Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Веб-застосунок для автоматизації проєктування логічних схем

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-3
Денис ОЛЬХОВИК
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Олег ЖУРИЛО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Ольховику Денису Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Веб-застосунок для автоматизації проєктування логічних схем

затверджена наказом по університету від "26" травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи

1) документація мови Javascript;
2) документація бібліотеки React;
3) документація Google Apps Script;
4) редактор програмного коду Visual Studio Code.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області;
2) аналіз використовуваних технологій;
3) програмна реалізація;
4) інструкція користувача;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд презентація – 10 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25-30.05.25	
2	Вибір технології розробки та інструменталь-	31.05.25-02.06.25	
	них засобів		
3	Розробка алгоритмічного забезпечення	03.06.25-05.06.25	
4	Розробка та відлагодження програмного	06.06.25-09.06.25	
	забезпечення		
5	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
6	Подання кваліфікаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Олег ЖУРИЛО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 73 с., 8 рис., 1 табл., 2 дод., 25 джерел.

АВТОМАТИЗАЦІЯ, ЛОГІЧНА СХЕМА, СИМУЛЯТОР, ЛОГІЧНИЙ ВЕНТИЛЬ, КОМПОНЕНТ, JAVASCRIPT, REACT, GOOGLE DRIVE, LOCALSTORAGE, GOOGLE APPS SCRIPT, GITHUB PAGES.

Метою кваліфікаційної роботи є створення веб-застосунку для автоматизації проєктування логічних схем. Основними можливостями створеного застосунку є побудова логічних схем у браузері, створення користувацьких логічних елементів, симуляція роботи схем у реальному часі, збереження даних на Google Drive, а також підтримка різних графічних стандартів відображення елементів.

У ході виконання кваліфікаційної роботи було проаналізовано існуючі інструменти для моделювання логічних схем, виявлено їх переваги та недоліки. На основі цього аналізу сформульовано вимоги до функціоналу, інтерфейсу та архітектури вебзастосунку, орієнтованого на навчальні потреби.

Для реалізації поставленого завдання використано JavaScript та фреймворк React, а також інструменти Vite та Google Apps Script для інтеграції з Google Drive. Хостинг реалізовано через GitHub Pages. Додатково застосовано LocalStorage для локального збереження проєктів.

ABSTRACT

Bachelor’s thesis: 73 pages, 8 figures, 1 table, 2 appendices, 25 sources.

AUTOMATION, LOGIC SCHEME, SIMULATOR, LOGIC GATE, COMPONENT, JAVASCRIPT, REACT, GOOGLE DRIVE, LOCALSTORAGE, GOOGLE APPS SCRIPT, GITHUB PAGES

The primary goal of this qualification work is to develop a web application for automating the design of logic circuits. The main features of the developed application include the construction of logic circuits in a browser, the creation of custom logic components, real-time simulation of circuit behaviour, saving data to Google Drive, and support for different graphical standards of element representation.

During the qualification work's implementation, existing logic circuit simulation tools were analyzed, and their advantages and disadvantages were identified. Based on this analysis, the requirements for the functionality, user interface, and architecture of the web application were formulated, focusing on educational needs.

JavaScript and the React framework were used to implement the task, along with Vite and Google Apps Script for integration with Google Drive. The application is hosted via GitHub Pages. Additionally, LocalStorage is used to store project data locally.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПЛІС – програмована логічна інтегральна схема

САПР – система автоматизованого проєктування

API – Application Programming Interface

BSD – Berkeley Software Distribution

CSS – Cascading Style Sheets

CSRF – Cross-Site Request Forgery

DOM – Document Object Model

ES – ECMAScript

FPGA – Field Programmable Gate Array

GAS – Google Apps Script

HTML5 – HyperText Markup Language version 5

HTTP – HyperText Transfer Protocol

IndexedDB – Indexed Database API

JSON – JavaScript Object Notation

JSX – JavaScript XML

MIT – Massachusetts Institute of Technology License

MVP – Minimum Viable Product

PostCSS – Post-processing CSS

SPA – Single Page Application

SVG – Scalable Vector Graphics

VHDL – VHSIC Hardware Description Language

XSS – Cross-Site Scripting

ВСТУП

У сучасному світі розвиток цифрових технологій значно вплинув на підходи до навчання, зокрема у сфері комп’ютерної інженерії. Освітні процеси дедалі більше переходять у віртуальне середовище, що стимулює попит на спеціалізовані вебзастосунки. Актуальності набувають інструменти, які дозволяють студентам не лише вивчати теорію, але й закріплювати знання шляхом практичного моделювання. Однією з навчальних дисциплін, у якій це актуально, є цифрова логіка, – основа побудови електронних пристроїв [1].

Існує чимало засобів для симуляції логічних схем, однак більшість мають суттєві недоліки: складний інтерфейс, обмежений функціонал, відсутність підтримки нестандартних елементів або застарілих, але все ще актуальних стандартів. Для новачків це створює бар’єр, який ускладнює засвоєння матеріалу. Особливої уваги потребують симулятори, адаптовані для використання в університетах та технічних коледжах. У таких навчальних закладах викладачі та студенти часто стикаються з потребою не лише у візуалізації схем, а й у створенні унікальних компонентів, що відображають певні логічні функції чи навчальні концепції. Доцільним є використання інструментів, що дозволяють створювати логічні схеми, перевіряти результати, виявляти помилки та зберігати проєкти у зручному форматі, зокрема із підтримкою хмарних сервісів, таких як Google Drive.

Метою роботи є розробка простого вебзастосунку для моделювання логічних схем, що дозволяє студентам створювати, редагувати, перевіряти та зберігати цифрові схеми у браузері. Особлива увага приділяється підтримці радянського та європейського стандартів зображення, можливості створення власних елементів та збереження проєктів у хмарному середовищі без потреби встановлення ПЗ. Такий застосунок сприятиме розвитку практичних навичок та підвищенню ефективності навчального процесу цифровій логіці.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис задачі

Логічні елементи є базовими блоками цифрової електроніки. Вони виконують логічні операції над бінарними сигналами та використовуються для створення складніших пристроїв – таких як арифметико-логічні блоки, пам’ять, процесори й мікроконтролери. Вивчення логічних елементів є основою підготовки фахівців з комп’ютерної інженерії, оскільки формує глибоке розуміння принципів роботи цифрових систем. Без цих знань неможливо ефективно проєктувати або аналізувати сучасні електронні пристрої, що використовуються в усіх сферах людської діяльності.

Окрім базових логічних операцій (AND, OR, NOT, XOR тощо), сучасні цифрові системи покладаються на комбінації логічних елементів для реалізації таких важливих компонентів, як мультиплексори, дешифратори, тригери, регістри та лічильники. Ці компоненти лежать в основі як простих вбудованих систем, так і складних процесорних архітектур [2].

Знання логічних елементів також необхідне при розробці схем на ПЛІС (FPGA) або при написанні коду на мовах апаратного опису, як-от Verilog чи VHDL. Це дає змогу інженеру не лише проєктувати, а й ефективно оптимізувати електронні схеми щодо швидкодії, енергоспоживання чи використаних ресурсів [3-4].

У сучасних умовах, коли дистанційне та самостійне навчання стали звичними елементами освіти, особливо зросла потреба в інструментах, які не лише дають теоретичні знання, а й дозволяють закріпити їх на практиці. Саме тому симулятори логічних схем набувають особливого значення – вони дають змогу наочно бачити роботу логічних елементів і перевіряти їхню взаємодію в реальному часі. Також вони дозволяють студентам експериментувати, створювати власні схеми, виявляти помилки та краще

розуміти зв’язок між логікою й електронікою без потреби у використанні реального лабораторного обладнання [5].

У межах вивчення цифрової логіки студенти часто отримують завдання реалізувати логічну схему за заданим функціональним описом. Наприклад, потрібно побудувати схему 3-входового мультиплексора, яка вибирає один із трьох вхідних сигналів на основі двох бітів керування. Для цього необхідно:

- скласти таблицю істинності пристрою;
- отримати логічні вирази для кожного з виходів;
- спростити їх (наприклад, методом карт Карно);
- реалізувати схему за допомогою базових логічних елементів (AND, OR, NOT) [6].

Без використання симулятора студенту доводиться вручну креслити схему, перевіряти логіку «на папері» або за допомогою таблиць істинності. У разі помилки вся робота виконується повторно. У той час як симулятор дозволяє побачити помилки миттєво та провести налагодження без переробки всієї схеми. Крім того, завдяки можливості візуального слідкування за зміною сигналів, значно полегшується розуміння принципу роботи схеми [7].

Крім освітньої мети, такі симулятори можуть бути корисними і для професіоналів, які займаються прототипуванням цифрових схем. Швидка перевірка логіки на ранньому етапі розробки дозволяє уникнути критичних помилок у подальших стадіях проєктування, що знижує витрати часу та ресурсів [8-11].

Попри наявність низки програм для моделювання логічних схем, студенти часто стикаються з труднощами при переході від теоретичного матеріалу до практичної реалізації. Багатьом важко уявити, як абстрактні логічні вирази перетворюються на функціональні цифрові пристрої. Крім того, значна частина існуючих програм є або надмірно складною в освоєнні, або недостатньо гнучкою для адаптації під конкретні освітні та навчальні потреби, а також індивідуальні цілі студентів [12-14].

Таким чином, актуальною є розробка простого, інтуїтивного інструменту для моделювання логічних схем, який поєднує візуальність, доступність і можливість розширення. Такий інструмент має сприяти глибшому розумінню цифрової логіки та забезпечувати гнучкість у створенні прикладів, експериментів і реалізації навчальних завдань різної складності.

1.2 Аналіз існуючих рішень

В рамках дослідження проведено порівняння кількох онлайн-середовищ для симуляції логічних схем. Основними критеріями для аналізу стали: інтерфейс, підтримка різних стандартів (Європейського та Радянського), створення власних елементів, функціонал збереження та зручність для наукових проєктів. Метою аналізу є оцінка переваг та недоліків кожного інструмента для обґрунтування вибору власного рішення [15].

1.2.1 Онлайн-середовище logic.ly

Інструмент має простий і приємний дизайн, що значно полегшує створення та редагування логічних схем. Важливою перевагою є підтримка двох стандартів позначень – Європейського та Радянського. Це дозволяє користувачам працювати у звичному для них форматі та легко адаптуватися до вимог різних освітніх або виробничих середовищ, (рисунок 1.1) [16]. Крім того, інтерфейс передбачає інтуїтивне розміщення елементів,.

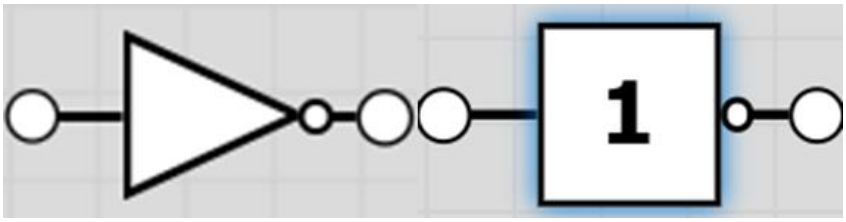

Рисунок 1.1 – Приклад використання Європейського та Радянського стандарту

Крім використання готових елементів, інструмент дозволяє створювати власні логічні вентилі з довільним зовнішнім виглядом та поведінкою (рисунок 1.2). Це корисно при моделюванні нестандартних схем або для навчальних цілей, коли потрібно акцентувати увагу на певних деталях. Функція кастомізації елементів робить симулятор гнучким інструментом для різних рівнів підготовки користувачів.

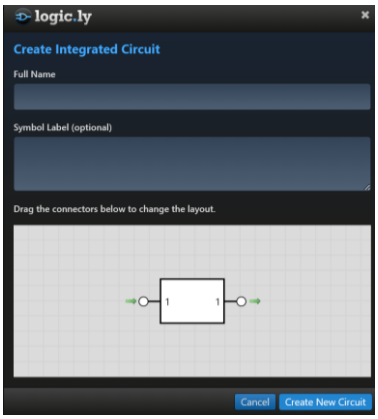

Рисунок 1.2 – Вікно створення вентилів

Однак безкоштовна версія має певні обмеження. Зокрема, при спробі збереження змін з’являється повідомлення про недоступність цієї функції (рисунок 1.3), що може бути незручним під час роботи над великими або тривалими проєктами.

