Київський національний університет імені Тараса Шевченка

Факультет радіофізики, електроніки та комп'ютерних систем

Кафедра комп'ютерної інженерії

«Система агрегації бронювань місць в закладах харчування»

«System for designing and simulating electronic circuits»

Дипломна робота магістра

студента 2 року навчання

спеціальності 123 «Комп'ютерна інженерія»

Дмитра ЗАЖИДЬКА

_____(підпис)

Науковий керівник:

кандидат фізико-математичних наук, доцент кафедри МТРФ

Дмитро ІВАНЕНКО

_____(підпис)

До захисту допускаю

Протокол засідання кафедри від

"___" _____2023р. №_____

Завідувач кафедрою

Юрій БОЙКО

Київ 2023

РЕФЕРАТ

Диплома робота за об’ємом складає 59 сторінок, містить 20 рисунків, 2 таблиці, 4 додатка, використано 16 інформаційних джерел.

Постановка задачі

Об’єктом даної роботи є моделювання та взаємодія у різних представленнях з електронними мікросхемами. Предметом роботи є програмний засіб для моделювання електронних мікросхем.

Головною метою роботи є розробка вебзастосунку для проєктування та моделювання електронних схем, а також порівняння розробленого засобу з наявними реалізаціями.

Методи розроблення: Метод створення програмного продукту вручну, використовуючи платформу для розробки вебзастосувань. Інструменти розроблення: безкоштовне, вільно поширюване програмне забезпечення Ngspice, мови програмування JavaScript, Python, бібліотека Django для реалізації серверної частини, бібліотека React.js для створення інтерфейсу користувача

Результати роботи: було проведено аналіз наявних рішень для моделювання мікросхем, що дозволило обрати основні критерії для розробки власного застосування. Розроблено систему, для проєктування та моделювання електронних мікросхем, оптимізовано цей застосунок внаслідок моделювання на серверній частині.

Ключові слова: МІКРОСХЕМА, EDA, ВЕБДОДАТОК, СЕРВЕР, КЛІЄНТ, DJANGO, REACT.JS, SPICE, NETLIST

ЗМІСТ

ВСТУП

В сучасному світі електроніка має ключове значення в різних галузях життя. Розвиток цієї сфери є безперервним, що вимагає від фахівців швидкого та якісного проєктування та симуляції електронних схем.

Одним з основних інструментів, що використовуються для розробки та аналізу електронних пристроїв є системи для проєктування та симуляції електронних схем. Такі системи дозволяють створювати, тестувати та аналізувати різні типи електронних пристроїв та схем, що забезпечує високу ефективність та точність розробки.

Для того, щоб створити реальну мікросхему, спершу її потрібно сконструювати та провести симуляцію роботи, для цього використовуються спеціальне програмне забезпечення – системи автоматизованого проєктування мікросхем. Такі системи є дуже важливим інструментом у сфері електронної інженерії. Вони дозволяють інженерам та дослідникам створювати, аналізувати та оптимізувати електронні схеми, забезпечуючи швидкий та точний процес проєктування. Багато з таких систем пропонують багатофункціональний інтерфейс, що дозволяє користувачам розробляти складні схеми з великою кількістю компонентів, проводити різноманітні експерименти та отримувати результати у режимі реального часу. На жаль нічого ідеального у світі не існує, тому в кожного рішення є свої переваги та недоліки. Основними проблемами при використанні систем автоматизованого проєктування систем є відсутність підтримки більше ніж 1 системи, висока ціна програмного забезпечення, а також іноді симуляція певних схем потребувала досить потужного обладнання.

Саме тому у цій роботі досліджуємо різні інструменти для проєктування та симуляції електронних схем та порівнюємо їх

характеристики, переваги та недоліки. Мета - з'ясувати, який з інструментів є найкращим для використання у конкретних ситуаціях та розробити систему для проєктування та симуляції електронних схем, яка задовольняє вимоги користувачів та має можливості для подальшого розширення та покращення.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОЗП — Оперативні Запам’ятовуючі Пристрої

ПЗ – Програмне забезпечення

ЦП – Центральний процесор

CSS - Cascading Style Sheets

DRF - Django Rest Framework

JS - JavaScript

JSON - JavaScript Object Notation

MVT - Model View Template

RAM - Random Access Memory

1 АНАЛІТИЧНИЙ ОГЛЯД ЗАВДАННЯ

1.1 Визначення електронних схем

Проєктування електронних схем, або проєктування ЕС, є частиною більш широкої галузі знань, відомої як електронна інженерія. У дисципліні електронної інженерії існує процес, відомий як проєктування електричних схем. Метою проєктування схеми є збірка колекції взаємопов'язаних елементів схеми, які виконують певну цільову функцію[1]. Простим прикладом є можливість додавання або множення чисел. Інший приклад - розробка мікропроцесора, який виконує комп'ютерні інструкції для виконання складних завдань.

Елементи схеми, що використовуються в цьому процесі, починаються з фундаментальних будівельних блоків, таких як транзистори, резистори, конденсатори та дроти. Ці елементи об'єднуються для формування більш складних функцій, таких як логічні вентилі або прецизійні підсилювачі, які потім об'єднуються для формування більш складних функцій, таких як суматори і помножувачі. Цей процес продовжує розвиватися, в результаті чого з'являються все більш складні блоки для побудови схем.

Дизайн схеми використовує дискретні, заздалегідь виготовлені елементи для формування схеми. У випадку з дизайном мікросхем є важлива відмінність. Тут елементи схеми виготовляються з мініатюрних компонентів, які реалізуються на кремнієвій підкладці за допомогою процесу, який називається фотолітографія. Процес фотолітографії створює різні геометричні форми на кремнієвій підкладці, де електричні властивості області, визначеної цією формою, змінюються. Основні елементи схеми створюються, коли ці області об'єднуються і накладаються одна на одну.

1.2 Важливість проєктування електронних схем

Проєктування ЕМС є критично важливою дисципліною. Вона є основою для розробки всіх мікроелектронних пристроїв, що використовуються сьогодні. Сюди входять мікропроцесори, які живлять ноутбуки та мобільні телефони, схеми обробки зображень, які живлять комп'ютерні монітори та телевізори, а також датчики, які використовуються в натільних та імплантованих медичних пристроях. Ці мікроелектронні пристрої також дозволяють дедалі ширше використовувати штучний інтелект (ШІ), який відкриває нові горизонти, такі як автономне водіння, машинний зір і обробка природної мови[1].

Впровадження ЕМС-технологій набуло широкого розповсюдження в нашому світі, а проєктування ЕМС формує фундаментальний набір дисциплін, необхідних для створення цих пристроїв.

1.3 Програми проєктування електронних схем

Системи автоматизації електронного проєктування (EDA) - це особлива категорія обладнання, програмного забезпечення, послуг та процесів, які використовують автоматизоване проєктування для розробки складних електронних систем, таких як друковані плати, інтегральні схеми та мікропроцесори[2]. Щільне розміщення елементів на друкованій платі або мікропроцесорі вимагає дуже складних конструкцій, і програмне забезпечення EDA значною мірою замінило ручне проєктування друкованих плат і напівпровідників автоматизованими, стандартизованими процесами, які сприяють швидкій розробці, зводячи до мінімуму помилки, дефекти та інші помилки при проєктуванні.

1.4 Аналіз технологій опису електронних схем

Опис електричних мікросхем - це процес створення опису логіки та функціональної поведінки електронної мікросхеми з використанням спеціалізованих мов програмування, таких як VHDL або Verilog. Опис мікросхем дозволяє інженерам проєктувати та тестувати цифрові схеми до їх фізичного створення, а також дозволяє проводити аналіз та оптимізацію елекбтричних характеристик мікросхеми. Опис мікросхем є важливою складовою в сучасному процесі проєктування електроніки. Існує кілька технологій, які можуть бути використані для моделювання та симуляції електронних мікросхем. Деякі з найбільш поширених технологій включають наступне:

Ngspice - це вільний та відкритий програмний пакет для симуляції електронних схем[3]. Цей пакет використовується для аналізу та проєктування аналогових, цифрових та змішаних схем, включаючи схеми з інтегральними мікросхемам.

Verilog та VHDL - це мови опису апаратного забезпечення (Hardware Description Languages, HDLs)[4]. Вони дозволяють користувачеві створювати моделі електронних мікросхем та виконувати їх симуляцію.

IBIS (Input/Output Buffer Information Specification) - це стандарт, який визначає формат вхідно-вихідних буферів для інтегральних схем. IBIS може бути використаний для моделювання поведінки вхідно-вихідних буферів на мікросхемі[5].

EDIF (Electronic Design Interchange Format) - це формат обміну даними для електронного дизайну[6]. EDIF може бути використаний для опису електронних мікросхем та їх компонентів.

Технологія	**Рік створення**	**Формат файлів**	**Призначення**	**Приклади використання**
Ngspice	1993	.cir	Симуляція електронних мікросхем	Моделювання поведінки електричних компонентів
Verilog	1984	.v, .sv, .vs	Опис апаратури в електронних системах	Розробка та верифікація цифрових схем, аналогових і сумішових схем
VHDL	1981	.vhd, .vhdl	Опис апаратури в електронних системах	Розробка імітаційних моделей, верифікація, проєктування та тестування цифрових схем
IBIS	1993	.ibs	Моделювання параметрів електронних схем	Моделювання параметрів сигналів в мережах передачі даних

EDIF	1989	.edf, .edn	Передача даних між електронними системами	Експорт та імпорт електричних схем між різними EDA-пакетами

Таблиця 1 Порівняння основних технологій опису мікросхем

Відповідно до опису технологій та таблиці можемо зробити висновок що найкращим буде застосування програмного пакета ngspice для розробки системи, через те, що, його призначення та приклади використання ідеально підходять під майбутню систему, окрім цього це ПЗ є кросплатформним програмним забезпеченням, тобто його можна використовувати на різних операційних системах, включаючи Windows, Linux та MacOS. Ngspice є також відкритим програмним забезпеченням, тому користувачі можуть змінювати його за потребою та використовувати для власних проєктів.

1.4 Огляд існуючих рішень

Основним з рішень яке використовується при виконанні лабораторних робіт студенти факультету радіофізики електроніки та комп'ютерних систем є - **Proteus Design Suite.**

Proteus Design Suite - це запатентований пакет програмних засобів, який використовується в основному для автоматизації електронного проєктування. Це Windows-додаток для захоплення схем, моделювання та проєктування макетів друкованих плат (PCB). Він може бути знайдений у

багатьох конфігураціях, залежно від розміру вироблених конструкцій та вимог до моделювання мікроконтролерів. Всі продукти PCB Design включають в себе автомаршрутизатор і базові можливості змішаного режиму SPICE моделювання.[7]

Програмне забезпечення використовується в основному інженерами та технічними спеціалістами з електронного дизайну для створення схем та електронних відбитків для виробництва друкованих плат (PCB), а також як інструмент швидкого створення прототипів для досліджень та розробок. Його також можна знайти в університетах по всьому світу, де викладають електроніку, вбудований дизайн і макетування друкованих плат для студентів. Він також має функції, які дозволяють віртуально моделювати проєкти Інтернету речей.

