UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto Fin de Grado en Ingeniería en Tecnologías de la Información

Herramienta para la enseñanza del Análisis de Circuitos a través de los teoremas de Thévenin y Norton

Autor: Marco Antonio Garzón Palos

Dirigido por: Carla Martín Villalba

Codirigido por: Alfonso Urquía Moraleda

Curso Académico: 2024-2025

Convocatoria: Diciembre - 2025

Herramienta para la enseñanza del Análisis de Circuitos a través de los teoremas de Thévenin y Norton

Proyecto Fin de Grado en Ingeniería en Tecnologías de la Información de Modalidad específica

Realizado por: Marco Antonio Garzón Palos

Dirigido por: Carla Martín Villalba

Codirigido por: Alfonso Urquía Moraleda

Agradecimientos

A mi padre, que siempre me apoyó y confió en mí y que, desde el cielo,
me transmitió la fuerza necesaria para levantarme una y otra vez.

A mi madre, que sin ella habría sido imposible
concluir este arduo camino.

A mi hermano. Sin él, simplemente, mi vida no tendría sentido.

Resumen

El análisis de circuitos eléctricos constituye una disciplina fundamental enmarcada dentro de la Física, concretamente, en la rama del electromagnetismo, y es un pilar básico en la formación de un ingeniero. Se trata de una materia de dificultad media-alta tan amplia que, a menudo, requiere una gran dedicación y horas de estudio para llegar a comprenderla en su totalidad.

La teoría de circuitos se apoya en los teoremas de Léon Charles Thévenin y Edward Lawry Norton que permiten reducir cualquier circuito lineal, por complejo que sea, a otro equivalente, mucho más sencillo y manejable, convirtiendo a estos teoremas en dos pilares básicos de esta disciplina.

A pesar de que en el mercado existen herramientas de simulación muy potentes, éstas están orientadas a un uso profesional con interfaces complejas y curvas de aprendizaje elevadas, convirtiéndolas en un nuevo desafío para estudiantes noveles. Para facilitar el estudio y superar esas barreras, sería deseable disponer de una herramienta orientada a mejorar la calidad y la forma en la que los alumnos estudian, aprenden y resuelven problemas de análisis de circuitos.

En el presente Trabajo de Fin de Grado se pretende cubrir esta necesidad educativa creando una solución software desarrollada bajo el nombre de **ThevenApp** que utilice sus teoremas para que se pueda analizar un circuito de corriente continua (DC) en régimen permanente (resistivo) y obtener su equivalente siendo una alternativa frente al software comercial. Además, ThevenApp integra una plataforma de *e-learning* para facilitar la interacción entre profesores y alumnos. Esta funcionalidad permite al docente poner a disposición del alumno todo tipo de contenido teórico, práctico y documentación de tal forma que el estudiante pueda realizar tareas que aporten valor a su aprendizaje.

La facilidad de uso de la aplicación, gracias a un diseño tipo *canvas* interactivo con funcionalidad de arrastrar y soltar componentes (*drag and drop*), la capacidad de realizar análisis complejos con un simple clic y la integración de una plataforma educativa convierten a este proyecto en una aportación significativa para el estudio de la Física y la Ingeniería.

Palabras clave: Análisis de circuitos, Thévenin, Norton, Java, JavaFX, Electricidad, Ley de Ohm, Leyes de Kirchhoff, Física, E-learning.

Abstract

Electrical circuit analysis constitutes a fundamental discipline framed within Physics, specifically in the branch of electromagnetism, and represents a basic pillar in an engineer's education. It is a subject of medium-high difficulty and such breadth that it often requires great dedication and hours of study to be fully understood.

Circuit theory relies on Léon Charles Thévenin's and Edward Lawry Norton's theorems, which allow reducing any linear circuit, however complex, to a much simpler and more manageable equivalent, making these theorems two basic pillars of this discipline.

Although powerful simulation tools exist in the market, these are oriented towards professional use, featuring complex interfaces and steep learning curves, which pose a new challenge for novice students. To facilitate study and overcome these barriers, it is desirable to have a tool aimed at improving the quality and the way students study, learn, and solve circuit analysis problems.

The present Bachelor's Thesis aims to cover this educational need by creating a software solution named **ThevenApp**, which uses these theorems to analyze direct current (DC) circuits in steady state (resistive) and obtain their equivalent, serving as an alternative to commercial software. Furthermore, ThevenApp integrates an *e-learning* platform to facilitate interaction between teachers and students. This functionality allows the instructor to provide the student with all kinds of theoretical and practical content and documentation, enabling the student to perform tasks that add value to their learning.

The application's ease of use, thanks to an interactive *canvas* design with *drag and drop* functionality, the ability to perform complex analyses with a single click, and the integration of an educational platform, make this project a significant contribution to the study of Physics and Engineering.

Keywords: Circuit Analysis, Thévenin, Norton, Java, JavaFX, Electricity, Ohm's Law, Kirchhoff's Laws, Physics, E-learning.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El análisis de circuitos eléctricos es una pieza fundamental en muchas ingenierías. Se erige como uno de los pilares esenciales en áreas como la electrónica, las telecomunicaciones, la ingeniería informática o las tecnologías de la información. La base para poder abordar sistemas complejos en estas disciplinas está en la comprensión de cómo interactúan los componentes básicos de un circuito eléctrico.

Se trata de una materia cuya amplitud y nivel de abstracción provoca en los estudiantes serias dificultades en la asimilación de algunas de las técnicas de reducción de circuitos y de algunos teoremas fundamentales, tales como los teoremas de Thévenin y Norton. Estos teoremas son cruciales para simplificar circuitos complejos en otros mucho más sencillos y manejables. Sin embargo, el uso y aplicación práctica puede resultar confusa para el estudiante.

De esta dificultad surge la necesidad de elaborar una herramienta de software diseñada para facilitar el aprendizaje, el uso y la aplicación de estos teoremas y conceptos a cualquier estudiante de física o ingeniería.

A pesar de que existen múltiples aplicaciones en el mercado dedicadas al análisis y simulación de circuitos (como SPICE o herramientas CAD profesionales), éstas suelen tener una curva de aprendizaje elevada y complejidad notable de forma que pueden convertirse en una dificultad añadida más que una ayuda para el estudiante en sus etapas iniciales.

El objetivo de este proyecto es cubrir esa necesidad y conseguir una herramienta sencilla, fácil e intuitiva que ayude al estudiante a aprender los conceptos básicos del análisis de circuitos, en concreto, a aplicar los teoremas de Thévenin y Norton sin la necesidad de invertir mucho tiempo en aprender a utilizar otras aplicaciones software profesionales más complejas. Esta idea de sencillez y potencial didáctico es la base de **ThevenApp**, nombre con el que se ha bautizado a la aplicación desarrollada en este proyecto.

1.2. Objetivos

La necesidad y la motivación de las que surge la idea de elaborar esta herramienta establecen, implícitamente, unas metas que esta debiera cumplir. Estas metas deben ser un punto de partida para la planificación de la aplicación y deben satisfacer las necesidades de los usuarios a los que va dirigida.

1.2.1. Objetivo General

El objetivo general de este proyecto, que da respuesta a la motivación expuesta en la sección 1.1, es el diseño y desarrollo de una herramienta software interactiva enfocada en fines educativos para la enseñanza y aprendizaje del análisis de circuitos. Esta aplicación permitirá fomentar el aprendizaje de los teoremas de Thévenin y Norton y su aplicación práctica mediante una interfaz gráfica interactiva, un motor de cálculo robusto y un sistema de *e-learning* que permita la gestión de contenidos y facilite a los profesores el seguimiento de los alumnos.

1.2.2. Objetivos Específicos

Para poder alcanzar el objetivo general, se han establecido los siguientes objetivos específicos concretos que se deben cumplir:

1.2.2.1. Desarrollo e Interfaz

- **Desarrollo de una interfaz gráfica de usuario (GUI) interactiva**: Desarrollar una interfaz gráfica intuitiva que permita diseñar circuitos eléctricos mediante la técnica de arrastrar y soltar (*drag and drop*), facilitando que el usuario se centre en la topología del diseño y abstrayendo los cálculos subyacentes.
- **Implementación de algoritmos de resolución**: Crear un motor de cálculo capaz de interpretar el esquema diseñado y aplicar los teoremas de Thévenin y Norton de forma automatizada e inmediata para analizar el circuito creado por el usuario y obtener su circuito equivalente.
- **Sistema de persistencia de datos**: Definir una estructura de datos que permita serializar los circuitos para su almacenamiento en un archivo local y posterior recuperación, garantizando la continuidad en el trabajo.

1.2.2.2. Plataforma Educativa y de Gestión de Contenidos

- **Creación de una plataforma didáctica**: Integrar el uso del simulador con una plataforma de contenidos interactiva para alumnos y profesores que facilite la enseñanza práctica y visual de estos teoremas.
- **Gestión de roles y progreso académico**: Desarrollar un sistema de base de datos que permita registrar usuarios con roles de profesor o estudiante, de forma que el docente

pueda realizar un seguimiento de los alumnos y asignar nuevos recursos didácticos, y estos puedan utilizarlos para avanzar en su formación.

1.2.2.3. Accesibilidad y Usabilidad

- **Accesibilidad**: Garantizar una herramienta accesible desde cualquier equipo y cualquier sistema operativo sin necesidad de realizar una instalación compleja para favorecer el uso en entornos educativos.
- **Usabilidad**: Garantizar una herramienta de uso sencillo e intuitivo que asegure una curva de aprendizaje mínima, de modo que el esfuerzo cognitivo del alumno se centre en el aprendizaje de los conceptos teóricos y prácticos puestos a su disposición.

1.2.3. Desglose de Tareas

Para garantizar el cumplimiento de los objetivos descritos, se ha definido el siguiente plan de trabajo que desglosa las actividades técnicas a realizar:

1. **Análisis y selección tecnológica**: Estudio de las diferentes posibilidades tecnológicas con el objetivo de seleccionar las adecuadas para este proyecto.
2. **Implementación del editor gráfico**: Desarrollo del lienzo de dibujo interactivo y la paleta de componentes con funcionalidad *drag and drop*.
3. **Desarrollo del motor de cálculo**: Programación de los algoritmos de análisis de circuitos y resolución de los sistemas de ecuaciones para la obtención de los equivalentes de Thévenin y Norton.
4. **Integración de la plataforma de *e-learning***: Implementación del sistema de registro y autenticación con control de roles (profesor/alumno) y secciones de contenido teórico, ejercicios y seguimiento del progreso realizado.
5. **Definición de la arquitectura de datos**: Definición de la estructura de la base de datos para la gestión de usuarios y roles, así como el formato de serialización (JSON) para el guardado de circuitos.
6. **Validación y pruebas**: Realización de pruebas unitarias de los algoritmos de cálculo comparando resultados con soluciones teóricas conocidas y pruebas de usabilidad de la interfaz.

1.3. Estructura de la memoria

Este documento se organiza de la siguiente manera:

- **Capítulo 1: Introducción.** Presentación del proyecto, la motivación, los objetivos y la estructura de la memoria.
- **Capítulo 2: Fundamentos teóricos.** Se describen los conceptos básicos del análisis de circuitos y los teoremas de Thévenin y Norton. Se explican también las diferentes tecnologías utilizadas en el diseño de la aplicación.

- **Capítulo 3: Análisis de requisitos.** Se especifican los requisitos funcionales y no funcionales del sistema. Contiene un diagrama de casos de uso.
- **Capítulo 4: Diseño de la aplicación.** Se da una descripción general de la aplicación, su diseño de contenido y el diseño de la interfaz de usuario.
- **Capítulo 5: Implementación.** Se describe como se ha llevado a cabo el desarrollo del software; algoritmos utilizados, base de datos y otros detalles de la implementación.
- **Capitulo 6: Pruebas y validación.** Se analizan los resultados obtenidos tras realizar pruebas de uso y validación del software.
- **Capítulo 7: Conclusiones y trabajos futuros.** Se exponen las conclusiones sobre el trabajo realizado y las posibilidades de trabajos futuros para la ampliación o mejora del software.
- **Bibliografía.**
- **Anexo I: Manual de usuario.** Contiene el manual de usuario de la aplicación.
- **Anexo II: Manual técnico de instalación y despliegue.** Contiene un manual técnico para instalación y despliegue de la aplicación.
- **Anexo III: Código fuente.** Contiene las instrucciones para acceder al repositorio en línea **GitHub** para descargar todo el código fuente de la aplicación.

Capítulo 2

Fundamentos teóricos

En este capítulo se presentan los conceptos teóricos fundamentales en los que se basa esta aplicación dedicada al análisis de circuitos. Concretamente, se hace una revisión de la teoría de circuitos, de los teoremas de Thévenin y Norton, que son los teoremas en los que se fundamenta la aplicación y del análisis nodal en circuitos eléctricos, que es una de las técnicas de análisis de circuitos que utiliza esta aplicación. Dado que el objetivo del proyecto es el aprendizaje y la aplicación práctica de estos teoremas, se hace una revisión de otras aplicaciones educativas y/o profesionales que están en el mercado.

2.1. Teoría de Circuitos

> *La electricidad es la propiedad fundamental de la materia que se manifiesta por la atracción o repulsión entre sus partes, originada por la existencia de electrones, con carga negativa, o protones, con carga positiva.*

Fuente: [1]

Para entender este concepto en su totalidad deberíamos empezar a hablar del átomo y los elementos que lo componen. De una forma muy resumida, un átomo se compone de un núcleo y una región que lo rodea que se conoce como nube de electrones. El núcleo a su vez se compone de dos tipos de partículas: los *protones*, con carga eléctrica positiva y los *neutrones* con carga eléctrica neutra; la nube de electrones, como su nombre indica, está compuesta de unas partículas denominadas *electrones* con carga eléctrica negativa. Un átomo que se encuentra en equilibrio eléctrico tiene el mismo número de protones y de electrones de forma que la carga del mismo es neutra. Sin embargo, los átomos tienden a intercambiar electrones con otros átomos creando lo que se conoce como *corriente eléctrica*.