Рисунок 1.3 – Повідомлення при спробі збереження в безкоштовній версії

1.2.2 Онлайн-середовище circuitverse.org

Основною перевагою інструмента circuitverse.org є велика кількість заготовлених логічних елементів, а також підтримка роботи на мобільних пристроях, що робить його зручним для використання в будь-якому середовищі. Крім того, існує можливість налаштування гарячих клавіш (рисунок 1.4), що значно пришвидшує процес розробки схем [17]. Інтерфейс інструмента інтуїтивно зрозумілий, що знижує поріг входу для новачків.

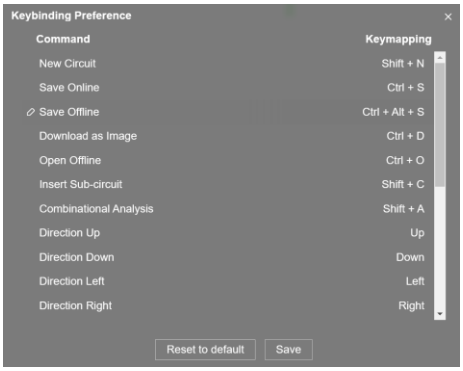

Рисунок 1.4 – Вікно налаштування гарячих клавіш

Корисною функцією є автоматичне створення Verilog-модуля на основі побудованої схеми, (рисунок 1.5). Це дозволяє створювати складніші симуляції та інтегрувати з цифровими проектами.

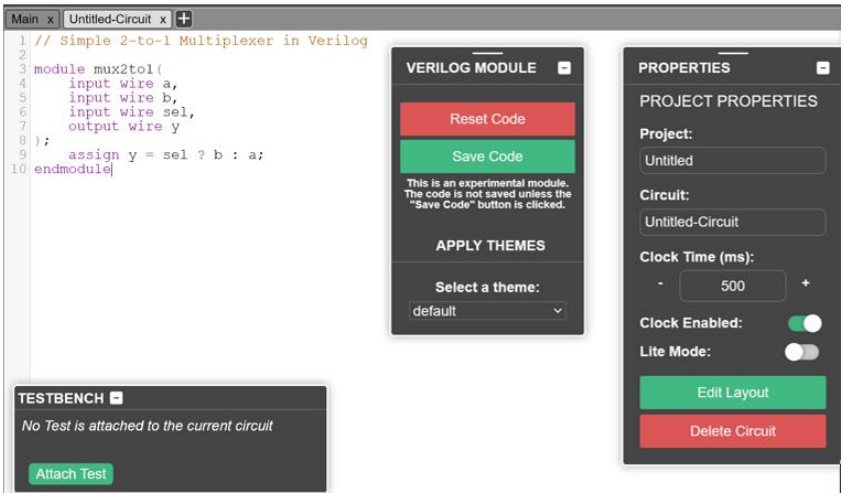

Рисунок 1.5 – Verilog код згенерований на основі побудованого модуля

Однак середовищі відсутня підтримка радянського стандарту, що може бути незручним для частини користувачів, зокрема в освітніх установах, де цей стандарт ще використовується.

1.2.3 Онлайн-середовище academo.org

Платформа academo.org має спрощену структуру та пропонує лише базовий функціонал, необхідний для побудови логічних схем. Такий підхід дозволяє зосередитися на основних елементах без зайвого ускладнення інтерфейсу. Основною перевагою є наявність форуму (рисунок 1.6), де користувачі можуть обмінюватися власними схемами, отримувати допомогу та ділитися досвідом, що підвищує навчальну цінність ресурсу [18]. Платформа підходить для базового ознайомлення з логічними елементами та принципами їх роботи.

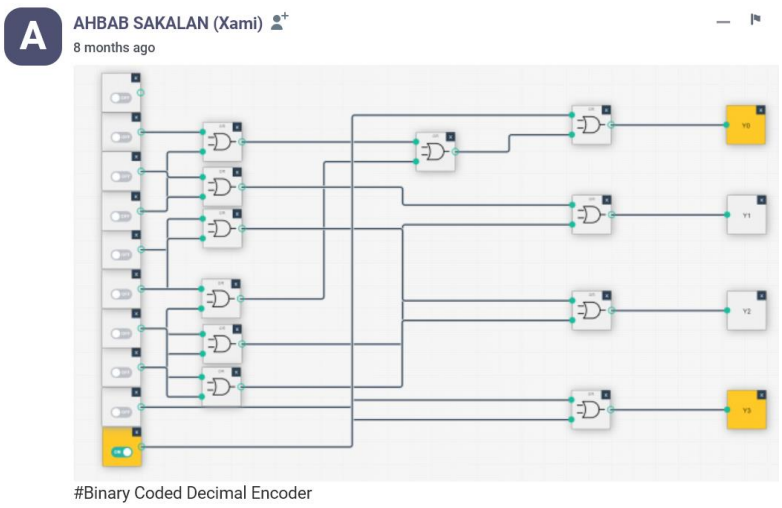

Рисунок 1.6 – Відображення логічної схеми, створеної іншим користувачем, у застосунку academo.org

Водночас платформа має і певні обмеження. Зокрема, відсутня можливість створювати власні елементи, а вибір компонентів реалізований не дуже зручно, що значно знижує ефективність роботи над більш складними або масштабними проєктами.

1.2.4 Онлайн-середовище sciencedemos.org.uk

Інструмент sciencedemos.org.uk надає можливість зберігати та завантажувати проєкти у форматі JSON (рисунок 1.7), що є зручним для тривалих або багатоетапних розробок. Такий формат дозволяє зберегти повний стан схеми та легко відновити її пізніше [19].

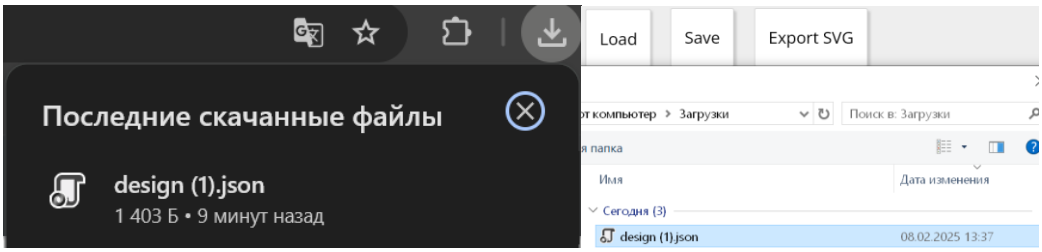

Рисунок 1.7 – Результат роботи функцій збереження та відновлення проєкту

Недоліком є відсутність підтримки створення власних елементів, що обмежує використання у складніших проєктах або в навчальних задачах, де потрібна модифікація компонентів. Інтерфейс інструмента залишається мінімалістичним, що спрощує його освоєння. Однак кількість доступних елементів обмежена, що знижує гнучкість під час роботи зі складними схемами.

1.2.5 Онлайн-середовище app.logic-gate.online

Середовище app.logic-gate.online має схожі обмеження, як і sciencedemos.org.uk: воно пропонує обмежену кількість готових елементів та не підтримує створення власних компонентів. Це зменшує його придатність для більш складних або нестандартних завдань. Проте, як і у попередньому розглянутому інструменті, тут реалізована підтримка формату JSON для збереження та відновлення проєктів, що дозволяє зручно працювати над схемами у декілька етапів [20].

1.2.6 Результати порівняльного аналізу

Після детального аналізу доступних інструментів можна зробити висновок, що більшість із них мають лише базовий набір функцій і не передбачають гнучкого налаштування візуального вигляду елементів. Багато сучасних рішень забезпечують функцію збереження проєктів у файли, що є зручним як для користувачів, так і для розробників. Оскільки зберігання даних у хмарних сервісах є поширеною практикою, доцільно реалізувати автоматичне збереження файлів користувача в Google Drive. Такий підхід забезпечує безперервність роботи навіть при втраті доступу до пристрою.

Дані, отримані в результаті порівняння інструментів для роботи з логічними схемами, надані у таблиці 1.1. Для зручності читання назви платформ у таблиці замінені на номери: logic.ly – 1, circuitverse.org – 2, academo.org – 3, sciencedemos.org.uk – 4, app.logic-gate.online – 5.

Таблиця 1.1 – Порівняльний аналіз існуючих рішень

Критерій	1	2	3	4	5
Зовнішній вигляд, схожий на ручне креслення	+			+	+
Підтримка Європейського стандарту	+	+	+	+	+
Підтримка Радянського стандарту	+				
Безкоштовне збереження прогресу на сервері з використанням серверної частини		+			
Безкоштовне збереження в хмарних сховищах					
Експорт файлів на ПК користувача		+		+	+
Імпорт файлів з ПК користувача		+		+	+
Можливість створювати власні вентилі	+	+			
Унікальний вигляд для вентилів					
Генерація таблиця істинності	+	+			
Налаштування кольорової схеми	+	+			

Продовження таблиці 1.1

Критерій	1	2	3	4	5
Наявність сітки	+	+	+	+	
Вирівнювання деталей по сітці				+	
Автоматичне з’єднання компонентів		+		+	
Навчальний інтерфейс для нових користувачів	+	+			
Можливість виконувати додаткові завдання (блок-схеми і т.д.)	+	+			
Можливість створення компонентів з використанням мов VHDL, Verilog		+			
Заготовлені деталі	23	54	22	9	10
Гарячі клавіші	+	+			
Налаштування гарячих клавіш		+			
Різні способи вводу та виводу(тестові набори, дисплеї)	+	+	+-		
Покрокове виконання	+				

Аналіз існуючих інструментів моделювання логічних схем дозволяє виокремити низку функцій, які виявилися корисними та можуть бути запозичені у процесі розробки власного вебзастосунку:

- підтримка Verilog/VHDL – можливість працювати з мовами опису апаратури значно розширює освітній потенціал симулятора;
- збереження проєктів у форматах JSON або в хмарних сховищах (Google Drive) – дозволяє зручно зберігати та обмінюватися схемами;
- гнучке створення та налаштування користувацьких вентилів, зокрема із власним візуальним оформленням – це надає користувачеві більше творчої свободи та адаптаційних можливостей;
- покрокова симуляція роботи проєкту та візуалізація сигналів, яка дозволяє не лише бачити результат, а й аналізувати процес поширення сигналів у схемі.

Водночас виявлено ряд проблем, які необхідно уникнути у новій системі:

- складність інтерфейсу: багато сучасних рішень мають перевантажені або незрозумілі інтерфейси, неадаптовані до новачків, що ускладнює навчання;
- відсутність підтримки радянських або альтернативних стандартів відображення логічних елементів, які досі застосовуються в освітніх матеріалах технічних ВНЗ;
- неможливість створення нових елементів у більшості платформ, що обмежує масштабованість і пристосованість до нестандартних задач;
- обмеження безкоштовних версій, які унеможливлюють збереження прогресу або обмежують доступ до розширеного функціоналу.

Таким чином, для досягнення максимальної ефективності вебзастосунку доцільно об’єднати найкращі практики існуючих платформ із урахуванням їх недоліків. Основний акцент має бути зроблений на простоті використання, візуальній інтуїтивності, підтримці розширення функціональності та зручності збереження результатів.

1.3 Постановка задачі

Метою даної роботи є створення сучасного вебзастосунку для моделювання логічних схем, орієнтованого на навчальні потреби у галузі комп’ютерної інженерії. Створюваний інструмент має забезпечувати можливість побудови, візуалізації, перевірки роботи та повторного використання цифрових схем із базових логічних елементів у режимі реального часу. Особлива увага приділяється доступності, простоті використання, а також можливості подальшого розширення функціоналу відповідно до потреб викладачів і студентів. Розробка вебзастосунку також передбачає інтеграцію з сучасними технологіями для забезпечення кросплатформенності. Актуальність такого інструменту обумовлена

недостатньою гнучкістю та складністю опанування існуючих рішень, що обмежує їх ефективність у навчальному процесі, особливо в умовах дистанційного або самостійного навчання.