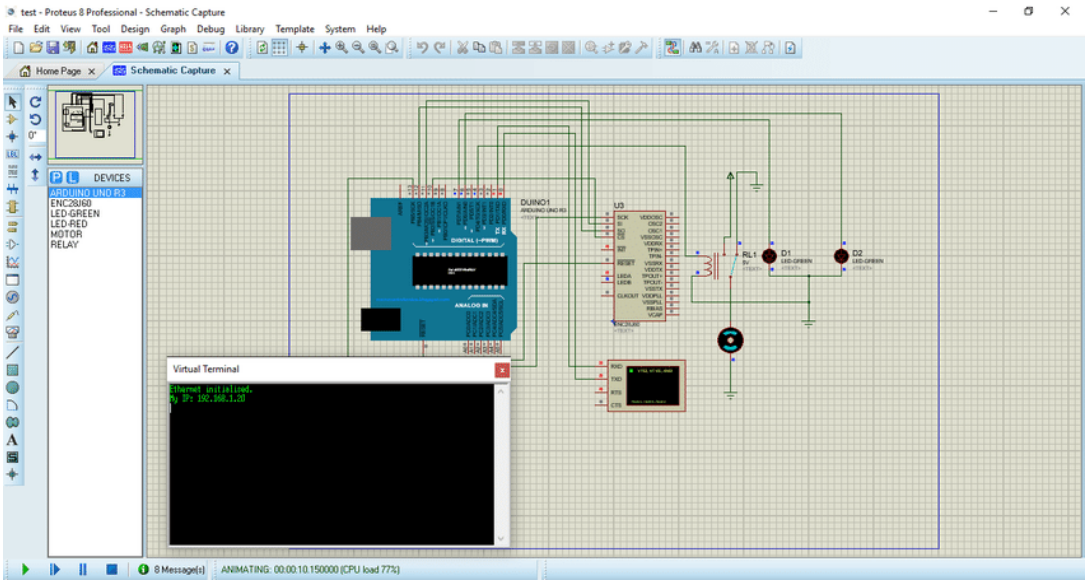


Рис.1 Загальний вигляд Proteus Design Suite

Переваги Proteus:

- Більше 15 мільйонів компонентів присутні в бібліотеці
- Наявність мікроконтролерів, таких як Raspberry Pi, PIC, Arduino
- Велика різноманітність інструментів аналізу

- Простий в освоєнні інтерфейс

Мінуси Proteus:

- Ціна пакетів надзвичайно висока.
- Немає безкоштовної версії програмного забезпечення.
- Відсутня кросплатформність.

Основною проблемою програмного забезпечення Proteus є те що в нього відсутня безкоштовна версія, точніше сказати що вона наявна, але має занадто обмежені можливості, а саме – заборонено зберігати проєкти як такі.

Окрім досить дорогого Proteus розглянемо популярний безкоштовний варіант з відкритим сирцевом кодом – **KiCAD.**

KiCAD є вільним програмним забезпеченням з деякими додатковими перевагами. Ви можете модифікувати програмне забезпечення на свій смак, редагуючи вихідний код. Він поставляється з 3D-переглядачем, який дозволяє візуалізувати друковану плату. Він виконує моделювання за допомогою "ngspice", який є безкоштовним інструментом SPICE[8].

Він чудово підходить для створення макетів друкованих плат, оскільки не має обмежень на розмір плати, кількість виводів або кількість шарів.

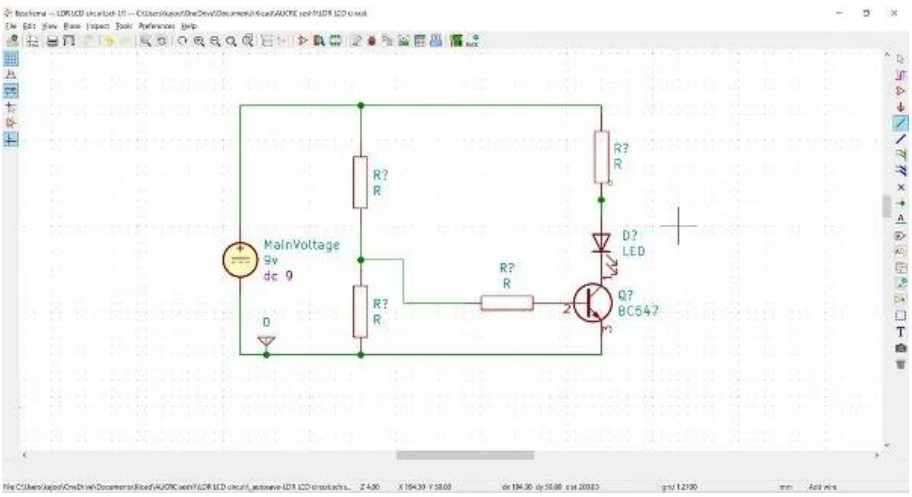


Рис. 2 Загальний вигляд ПЗ KiCAD

Переваги:

- Безкоштовне програмне забезпечення без обмежень.
- Потужний 3D-переглядач
- Немає обмежень на розмір плати або кількість шарів.

Мінуси:

- Відсутність функцій автомаршрутизації.
- Складна бібліотека компонентів.

Обидва розглянутих додатка є десктоп додатками, відповідно всі обчислення та симуляція відбувається безпосередньо на комп'ютері користувача, тому розглянемо також веб-додаток для побудови електронних схем - **CircuitLab**

CircuitLab - це браузерний програмний інструмент для створення схем і моделювання електричних ланцюгів, який допомагає швидко проєктувати і аналізувати аналогові та цифрові електронні системи[9].

Переваги:

- Крос платформність
- Обчислення та симуляція на стороні сервера

Недоліки:

- Обмеження в безкоштовній версії
- Відсутність підтримки PCB(printed circuit board)

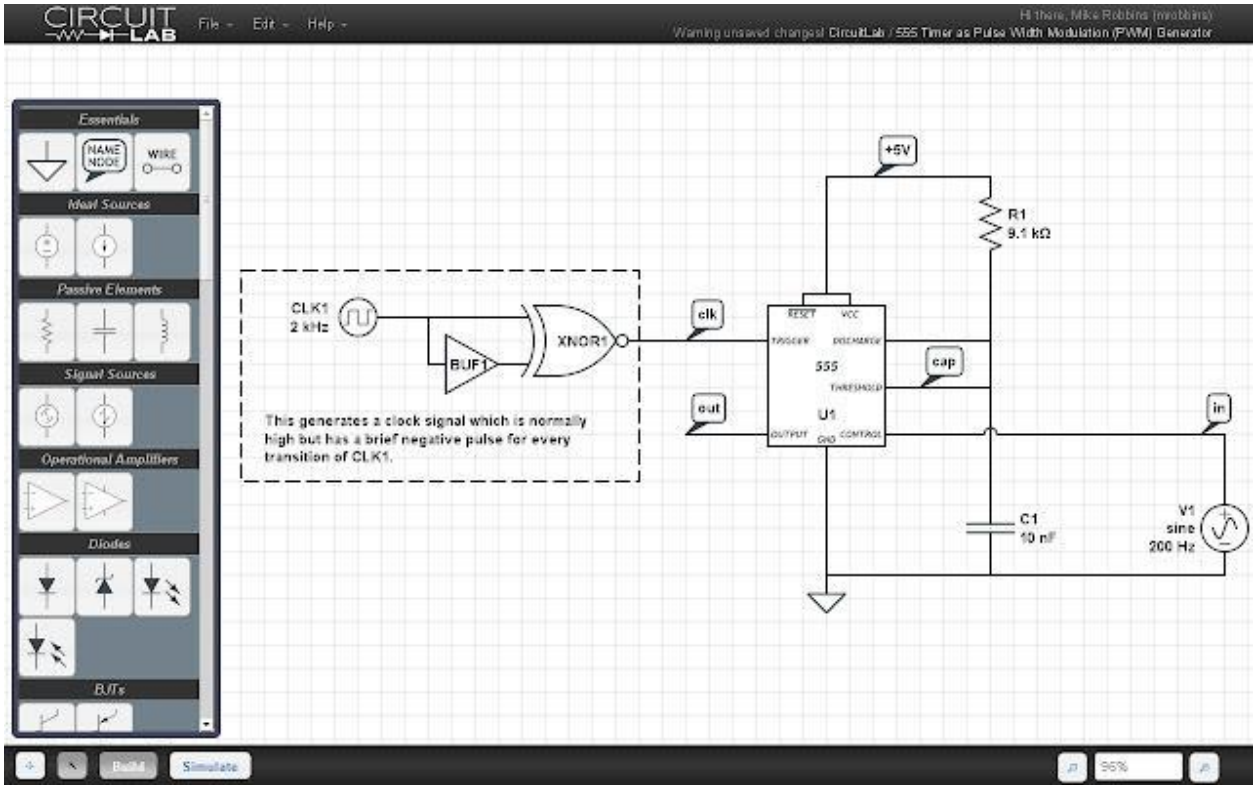


Рис. 3 Загальний вигляд CircuitLab

1.5 Постановка задачі

Розглянувши загальні відомості а також існуючі рішення можемо відповідні вимоги до майбутньої системи:

1. Інтерфейс користувача схожий на існуючі рішення – ця вимога необхідна для того щоб користувачі мали можливість безболісно змінювати програмне забезпечення.
2. Рішення повинне бути безкоштовним.
3. Кросплатформність – рішення повинно бути кросплатформне оскільки різні користувачі використовують різні операційні системи.

Відповідно до вимог – найкращим рішенням буде створення вебдодатка, оскільки таке рішення буде кросплатформнним, безкоштовним а також симуляція буде на сервері, а не на комп’ютері користувача, що дозволить зменшити навантаження на систему користувача.

2 РЕАЛІЗАЦІЯ ВЕБДОДАТКА

2.1 Загальна структура застосування

Веб-застосування складається з 3 різних частин, таких як – клієнтська частина, серверна частина та база даних. Відповідно використовується клієнт-серверна архітектура.

Архітектура **клієнт-сервер** відноситься до систем, які розміщують, надають та управляють більшістю ресурсів та послуг, які запитує клієнт. У цій моделі всі запити та послуги надаються через мережу, і вона також називається мережевою обчислювальною моделлю або мережею клієнт-сервер.

Клієнт-серверна архітектура, яку також називають клієнт-серверною моделлю, є мережевим додатком, який розподіляє завдання та робочі навантаження між клієнтами та серверами, які знаходяться в одній системі або пов'язані комп'ютерною мережею.

Дана архітектура, як правило, включає робочі станції, ПК або інші пристрої декількох користувачів, підключені до центрального сервера через підключення до Інтернету або іншої мережі. Клієнт надсилає запит на отримання даних, а сервер приймає та обробляє запит, надсилаючи пакети даних назад користувачеві, який їх потребує.

