НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО **Декан факультету**	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** **Завідувач кафедри інформаційних систем і технологій**
_____ Ігор БОЛБОТ (підпис)	**_____ Михайло ШВИДЕНКО** (підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна система моделювання впливу умов експлуатації на електронні прилади»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми д.е.н., професор	_____ (підпис)	**Роман РУДЕНСЬКИЙ**
Керівник бакалаврської кваліфікаційної роботи науковий ступінь та вчене звання	_____ (підпис)	**Вікторія СМОЛІЙ**
Виконав	_____ (підпис)	**Олексій МАКСИМЕЦЬ**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних систем і технологій

к.е.н., доцент _____ **Михайло ШВИДЕНКО**
(підпис)

«___» _____ 2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

_____Максимець Олексій Вікторович_____
(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«Інформаційна система моделювання впливу умов експлуатації на електронні прилади» затверджена наказом від «6» січня 2026 р. № 46 «С»
Термін подання завершеної роботи на кафедру «25» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
Спосіб збору, задання та обробки параметрів умов експлуатації електронних приладів з метою моделювання їх впливу на технічні характеристики, надійність та працездатність електронних компонентів.
Перелік питань, що підлягають дослідженню:

- Аналіз умов експлуатації електронних приладів та факторів їх впливу на надійність і функціональні характеристики.
- Проєктування та розробка інформаційного забезпечення системи моделювання впливу умов експлуатації на електронні прилади.
- Проєктування та розробка програмного забезпечення системи.
- Впровадження та експлуатація інформаційної системи.

Перелік графічних документів (за необхідності).
Дата видачі завдання «6» січня 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Вікторія СМОЛІЙ**

Завдання взяв до виконання _____ (підпис) **Олексій МАКСИМЕЦЬ**

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних.

ІС – інформаційна система.

ППЗ – прикладне програмне забезпечення.

СУБД – система управління базами даних.

UML – уніфікована мова моделювання.

ER-діаграма – діаграма сутностей і зв’язків.

GUI – графічний інтерфейс користувача.

API – програмний інтерфейс взаємодії модулів.

JSON – текстовий формат обміну структурованими даними.

SQLite – вбудована реляційна система керування базами даних.

SQL – мова структурованих запитів.

CAD – система автоматизованого проєктування.

CAE – система інженерного аналізу.

UUID – універсальний унікальний ідентифікатор.

PyQt5 – бібліотека Python для створення графічного інтерфейсу.

Matplotlib – бібліотека для побудови графіків.

Kivy – бібліотека Python для створення мобільних інтерфейсів.

Buildozer – інструмент для створення Android-збірок Python-додатків.

APK – інсталяційний файл Android-додатку.

SDK – набір засобів розробки програмного забезпечення.

NDK – набір засобів для роботи з нативним кодом Android.

QGraphicsScene – клас для роботи з графічною сценою у PyQt.

QGraphicsView – клас для відображення графічної сцени у PyQt.

ВСТУП

Під час розробки електронних систем важливе значення має не лише правильність електричної схеми, а й можливість аналізу поведінки компонентів у структурі пристрою. У реальних умовах навіть незначні зміни конфігурації з’єднань можуть впливати на стабільність роботи системи, поширення сигналу та загальну надійність пристрою. Через це під час проєктування виникає потреба у програмних засобах, які дозволяють поєднати структурне представлення системи з моделюванням її поведінки.

Значна частина сучасних CAD та CAE-систем орієнтована на промислове використання і містить велику кількість спеціалізованих модулів для електротехнічного аналізу, трасування друкованих плат та фізичного моделювання процесів. Через складність таких платформ користувачу часто необхідно витрачати додатковий час на налаштування середовища, бібліотек компонентів і параметрів розрахунку. Для навчальних або демонстраційних задач подібний підхід не завжди є доцільним.

У межах проекту було створено програмний додаток, який дозволяє не лише зберігати схему пристрою, а й спостерігати за зміною надійності у часі. Система не замінює промислові комплекси електротехнічного проектування, але дає зрозумілий інструмент для попереднього аналізу структури, перевірки впливу зовнішніх умов і формування технічного звіту. Саме така прикладна спрямованість робить роботу корисною для навчального та демонстраційного використання.

Актуальність теми пов’язана з тим, що електронні пристрої працюють не в ідеальних умовах. На їхню роботу впливають фактори навколишнього середовища та загальна структура з’єднань. Навіть простий пристрій може поводитися по-різному, якщо один і той самий набір компонентів використовується в сухому приміщенні, у вологому середовищі або біля

джерела механічних коливань. Через це оцінка надійності електронних приладів є практично важливою задачею.

Метою розробки є створення інформаційної системи моделювання впливу умов експлуатації на електронні прилади. Для досягнення мети було передбачено формування графової моделі електронної структури, реалізацію введення та редагування компонентів, збереження проектів у базі даних і файлах, виконання симуляції, побудову графіків та подача звітів українською й англійською мовами.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі подано системний аналіз предметної області, постановку задачі та моделі майбутньої системи. У другому розділі описано інформаційне забезпечення, логічну модель даних і структуру бази. У третьому розділі наведено прикладне програмне забезпечення, вибір інструментів і опис алгоритмів. У четвертому розділі подано тестування, вимоги до середовища та склад інсталяційного пакету.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Предметна область індивідуального проекту пов’язана з оцінюванням надійності електронних приладів під впливом умов експлуатації. Електронний пристрій у межах роботи подається як сукупність компонентів, що з’єднані між собою та передають живлення від джерела до навантаження. Так можна описувати пристрій не лише як набір елементів, а як структуру, де важливе значення має порядок з’єднання, наявність резервних гілок і працездатність кожного вузла.

Одним із ключових завдань системи є автоматична перевірка цілісності структури електронної моделі перед запуском симуляції. Програмний додаток повинен визначати наявність коректного шляху між джерелом живлення та кінцевими вузлами, виявляти ізольовані компоненти та аналізувати конфігурацію з’єднань. Такий підхід дозволяє зменшити кількість помилок під час побудови моделі та підвищує достовірність результатів симуляції.

Користувач системи виконує побудову структури електронної моделі шляхом додавання компонентів, редагування їх параметрів та створення зв’язків між вузлами. Після введення даних програмний додаток автоматично виконує перевірку коректності структури, класифікацію типу системи, аналіз зв’язності компонентів, розрахунок показників надійності, пошук критичних вузлів, побудову графіка результатів і формування технічного звіту. Також автоматизовано збереження результатів симуляції та повторне завантаження проектів із бази даних або JSON-файлів.

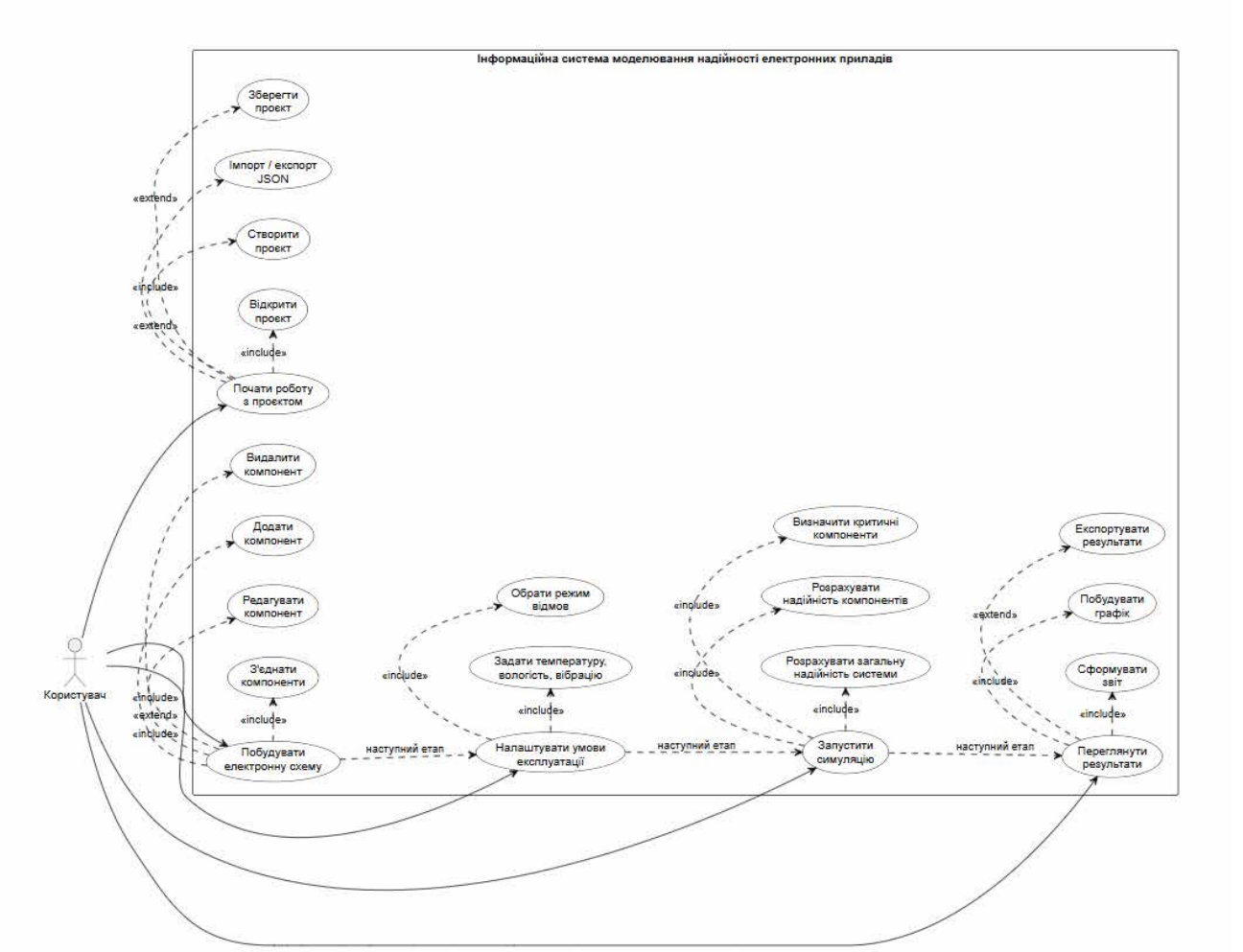


Рис. 1.1 – Діаграма прецедентів користувача системи.

На діаграмі прецедентів *рис. 1.1* передбачено одного основного актора – користувача програмного додатку. Він створює проект, додає систему, працює з компонентами, зберігає дані, запускає симуляцію та переглядає результати. Така модель показує межі відповідальності системи та допомагає відокремити ручні дії користувача від автоматичних дій програмного забезпечення.

Основними вимогами користувача є простота введення даних, наочне подання структури, швидкий запуск симуляції та зрозумілий звіт. Через це в інтерфейсі використано графічну сцену, панелі параметрів, окреме вікно налаштувань проекту та модальне вікно звіту. У результаті зменшується

кількість зайвих переходів і дозволяє працювати з моделлю без ручного редагування службових файлів.

Функціональні вимоги, наведені у таблиці 1.2, визначають основні можливості програмного додатку та перелік дій, які система повинна підтримувати під час роботи користувача. До них належать створення та збереження проектів, додавання і редагування компонентів, побудова зв’язків між вузлами та аналіз структури електронної системи. Окрему увагу приділено підтримці роботи з кількома системами в межах одного проекту, а також можливості імпорту й експорту даних у форматі JSON. Також система повинна забезпечувати автоматичне визначення типу структури та пошук критичних вузлів, які найбільше впливають на загальну надійність системи.

Функціональні вимоги до системи *таблиця 1.2.*

Таблиця 1.2 – Функціональні вимоги системи.

№	**Функціональна вимога**	**Опис**
1	Створення проєкту	Можливість створення нового проєкту моделювання
2	Збереження проєкту	Збереження структури системи та параметрів у базі даних
3	Імпорт та експорт JSON	Завантаження та збереження проєктів у форматі JSON
4	Робота з декількома системами	Підтримка декількох електронних систем у межах одного проєкту
5	Додавання компонентів	Додавання електронних елементів до робочої області
6	Редагування компонентів	Зміна параметрів і характеристик компонентів

7	Видалення компонентів	Видалення вузлів та з’єднань зі структури системи
8	Побудова з’єднань	Формування зв’язків між компонентами
9	Аналіз структури	Автоматичне визначення типу структури системи
10	Визначення критичних вузлів	Пошук компонентів, які найбільше впливають.

Нефункціональні вимоги *таблиця 1.3*

Таблиця 1.3 – Не функціональні вимоги системи.

№	**Нефункціональна вимога**	**Опис**
1	Зручність інтерфейсу	Інтерфейс повинен бути зрозумілим та простим у використанні
2	Стабільність роботи	Система повинна працювати без критичних помилок
3	Продуктивність	Симуляція повинна виконуватись за короткий проміжок часу
4	Масштабованість	Можливість роботи з великими схемами
5	Модульність	Програмний код повинен мати модульну структуру
6	Розширюваність	Можливість додавання нових компонентів та алгоритмів
7	Локальне збереження даних	Робота без необхідності використання серверів
8	Підтримка Windows	Коректна робота у середовищі Windows
9	Збереження структури системи	Відсутність втрати даних під час збереження
10	Візуалізація результатів	Автоматичне оновлення графіків та звітів
11	Мінімальні системні вимоги	Можливість запуску на звичайних персональних комп’ютерах
12	Підтримка бази	Використання SQLite для

	даних	збереження результатів

Нефункціональні вимоги визначають загальні характеристики програмного додатку, які впливають на зручність роботи, стабільність системи та можливість її подальшого розвитку. На відміну від функціональних вимог, вони описують не конкретні дії користувача, а умови, за яких система повинна працювати. У таблиці 1.3 наведено основні нефункціональні вимоги, які були враховані під час проектування та реалізації програмного додатку.

1.2 Огляд інформаційних джерел та існуючих рішень

У процесі аналізу існуючих рішень було встановлено, що більшість професійних систем орієнтована на точне електротехнічне або мультифізичне моделювання. До таких систем належать ANSYS Sherlock та COMSOL Multiphysics, порівняння яких наведено в *табл. 1.1*. Ці програмні комплекси дозволяють виконувати складні теплові, механічні та електричні розрахунки, аналізувати поведінку електронних компонентів у різних умовах експлуатації та прогнозувати можливі відмови. Водночас подібні системи мають складний інтерфейс, високий поріг входження та потребують значних обчислювальних ресурсів, що ускладнює їх використання для навчальних цілей.

Окрему увагу було приділено підходам до оцінки надійності електронних систем. У класичних моделях прогнозування враховуються зовнішні та експлуатаційні фактори, що впливають на деградацію компонентів [1]. Подібні підходи використовуються у промислових методиках аналізу надійності електронного обладнання та дозволяють оцінювати деградацію компонентів у процесі роботи. Однак більшість професійних рішень зосереджена переважно на точності математичних моделей і меншою мірою орієнтована на наочне представлення структури системи та візуальний аналіз впливу окремих факторів.

Таблиця 1.1 – порівняння з існуючими аналогами.

Критерій	**Розроблена система**	**ANSYS Sherlock**	**COMSOL Multiphysics**
Основне призначення	Моделювання впливу температури, вологості, вібрації та часу на надійність електронних компонентів	Прогнозування надійності електронних плат і компонентів	Мультифізичне моделювання електричних, теплових і механічних процесів
Урахування температури	+	+	+
Урахування вібрації	+	+	+
Урахування вологості	+	Частково/через моделі середовища	Можливо через фізичні моделі
Побудова структури з компонентів	+	+ переважно для плат	+
Аналіз надійності	+	Основна функція	Можливий через моделі
Графіки результатів	+	+	+
Формування звіту	+	+	+
Збереження у БД	+, SQLite	- не як основна функція	- не як основна функція
Простота використання	Вища, бо система вузько орієнтована на навчальну задачу	Складна професійна система	Складна професійна система
Недолік	Спрощені математичні моделі	Складність і комерційність	Високий поріг входження

Під час аналізу предметної області було визначено, що для навчального програмного додатку важливими є не лише математичні розрахунки, а й можливість візуально демонструвати вплив температури, вологості, вібрації та

часу роботи на різні типи компонентів. Саме тому в роботі використано спрощений підхід до моделювання, який дозволяє поєднати графову структуру системи, симуляцію деградації компонентів, побудову графіків та автоматичне генерування технічного звіту в межах одного програмного додатку[2].

1.3 Моделювання предметної області

Моделювання предметної області виконано за допомогою UML-діаграм та структурних схем. Основна ідея полягає в тому, що електронна система подається у вигляді графа. Вузли графа відповідають компонентам, а ребра – зв’язкам між ними. Джерелами вважаються power, battery та voltage_source. Навантаженнями вважаються motor, sensor, cooler і controller_unit. Якщо явне навантаження відсутнє, крайні елементи графа можуть розглядатися як кінцеві вузли.

Для опису поведінки користувача було рис.1.2 подано діаграму активності системи моделювання. На початку користувач створює або відкриває проект, після чого додає компоненти, налаштовує їх параметри та формує електронну структуру шляхом створення зв’язків між вузлами. Далі виконується перевірка коректності схеми. Якщо структура містить помилки або не має допустимого шляху між джерелом живлення та навантаженням, програмний додаток повідомлення про помилку. У разі успішної перевірки користувач задає параметри середовища: температуру, вологість, рівень вібрації та час моделювання, після чого запускається симуляція. Під час виконання алгоритму окремо обчислюється надійність компонентів, виконується передача напруги через елементи та аналізується структура системи. Після завершення розрахунків визначає загальну надійність, знаходить критичні компоненти, будує графік, формує технічний звіт і зберігає результати моделювання.