Важливим напрямком автоматизації є здатність зберігати та повторно використовувати створені користувачем схеми або логічні елементи як окремі модулі у нових проєктах. Це дозволяє будувати складні системи на основі раніше створених компонентів, пришвидшує виконання типових завдань і сприяє покращенню якості навчання завдяки стандартизації та накопиченню власної бібліотеки цифрових елементів.

Досягнення поставленої мети передбачає виконання ряду взаємопов’язаних завдань. На першому етапі необхідно провести аналіз існуючих інструментів моделювання логічних схем, виявити їх основні переваги та недоліки, а також типові функціональні обмеження, які заважають ефективному використанню цих засобів у навчальних цілях. Подальший етап полягає у формуванні набору вимог до функціональності майбутнього застосунку, включаючи підтримку основних логічних елементів, можливість створення довільних схем, наочне відображення змін сигналів, а також простий і зрозумілий інтерфейс користувача. Наступним кроком є проєктування архітектури вебзастосунку таким чином, щоб забезпечити його масштабованість і підтримку подальших модифікацій.

Важливим аспектом роботи є реалізація інтерфейсу, що забезпечує інтуїтивну взаємодію користувача з логічними елементами, а також підтримує базові операції редагування схеми, зокрема додавання, видалення та переміщення елементів, з’єднання контактів, масштабування та скасування дій. Крім того, необхідно забезпечити можливість створення власних логічних компонентів, які будуть поєднувати логіку і візуальне представлення за прикладом готових елементів. Окрему увагу слід приділити реалізації алгоритмів поширення сигналів по схемі та їх візуалізації, що дозволить користувачам в режимі реального часу спостерігати за результатами роботи створеної конфігурації. Завершальним етапом

дослідження є тестування розробленого інструменту в умовах реального навчального процесу, збір відгуків користувачів та оцінка ефективності запропонованого підходу для формування практичних навичок з цифрової логіки.

Запропонований вебзастосунок має забезпечити низку функціональних переваг, які відрізнятимуть його від існуючих аналогів та зроблять більш придатним для використання в навчальному процесі. Однією з переваг є можливість створення власних логічних елементів із заданням їхнього візуального вигляду. Це дозволить користувачеві формувати нові компоненти схеми відповідно до специфіки завдання та забезпечить кращу наочність у процесі навчання. Інтерфейс застосунку передбачатиме інтуїтивне управління елементами схеми, що дозволить працювати з програмою без додаткового навчання.

У проєкті передбачається підтримка збереження проєктів в хмарних сховищах, зокрема Google Drive, що забезпечить гнучкість при організації навчання. Окремим завданням є реалізація підтримки графічних стандартів різних типів, зокрема європейського та радянського, що дозволить використовувати програму відповідно до вимог конкретного навчального закладу. Комплексна реалізація зазначених можливостей сприятиме ефективному використанню запропонованого рішення у процесі вивчення цифрової логіки.

2 ВИКОРИСТОВУВАНІ ТЕХНОЛОГІЇ

2.1 Обґрунтування технологічного стеку

Основною технологією клієнтської частини проєкту є JavaScript – універсальна мова програмування, яка є стандартом для веброзробки. Вона підтримується всіма сучасними браузерами, що забезпечує кросплатформеність і доступність застосунку для широкого кола користувачів. Однією з головних переваг JavaScript є розвинена екосистема – наявність великої кількості бібліотек та інструментів, які спрощують розробку та дозволяють більше уваги приділяти безпосередньо бізнес-логіці застосунку [21].

Клієнтська частина проєкту реалізована з використанням сучасного технологічного стеку, заснованого на JavaScript та фреймворку React. React дозволяє будувати динамічні інтерфейси за компонентною архітектурою – це дає змогу повторно використовувати елементи, підтримувати чистоту структури та легко масштабувати застосунок у майбутньому. Однією з переваг React є віртуальний DOM, що забезпечує високу продуктивність завдяки ефективному оновленню лише змінених частин інтерфейсу [22].

У проєкті використовується JSX – розширення синтаксису JavaScript, що дозволяє описувати компоненти у вигляді, близькому до HTML. JSX транспілюється у звичайний JavaScript за допомогою Babel – інструменту-транспілятора, який також забезпечує сумісність з різними браузерами. Завдяки Babel розробка ведеться з використанням сучасного синтаксису без урахування обмежень старих рушіїв.

Для стилізації інтерфейсу застосовано: CSS-модулі, адаптивну верстку, flexbox- та grid-сітки. Завдяки розділенню стилів за компонентами досягається більша гнучкість та зменшується ймовірність конфліктів. Основна HTML-структура базується на стандарті HTML5, що забезпечує

підтримку семантичних тегів, кращу доступність та інтеграцію з інструментами браузера.

Для зручної інтеграції React та пришвидшення процесу розробки використано інструмент Vite – сучасний білдер і дев-сервер, який забезпечує миттєве перезбирання коду, гаряче оновлення модулів у браузері та ефективну оптимізацію проєкту на етапі підготовки. Vite підтримує JSX, TypeScript, CSS та інші можливості, що робить його сумісним з React.

Для керування залежностями використано npm – стандартний менеджер пакетів для JavaScript-проєктів. За допомогою npm підключено React, Vite, інші бібліотеки, а також налаштовані npm-скрипти для локальної розробки та збірки застосунку.

Як середовище розробки використовується Visual Studio Code – легкий редактор коду з підтримкою розширень для React, інтеграцією з Git, форматуванням коду, дебагінгом та вбудованим терміналом. Visual Studio Code підходить для роботи з великими проєктами та зручний у командній розробці.

Для реалізації збереження даних, створених користувачем, використана технологія Google Apps Script, яка дозволила зберігати логічні вентилі, створені користувачами, у хмарному середовищі Google Drive, що робить їх доступними з будь-якого пристрою та не вимагає створення власного бекенду чи окремого сервера. Google Apps Script інтегрується з сервісами Google, що спрощує збереження та обробку даних.

Для асинхронної взаємодії з Google Apps Script застосовувався вбудований браузерний інтерфейс Fetch API. Його використання дозволяє здійснювати HTTP-запити без сторонніх бібліотек, підтримує проміси і дозволяє легко обробляти як текстові, так і JSON-відповіді. Його синтаксис є лаконічним, а інтеграція з async/await – зручною для послідовної обробки даних.

Для тимчасового збереження стану користувацької сесії та незавершених схем між оновленнями сторінки або тимчасовим виходом з

браузера використано LocalStorage. Це вбудований механізм браузера, який дозволяє зберігати дані у форматі ключ-значення безпосередньо на пристрої користувача. LocalStorage працює синхронно, має простий API і не потребує налаштування з боку сервера, що обумовлює його доцільність для збереження даних невеликого обсягу.

У проєкті використовують формат векторної графіки SVG (масштабована векторна графіка), який застосовується для візуалізації з'єднань між елементами схеми. Завдяки SVG забезпечується чіткість і адаптивність незалежно від розміру чи масштабування сторінки.

Процес розробки вебзастосунка передбачає використання інструментів налагодження, що вбудовані в браузери, зокрема DevTools. За допомогою вкладки Console відстежувались значення змінних, робота функцій та логіка зміни станів. Для цього застосовувались виклики console.log(), які допомагали відслідковувати появу помилок або перевіряти, чи правильно працює логіка підключення елементів, відображення сигналів тощо. Також за допомогою вкладок Elements та Application перевірялась структура HTML, правильність CSS-стилів, робота запитів до Google Apps Script та збереження до LocalStorage. Крім того, проєкт тестувався у кількох браузерах (Google Chrome, Mozilla Firefox), щоб переконатися у стабільній роботі та коректному відображенні інтерфейсу на різних рушіях.

Для керування версіями коду та хостингу використовувався Git та GitHub. Репозиторій з проєктом зберігався на GitHub, а фронтенд-частина розгорнута на GitHub Pages – сервісі безкоштовного хостингу статичних сайтів. Це дозволяє швидко публікувати застосунок і тестувати його на різних пристроях без складної серверної інфраструктури.

2.2 Фреймворки та інструменти

React – це інструмент, створений для побудови інтерфейсів користувача. Кожен компонент є незалежним блоком з власним станом і

логікою. Компоненти можна повторно використовувати в різних частинах застосунку, що значно спрощує розробку великих і складних систем. Окрім цього, React використовує віртуальний DOM – технологію, яка дозволяє ефективно оновлювати інтерфейс користувача, змінюючи лише ті частини DOM, які були змінені, без повного перерендеру сторінки. Це підвищує продуктивність навіть у великих проєктах з великою кількістю динамічного контенту. Завдяки цим властивостям React став стандартом у сучасній фронтенд-розробці [22].

Однією з переваг React є JSX – спеціальний синтаксис, що дозволяє описувати структуру інтерфейсу у вигляді, близькому до HTML. JSX дозволяє писати зрозумілий розмітковий код, який потім транспілюється у звичайний JavaScript. Наприклад, замість виклику React.createElement('h1', null, 'Привіт'), можна написати <h1>Привіт</h1>. Це суттєво спрощує структуру компонентів і робить код схожим на шаблони з підтримкою використання JavaScript. У JSX можна вставляти змінні, писати логіку умов, використовувати вирази, додавати обробники подій тощо. Перед виконанням JSX обов’язково компілюється, найчастіше за допомогою Babel або таких білдерів, як Vite, які це роблять автоматично. Завдяки цьому розробник отримує зрозумілий, декларативний спосіб опису інтерфейсів, де в одному файлі можна одночасно описати і логіку, і візуальну частину компонента. JSX також має деякі обмеження – наприклад, усі елементи мають бути вкладені у єдиний кореневий елемент, а атрибути записуються в camelCase-нотації (наприклад, className замість class, onClick замість onclick) [22].

Vite – це інструмент для збирання фронтенд-застосунків, розроблений з урахуванням потреб сучасної веброзробки. Основна перевага Vite полягає в його високій швидкості запуску й оновлення під час розробки. Це стало можливим завдяки використанню ES-модулів і сучасного JavaScript-движка в браузері. На відміну від класичних білдерів, таких як Webpack, які спочатку компілюють увесь проєкт, Vite працює за принципом on-demand – він обробляє лише ті модулі, які потрібні на поточний момент. Це дає змогу

почати роботу майже миттєво, а також суттєво пришвидшує гаряче перезавантаження, що важливо для зручної розробки. Крім того, Vite підтримує TypeScript, JSX, CSS-модулі, PostCSS та інші сучасні технології одразу, без додаткового налаштування. У режимі продакшн Vite використовує Rollup для збірки, що забезпечує високу продуктивність і оптимізацію фінального коду. У результаті це дозволяє зменшити розмір бандлів і підвищити швидкість завантаження застосунку для кінцевого користувача.

Під час розробки проєкту застосована хмарна платформа Google Apps Script для реалізації збереження даних користувача без необхідності створення окремого серверного середовища. Створено систему, яка організовує взаємодію між клієнтською частиною та віддаленим сховищем Google Drive. Цей підхід дозволив забезпечити збереження логічних вентилів, зображень та інших даних у вигляді текстових та графічних файлів. Процес роботи з Google Apps Script включав створення Web App, що дозволяє приймати як GET-, так і POST-запити. На практиці реалізована передача структурованих JSON-даних та зображень у форматі base64. Зчитування даних реалізовувалося через звернення до API Apps Script, яке формувало відповідь у вигляді масиву об’єктів, що містили метадані та вміст кожного файлу. Однією з особливостей роботи була необхідність правильно обробляти помилки при запитах, зокрема враховуючи можливу відсутність папки або некоректний формат відповіді [23].

2.3 Хостинг та розгортання

Для хостингу фронтенд-частини проекту був вибраний сервіс GitHub Pages, який дозволяє швидко та безкоштовно публікувати статичні вебсторінки. Він підходить для простих застосунків і MVP та надає безкоштовний хостинг з доменом у форматі username.github.io, що дозволяє уникнути витрат на сервери і складних налаштувань. GitHub Pages також

підтримує інтеграцію з GitHub репозиторіями, що дозволяє автоматично оновлювати сторінки після кожного коміту.