Коротко підсумуємо:

- Спочатку клієнт надсилає свій запит через мережевий пристрій
- Потім мережевий сервер приймає і обробляє запит користувача
- Нарешті, сервер відповідає клієнту

Для взаємодії сервера та клієнта використовується **REST API** (також відомий як RESTful API) - це інтерфейс прикладного програмування (API або

веб-API), який відповідає обмеженням архітектурного стилю REST і дозволяє взаємодіяти з RESTful веб-сервісами. REST розшифровується як representational state transfer (репрезентативна передача стану) і був створений вченим Роєм Філдінгом (Roy Fielding). REST забезпечує набір архітектурних обмежень, які, при застосуванні в цілому, підкреслюють масштабованість взаємодії компонентів, загальність інтерфейсів, незалежне розгортання компонентів, а також проміжні компоненти для зменшення затримки взаємодії, забезпечення безпеки та інкапсуляції застарілих систем[10].

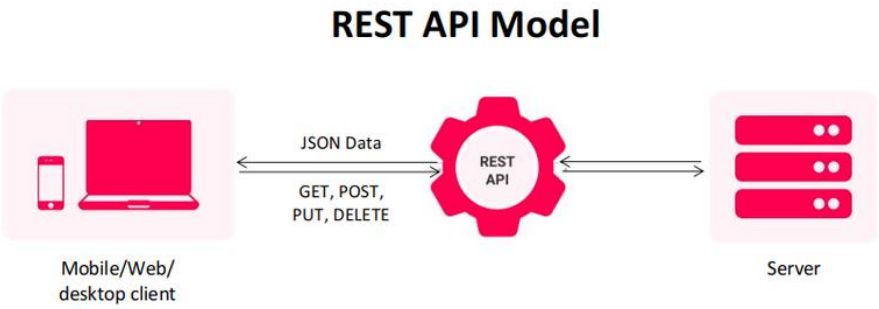


Рис. 4 REST API

2.2 Серверна частина

Для процесу розробки серверної частини використовувалась платформа для розробки Docker. Docker - це відкрите програмне забезпечення, яке надає можливість запуску та управління контейнерами[11]. Контейнер - це стандартизований пакет програмного забезпечення, який містить все необхідне для запуску додатка, включаючи код, залежності, конфігурацію та інші ресурси. Відповідно на рисунку 5 зображено вміст використаного Dockerfile.

```
FROM python:3.10.6

ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

ARG BUILD_ENVIRONMENT=production
ENV BUILD_ENV ${BUILD_ENVIRONMENT}
ARG APP_HOME=/code
WORKDIR ${APP_HOME}

COPY . ${APP_HOME}
RUN pip install -r requirements.txt
```

Рис 5. Dockerfile для Django застосунку

Для керування та запуску Docker контейнерами було використано Docker Compose. За допомогою Docker Compose можна описати всі контейнери, що потрібні для роботи додатка, їх залежності та конфігурацію. Кожен контейнер може бути запущений окремо, або разом з іншими контейнерами у складі багатоконтейнерного додатка.

Docker Compose дозволяє налаштувати середовище розробки та тестування додатків, що допомагає забезпечити однаковість усіх середовищ та уникнути можливих проблем залежностей та конфігурації. Крім того, Docker Compose дозволяє запускати та зупиняти багатоконтейнерні додатки з однієї команди, що спрощує процес управління додатками. Вміст файлу docker-compose.yml розміщено в додатку Б.

Для розробки серверної частини використовувалась мова програмування Python, а саме - фреймворк Django.

Django є найбільш популярним фреймворком для веб-розробки на Python, що дозволяє швидко створювати безпечні та легкі в обслуговуванні веб-сайти. Внаслідок того, що Django був розроблений досвідченими розробниками, він забезпечує велику частину функціоналу для веб-розробки, тому розробники можуть зосередитися на написанні додатка, не витрачаючи час на створення основних функцій[12]. Django є безкоштовним та з відкритим вихідним кодом, має живу та активну спільноту, хорошу документацію та багато варіантів безкоштовної та платної підтримки. Розробка з використанням Django є дуже швидкою завдяки використанню патерну MVT.

MVT, що означає Model View Template, є патерном проєктування програмного забезпечення, який складається з трьох ключових компонентів: Моделі, Представлення та Шаблону.

Моделі(Model) забезпечують доступ до бази даних та обробку даних на цьому рівні.

Шаблон(Template) відповідає за представлення користувацького інтерфейсу та повністю обробляє його частину.

Представлення(View) використовується для виконання бізнес-логіки та взаємодії з моделлю для передачі даних та обробки шаблону.

Django було обрано як фреймовк для серверної частини додатка саме через велику кількість вбудований функцій, а саме:

- ORM
- Вбудований інтерфейс адміністратора
- Диспетчер URL, на основі регулярних виразів
- Система хешування

- Бібліотека для роботи з формами (наслідування, побудова форм за існуючою моделлю БД)

Серед недоліків Django можна виділити те що він не працює з REST API, але це можна виправити за допомогою біблотеки Django Rest Framework(DRF)

Django Rest Framework - це потужний і гнучкий інструментарій для побудови Web API.

Він має такі основні особливості:

- Стандартизація: DRF надає стандартизовані підходи до розробки RESTful API, що дозволяє розробникам зосередитися на функціональності своїх додатків.
- Підтримка форматів: DRF підтримує різні формати даних, такі як JSON, XML, YAML, HTML та інші.
- Аутентифікація та авторизація: DRF надає вбудовану підтримку авторизації та аутентифікації, включаючи використання токенів, сесій та OAuth.
- Серіалізація даних: DRF надає інструменти для серіалізації та десеріалізації даних, що дозволяє передавати дані у вигляді JSON, XML та інших форматах.
- Пагінація: DRF надає інструменти для створення сторінкованого виведення даних, що дозволяє зменшити завантаження сервера.
- Підтримка CRUD-операцій: DRF надає готові інструменти для створення, читання, оновлення та видалення даних, що дозволяє швидко розробляти API.

- Модульність: DRF підтримує модульну архітектуру, що дозволяє розробникам використовувати тільки ті функції, які потрібні їм у конкретному проєкті.

Також окрім самого Django та бібліотек що його доповнюють в серверній частині було розміщено бізнес-логіку додатка, а саме зберігання та симуляцію розроблених схем.

Для зберігання користувачів та їх схем буде використовуватись база даних. Трьома найбільш широко використовуваними системами управління базами даних для Django є SQLite, MySQL та PostgreSQL. Спільнота Django та офіційна документація Django стверджують, що PostgreSQL є найкращою базою даних для веб-додатків Django. В офіційній документації Django про PostgreSQL сказано наступне: «Django забезпечує підтримку ряду типів даних, які будуть працювати тільки з PostgreSQL. Немає ніякої фундаментальної причини, чому (наприклад) не існує модуля contrib.mysql, за винятком того, що PostgreSQL має найбагатший набір функцій з підтримуваних баз даних, тому її користувачі мають найбільше переваг».

Фреймворк Django підтримує PostgreSQL 9.5 і вище. Для встановлення з'єднання між Django та базами даних PostgreSQL потрібен адаптер psycopg2 версії 2.5.4 або вище. Для виконання операцій з базами даних на PostgreSQL Django використовує модуль «django.contrib.postgres». З усіх доступних баз даних PostgreSQL надає найбагатший набір функцій і повну сумісність для розширеного функціоналу.

Відповідно було створено базу даних, основними таблицями є - таблиця користувача(User), та таблицю проєкта, на рисунку 6 зображена спрощена діаграма створеної бази даних. Повна діаграма таблиць бази даних розміщена в додатку А

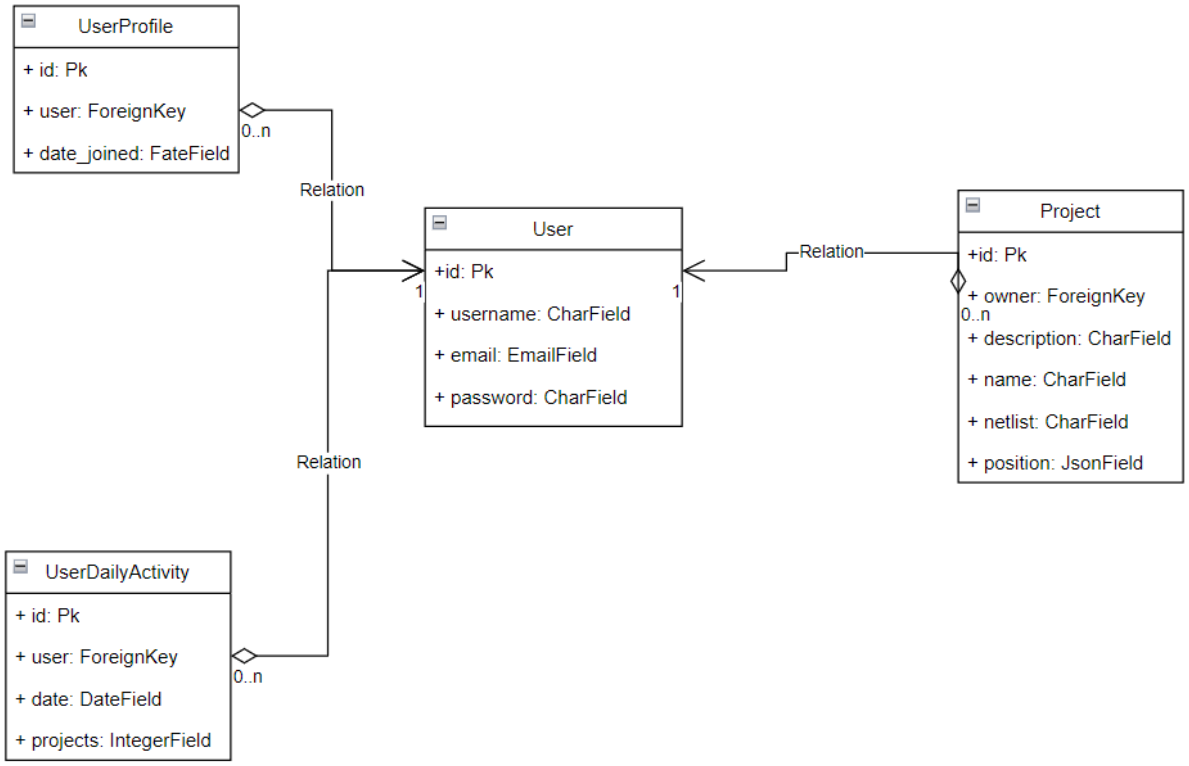


Рис. 6 Спрощена діаграма бази даних всього застосунку

Основними поля в таблиці Project є:

- Name - ім’я проєкту, являє собою поле типу CharField(nvarchar) з максимальною довжиною - 250 символів
- Description - опис проєкту, текстове поле(text), яке може бути пустим, не має обмежень по кількості символів
- Owner - власник проєкту, це поле є зовнішнім ключем(ForeignKey) від моделі User
- Positions - поле типу JsonField що містить в собі JSON в якому зберігається розташування всіх елементів на створеній користувачем схемі. Кожен елемент відповідно відповідно має свою назву та місце, що складається з 2 полів, це відповідно позиції по X та Y.