У структурі системи виділено кілька основних модулів: інтерфейс користувача, модель графа, модуль симуляції, модуль збереження, модуль бази

даних, модуль компонентів і модуль звітності. Такий поділ дозволяє не змішувати код інтерфейсу з кодом розрахунків. Під час подальшого розширення можна змінити модель компонента без повного переписування вікон програми

На рис. 1.3 наведено UML-діаграму послідовності взаємодії основних модулів програмного додатку під час роботи користувача із системою. Діаграма показує процес створення або відкриття проекту, побудови електронної структури, налаштування параметрів симуляції та запуску розрахунків.

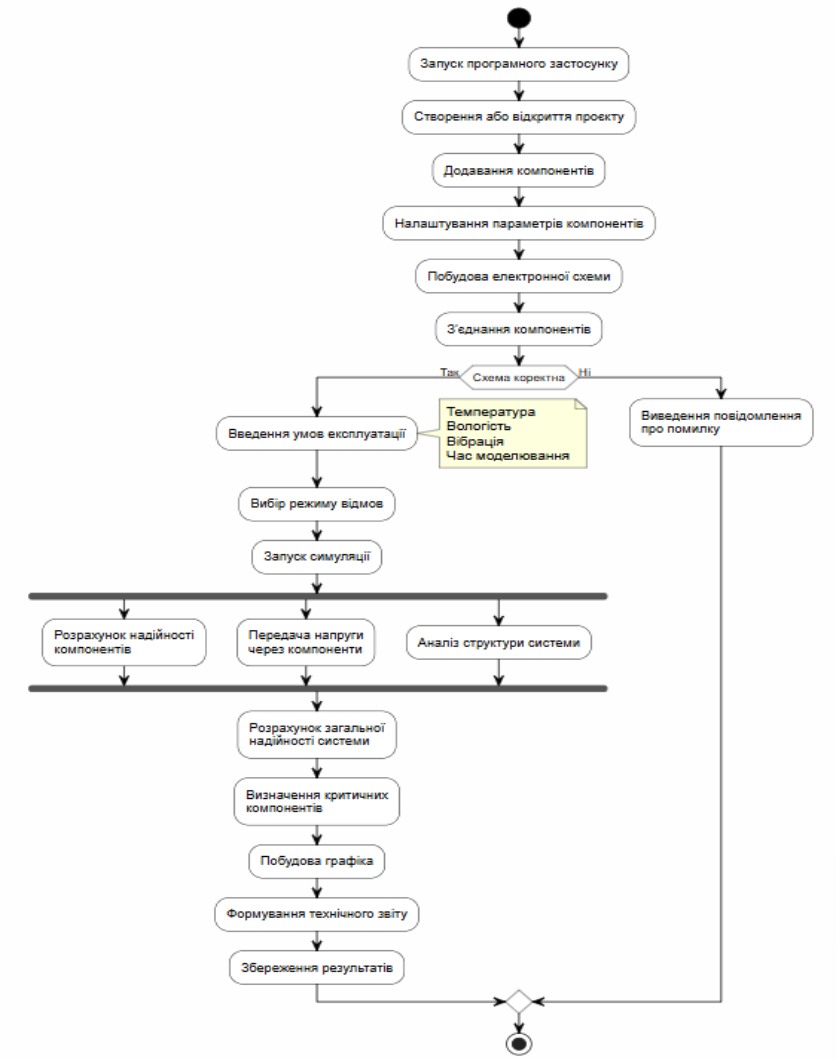


Рис. 1.2 – Діаграма активності.

Користувач взаємодіє з графічним інтерфейсом GUI CircuitEditor, і далі дані передаються до моделі структури CircuitModel та модуля симуляції

Simulator. Під час запуску симуляції модуль Simulator отримує інформацію про вузли та зв’язки системи, виконує цикл розрахунку надійності для кожного компонента через модулі Component Models, а потім формує загальний коефіцієнт надійності системи та визначає критичні компоненти. Після завершення обчислень виконується побудова графіка через модуль Chart Module, створення технічного звіту через Report Generator та збереження результатів у базі даних.

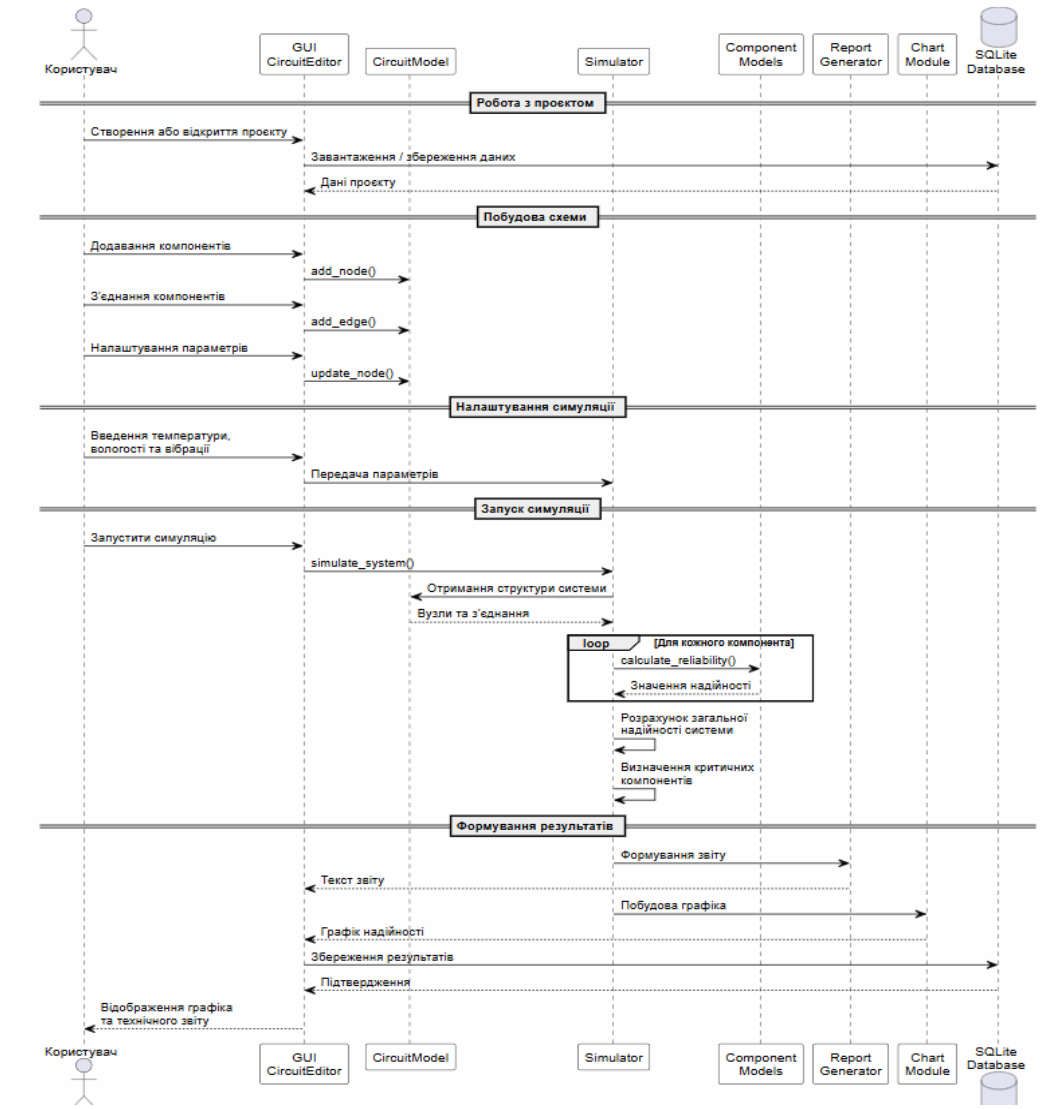


Рис. 1.3 – Діаграма послідовностей.

Для планування реалізації використано поділ на короткі етапи. Спочатку було сформовано концепцію та вимоги, потім створено графічний інтерфейс, після цього реалізовано базу даних, JSON-обмін і симуляцію[3]. Окремим етапом виконано локалізацію інтерфейсу й звіту українською мовою, а також покращення графіків. Такий поділ відповідає спринтовому підходу й дає можливість поступово отримувати працездатні версії додатку.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Інформаційне забезпечення програмного додатку побудовано навколо кількох сутностей: проект, система, компонент, зв’язок і запуск симуляції. Проект містить загальні метадані: назву, автора, опис і дату створення. Система належить до проекту та містить параметри експлуатації: температуру, вологість, вібрацію, максимальний час симуляції, режим відмов і поріг відмови.

Компонент є окремим вузлом графа. Для нього зберігається ідентифікатор, тип, назва, координати на графічній сцені та набір параметрів у форматі JSON. Такий спосіб дозволяє зберігати різні параметри для різних типів компонентів без створення окремої таблиці для кожного типу. Наприклад, резистор має resistance, конденсатор має capacitance, реле має coil_voltage, мотор має rated_power.

Зв’язок описує ребро графа. Він містить ідентифікатор зв’язку, вихідний вузол, порт вихідного вузла, цільовий вузол і порт цільового вузла. Завдяки цьому можна зберігати не лише факт з’єднання, а й напрямок побудови структури в інтерфейсі. Для розрахунків граф обробляється як мережа шляхів від джерела до навантаження.

Запуск симуляції зберігає назву проекту, назву системи, фінальну надійність, дані компонентів у JSON та текст сформованого звіту. Це дозволяє не втрачати результати після закриття програми. У разі потреби останні результати можна порівнювати з новими запусками після зміни умов експлуатації.

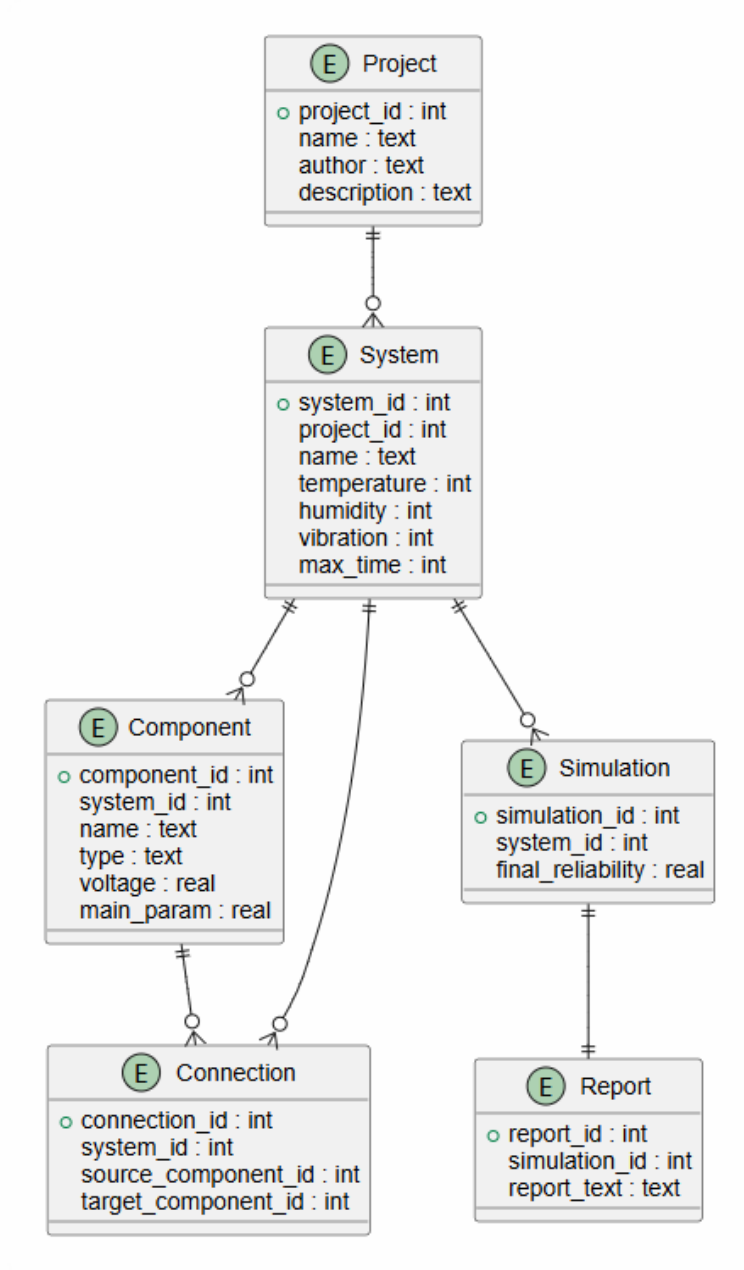


Рис. 2.1 – ER-діаграма інформаційної бази

Логічна модель даних *рис. 2.1* відповідає вимогам збереження проектної інформації. У ній відокремлено загальні відомості про проект, параметри систем, вузли графа, зв’язки та результати симуляції. Такий поділ зменшує дублювання даних і дозволяє завантажувати проект разом з усіма його системами.

Важливою особливістю є використання JSON для параметрів компонента. У різних компонентів набір характеристик не збігається. Зберігання параметрів у текстовому JSON-полі дозволяє додавати нові типи компонентів без зміни структури таблиці project_nodes. Це не порушує логіку додатку, оскільки обробка параметрів виконується на рівні Python-коду.

2.2 Вибір системи управління інформаційною базою данних

Для інформаційної бази обрано SQLite. Цей вибір пояснюється тим, що програмний додаток є локальним і не потребує мережевого доступу до бази. SQLite зберігає дані в одному файлі, не потребує окремого сервера та підтримується стандартним модулем Python sqlite3 [5]. Це спрощує встановлення і запуск системи на комп’ютері користувача.

Для бакалаврського індивідуального проекту SQLite є практичним варіантом, оскільки дозволяє продемонструвати роботу з таблицями, записом, читанням, оновленням і видаленням даних. У програмі реалізовано збереження проекту, завантаження проекту зі списку, видалення проекту та запис результатів симуляції.

Порівняно з серверними СУБД, SQLite має менші вимоги до налаштування. Для невеликого локального додатку це є перевагою. Обмеженням є те, що база не орієнтована на одночасну роботу великої кількості користувачів. Однак у межах розробленої системи така можливість не є обов’язковою, тому вибір є обґрунтованим.

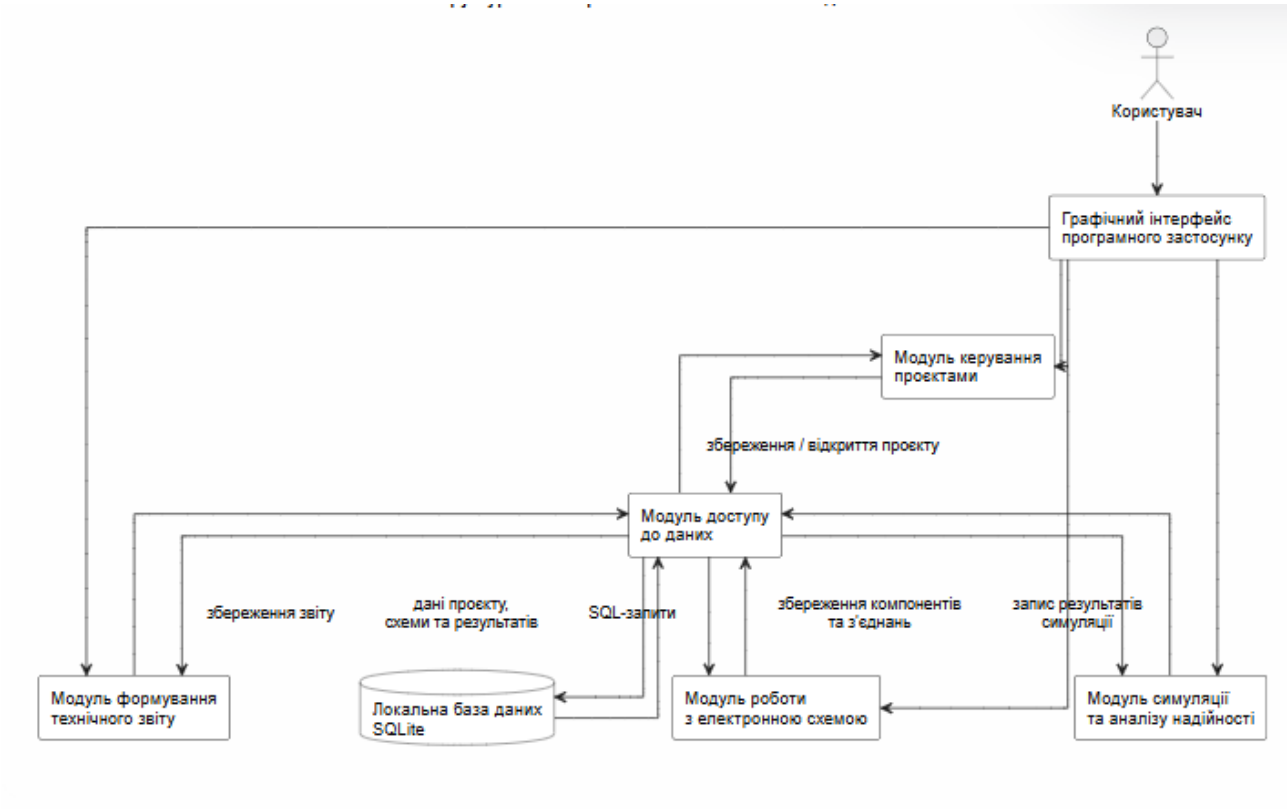


Рис. 2.2 – Структурна схема роботи з локальною базою даних.

2.3 Створення інформаційної бази данних

Фізична структура бази даних складається з таблиць projects, project_systems, project_nodes, project_edges і simulation_runs. Таблиця projects зберігає загальні відомості про проект. Таблиця project_systems зберігає системи, що належать проекту. Таблиця project_nodes відповідає за компоненти. Таблиця project_edges зберігає зв'язки між компонентами. Таблиця simulation_runs фіксує результати запуску симуляції.