Для реалізації збереження даних, створених користувачем, використано Google Apps Script для інтеграції з Google Drive. Google Apps Script надає можливість зберігати дані та взаємодіяти з API без необхідності налаштовувати власний сервер. За допомогою GAS можна створювати, редагувати та читати файли з Google Drive. Цей підхід забезпечує безкоштовне збереження даних на хмарних сервісах і дозволяє зберігати дані навіть без необхідності налаштовувати сервер.

Безкоштовний хостинг на GitHub Pages і можливість збереження даних через GAS дозволяють оцінити можливості розгортання MVP без додаткових витрат. Це дозволяє зосередитися на розробці функціоналу, не витрачаючи час і ресурси на налаштування інфраструктури.

2.4 Безпека збереження даних

У процесі вибору технологій для збереження даних окрему увагу приділено питанню безпеки. У застосунку використовуються два основні механізми збереження: локальне сховище браузера LocalStorage та хмарне збереження через Google Apps Script з інтеграцією у Google Drive. Обидва ці варіанти мають певні переваги у простоті впровадження, але також містять обмеження з точки зору захисту даних.

LocalStorage дозволяє зберігати дані на стороні користувача без участі сервера, що позитивно впливає на швидкодію та зменшує залежність від зовнішніх ресурсів. Проте слід враховувати, що дані, збережені у LocalStorage, не шифруються за замовчуванням. Будь-який користувач, який має доступ до браузера або відкриє консоль розробника, може переглянути або змінити ці дані. Це створює потенційний ризик, якщо в LocalStorage зберігаються критичні налаштування або конфіденційна інформація. У контексті цього застосунку такий ризик є обмеженим, оскільки зберігається

лише стан інтерфейсу, але в майбутньому слід враховувати потребу в додаткових засобах захисту або переході на більш безпечне локальне сховище.

Для хмарного збереження використовується Google Apps Script, який надає доступ до файлів Google Drive і дозволяє реалізовувати просту серверну логіку без потреби у власному сервері. Але ця платформа має свої обмеження. Наприклад, усі операції з файлами виконуються в контексті облікового запису користувача, і якщо скрипт працює як вебзастосунок, важливо правильно налаштувати права доступу, щоб уникнути несанкціонованого зчитування або перезапису даних. Також відсутність шифрування на рівні переданих даних між клієнтом і сервером вимагає впевненості в тому, що передаються лише не чутливі дані. У поточному проєкті це виправдано, бо вся інформація, що передається, є публічною за своєю природою – логічні формули, координати, зображення елементів [23].

Використання сторонніх сервісів або повноцінних бекендів у цьому проєкті було свідомо обмежено. Такий підхід зменшує кількість точок доступу до даних і спрощує загальну архітектуру. Водночас це означає, що на стороні клієнта відсутні захисні механізми, характерні для серверних рішень – такі як автентифікація, авторизація, шифрування, захист від CSRF або XSS-атак. У контексті навчального застосунку це є допустимим компромісом, однак у разі переходу до використання системи на реальних даних або у середовищі з багатьма користувачами ці аспекти потребують обов’язкового доопрацювання.

2.5 Ліцензії використаних технологій

Використання сторонніх бібліотек і технологій у проєкті вимагає розуміння умов їх розповсюдження та ліцензування. Це особливо важливо при публікації проєкту у відкритому доступі або подальшому комерційному використанні. У складі застосунку використовуються компоненти, що

розповсюджуються за відкритими ліцензіями, що дозволяє уникнути юридичних обмежень і зобов’язань, пов’язаних із платними або закритими рішеннями.

Основна бібліотека React, що відповідає за побудову інтерфейсу, має ліцензію MIT, яка дозволяє вільно використовувати, змінювати та розповсюджувати програмне забезпечення без вимоги відкривати вихідний код або вказувати джерело в публічних інтерфейсах. Така ліцензія зручна як для індивідуального навчання, так і для подальшого розвитку у відкритому або закритому середовищі. З аналогічною ліцензією поширюється Vite, що використовується як система збірки. Вона також не містить обмежень щодо використання в комерційних або приватних проєктах, що робить її сумісною з будь-якими сценаріями розвитку.

Менеджер пакетів npm, який використовується для керування залежностями, надає доступ до великої кількості бібліотек, більшість з яких поширюються за відкритими ліцензіями типу MIT, Apache 2.0 або BSD. Під час створення застосунку перевірялись основні залежності на предмет наявності ліцензії, що дозволяє використовувати їх в освітньому проєкті.

Окремим елементом є використання Google Apps Script. Ця технологія надається в межах облікового запису Google, її використання є безкоштовним, але обмежується умовами сервісу. Сценарії, створені на основі Google Apps Script, також можуть бути поширені, однак із врахуванням того, що виконання коду відбувається на стороні інфраструктури Google і підпорядковується відповідним політикам використання сервісів Google [23].

Усі обрані технології не накладають обмежень на збереження, зміну чи повторне використання створеного застосунку. Це забезпечує гнучкість у розгортанні, тестуванні та демонстрації проєкту як прикладу роботи, без необхідності отримання окремих дозволів або ліцензійних угод. З правової точки зору такий підхід є оптимальним для індивідуальних розробників і навчальних цілей.

2.6 Технологічне забезпечення розвитку застосунку

У майбутньому можливе вдосконалення застосунку шляхом оновлення або заміни деяких технологій на більш ефективні або гнучкі. Наприклад, замість Vite може бути використано Next.js або Astro у разі потреби серверного рендерингу, кращої оптимізації для пошукових систем або розширеної маршрутизації. Якщо обсяг даних зростатиме, або потрібно буде зберігати складні структури, замість LocalStorage доцільно перейти на IndexedDB або хмарні сервіси типу Firebase, які також дозволяють працювати з багатьма пристроями одночасно. Google Apps Script є зручним рішенням на початковому етапі, але при зростанні навантаження або необхідності складнішої обробки даних доцільно створити власний сервер на Node.js з використанням бази даних. Стилізацію інтерфейсу, яка зараз реалізована засобами CSS, можна спростити за допомогою Tailwind CSS або styled-components, що дозволяють швидко створювати адаптивні інтерфейси без дублювання стилів. Якщо функціональність React виявиться обмеженою або буде потреба в покращенні продуктивності, можливий перехід на інші бібліотеки, наприклад, SolidJS або Svelte. Для керування станом додатку слід додати Redux. Ці зміни не є обов’язковими, але їх доцільно розглядати як варіанти масштабування та оптимізації проєкту відповідно до нових вимог.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Архітектура та принципи роботи програмної системи

Для розробки програмного забезпечення було обрано компонентну архітектуру, притаманну сучасним односторінковим вебзастосункам (SPA), з використанням бібліотеки React. У цій архітектурі основними будівельними блоками є компоненти – ізольовані, повторно використовувані частини інтерфейсу, які можуть мати власний стан і логіку.

Застосунок складається з таких основних компонентів, як інструментальна панель, полотно для побудови схем, логічні елементи (вентилі), панель властивостей і модальні вікна. Всі компоненти взаємодіють між собою через систему передачі властивостей (props) і використовують локальний або глобальний стан для зберігання даних. Для глобального управління станом може використовуватись React Context або Redux, залежно від складності реалізації. Така архітектура дозволяє ефективно розділити логіку програми на незалежні модулі, що спрощує тестування, масштабування та повторне використання коду. Крім того, компонентний підхід значно пришвидшує розробку та полегшує підтримку інтерфейсу користувача. Кожен компонент реалізований із урахуванням масштабованості та повторного використання у майбутньому.

Застосунок побудовано на основі головного компонента App.jsx, який відповідає за рендеринг усього інтерфейсу користувача та робочого простору. У цьому компоненті підключаються всі функціональні модулі, зокрема:

- SaveMenu (файл ./SaveMenu) – модальне вікно для налаштування параметрів збереження;

- Overlay (файл ./Overlay) – модальне вікно для створення нових вентилів з можливістю задання їх назви, вигляду та позицій контактів;

- GateButtons (файл ./GateButtons) – бічне меню з усіма вентилями користувача та короткою інформацією про них;

- SearchMenu (файл ./SearchMenu) – панель пошуку, розташована у верхньому лівому куті, з полями для фільтрації вентилів;

- SelectionBox (файл ./SelectionBox) – механізм виділення кількох об’єктів одночасно;

- Gate (файл ./Objects/Gate) – універсальний компонент для відображення вентилів із можливістю налаштування параметрів;

- Lamp (файл ./Objects/Lamp) – універсальний компонент для відображення ламп із можливістю налаштування параметрів;

- Link (файл ./Objects/Link) – універсальний компонент для відображення дротів із можливістю налаштування параметрів;

- Button (файл ./Objects/Button) – універсальний компонент для відображення кнопок із можливістю налаштування параметрів.

Окремо імпортуються функції для роботи з об’єктами:

- функціїї stopCreatingLink, onConnectorClick, addJoint, increaseLinkId } (з ./Objects/Object) – скасування або запуск створення дроту, додавання згину, генерація нового ID;

- функція generateFormula (з ./FormulaBilding) – генерація логічного рівняння на основі побудованої схеми;

- функція getPositionLamp (з ./Objects/Lamp) – визначення координат лампи;

- функція getPositionGate (з ./Objects/Gate) – визначення координат вентиля;

- функція getPositionButton (з ./Objects/Button) – визначення координат кнопки.

У файлах ./Objects/Gate, ./Objects/Lamp та ./Objects/Button підключаються функції з ./Objects/Object, які відповідають за видалення, переміщення, з’єднання дротами та передачу логічного сигналу. Ці функції забезпечують коректну взаємодію компонентів у процесі симуляції схем.

У файлі ./GateButtons підключається функція loadFromGoogleDrive з ./GoogleDriveSaving для завантаження даних про логічні вентилі з Google Drive.

У файлі ./Overlay підключаються посилання на масиви formula, img, connectors, highlight, names з ./GateButtons для оновлення бічної панелі при додаванні нового вентиля. Також додаються функції saveImgToGoogleDrive і saveTxtToGoogleDrive з ./GoogleDriveSaving для збереження нового вентиля на Google Drive.

У файлі ./SaveMenu підключаються посилання на масиви formula, img, connectors, names з ./GateButtons для оновлення бічної панелі при підключенні папки з вентилями з Google Drive. Також додаються функції saveImgToGoogleDrive і saveTxtToGoogleDrive для збереження створених раніше вентилів у підключену папку на Google Drive.

3.2 Реалізація логіки застосунку

У цьому розділі детально описано основні компоненти логіки застосунку, що забезпечують його функціональність та зручність для користувача. Особливу увагу приділено інтерфейсу, роботі з логічними елементами, пошуку вентилів, можливості створення власних компонентів, а також реалізації збереження даних як локально, так і в хмарному середовищі Google Drive.

3.2.1 Загальний опис інтерфейсу

Інтерфейс застосунку складається з верхнього заголовка – елемента header, який містить основні кнопки для взаємодії з робочим полем. Зокрема, це кнопки додавання лампочок (addLamp), кнопок (addButton), створення нових вентилів, збереження даних (SaveMenu) та перемикання відображення сітки (Grid). Зліва розташована бокова панель GateButtons, яка динамічно

змінюється залежно від вмісту і призначена для швидкого доступу до наявних логічних вентилів. У верхній частині цієї панелі знаходиться форма пошуку SearchMenu, що включає текстові поля search-field для введення назви вентиля, кількості входів і виходів. Після заповнення відповідних полів користувач може натиснути кнопку find, після чого перелік вентилів автоматично фільтрується за заданими критеріями. Кнопка reset очищає всі поля форми та повертає повний список елементів. Основну частину інтерфейсу займає робоче поле – центральна зона, де користувач безпосередньо взаємодіє з логічними компонентами. Додавання нових елементів здійснюється через відповідні кнопки інтерфейсу, які викликають функції додавання об’єктів до відповідних масивів – кнопок, вентилів або лампочок. Робоче поле оновлюється у фіксованій послідовності: спочатку рендеряться кнопки (buttons), потім вентилі (gates), далі лампочки (lamps), і лише після цього з’єднання між елементами (links). Це забезпечує коректну візуалізацію і відображення актуального стану всієї схеми. У процесі роботи можуть з’являтися модальні вікна, зокрема Overlay для створення нового вентиля або SaveMenu для збереження проєкту на Google Drive. Такі вікна розміщуються поверх основного інтерфейсу, частково затемнюючи його, щоб зосередити увагу користувача на поточному діалозі і запобігти випадковій взаємодії з іншими елементами.