- Netlist - поле що містить в собі інформацію про список електричних компонентів і з'єднань між ними, що представляє логіку схеми електричної схеми для тестування апаратної частини електронних пристроїв. Інформація в даному полі представлена в формі SPICE, приклад представлений на рисунку 7.

```
pierce oscillator
c1 1 0 100e-12
c2 3 0 100e-12
c3 2 3 99.5e-15
c4 1 3 25e-12
l1 2 4 2.55e-3
r1 1 3 1e5
r2 3 5 2.2e3
r3 1 4 6.4
v1 5 0 12
Q1 3 1 0 NBJT
.MODEL NBJT NPN (BF=100)
.print tran  v(2) v(3)

.tran 1ns 1us  UIC
.ic v(2)=-10000.0 v(5)=12.0
```

Рис. 7 Приклад кола описаного за допомогою netlist SPICE

Для роботи з проєктами користувача було розроблено кінцеві точки за допомогою DRF ViewSet класів. ViewSet - це клас в Django Rest Framework, який надає швидкий і простий спосіб визначення логіки для взаємодії з різними джерелами даних в RESTful API. ViewSet забезпечує повну набір CRUD-операцій (створення, читання, оновлення та видалення) для даних моделі, і дозволяє легко додавати додаткові дії (actions) для взаємодії з моделлю. Він включає в себе методи для операцій на списку (list), одиничних

елементах (retrieve), створення (create), оновлення (update), видалення (destroy), а також для додаткових дій (actions), які можуть бути визначені користувачем. ViewSet дозволяє зручно працювати з RESTful API в Django, розподіляючи код для різних операцій в один клас, що зменшує кількість коду та забезпечує більш зрозумілу та структуровану архітектуру додатка.

Оскільки використовується REST підхід, то і відповідно формат даних в якому сервер отримує та відправляє результат буде JSON. JSON (або JavaScript Object Notation) є одним з найпопулярніших форматів даних для передачі даних між клієнтом та сервером в RESTful API. Переваги використання JSON для REST:

- Простота читання та розуміння: JSON має просту структуру та легко читається, що робить його зручним для розробників та API-клієнтів.
- Кросплатформова сумісність: JSON підтримується всіма сучасними мовами програмування та платформами, тому він є ідеальним форматом для забезпечення кросплатформної сумісності між клієнтом і сервером.
- Низька вага: JSON має малу вагу порівняно з іншими форматами даних, такими як XML або HTML, що знижує обсяг переданих даних та поліпшує продуктивність передачі даних в RESTful API.
- Простота парсингу: JSON може бути легко та швидко розпакований на стороні клієнта за допомогою вбудованих методів парсингу в більшості мов програмування, що дозволяє забезпечити швидку та ефективну передачу даних.

Всі кінцеві точки було додані в документацію Swagger (рисунок 8). Swagger (також відомий як OpenAPI) - це інструмент, який дозволяє описувати RESTful API у форматі JSON або YAML файлу. Він дозволяє розробникам та споживачам API отримувати інформацію про ресурси, методи, параметри та

відповіді без необхідності читати документацію у вигляді тексту або HTML-сторінок. Swagger дозволяє генерувати код для різних мов програмування, що спрощує процес розробки та інтеграції API в різні додатки. Крім того, він може бути використаний для автоматичної генерації документації API, що полегшує роботу з ним для розробників та споживачів.

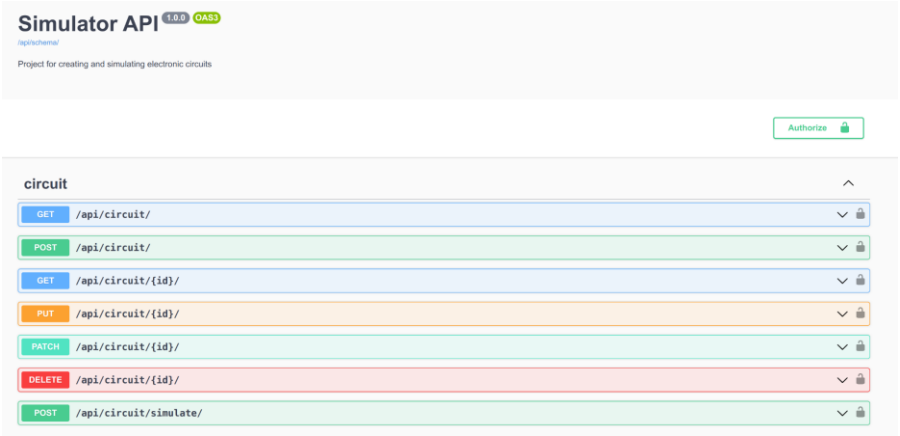


Рис. 8 Документація кінцевих точок за допомогою Swagger

Для створення нових ресурсів на сервері та в базі даних використовується метод POST. Оскільки для взаємодії клієнтської частини та серверної використовується JSON формат, а саме коло зберігається в форматі SPICE Netlist було реалізовано алгоритм для конвертації JSON в Netlist. На рисунку 8 наведений приклад такої конвертації, відповідно в JSON об’єкті містяться 2 ключі верхнього рівня - “components” та “connections”. Сам алгоритм міститься в додатку Б.

Рис. 9 Приклад конвертації JSON в Netlist

Після конвертації відбувається валідація створеного Netlist, для цього було використано бібліотеку PySpice, яка має вбудований синтаксичний аналізатор SPICE Netlist. Також було покращено алгоритм валідації, шляхом використання генераторів, що зменшило використання оперативної пам’яті сервера**.**

PySpice - це модуль Python з відкритим вихідним кодом, який надає інтерфейс Python до симуляторів схем Ngspice та Xyce. Відповідно для функціювання PySpice потрібний додатковий модуль Ngspice.

Основні переваги PySpice:

- підтримка симуляторів схем Ngspice і Xyce
- підтримка платформ Linux, Windows і MacOS
- наявний синтаксичний аналізатор SPICE netlist

- наявний орієнтований об'єктний API для визначення схеми

Після конвертації та валідації відбувається симуляція електричного кола, створений Netlist передається в симулятор, для симуляції використовується Ngspice.

Для обробки отриманих в результаті симуляції результатів використовується також бібліотека PySpice, яка напряму співпрацює з Ngspice. Після цього отриманий результат повертається у відповіді клієнту.

Але такий спосіб симуляції не підходить для стандартних REST API запитів через протокол HTTP, через те що HTTP не підтримує обмін між браузером та сервером в режимі реального часу, тому був використаний протокол WebSocket.

WebSocket – протокол напряму призначений для обміну даними між сервером на браузером (клієнтською частиною) в режим реального часу. На щастя в сучасному світі всі браузери підтримують даний протокол, тому проблем з його підтримкою не повинно бути.

За роботу Django з WebSocket відповідає бібліотека **Django Channels.** Channels - це проєкт, який бере Django і розширює його можливості за межі HTTP - для роботи з WebSockets, протоколами чату, протоколами IoT тощо. Він побудований на специфікації Python під назвою ASGI. Базується на власній підтримці ASGI в Django.

У той час як Django все ще працює з традиційним HTTP, Channels дає можливість працювати з іншими з'єднаннями в синхронному або асинхронному стилі. При цьому зберігається синхронний і простий у використанні характер Django, що дозволяє вибирати спосіб написання коду - синхронно в стилі, подібному до представлень Django, повністю асинхронно, або суміш обох способів. Крім того, він забезпечує інтеграцію з системою

авторизації Django, системою сеансів та іншим, що робить його більш простим, ніж будь-коли, для розширення вашого проєкту, що працює лише на HTTP, на інші протоколи. Channels зовсім не сповільнюють роботу Django з HTTP запитами оскільки використовується інший протокол.

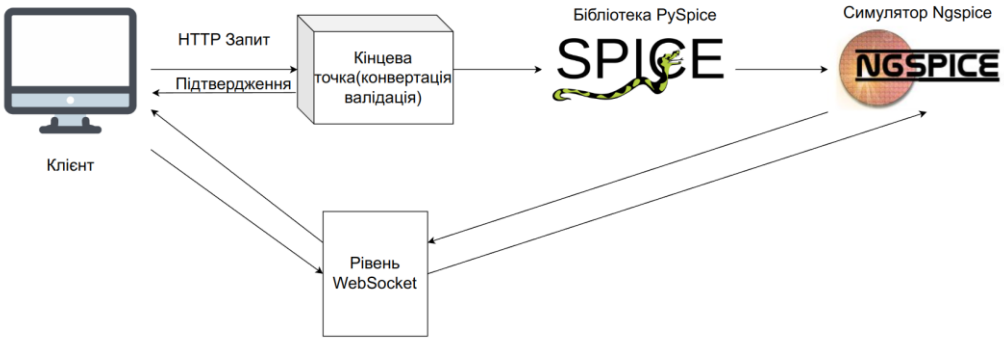


Рис. 10 Принцип роботи симуляції.

Відповідно при роботі з WebSocket користувач може в будь-який процес розпочати та закінчити симуляцію.

Для асинхронного запуску задач, таких як, відправлення повідомлень по електронній пошті, а також зберігання дій користувача використовується система Celery.

Celery - це система, призначена для створення і керування асинхронними завданнями в додатках Python. Завдяки Celery, розробники можуть створювати розподілені додатки, які здатні оброблювати великі обсяги роботи, такі як задачі, які вимагають багато часу для виконання або навіть запуск задач в заданий час.

Celery складається з трьох компонентів: брокера, воркера та клієнта. Брокер - це проміжна структура, яка дозволяє воркерам знаходити завдання для виконання та надсилати результати. Воркер - це процес або група

процесів, які забезпечують фактичне виконання завдань. Клієнт - це бібліотека, яка використовується для створення завдань та надсилання їх до брокера.

В якості брокера для Celery використовується база даних Redis. Redis (Remote Dictionary Server) - це відкрита, високошвидкісна, ключ-значення, розподілена система зберігання даних, яка працює в оперативній пам'яті та може зберігати дані на диску. Redis підтримує широкий набір структур даних, включаючи рядки, хеші, списки, множини та сортовані множини.

Redis є дуже швидким, оскільки весь доступ до даних відбувається в оперативній пам'яті, що дозволяє досягти дуже високої швидкості операцій з даними. Крім того, Redis підтримує реплікацію та кластеризацію, що дозволяє розподіляти дані на кілька серверів та забезпечувати високу доступність та масштабованість системи.

Redis використовується для різноманітних задач, включаючи кешування даних, сесії користувачів, розподілені локальні та глобальні лічильники, повідомлення та черги, зберігання конфігурацій та інших даних. Redis також використовується як брокер повідомлень для різноманітних систем, включаючи Celery.

2.3 Клієнтська частина

На стороні користувача, як вже було сказано буде використовуватись односторінкова архітектура.