Для створення таблиць використано SQL-команди CREATE TABLE IF NOT EXISTS. Це дозволяє запускати програму багато разів без повторного створення вже наявних таблиць. Якщо файл бази відсутній, він створюється автоматично. У результаті користувачу не потрібно вручну готувати базу перед запуском додатку.

Операції з базою даних винесені в окремий модуль database.py. Такий підхід робить код головного вікна простішим. Інтерфейс викликає готові функції save_project, load_project, list_projects, delete_project і save_simulation. У результаті логіка збереження не змішується з логікою кнопок, сцен і графіків.

```
def init_db():
        conn = get_connection()
        cur = conn.cursor()

        cur.execute("""
        CREATE TABLE IF NOT EXISTS projects (
                project_id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                author TEXT,
                description TEXT,
                created_at TEXT NOT NULL
        )
        """)

        cur.execute("""
        CREATE TABLE IF NOT EXISTS project_systems (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                project_id INTEGER NOT NULL,
                system_key TEXT NOT NULL,
                name TEXT NOT NULL,
                temperature INTEGER,
                humidity INTEGER,
                vibration INTEGER,
                max_time INTEGER
        )
        """)

        cur.execute("""
        CREATE TABLE IF NOT EXISTS project_nodes (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                project_id INTEGER NOT NULL,
                system_key TEXT NOT NULL,
                node_id TEXT NOT NULL,
                component_type TEXT NOT NULL,
                name TEXT NOT NULL,
                x REAL NOT NULL,
                y REAL NOT NULL,
                params_json TEXT NOT NULL
        )
        """)
```

```
    cur.execute("""
    CREATE TABLE IF NOT EXISTS project_edges (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            project_id INTEGER NOT NULL,
            system_key TEXT NOT NULL,
            edge_id TEXT NOT NULL,
            source_id TEXT NOT NULL,
            source_port INTEGER NOT NULL,
            target_id TEXT NOT NULL,
            target_port INTEGER NOT NULL
    )
    """)

    cur.execute("""
    CREATE TABLE IF NOT EXISTS simulation_runs (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            project_name TEXT,
            system_name TEXT,
            final_reliability REAL,
            component_data_json TEXT,
            report_text TEXT,
            created_at TEXT NOT NULL
    )
    """)

    conn.commit()
    conn.close()
```

Рис. 2.3 – Код створення бази даних

Фрагмент коду *рис.2.3* показує створення таблиць для проектів і компонентів. Для вузлів використовується поле params_json, у якому зберігаються параметри конкретного компонента. Це дозволяє однаково зберігати резистор, сенсор, реле, мотор або джерело живлення, хоча їхні характеристики відрізняються.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Програмний додаток побудовано за модульним принципом. Такий підхід дозволяє розділити інтерфейс, логіку симуляції, роботу з даними та математичні розрахунки на окремі частини. Це спрощує підтримку коду, зменшує залежність між модулями та дозволяє змінювати окремі елементи системи без повного переписування програми.

Основним модулем є головний файл програми, який відповідає за створення головного вікна, запуск графічного інтерфейсу, обробку дій користувача та взаємодію між іншими частинами системи. Саме через нього виконується запуск симуляції, відкриття вікон налаштувань, побудова графіків та формування технічного звіту.

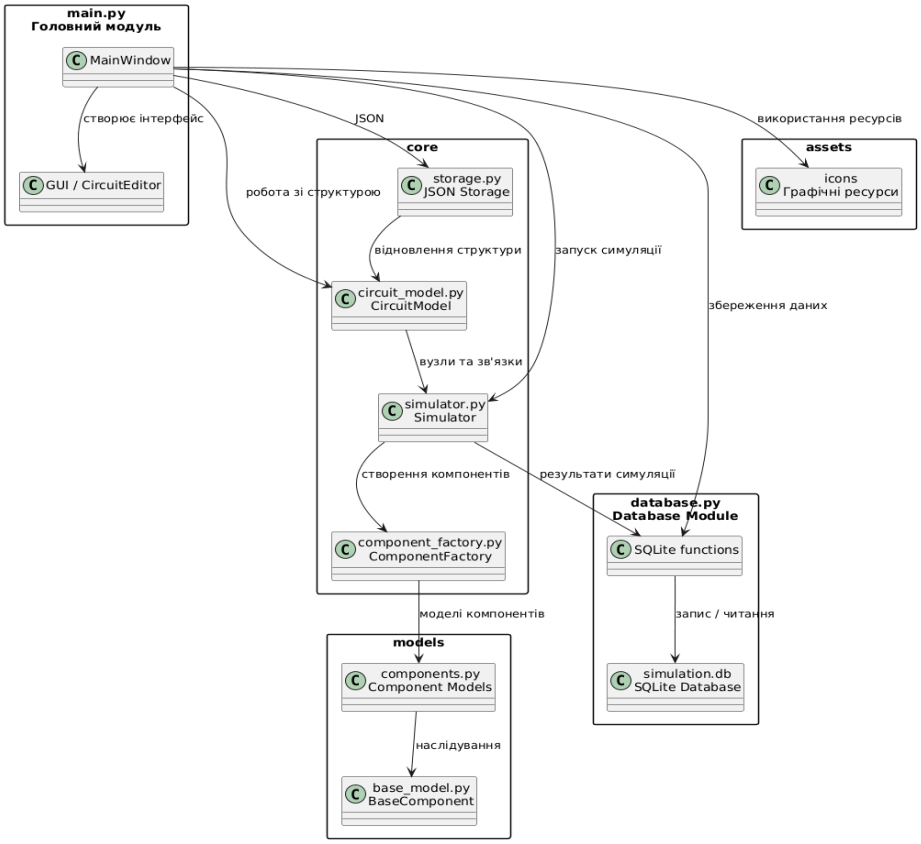


Рис. 3.1 – Діаграма пакетів.

Модуль circuit_model використовується для опису внутрішньої структури електронної системи. У ньому реалізовано вузли, зв’язки між компонентами та механізм представлення структури у вигляді взаємопов’язаних елементів. Це дозволяє виконувати аналіз шляхів передачі живлення та перевіряти зв’язність системи під час симуляції.

Розрахунки надійності виконуються в модулі simulator. У ньому реалізовано алгоритми визначення типу структури, пошуку робочих шляхів, обчислення коефіцієнтів деградації та формування кінцевих результатів симуляції. Після завершення розрахунків модуль передає результати до графічного інтерфейсу, вікна графіків і системи формування звіту.

Для створення компонентів використовується модуль component_factory. Його завдання полягає у створенні об’єкта потрібного типу відповідно до вибраного користувачем компонента. Завдяки цьому логіка створення елементів зосереджена в одному місці, що спрощує додавання нових типів компонентів у майбутньому. Їх візуалізація *рис.3.2* це графічне подання у вигляді прямокутних вузлів із короткими позначеннями. Кожен вузол можна переміщувати по сцені. Зв’язки між компонентами створюються через порти. Під час переміщення вузла всі пов’язані лінії автоматично оновлюють положення. Це робить роботу зі структурою зручною та наочною.

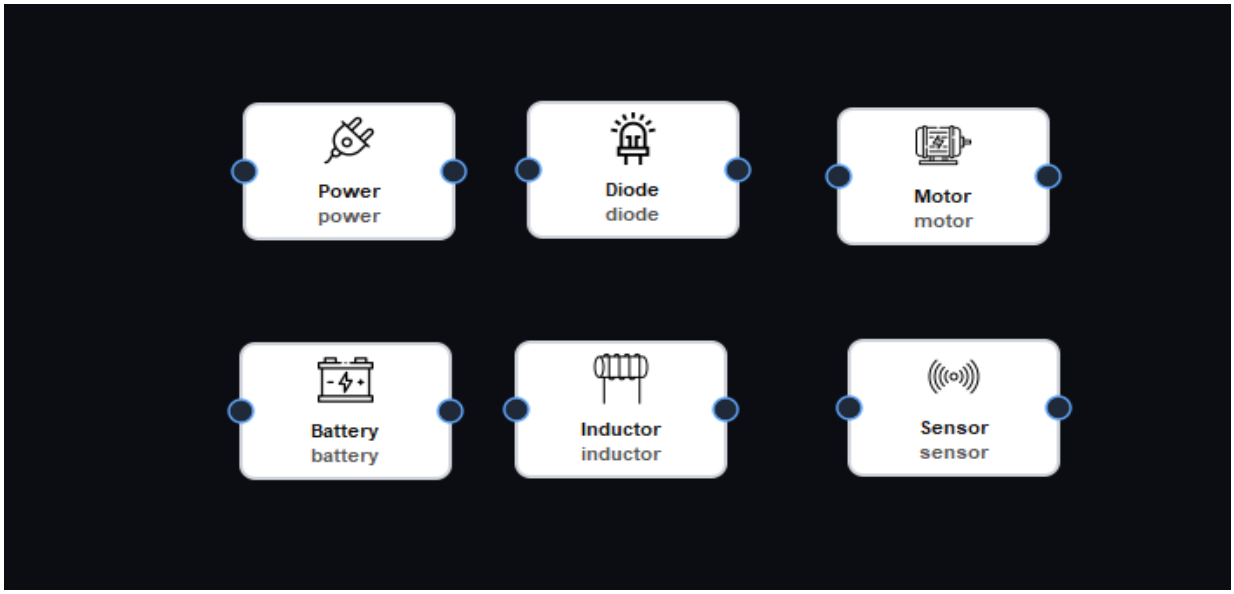


Рис. 3.2 – Компоненти системи.

Робота зі збереженням даних поділена між двома окремими модулями. Модуль storage відповідає за експорт та імпорт JSON-файлів, які використовуються для перенесення проектів між різними пристроями. Модуль database забезпечує роботу з SQLite та використовується для локального збереження проектів, параметрів компонентів і результатів симуляції.

3.2 Вибір інструментарію для створення ППЗ

Мовою програмування для реалізації програмного додатку обрано Python. Вибір цієї мови обгрунтовується швидкістю розробки, зрозумілим синтаксисом та великою кількістю доступних бібліотек для створення графічних інтерфейсів, роботи з файлами, моделювання та побудови графіків.

Matplotlib використано для показу результатів симуляції [6]. Вона застосовується для побудови графіків зміни коефіцієнта надійності в часі. У програмі підтримується відображення як загальної кривої системи, так і окремих графіків компонентів.

Для реалізації графічного інтерфейсу використано бібліотеку PyQt5 [6]. Яка забезпечує створення багатовіконного настільного додатку з підтримкою

інтерактивних елементів, діалогових форм і графічних сцен. За допомогою PyQt5 реалізовано головне вікно програми, бібліотеку компонентів, панелі параметрів симуляції та вікна технічних звітів. Особливу роль у проекті відіграють класи QGraphicsScene та QGraphicsView, оскільки саме вони використовуються для побудови редактора електронної структури.

Було використано модуль генерації uuid. Генерація UUID дозволяє уникнути повторення ідентифікаторів навіть при великій кількості компонентів у проекті. Це спрощує роботу із збереженням графової структури, оскільки кожен вузол отримує власний стабільний ідентифікатор незалежно від порядку створення або розташування на сцені.

Під час розробки також розглядалася можливість створення мобільної версії системи. Для цього використовувалася бібліотека Kivy[8], яка дозволяє створювати графічні інтерфейси мовою Python для Android-пристроїв. Kivy підтримує сенсорне керування, адаптивні елементи інтерфейсу та роботу з мобільними екранами різного розміру. Для формування Android-збірки застосовано Buildozer9, який автоматизує створення APK-файлів і виконує налаштування Android SDK, NDK та Python-залежностей. Це дозволяє запускати Python-додатки на Android без використання Java або Kotlin як основних мов розробки.

Для реалізації алгоритмів симуляції використовувалися базові принципи теорії надійності електронних систем. Для розрахунку деградації компонентів застосовано експоненційну модель зміни коефіцієнта надійності, а логіка передачі напруги між вузлами частково базується на принципах електричних кіл та законі Ома. Для аналізу структури системи та пошуку шляхів між джерелами живлення і навантаженнями використовувалися графові підходи та алгоритми обходу графів [10].

3.3 Алгоритмізація та програмування модулів

Головним алгоритмом системи є симуляція надійності *рис.3.4*. На початку зчитуються параметри поточної системи: температура, вологість, вібрація, максимальний час, режим відмов і поріг відмови. Потім для кожного моменту часу обчислюється надійність кожного компонента. Після цього визначається активність вузлів і наявність робочих шляхів від джерела до навантаження.

Для кожного компонента використано власну модель деградації[11]. Наприклад, батарея втрачає залишкову ємність, резистор змінює опір, конденсатор втрачає ємність, транзистор втрачає коефіцієнт підсилення, реле накопичує знос контактів, сенсор отримує похибку вимірювання, конвертер втрачає ефективність, мотор накопичує механічний знос. Це робить графіки більш різними та наближає симуляцію до логіки реальної експлуатації.

Напруга задається переважно в джерелах живлення. Інші компоненти отримують вхідну напругу від попередніх елементів і впливають на неї відповідно до власної логіки. Діод має порогове падіння, конвертер змінює вихідну напругу відповідно до ефективності, мотор потребує мінімальної напруги, а контролер працює лише при стабільному живленні. Це виглядає професійніше, ніж наявність окремої напруги в кожному елементі без зв’язку зі структурою.

Алгоритм класифікації структури *рис.3.3* використовує пошук простих шляхів у графі. Якщо шляхів від джерела до навантаження немає, структура вважається роз'єднаною або некоректною. Якщо шлях один, структура є послідовною. Якщо шляхів кілька, але є розгалуження, структура визначається як комбінована резервована. Якщо шляхів кілька без складного злиття, структура визначається як паралельна[12].

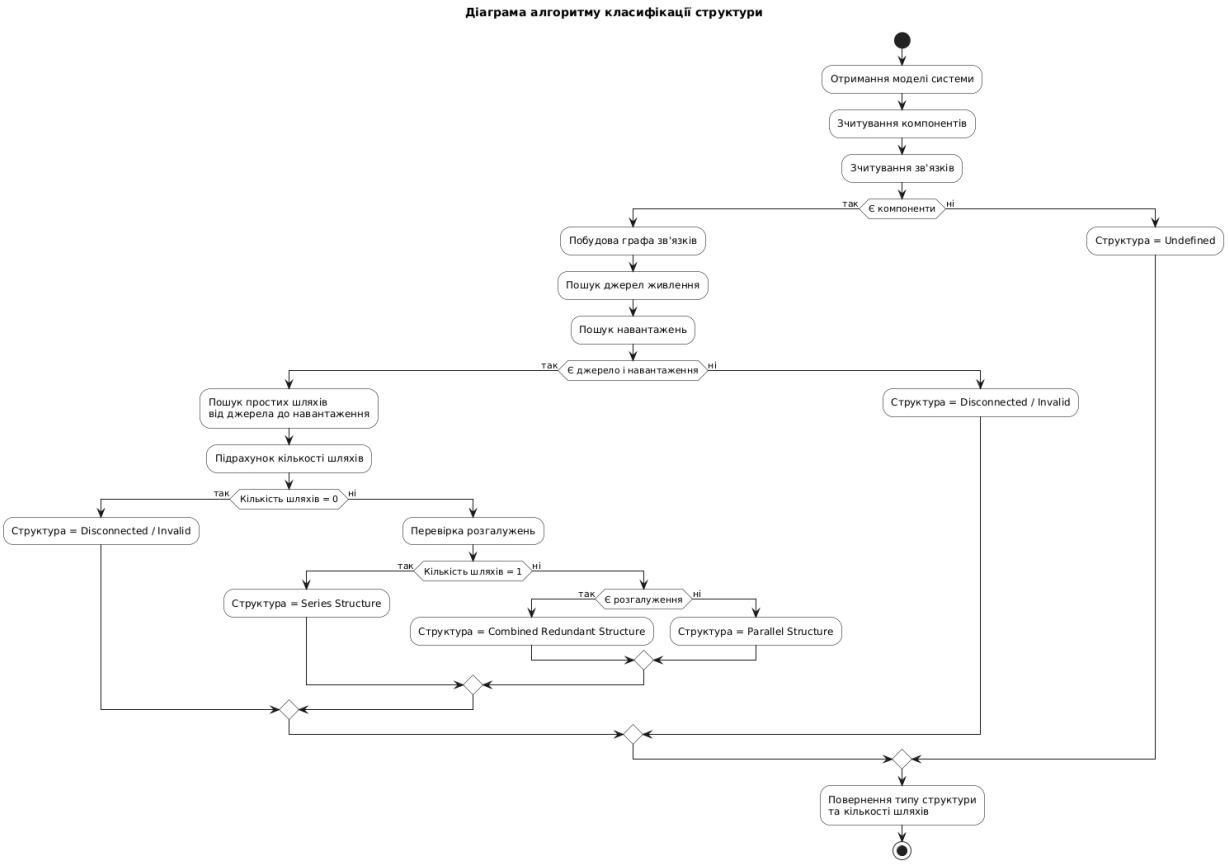


Рис. 3.3 – Діаграма алгоритму класифікації структури.