3.2.2 Робота з об’єктами

Кожен елемент після додавання на робоче поле можна переміщати, утримуючи ліву кнопку миші. Після натискання кнопки миші на елементі викликається функція startDragging. Вона спрацьовує лише якщо натискання було не по конектору. В ній фіксується початок перетягування – встановлюється флаг dragging у true і обчислюється зсув миші відносно лівого верхнього кута елемента, щоб при переміщенні курсор не зміщувався. Під час руху миші викликається функція updateDraggingPosition. В ній

координати миші використовуються для обчислення нової позиції елемента. Межі робочої області беруться з contentRef, і елемент не дозволяється виводити за межі. Якщо ввімкнено режим сітки, координати округлюються до кратних 20 пікселів, із зсувом для центрування. Далі, коли елемент переміщується, оновлюються координати дротів, підключених до цього елемента. Це робиться в межах тієї ж функції: проходиться по кожному дроту, знаходяться відповідні конектори, і координати кінців дроту зміщуються згідно з новою позицією елемента. Завершується процес функцією stopDragging, яка просто скидає флаги dragging та isConnector.

Видалення елементів відбувається шляхом натискання середньої кнопки миші. При цьому спрацьовує функція deleteObject, яка приймає ідентифікатор елемента та поточні масиви зв’язків і об’єктів. Спочатку з масиву зв’язків видаляються всі ті, які були пов’язані з цим елементом – тобто де він виступає або джерелом, або ціллю з’єднання. Далі з масиву об’єктів видаляється сам елемент, і обидва оновлені масиви передаються в відповідні функції для збереження нового стану. Це гарантує, що після видалення елемента не залишиться жодних зайвих зв’язків або посилань на нього в логіці програми.

Для групового виділення реалізовано механізм, аналогічний до графічних редакторів: при утриманні правої кнопки миші з’являється прямокутна область, що динамічно змінюється між точкою натискання та поточним положенням курсору. Після відпускання кнопки всі елементи, які повністю або частково потрапили в межі цього прямокутника, вважаються виділеними. Після цього з вибраною групою можна виконувати різні дії: переміщення, копіювання або видалення. Для цього функція triggerEventForSelected штучно генерує події mousedown, mousemove або mouseup для кожного вибраного елемента, відтворюючи звичну поведінку перетягування. Це забезпечує узгоджену взаємодію між виділеними об’єктами та логікою їх обробки на сторінці. Щоб уникнути одночасного запуску великої кількості подій, між їх викликами застосовується невелика

затримка. Положення прямокутника оновлюється в межах контейнера, що запобігає виходу за кордони екрану. Всі відповідні події передаються до елементів через симуляцію, завдяки чому забезпечується синхронне оновлення їх стану.

У кожного логічного вентиля в параметрах зберігається формула, яка визначає його поведінку. Коли змінюються вхідні сигнали, ця формула автоматично обчислюється за допомогою функції evaluateLogicalExpression. Вираз попередньо розбивається на послідовність токенів, серед яких можуть бути логічні оператори (AND, OR, NOT, BUFFER), імена змінних та дужки. Далі відбувається покрокова обробка цих токенів з використанням двох стеків: один призначений для збереження значень, інший – для операцій. У процесі обчислення функція calculateRemainingOp контролює порядок виконання логічних операцій, дотримуючись правил пріоритету. Вона перевіряє, чи залишилися ще операції, що можуть бути виконані на поточному етапі, і поступово їх обробляє. Унарні оператори, як-от NOT або BUFFER, виконуються безпосередньо в цій функції, тоді як для бінарних операторів (наприклад, AND та OR) викликається функція applyLogicalOp, яка застосовує відповідну логічну операцію до двох останніх значень у стеку результатів. Після завершення обчислення результат записується на вихід вентиля, що дозволяє використовувати його в подальших з’єднаннях або симуляціях логічної схеми. Такий підхід забезпечує гнучку та уніфіковану обробку складних логічних виразів, незалежно від кількості входів чи виду операцій.

Щоб додати дріт для з’єднання елементів, потрібно натиснути на один із «входів» або «виходів» будь-якого елемента. У цей момент викликається функція onConnectorClick із файлу Object.jsx – вона є універсальною для всіх типів елементів. Спочатку функція перевіряє, чи не перевищує кількість з’єднань допустиму норму: для входу дозволено лише одне з’єднання. Якщо ця умова порушена, процес створення дроту переривається через виклик stopCreatingLink.

Після першого кліку користувач може натискати на будь-яке порожнє місце, щоб задати додаткові точки згину дроту. У такі моменти викликається функція addJoint, яка додає координати згину до масиву coordinates. Кількість згинів не обмежується – їх можна додавати скільки завгодно. Завершується створення з’єднання другим кліком по іншому «входу» або «виходу». Знову спрацьовує функція onConnectorClick, яка цього разу додатково перевіряє три умови: по-перше, щоб обидва контакти не належали одному елементу, по-друге – щоб типи контактів були різні (тобто вхід і вихід), і по-третє – щоб таке з’єднання ще не існувало. Лише після цього дріт вважається завершеним і додається до загальної схеми.

Дроти виконують роль провідників логічних значень між елементами схеми. Вони можуть передавати лише два стани – логічний «0» або логічну «1». Джерелом сигналу можуть бути кнопки, які встановлюють певне значення на виході при натисканні або утриманні. Вентилі отримують ці значення на свої входи, обчислюють результат згідно з власною логікою і передають його далі на вихід, до якого можуть бути підключені інші елементи. Лампочки використовуються для візуалізації результату – вони змінюють свій стан залежно від значення дроту, підключеного до їхнього входу. Якщо дріт несе значення «1», лампочка світиться, якщо «0» – залишається вимкненою. Таким чином, дроти формують зв’язки між усіма елементами схеми та забезпечують поширення логічних сигналів у реальному часі.

3.2.3 Інструмент пошуку вентилів

У лівому верхньому куті застосунку розташоване меню пошуку, яке складається з трьох текстових полів і двох кнопок. Текстові поля призначені для введення параметрів пошуку за трьома критеріями: назва вентиля (Name), кількість входів (Inputs) та кількість виходів (Outputs). Кнопки використовуються для запуску пошуку та скидання введених параметрів.

Коли користувач вводить значення хоча б в одне з полів і натискає кнопку запуску пошуку, з введених даних формується пошукова маска. Ця маска передається в логіку фільтрації бокової панелі, де відбувається перевірка кожного вентиля на відповідність умовам. Пошук за назвою (Name) є нечутливим до регістру – тобто введене значення повинно входити до назви вентиля повністю або частково, незалежно від великих чи малих літер. Наприклад, введення "and" відфільтрує як "AND", так і "CustomAnd". Поля Inputs та Outputs використовуються для фільтрації за точною кількістю входів і виходів відповідно. Якщо в полі Inputs вказано значення "3", то до результату пошуку потраплять лише ті вентилі, у яких рівно три входи. Те саме стосується виходів. Якщо якесь із полів залишене порожнім, відповідний параметр не враховується при фільтрації. Наприклад, якщо заповнено лише поле Name, то пошук відбудеться лише за назвою, а кількість входів та виходів не впливатиме на результат. Кнопка скидання повністю очищає всі введені параметри та повертає бокову панель у початковий стан, тобто відображає всі наявні вентилі без фільтрації. Такий підхід дозволяє користувачу швидко знаходити потрібний елемент навіть у великому списку, комбінуючи кілька критеріїв одночасно або використовуючи лише один з них.

3.2.4 Створення нового вентиля на основі побудованої схеми

Після завершення побудови логічної схеми користувач повинен натиснути кнопку в верхній частині інтерфейсу, яка ініціює процес генерації нового вентиля. У цей момент викликається функція generateFormula з модуля FormulaBuilding.jsx. Вона відповідає за побудову деревоподібної структури на основі даних з масивів buttons, gates і lamps, які зберігають інформацію про всі компоненти схеми. Оскільки ламп може бути кілька, формується або одне дерево, або декілька – по одному на кожну лампу. Такий підхід використовується бо в кожного виходу майбутнього вентиля

може бути унікальна та незалежна логіка, яку треба правильно опрацьовувати та не заважати роботі інших вентилів.

Далі ця структура конвертується у текстову формулу, яка відображає логіку роботи вентиля. Усі кнопки, що виступають як входи, перетворюються на змінні виду v0, v1, v2 і так далі. Це майбутні входи нового елемента. Якщо в схемі є зациклення, вони враховуються через спеціальні змінні oN, де N означає, через скільки кроків обчислення буде доступний результат для підстановки. Наприклад, якщо вказано o3, це означає, що значення в цьому місці стане відомим після трьох обчислень. Результати всіх таких проміжних обчислень накопичуються в окремому масиві, який оновлюється при кожному проходженні. Якщо обчислення відбувається вперше, масив ініціалізується логічними нулями. Приклад роботи механізму створення нового вентиля на основі схеми вентиля XOR (рисунок 3.1):

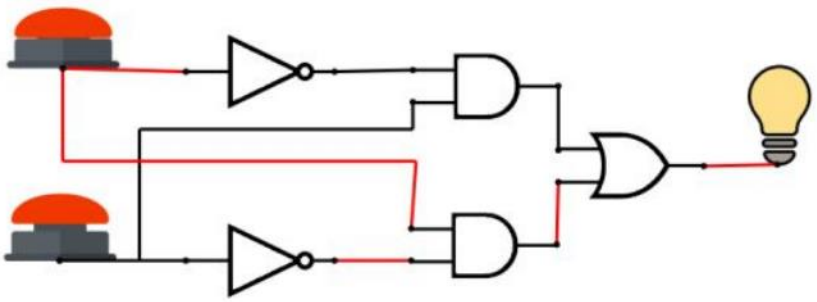

Рисунок 3.1 – Схема логічного вентиля XOR

Така побудована користувачем схема логічного вентиля XOR буде перетворена на дерево вигляду (лістинг 3.1):

Лістинг 3.1 – Дерево, сформоване на основі логічної схеми

```
{
  "formula": "OR",
  "children": [
    {
      "formula": "AND",
      "children": [
```

```
        { "formula": "NOT", "children": [ { "formula": "v0" }]},
        { "formula": "v1" }
      ]},
    {
      "formula": "AND",
      "children": [
       { "formula": "v0" },
       { "formula": "NOT", "children": [ { "formula": "v1" } ] }
      ]
    }
  ]
}
```

Це дерево конвертується у вираз ((NOT(v0))AND(v1)) OR ((v0)AND(NOT(v1))). Після генерації формули відкривається модальне вікно. У ньому користувач повинен вказати назву нового елемента, завантажити зображення (бажано у форматі PNG), та налаштувати положення контактів відповідно до зображення. Початкові координати входів і виходів розраховуються автоматично – на основі розташування кнопок і лампочок у побудованій користувачем схемі. Кнопки інтерпретуються як майбутні входи нового логічного елемента, лампочки – як його виходи.

Такий підхід дозволяє зберегти просторову логіку: якщо кнопка була розміщена у верхній лівій частині схеми, то і відповідний вхід нового вентиля за замовчуванням з’явиться пропорційно в тій самій області. Це суттєво спрощує орієнтування під час створення нових елементів, адже відносне положення контактів лишається незмінним незалежно від масштабу чи розміру картинки. Таким чином, форма компонента автоматично узгоджується з логікою побудованої користувачем структури, що забезпечує інтуїтивне розташування контактів без потреби вручну переносити кожен з них. За потреби, положення кожного входу та виходу можна уточнити до одного пікселя через спеціальне меню правої панелі.