Teniendo en cuenta esto, podemos definir un *circuito eléctrico* como un conjunto de componentes conectados entre sí de forma que circula una corriente eléctrica entre ellos (ver Figura 2.1). La *teoría de circuitos* es la herramienta matemática que permite calcular la *tensión* y la *corriente eléctrica* en un circuito.

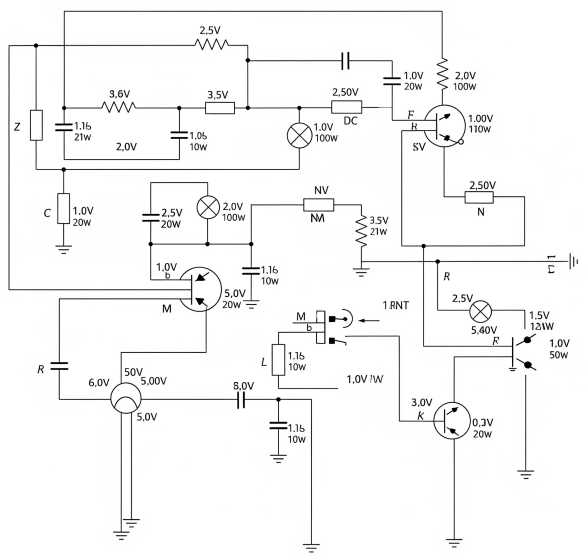


Figura 2.1: Ejemplo de circuito eléctrico.

2.1.1. Corriente eléctrica

Se define la **corriente eléctrica** ***i*** como la variación de carga eléctrica respecto del tiempo:

$$i = \frac{dq}{dt} \tag{2.1}$$

Se entiende, por tanto, la corriente eléctrica como el desplazamiento de carga eléctrica a través de un material conductor (ver Figura 2.2). Para que se produzca, debe haber un *campo eléctrico* que ejerza la fuerza necesaria sobre esas cargas y es generado por lo que se denomina *tensión eléctrica*. Cuando los electrones se desplazan constantemente en una única dirección - normalmente, desde el polo negativo (exceso de electrones) al polo positivo (deficiencia de electrones) - se denomina **corriente continua CC**; como ejemplo podríamos exponer las baterías o pilas. Cuando los electrones se desplazan alternativamente hacia ambas direcciones de forma periódica - se generan oscilaciones - se denomina **corriente alterna CA**; esta es, precisamente, la corriente que llega a nuestros hogares. En este proyecto, se usa el concepto de corriente continua, dejando el de corriente alterna para una posible ampliación.

La unidad de corriente eléctrica en el Sistema Internacional es el **amperio, A**.

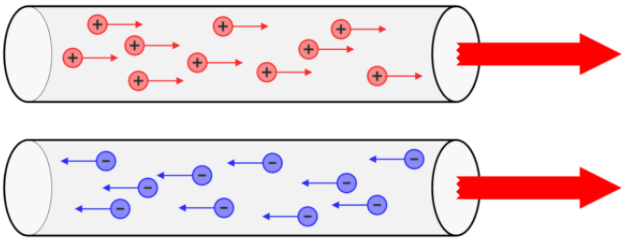


Figura 2.2: Corriente eléctrica [2].

2.1.2. Tensión eléctrica

Se define la **tensión eléctrica** u como la energía por unidad de carga necesaria para mover una carga eléctrica desde un punto a otro punto (ver Figura 2.3).

$$u = \frac{dW}{dq} \tag{2.2}$$

Una definición más apropiada en *teoría de circuitos* sería decir que la tensión eléctrica es la diferencia de *potencial eléctrico* existente entre dos puntos de un circuito de forma que se crea la fuerza necesaria para mover electrones de un punto a otro generando una *corriente eléctrica*.

La **teoría de circuitos** contiene técnicas y teoremas que permiten conocer la corriente eléctrica que circula en un punto de un circuito o a través de uno de los componentes del circuito y la tensión eléctrica existente entre dos puntos cualesquiera de un circuito eléctrico.

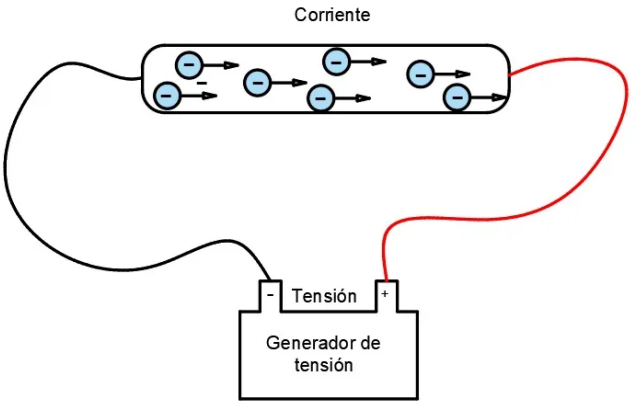


Figura 2.3: Generador de tensión [3].

2.1.3. Tipos de componentes

No todos los componentes de un circuito eléctrico actúan de la misma manera. En general, existen dos tipos de componentes en un circuito:

- **Componentes activos:** son los componentes que excitan el circuito, es decir, aportan energía al circuito.
  - **Fuente de tensión**: es un componente cuya utilidad es la de mantener una tensión constante entre sus terminales independientemente de la corriente que circule a través de él, es decir, suministra una tensión o voltaje al circuito. Un caso particular sería el de una **fuente de tensión dependiente** cuya tensión estará controlada por otra tensión o corriente existente en algún punto del circuito. La unidad en el Sistema Internacional de Unidades de tensión o voltaje es el **Voltio, V**.
  - **Fuente de corriente**: es un componente cuya utilidad es la de proporcionar un flujo de corriente eléctrica constante. Un caso particular es el de una **fuente de corriente dependiente** cuyo valor estará controlado por otra tensión o corriente existente en algún punto del circuito. La unidad en el Sistema Internacional de Unidades de corriente eléctrica es el **Amperio, A**.
- **Componentes pasivos**: son los componentes que, de una manera u otra, consumen la energía del circuito bien sea para disiparla, almacenarla o modificarla.
  - **Resistencia**: es un componente cuya función es limitar el flujo de corriente eléctrica disipando la energía en forma de calor. Su magnitud física es la **Resistencia** cuya unidad en el Sistema Internacional de Unidades es el **Ohmio,** Ω.
  - **Condensador**: (o capacitor) es un componente cuya función es la de almacenar temporalmente energía en forma de campo eléctrico y, así, suavizar fluctuaciones de voltaje en el circuito. En *corriente continua* actúa como un **cortocircuito**. Su magnitud física es la **Capacidad** o **Capacitancia** cuya unidad en el Sistema Internacional de Unidades es el **Faradio, F**.

- **Bobina**: (o inductor) es un componente cuya función es la de almacenar temporalmente energía en forma de campo magnético y, así, suavizar cambios de corriente. En *corriente continua* actúa como **circuito abierto**. Su magnitud física es la **Inductancia** cuya unidad en el Sistema Internacional de Unidades es el **Henrio, H**.

En la Tabla 2.1 se puede ver los diferentes componentes que forman parte de un circuito eléctrico y la magnitud física que representan.

Tabla 2.1: Tipos de componentes de un circuito.

Tipo	**Componente**	**Magnitud**	**Unidad**	**Símbolo**
Activos	Fuente de Tensión	Tensión (Voltaje)	Voltio, V	
	Fuente de Corriente	Corriente	Amperio, A	
Pasivos	Resistencia	Resistencia	Ohmio, Ω	
	Condensador	Capacidad	Faradio, F	
	Bobina (Inductor)	Inductancia	Henrio, H	

2.2. Ley de Ohm

> *"La intensidad de la corriente eléctrica que circula por un conductor eléctrico es directamente proporcional a la diferencia de potencial e inversamente proporcional a la resistencia del mismo."* - **Georg Simon Ohm**.

Fuente: [4]

Tal como dice la definición, la Ley de Ohm establece que la corriente eléctrica es directamente proporcional a la tensión o voltaje e inversamente proporcional a la resistencia. Matemáticamente se podría expresar de la siguiente forma:

$$V = I \cdot R \tag{2.3}$$

Esta ley es fundamental en el estudio de los circuitos eléctricos al relacionar las dos magnitudes de estudio de la teoría de circuitos: la *tensión* y la *corriente*.

2.3. Leyes de Kirchhoff

Las Leyes de Kirchhoff son dos principios fundamentales en el análisis de circuitos. Se basan en la *Ley de Conservación de la Energía* y fueron formuladas por **Gustav Kirchhoff** [4].

2.3.1. Ley de Corrientes de Kirchhoff (LCK)

La Ley de Corrientes de Kirchhoff (LCK) establece que la suma algebraica de las corrientes que confluyen en un nodo es cero o, dicho de otra manera, la suma de las corrientes que entran en un nudo es exactamente igual a la suma de las corrientes que salen.

Matemáticamente, la LCK se podría expresar de la siguiente manera sobre un nudo:

$$\sum_{k=1}^{n} I_k = 0 \tag{2.4}$$

2.3.2. Ley de Voltajes de Kirchhoff (LVK)

La Ley de Voltajes de Kirchhoff (LVK) establece que la suma algebraica de las tensiones en una malla es cero o, dicho de otra forma, las caídas de tensión dentro de una malla es igual a la tensión suministrada.

Matemáticamente, la LVK se podría expresar de la siguiente manera sobre una malla:

$$\sum_{k=1}^{n} V_k = 0 \tag{2.5}$$

2.4. Teorema de Thévenin

El **Teorema de Thévenin** establece que cualquier circuito eléctrico lineal, por complejo que sea, puede reducirse a un circuito equivalente con una fuente de tensión en serie con una resistencia. A la tensión que suministra esta

fuente se le denomina **Tensión o Voltaje de Thévenin, Vth** y a la resistencia equivalente se le denomina **Resistencia Thévenin, Rth**. La resistencia o, en general, los componentes que permanecen del circuito original y no forman parte del equivalente sino que se conectan a él, es la resistencia o circuito de **carga**.

Para obtener la tensión Thévenin, *Vth*, entre dos puntos del circuito A y B, se realizan los siguientes pasos:

1. Se desconecta el circuito de carga dejando abiertos los terminales A y B.
2. Se calcula la tensión entre estos dos terminales (tensión de circuito abierto) utilizando cualquiera de las técnicas de análisis de circuitos.
3. La tensión obtenida es *Vth*.

Para obtener la resistencia de Thévenin, *Rth*, entre dos puntos del circuito A y B, se realizan los siguientes pasos:

1. Se desconecta el circuito de carga dejando abiertos los terminales A y B.
2. Se cortocircuitan todas las fuentes de tensión del circuito.
3. Se abren todas las fuentes de corriente del circuito.
4. Se calcula la resistencia equivalente vista desde los terminales A y B.
5. La resistencia obtenida es *Rth*.

El siguiente paso para obtener el circuito equivalente de Thévenin es conectar la fuente de tensión Thévenin en serie con la resistencia de Thévenin y a éstas conectar el circuito de carga original. En la Figura 2.4 se muestra un esquema de la obtención de un circuito equivalente de Thévenin.

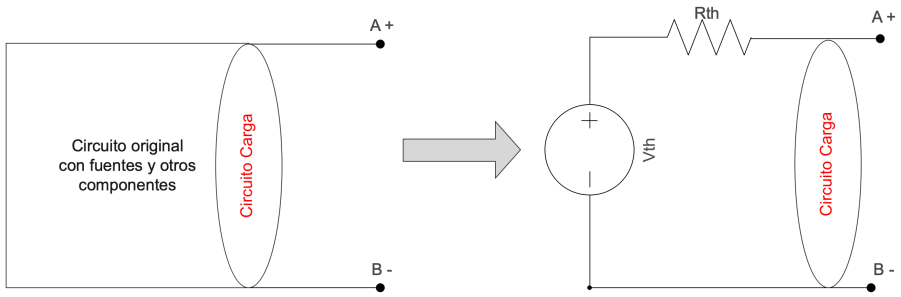


Figura 2.4: Circuito equivalente de Thévenin.

2.5. Teorema de Norton

El **Teorema de Norton** establece que cualquier circuito eléctrico lineal, por complejo que sea, puede reducirse a un circuito equivalente con una fuente de corriente en paralelo con una resistencia. A la corriente que suministra esta fuente se le denomina **Corriente de Norton,** I_N, y a la resistencia equivalente se le denomina **Resistencia de Norton,** R_N.

A menudo, la resistencia equivalente se encuentra como *resistencia Thévenin*, debido a que es exactamente la misma resistencia - y se calcula de la misma forma - que la resistencia en el circuito equivalente de Thévenin. Los componentes que permanecen del circuito original y no forman parte del equivalente sino que se conectan a él, forman el circuito de carga.

Para obtener la corriente de Norton, I_N, entre dos puntos del circuito A y B, se realizan los siguientes pasos:

1. Se desconecta el circuito de carga, y se conectan los terminales formando un *cortocircuito*.
2. Se calcula la corriente que circula entre estos terminales a través del cortocircuito (corriente de cortocircuito) utilizando cualquiera de las técnicas de análisis de circuitos.
3. La corriente obtenida es I_N.

Para obtener la resistencia de Norton, I_N, entre dos puntos del circuito A y B, se realizan los siguientes pasos:

1. Se desconecta el circuito de carga y se dejan abiertos los terminales A y B.
2. Se cortocircuitan las fuentes de tensión del circuito.
3. Se abren todas las fuentes de corriente del circuito.
4. Se calcula la resistencia equivalente vista desde los terminales A y B.
5. La resistencia obtenida es R_N.