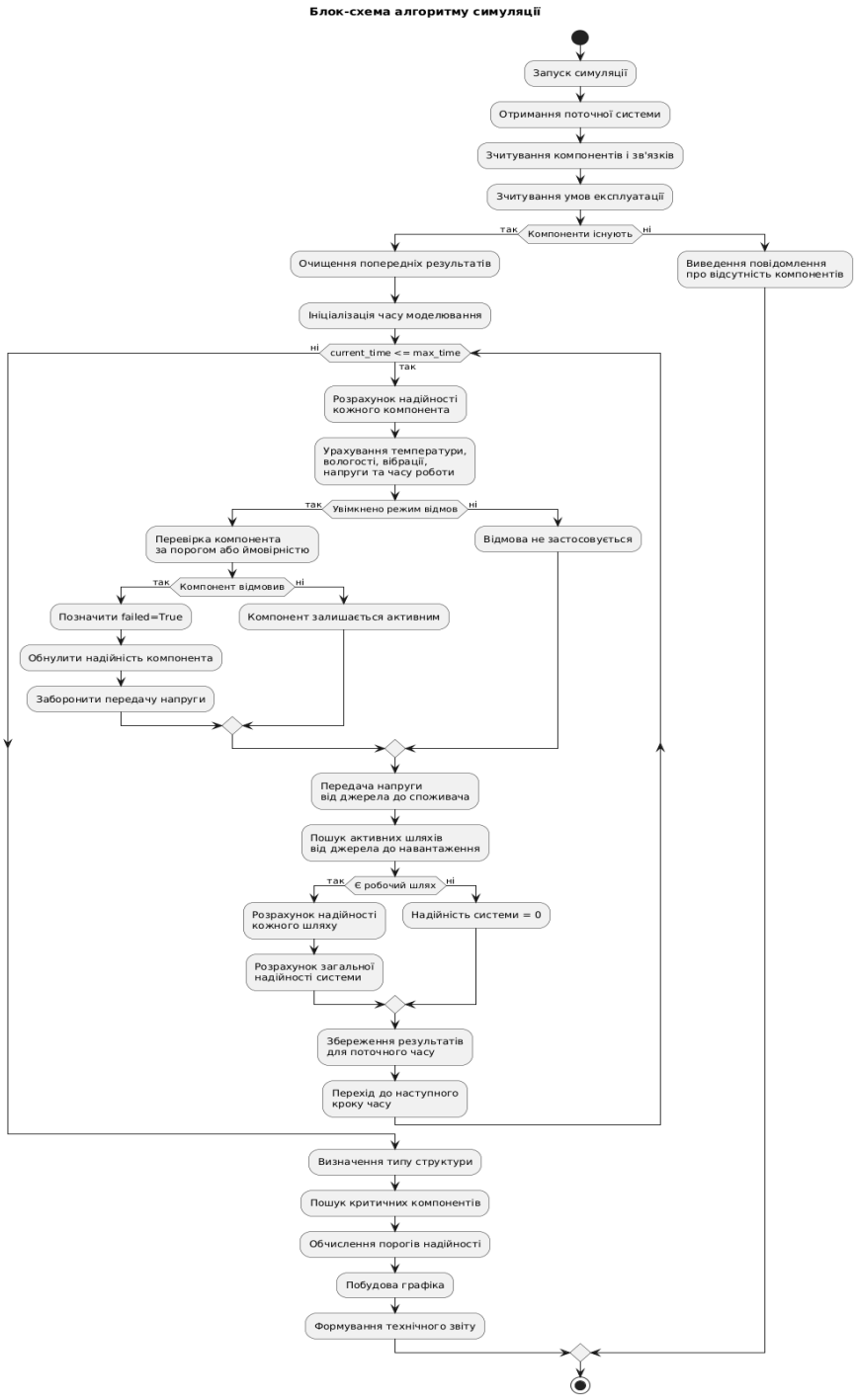


Рис. 3.4 – Блок-схема алгоритму симуляції надійності.

```
def simulate_one_system(system):
    for current_time in range(0, max_time + 1, step):
        for node in model.nodes.values():
            reliability = compute_node_reliability(node, conditions)
            node_reliabilities[node.node_id] = reliability
        states = propagate_operation_states(model, failed_nodes)
                        system_reliability, paths = evaluate_system_reliability(model,
node_reliabilities, states)
        system.times.append(current_time)
        system.results.append(system_reliability)
```

Рис. 3.5 – Частина коду симуляції системи

Фрагмент *рис. 3.5* коду показує загальну послідовність симуляції. Реальний модуль містить додаткову обробку відмов, формування історії компонентів, визначення критичних вузлів і підготовку звіту. Винесення цього алгоритму в окремий файл simulator.py дозволило уникнути дублювання коду в інтерфейсі користувача.

3.3.1 Обробка відмов у системі

Реалізовано три режими роботи системи: режим без відмов, пороговий режим і ймовірнісний режим відмов. Завдяки цьому можна порівнювати поведінку системи при різних сценаріях експлуатації та аналізувати вплив деградації компонентів на загальну працездатність структури.

У режимі без відмов компоненти не вимикаються примусово навіть при сильному зниженні коефіцієнта надійності R. У цьому випадку система демонструє поступове старіння елементів і дозволяє спостерігати зміну графіка надійності без фізичного руйнування структури. Такий режим використовується для аналізу деградації та порівняння різних типів компонентів.

Пороговий режим працює за принципом критичного значення. Якщо коефіцієнт надійності компонента стає нижчим за заданий поріг

failure_threshold, компонент вважається відмовленим. Після цього вузол додається до списку failed_nodes, а його напруга та працездатність примусово обнуляються.

Перевірка відмови реалізована таким чином *рис. 3.6:*

```
if reliability <= failure_threshold:
    failed = True
```

Рис. 3.6 – Код перевірки.

Після відмови компонент більше не може передавати напругу далі по структурі *рис. 3.7:*

```
if failed:
reliability = 0.0
output_voltage = 0.0
is_active = False
```

Рис. 3.7 – Код зміни стану.

Така реалізація дає можливість моделювати реальну втрату працездатності вузла та її вплив на всю систему. Якщо відмовлений компонент знаходиться у послідовній структурі, це може призвести до повної втрати робочого шляху між джерелом живлення та навантаженням. Окремо реалізовано ймовірнісний режим відмов. У цьому випадку відмова відбувається не миттєво за порогом, а залежить від випадкової події та поточного значення надійності компонента. Чим нижчим стає R, тим вищою є ймовірність відмови.
Для цього використовується випадкове число і після цього виконується перевірка *рис. 3.8.*

```
failure_probability = (1.0 - reliability) * 0.15
return random.random() < failure_probability
```

Рис. 3.8 – перевірка порогу можливості відмови.

Ймовірнісний режим не претендує на повну відповідність промисловим статистичним моделям, однак добре демонструє різницю між поступовою деградацією та раптовою відмовою компонента. Завдяки цьому графіки симуляції виглядають більш природно, а результати різних запусків можуть відрізнятися між собою.

Для того щоб увімкнути режим відмов потрібно обрати його в вікні режимів *рис. 3.9* та вказати поріг при якому компонент відмовить.

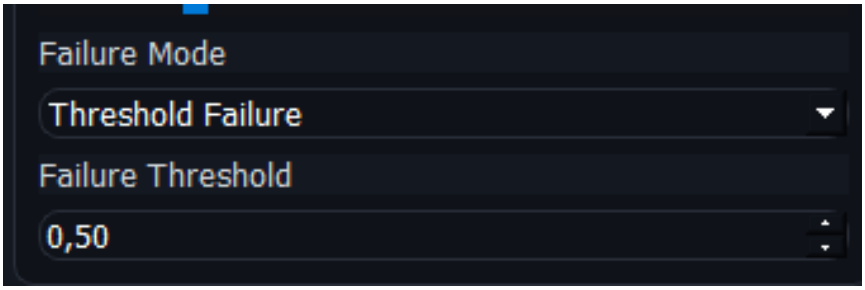


Рис. 3.9 – Панель зміни режиму відмов.

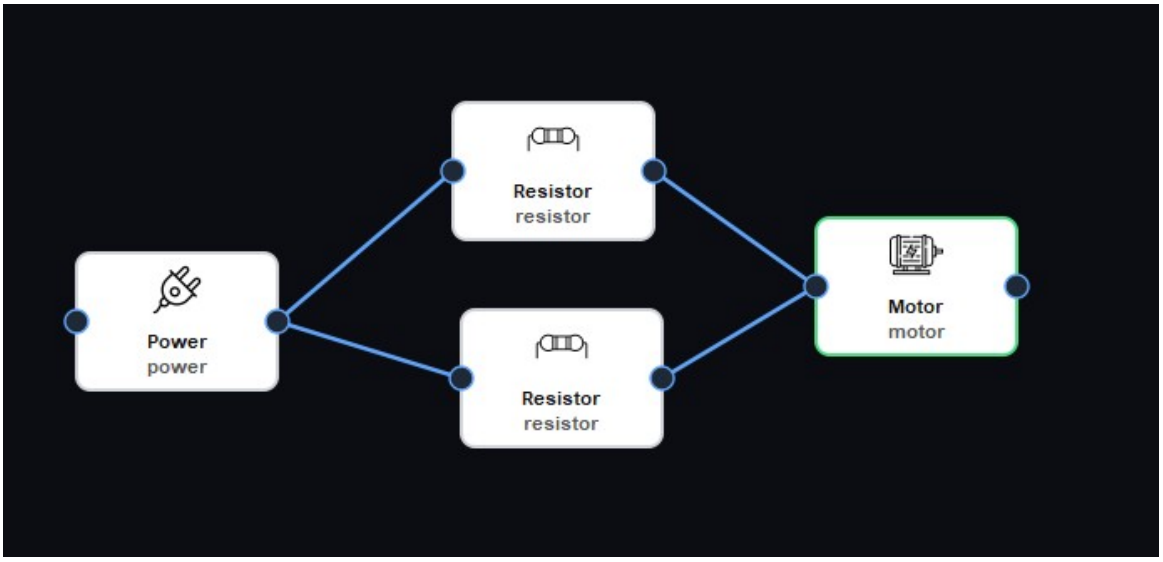


Рис. 3.10 – Система над якою проводимо симуляцію.

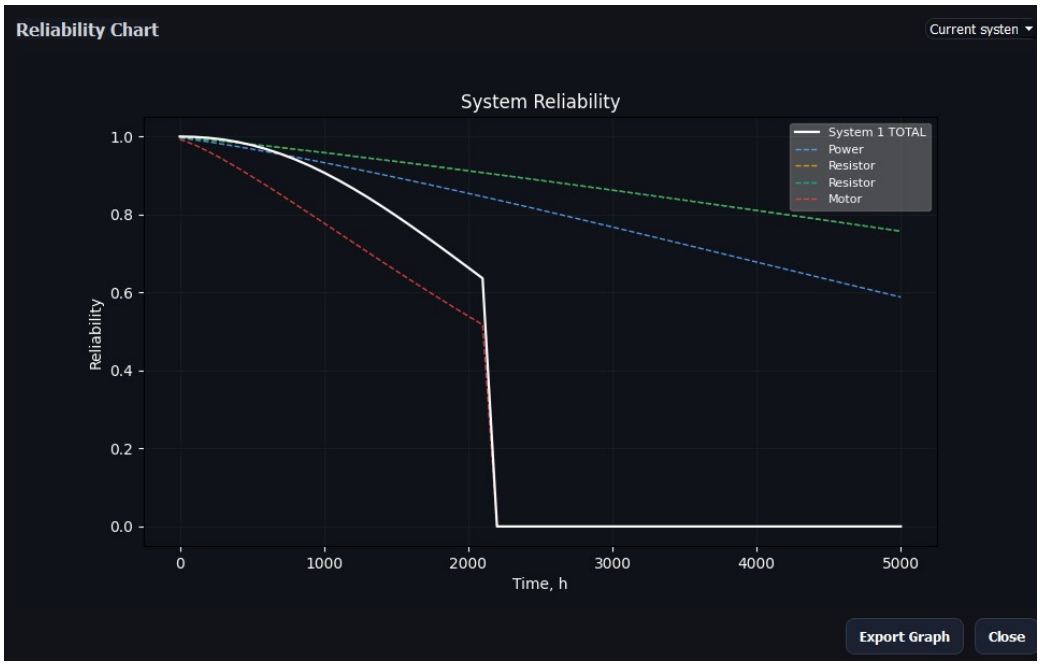


Рис. 3.11 – Графік симуляції системи з відмовами.

3.3.2 Локалізація інтерфейсу та звіту

Програмний додаток підтримує двомовний інтерфейс. Користувач може перемикатися між українською та англійською мовами без перезапуску програми. Локалізація охоплює основні елементи графічного інтерфейсу, технічний звіт, підписи кнопок, назви вкладок, повідомлення стану та діалогові вікна.

Перемикання мови реалізовано через окрему функцію оновлення інтерфейсу. Після вибору нової мови виконується зміна тексту для всіх основних елементів програми. Оновлюються підписи кнопок, груп компонентів, панелі інструментів і назви систем.

Для перекладу використано словники відповідностей *рис. 3.12*:

```
TRUCTURE_TRANSLATIONS = {
        "Series structure": "Послідовна структура",
        "Parallel structure": "Паралельна структура",
        "Combined structure with redundancy": "Комбінована структура з
```

```
резервуванням",
                "General graph structure": "Графова структура",
        "Disconnected / invalid": "Розірвана / некоректна структура",
"Undefined": "Невизначено",
        if language == "UA":
            lines.append("ТЕХНІЧНИЙ ЗВІТ ПРО НАДІЙНІСТЬ")
        else:
            lines.append("TECHNICAL RELIABILITY REPORT")


        if language == "UA":
            lines.append("ТЕХНІЧНИЙ ЗВІТ ПРО НАДІЙНІСТЬ")
        else:
            lines.append("TECHNICAL RELIABILITY REPORT")
```

Рис. 3.12 – Словник відповідностей

Технічний звіт також підтримує дві мови. Під час формування документа перевіряється поточна мова інтерфейсу, після чого вибирається відповідний набір текстових шаблонів.

Таблиця 3.1 – Частковий технічний звіт двома мовами

TECHNICAL RELIABILITY REPORT	ТЕХНІЧНИЙ ЗВІТ ПРО НАДІЙНІСТЬ
==========	==========
Project:	Проєкт:
Author:	Автор:
Description:	Опис:
System: System 1	Система: System 1
1. ENVIRONMENTAL CONDITIONS	1. УМОВИ ЕКСПЛУАТАЦІЇ
----------	----------
Temperature: 37 °C	Температура: 37 °C
Humidity: 8 %	Вологість: 8 %
Vibration: 0 %	Вібрація: 0 %
Simulation time: 5000 h	Час моделювання: 5000 год
Failure mode: Threshold Failure	Режим відмов: Без відмов
Failure threshold: 0.5	Поріг відмови: 0.5
2. SYSTEM STRUCTURE	2. СТРУКТУРА СИСТЕМИ
----------	----------

Structure type: Combined Redundant Structure Number of source-to-load paths: 2 Number of elements: 5 Number of connections: 5 3. COMPONENT LIST ---------- -- - Power | type=power | params={'voltage': 12.0, 'output_voltage': 12.0} - Transistor | type=transistor | params={'voltage': 5.0, 'gain': 100.0} - Resistor | type=resistor | params={'voltage': 5.0, 'resistance': 1000.0} - Resistor | type=resistor | params={'voltage': 5.0, 'resistance': 1000.0} - Motor | type=motor | params={'voltage': 12.0, 'power': 50.0} 4. CONNECTION LIST ---------- -- - Power:port1 -> Transistor:port0 - Transistor:port1 -> Resistor:port0 - Transistor:port1 -> Resistor:port0 - Resistor:port1 -> Motor:port0 - Resistor:port1 -> Motor:port0 5. FINAL SIMULATION RESULTS ---------- -- Final system reliability: 0.067779 System status: System failed	Тип структури: Combined Redundant Structure Кількість шляхів від джерела до споживача: 2 Кількість елементів: 5 Кількість зв'язків: 5 3. СПИСОК КОМПОНЕНТІВ ---------- -- - Power | тип=Джерело живлення | параметри={'voltage': 12.0, 'output_voltage': 12.0} - Transistor | тип=Транзистор | параметри={'voltage': 5.0, 'gain': 100.0} - Resistor | тип=Резистор | параметри={'voltage': 5.0, 'resistance': 1000.0} - Resistor | тип=Резистор | параметри={'voltage': 5.0, 'resistance': 1000.0} - Motor | тип=Двигун | параметри={'voltage': 12.0, 'power': 50.0} 4. СПИСОК ЗВ'ЯЗКІВ ---------- -- - Power:порт1 -> Transistor:порт0 - Transistor:порт1 -> Resistor:порт0 - Transistor:порт1 -> Resistor:порт0 - Resistor:порт1 -> Motor:порт0 - Resistor:порт1 -> Motor:порт0 5. ФІНАЛЬНІ РЕЗУЛЬТАТИ СИМУЛЯЦІЇ ---------- -- Фінальна надійність системи: 0.067779 Стан системи: існує робочий шлях

3.3.3 Побудова графіків результатів

Графік надійності є одним із головних способів візуального подання результатів симуляції. Під час роботи програми для кожного моменту часу обчислюється коефіцієнт надійності R, після чого результати записуються у відповідні масиви.

Для збереження часових значень використовується список:

system.times.append(current_time)

Для збереження результатів симуляції:

system.results.append(system_reliability)

Окремо зберігається історія зміни надійності кожного компонента:

system.component_history[node.node_id]["values"].append(reliability)

Для побудови графіків використовується бібліотека Matplotlib. У вікні результатів можуть відображатися як загальна крива системи, так і окремі криві компонентів. Це дозволяє визначити, який саме вузол деградує найшвидше та найбільше впливає на систему.

Щоб графік залишався читабельним при великій кількості компонентів, лінії окремих елементів зроблено тоншими, а загальну криву системи виділено більшою товщиною. Завдяки цьому основний результат симуляції добре помітний навіть у складних структурах.

Під час розробки було додано перевірки на неповні або пошкоджені дані. Якщо одна із систем не містить результатів симуляції, вона автоматично пропускається під час побудови графіка:

if not system.results: continue

Для різних типів структур графіки мають різну поведінку. У послідовних системах надійність знижується швидше через залежність від кожного компонента. У паралельних структурах деградація відбувається повільніше завдяки резервним шляхам. Комбіновані структури демонструють змішану поведінку залежно від розташування критичних вузлів.

У результаті графічне подання *рис 3.11* дозволяє швидко оцінити стан системи, визначити проблемні компоненти та порівнювати різні структури між собою.