Після завершення налаштувань натискається кнопка «Create», яка додає новий вентиль до загального масиву gates. Якщо активовано синхронізацію з Google Drive, система також зберігає всі відповідні файли у визначену папку. Якщо натиснуто «Cancel», модальне вікно закривається без збереження змін.

3.2.5 Механізм роботи збереження на Google Drive

Механізм збереження користувацьких компонентів на Google Drive реалізується через взаємодію веб-застосунку з сервісом Google App Script. Користувач самостійно створює окрему папку на своєму Google Drive, надає до неї загальний доступ за посиланням і вводить це посилання у спеціальне поле, що відкривається після натискання кнопки у правому верхньому куті інтерфейсу. Введене посилання зберігається у LocalStorage, завдяки чому не потрібно вводити його повторно при кожному наступному запуску платформи. Коли ця адреса вже збережена у LocalStorage, система автоматично звертається до Google Drive і завантажує користувацькі компоненти. Якщо посилання відсутнє, платформа працює зі стандартним набором логічних елементів.

Увесь обмін даними між клієнтською частиною та Google Drive здійснюється через скрипти, написані на Google App Script. Система дозволяє як збереження, так і завантаження файлів. Після активації «Upload to Google Drive», система завантажує зображення логічних вентилів, конвертує їх з PNG до формату base64 та завантажує до Google Drive, де воно набуває вигляду звичайної картинки, зберігає їх URL, а також текстовий файл, що містить технічну інформацію про компонент та посилання на відповідне зображення.

Для реалізації такого функціоналу було створено 3 скрипти UploadPNG, UploadTextFile, ListFolder. UploadPNG відповідає за завантаження зображень і повертає пряме посилання на PNG-файл після його розміщення. UploadTextFile створює текстовий файл з описом вентиля, включаючи його назву, кількість входів і виходів, формули логіки роботи, координати з’єднувачів та посилання на зображення. Весь вміст формується у вигляді JSON-об’єкта (лістинг 3.2). ListFolder – забезпечує повне зчитування вмісту папки: сканує всі текстові файли та повертає їх у вигляді масиву JSON-структур для подальшої обробки на клієнтській стороні.

Лістинг 3.2 – JSON-об’єкт з описів логічного вентиля XOR (файл XOR.txt)

```
{
  "name": "XOR",
  "inputs": 2,
  "outputs": 1,
  "formula": ["((NOT(v0))AND(v1))OR((v0)AND(NOT(v1)))"],
  "img":"https://lh3.google.com/u/0/d/1IHMSd9P7QkJ-
EHigNgbNOkk3ASfbTVph",
  "connectors": [
    {"id": 1,"type": "output","top": 11,"left": 83},
    {"id": 2,"type": "input","top": 1,"left": -8},
    {"id": 3,"type": "input","top": 21,"left": -8}
  ]
}
```

Формат збереження компонентів у текстових файлах стандартизований. Кожен JSON-файл описує один логічний вентиль (лістинг 3.2). Основні поля включають назву ("name"), кількість входів ("inputs") та виходів ("outputs"), масив логічних формул ("formula"), що відповідає поведінці кожного виходу, посилання на зображення компонента ("img"), а також список з’єднувачів ("connectors"), де для кожного елемента вказано його унікальний ідентифікатор ("id"), тип (вхід або вихід), та координати розташування на зображенні ("top" і "left"), це зручно використовувати замість звичних "x" та "y", бо в веб розробці позиція об’єкта визначається його зміщенням відносно лівого верхнього кута вікна браузеру.

3.2.6 Направляюча сітка

У правому верхньому куті інтерфейсу розміщена кнопка, яка активує направляючу сітку. Після активації на всьому робочому полі відображається сітка з фіксованим розміром клітинок 20 на 20 пікселів. Увімкнення сітки впливає на поведінку компонентів при їх переміщенні: вони можуть зміщуватись лише на величину кратну розміру клітинки, що забезпечує точне і рівномірне розташування елементів. У режимі сітки також обмежено спосіб прокладання з’єднувальних дротів – дозволено лише прямі лінії з поворотами під кутом 90 градусів. Це гарантує візуальну впорядкованість з’єднань і

дозволяє легше орієнтуватися в складних схемах. Додатково реалізовано функцію автоматичного з’єднання компонентів: якщо вихід одного елемента наближається до входу іншого на допустиму відстань, між ними автоматично створюється дріт. Це досягається шляхом перебору всіх можливих пар контактів між елементами та перевірки умов їх просторового перетину або дотику.

3.2.7 Локальне збереження даних

Механізм локального збереження стану робочого поля побудований на обміні даними у форматі JSON. Після кожного переміщення елемента, зміни його властивостей або встановлення нового з’єднання – поточний об’єкт (лістинг 3.3) зі всіма характеристиками записується в LocalStorage під єдиним ключем. Завдяки цьому при перезавантаженні сторінки відновлюється стан робочого поля буквально в тому самому вигляді, в якому його було залишено. Такий механізм працює за допомогою localStorage.setItem('key', JSON.stringify(state)) для збереження і JSON.parse(localStorage.getItem('key')) для відновлення.

Лістинг 3.3 – JSON-структура для збереження даних в LocalStorage

```
{
   "key":{
      "elements":[
         {"id":0,
     "connectors":[{"id":1,"type":"output","top":33,"left":31}],
            "isSelected":false,
            "position":{"top":677,"left":501},
            "type":"button"
         },
         {"id":1,
     "connectors":[{"id":1,"type":"input","top":53,"left":10}],
            "isSelected":false,
            "position":{"top":657,"left":898},
            "type":"lamp"
         }
      ],
      "links":[
         {"id":2,
```

```
            "coordinates":{"x":[323,459],"y":[633,628]},
            "idObject1":0,
            "idConnector1":1,
            "idObject2":1,
            "idConnector2":1,
            "state":false
         },
      ]
   }
}
```

Об’єкт з ключем "key" (лістинг 3.3) використовується як кореневий елемент для збереження даних у LocalStorage. Він містить повну структуру стану робочого поля, яка поділена на два основні розділи: "elements" та "links". Це дозволяє відновлювати інтерфейс користувача після перезавантаження сторінки або при відкритті проєкту в майбутньому.

Розділ "elements" містить масив об’єктів, кожен з яких представляє елемент, розташований на полі. Основним ідентифікатором елемента є поле "id", що забезпечує унікальність. "position" визначає координати елемента у пікселях щодо верхнього лівого кута області перегляду. Це дає змогу точно відтворювати розташування елементів. Поле "type" описує категорію елемента: це може бути кнопка, лампа або логічний вентиль. Статус виділення зберігається у полі "isSelected", що використовується, наприклад, при копіюванні або груповому редагуванні. Елемент може мати один або кілька "connectors" – точок підключення до інших елементів. Кожен конектор описується полями "id", "type" (вхід чи вихід) та координатами "top" і "left" відносно елемента, що дозволяє точно визначити розташування конектора на зображенні. Якщо елемент є логічним вентилем, він додатково має поле "formula", яке містить математичний вираз, що описує логіку обчислення його виходу. Для візуального представлення елементу також передбачене поле "img", що містить шлях до зображення, яке відображається на полі.

Розділ "links" описує зв’язки між конекторами елементів. Кожен зв’язок має унікальний "id", а також координати "x" і "y" – масиви з двома значеннями, які представляють точки початку і завершення лінії, що

малюється. На відміну від координат самих елементів, тут застосовуються координати у звичному вигляді, оскільки лінії зв’язків створюються за допомогою SVG, де координатна система прив’язана до горизонтальної та вертикальної осей. Крім цього, зв’язок має поля "idObject1" і "idObject2", що вказують на ідентифікатори об’єктів, між якими він встановлений, та "idConnector1" і "idConnector2", які визначають, до яких саме конекторів елементів підключено лінію. Це дозволяє однозначно відтворити логічну структуру схеми. Поле "state" зберігає логічне значення – 0 або 1 – що відповідає поточному стану сигналу в цьому з’єднанні.

Таке рішення вирішує дві головні задачі: не потребує зовнішньої бази даних і працює офлайн, а також дозволяє зберігати складні мережі елементів зі зв’язками простої структури без додаткового налаштування серверної частини. Якщо знадобиться версія з віддаленим бекапом, досить доповнити цей локальний підхід синхронізацією з хмарним сховищем (наприклад, Google Drive), повторюючи той самий варіант обміну через JSON.

3.3 Реалізація зовнішнього вигляду застосунку

Інтерфейс Logic Gates Lab побудований на принципі чіткого візуального поділу функціональних областей. Верхня панель має темно-сірий фон й містить елементи керування, що завжди залишаються видимими. Бічна панель такого ж кольору використовується для відображення переліку доступних логічних елементів. Основна частина інтерфейсу – це робоча зона для побудови схем, білого кольору, яка займає більшу частину екрану для зручності роботи з елементами. Усі вікна, модальні форми й діалоги виконані з білим фоном, з легким тінюванням та закругленими кутами. Сітка на робочій зоні реалізується як фон, створений за допомогою CSS-градієнтів, що дозволяє точно вирівнювати елементи без перевантаження DOM. Для візуального зворотного зв’язку реалізовано підсвічування входів і виходів логічних елементів у вигляді зелених (вхід) і червоних (вихід) блоків із

напівпрозорістю. Вони з’являються лише при наведенні курсора, щоб не відволікати увагу під час роботи.

Анімації використовуються для індикатора завантаження (loader) створеного з використанням CSS @keyframes та градієнтів. Її тривалість залежить від часу завантаження ресурсів застосунку. Макет побудований з використанням сучасних інструментів CSS – Flexbox та Grid Layout, що забезпечує гнучке вирівнювання елементів у межах будь-яких розмірів екрана. Завдяки цьому інтерфейс залишається адаптивним і стабільним на різних пристроях. Вся структура CSS організована модульно – стилі розділено на окремі блоки відповідно до функціональних зон, що спрощує підтримку й дозволяє швидко масштабувати інтерфейс у майбутньому.

3.4 Розгортання проєкту

Проєкт розгортається на GitHub Pages. Це хостинг, що дозволяє публікувати статичні файли напряму з репозиторію. Для правильного формування шляхів у продакшн-версії застосунку в package.json прописується поле homepage з URL репозиторію. У даному випадку використовується адреса https://DenysOlkhovykNure.github.io/logic-gates-lab. Після збірки всі файли з директорії dist автоматично завантажуються в гілку gh-pages, яку GitHub використовує для хостингу. У package.json прописано кілька скриптів для керування проєктом:

- "dev" – запускає Vite у режимі розробки. Відкриває локальний сервер з автоматичним оновленням при зміні коду;
- "build" – виконує продакшн-збірку проєкту. Генерує оптимізовані HTML/CSS/JS-файли в директорії dist;
- "lint" – запускає ESLint для перевірки коду. Додає обмеження: не допускає попереджень і не дозволяє непотрібні директиви;
- "preview" – запускає локальний сервер на базі зібраного проєкту (dist). Використовується для перевірки готової версії перед деплоєм;

- "deploy" – виконує збірку і одразу публікує згенеровані файли на GitHub Pages за допомогою пакету gh-pages.

Для автоматизації розгортання може бути використаний GitHub Actions. Це дозволяє створити workflow, який автоматично запускається при пуші в репозиторій. У ньому виконуються кроки: установка залежностей (npm ci), збірка (npm run build) і публікація (gh-pages -d dist). Усе виконується без втручання вручну. Альтернативний варіант – запуск скрипта npm run deploy вручну. У цьому випадку спочатку відбувається збірка, потім вміст dist публікується у гілці gh-pages. Обидва варіанти повністю сумісні і можуть використовуватись залежно від потреб.