El siguiente paso para obtener el circuito equivalente de Norton es conectar la fuente de corriente de Norton en paralelo con la resistencia de Norton y a éstas conectar el circuito de carga original. En la Figura 2.5 se muestra un esquema de la obtención de un circuito equivalente de Norton.

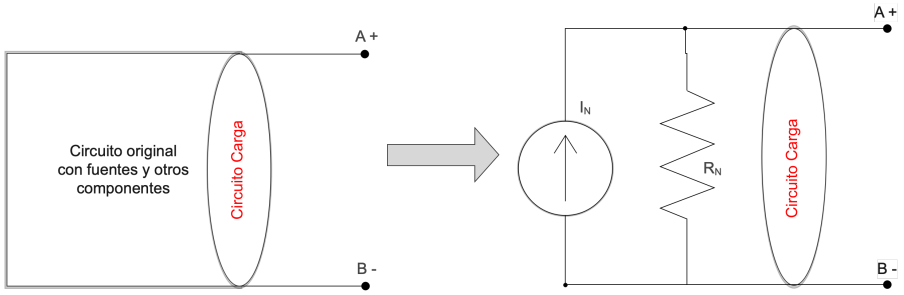


Figura 2.5: Circuito equivalente de Norton.

2.6. Análisis nodal en circuitos eléctricos

El método de **nodos** es una técnica de análisis de circuitos que permite encontrar los *voltajes* en los nodos de un circuito haciendo uso de la **Ley de Corriente de Kirchhoff (LCK)**. Una vez obtenidas se puede aplicar la **Ley de Ohm** para obtener las otras magnitudes.

2.6.1. Método básico de análisis nodal

La LCK genera una ecuación en cada nodo sumando las corrientes que entran en el nodo e igualando esa suma a la suma de corrientes que salen del nodo. Por tanto, el número de ecuaciones para resolver el circuito es igual al número de nodos que contenga el circuito.

Los pasos a seguir para resolver un circuito por el método de nodos son los siguientes:

1. El circuito debe ser una red plana, es decir, no debe haber cruces entre conductores.
2. Se debe determinar el número de nodos, *reales* o *ficticios*.[1]
3. Se debe asignar una *tensión* o *voltaje* para cada nodo del circuito, escogiendo un **nodo de referencia** cuyo voltaje será 0 voltios.
4. Verificar el modelo de circuito y aplicar los criterios adecuados para armar las ecuaciones.[2]
5. Ordenar y resolver las ecuaciones de nodos para obtener los voltajes.
6. Usar la *Ley de Ohm* para resolver el resto del circuito.

Para poder realizar el análisis correctamente se debe tener en cuenta el *modelo* de circuito según los componentes que tenga para operar de una manera u otra. Podemos considerar los siguientes modelos:

[1] Se consideran *reales* donde confluyen tres o más ramas eléctricas y *ficticios* donde confluyen sólo dos ramas eléctricas.

[2] Ver los modelos de circuitos en el siguiente apartado.

- Circuito que contiene SOLAMENTE **fuentes de corriente independientes**: en este caso, sólo hay que aplicar la LCK en cada nodo para encontrar las ecuaciones nodales.
- Circuito que contiene una o más **fuentes de tensión independientes**: en este caso en el que alguna rama conectada a un nodo contiene una fuente de tensión independiente, el voltaje del nodo será igual al valor de la fuente.
- Circuito que contiene **fuentes dependientes**: en este caso, se aplicarán los dos puntos anteriores como corresponda pero teniendo en cuenta que éstas fuentes tendrán un valor controlado por algún parámetro conocido o no del circuito lo que podrá complicar ligeramente la ecuación del nodo.
- Circuito que contiene **supernodos**[3]: en este caso, se debe reemplazar la fuente por un *cortocircuito*, de forma que se reducen los nodos a un único nodo y se aplica la LCK en ese nodo. Posteriormente, se obtiene una ecuación más denominada **ecuación de restricción**, relacionando los voltajes de los dos nodos involucrados.

2.6.2. Análisis Nodal Modificado (MNA)

Se dedica este apartado para comentar la técnica del *Análisis Nodal Modificado* [5] que es la utilizada por la aplicación creada para este proyecto. Existen múltiples herramientas algebraicas para resolver los sistemas que se forman utilizando esta técnica pero el uso de matrices facilita las operaciones y la escritura de código para programar la aplicación.

Se trata de aplicar el *álgebra lineal* para aplicar la LCK en un circuito. La idea es escribir todas las ecuaciones nodales en forma matricial:

$$Y \cdot V = I \tag{2.6}$$

- **Y** es la **matriz de conductancias** del circuito. La **conductancia** es la magnitud inversa a la resistencia. Una alta conductancia implica una baja resistencia. Su unidad en el Sistema Internacional es el **Siemens, S**.
- **V** es el **vector de tensiones** de nodo desconocidas respecto del nodo de referencia.
- **I** es el **vector de corrientes** independientes que entran en cada nodo.

Los pasos para resolver un circuito con la técnica de análisis nodal usando matrices serían los siguientes:

1. Se selecciona el nodo de referencia o tierra.
2. Se nombran las tensiones desconocidas de cada nodo como V_1, V_2, ..., V_N.
3. Convertir las resistencias a conductancias G=1/R.
4. Se escribe la KCL en cada nodo (que no sea el de referencia): suma de corrientes que entran igual a la suma de corrientes que salen del nodo.
5. Se expresa cada corriente por la Ley de Ohm:

$$I_{ij} = \frac{V_i - V_j}{R_{ij}} = (V_i - V_j) \cdot G_{ij} \qquad\qquad G_{ij} = \frac{1}{R_{ij}} \tag{2.7}$$

6. Armar la matriz de conductancias Y:
   a) Para un circuito con n nodos, se crea una matriz de $n \times n$ inicializada a cero.
   b) Se rellena la diagonal Y_{ii}:

$$Y_{ii} = \sum \text{Conductancias conectadas al nodo } i \tag{2.8}$$

   c) Se rellenan los elementos fuera de la diagonal Y_{ij} con $(i \neq j)$ de la siguiente forma:
      - Si existe rama directa entre i y j con admitancia G_{ij}, entonces:

$$Y_{ij} = -G_{ij} \tag{2.9}$$

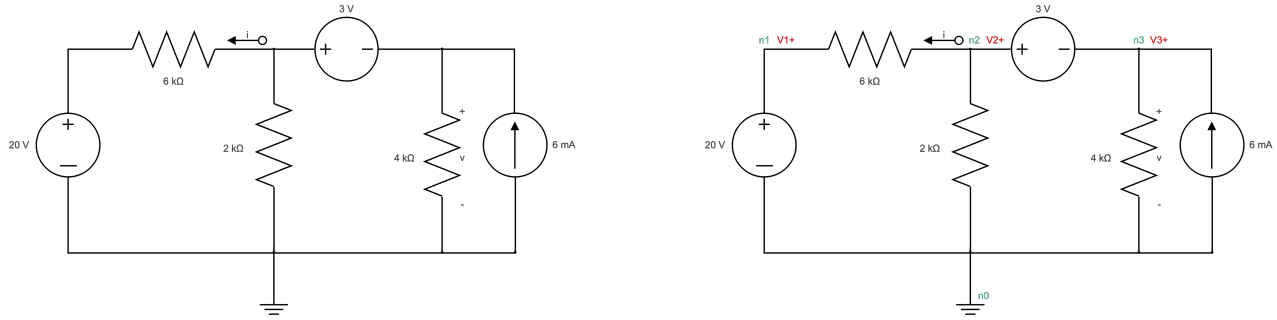
      - Si no hay conexión directa, permanece en cero.

[3]Se forma un **supernodo** cuando hay una fuente de tensión entre dos nodos y ninguno de éstos es el nodo de referencia (nodo a 0V).

7. Armar el vector de corrientes I:
   a) Se crea un vector columna de tamaño n (para n nodos).
   b) Para cada nodo i se suman las fuentes de corriente que entran al nodo y se restan las que salen (por convenio).
   c) Si no hay fuentes de corriente en un nodo se asigna un cero.
8. Gestionar las fuentes de tensión:
   a) Si un nodo está conectado a tierra mediante una fuente de tensión significa que la tensión en ese nodo es conocida y, por tanto, se quita del sistema lo que implica reducir el número de nodos, n y ajustar el vector I. Además, en cada ecuación de nodo vecino se debe sustituir V_i por el valor de la fuente.
   b) En el caso de que haya una fuente de tensión entre dos nodos (supernodo) y ninguno de ellos es el nodo de tierra, se deben combinar los dos nodos en un **supernodo**; se escribe una única ecuación LCK para el nodo ignorando la fuente de tensión interna; se añade la ecuación de restricción (ecuación de la fuente): $V_a - V_b = V_s$.
   c) Se debe usar la matriz ampliada para incluir la ecuación de restricción.
9. Resolución del sistema matricial por cualquier método teniendo en cuenta que la matriz **Y no sea singular**, es decir, su determinante debe ser distinto de cero ($\det Y \neq 0$).

2.6.3. Un ejemplo de análisis nodal usando matrices

En esta sección, se muestra un ejemplo de análisis de un circuito usando la técnica de análisis nodal con matrices. En concreto, se analiza el circuito de la Figura 2.6 [6].



(a) Circuito original.

(b) Circuito con nodos numerados y tensiones asignadas.

Figura 2.6: Aplicación del análisis nodal en un circuito con distintas representaciones

En el esquema de la Figura 2.6a, se ve el circuito de partida con tres resistencias, dos fuentes de tensión independientes y una fuente de corriente independiente. Para usar el análisis nodal, en primer lugar, se asignará un número de nodo y se elegirá un nodo de referencia; además, se asignará un voltaje a cada nodo tal como se había explicado. En la Figura 2.6b, se ve el mismo circuito con los nodos numerados y los voltajes asignados. Se van a convertir las resistencias en conductancias de la siguiente manera:

- $R_1 = 6\,k\Omega \quad \longrightarrow \quad G_1 = \frac{1}{R_1} = 0,167\,mS$
- $R_2 = 2\,k\Omega \quad \longrightarrow \quad G_1 = \frac{1}{R_2} = 0,5\,mS$
- $R_3 = 4\,k\Omega \quad \longrightarrow \quad G_1 = \frac{1}{R_3} = 0,25\,mS$

El nodo n_1 es un nodo ficticio que conecta a tierra (nodo n_0) a través de una fuente de tensión independiente de 20V, por tanto, V_1 adopta el valor de la fuente: $V_1 = 20\,V$. Los nodos n_2 y n_3 están separados por una fuente de tensión de 3V, por tanto, forman un *supernodo* que tomará el nombre temporal n_{23}. Las incógnitas son las

siguientes, donde I_{23} es la corriente a través de la fuente de tensión de 3V:

$$x = \begin{bmatrix} V_2 \\ V_3 \\ I_{23} \end{bmatrix} \tag{2.10}$$

La forma de la ecuación es la siguiente:

$$\begin{bmatrix} \boldsymbol{G} & \boldsymbol{B} \\ \boldsymbol{B^T} & \boldsymbol{0} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{V} \\ \boldsymbol{I_{vs}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{E} \end{bmatrix} \tag{2.11}$$

Se arma la **matriz G** de conductancias, teniendo en cuenta el nodo 2 conectado a tierra, el nodo 3 conectado a tierra y el nodo 2 ->3, cuya conductancia sería 0.

$$G = \begin{bmatrix} \frac{1}{6000} & 0 \\ 0 & \frac{1}{4000} \end{bmatrix} \tag{2.12}$$

Como se puede ver, se forma la matriz de conductancias en unidades del SI (en Siemens). Se arma, ahora la **matriz B** que es la matriz de incidencia de las fuentes de tensión. Para cada fuente de tensión entre dos nodos, se introduce una incógnita de corriente para la fuente definida desde el terminal negativo al terminal positivo y se añade una columna por cada fuente. Estas columnas se rellenan con +1 en la fila del nodo positivo (corriente que entra en el nodo) y -1 en la fila del nodo negativo (corriente que sale del nodo) y 0 en las demás filas para nodos no conectados a la fuente.

$$B = \begin{bmatrix} -1 \\ +1 \end{bmatrix} \tag{2.13}$$

Por tanto, la ecuación de la fuente de tensión se escribe de la siguiente forma:

$$B^T \cdot V = E \Longrightarrow (-1) \cdot V_2 + (1) \cdot V_3 = 3 \Longrightarrow V_3 - V_2 = 3 \tag{2.14}$$

Ahora se forma el **vector de corrientes, I** inyectadas en cada nodo. En el nodo 2, no hay fuente de corriente pero si está el término conocido V_1; si aquí se aplica la LCK:

$$(V_2 - V_1) \cdot G_{6k} - I_{23} = 0 \Longrightarrow G_{6k} \cdot V_2 - I_{23} = G_{6k} \cdot V_1 \tag{2.15}$$

Por tanto:

$$I_2 = G_{6k} \cdot V_1 = \frac{1}{6000} \cdot 20 = 3{,}333mA \tag{2.16}$$

En el nodo 3 entra la fuente independiente de 6mA.

$$I = \begin{bmatrix} 0{,}00333A \\ 0{,}006A \end{bmatrix} \tag{2.17}$$

El **vector E** de valores de las fuentes de tensión será $E = \begin{bmatrix} 3 \end{bmatrix} V$. El sistema completo, por tanto, sería:

$$\begin{bmatrix} \frac{1}{6000} & 0 & -1 \\ 0 & \frac{1}{4000} & +1 \\ -1 & +1 & 0 \end{bmatrix} \cdot \begin{bmatrix} V_2 \\ V_3 \\ I_{23} \end{bmatrix} = \begin{bmatrix} 0{,}00333 \\ 0{,}006 \\ 3 \end{bmatrix} \tag{2.18}$$

Al resolver este sistema se obtienen los valores buscados de tensión y corriente:

$$V_2 = 20{,}6V \quad V_3 = 23{,}6V \quad I_{23} = 0{,}10mA \tag{2.19}$$

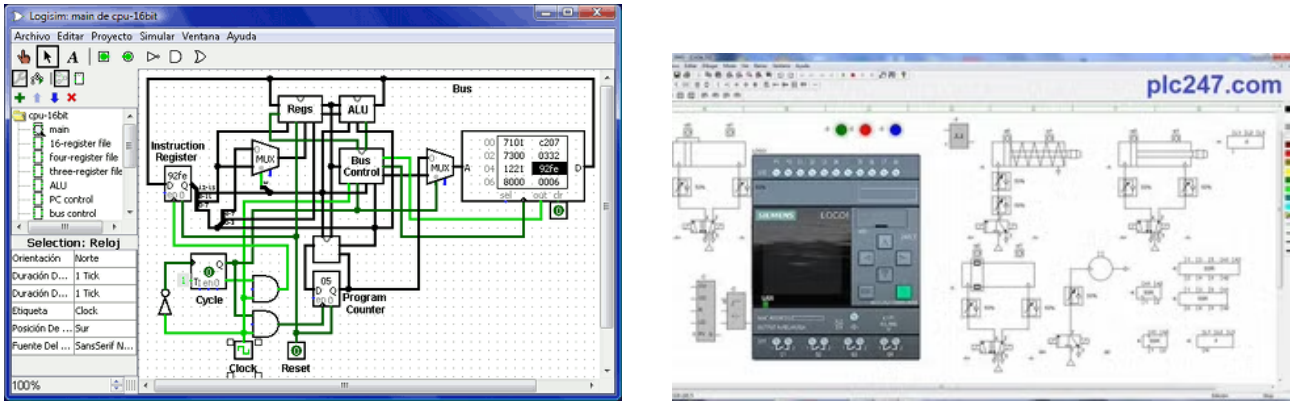
Teniendo esto resuelto, se conocen todas las tensiones y corrientes del circuito y, por tanto, el circuito quedaría resuelto. Esto es un ejemplo de un análisis nodal matricial que, como ya se ha comentado, es la técnica que utiliza la aplicación creada en este proyecto.