3.4 Детальний опис роботи апарату розрахунку

Симуляцію у програмі реалізовано як спрощений математичний апарат теорії надійності електронних систем [13],. Основою розрахунків є експоненційна модель деградації компонентів, яка дозволяє оцінювати ймовірність безвідмовної роботи елемента залежно від часу експлуатації та умов середовища.

Базова модель надійності описується формулою експоненціальної моделі *рис. 3.13*:

$$R(t)=e^{-\lambda t}$$

Рис. 3.13 – формула класичної експоненціальної моделі надійності

де:

R(t) — коефіцієнт надійності компонента;

λ — інтенсивність деградації або відмов;

t — час роботи системи.

У класичній теорії надійності така модель використовується для опису поступового зниження працездатності електронних компонентів у часі. У програмному додатку базова формула була розширена додатковими коефіцієнтами навантаження, що враховують температуру, вологість, вібрацію, напругу та особливості конкретного типу компонента.

Загальний коефіцієнт деградації формується як комбінація декількох факторів впливу *рис. 3.14*:

$$k=k_0 \ *ktemp*khumidity*kvoltage*ktime$$

Рис. 3.14 – Формула обрахунку коефіцієнту деградації.

Таким чином отримано різну поведінку для різних типів елементів. Наприклад, двигун сильніше реагує на механічне навантаження та температуру, тоді як конденсатор більш чутливий до перегріву та вологості.

Апарат розрахунку побудовано так, щоб програмний додаток міг працювати з різними структурами без ручного втручання у код. Користувач створює систему на графічній сцені, а програма автоматично перетворює розміщені вузли та зв’язки на графову модель. Після цього виконується пошук джерел живлення, навантажень і можливих шляхів між ними. У результаті можна однаково обробляти як просту послідовну схему, так і складні структури з резервними гілками та кількома маршрутами передачі напруги.

Для аналізу структури використовуються алгоритми обходу графа та пошуку простих шляхів між вузлами [11]. Це дозволяє визначати тип структури, кількість активних шляхів та критичні вузли системи.

Кожен компонент у програмі реалізує два основні методи: calculate_reliability() та evaluate_operation(). Перший метод відповідає за обчислення коефіцієнта надійності R, який поступово зменшується від 1 до 0. Другий метод визначає працездатність компонента при поточній напрузі та напругу, яку вузол передає далі по структурі.

Наприклад, для двигуна враховується механічне зношення, температурне навантаження, потужність та недостатня напруга живлення *рис. 3.15*:

```
mechanical_wear = 1.0 + time * 0.00006
heat_stress = 1.0 + max(temp - 45, 0) * 0.018
power_stress = 1.0 + rated_power / 70.0
low_voltage_stress = 1.0 + max(9.0 - voltage, 0) * 0.35
```

Рис. 3.15 – Код обрахунку надійності двигуна

Отримані коефіцієнти використовуються для формування загального значення деградації компонента. Після цього обчислюється підсумковий коефіцієнт надійності *рис. 3.16*:

```
k = 0.000030 * mechanical_wear * heat_stress * power_stress * low_voltage_stress
r = math.exp(-k * time)
```

Рис. 3.16 – Код обрахунку коефіцієнту надійності.

Чим більшим стає значення k, тим швидше знижується коефіцієнт надійності компонента. Це дозволяє моделювати різні режими роботи системи та демонструвати вплив умов експлуатації на деградацію елементів.

Для різних компонентів використано різні вторинні показники. У резистора це зміна опору, у конденсатора — залишкова ємність, у реле — зношення контактів, у двигуна — механічний знос, у сенсора — похибка вимірювання. Основний показник R використовується для загального розрахунку надійності, а вторинний показник допомагає пояснити, що саме погіршується в конкретному компоненті.

Наприклад, у резисторі вторинним показником є дрейф опору рис. 3.17:

```
drift = resistance * (
    1.0 +
    0.001 * (temp - 25) +
    0.000004 * time
)
```

Рис. 3.17 – Особливі характеристики резистора.

Для конденсатора розраховується втрата ємності *рис. 3.15*:

```
loss = capacitance * (
    1.0 -
    min(0.85, 0.00014 * time * electrolyte_stress))
```

Рис. 3.18 – Особливі характеристики конденсатора.

Окремо реалізовано перевірку працездатності компонента. Наприклад, двигун вважається активним лише тоді, коли отримує не менше 9 В:

У розрахунковій частині кожен компонент розглядається як вузол із власним типом та набором параметрів. Напруга задається у джерелі живлення, а інші компоненти отримують її через з’єднання. У результаті резистор, діод, конвертер, реле, сенсор, контролер або мотор не існують ізольовано, а залежать

від того, де саме вони розміщені у структурі. Це робить модель зрозумілішою і більш логічною для пояснення в межах індивідуального проекту.

Для розрахунку надійності використовується показник R у межах від 0 до 1. Значення 1 відповідає повністю надійному стану, а значення 0 означає втрату працездатності. На практиці компонент поступово знижує R під дією часу та умов середовища. У програмі це подано через експоненційні залежності та додаткові коефіцієнти стресу *рис.3.19.* Вони не повторюють повністю промислові методики, але дозволяють показати загальний принцип старіння елементів.

Температура впливає майже на всі компоненти, але сила впливу відрізняється. Для електролітичних і напівпровідникових елементів нагрівання має більший ефект, тому конденсатор, транзистор, контролер і сенсор швидше втрачають надійність при високій температурі. Мотор і кулер також отримують додаткове навантаження, оскільки в них присутня механічна частина. Для резистора температура викликає зміну опору, що відображається у вторинній характеристиці.

```
def stress_factor(
    temp,
    humidity,
    voltage,
    nominal_voltage=5.0
):
    temp_stress = 1.0 + max(temp - 25, 0) * 0.020
    cold_stress = 1.0 + max(0 - temp, 0) * 0.010
    humidity_stress = 1.0 + max(humidity - 60, 0) * 0.015
    voltage_stress = (
        1.0 +
        abs(voltage - nominal_voltage)
```

```
    / max(nominal_voltage, 1.0)
    * 0.30
)
return (
    temp_stress *
    cold_stress *
    humidity_stress *
    voltage_stress
)
```

Рис. 3.19 – Функція стресу.

Вологість у моделі впливає на елементи, чутливі до забруднення, корозії та витоків. Сенсор, конденсатор, реле та контролер реагують на підвищену вологість сильніше, ніж резистор або діод. Такий розподіл дозволяє отримувати різні графіки навіть за однакових початкових умов. Якщо вологість підняти до високого рівня, у звіті можна побачити швидше зниження R саме для вразливих вузлів.

Вібрація враховується через окремий коефіцієнт у модулі симуляції. Для кожного типу компонента задано власну чутливість. Мотор, кулер і реле мають найбільший коефіцієнт, оскільки в реальних умовах механічні вузли та контакти сильніше залежать від коливань. Резистор і діод мають менший коефіцієнт. Завдяки цьому вібрація не впливає на всі елементи однаково, а створює більш природну поведінку графіка.

Напруга використовується не як випадковий параметр у кожному елементі, а як сигнал, що передається від джерела. Джерело живлення формує вихідну напругу, а компоненти змінюють її відповідно до своєї логіки. Діод віднімає порогове падіння, конвертер зменшує або змінює напругу відповідно до ефективності, мотор потребує мінімального рівня живлення.

Таблиця.3.4 — Стрес кожного компоненту

Компонент	Temp k	Humidity k	Vibration k	Voltage k	Time k	Особливості впливу
Power Supply	0.020	0.015	0.0012	0.30	0.00003	Перегрів і перенапруга прискорюють деградацію стабілізації живлення
Battery	0.010	0.006	0.0010	0.12	0.00004	Ємність батареї поступово зменшується під час роботи
Voltage Source	0.015	0.004	0.0008	0.35	0.00002	Чутливий до перевищення номінальної напруги
Resistor	0.014	0.004	0.0005	0.18	0.00002	Температура викликає зміну електричного опору
Capacitor	0.020	0.012	0.0011	0.030	0.00008	Висока температура та вологість прискорюють втрату ємності
Inductor	0.010	0.005	0.0014	0.010	0.00003	Вібрація впливає на обмотки та механічну стабільність
Diode	0.018	0.006	0.0006	0.025	0.00003	Перегрів збільшує втрати та ризик пробою переходу
Transistor	0.020	0.005	0.0009	0.040	0.00004	Чутливий до теплового та електричного перевантаження
Semiconductor	0.022	0.010	0.0010	0.035	0.00004	Стабільність роботи залежить від температури та живлення
Relay	0.012	0.010	0.0022	0.020	0.00005	Контакти реле деградують через механічний знос
Sensor	0.010	0.018	0.0018	0.050	0.00003	Вологість і нестабільне середовище впливають на точність
Converter	0.018	0.008	0.0013	0.020	0.00005	Перевантаження підвищує теплові втрати енергії
Fuse	0.012	0.004	0.0007	0.020	0.00002	Перевищення навантаження прискорює старіння елемента

Controller Unit	0.020	0.010	0.0015	0.080	0.00004	Нестабільне живлення викликає логічні помилки роботи
Cooler	0.014	0.006	0.0020	0.015	0.00005	Вібрація та механічний знос зменшують ефективність охолодження
Motor	0.018	0.008	0.0025	0.035	0.00006	Основними факторами деградації є перегрів і механічне навантаження

Пояснення показників:

Позначення коефіцієнтів:
Very High — понад 0.020
High — 0.010 – 0.020
Medium — 0.005 – 0.010
Low — 0.001 – 0.005
Very Low — менше 0.001
Для коефіцієнта часу (Time k) використовується окрема шкала:
High aging rate — понад 0.00005
Medium aging rate — 0.00003 – 0.00005
Low aging rate — менше 0.00003

Пошук робочих шляхів виконується після визначення станів компонентів. Якщо джерело активне, а навантаження має шлях через активні вузли, система вважається працездатною. Якщо всі шляхи заблоковані, фінальна надійність дорівнює нулю. Такий механізм дозволяє показувати різницю між просто низькою надійністю компонента і повною втратою робочого шляху.

Критичні вузли визначаються на основі двох ознак. По-перше, враховуються компоненти, які входять у всі знайдені шляхи. Якщо такий компонент відмовляє, резервування не допомагає. По-друге, враховуються компоненти з найнижчою фінальною надійністю. У звіті ці вузли подаються окремим списком, щоб користувач міг швидко побачити слабкі місця структури.

3.5 Детальний опис інтерфейсу користувача

Головне вікно програмного додатку розділене на логічні зони. Ліва частина відповідає за роботу з системами та бібліотекою компонентів *рис. 3.21*. Центральна зона є робочою областю, де користувач бачить електронну структуру у вигляді графа. Права частина містить інформацію про вибраний елемент, параметри симуляції та результати аналізу. Такий поділ не перевантажує екран і дозволяє швидко орієнтуватися в програмі.

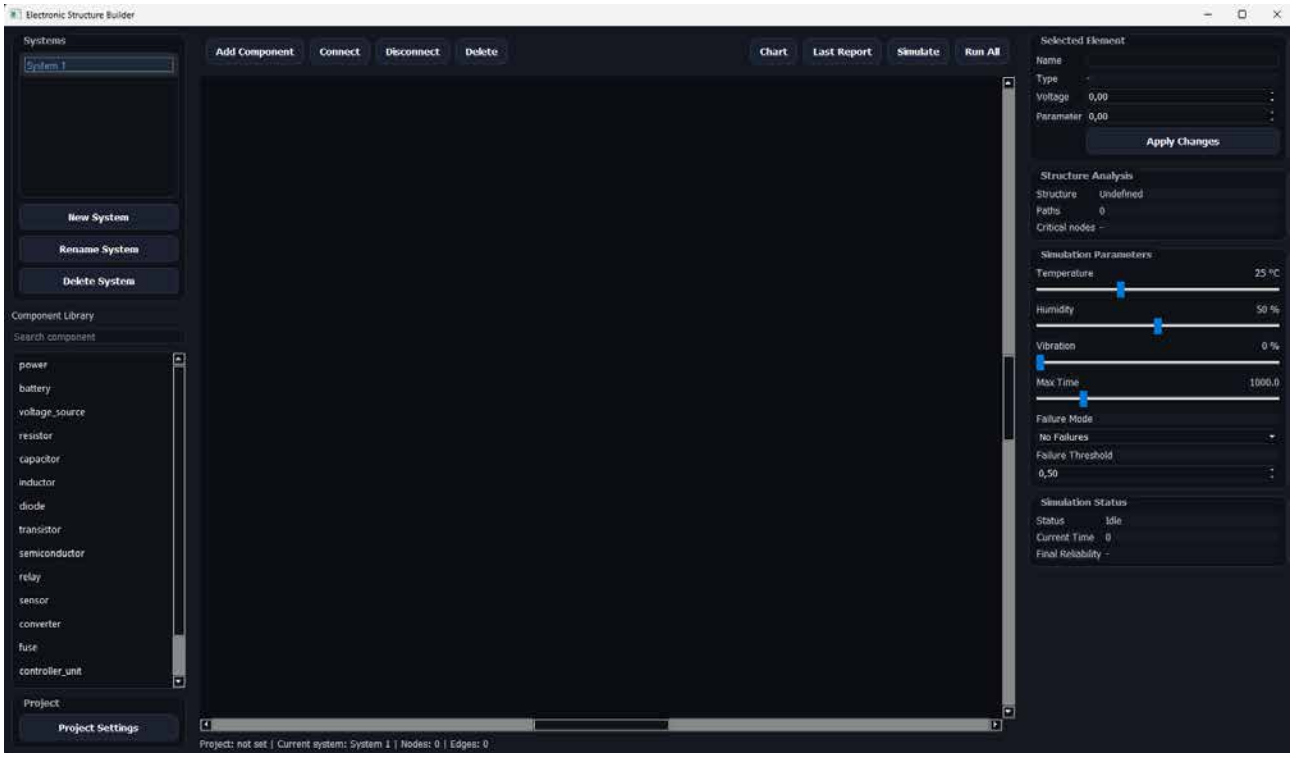


Рис. 3.20 – Головне вікно.

Бібліотека компонентів містить базові типи електронних елементів: джерело живлення, батарею, джерело напруги, резистор, конденсатор, індуктивність, діод, транзистор, напівпровідник, реле, сенсор, конвертер, запобіжник, контролер, кулер і мотор. Для швидкого пошуку передбачено текстове поле. Якщо ввести частину назви компонента, список автоматично приховує непотрібні позиції.

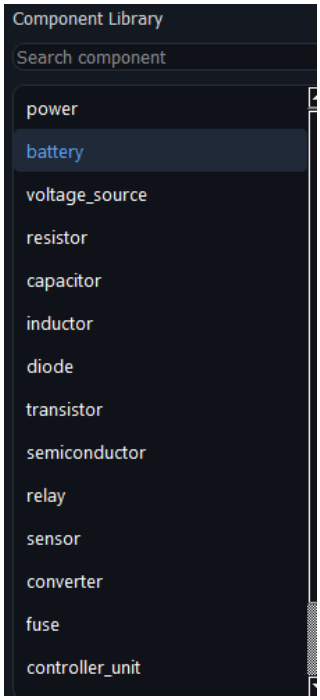


Рис. 3.21 – бібліотека компонентів.

Додавання компонента може виконуватися двома способами. Перший спосіб полягає у подвійному натисканні на елемент у бібліотеці. У такому разі компонент додається з типовими параметрами. Другий спосіб полягає у відкритті діалогового вікна додавання, де можна відразу вказати назву, тип і основні параметри. Це зручно, коли потрібне не типове значення, а конкретна конфігурація.

Редагування вибраного компонента виконується через праву панель параметрів *рис. 3.22*. Після натискання на вузол у графічній сцені відображаються його назва, тип, напруга або головний параметр. Користувач може змінити назву та числову характеристику, після чого застосувати зміни. Оновлення відразу впливає на наступний запуск симуляції.

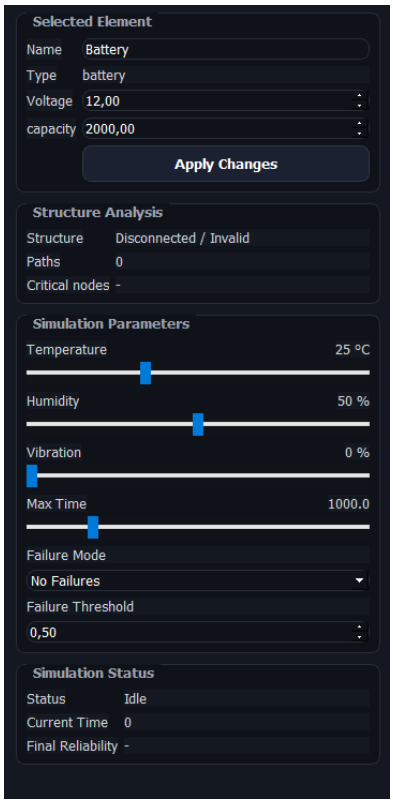


Рис. 3.22 – Права панель налаштування компонентів та симуляції.

Режим з'єднання розташований на верхній панелі керування *рис. 3.23* дозволяє створювати зв'язки між компонентами. Користувач вмикає кнопку з'єднання, вибирає порт першого вузла, а потім порт другого вузла. Після цього між ними з'являється лінія. Якщо вузол переміщується, лінія оновлюється автоматично. Це робить роботу з графом наочною[14].

Рис. 3.23 – Верхня панель керування.

Панель параметрів симуляції містить регулятори температури, вологості, вібрації та максимального часу. Окремо задається режим відмов і поріг відмови. Завдяки цьому можна швидко порівняти, як одна й та сама структура поводиться в нормальних і складних умовах. Наприклад, підвищення температури та вібрації сильніше впливає на мотор, кулер і реле.