Публікація проєкту на GitHub Pages забезпечує хостинг статичних файлів, доступних через веб-браузер. Це дозволяє користувачам відкривати застосунок за прямим посиланням без додаткових кроків. Застосунок збирається локально, після чого його вміст завантажується у гілку gh-pages, яка використовується GitHub для розгортання сайту. У результаті, сторінка завжди відображає актуальний стан проєкту. Такий спосіб розгортання дозволяє уникнути потреби в окремому сервері або налаштуванні хостингу. GitHub Pages підтримує HTTPS за замовчуванням, що забезпечує безпечне з'єднання. Процес оновлення проєкту може відбуватись вручну через запуск відповідного скрипта або автоматично через CI/CD, залежно від конфігурації репозиторію.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Цільова аудиторія та мета застосунку

Цільовою аудиторією застосунку є студенти, викладачі та всі, хто вивчає цифрову логіку, комп’ютерну інженерію або електроніку. Застосунок орієнтований на користувачів, які не мають великого досвіду роботи з професійними САПР-системами, але потребують інструмент для побудови логічних схем і перевірки їх роботи. Це можуть бути як першокурсники технічних спеціальностей, так і учні старших класів технічних ліцеїв. Водночас застосунок може бути корисним і для викладачів, які шукають простий спосіб демонстрації роботи логічних елементів у навчальному процесі.

Logic Gates Lab – це вебзастосунок для побудови та перевірки роботи цифрових схем на основі логічних елементів. Основна мета – надати простий інструмент, який дозволяє користувачу створювати цифрові схеми, візуалізувати їхню логіку та одразу бачити результати симуляції. Серед основних дій – створення нової схеми, розміщення логічних елементів на робочому полі, налаштування зв’язків між ними, а також запуск процесу симуляції, який показує, як дані передаються по схемі. Користувач може використовувати стандартні компоненти, такі як AND, OR, NOT, XOR, кнопки для подачі сигналу, лампочки для відображення результату та дроти, які з’єднують ці елементи. Усі зміни, зроблені на схемі, можна протестувати в режимі реального часу – кожне оновлення одразу впливає на результат, що дозволяє точно вивчати логіку та помічати помилки.

Застосунок підходить як для навчальних, так і для практичних цілей. Він дозволяє перевіряти правильність цифрових схем без потреби у фізичному обладнанні або складному ПЗ. Інтерфейс побудований так, щоб навіть початківець міг швидко розібратись у принципах побудови логічних

систем. Також є можливість створення власних компонентів зі своїм зовнішнім виглядом і поведінкою, що дає простір для експериментів і глибшого розуміння теми. Метою Logic Gates Lab є не лише створення віртуального конструктора, а й забезпечення середовища для навчання, експериментів і побудови складніших цифрових структур. Завдяки простому, але гнучкому підходу користувач може поступово переходити від простих схем до більш складних обчислювальних логік, будуючи цілісні системи з мінімальними обмеженнями.

4.2 Системні вимоги

Застосунок розроблений як клієнтський вебзастосунок, тому не потребує встановлення додаткового програмного забезпечення чи драйверів. Усі обчислення, візуалізація та взаємодія з користувачем виконуються безпосередньо в браузері.

Для коректної роботи достатньо будь-якого сучасного веб-браузера, що підтримує стандарт HTML5, Canvas API та JavaScript ES6. Рекомендованими є останні версії Google Chrome, Mozilla Firefox або Microsoft Edge. Мобільні браузери також здатні відображати інтерфейс, однак через обмежений розмір екрана використання на смартфонах не є зручним для складніших схем. Для збереження схем використовується механізм LocalStorage, тому пристрій повинен підтримувати цю технологію. Підключення до Інтернету потрібне лише під час першого завантаження сторінки, після чого застосунок може працювати в автономному режимі, доки не буде оновлено або перезавантажено сторінку. Проєкт не накладає високих вимог до апаратного забезпечення. Навіть пристрої з 1–2 ГБ оперативної пам’яті та базовим процесором здатні забезпечити комфортну роботу з простими або середньо складними схемами. Це дозволяє використовувати застосунок як у сучасних навчальних класах, так і на застарілому обладнанні.

4.3 Інтерфейс користувача

Застосунок реалізовано у лаконічному та функціональному візуальному стилі, що орієнтований на практичну роботу з логічними схемами. Інтерфейс побудовано у темних тонах з чітким контрастним окресленням елементів, що полегшує сприйняття навіть при великій кількості компонентів на полі. Іконки на кнопках управління мають стандартизоване графічне зображення, що відповідає загальноприйнятим схемотехнічним позначенням. Такий стиль забезпечує швидке розпізнавання функцій без необхідності додаткових підписів. Вони розташовані компактно по краям робочого вікна браузера, що дозволяє зосередити основну увагу користувача на робочому полі. Зовнішнє оформлення поєднує простоту, інформативність і зручність, що робить інтерфейс придатним як для навчальних, так і для проєктних задач. Для роботи з інтерфейсом застосунку користувач повинен орієнтуватися на основні елементи керування, розміщені у верхній та лівій частинах екрана.

У верхній панелі розташовано заголовок з кнопками, що відповідають за основні функції: додавання лампочок, додавання кнопок, створення нових вентилів, збереження логічних вентилів на Google Drive та вмикання або вимикання відображення сітки для зручності компонування елементів на робочому полі. Зліва знаходиться панель доступу до логічних вентилів. У її верхній частині користувач може скористатися формою пошуку: у відповідних полях вводиться назва, кількість входів та виходів, після чого натискається кнопка пошуку. Для скидання фільтрів використовується кнопка скидання, після натискання якої список логічних елементів відновлюється до повного. Нижче відображається перелік логічних елементів, які можна створити на робочому полі натисканням на потрібний, для побудови схеми. Центральну частину інтерфейсу займає робоче поле. Тут користувач створює та модифікує логічну схему, додаючи елементи з лівої панелі або за допомогою кнопок у заголовку. У разі натискання кнопки

для створення нового вентиля або збереження логічних вентилів на Google Drive з’являються модальні вікна, які затемнюють робоче поле й блокують взаємодію з іншими елементами. Усі дії у таких вікнах завершуються підтвердженням або скасуванням, після чого користувач повертається до головного інтерфейсу.

4.4 Робота з елементами на робочому полі: виділення, переміщення, з’єднання

У системі моделювання логічних схем робота з елементами на робочому полі передбачає низку дій, пов’язаних із їх переміщенням, виділенням, з’єднанням та видаленням. Після розміщення елемента на полі користувач має можливість змінити його положення, переміщуючи мишею. Для цього необхідно натиснути ліву кнопку миші безпосередньо на тілі самого елемента, виключаючи області контактів, які відповідають за з’єднання. Переміщення реалізується шляхом перетягування мишею до нового положення. Якщо на момент переміщення активовано режим сітки, координати елемента автоматично вирівнюються відповідно до вузлів координатної сітки, що дозволяє підтримувати чітку геометрію схеми. При переміщенні елемента всі з’єднання, що були до нього підключені, пересуваються синхронно, зберігаючи логічну структуру зв’язків.

Для видалення елемента передбачене використання середньої кнопки миші. Натисканням на будь-який елемент цією кнопкою він негайно вилучається з робочого поля. При цьому видаляються не лише сам елемент, але і всі дроти, які до нього підключені, що унеможливлює існування зайвих з’єднань.

Можливість групової взаємодії з кількома елементами передбачає використання рамки виділення. Для цього користувач натискає праву кнопку миші і, не відпускаючи її, створює прямокутну область, у межах якої знаходяться потрібні об’єкти. Після завершення виділення (відпускання

кнопки) ці елементи вважаються активними для наступних дій, таких як переміщення, копіювання або одночасне видалення.

З’єднання елементів між собою реалізується через інтерфейс входів і виходів. Початковою дією є натискання на контакт виходу або входу будь-якого елемента. Після цього користувач має можливість створити маршрут з’єднання, позначаючи проміжні точки у вільному просторі, що змінюють напрям дроту. Завершення з’єднання відбувається шляхом натискання на контакт іншого елемента. У системі реалізовано перевірку коректності з’єднання: допустиме лише одне підключення до кожного входу, а також з’єднання повинно відбуватись між різними елементами. У разі недотримання цих умов створення з’єднання автоматично скасовується.

Керування сигналами в схемі здійснюється за допомогою елементів типу кнопок. Вони формують логічний сигнал, що передається по з’єднаннях до інших елементів. Логічні компоненти, залежно від свого типу, обробляють отримані сигнали відповідно до вбудованих правил. Результатом обробки є нові сигнали, які передаються далі за схемою. Для візуалізації результатів використовуються лампочки. Вони мають два стани: активний та неактивний. При надходженні сигналу зі значенням логічної одиниці лампочка вмикається. У випадку сигналу з нульовим значенням вона залишається вимкненою. Усі обчислення в системі виконуються автоматично при будь-якій зміні вхідних даних, що забезпечує динамічне оновлення результатів у режимі реального часу.

4.5 Створення користувацьких вентилів

Для створення користувацького логічного елемента спочатку необхідно побудувати електричну схему, яка відображає логіку майбутнього елемента. У робочій області потрібно розташувати всі необхідні компоненти – це можуть бути логічні вентилі, кнопки, які виконують роль вхідних сигналів, та лампочки, що слугують виходами. Кожен елемент у схемі повинен бути

коректно з’єднаний дротами з іншими компонентами відповідно до логічної структури. Після завершення монтажу слід переконатися, що схема функціонує відповідно до очікуваної логіки. Для цього необхідно протестувати її, змінюючи стани входів і перевіряючи реакцію виходів. На цьому етапі виявляються помилки проєктування або помилкові з’єднання.

Коли схема перевірена і працює належним чином, можна перейти до створення нового логічного елемента. У верхній частині інтерфейсу наявна кнопка, яка запускає процедуру генерації користувацького компонента. Після активації цієї команди відбувається автоматичне зчитування всіх об’єктів, розміщених у поточній схемі. На основі просторового розташування визначається, які компоненти є входами, а які – виходами. Після цього система аналізує схему, будуючи відповідні логічні вирази. Для кожного виходу формується окрема формула, яка описує залежність цього виходу від вхідних сигналів. Якщо схема містить кілька індикаторів, буде згенеровано кілька незалежних логічних формул.

Після завершення логічного аналізу відкривається вікно налаштувань нового елемента. Користувач має можливість надати йому унікальну назву, а також завантажити графічне представлення у вигляді зображення формату PNG. Це зображення буде відображатися у бібліотеці компонентів та на схемах під час його використання. Зображення для нового елемента доцільно обирати таким чином, щоб воно чітко та зрозуміло відображало функціональне призначення компонента. Краще використовувати прості іконки або малюнки з мінімумом деталей, щоб їх було легко розпізнати навіть у маленькому розмірі на схемі. Формат PNG обирається через підтримку прозорого фону, що дозволяє зображенню гармонійно відображатися поряд з різними фонами та елементами інтерфейсу. Крім того, система пропонує автоматично розраховані координати розміщення входів і виходів, які базуються на початкових позиціях кнопок та лампочок у побудованій схемі. За потреби, ці координати можна відкоригувати вручну з точністю до одного пікселя, використовуючи інструменти правої панелі

керування. Це особливо важливо у випадках коли потрібно точно вирівняти контакти для забезпечення естетичності та зручності використання.

Після завершення всіх налаштувань слід натиснути кнопку підтвердження, яка остаточно створює новий логічний елемент і додає його до наявної бібліотеки. Якщо в системі активовано синхронізацію з Google Drive, всі файли, пов’язані з новим компонентом, включаючи опис, формули та зображення, автоматично зберігаються у відповідній директорії в хмарному сховищі. Якщо ж користувач вирішує скасувати створення елемента, натискаючи відповідну кнопку, усі проміжні дані буде видалено, і вікно налаштувань закриється без внесення змін до бібліотеки.

4.6 Використання направляючої сітки

Щоб забезпечити рівне та точне розташування елементів на схемі, у правому верхньому куті передбачена кнопка для ввімкнення направляючої сітки. Після активації на робочому полі з’являється сітка з клітинками розміром 20x20 пікселів.

У режимі сітки переміщення компонентів здійснюється зі «зчепленням» до сітки – позиції елементів змінюються лише кратно розміру клітинки. Це виключає довільне розташування і допомагає підтримувати точність і акуратність компонування.

Прокладка дротів у цьому режимі обмежується прямолінійними сегментами з кутами 90 градусів, що забезпечує чистий і структурований вигляд схеми, особливо при складних конфігураціях.