Односторінкова архітектура(single-page architecture, SPA) - як випливає з назви, це одна сторінка, і вся інформація доступна на цій одній сторінці. Кожного разу, коли ви натискаєте на щось або прокручуєте

сторінку, змінюються лише кілька елементів, а решта залишаються незмінними. Моніторинг односторінкових додатків працює в браузері і не вимагає додаткового часу завантаження сторінки. Рисунок 2.1 схематично зображує різницю між звичайними веб-додатками та SPA

З технічної точки зору, односторінкові веб-додатки - це підхід до дизайну за допомогою HTML5 та сучасних браузерів, що дозволяє перенести логіку сторінки та інтерфейсу користувача з сервера на браузер за менший час. Він використовує JavaScript, HTML та CSS для динамічного завантаження ресурсів.

HTML - HyperText Markup Language (мова розмітки гіпертексту). Це стандартна мова розмітки для створення веб-сторінок. Вона дозволяє створювати та структурувати розділи, абзаци та посилання за допомогою елементів HTML (будівельних блоків веб-сторінки), таких як теги та атрибути[13].

CSS розшифровується як Cascading Style Sheets (каскадні таблиці стилів) і використовується для додавання стилю до веб-сторінки, визначаючи, як сайт відображається в браузері. CSS унікальна тим, що вона не створює нових елементів, як HTML або JavaScript. Замість цього, це мова, яка використовується для стилізації елементів HTML. CSS відповідає за стиль, розмір, позиціонування, колір та інше на веб-сайті. Це також те, що контролює, як стиль веб-сайту змінюється між десктопною та мобільною версіями.

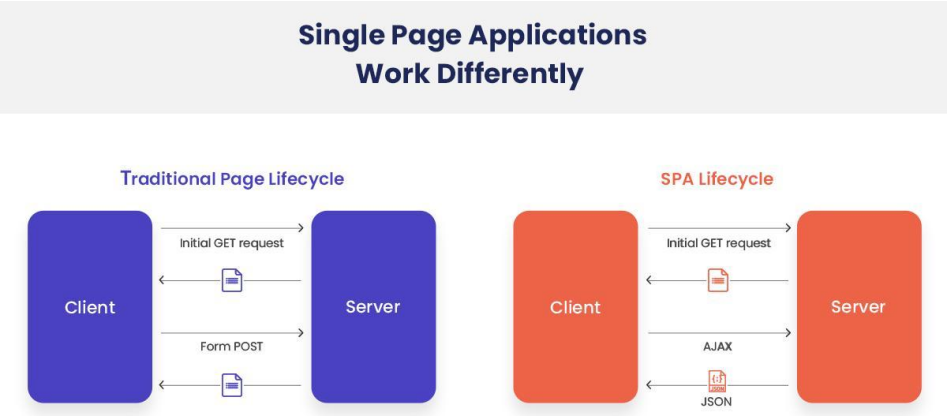


Рис. 12 Схема роботи SPA та звичайних веб-додатків

Основною бібліотекою для створення клієнтської частини було обрано бібліотеку React.

React - це JavaScript бібліотека, яка покликана спростити розробку візуальних інтерфейсів. Розроблена у Facebook та випущена у 2013 році, вона лежить в основі одних з найбільш широко використовуваних додатків, серед яких Facebook та Instagram, а також незліченна кількість інших додатків. Його основна мета - полегшити міркування про інтерфейс та його стан у будь-який момент часу інтерфейсу та його стану в будь-який момент часу, розділяючи інтерфейсу на набір компонентів[14].

Основною перевагою React над іншими популярними рішеннями є те що React це не фреймворк, а бібліотека. Внаслідок цього у розробників є можливість самим керувати потоком (flow) додатка. А також бібліотека є більш легкою в плані залежностей на відміну від фреймворку.

Для керування станом React додатка було обрано бібліотеку Redux. Redux - це просто сховище для зберігання стану змінних в додатку. Redux створює процес і процедури для взаємодії зі сховищем, щоб компоненти не просто оновлювали або читали сховище випадковим чином.

Redux як бібліотека може бути використана з будь-яким фреймовокром чи бібліотекою для збереження стану додатка, для кращої взаємодії React з Redux буде використовуватись бібліотека React Redux.

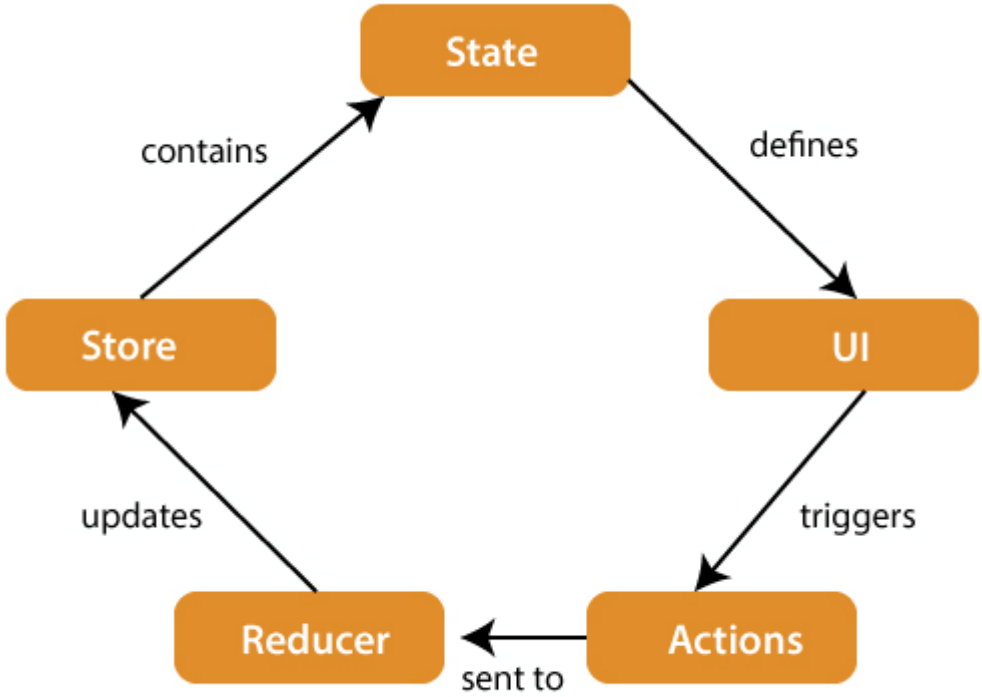


Рис. 13 – Взаємодія Redux компонентів

Робочу область було реалізовано в вигляді HTML елемента canvas, що має кілька переваг:

- Відображення графіки: Canvas дає можливість відображати складні графічні елементи, такі як діаграми, графіки, карти, анімації та інші, які складно реалізувати за допомогою звичайних HTML-елементів.
- Швидкість обробки: Canvas використовує апаратний прискорювач, що дозволяє відображати графіку з високою швидкістю та мінімізувати навантаження на процесор.

- Інтерактивність: Canvas може бути інтерактивним, тобто реагувати на дії користувача, такі як кліки миші, переміщення курсора тощо.
- Анімація: Canvas дозволяє створювати анімацію в реальному часі, що є дуже важливим для візуалізації даних та інтерактивних веб-додатків.
- Незалежність від розміру екрану: Canvas розміщується на сторінці, як звичайний HTML-елемент, і може бути налаштований таким чином, щоб автоматично адаптуватися до розміру екрану.

Для створення використовувалась бібліотека react-konva[15]. За допомогою компонентів “Stage” та “Layer” було створено canvas та різноманітні елементи, які можна додавати на полотно після натискання на відповідний елемент. Для переміщення елементів за допомогою drag-n-drop, було встановлено обробники подій миші лише при переході в режим drag-n-drop. Таким чином, було зменшено кількість обробників подій, що виконуються в процесі роботи з canvas та покращили загальну продуктивність. У загальному, завдяки використанню бібліотеки React та react-konva, було створено функціональний та ефективний механізм для роботи з canvas, що забезпечував зручну та інтуїтивну роботу з елементами робочої області.

Список елементів було реалізовано за допомогою опису кожного елементу в файлі components.json, на даний момент components.json має 20 компонентів, серед яких: транзистор, діод, конденсатор, джерела струму, логічні елементи(AND, OR, NAND, NOR, XOR). Робоча область разом зі списком елементів представлена на рисунку 13

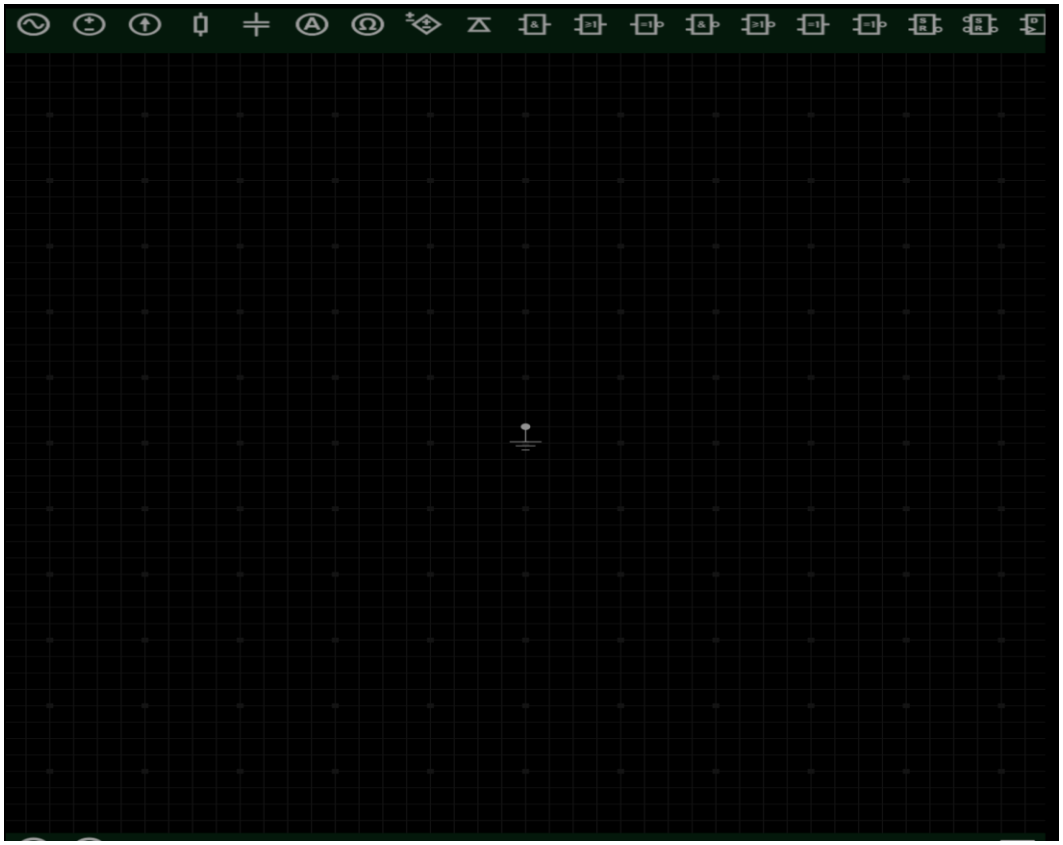


Рис. 13 Робоча область та список доступних елементів

На рисунку 14 зображено загальний інтерфейс розробленого додатка, він складається з описаних вище робочої області та списку елементів, списку створених електронних кіл користувачем, та опис відкритого електронного кола.