Вікно налаштувань проекту *рис. 3.24* містить службові дії. У ньому задаються назва проекту, автор, опис і мова інтерфейсу. Також у цьому вікні розміщено кнопки для збереження та відкриття проекту з бази даних, а також для збереження й відкриття JSON-файлу. Перенесення цих кнопок у діалог налаштувань зробило головне вікно чистішим.

Вікно графіка *рис 3.11* відкривається після запуску симуляції або через кнопку перегляду графіка. Для поточної системи на графіку показується загальна надійність і надійність окремих компонентів. Це корисно для порівняння різних структур або різних умов експлуатації. Вікно звіту показує текстовий результат симуляції. Звіт можна переглянути відразу після запуску або пізніше через кнопку останнього звіту. У ньому подано параметри середовища, структуру системи, список елементів, список зв’язків, пороги надійності, критичні вузли та висновок. Це дозволяє використовувати результат не лише на екрані, а й у пояснювальній записці або додатках.

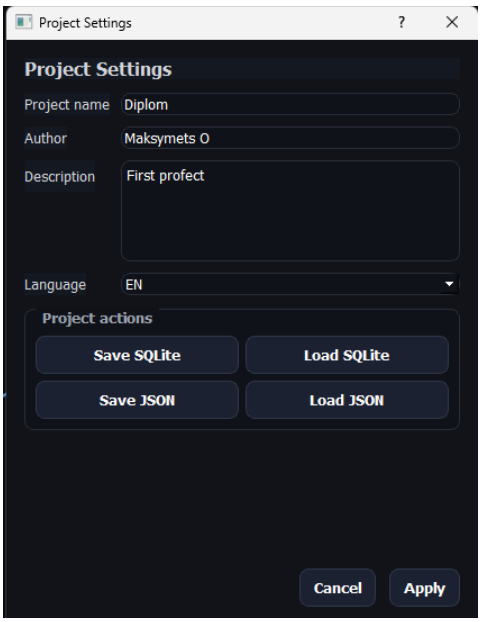


Рис. 3.24 – Вікно налаштувань проекту.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Тестування системи

Тестування програмного додатку виконувалося за основними сценаріями роботи користувача. Було перевірено створення нового проекту, додавання систем, перейменування систем, видалення систем, пошук компонентів, швидке додавання елементів із бібліотеки, ручне додавання через діалогове вікно, редагування параметрів і видалення вибраних елементів.

Окремо перевірено побудову зв’язків. Для цього створювалися послідовні, паралельні та комбіновані структури.

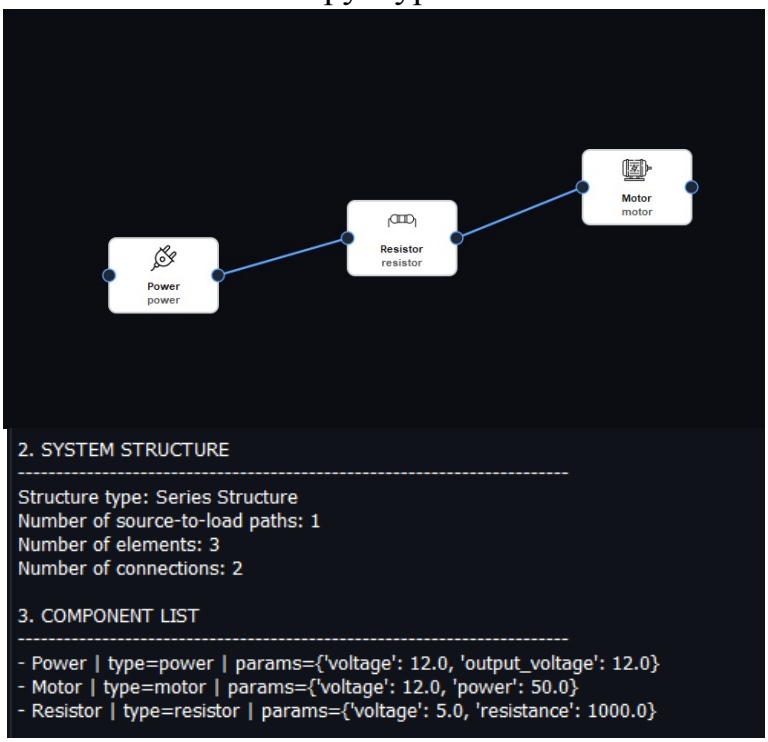
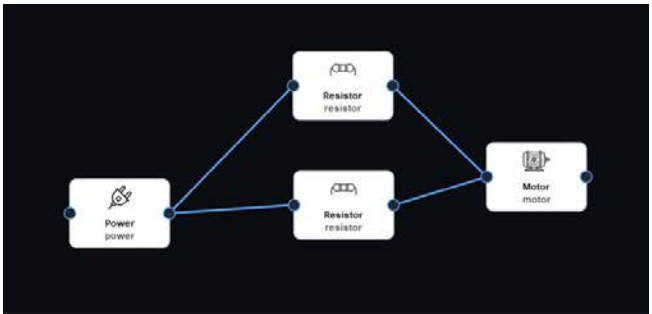


Рис. 4.1 – Приклад тестування послідовної структури.

На *рис. 4.1* наведено приклад послідовної структури. У такій конфігурації всі компоненти утворюють один робочий шлях між джерелом живлення та навантаженням. Відмова будь-якого критичного елемента призводить до втрати працездатності всієї системи.

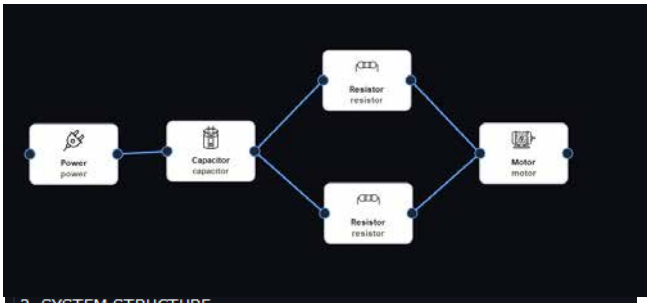



```
2. SYSTEM STRUCTURE
----------------------------------------------------------------------
Structure type: Parallel Structure
Number of source-to-load paths: 2
Number of elements: 4
Number of connections: 4

3. COMPONENT LIST
----------------------------------------------------------------------
- Power | type=power | params={'voltage': 12.0, 'output_voltage': 12.0}
- Motor | type=motor | params={'voltage': 12.0, 'power': 50.0}
- Resistor | type=resistor | params={'voltage': 5.0, 'resistance': 2000.0}
- Resistor | type=resistor | params={'voltage': 5.0, 'resistance': 3000.0}

4. CONNECTION LIST
----------------------------------------------------------------------
- Power:port1 -> Resistor:port0
- Resistor:port1 -> Motor:port0
- Resistor:port1 -> Motor:port0
- Power:port1 -> Resistor:port0
```

Рис. 4.2 – Приклад тестування паралельної структури.




```
2. SYSTEM STRUCTURE
----------------------------------------------------------------------
Structure type: Combined Redundant Structure
Number of source-to-load paths: 2
Number of elements: 5
Number of connections: 5

3. COMPONENT LIST
----------------------------------------------------------------------
- Power | type=power | params={'voltage': 12.0, 'output_voltage': 12.0}
- Motor | type=motor | params={'voltage': 12.0, 'power': 50.0}
- Resistor | type=resistor | params={'voltage': 5.0, 'resistance': 2000.0}
- Resistor | type=resistor | params={'voltage': 5.0, 'resistance': 3000.0}
- Capacitor | type=capacitor | params={'voltage': 5.0, 'capacitance': 100.0}

4. CONNECTION LIST
----------------------------------------------------------------------
- Resistor:port1 -> Motor:port0
- Resistor:port1 -> Motor:port0
- Power:port1 -> Capacitor:port0
- Capacitor:port1 -> Resistor:port0
- Capacitor:port1 -> Resistor:port0
```

Рис. 4.3 – Приклад тестування комбінованої структури.

Паралельна структура що знаходиться рис. 4.2. Система містить резервний шлях передачі живлення, тому працездатність може зберігатися навіть при деградації окремих компонентів, а рис. 4.3 нпоказує комбіновану структуру, що поєднує послідовні та паралельні з’єднання. Така конфігурація дозволяє дослідити вплив резервування та критичних вузлів на загальну надійність системи.

Для прикладу роботи застосунку створимо 2 ідентичні системи, але просимулюємо їх за різних умов та моделей відмов.

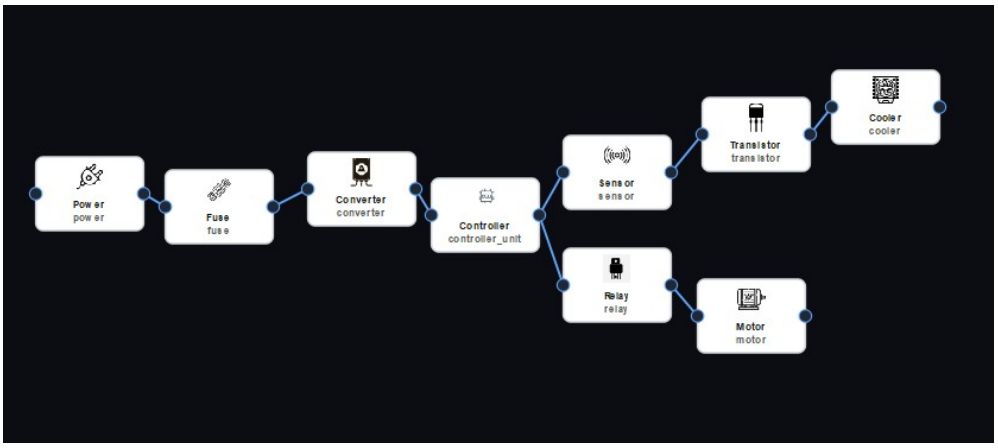


Рис. 4.4 – Система з Controller utit

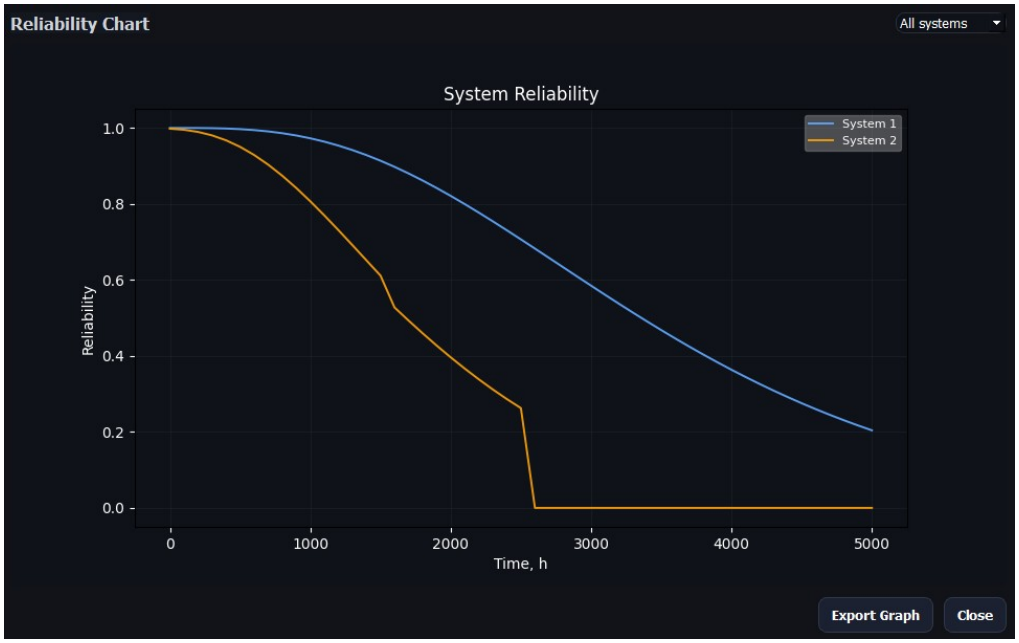


Рис. 4.5 – Графік обох Систем для порівняння.

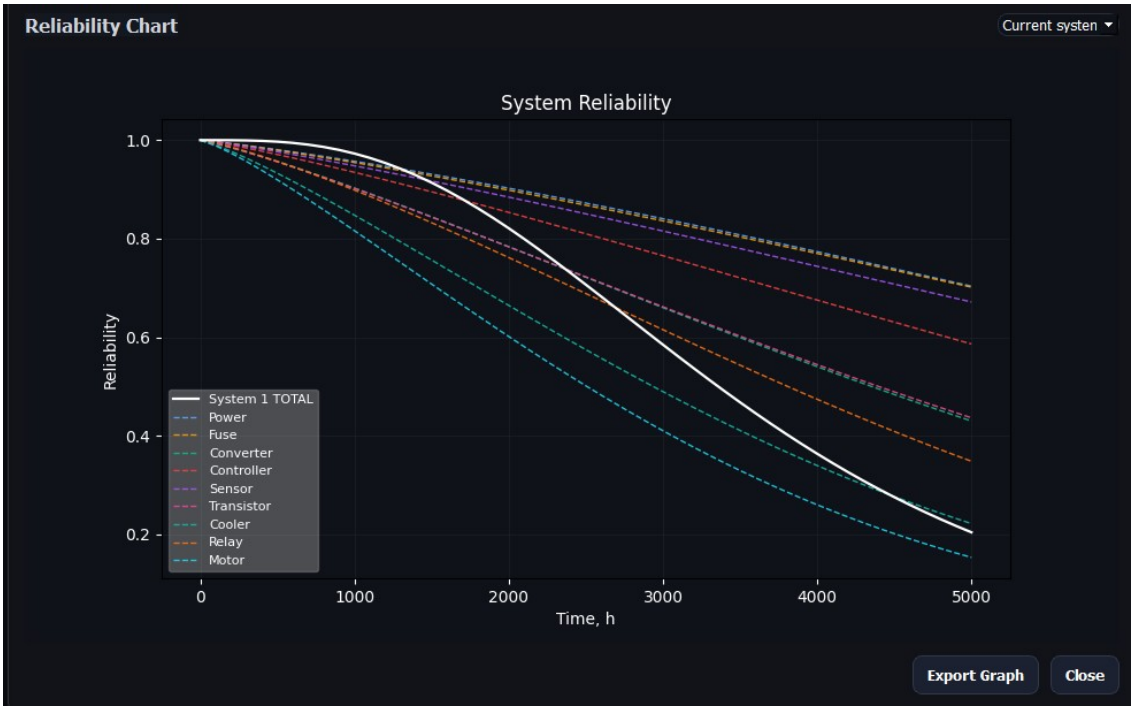


Рис. 4.6 – Графік Системи 1 без відмов.

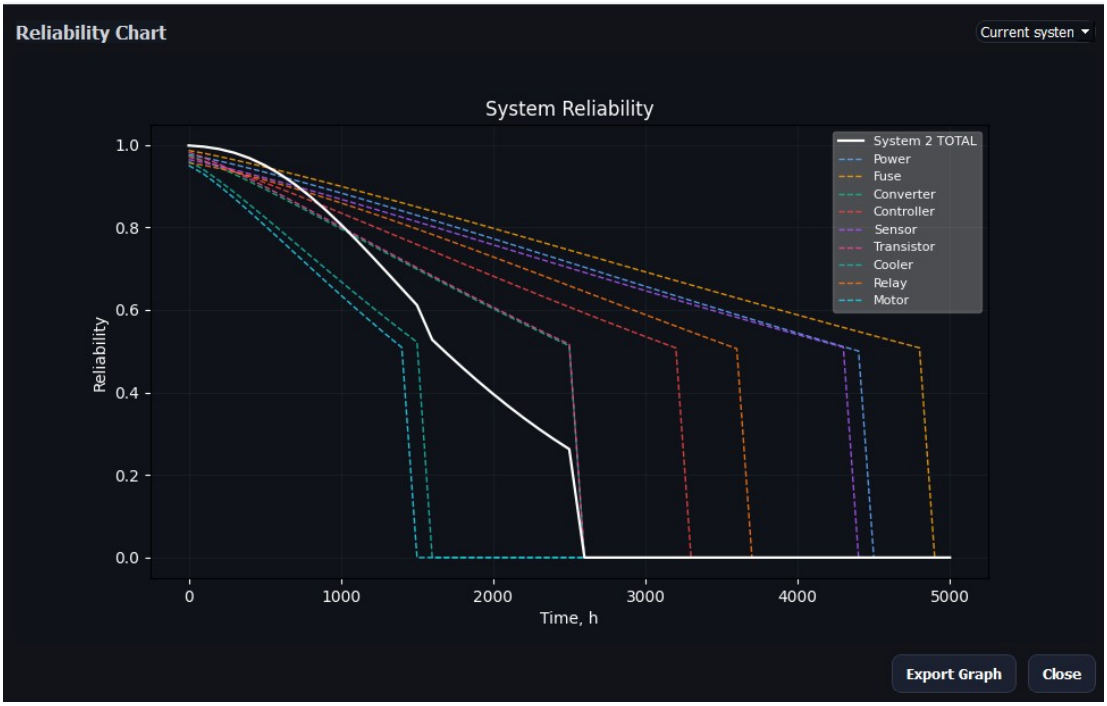


Рис. 4.7 – Графік Системи 2 з відмовами при порозі 0.5.