2.7. Aplicaciones educativas en el aprendizaje de circuitos

En el mercado existen múltiples aplicaciones orientadas al aprendizaje de circuitos eléctricos, cada una de ellas, enfocada en alguna técnica concreta de análisis o síntesis de circuitos; otras se basan en el tipo de circuitos (lógicos, digitales, analógicos, etc.).

Algunas aplicaciones están diseñadas para uso de estudiantes de ingeniería o de módulos de electricidad, electrotecnia, física, etcétera. Otras están diseñadas para el uso profesional. Esta es una breve revisión de algunas de ellas:

- **Logisim** [7]: es un software de libre distribución para el diseño y simulación de circuitos lógicos por medio de *drag and drop*. Está orientado al diseño de CPUs con propósito educativo.
- **CircuitX**: se trata de una aplicación para dispositivos móviles para diseñar y simular circuitos adecuada para estudiantes o aficionados.
- **CADe_SIMU** [8]: es otro software gratuito para el diseño y simulación de circuitos. Su punto fuerte es la facilidad de uso y el poco consumo de recursos.



(a) Ventana de Logisim 2.7.1.

(b) Ventana de CADe_SIMU.

Figura 2.7: Interfaces de Logisim y CADe_SIMU.

Mención especial merece el software **PSpice** [9]. Se trata de una herramienta de simulación de circuitos electrónicos profesional y muy utilizada. Su base (y la de otros simuladores similares) está en el programa SPICE (Simulation Program with Integrated Circuit Emphasis) que fue desarrollado por la Universidad de California en Berkeley en la década de 1970. PSpice es una versión comercial de este programa SPICE desarrollada por la empresa *Cadence Design Systems* adaptada y mejorada para trabajar en ordenadores personales con una interfaz gráfica de usuario más amigable.

Aunque no está orientado especialmente al aprendizaje sino a un usuario más avanzado (ingenieros o diseñadores), tiene un gran número de posibilidades tanto en diseño como en simulación. Para el análisis numérico de cálculo de circuitos, utiliza el método de análisis nodal modificado (MNA) similar al que usa la aplicación desarrollada en este proyecto.

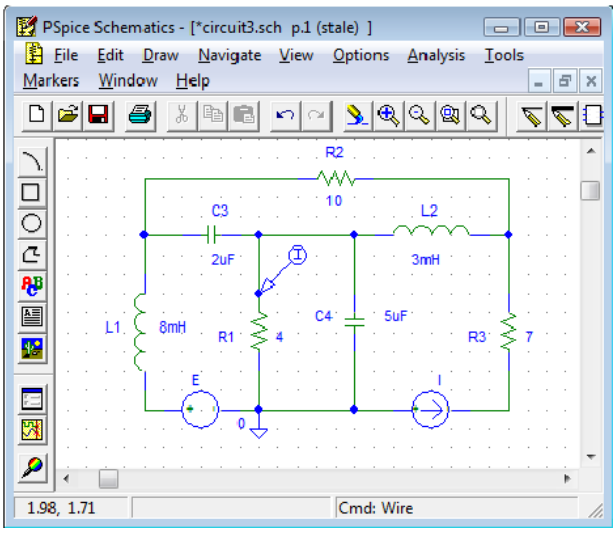


Figura 2.8: Ventana de PSpice.

2.8. Conclusiones

Existen en el mercado un buen número de aplicaciones, tanto gratuitas como comerciales, cada una de ellas, enfocadas a alguna técnica o a algún tipo de circuito concreto, sobretodo, a la simulación de circuitos eléctricos y, aunque PSpice aborda casi todo el diseño y simulación de circuitos, su aprendizaje requiere horas de entrenamiento y, de entre las aplicaciones orientadas a estudiantes, ninguna de las aquí mencionadas hace hincapié en la obtención de circuitos equivalentes más sencillos.

Además, como se puede comprobar, el análisis de circuitos es una disciplina relativamente compleja que requiere sentar una buena base en cualquier alumno en su aprendizaje ya que, a medida que los circuitos se complican, los cálculos empiezan a ser más complejos y más tediosos. Cuando, posteriormente, se analizan circuitos en corriente alterna, circuitos en régimen transitorio o circuitos que contienen diodos, transistores, etcétera, el hecho de obtener un circuito equivalente tan sencillo como el que se consigue usando los teoremas de Thévenin y Norton, es una tarea que cualquier alumno debe dominar.

Estas razones, hacen que la aplicación *ThevenApp* sea una alternativa para que los estudiantes sienten sus bases en el análisis de circuitos.

Capítulo 3

Análisis de requisitos

En este capítulo se analizan los requisitos que debe cumplir la aplicación. En primer lugar, se hace un recorrido por los requisitos funcionales tales como el diseño del circuito con la técnica de «drag and drop», el conexionado manual, asignación de nodos, el análisis según los teoremas de Thévenin y Norton, la gestión de teoría y ejercicios en la sección de e-learning y la gestión de archivos. Posteriormente, se hace una revisión de los requisitos no funcionales tales como la usabilidad, accesibilidad, rendimiento, mantenibilidad, etcétera. Por último se muestra el diagrama de casos de uso de la aplicación.

3.1. Requisitos funcionales

El estudio tanto de los requisitos funcionales como no funcionales de una aplicación de software supone una parte importante dentro de su desarrollo. El software debe cumplir unas ciertas características en base a lo que fue diseñado y en base a los objetivos que se establecieron antes de comenzar su desarrollo.

Los requisitos funcionales indican qué debe hacer el software, cómo debe comportarse o cómo se interacciona con él. Cuando surge la idea del diseño de una aplicación se establecen una serie de objetivos y/o requisitos que debe cumplir y que la hacen útil para el propósito para el que fue diseñada.

3.1.1. Diseño de circuitos

Uno de los requisitos funcionales de la aplicación es que se pueda diseñar un circuito eléctrico en un área de dibujo arrastrando componentes de un panel y poder conectarlos con cables. Esto requiere que el usuario pueda navegar por un panel de componentes, seleccionar un componente en concreto, arrastrarlo hasta el área de dibujo y soltar donde considere oportuno. También deberá poder moverse ese mismo componente por todo el área de dibujo, seleccionarlo, modificarlo, escalarlo o eliminarlo.

Para modificar ciertos parámetros, se deberá abrir un panel de propiedades del componente, donde se dará toda la información relevante del componente y donde se podrá modificar su valor, indicar si es un componente de carga, modificar los nodos de conexión y, en el caso de una fuente dependiente, modificar los nodos de control y el tipo de magnitud que controla su valor -*tensión o corriente*-.

Otro requisito funcional será poder conectar los componentes a través de cables de una manera ordenada y estética. En este punto se debe tener en cuenta que los cables deben seguir trayectorias ortogonales automatizadas pudiendo crear puntos de control de movimiento y eliminarlos y que, una vez conectados a un componente, si el componente se mueve, el cable debe acompañar el movimiento.

Se requiere un sistema automático de gestión de nodos para evitar que el usuario pueda cometer errores de

numeración manual. Por tanto, el sistema, cada vez que se genera una conexión, debe asignar un número de nodo.

El componente de *tierra* debe ser único y situarse donde el usuario estime oportuno. Por facilidad de cálculo, el sistema asignará a este nodo el número 0 y reasignará a los demás nodos del número 1 en adelante.

En el caso de que algún cambio comprometa la integridad del circuito se avisará al usuario por medio de ventanas de diálogo con las que podrá interactuar.

A modo de resumen:

- [**RF-DIS-01**] Debe existir un área de diseño donde se podrá dibujar o diseñar un circuito.
- [**RF-DIS-02**] Debe existir un panel de componentes desde donde se podrán arrastrar al área de dibujo.
- [**RF-DIS-03**] Estos componentes se deben poder mover por todo el área de dibujo arrastrando los cables que tengan conectados, escalar haciendo uso de la rueda del ratón o eliminar con una tecla del teclado.
- [**RF-DIS-04**] Al seleccionar un componente se debe abrir un panel de propiedades donde se podrán modificar sus parámetros.
- [**RF-DIS-05**] Se podrán conectar componentes a través de cables de una forma clara y ordenada.
- [**RF-DIS-06**] Se avisará al usuario si intenta realizar una acción que comprometa el funcionamiento del circuito.

3.1.2. Análisis de un circuito

Una vez que el usuario ha diseñado un circuito, éste debe poder analizarse siguiendo la técnica para la que fue diseñada la aplicación.

Para ello, el usuario podrá elegir el o los componentes de carga, es decir, los componentes que permanecerán intactos y se conectarán al circuito equivalente obtenido. Deberá haber dos botones para que el usuario pueda elegir si desea obtener un *circuito equivalente de Thévenin* o un *circuito equivalente de Norton* y seleccionar los nodos sobre los que se calcula y donde se conectarán, posteriormente, los componentes de carga.

Una vez, elegida la opción, la aplicación deberá realizar los cálculos necesarios para obtener el circuito equivalente y mostrarlo al usuario con los componentes de carga conectados, el cableado y, por supuesto, la fuente y resistencia equivalentes a la que van conectados.

El usuario podrá consultar la información sobre el análisis realizado en un panel dinámico que contendrá los datos calculados y un resumen del mismo.

En resumen:

- [**RF-ANA-01**] Debe existir una sección de análisis donde se cargará el circuito diseñado para su posterior análisis.
- [**RF-ANA-02**] Deben existir dos botones - uno para *Thévenin* y otro para *Norton* - y un selector de nodos de control.
- [**RF-ANA-03**] Una vez elegida la opción, el sistema deberá mostrar el circuito equivalente con los componentes de carga conectados de forma visual.
- [**RF-ANA-04**] El usuario tendrá a su disposición un resumen del análisis realizado y qué ventajas conlleva.

3.1.3. Aprendizaje de circuitos

Uno de los objetivos principales es que, además de una herramienta de diseño y análisis de circuitos, cualquier estudiante de ingeniería, física, electricidad, etc. pueda aprender análisis de circuitos y, en particular, los teoremas

de Thévenin y Norton y llevar un seguimiento de su aprendizaje.

Para ello, la aplicación debe contar con una base de datos que gestione dos tipos de usuario: *profesor* y *alumno*. Con el rol de *alumno*, el usuario tendrá acceso a teoría sobre análisis de circuitos u otros recursos que ponga a su disposición un *profesor* y acceso a ejemplos y ejercicios que podrá realizar el alumno. La aplicación por su parte guardará el progreso del estudiante y puntuará los ejercicios realizados.

El rol de *profesor* permitirá valorar el aprendizaje de uno o varios alumnos, ponerles a su disposición recursos variados sobre los circuitos eléctricos así como actualizar sus ejercicios, asignarles nuevas tareas o mostrarles notas sobre las tareas ya realizadas.

En resumen:

- [**RF-EDU-01**] Debe contar con una base de datos para gestión de usuarios y ejercicios de la aplicación.
- [**RF-EDU-02**] Debe contar con un espacio interactivo de teoría y ejercicios a disposición del estudiante.
- [**RF-EDU-03**] Debe poder elegirse entre dos roles: alumno y profesor.
- [**RF-EDU-04**] El rol de alumno debe poder acceder a todo el material, incluido el proporcionado por el profesor y llevar un seguimiento de su aprendizaje.
- [**RF-EDU-05**] El rol de profesor debe poder poner a disposición del alumno recursos, ejercicios nuevos u otro material que considere oportuno, dejar notas al alumno y valorar su actividad.

3.1.4. Otros requisitos funcionales

Además de los requisitos funcionales para los que fue diseñada la aplicación, debe realizar otro tipo de tareas importantes para el buen funcionamiento de la aplicación.

Una vez diseñado un circuito, debe poder guardarse en un archivo. Este archivo se podrá cargar, posteriormente, para continuar trabajando con él o para realizar un análisis.

Para facilitar el uso de la aplicación, debe contar con un registro de archivos recientes que el usuario podrá seleccionar para cargar un archivo concreto, recientemente, utilizado y borrar este registro.