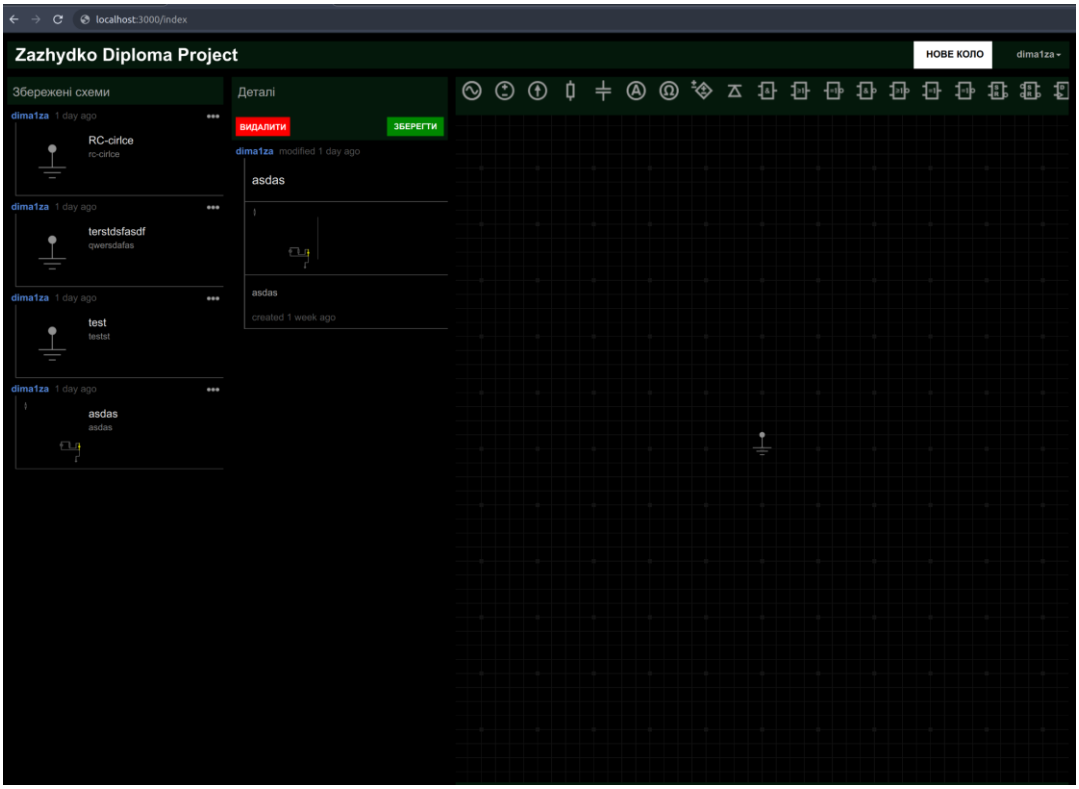


Рис. 14 Інтерфейс всього додатка

Для процесу симуляції було реалізовано використання протоколу WebSocket зі сторони клієнтського додатка. Було використано бібліотеку “react-use-websocket”, вона предоставляє пакет, що надає React хуки для спрощення використання веб-сокетів в додатках на React, окрім цього дозволяє використовувати веб-сокети в компонентах React за допомогою хуків, таких як useWebSocket, useSendJsonMessage та useJsonMessage.

- useWebSocket дозволяє підключитися до сервера веб-сокета та отримувати повідомлення з сервера в реальному часі.
- useSendJsonMessage дозволяє відправляти повідомлення на сервер у форматі JSON.
- useJsonMessage дозволяє обробляти отримані повідомлення в форматі JSON.

Відображення графіків було реалізовано за допомогою бібліотеки @nivo/line. Бібліотека містить велику кількість налаштувань і можливостей, що дозволяє створювати різноманітні графіки за допомогою невеликої кількості коду[16]. Деякі з можливостей бібліотеки:

- Підтримка декількох наборів даних на одному графіку
- Підтримка декількох типів лінійних графіків (з точками, без точок, з кривими, з відображенням даних у вигляді стовпчиків)
- Автоматичне масштабування осей
- Відображення даних у вигляді анімації
- Відображення легенди та міток на осі
- Підтримка інтерактивного взаємодії з графіком (наприклад, відображення значень при наведенні курсору на графік)
- Підтримка декількох типів маркерів для даних

Оскільки бібліотека підтримує роботу з canvas об’єктами, то її відображення здійснюється в тому самому canvas об’єкті що і робоча область, відповідний код наведено в додатку Г, на рисунку 15 зображено вигляд графіків в процесі симуляції.

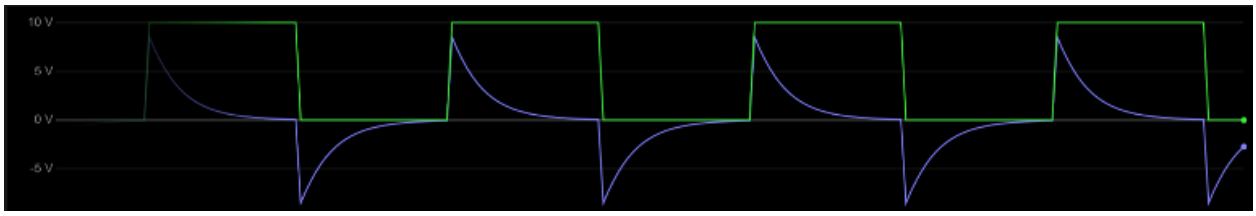


Рис. 15 Генерація графіків в процесі симуляції

3 АНАЛІЗ РОЗРОБЛЕНОГО ДОДАТКА

3.1 Моделювання схеми

Для демонстрації було промодельовано схему за допомогою створеного додатка, для порівняння було реалізовано таку саму схему в Proteus Design Suite. Тестування було виконано на електронному пристрої з таким набором характеристик:

- Процесор - Intel Core I5 7200u
- Кількість оперативної пам’яті - 8 Гб

Було створено схему RC-кола диференціального типу, для цього було використано 4 елементи: Джерело напруги, земля, резистор конденсатор. Значення елементів:

- Опір резистора - 1 kOm
- Місткість конденсатора - 100uF
- Джерело напруги - Pulse, мінімальна напруга - 0V, максимальна напруга - 10V

На рисунку 16 зображено створену мікросхему разом з процесом симуляції схеми.

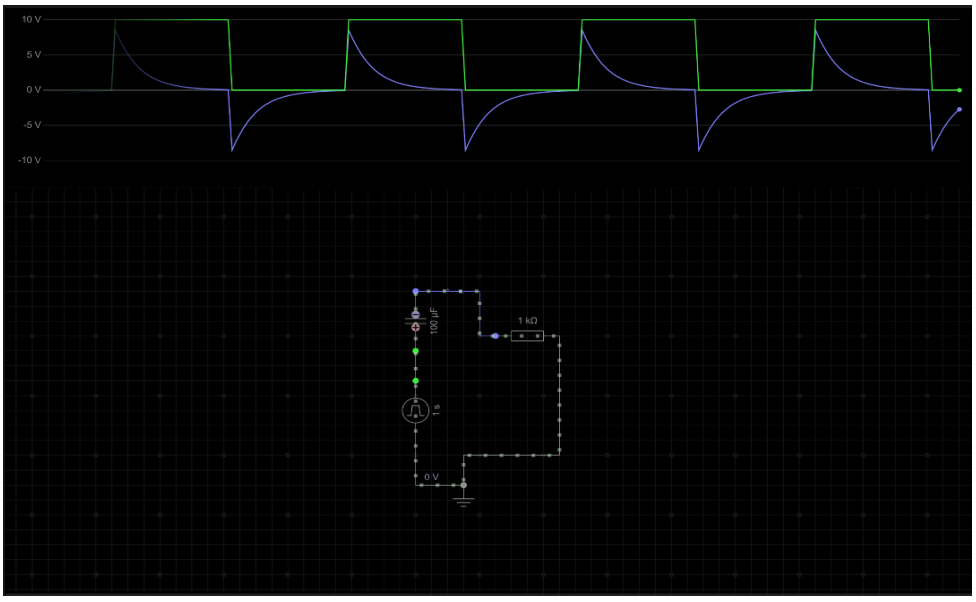


Рис 16. Процес симуляції створеної схеми

Шляхом використання Ngspice результат симуляції можна отримати без використання додаткових елементів оскільки за допомогою Ngspice є можливість вказати вхідний та вихідний сигнали.

Розглянемо збережену схему в адміністративній сторінці(рисунок 16). В ній міститься інформація про дату створення, дату останнього оновлення, назва та опис схеми, також є автоматично згенероване JSON поле з даними про положення елементів на схемі, а також сам опис схеми за допомогою SPICE Netlist в текстовому полі.

