть              розмір")

self.door1_height_input                    =                    QLineEdit()
self.door1_height_input.setValidator(QDoubleValidator(0.0,  10000.0,  2))
self.door1_height_input.setPlaceholderText("Введіть              розмір")

form_layout.addWidget(QLabel("Ширина                            дверей"))
form_layout.addWidget(self.door1_width_input)

form_layout.addWidget(QLabel("Висота                            дверей"))
form_layout.addWidget(self.door1_height_input)

main_layout.addLayout(form_layout)
```

```
        # Кнопка "Застосувати"
        submit_btn = QPushButton("Застосувати")
        submit_btn.clicked.connect(self.accept_inputs)

        btn_layout = QHBoxLayout()
        btn_layout.setContentsMargins(10, 10, 20, 20)
        btn_layout.addStretch()
        btn_layout.addWidget(submit_btn)

        main_layout.addLayout(btn_layout)

        self.adjustSize() # Автоматична висота

    def accept_inputs(self):
        try:
            door1_width = float(self.door1_width_input.text())
            door1_height = float(self.door1_height_input.text())
            door2_width = float(self.door2_width_input.text())
            door2_height = float(self.door2_height_input.text())

            self.accept()

        except ValueError:
            QMessageBox.warning(self, "Помилка", "Будь ласка, введіть коректні значення.")
```