- [**RF-PER-01**] Debe poder guardarse un circuito ya diseñado en un archivo en disco y cargarse posteriormente para continuar trabajando.
- [**RF-PER-02**] Debe mostrarse un mensaje de error o advertencia de log cuando se realice una acción no válida.
- [**RF-PER-03**] Debe mantener un registro de archivos recientes que el usuario puede usar o reiniciar.

3.2. Requisitos no funcionales

No tiene mucho sentido que una aplicación realice ciertas tareas, si el usuario no puede acceder a ellas, si no se hacen en un tiempo adecuado, si la aplicación se bloquea o si la seguridad del usuario se ve comprometida. Los *requisitos no funcionales* de una aplicación establecen una serie de objetivos que ésta debe cumplir a parte de todas sus funcionalidades. Estos requisitos se centran en cuestiones como usabilidad, rendimiento, portabilidad, mantenibilidad, escalabilidad o seguridad; es decir, los requisitos no funcionales se centran en *cómo debe funcionar un sistema* en lugar de *qué debe hacer*. Los requisitos no funcionales establecen lo que debe cumplir una aplicación para que, aparte de funcionar, sea un software de calidad.

3.2.1. Usabilidad

La interfaz de usuario debe ser sencilla e intuitiva. Puesto que es una aplicación diseñada para estudiantes, cualquier usuario no experimentado debe saber utilizarla.

La aplicación debe informar al usuario de las acciones realizadas; para ello contará con un gestor de mensajes que se mostrarán en pantalla cuando el usuario realice alguna opción importante tal como añadir un componente al circuito, eliminarlo, guardar un circuito en archivo u otras tareas.

Además, la aplicación mantendrá un registro de logs que el usuario podrá consultar, guardar o imprimir cuando surja cualquier error o comportamiento inesperado.

3.2.2. Rendimiento

No tendría sentido poder realizar una tarea como analizar un circuito si no lo hace en un tiempo adecuado. La aplicación debe realizar todas las tareas en un tiempo razonable.

Se debe poder diseñar un circuito, por complejo que sea, en relativamente poco tiempo. Se debe poder arrastrar componentes en tiempo real. Conectar cables debe ser una tarea de unos pocos clics.

La obtención de circuito equivalente, la gestión de archivos, el uso de la base de datos, etc. son tareas que se deben realizar en poco tiempo. Además la aplicación no debe mostrar síntomas de bloqueo al realizar cualquiera de las tareas para las que fue diseñada.

3.2.3. Portabilidad

Al realizar una aplicación, ésta debería poder usarse desde diferentes plataformas pues, si sólo se diseña para un sistema concreto, sería difícil que todos los estudiantes pudieran acceder a ella. Por tanto, la portabilidad es un requisito no funcional que se debe tener en cuenta. La aplicación debe poder usarse, al menos, desde los sistemas operativos más importantes como Windows, Mac OS o Linux y la base de datos debe ser compatible con SQLite o PostgreSQL.

3.2.4. Mantenibilidad

Cuando un programador o diseñador realiza una aplicación y la pone a disposición de los usuarios, normalmente, otros programadores podrán realizar mejoras o gestionar errores no detectados durante el proceso de desarrollo del software. Por esta razón el código debe mantener una buena documentación **Javadoc**, debe diseñarse con modularidad pues es más fácil mantener códigos pequeños enfocados en una única tarea que un gran número de líneas de código.

Un buen registro de logs también es clave para que un usuario experimentado pueda corregir errores que puedan surgir durante el uso de la aplicación.

3.2.5. Escalabilidad

La aplicación debe dejar la puerta abierta para que, en el futuro, otros programadores puedan añadir ciertas funcionalidades que no son, por ahora, objetivo del proyecto.

El objetivo de la aplicación en este proyecto es diseñar y analizar circuitos resistivos en corriente continua que podrán contener fuentes independientes o dependientes de tensión o corriente y resistencias. En el futuro, se podrá ampliar para contemplar circuitos en corriente alterna, gestionar componentes como condensadores, bobinas, diodos o transistores o realizar otros tipos de análisis utilizando técnicas diferentes a los teoremas de Thévenin y Norton.

3.2.6. Seguridad

La seguridad de una aplicación es un factor importante a tener en cuenta durante el desarrollo de la misma, sobretodo si la aplicación conecta con una base de datos en la que se incluyen datos de carácter personal: nombre, apellidos, contraseñas, calificaciones u cualquier dato similar.

La aplicación debe garantizar la seguridad de los datos que se generan en la base de datos ante posibles filtraciones; para ello los datos de acceso de alumnos y profesores deben guardarse cifrados y sólo los usuarios registrados podrán acceder al material.

3.3. Tablas de requisitos funcionales y no funcionales

La Tabla 3.1 contiene los principales requisitos funcionales y no funcionales que la aplicación debe cumplir.

Tabla 3.1: Requisitos funcionales de la aplicación.

Requisitos Funcionales	
Diseño	• **[RF-DIS-01]** Disponer de un área de diseño. • **[RF-DIS-02]** Disponer de un panel de componentes arrastrables al área de diseño. • **[RF-DIS-03]** Poder mover, escalar o eliminar componentes. • **[RF-DIS-04]** Modificar propiedades de un componente. • **[RF-DIS-05]** Conexión ordenada mediante cables. • **[RF-DIS-06]** Avisos al usuario ante acciones críticas.
Análisis	• **[RF-ANA-01]** Disponer de una sección de análisis. • **[RF-ANA-02]** Disponer de dos botones y selector de nodos para Thévenin/Norton. • **[RF-ANA-03]** Mostrar gráficamente el equivalente seleccionado. • **[RF-ANA-04]** Mostrar información del análisis realizado. • **[RF-ANA-05]** Validación del circuito previa al análisis.
E-Learning	• **[RF-EDU-01]** Disponer de una base de datos de usuarios. • **[RF-EDU-02]** Disponer de un espacio interactivo de teoría y ejercicios. • **[RF-EDU-03]** Posibilidad de elegir entre dos roles: alumno y profesor. • **[RF-EDU-04]** El alumno podrá acceder al material y realizar un seguimiento. • **[RF-EDU-05]** El profesor podrá publicar material y ejercicios. • **[RF-EDU-06]** El profesor podrá proponer como ejercicios los diseños propios realizados en la aplicación.
Persistencia	• **[RF-PER-01]** Guardar/Cargar circuito desde archivo. • **[RF-PER-02]** Mostrar log de errores o advertencias. • **[RF-PER-03]** Mantener una lista de archivos recientes.

Tabla 3.2: Requisitos no funcionales de la aplicación.

Requisitos No Funcionales	
Usabilidad	• Disponer de una interfaz de usuario sencilla e intuitiva. • Informar al usuario mediante mensajes de las acciones importantes. • Gestión de logs para consulta, exportación o impresión.
Rendimiento	• Realización de tareas en tiempo razonable. • No se deben mostrar símbolos de bloqueo.
Portabilidad	• Se podrá utilizar desde Windows, Mac OS o Linux. • Base de datos compatible con SQLite o PostgreSQL.
Mantenibilidad	• El código debe estar bien documentado con Javadoc. • Debe tener un registro de logs para facilitar el mantenimiento.
Escalabilidad	• Debe poder actualizarse en el futuro incluyendo nuevas funcionalidades.
Seguridad	• Debe garantizarse la seguridad de datos personales. • Los datos deberán estar cifrados.

3.4. Diagrama de casos de uso

Un diagrama de casos de uso es una herramienta del **lenguaje unificado de modelado (UML)** que se utiliza para especificar la funcionalidad de un sistema y sus requisitos. Se trata de una representación gráfica que muestra la interacción entre usuarios y un sistema, es decir, cómo el usuario interactúa con una aplicación [10].

En esta sección, se presenta el diagrama de casos de uso de la aplicación *ThevenApp*. Dado que la aplicación está dividida en tres paneles o secciones bien diferenciadas (diseño, análisis y *e-learning*) se ha optado por realizar un diagrama de cada sección para mejorar la visualización del mismo.

En las Figuras 3.1, 3.2 y 3.3 se muestran los diagramas de casos de uso de las secciones de diseño, análisis y *e-learning* de la aplicación.

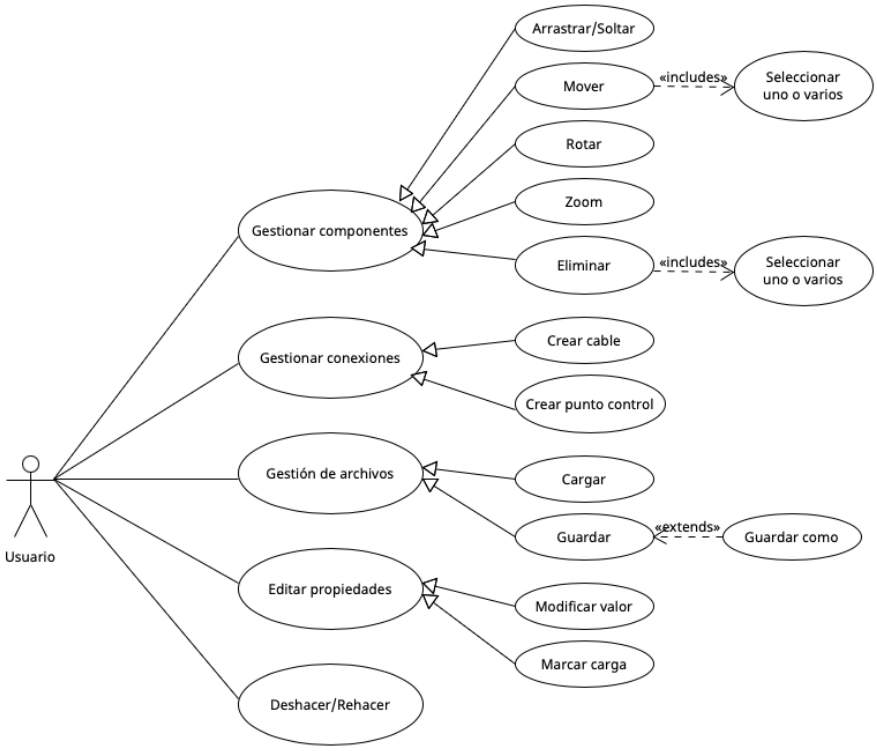


Figura 3.1: Diagrama de Casos de Uso del Panel de Diseño.

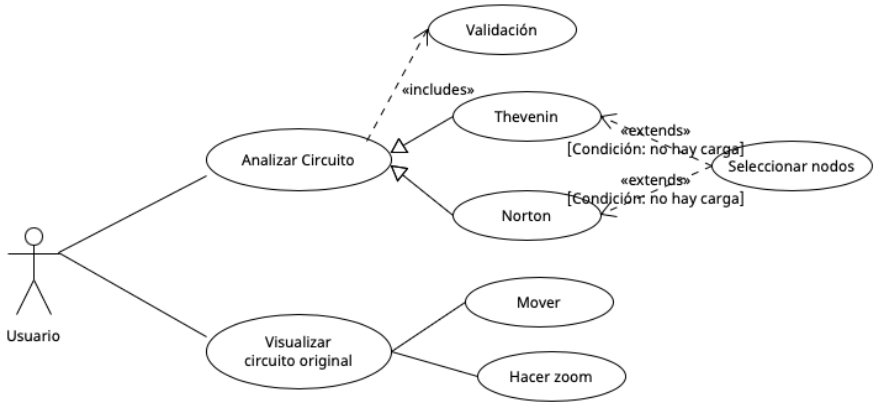


Figura 3.2: Diagrama de Casos de Uso del Panel de Análisis.

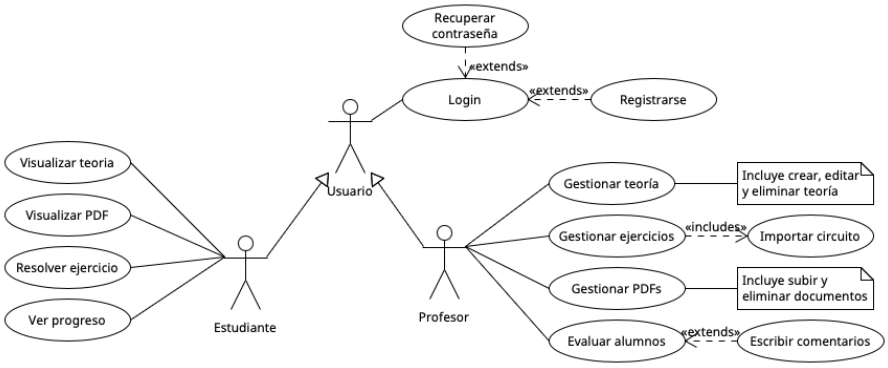


Figura 3.3: Diagrama de Casos de Uso del Panel de *e-learning*.

3.5. Conclusiones

Establecer unos requisitos funcionales y no funcionales que debe cumplir un sistema software durante el proceso de planificación y análisis del ciclo de vida es una tarea fundamental para que el proyecto concluya de forma exitosa.

En este proyecto se han establecido unos requisitos firmes que la aplicación deberá cumplir teniendo en cuenta para qué se desarrolla, a quién va dirigido y cuáles son los objetivos de este software.

El hecho de tener claros los requisitos que se deben cumplir, facilita las etapas de desarrollo, pruebas, uso final y la mantenibilidad posterior obteniendo, así, un software de calidad que cumple con las recomendaciones establecidas en ingeniería de software.

Capítulo 4

Diseño de la aplicación

Realizar el diseño de una aplicación requiere tomar un gran número de decisiones para que el resultado sea fiel a los objetivos. En primer lugar, se debe evaluar a qué público va dirigido, para qué se va a usar la aplicación, dónde y cómo. Estas evaluaciones llevan a tener que decidir cuál será el lenguaje de programación principal de la aplicación; elegir si será una aplicación web o una aplicación de escritorio; cómo será la interfaz gráfica de usuario; qué tipo de base de datos se utilizará y otras decisiones que darán como resultado una aplicación software bien diseñada.