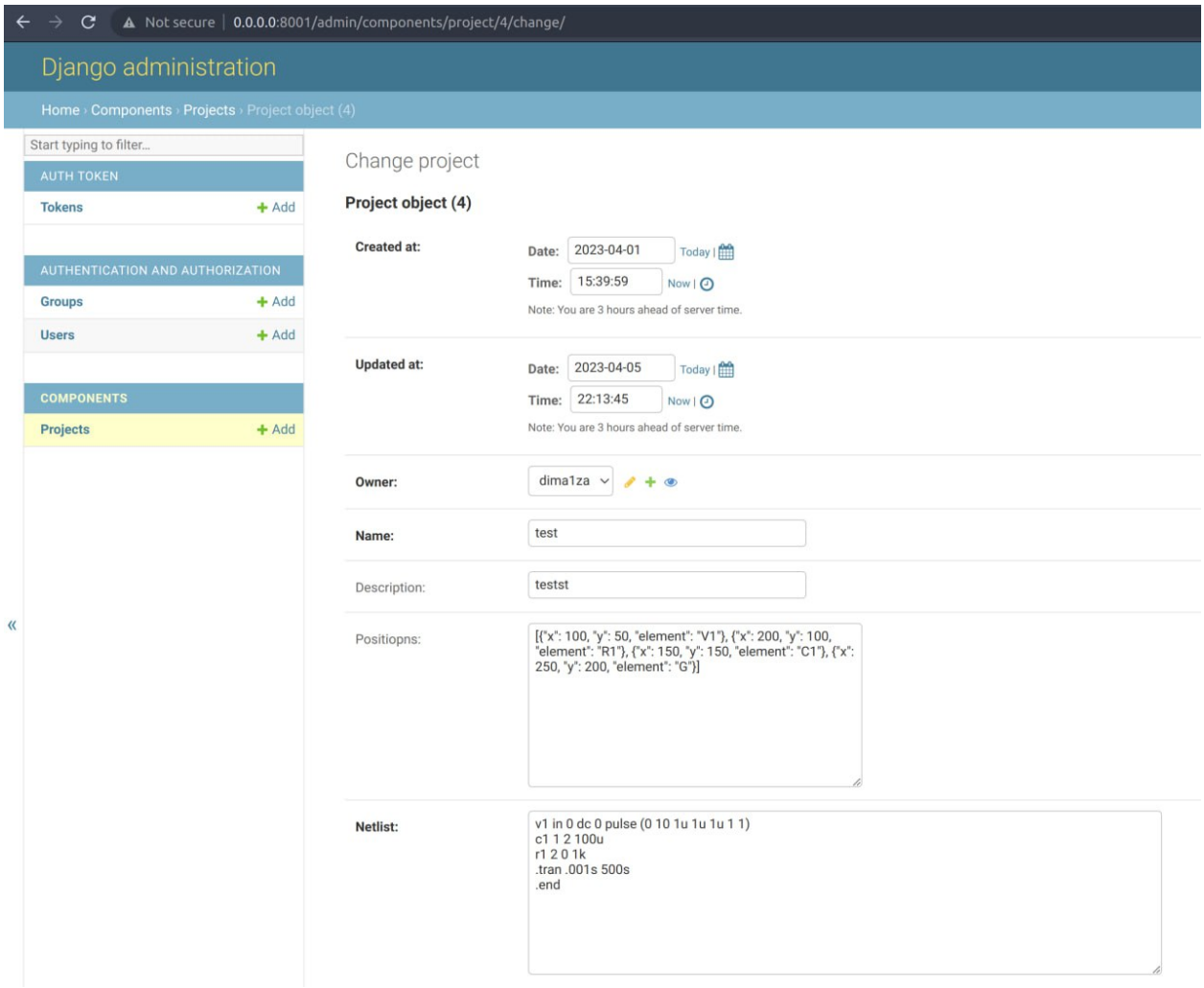


Рис. 17 Створена схема в адміністративній сторінці Django

3.2 Порівняння з існуючими рішеннями

Для порівняння з наявними рішення було реалізовано таку саму схему з такими самими характеристиками в застосуванні Proteus Design Suite. Для демонстрації результатів симуляції було використано Digital Oscilloscope, цьому відповідає рисунок 17.

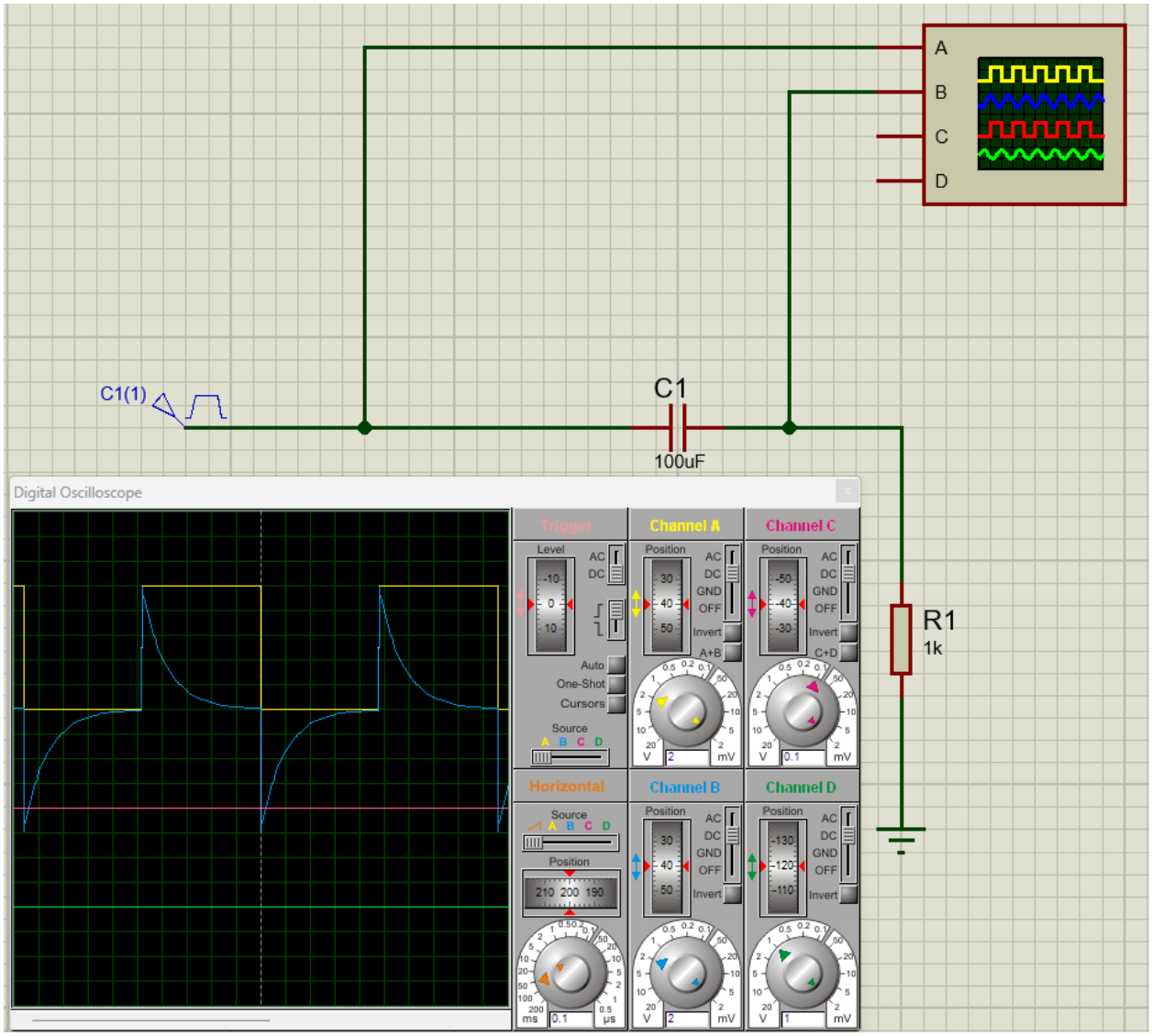


Рис. 18 Симуляція RC-кола за допомогою Proteus

Порівнюючи графіки вхідного та вихідного фронту можемо сказати що результат є однаковим, оскільки Proteus також заснований на SPICE.

Порівнюючи використаних ресурсів комп’ютера основними показниками були використання ЦП у відсотках оскільки в інших одиницях виміряти це неможливо та використання оперативної пам’яті

Вимірювання навантаження створеного застосунку проводилось у 4 етапи:

- Вимірювання навантаження при запуску стартової сторінки браузера;
- Вимірювання навантаження при відкритті вебсайту з останньою зміненою схемою (у моєму випадку - RC-коло);
- Вимірювання навантаження під час запуску симуляції через протокол Websocket, коли навантаження буде максимальним;
- Вимірювання навантаження після 2 хвилин симуляції.

На рисунку 19 зображено результати вимірювання всіх етапів.

Стартова сторінка браузера Firefox

Process Name	User	% CPU	ID	Memory
firefox	dima	1,64	476801	137,4 MB

Відкритий вебсайт розробленого додатку

Process Name	User	% CPU	ID	Memory
firefox	dima	2,13	467605	227,7 MB

Запуск симуляції RC кола

Process Name	User	% CPU	ID	Memory
firefox	dima	5,22	467605	364,3 MB

2 хвилини симуляції

Process Name	User	% CPU	ID	Memory
firefox	dima	4,95	467605	395,4 MB

Рис. 19 Результати вимірювання створеного застосунку

Для вимірювання навантаження ПЗ Proteus на обчислювальний апарат користувача було використану цю саму схему, але в 3 етапи без етапу стартової сторінки. Відповідні результати вимірювання зображені на рисунку 20.

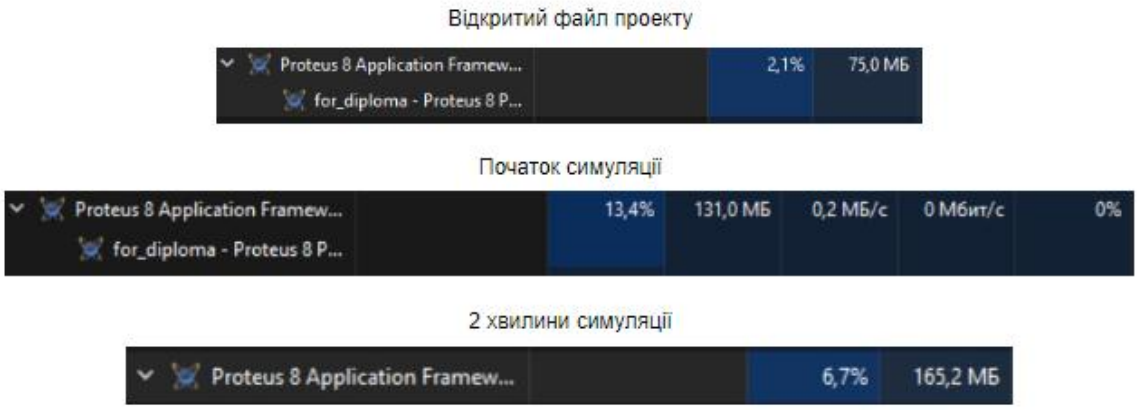


Рис. 20 Навантаження ПЗ Proteus на комп’ютер користувача

Відповідно до цих значень можемо створити таблицю-порівняння використаних ресурсів.

Процес	**Створене застосування**		**Proteus Design Suite**	
	ЦП, %	**ОЗУ, МБ**	**ЦП, %**	**ОЗУ, МБ**
Порожнє вікно	1,64	137,4	-	-
Без симуляції	2,13	227,7(90,3)	2.1	75
Початок симуляції	5,22	364,3(226,9)	13.4	131
Через 2 хвилини	4,95	395,4(258)	6.7	165
Середнє значення	3,485	281,2	7.4	123,6

Таблиця 2 Використання апаратних ресурсів комп’ютера

Оскільки для користування вебзастосунком відповідно необхідний браузер, то використання ОЗП створеним застосуванням є більшим через те, що сучасні браузери навіть при створенні нової порожньої сторінки можуть використовувати 100-200 Мб, а то й більше оперативної пам’яті то доцільним буде порівнювати використання оперативної пам’яті самого вікна з відкритим застосуванням, а не всю пам’ять використану браузером. Але навіть в цьому випадку, використання ОЗП створеним застосуванням є значно більшим ніж використання програмним забезпеченням Proteus, це можна пояснити використанням новітніх технологій, таких як React, за допомогою якого було реалізовано безшовний перехід від одного проєкт до іншого(SPA), а також використанням бібліотек, для відображення робочої зони(canvas) та графіків.

При цьому шляхом запуску процесу симуляції на стороні сервера можемо наглядати, що використання ресурсів процесора значно зменшилось, в порівнянні з ПЗ Proteus, відповідно створений застосунок є доцільно використовувати на системах зі слабким або застарілим процесором.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено та застосування, яке дозволяє користувачам моделювати та взаємодіяти з електронними мікросхемами.

Проведено аналіз технологій опису електронних мікросхем, що дозволило обрати найкраще рішення для розробки майбутнього застосунку.