Таблиця 4.1 Технічний звіт систем1 та 2.

```
TECHNICAL RELIABILITY REPORT
================================================================================
==
Project:
Author:
Description:
System: System 1

1. ENVIRONMENTAL CONDITIONS
--------------------------------------------------------------------------
Temperature: 25 °C
Humidity: 7 %
Vibration: 0 %
Simulation time: 5000 h
Failure mode: No Failures
Failure threshold: 0.5

2. SYSTEM STRUCTURE
--------------------------------------------------------------------------
Structure type: Combined Redundant Structure
Number of source-to-load paths: 4
Number of elements: 9
Number of connections: 8

3. COMPONENT LIST
--------------------------------------------------------------------------
- Power | type=power | params={'voltage': 12.0, 'output_voltage': 12.0}
- Fuse | type=fuse | params={'voltage': 12.0, 'rated_current': 2.0}
- Converter | type=converter | params={'voltage': 12.0, 'efficiency': 90.0}
- Controller | type=controller_unit | params={'voltage': 5.0, 'complexity': 1.0}
- Sensor | type=sensor | params={'voltage': 5.0, 'sensitivity': 1.0}
- Transistor | type=transistor | params={'voltage': 5.0, 'gain': 100.0}
- Cooler | type=cooler | params={'voltage': 12.0, 'speed': 1200.0}
- Relay | type=relay | params={'voltage': 12.0, 'coil_voltage': 12.0}
- Motor | type=motor | params={'voltage': 12.0, 'power': 50.0}

4. CONNECTION LIST
--------------------------------------------------------------------------
- Power:port1 -> Fuse:port0
- Fuse:port1 -> Converter:port0
- Converter:port1 -> Controller:port0
- Controller:port1 -> Sensor:port0
- Sensor:port1 -> Transistor:port0
```

- Transistor:port1 -> Cooler:port0
- Controller:port1 -> Relay:port0
- Relay:port1 -> Motor:port0

5. FINAL SIMULATION RESULTS

Final system reliability: 0.204175
System status: Operational path exists

6. RELIABILITY THRESHOLDS

R <= 0.9: reached at t = 1600 h
R <= 0.8: reached at t = 2100 h
R <= 0.7: reached at t = 2600 h
R <= 0.5: reached at t = 3400 h

7. CRITICAL NODES

- Fuse
- Converter
- Motor
- Cooler
- Relay

8. COMPONENT FINAL STATES

- Power | type=power | R=0.703596 | secondary=7.200000 | active=True | failed=False | output_voltage=12.000
- Fuse | type=fuse | R=0.701620 | secondary=1.000000 | active=True | failed=False | output_voltage=12.000
- Converter | type=converter | R=0.430037 | secondary=62.000000 | active=True | failed=False | output_voltage=10.800
- Controller | type=controller_unit | R=0.586378 | secondary=2.068109 | active=True | failed=False | output_voltage=10.800
- Sensor | type=sensor | R=0.671589 | secondary=4.926162 | active=True | failed=False | output_voltage=10.800
- Transistor | type=transistor | R=0.436566 | secondary=50.000000 | active=True | failed=False | output_voltage=10.500
- Cooler | type=cooler | R=0.221893 | secondary=480.000000 | active=True | failed=False | output_voltage=9.900
- Relay | type=relay | R=0.348325 | secondary=1.000000 | active=False | failed=False | output_voltage=0.000
- Motor | type=motor | R=0.153246 | secondary=1.000000 | active=False | failed=False | output_voltage=0.000

9. CONCLUSION

The system reliability is limited. Redundancy or parameter optimization is recommended.
The combined structure contains both serially critical and parallelly redundant fragments.

==========
==

TECHNICAL RELIABILITY REPORT
==========
==
Project:
Author:
Description:
System: System 2

1. ENVIRONMENTAL CONDITIONS

Temperature: 89 °C
Humidity: 6 %
Vibration: 20 %
Simulation time: 5000 h
Failure mode: Threshold Failure
Failure threshold: 0.5

2. SYSTEM STRUCTURE

Structure type: Combined Redundant Structure
Number of source-to-load paths: 4
Number of elements: 9
Number of connections: 8

3. COMPONENT LIST

- Power | type=power | params={'voltage': 12.0, 'output_voltage': 12.0}
- Fuse | type=fuse | params={'voltage': 12.0, 'rated_current': 2.0}
- Converter | type=converter | params={'voltage': 12.0, 'efficiency': 90.0}
- Controller | type=controller_unit | params={'voltage': 5.0, 'complexity': 1.0}
- Sensor | type=sensor | params={'voltage': 5.0, 'sensitivity': 1.0}
- Transistor | type=transistor | params={'voltage': 5.0, 'gain': 100.0}
- Cooler | type=cooler | params={'voltage': 12.0, 'speed': 1200.0}
- Relay | type=relay | params={'voltage': 12.0, 'coil_voltage': 12.0}
- Motor | type=motor | params={'voltage': 12.0, 'power': 50.0}

4. CONNECTION LIST

- Power:port1 -> Fuse:port0
- Fuse:port1 -> Converter:port0

- Converter:port1 -> Controller:port0
- Controller:port1 -> Sensor:port0
- Sensor:port1 -> Transistor:port0
- Transistor:port1 -> Cooler:port0
- Controller:port1 -> Relay:port0
- Relay:port1 -> Motor:port0

5. FINAL SIMULATION RESULTS

Final system reliability: 0.000000
System status: System failed

6. RELIABILITY THRESHOLDS

R <= 0.9: reached at t = 800 h
R <= 0.8: reached at t = 1100 h
R <= 0.7: reached at t = 1300 h
R <= 0.5: reached at t = 1700 h

7. CRITICAL NODES

- Power
- Fuse
- Converter

8. COMPONENT FINAL STATES

- Power | type=power | R=0.000000 | secondary=0.000000 | active=False | failed=True | output_voltage=0.000
- Fuse | type=fuse | R=0.000000 | secondary=0.000000 | active=False | failed=True | output_voltage=0.000
- Converter | type=converter | R=0.000000 | secondary=0.000000 | active=False | failed=True | output_voltage=0.000
- Controller | type=controller_unit | R=0.000000 | secondary=0.000000 | active=False | failed=True | output_voltage=0.000
- Sensor | type=sensor | R=0.000000 | secondary=0.000000 | active=False | failed=True | output_voltage=0.000
- Transistor | type=transistor | R=0.000000 | secondary=0.000000 | active=False | failed=True | output_voltage=0.000
- Cooler | type=cooler | R=0.000000 | secondary=0.000000 | active=False | failed=True | output_voltage=0.000
- Relay | type=relay | R=0.000000 | secondary=0.000000 | active=False | failed=True | output_voltage=0.000
- Motor | type=motor | R=0.000000 | secondary=0.000000 | active=False | failed=True | output_voltage=0.000

9. CONCLUSION

The system cannot provide a valid operational path from source to load under the selected conditions.
The combined structure contains both serially critical and parallelly redundant fragments.

В Таблиця 4.1 наведено приклади технічних звітів, сформованих після завершення симуляції комбінованої електронної системи. Звіт містить інформацію про умови експлуатації, тип структури, список компонентів, електричні зв’язки, результати моделювання та фінальний стан кожного елемента.

У першому тесті система працювала при помірній температурі та низькій вологості, тому структура зберегла працездатність навіть після деградації окремих компонентів. Фінальний коефіцієнт надійності залишився вищим за нуль, що свідчить про наявність робочого шляху між джерелом живлення та навантаженням. Найбільш критичними компонентами виявилися мотор,охолодження та перетворювач.

У другому тесті температура та рівень вібрації були значно вищими, а режим симуляції використовував порогову модель відмов. У результаті всі основні компоненти поступово перейшли в стан відмови, а система втратила робочий шлях передачі живлення. Це призвело до повної втрати працездатності структури та нульового значення фінальної надійності.

Також було перевірено збереження та завантаження проектів.

Для SQLite перевірялося створення запису, відкриття проекту зі списку та видалення. Для JSON перевірялося збереження файлу, повторне відкриття та відновлення координат компонентів.

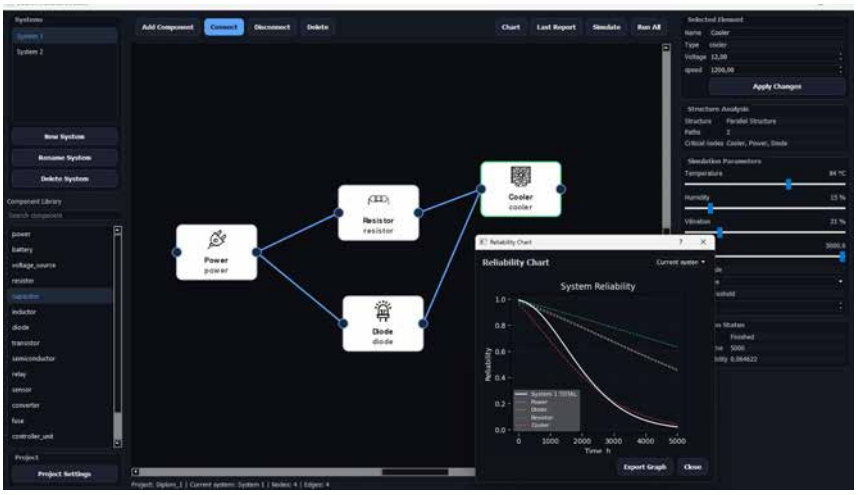


Рис. 4.8 – Головний екран з графіком.

Для тестування збереження проекту в базу данних давайте збережемо наявну систему.

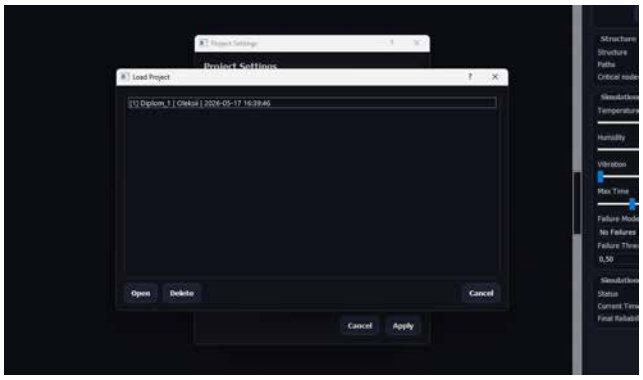


Рис. 4.9 – Список проектів

Та відкриємо його через список збережених проектів.

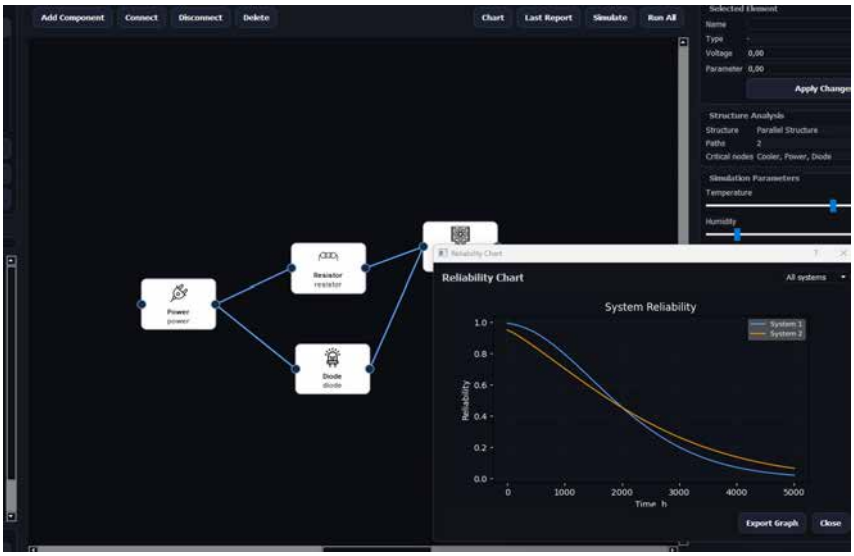


Рис. 4.10 – Завантажений проект з БД.

На рис. 4.10 видно що збереглася не лише сама система а й графік який був побудований при попередній симуляції системи.

Ці перевірки підтвердили, що проект можна не лише створити, а й продовжити роботу з ним після закриття програми.

4.2 Вимоги до апаратного та програмного забезпечення

Для запуску програмного додатку потрібен персональний комп’ютер або ноутбук з операційною системою Windows. Мінімальні вимоги включають процесор із підтримкою сучасної версії Python, 4 ГБ оперативної пам’яті та вільне місце для файлів проекту. Для комфортної роботи бажано мати екран із роздільною здатністю не нижче 1366×768.

З програмного забезпечення потрібна встановлена версія Python, бібліотеки PyQt5 та Matplotlib. SQLite використовується через стандартний модуль Python, тому окреме встановлення сервера бази даних не потрібне. Для роботи з проектом також потрібні файли модулів core, database.py, main.py та ресурси, якщо вони застосовуються.

Система є локальною, тому не потребує підключення до мережі під час роботи. Інтернет може знадобитися лише для встановлення бібліотек. База даних зберігається в локальному файлі. JSON-файли можуть зберігатися в будь-якій папці, доступній користувачу.

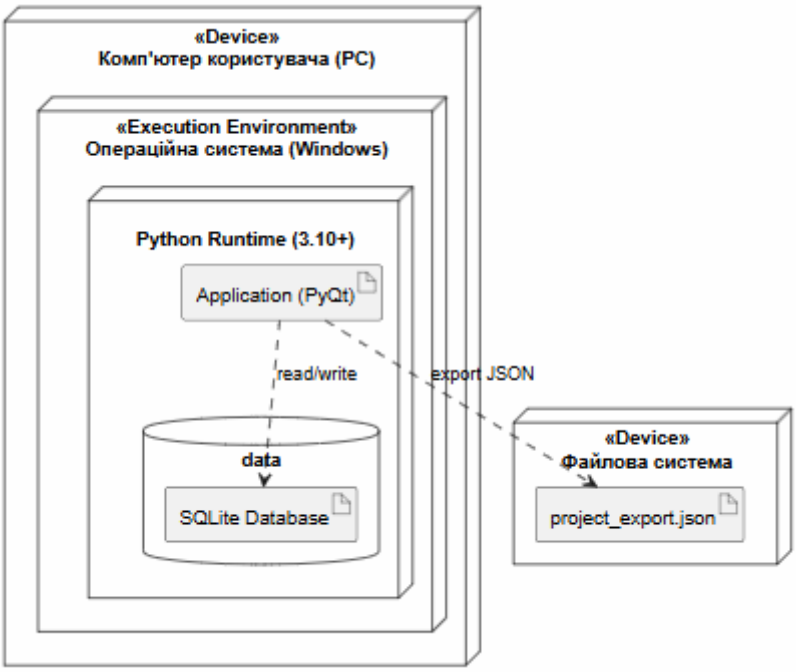


Рис. 4.7 – Діаграма розгортання системи на комп'ютері користувача.

4.3 Склад інсталяційного пакету

До складу інсталяційного пакету входять головний файл програми, модулі ядра, модуль компонентів, модуль симуляції, модуль бази даних, модуль JSON-сховища та файл бази даних, який може створюватися автоматично при першому запуску. Також можуть входити файли зображень або іконок, якщо вони використовуються для оформлення інтерфейсу.

Для запуску з вихідного коду користувач переходить у папку проекту та запускає main.py. Якщо використовується віртуальне середовище, перед запуском активується відповідна папка середовища. Після запуску

відкривається головне вікно, де можна створити проект, додати компоненти та виконати симуляцію.

У разі підготовки виконуваного файлу можна використати інструменти пакування Python-додатків. Проте в межах індивідуального проекту основним способом запуску залишено запуск через Python, оскільки це спрощує перевірку коду, демонстрацію роботи модулів і внесення змін.

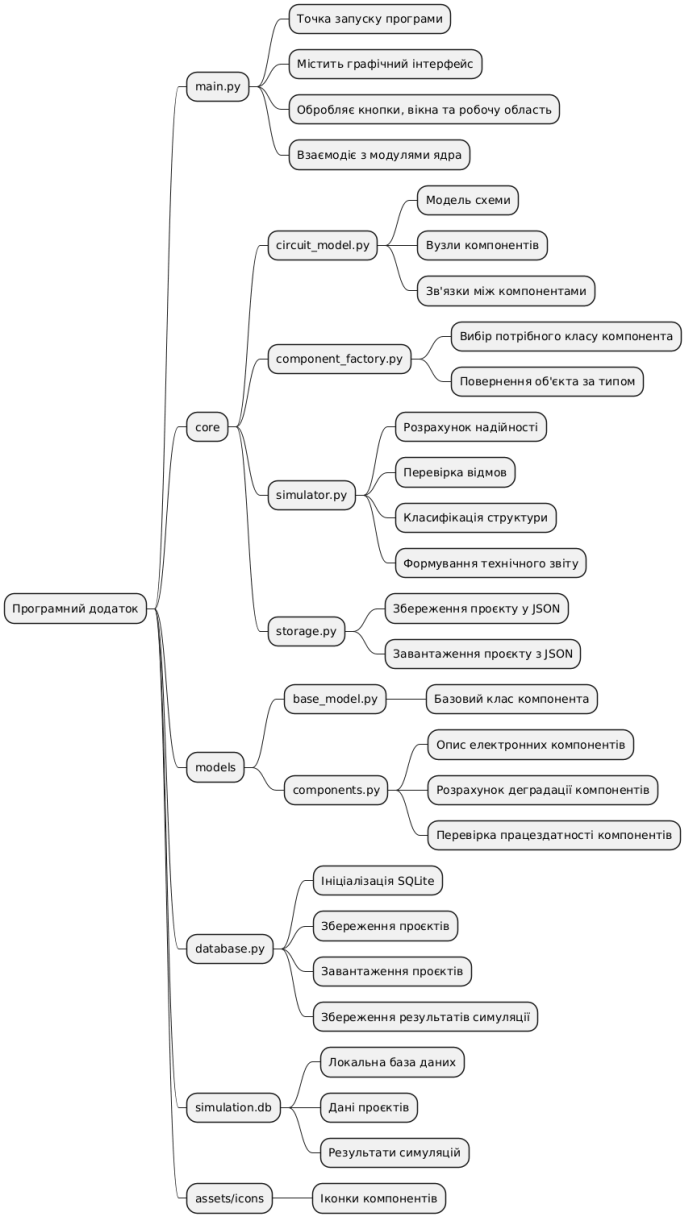


Рис. 4.8 – Склад файлів програмного продукту.