4.1. Descripción general de la aplicación

El *punctum saliens* de este proyecto es la aplicación **ThevenApp**. Este es el nombre que se ha dado a la aplicación juntando el nombre de *Thévenin* y *Application* (aplicación en su traducción al inglés) en referencia a uno de los teoremas en los que se basa la aplicación: la obtención de un circuito equivalente de Thévenin o Norton a partir de un circuito eléctrico de tipo resistivo en corriente continua.
En la Figura 4.1 se puede ver el logotipo de la versión 1.0 de la aplicación.

Figura 4.1: Logotipo de la aplicación ThevenApp.

ThevenApp es una herramienta de escritorio, desarrollada en lenguaje **Java** [11] orientada, principalmente, a estudiantes y profesores de ingeniería eléctrica (o similar). Un objetivo de la aplicación es calcular de forma rápida y visual los circuitos equivalentes de Thévenin o Norton de un circuito original que podrá ser diseñado por el usuario. Este circuito será de tipo resistivo en corriente continua (DC), es decir, podrá contener fuentes dependientes o independientes de tensión o corriente y resistencias. El usuario podrá componer su propio circuito arrastrando componentes de un panel de componentes y conectando cables haciendo clics en los puntos de conexión de estos componentes. Seleccionando un componente se abrirá un panel de propiedades donde podrá modificar sus parámetros; también podrá rotar el componente para ponerlo en una posición adecuada o aumentar y disminuir su tamaño.

Estos diseños se podrán guardar en un archivo en disco y, posteriormente, cargar ese archivo en la aplicación.

Una vez diseñado un circuito, el usuario podrá seleccionar la opción de obtener un circuito equivalente de Thévenin o de Norton y la aplicación realizará los cálculos necesarios para, finalmente, mostrar gráficamente el nuevo circuito obtenido.

Puesto que la aplicación está orientada, principalmente, a la docencia, el usuario podrá acceder a una sección de *e-learning* donde podrá registrarse con rol de profesor o alumno. Un usuario con rol de alumno podrá acceder

a material didáctico donde podrá aprender la materia de *análisis de circuitos* y resolver ejercicios que pondrán en práctica los conocimientos adquiridos, incluso, usar la aplicación para comprobar sus soluciones. Además, podrá llevar un seguimiento de su evolución. Un usuario con rol de profesor podrá añadir material didáctico nuevo y poner a disposición del alumno nuevos ejercicios.

La aplicación comienza con un panel de bienvenida donde se ve el logotipo de *ThevenApp* junto a una lista de archivos recientes que se podrán cargar, crear un circuito nuevo o acceder a la sección de *e-learning* directamente desde este panel (ver Figura 4.2).

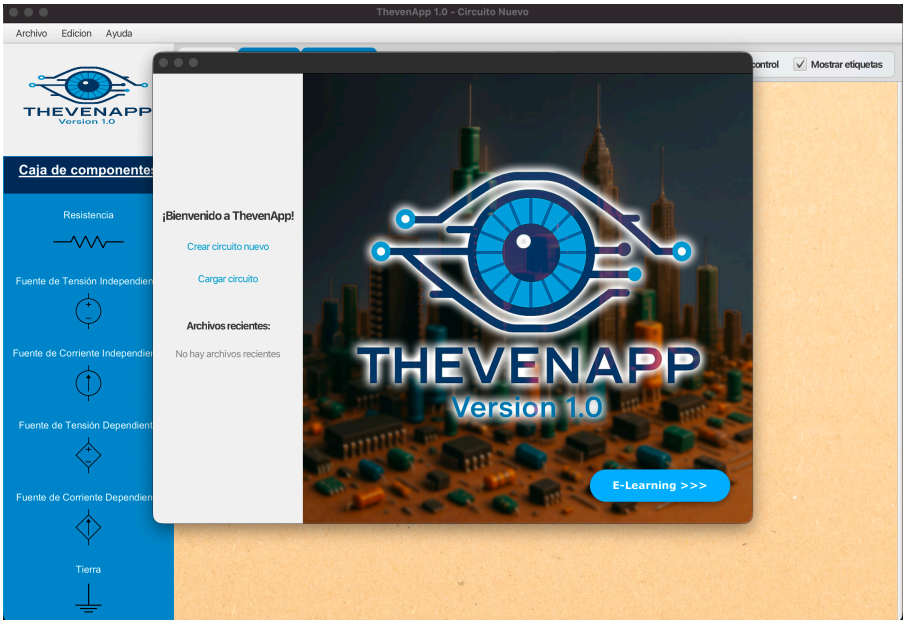


Figura 4.2: Panel de bienvenida de la aplicación.

Posteriormente, la aplicación carga con tres paneles principales:

- **Panel de Diseño**: donde se podrá diseñar el circuito.
- **Panel de Análisis**: donde se podrá obtener el circuito equivalente de Thévenin o Norton.
- **Panel de E-Learning**: espacio interactivo con material didáctico.

Una vez accedemos, la aplicación se abre en el panel de diseño desde donde se accede al panel de componentes para poder diseñar un circuito nuevo o modificar uno existente (ver Figura 4.3).

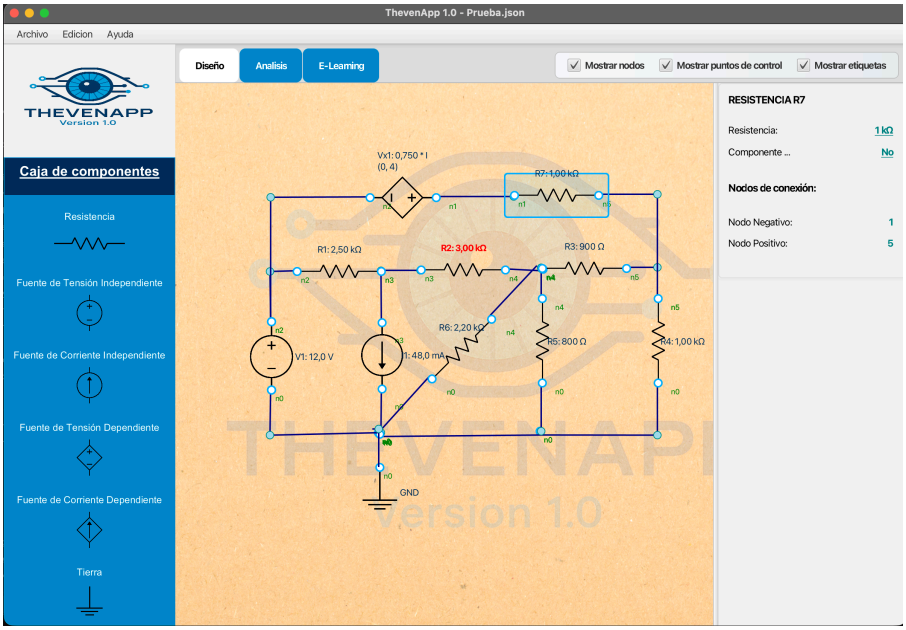


Figura 4.3: Panel de Diseño de la aplicación.

Una vez diseñado el circuito, se puede acceder al panel de análisis donde se pondrá a trabajar el motor de cálculo para obtener un circuito equivalente al que se ha diseñado en el panel anterior (ver Figura 4.4).

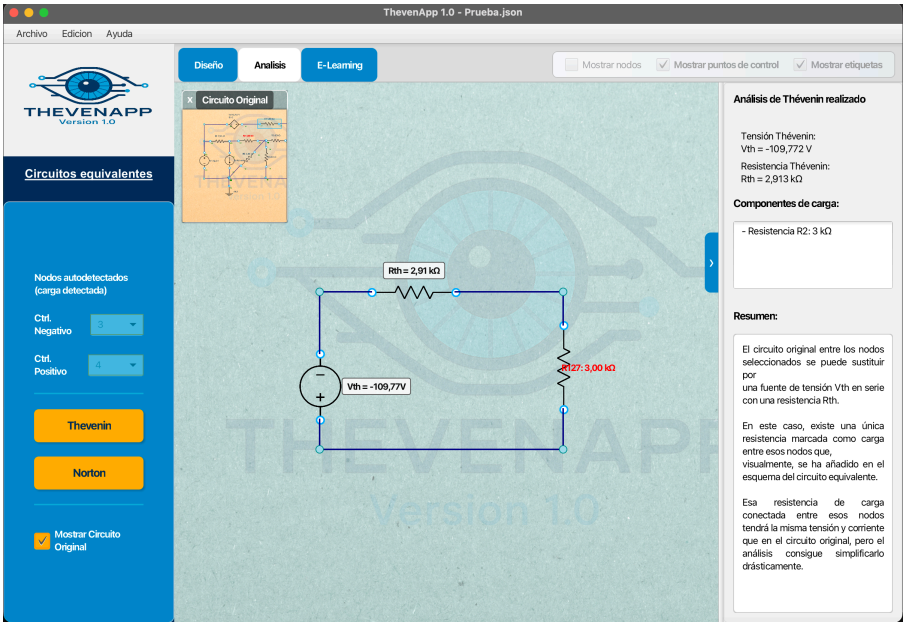


Figura 4.4: Panel de Análisis de la aplicación.

Posteriormente, se puede acceder al panel de *e-learning* para trabajar con el material didáctico disponible (ver Figura 4.5).

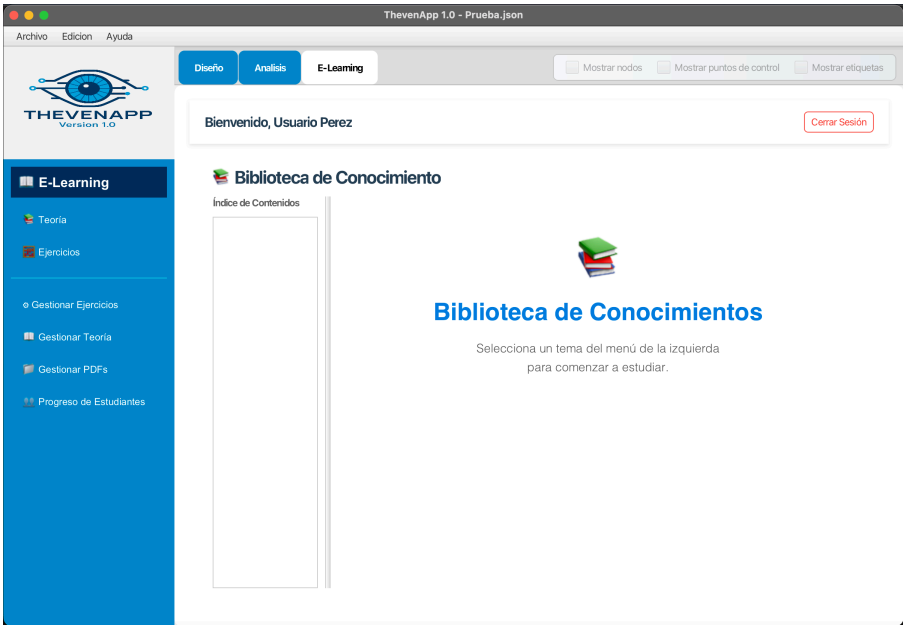


Figura 4.5: Panel de *E-Learning* de la aplicación.

4.2. Patrón MVC (Modelo-Vista-Controlador)

Hoy en día es, prácticamente, imposible pensar en una aplicación dirigida a usuarios que no esté implementada bajo el patrón MVC. Aplicaciones web, aplicaciones móviles o las aplicaciones de escritorio implementan esta arquitectura.

El patrón MVC es una arquitectura de software que separa las responsabilidades en tres capas diferentes:

- **Modelo:** El modelo se encarga de la lógica de negocio, es decir, todo lo que tiene que ver con los cálculos, la gestión de datos... En el caso de esta aplicación se refleja en la definición de componentes y cables, las clases de realización de cálculos, la gestión de archivos, los DTOs o la gestión de usuarios y contenido.
- **Vista:** La vista es la que se encarga de mostrar al usuario una interfaz amigable, con facilidad para interactuar y visualmente atractiva. Gracias al uso de JavaFX esta parte se ve, claramente, diferenciada en la aplicación a través de los archivos FXML o las hojas de estilo CSS.
- **Controlador:** El controlador es quien gestiona la interacción de la vista con el modelo de dominio. Cuando un usuario hace un clic en la interfaz, el controlador emite una orden al modelo para que gestione la petición, recoge el resultado y se lo vuelve a pasar a la vista para mostrarlo al usuario. De esto se encargan, los controladores de los FXML realizados en la aplicación.

La aplicación *ThevenApp* implementa este patrón por diversas razones entre las que se incluyen:

- *Separación de responsabilidades*: en una aplicación como esta que cuenta con una interfaz relativamente compleja y un motor de cálculo con gran responsabilidad, se facilita todo haciendo cada parte independiente y conectándolas a través de controladores.
- *Mantenimiento*: si alguna parte de la aplicación falla, no hay porqué tocar el resto.
- *Escalabilidad*: se podría aumentar la aplicación añadiéndole nuevas funcionalidades sin tener que romper todo el código.
- *Reutilización*: si se quisiera utilizar el motor de cálculo para realizar otra aplicación, se obtendría con total facilidad.

4.3. Modelo de dominio

Esta sección se centra en detallar el modelo de dominio de *ThevenApp* representando la definición de conceptos y las reglas aplicables a estos conceptos.

El modelo de dominio parte de la creación de clases que definan los componentes a utilizar en la aplicación y cómo se relacionan entre ellos para formar un *circuito eléctrico*. La programación orientada a objetos se manifiesta notablemente en estas definiciones ya que, se crea una clase que define alguno de los componentes y, posteriormente, se instancia las veces que sea necesario para crear objetos de esa clase con diferentes parámetros. En el caso de resistores y fuentes de tensión o corriente independientes basta con asignarles el valor de la magnitud que representan y asignar los nodos negativo y positivo con los que se conectarán a otros componentes a través de cables. Cuando dos componentes comparten un nodo significa que están conectados entre sí.