Для полегшення з’єднання елементів реалізовано автоматичне підключення: якщо вихід одного компонента знаходиться в межах допустимої відстані від входу іншого, між ними автоматично прокладається дріт. Система перевіряє всі можливі пари контактів і створює з’єднання при їхньому близькому розташуванні.

4.7 Гарячі клавіші та комбінації клавіш

Реалізовано дві основні функції для зручного редагування схеми: копіювання виділених елементів і їх вставка. Перший крок – виділення елементів, які потрібно скопіювати. Для цього треба натиснути праву кнопку миші на робочому полі і, утримуючи її, протягни прямокутну область, що охоплює всі потрібні компоненти. Це виділення працює як рамка – усі елементи, що потрапили в цю зону, автоматично відмічаються для подальших дій.

Після того, як було виділено потрібні елементи, слід натиснути комбінацію клавіш Ctrl+C – це стандартна команда копіювання. Вона зберігає вміст виділення у тимчасовій пам’яті, щоб можна було швидко працювати з цими даними далі.

Друга функція – вставка скопійованих елементів. Натиснувши комбінацію Ctrl+V всі раніше виділені компоненти з’являться на робочому полі. Вставлені елементи можна вільно переміщувати, розміщуючи їх у будь-якому зручному місці. Це дає змогу легко дублювати потрібні частини схеми без повторного створення кожного елемента вручну.

Такі інструменти особливо корисні для роботи з великими схемами або повторюваними блоками. Вони значно скорочують час на редагування і зменшують ризик помилок, адже не потрібно повторно налаштовувати компоненти.

4.8 Збереження даних та обмін бібліотеками компонентів

У системі реалізовано механізм збереження та обміну бібліотеками логічних компонентів, що дозволяє користувачам не тільки ділитися напрацюваннями, а й зберігати свої особисті бібліотеки у хмарному сховищі Google Drive. Для збереження користувач створює на Google Drive окрему папку, налаштовує до неї загальний доступ із правами редактора, після чого

вставляє посилання на цю папку у відповідне поле в меню збереження вебзастосунку, активує опцію Upload to Google Drive та натискає кнопку Save. У результаті, всі створені ним компоненти (зображення та текстові файли з технічними даними) завантажуються до цієї папки.

Якщо користувач хоче поділитися бібліотекою з колегами, він передає їм це саме посилання. Інші користувачі вставляють його у своє поле збереження, деактивують опцію Upload to Google Drive та натискають Save. Перед збереженням даних застосунок автоматично перевіряє правильність посилання та доступність папки для запису. Після цього компоненти завантажуються у вебзастосунок і стають доступними для використання. Якщо ж користувач хоче використовувати Google Drive лише для власних потреб, він просто не передає посилання нікому. Таким чином, його бібліотека залишається приватною, але водночас доступною для повторного використання навіть після перезапуску програми або зміни пристрою.

Слід зазначити, що локальна робота вебзастосунку також зберігається – якщо користувач просто закриє вікно браузера, всі компоненти, розташовані на робочому полі, залишаться на своїх місцях при наступному запуску. Проте якщо користувач створив нові вентилі або редагував бібліотеку, і не підключив збереження на Google Drive – ці напрацювання можуть бути втрачені при очищенні кешу браузера або зміні пристрою. Тому для надійного збереження бібліотек рекомендується підключати Google Drive.

Якщо користувач хоче відмовитися від використання зовнішньої бібліотеки – достатньо натиснути кнопку Delete. Після цього при наступному запуску буде використано лише стандартні вбудовані логічні вентилі.

Основна мета описаного функціоналу – забезпечити обмін шаблонами між користувачами та спростити командну роботу. Наприклад, викладач може створити набір логічних елементів і поширити їх серед студентів для навчальних цілей. Такий підхід спрощує командну роботу, дозволяє зручно структурувати особисті напрацювання, а використання хмарного збереження робить систему більш гнучкою та незалежною від конкретного пристрою.

Користувач також має змогу редагувати вміст своєї хмарної бібліотеки безпосередньо через інтерфейс Google Drive, наприклад, перейменовувати файли, переміщувати їх між папками або видаляти непотрібні компоненти. Зміни автоматично враховуються при наступному завантаженні бібліотеки у вебзастосунок. Це дозволяє зручно підтримувати порядок у власних наборах компонентів без необхідності повторно завантажувати всю бібліотеку. Крім того, структура збереження у форматі PNG та JSON забезпечує простоту резервного копіювання – достатньо скопіювати всю папку на інший обліковий запис або пристрій.

4.9 Додаткові можливості використання логічних вентилів

Логічні вентилі зазвичай використовуються в комп’ютерній логіці, однак їхні принципи можна застосовувати і в ширшому контексті. Якщо замінити стандартні умовні позначення вентилів спеціально розробленими зображеннями, виникає можливість створення блок-схем, які не мають безпосереднього стосунку до електроніки. Такий підхід дозволяє візуалізувати послідовності дій, логіку виконання завдань або структуру складних процесів, що особливо корисно при розробці алгоритмів чи описі сценаріїв прийняття рішень.

В освітньому середовищі ці принципи можуть використовуватись для побудови ментальних карт, діаграм рішень або знаннєвих структур, де важливими є зв’язкі між поняттями, подіями чи діями. Наприклад, візуалізація математичних задач також може базуватися на такому підході: наприклад, у вигляді графів імовірностей, дерев рішень або комбінаторних конструкцій, які полегшують розуміння логіки задачі та її внутрішньої структури.

У технічних або виробничих процесах елементи, що працюють за логічним принципом, можуть представляти пристрої, етапи технологічного циклу або операції, а зв’язки між ними – порядок взаємодії чи умови

переходу між станами. Це дозволяє створювати наочні моделі технологічних систем, що сприяє оптимізації виробничих процесів або навчанню персоналу.

У моделюванні фізичних, хімічних чи біологічних явищ така візуалізація покаже взаємодії між об’єктами, причинно-наслідкові зв’язки, циклічні процеси чи гілки розвитку подій. Наприклад, можна змоделювати біохімічну реакцію як послідовність умовних переходів між речовинами, де логічна структура визначає порядок та умови перетворень.

У сфері адміністрування та бізнесу використання принципів логічних вентилів може допомагати в побудові моделей документообігу, етапів погодження, процедур контролю якості або механізмів ухвалення рішень. Кожен блок у схемі виконує роль певного кроку або рішення, а зв’язки визначають логіку його включення до загальної системи.

Навіть сюжетну структуру літературного твору можна представити у вигляді логічної схеми, де окремі події чи стани позначаються умовними блоками, а лінії між ними фіксують розвиток дії, її альтернативи або наслідки. Такий підхід дозволяє побачити логіку розвитку подій, що особливо важливо при аналізі сюжетів, побудові сценаріїв або навчанні наративному мисленню.

Усі ці приклади демонструють універсальність логічних принципів, що лежать в основі вентильних схем, і можливість їх застосування як інструменту для моделювання, пояснення та структуризації складних систем у різних сферах діяльності.

ВИСНОВКИ

У ході кваліфікаційної роботи розроблено вебзастосунок для автоматизації проєктування логічних схем, орієнтований на потреби студентів та викладачів у сфері комп'ютерної інженерії. Такий інструмент дозволяє значно покращити процес вивчення цифрової логіки, забезпечуючи зручне, наочне та доступне середовище для побудови та перевірки логічних схем без потреби у фізичному лабораторному обладнанні.

Під час виконання роботи:

- проведено аналіз популярних інструментів для симуляції логічних схем та виявлено їх обмеження;
- визначено перелік ключових функцій, необхідних для ефективного навчального застосунку;
- реалізовано можливість створення користувацьких логічних елементів із власним візуальним виглядом;
- реалізовано візуальну симуляцію сигналів у режимі реального часу;
- забезпечено збереження схем у хмарі за допомогою Google Apps Script та Google Drive.

Застосунок має простий інтерфейс, підтримує європейські та радянські графічні стандарти, дозволяє створювати бібліотеки логічних елементів і повторно використовувати компоненти у нових схемах. Завдяки хостингу на GitHub Pages і збереженню в хмарі, не потребує складного налаштування або встановлення. Проєкт уже протестували студенти, що підтвердило його практичну користь для навчання.

Результати роботи представлені на 29-му Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» [24] та на П'ятнадцятій міжнародній науково-технічній конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління» [25].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Floyd T. L. Digital Fundamentals. New York : Pearson, 2014. 953 p.

2. Kumar A. A. Fundamentals of digital circuits. Delhi : PHI Learning Pvt. Ltd., 2016. 1070 p.

3. Mano M. M., Ciletti M. D. Digital design with an introduction to the Verilog HDL. New York : Pearson, 2017. 1214 p.

4. Wick K. Introduction to digital logic: applied digital logic exercises using FPGAs. New York : Cambridge University Press, 2017. 18 p.

5. Savolainen V. Digital logic simulator comparison in educational purpose. Oulu : Університет прикладних наук Оулу, 2020. 23 p.

6. Feher J. Introduction to digital logic with laboratory exercises. Illinois : Creative Commons Attribution, 2009. 105 p.

7. Miczo A. Digital logic testing and simulation. Hoboken : John Wiley & Sons, Inc., 2003. 657 p.

8. Saha A., Manna N. Digital principles and logic design. Boston : Jones & Bartlett Learning, 2009. 489 p.

9. Rafiquzzaman M. Fundamentals of digital logic and microcomputer design. Hoboken : John Wiley & Sons, Inc., 2005. 813 p.

10. Miroschnyk M. A., Shkil A. S., Kulak E. N., Kucherenko D. Y. Design automation of easy-tested digital finite state machines. *Radio Electronics, Computer Science, Control: Zaporizhzhia National Technical University*. 2018. Vol. 2. P. 117-124. DOI: http://dx.doi.org/10.15588/1607-3274-2018-2-13

11. Miroshnyk M. A., Poroshyn S. M., Shkil A. S., Kulak E. N., Filippenko I. V., Kucherenko D. Y. Design of Logical Control Units Based on Finite State Machines Patterns. *16th IEEE East-WEST Design & TEST Symposium (EWDTS-2018)*. 2018. Vol. 9. P. 208-216.

12. Braun E. L. Digital computer design: logic, circuitry, and synthesis. New York : Academic Press, 1963. 582 p.

13. Holdsworth B., Woods C. Digital logic design. Pondicherry : Integra Software Services Pvt. Ltd., 2002. 500 p.

14. Ding Y. Li S., Liu J., Wu X. DCLab: A Web-based System for Digital Logic Experiment Teaching. *2018 IEEE Frontiers in Education Conference (FIE)*. 2018. P. 1-5. DOI: https://doi.org/10.1109/FIE.2018.8658917

15. Uzedhe G. O., Inyiama H. C., Udeze C. C., Mbonu E. S. Microcontroller Based Real-Time Emulator for Logic Gate and Structured Logic Devices. *International Journal of Science and Technology*. 2013. Vol. 9. P. 639-647.

16. Teach logic gates + digital circuits effectively – with Logicly. URL: https://logic.ly (дата звернення 20.05.2025).

17. Dive into the world of Logic Circuits for free. URL: https://circuitverse.org/ (дата звернення 20.05.2025).

18. Academo. URL: https://academo.org/ (дата звернення 20.05.2025).

19. Online Science Demos. URL: https://sciencedemos.org.uk/ (дата звернення 20.05.2025).

20. Logic Gate Simulator Online. URL: https://www.logic-gate.online/ (дата звернення 20.05.2025).

21. Svekis L. L., van Putten M., Percival R JavaScript from beginner to professional. Livery Place : Packt Publishing Ltd., 2021. 545 p.

22. Griffiths D., Griffiths. D. React cookbook recipes for mastering the React framework. O'Reilly, 2021. 50 p.

23. Ferreira J. Google Apps Script: web application development essentials. 2nd ed. O'Reilly, 2014. 216 p.

24. Ольховик Д. В. Автоматизація проектування та симуляції схем логічних вентилів. *XXIX Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті»*. 2025. Т. 5. С. 30-31.

25. Ольховик Д. В., Іващенко Г. С. Автоматизація проектування та симуляції схем логічних вентилів. *П'ятнадцята міжнародна науково-технічна конференція «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління»*. 2025. Т. 2. С. 22.