Використано готовий алгоритм валідації Netlists типу SPICE та оптимізовано його шляхом використання генераторів, що дозволило зменшити використання оперативної пам’яті при процесі валідації.

В готовому застосунку було реалізовано електронну мікросхему за допомогою інтерфейсу користувача, для перевірки було реалізовано таку саму схему в ПЗ Proteus Design Suite, що дозволило переконатись в точності створеного застосунку.

Було порівняно використання апаратних ресурсів комп’ютера при розробці за допомогою створеного рішення та ПЗ Proteus. Створене рішення показало гірші результати використання ОЗП шляхом використання новітніх засобів розробки вебзастосунків, але при цьому внаслідок запуску симуляції на стороні сервера, значно зменшилось навантаження на центральний процесор.

ПЕРЕЛІК ДЖЕРЕЛ

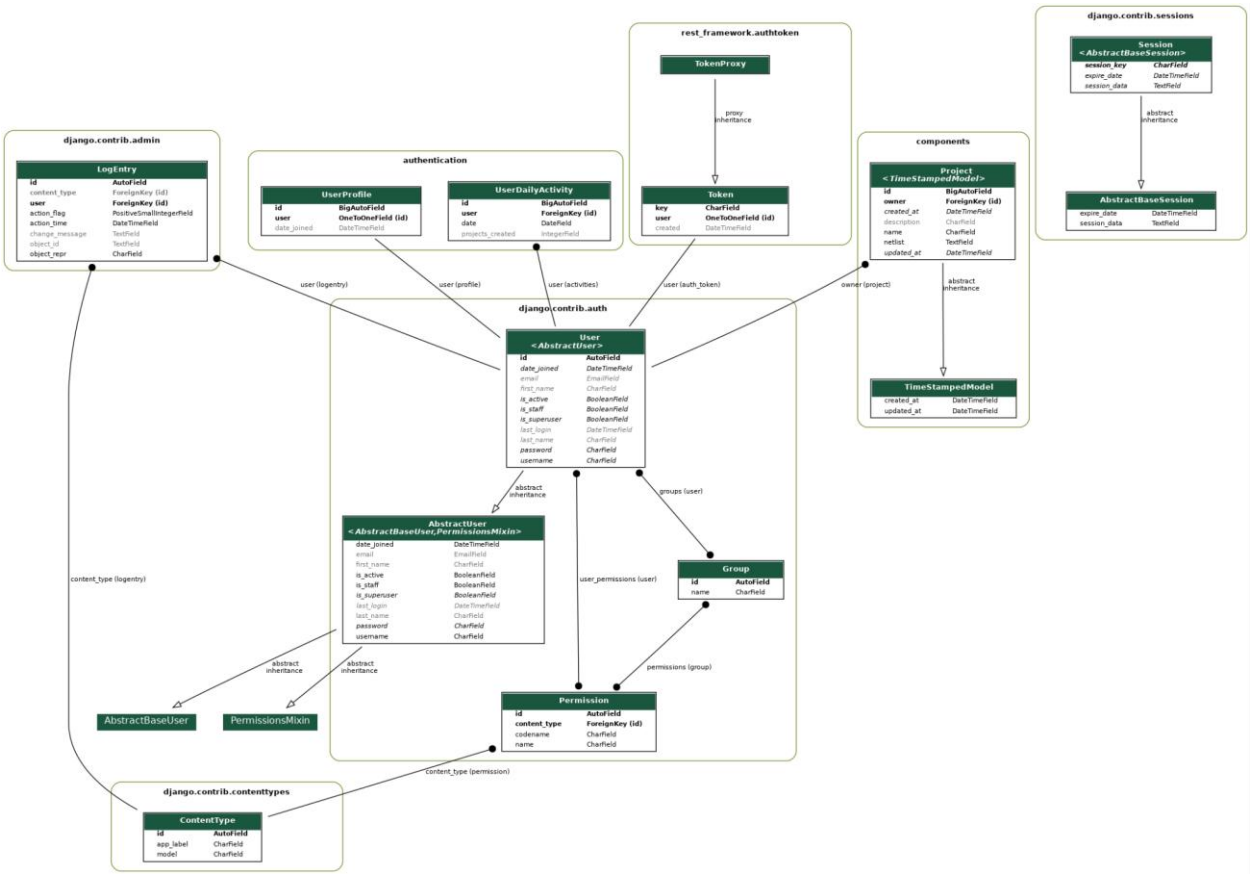
1. Kularatna N. (2008) - Electronic Circuit Design From Concept to Implementation. 2-3 сторінки
2. Mr. Arash Nourimand, Dr. Alison Olechowski (2020) - Prominence of Conceptual Design with Computer-Aided Design Tools for Junior and Senior Product Designers. 2 сторінка
3. Ngspice, the open source simulator [Електронний ресурс] – Режим доступу до ресурсу: https://ngspice.sourceforge.io/
4. What is the Difference Between Verilog and VHDL [Електронний ресурс] – Режим доступу до ресурсу: https://pediaa.com/what-is-the-difference-between-verilog-and-vhdl/
5. IBIS Simulation [Електронний ресурс] – Режим доступу до ресурсу: https://www.tina.com/ibis-simulation/
6. Electronic Design Interchange Format [Електронний ресурс] – Режим доступу до ресурсу: https://www.iue.tuwien.ac.at/phd/minixhofer/node53.html
7. Proteus Design Suite [Електронний ресурс] - Режим доступу до ресурсу: https://www.labcenter.com/
8. KiCad Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.kicad.org/6.0/en/getting_started_in_kicad/getting_started_in_kicad.html
9. CircuitLab TextBook Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://ultimateelectronicsbook.com/introduction/
10. Masse M. (2012) - REST API Design Rulebook. 4-5c
11. Docker overview [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/get-started/overview/

12. Django documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.djangoproject.com/
13. What Is HTML? Hypertext Markup Language Basics Explained [Електронний ресурс] – Режим доступу до ресурсу: https://www.hostinger.com/tutorials/what-is-html
14. Banks A. Porcello E. (2020) - Learning React Second Edition Modern Patterns for Developing React Apps. 22-23c
15. Getting started with react and canvas via Konva [Електронний ресурс] – Режим доступу до ресурсу: https://konvajs.org/docs/react/index.html
16. Nivo About [Електронний ресурс] – Режим доступу до ресурсу: https://nivo.rocks/about/

ДОДАТКИ

Додаток А

Діаграма бази даних всього застосунку



Додаток Б

docker-compose.yml

```
version: '3.3'

services:


  db:

    image: postgres

    volumes:

      - postgres_data:/var/lib/postgresql/data:Z

    environment:

      - POSTGRES_DB=postgres

      - POSTGRES_USER=postgres

      - POSTGRES_PASSWORD=postgres


  web:

    build:

      context: .

      dockerfile: Dockerfile

    command: bash -c "/code/scripts/start_server.sh"

    volumes:

      - .:/code
```

```
    ports:
      - "8001:8001"
    restart: on-failure
    environment:
      - POSTGRES_NAME=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_HOST=db
    depends_on:
      - db


  celery:
    build:
      context: .
    command: nice -12 celery -A sparks worker -E -l info -Q extractors,url_processors,summary_processors,questions_answers --autoscale=10,0
    volumes:
      - .:/code:z
    restart: on-failure
    depends_on:
      - db
```

```
    - redis
  environment:
    - REDIS_HOST=redis
    - POSTGRES_DB=${POSTGRES_DB}
    - POSTGRES_USER=${POSTGRES_USER}
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    - POSTGRES_HOST=db

celerybeat:
  build:
    context: .
  command: celery -A sparks beat -l INFO
  volumes:
    - .:/code:z
    - node_modules:/code/node_modules
  restart: on-failure
  depends_on:
    - db
    - redis
  environment:
    - REDIS_HOST=redis
```

```
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_HOST=db
volumes:
  postgres_data:
```

Додаток В

```
def json_to_netlist_converter(data):
netlist = ''
# Write the netlist header
netlist += "* Circuit: " + data["name"] + "\n"
netlist += "* Description: " + data["description"] + "\n"

# Write the component information
for component in data["components"]:
    netlist += component["type"] + " " + component["name"] + " "
    for param in component["parameters"]:
        netlist += param + "=" + str(component["parameters"][param]) + " "
    netlist += "\n"

# Write the connection information
for connection in data["connections"]:
    for i in range(100):
        if connection.get(f"node{i + 1}"):
            netlist += (
                connection[f"node{i}"] + " " + connection[
```

```
                f"node{i}"] + " " +
                    connection["type"] + "\n")
        else:
            break

return netlist
```

Додаток Г

```
import { ResponsiveLine } from '@nivo/line'

const MyResponsiveLine = ({ data /* see data tab */ }) => (
    <ResponsiveLine
        data={data}
        margin={{ top: 50, right: 160, bottom: 50, left: 60 }}
        xScale={{ type: 'linear' }}
        yScale={{ type: 'linear', stacked: true, min: 0, max: 2500 }}
        yFormat=" >-.2f"
        curve="monotoneX"
        axisTop={null}
        axisRight={{
            tickValues: [
                0,
                500,
                1000,
                1500,
                2000,
                2500
            ],
```

```
    tickSize: 5,
    tickPadding: 5,
    tickRotation: 0,
    format: '.2s',
    legend: '',
    legendOffset: 0
}}
axisBottom={{
    tickValues: [
        0,
        20,
        40,
        60,
        80,
        100,
        120
    ],
    tickSize: 5,
    tickPadding: 5,
    tickRotation: 0,
    format: '.2f',
```

```
    legend: 'price',
    legendOffset: 36,
    legendPosition: 'middle'
}}
axisLeft={{
    tickValues: [
        0,
        500,
        1000,
        1500,
        2000,
        2500
    ],
    tickSize: 5,
    tickPadding: 5,
    tickRotation: 0,
    format: '.2s',
    legend: 'volume',
    legendOffset: -40,
    legendPosition: 'middle'
}}
```

```
enableGridX={false}
colors={{ scheme: 'spectral' }}
lineWidth={1}
enablePoints={false}
pointSize={4}
pointColor={{ theme: 'background' }}
pointBorderWidth={1}
pointBorderColor={{ from: 'serieColor' }}
pointLabelYOffset={-12}
useMesh={true}
gridXValues={[ 0, 20, 40, 60, 80, 100, 120 ]}
gridYValues={[ 0, 500, 1000, 1500, 2000, 2500 ]}
legends={[
    {
        anchor: 'bottom-right',
        direction: 'column',
        justify: false,
        translateX: 140,
        translateY: 0,
        itemsSpacing: 2,
        itemDirection: 'left-to-right',
```

```
            itemWidth: 80,
            itemHeight: 12,
            itemOpacity: 0.75,
            symbolSize: 12,
            symbolShape: 'circle',
            symbolBorderColor: 'rgba(0, 0, 0, .5)',
            effects: [
              {
                on: 'hover',
                style: {
                  itemBackground: 'rgba(0, 0, 0, .03)',
                  itemOpacity: 1
                }
              }
            ]
          }
        ]}
    />
)
```