Para el caso de fuentes dependientes, éstas estarán definidas por una ganancia o transconductancia multiplicada por la corriente o tensión entre dos puntos, por tanto, para poder crearlas se necesita este parámetro, los nodos de posición con los que se conectan a otros componentes, los dos nodos de control que regulan la tensión o la corriente de la fuente y la magnitud (tensión o corriente) de la que depende.

Por tanto, un circuito quedará definido como un conjunto de componentes interconectados entre sí. Debido a que se trata de una aplicación orientada a fines educativos, se ha intentado que disponga de las funcionalidades necesarias para aprender y que sea lo más sencilla posible; por ello, se ha optado porque trabaje sólo con fuentes de tensión y/o corriente, dependientes y/o independientes como elementos activos y resistencias como elementos pasivos en corriente continua pero ampliable al uso con corriente alterna y otros componentes como condensadores o inductores (impedancias capacitivas o inductivas).

Otra parte del modelo de dominio se centra en la gestión de los usuarios que pueden acceder al módulo de *e-learning*. Para ello, se establecen dos roles: *estudiante y profesor*. Cada uno de ellos puede realizar diferentes acciones dentro de la aplicación. Un profesor puede gestionar la teoría, gestionar ejercicios, gestionar documentos PDF o evaluar a los alumnos; sin embargo, un estudiante solo puede visualizar la teoría publicada por el profesor, leer los documentos disponibles, resolver ejercicios propuestos por el profesor. Consecuentemente, existen clases qué definen qué es una teoría, qué es un ejercicio o cómo se gestiona un documento PDF completando, esto, el submodelo de *e-learning* del modelo de dominio de la aplicación. En las Figuras 4.6 y 4.7 se muestran los diagramas de clases del submodelo de dominio de circuitos y de *e-learning*.

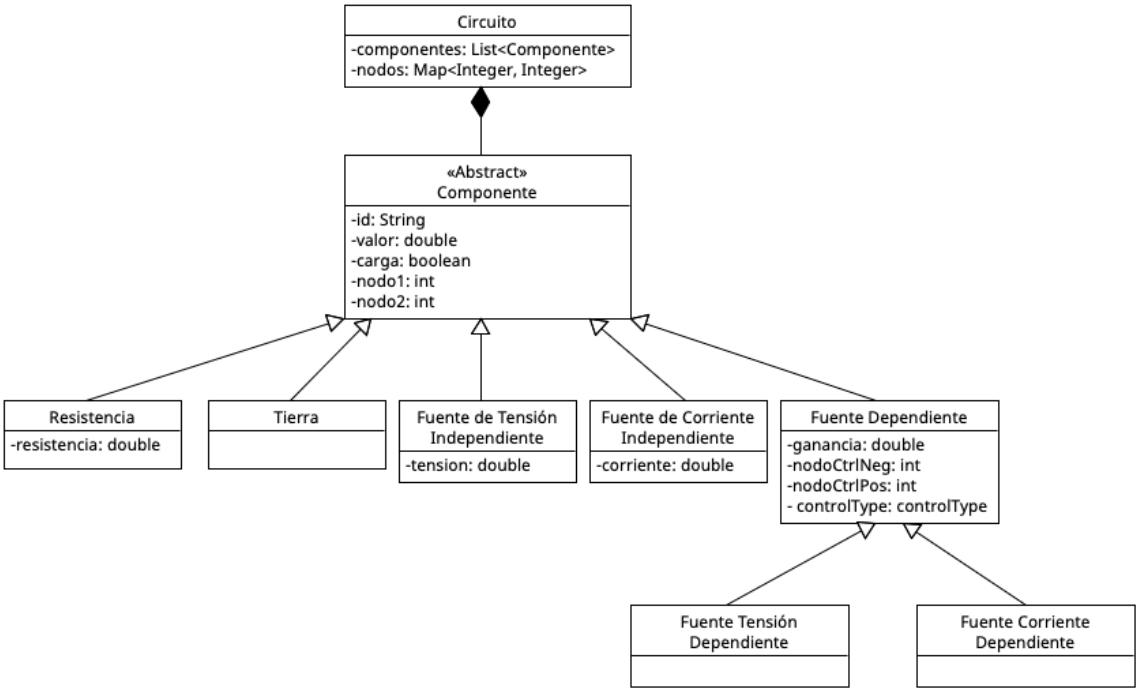


Figura 4.6: Diagrama de clases del submodelo de dominio de circuitos.

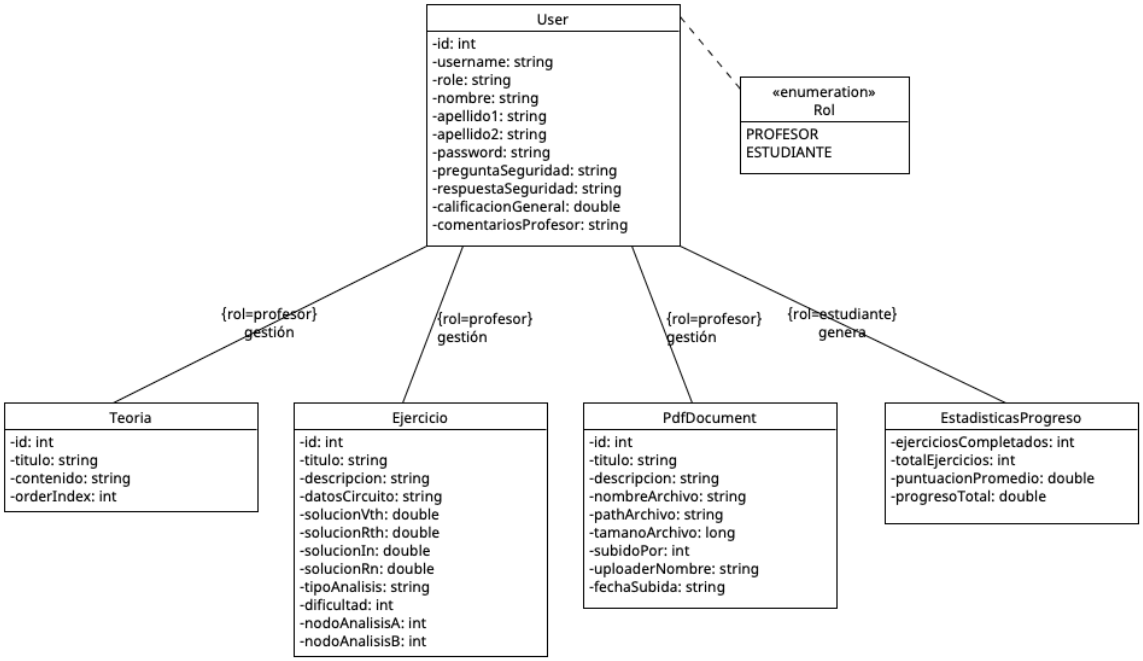


Figura 4.7: Diagrama de clases del submodelo de dominio de *e-learning*.

4.4. Tecnologías utilizadas

Una proyecto como este requiere el uso de diferentes tecnologías. En primer lugar, se debe decidir el tipo de aplicación que se va a crear: si es una aplicación de escritorio o se trata de una aplicación web en función del

público objetivo, cómo se va a utilizar, qué fines persigue, etcétera. Una vez seleccionado, se deberá decidir un lenguaje de programación adecuado que será la base de la aplicación; se debe elegir cómo será y cómo se diseñará la interfaz gráfica de usuario o qué motor de base de datos utilizará. Todas estas decisiones requieren una gran responsabilidad y son una parte fundamental en el desarrollo de software.

4.4.1. Aplicación de escritorio

La elección de una *aplicación de escritorio* para el desarrollo de *ThevenApp* se ha basado en una serie de factores que hacen que sea la modalidad más adecuada para el proyecto por las siguientes razones:

- **Interfaz gráfica avanzada**: la aplicación requiere una interacción avanzada con el usuario para poder arrastrar y soltar componentes, rotarlos, escalarlos o conectarlos mediante cables. Estas razones llevan a seleccionar **JavaFX** [12] para realizar la interfaz gráfica de usuario.
- **Rendimiento**: La aplicación requiere realizar cálculos complejos con matrices para el análisis nodal. Estas operaciones se ejecutan con mejor rendimiento con procesamiento local que con procesamiento a través de un servidor remoto tal como sería en una aplicación web.
- **Conexión a Internet**: En el caso de haber creado una aplicación web, requeriría que el usuario tuviera acceso a Internet para poder ejecutarla. Se ha preferido no tener que depender de este requisito al ser una aplicación educativa.
- **Simplicidad**: Al ser un proyecto desarrollado por una única persona, ésta es una opción más sencilla de elaborar para no tener la necesidad de gestionar servidores backend, bases de datos remotas, autenticación web o despliegues en la nube.

Todas estas razones han llevado a seleccionar la modalidad de *aplicación de escritorio* para el desarrollo de *ThevenApp* por ser la más adecuada tanto desde el punto de vista funcional como desde el de la viabilidad del proyecto.

4.4.2. Java y paradigma orientado a objetos

Figura 4.8: Logotipo de Java [11].

En la Figura 4.8 se ve el logotipo de *Java*, un lenguaje de programación orientado a objetos de alto nivel, propiedad de **Oracle**. Se trata de un lenguaje ampliamente utilizado en entornos profesionales y empresariales y uno de los principales lenguajes que se aprende y utiliza en los grados de informática o de tecnologías de la información.

La *programación orientada a objetos* (POO) es un paradigma de programación que se centra en la manipulación de objetos o instancias de clases para resolver problemas complejos de una manera sencilla y con un código reutilizable, organizado, modular, flexible y adaptable. En lugar de tratar directamente con funciones, tratamos con clases (moldes) e instancias (objetos creados con el molde).

Java es un lenguaje de programación que utiliza, precisamente, este paradigma. Se ejecuta en una máquina virtual JVM [13] lo que implica que se puede ejecutar desde cualquier ordenador (cualquier sistema operativo) lo cual es una gran ventaja para este tipo de aplicación. Además, la JVM actúa como una barrera de seguridad aislando el código del sistema subyacente lo que evita que código malicioso pueda dañar el sistema.

Por otro lado, existe una infinidad de bibliotecas Java que facilitan, enormemente, el desarrollo de aplicaciones complejas. Además, Java cuenta con un recolector de basura que gestiona automáticamente la asignación y liberación de memoria, una tarea que elimina un tedioso trabajo propenso a errores que otros lenguajes como C o C++ no tienen.

En resumen, Java es un lenguaje adecuado para realizar esta aplicación por las siguientes razones:

- Es el lenguaje más utilizado en los grados de ingeniería informática o tecnologías de la información y, por tanto, la mejor opción en cuanto a conocimiento.
- Es un lenguaje orientado a objetos con las ventajas que ello conlleva: reutilizable, organizado, modular, flexible y adaptable.
- A través de la máquina virtual JVM se puede ejecutar en cualquier sistema operativo.
- Es más seguro que otros lenguajes.
- Existen muchas bibliotecas Java que facilitan el trabajo simplificando lo que, de otra manera, serían infinidad de líneas de código complejo.
- Gestión de memoria automático.

4.4.3. Patrones de diseño

Los **patrones de diseño** son soluciones a problemas comunes que surgen a la hora de escribir código. No son bibliotecas o frameworks sino más bien soluciones de código reutilizables que resuelven problemas comunes de forma que el código sea más flexible y legible mejorando la comunicación entre clases e instancias de clases (objetos). Los patrones de diseño se clasifican en tres categorías según su propósito. En la Tabla 4.1 se puede ver una descripción de los principales patrones de diseño [14].

Tabla 4.1: Clasificación y descripción de patrones de diseño.

Categoría	Patrón	Descripción
Creacionales	**Abstract Factory**	Crea familias de objetos relacionados sin especificar clases concretas.
	Builder	Permite construir objetos complejos paso a paso con el mismo código de construcción.
	Factory Method	Define una interfaz para crear objetos, dejando a las subclases el tipo de objeto a crear.
	Prototype	Permite copiar objetos existentes sin depender de sus clases.
	Singleton	Garantiza que una clase tenga una única instancia y proporciona acceso global a ella..
Estructurales	**Adapter**	Permite la colaboración entre objetos con interfaces incompatibles.
	Bridge	Separa una abstracción de su implementación para que evolucionen por separado.
	Composite	Permite tratar objetos individuales y compuestos de forma uniforme.
	Decorator	Añade responsabilidades a un objeto de forma dinámica.
	Facade	Proporciona una interfaz simple para un conjunto de interfaces complejas.
	Flyweight	Reduce el uso de memoria compartiendo instancias similares.
	Proxy	Proporciona un representante de otro objeto para controlar su acceso.

Continúa en la siguiente página

Categoría	Patrón	Descripción
Comportamiento	**Chain of Responsibility**	Envía una petición a lo largo de una cadena de objetos receptores.
	Command	Encapsula una petición como un objeto independiente del emisor o receptor.
	Iterator	Permite recorrer una colección sin exponer su representación interna.
	Mediator	Gestiona la comunicación entre objetos para reducir dependencias.
	Memento	Guarda y restaura el estado interno de un objeto sin violar el encapsulamiento.
	Observer	Notifica a múltiples objetos cuando cambia el estado de otro.
	State	Permite cambiar el comportamiento de un objeto cuando su estado cambia.
	Strategy	Permite seleccionar dinámicamente un algoritmo entre varios posibles.
	Template Method	Define el esqueleto de un algoritmo y deja a las subclases redefinir pasos.
	Visitor	Permite agregar nuevas operaciones a una estructura de objetos sin modificarla.