4.4 Практичні результати роботи програмного додатку

Після реалізації основних модулів було отримано працездатний програмний додаток, який дозволяє виконати повний цикл роботи з проектом. Створюється нова система, додаються компоненти, будуються зв’язки, задаються умови експлуатації, запускається симуляція, переглядається графік і формується звіт. Усі ці дії виконуються через графічний інтерфейс, тому користувачу не потрібно напряму змінювати структуру бази або JSON-файлу.

Практична перевірка показала, що одна й та сама структура може давати різні результати залежно від умов. При нормальній температурі, помірній вологості та нульовій вібрації графік знижується повільніше. Якщо підвищити температуру та вібрацію, швидше погіршуються механічні й силові компоненти. Якщо підвищити вологість, сильніше реагують сенсор, конденсатор, контролер і реле.

Окрему користь має режим порогової відмови. Він дозволяє побачити не лише поступове зниження показника R, а й момент, коли компонент вважається непридатним для роботи. Після цього шляхи, які проходять через такий компонент, перестають працювати. Це добре демонструє різницю між умовною надійністю та реальною працездатністю структури.

Ймовірнісний режим робить поведінку системи менш передбачуваною. За однакових початкових умов різні запуски можуть дати різні моменти відмови. Це відповідає ідеї, що реальні відмови не завжди настають строго в один момент. У навчальному застосуванні такий режим корисний для пояснення того, чому резервування підвищує шанси на збереження робочого шляху.

Найбільш наочним результатом стала різниця між послідовною та паралельною структурою. У послідовній структурі відмова одного компонента майже одразу впливає на всю систему. У паралельній структурі одна гілка може втратити працездатність, але інша ще забезпечує шлях до навантаження. Це добре видно на графіках і в текстовому звіті але по при це саме комбінована

структура дає найцікавіший результат, оскільки в ній одночасно є критичні та резервні ділянки. Якщо відмовляє компонент, який стоїть до розгалуження, резервні гілки не допомагають. Якщо відмовляє елемент лише в одній гілці, система може залишатися працездатною. Саме тому список критичних вузлів у звіті є важливим для аналізу.

Під час роботи над інтерфейсом було помітно, що велика кількість кнопок у лівій панелі ускладнює сприйняття. Після перенесення операцій збереження, завантаження та вибору мови у вікно налаштувань проекту головне вікно стало чистішим. Це покращило вигляд системи та зробило її більш професійною для демонстрації.

Побудова графіків також потребувала коригування. На початку загальна лінія системи перекривала лінії компонентів. Після зменшення товщини ліній, додавання різних кольорів і перевірки довжини масивів графік став стабільнішим. Додатково було додано обробку помилок, щоб у разі некоректних даних програма показувала повідомлення, а не закривалася.

Результати роботи підтверджують, що створена система підходить для демонстрації впливу експлуатаційних умов. Вона не потребує складного налаштування, працює локально, зберігає проекти, показує графіки та формує звіт. Для бакалаврської роботи це дає достатню базу для пояснення як програмної реалізації, так і прикладної частини задачі.

Подальше розширення може включати точніший розрахунок струму та потужності. Для цього до моделі потрібно додати передачу не лише напруги, а й струму, опору навантаження та обмеження потужності джерела. Така доробка зробила б систему ближчою до реальних електротехнічних процесів, однак навіть поточна версія вже демонструє основну ідею впливу умов експлуатації на електронні прилади.

4.5 Обмеження та можливості подальшого розвитку

Поточна версія програмного додатку має навчально-прикладний характер. У ній зроблено акцент на зрозумілій структурі, доступному інтерфейсі та наочній зміні надійності. Розрахунки виконуються за спрощеними залежностями, тому результати слід розглядати як попередню оцінку, а не як сертифікований інженерний висновок.

Основним напрямом подальшого розвитку є поглиблення електричної моделі. Доцільно додати розрахунок струму в гілках, сумарного споживання навантажень, обмеження максимальної потужності джерела живлення та нагрівання компонентів від власного розсіювання. Це дозволило б точніше показувати вплив резисторів, моторів, конвертерів і запобіжників.

Також перспективним є збереження історії кількох запусків симуляції. У такому разі користувач міг би порівнювати не лише різні системи, а й різні умови для однієї системи. Наприклад, один графік показував би нормальні умови, другий – підвищену температуру, третій – високу вологість і вібрацію.

Ще одним напрямом є розширення звітності. Текстовий звіт уже містить основні результати, але в майбутньому можна додати автоматичний експорт у PDF або DOCX, вставлення графіка в звіт, таблицю компонентів і короткі пояснення причин зниження надійності. Це зробило б систему зручнішою для підготовки навчальних матеріалів.

Незважаючи на обмеження, створена система має завершений набір основних функцій. Вона дозволяє побудувати структуру, зберегти її, виконати симуляцію, отримати графік і сформувати звіт. У результаті індивідуальний проект демонструє повний цикл розробки програмної системи від постановки задачі до практичної перевірки.

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи було виконано розробку інформаційної системи моделювання впливу умов експлуатації на електронні прилади. Результатом став досить простий у викормстанні програмний додаток, який дозволяє створювати електронні структури у вигляді графа, задавати параметри компонентів, запускати симуляцію та аналізувати зміну надійності в часі.

Було реалізовано введення даних через текстові поля, списки, перемикачі та числові регулятори. Передбачено пошук компонентів у бібліотеці, редагування вибраного елемента, створення зв’язків між вузлами та видалення непотрібних об’єктів. Інтерфейс побудовано так, щоб основні дії були доступні без ручного редагування файлів.

Інформаційна база реалізована на SQLite. Вона зберігає проекти, системи, компоненти, зв’язки та результати симуляції. Додатково реалізовано імпорт і експорт проектів у форматі JSON. Це дає можливість переносити проект між комп’ютерами та зберігати окремі версії структури.

Модуль симуляції винесено в окрему частину програми. Він виконує розрахунок надійності компонентів, враховує температуру, вологість, вібрацію, напругу та час, визначає активні вузли, шукає робочі шляхи та формує технічний звіт. Для компонентів використано різні моделі деградації, тому графіки відображають відмінності між елементами.

Перевагою розробленої системи є поєднання графічного конструювання структури, локальної бази даних, файлового обміну, симуляції, графіків і звітності. Обмеженням є те, що математична модель є спрощеною і не претендує на повну заміну професійних інженерних комплексів. Проте вона достатньо добре показує логіку впливу експлуатаційних факторів і може використовуватися в навчальному процесі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. MIL-HDBK-217F. Reliability Prediction of Electronic Equipment. URL:https://www.navsea.navy.mil/Portals/103/Documents/NSWC_Crane/SD-18/Test%20Methods/MILHDBK217.pdf
2. NASA Systems Engineering Handbook. NASA. URL: https://www.nasa.gov/reference/4-0-system-design-processes/
3. JSON. JavaScript Object Notation. URL: https://www.json.org/json-en.html
4. Python Documentation. Python Software Foundation. URL:https://docs.python.org/3/
5. sqlite3 — DB-API 2.0 interface for SQLite databases. Python Documentation. URL: https://docs.python.org/3/library/sqlite3.html
6. Matplotlib Documentation. URL: https://matplotlib.org/stable/users/index.html
7. Qt for Python Documentation. URL: https://doc.qt.io/qtforpython-6/
8. Kivy Documentation. URL:https://kivy.org/doc/stable/
9. Buildozer Documentation. URL: https://buildozer.readthedocs.io/en/latest/
10. Electronics Tutorials. Ohm’s Law and Power in Electrical Circuits. URL: https://www.electronics-tutorials.ws/dccircuits/dcp_2.html
11. SparkFun. Voltage, Current, Resistance, and Ohm’s Law. URL: https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law/all
12. NetworkX Documentation. Graph algorithms and paths. URL:https://networkx.org/documentation/stable/reference/algorithms/shortest_paths.html
13. Practical Weibull Analysis. 5th Edition. URL:https://asqrrd.org/wp-content/uploads/2020/12/Practical-Weibull-Analysis-Monograph-5th-Ed.pdf
14. Graph Theory and Network Analysis. GeeksforGeeks. URL: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

ДОДАТОК А

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Максимець Олексій Вікторович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

- Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Інформаційна система моделювання впливу умов експлуатації на електронні прилади» є результатом моєї особистої праці.
- Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
- Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [+] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [+] стилістичного редагування та перевірки граматики;
        - [+] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____ (підпис) **Олексій МАКСИМЕЦЬ**

ДОДАТОК Б

circuit_model.py

```
import uuid
from dataclasses import dataclass, field
from typing import Dict, Optional

@dataclass
class NodeData:
    node_id: str
    component_type: str
    name: str
    x: float
    y: float
    params: Dict[str, float] = field(default_factory=dict)


@dataclass
class EdgeData:
    edge_id: str
    source_id: str
    source_port: int
    target_id: str
    target_port: int


class CircuitModel:
    def __init__(self):
        self.nodes: Dict[str, NodeData] = {}
```

```
        self.edges: Dict[str, EdgeData] = {}

    def add_node(self, component_type: str, name: str, x: float, y: float, params:
Dict[str, float]) -> NodeData:
        node = NodeData(
            node_id=str(uuid.uuid4()),
            component_type=component_type,
            name=name,
            x=x,
            y=y,
            params=params.copy()
        )
        self.nodes[node.node_id] = node
        return node

    def update_node_position(self, node_id: str, x: float, y: float) -> None:
        if node_id in self.nodes:
            self.nodes[node_id].x = x
            self.nodes[node_id].y = y

    def remove_node(self, node_id: str) -> None:
        if node_id not in self.nodes:
            return

        edges_to_remove = [
            edge_id for edge_id, edge in self.edges.items()
            if edge.source_id == node_id or edge.target_id == node_id
        ]
```

```
            for edge_id in edges_to_remove:
                self.remove_edge(edge_id)

            del self.nodes[node_id]

        def add_edge(self, source_id: str, source_port: int, target_id: str, target_port:
int) -> Optional[EdgeData]:
            if source_id == target_id and source_port == target_port:
                return None

            if source_id not in self.nodes or target_id not in self.nodes:
                return None

            for edge in self.edges.values():
                same = (
                    edge.source_id == source_id and
                    edge.source_port == source_port and
                    edge.target_id == target_id and
                    edge.target_port == target_port
                )
                reverse = (
                    edge.source_id == target_id and
                    edge.source_port == target_port and
                    edge.target_id == source_id and
                    edge.target_port == source_port
                )
                if same or reverse:
                    return None
```

```
        edge = EdgeData(
            edge_id=str(uuid.uuid4()),
            source_id=source_id,
            source_port=source_port,
            target_id=target_id,
            target_port=target_port
        )
        self.edges[edge.edge_id] = edge
        return edge

    def remove_edge(self, edge_id: str) -> None:
        if edge_id in self.edges:
            del self.edges[edge_id]
```

simulator.py

```
def simulate_system(
        system,
        project_meta: Dict[str, str] = None,
        language: str = "EN"
):

        if project_meta is None:
                project_meta = {
                        "project_name": "Unnamed project",
                        "author": "",
                        "description": ""
                }
```

```
model = system.model

max_time = int(system.max_time)

step = max(10, max_time // 50)

system.times = []
system.results = []

system.component_history = {
        node.node_id: {
                "name": node.name,
                "values": []
        }
        for node in model.nodes.values()
}

system.last_component_results = {}
system.last_critical_nodes = []

system.last_structure_name = "Undefined"

system.last_thresholds = {
        0.9: None,
        0.8: None,
        0.7: None,
        0.5: None
```

```
        }

        system.last_path_count = 0

        final_paths: List[List[str]] = []

        failed_nodes: Set[str] = set()

        for current_time in range(0, max_time + 1, step):

            node_reliabilities: Dict[str, float] = {}

            component_snapshot: Dict[str, Dict] = {}

            for node in model.nodes.values():

                reliability, secondary_metric = compute_node_reliability(
                    node,
                    system.temperature,
                    system.humidity,
                    system.vibration,
                    current_time
                )

                failed = False

                if node.node_id in failed_nodes:
                    failed = True
```

```
                    else:

                        failed = check_component_failure(
                            system,
                            reliability
                        )

                        if failed:
                            failed_nodes.add(node.node_id)

                    if failed:
                        reliability = 0.0
                        secondary_metric = 0.0

                    node_reliabilities[node.node_id] = reliability

                    system.component_history[node.node_id]
["values"].append(
                        reliability
                    )

                    component_snapshot[node.node_id] = {
                        "name": node.name,
                        "type": node.component_type,
                        "reliability": reliability,
                        "secondary_metric": secondary_metric,
                        "failed": failed,
```

```
                        "is_active": not failed,
                        "output_voltage": 0.0 if failed else None,
                    }

                states = propagate_operation_states(
                    model,
                    failed_nodes
                )

                system_reliability, paths, working_loads =
evaluate_system_reliability(
                    model,
                    node_reliabilities,
                    states
                )

                for node_id, state in states.items():

                    component_snapshot[node_id]["is_active"] = bool(
                        state.get("is_active", False)
                    )

                    component_snapshot[node_id]["output_voltage"] = float(
                        state.get("output_voltage", 0.0)
                    )

                    component_snapshot[node_id]["voltage_drop"] = float(
                        state.get("voltage_drop", 0.0)
```

```
                )

                component_snapshot[node_id]["notes"] = state.get(
                    "notes",
                    ""
                )

        system.times.append(current_time)

        system.results.append(system_reliability)

        if current_time == max_time:
                system.last_component_results = component_snapshot
                final_paths = paths

structure_name, path_count = classify_structure(model)

system.last_structure_name = structure_name

system.last_path_count = path_count

system.last_thresholds = compute_threshold_times(
        system.times,
        system.results
)

system.last_critical_nodes = find_critical_nodes(
        model,
```

```
            final_paths,
            system.last_component_results
        )

        system.last_report_text = build_report_text(
            system,
            project_meta,
            language
        )

        return system.last_component_results
```

ДОДАТОК В

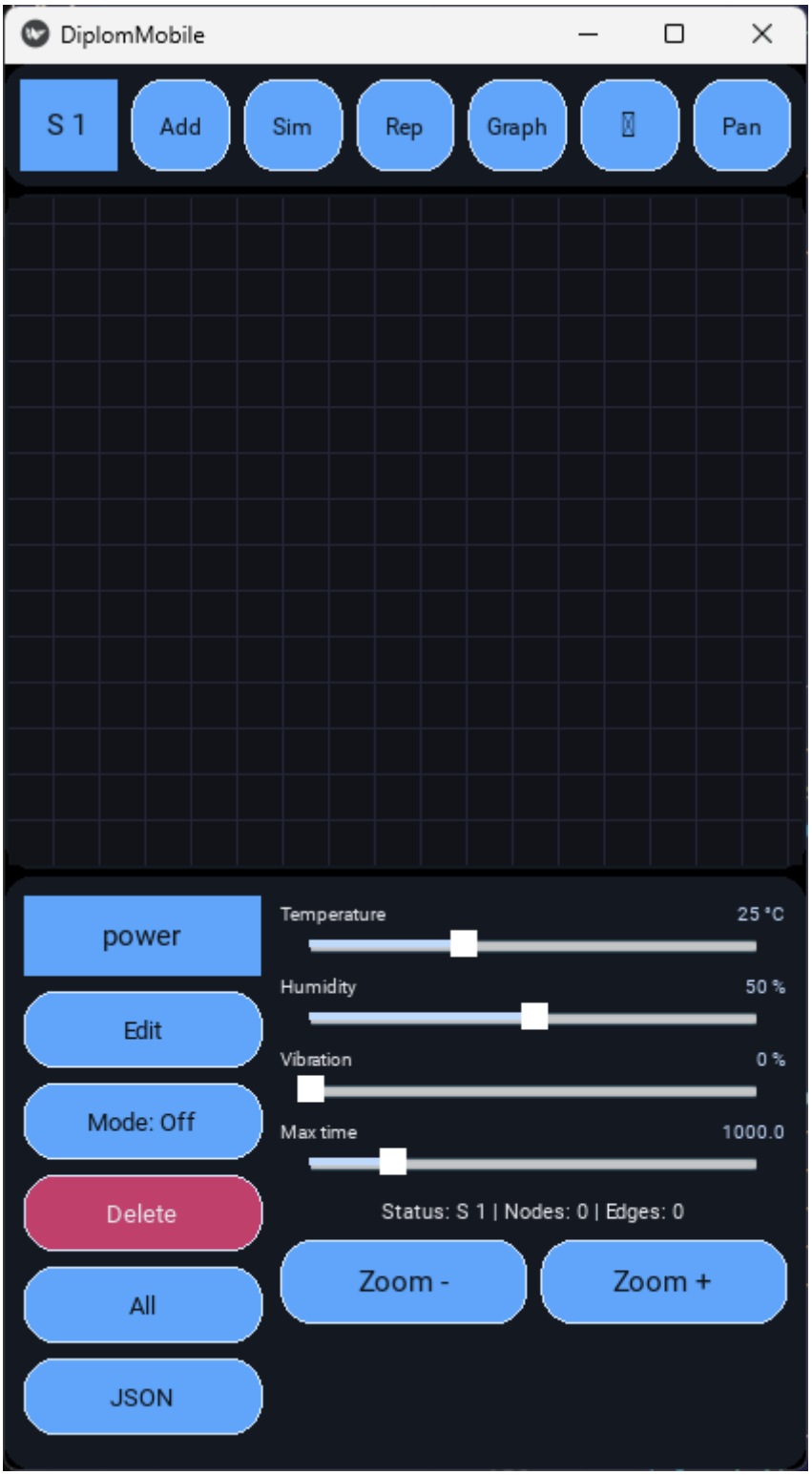


Рис. В – Інтерфейс мобільного застосунку.