Para el desarrollo de la aplicación se han usado algunos de estos patrones de diseño para mejorar el código y hacerlo más sencillo.

- **Factory**: Se ha utilizado este patrón de diseño para facilitar la creación de instancias de componentes como resistencias o fuentes de tensión y de corriente. El uso de este patrón permite crear instancias de un componente con un código más limpio y desacoplado, permite extender esta creación a nuevos componentes en el futuro (condensadores, bobinas, etcétera), reutilización en la creación del objeto de dominio y su representación visual y mejor mantenimiento. Además, crear instancias para pruebas se hace más sencillo sin tener que tocar el resto del código.
- **Builder**: Se ha utilizado el patrón *Builder* para la creación de cables. Con este patrón se hace más simple y legible la línea de código para crear cables.
- **Singleton**: Se ha utilizado el patrón *Singleton* en dos áreas críticas en las que se necesita garantizar la unicidad de las instancias. Por un lado, se aplica al componente de Tierra en el circuito para asegurar una única referencia en el circuito (potencial cero). También se utiliza en el gestor de operaciones de deshacer/rehacer (*Undo/Redo*). Una única instancia es fundamental para mantener la pila de estados de una forma coherente, ya que, si hubiera más instancias podrían provocar inconsistencias si cada una intentara gestionar el historial de estados o acciones de forma simultánea.
- **Command**: Se ha utilizado el patrón *Command* para implementar la función *deshacer/rehacer* en la aplicación.
- **Composite**: Se utiliza el patrón *Composite* para operaciones que requieren emplear varios *command* a la vez como, por ejemplo, operaciones deshacer/rehacer de varios componentes al mismo tiempo.
- **Adapter**: Se utiliza el patrón *Adapter* para modificar las propiedades de los componentes. Es útil tener toda la lógica de gestión (cálculo, cuadros de diálogo, etc.) en una clase con métodos estáticos; cuando el controlador necesita modificar un parámetro llama a los métodos de la interfaz siendo el adaptador quien delega la comunicación con la clase original. Esto resulta un avance, porque si se quisiera modificar la forma en que se modifican los parámetros de componentes en la GUI no sería necesario tocar el controlador que hace la llamada sino, únicamente, utilizar otro adaptador. Esto permite reducir el acoplamiento entre la lógica de control y la implementación de la interfaz de usuario.
- **Observer**: El patrón *Observer* se utiliza frecuentemente en la aplicación. Ciertos comportamientos en la interfaz dependen, precisamente, de "escuchar" cambios que se producen en otros lugares. Los interruptores o *checkbox* de la aplicación implementan el patrón ya que de ellos depende, por ejemplo, mostrar

u ocultar etiquetas o puntos de control, o mostrar u ocultar la captura del circuito original dentro del panel de *Análisis*. Además, los movimientos de componentes deben ser "*escuchados*" por los cables y las etiquetas así como los cambios en las propiedades de los componentes hacen que se actualicen ciertos comportamientos.

- **Facade**: El patrón *Facade* o *Fachada* se utiliza en la aplicación para ofrecer una interfaz simplificada de algunas partes complejas. Se utiliza en la gestión de ventanas emergentes de forma que se permita abrir diálogos creados en archivos FXML con solo llamar a un método estático. Por otra parte, también se aplica en el renderizado de circuitos equivalentes donde hay una clase que se encarga de realizar toda la lógica (creación de componentes, creación de cables, etc.) de forma que los algoritmos de Thévenin y Norton puedan dibujar sus esquemas sin conocer los detalles de implementación del esquema.

4.4.4. JavaFX para interfaz gráfica

JavaFX [15] es una plataforma de desarrollo para construir interfaces gráficas de usuario (GUI) en aplicaciones Java. Inicialmente, fue desarrollada por *Sun Microsystems* y, posteriormente, mantenida por *Oracle*; sin embargo, actualmente se distribuye de forma independiente y de código abierto con el nombre de **OpenJFX**.

Los diseños realizados con JavaFX son modernos, visualmente muy atractivos y muy interactivos. Una de las grandes ventajas de JavaFX es que tiene un diseño modular; separa la lógica y el diseño, lo que se integra perfectamente con el patrón MVC (*Modelo-Vista-Controlador*).

Además, JavaFX utiliza tecnologías como FXML (lenguaje de marcado) para crear interfaces declarativas y CSS (hojas de estilo) para definir estilos propios tal como se haría en un documento HTML, lo que permite una gran flexibilidad. Ofrece un conjunto completo de componentes visuales tales como botones, paneles, menús, etc., soporte para animación 2D/3D y otras muchas opciones. Se compone de 3 capas principales:

1. **Stage** o escenario: ventana principal de la aplicación que contiene al resto de componentes.
2. **Scene** o escena: es el contenedor de todos los elementos visuales dentro del *Stage*.
3. **Node** o nodo: son los elementos gráficos que están dentro de una escena tales como botones, menús, etc.

En la aplicación *ThevenApp* se ha empleado JavaFX para construir toda la interfaz gráfica de usuario, obteniendo, así, una aplicación moderna, sencilla y eficiente. Cada panel se ha realizado escribiendo un archivo FXML declarando todos los componentes gráficos que lleva y asociándolo a un controlador escrito en Java que implementa la interacción con el usuario. Esta estructura aplica el patrón MVC separando la *vista* (FXML) del *controlador*.

Los paneles que conforman la aplicación son los siguientes:

- **Panel Principal**: es la ventana principal de la aplicación y la que contiene al resto. En ella está la barra de menús con los desplegables habituales (*Archivo, Edición, Ayuda*), las pestañas que permiten intercambiar los paneles principales y un pequeño panel donde se pueden seleccionar ciertas opciones visuales. Es un *BorderPane* que contiene los menús y un *AnchorPane* dentro del cual van el resto de paneles.
- **Panel de Bienvenida**: es un panel (*HBox*) con el que inicia la aplicación que contiene el logotipo y la versión de la aplicación y contiene ciertos enlaces (en un *VBox*) para iniciar con un circuito nuevo, cargar desde archivo y con un menú de archivos recientes.
- **Panel de Diseño**: es uno de los tres paneles principales de la aplicación. Contiene un panel dividido (*SplitPane*) donde se encuentra el área de dibujo (*AnchorPane*) en el que se pueden colocar componentes (*StackPane*) o realizar las conexiones y un **panel de propiedades** (*AnchorPane*) donde se pueden modificar los parámetros de un componente y que contiene varios *GridPane* para dividir secciones de modificación según el tipo de componente seleccionado y varias etiquetas (*Label*) o hipervínculos (*Hyperlink*).
- **Panel de Análisis**: es otro de los tres paneles principales de la aplicación. Contiene un panel dividido (*SplitPane*) donde se encuentra un área de dibujo (*AnchorPane*) que es donde se dibujará el circuito equivalente seleccionado y un **panel informativo** similar al panel de propiedades del área de diseño pero que en este caso se usa para dar información sobre el análisis realizado.

- **Panel de *E-Learning***: es el tercer panel del grupo de paneles principales. Este panel es el más completo de todos. Es un panel *AnchorPane* en cuyo interior se albergan varios *VBox* con diferentes contenidos como etiquetas, cuadros de texto, botones, barras de progreso o selectores de opciones.
- **Panel de Componentes**: es un panel que sólo está activo cuando el usuario se encuentra en el panel de diseño. Se trata de un *ScrollPane* que contiene los componentes en forma de *StackPane* arrastrables al área de dibujo.
- **Panel de Opciones de Análisis**: es un panel que sólo está activo cuando el usuario se encuentra dentro del panel de análisis. Se trata de un *VBox* que contiene las opciones necesarias para realizar el análisis. Contiene dos botones (*Button*) para seleccionar el tipo de análisis a realizar, un *checkbox* para mostrar u ocultar el cuadro con la miniatura del circuito original y dos *combobox* para seleccionar los nodos de análisis cuando están disponibles.
- **Panel de navegación de *e-learning***: este panel se corresponde con un menú en la parte izquierda de la ventana principal que se activa cuando el usuario se encuentra en el área de *e-learning*. Está formado por un *VBox* que contiene diversos botones y etiquetas.

Esta interfaz se organiza en pestañas (*TabPane*) que permiten seleccionar uno de los tres paneles principales[1] y así trabajar con la aplicación en un modo u otro.

En la aplicación se emplean mecanismos de *observable bindings* y *listeners* para controlar el comportamiento dinámico tal como actualizar etiquetas al modificar un parámetro, actualizar posiciones al mover un componente, habilitar o deshabilitar botones, etcétera.

El uso de JavaFX se justifica, por tanto, por diferentes motivos que se exponen a continuación:

- La separación entre vista (FXML) y lógica (Controller) favorecen el patrón MVC y proporcionan modularidad y un diseño moderno.
- JavaFX proporciona soporte para eventos de ratón, teclado, *drag-n-drop*, etc. lo que favorece la interacción para el usuario.
- El estilo es totalmente personalizable con hojas de estilo CSS sin tener que modificar código Java.
- No depende de conectividad ni de APIs externas para funcionar a diferencia de las aplicaciones web.
- El resultado es una aplicación de escritorio multiplataforma que se podrá usar en Windows, Linux o MacOS.

4.4.5. Base de datos

La sección de *e-learning* de la aplicación requiere diferenciar los roles de estudiante y profesor para que, los primeros puedan acceder a contenidos y ejercicios sobre el Análisis de Circuitos y llevar un seguimiento de su evolución y los segundos deben poner a disposición de los estudiantes el material que consideren oportuno, ejercicios de dificultad creciente o, incluso, evaluar y comentar a los estudiantes acerca de su evolución. Para ello, es necesario una *base de datos* que gestione el registro y el acceso de usuarios registrados al contenido.

SQLite es un motor de base de datos relacional, ACID y auto-contenido que funciona embebido en la aplicación. Los datos se almacenan en un único fichero en disco y se accede a ellos mediante la librería del propio acceso de la app. En la Figura 4.9 se puede ver el logotipo del motor SQLite.

Figura 4.9: Logotipo de SQLite [16].

SQLite presenta varias razones por las que es la opción más conveniente para una aplicación de escritorio:

- No requiere, apenas, configuración. No necesita instalación ni administración de servidores.
- Portabilidad. Se trata de un archivo que se puede copiar, versionar y mover entre dispositivos.
- Bajo consumo. No requiere gran consumo de recursos

[1] Panel de diseño, panel de análisis y panel de aprendizaje

y tiene un rendimiento ideal en monousuario garantizando consistencia y durabilidad de los datos.
- Se puede distribuir la aplicación con una base de datos pre-cargada, por ejemplo, con ejercicios o teoría.
- No requiere acceso a Internet.

Existen otras opciones de bases de datos muy populares que se han rechazado por los siguientes motivos:

- **PostgreSQL o MySQL/MariaDB**: son motores cliente-servidor que requieren acceso a Internet, configuraciones de red o puertos y continuo mantenimiento. Sería ideal para multiusuario o para una alta concurrencia pero este no es el caso de esta aplicación.
- **H2 o Apache Derby**: tiene una fuerte dependencia de la Máquina Virtual de Java (JVM), el formato, si es antiguo, puede requerir migraciones y no tiene tantas herramientas para gestionarla.
- **Formatos no relacionales**: este tipo de formatos tales como JSON, CSV u otro tipo de archivos planos no facilitan las transacciones, las consultas son limitadas y no son consistentes. Aunque JSON se utiliza para almacenar circuitos en disco, no es útil para la gestión de usuario y ejercicios.

Todas estas razones llevan a decidir que SQLite es una elección coherente y eficiente con una buena integración con JavaFX y, por tanto, es el motor de base de datos que utiliza la aplicación.

4.5. Diseño de la interfaz de usuario

La interfaz de usuario se ha diseñado con la intención de que sea sencilla y moderna pero, sobre todo, fácil de utilizar para cualquier usuario. El hecho de que se trate de una aplicación orientada a fines educativos requiere que se pueda utilizar sin necesidad de pasar horas aprendiendo sus utilidades y no una aplicación compleja en la que el estudiante deba consultar manuales de usuario, tutoriales u otros recursos para poder utilizarla. Con este objetivo se ha realizado un diseño basado en tres secciones principales tal como se indica en la Sección 4.1.

4.5.1. Inicio de la aplicación

La aplicación se inicia con un **panel de bienvenida** (Figura 4.10) donde se muestra el logotipo de la aplicación y desde donde podemos acceder a las funciones principales a través de diferentes atajos que llevan a:

- *Crear un nuevo circuito*: permite iniciar la aplicación con un nuevo circuito, con el área de diseño completamente limpia para comenzar a realizar un diseño nuevo.
- *Cargar circuito*: permite cargar un circuito almacenado en un archivo JSON local. Esta acción abre un cuadro de diálogo del explorador de archivos para seleccionar el archivo deseado.
- *Archivos recientes*: muestra una pequeña lista con los últimos archivos JSON utilizados en la aplicación ordenados por última modificación. Esto mejora significativamente la usabilidad de la aplicación permitiendo al usuario tener un acceso directo a sus últimos trabajos.
    - *Borrar historial*: opción que permite al usuario borrar todo el historial de archivos recientes.
- *E-Learning*: se incluye un acceso directo a la sección de *e-learning* con la intención, también, de mejorar la usabilidad y evitar así, que el propio usuario tenga que navegar por otras secciones para acceder al contenido teórico o a los ejercicios.

De forma invisible de cara al usuario, cuando se entra en la aplicación, ésta crea una carpeta (si no existiera ya) en el directorio "home" del usuario (según el Sistema Operativo) de nombre *"ThevenApp"*. Dentro de ella se encuentra el archivo de la base de datos con el nombre *"thevenapp-database.db"* y dos carpetas diferentes una de nombre *"pdfs"* que es donde se almacenan los PDFs subidos por el profesor desde la plataforma de *e-learning* y otra de nombre *"logs"* donde se almacenan los archivos de log generados por la aplicación.

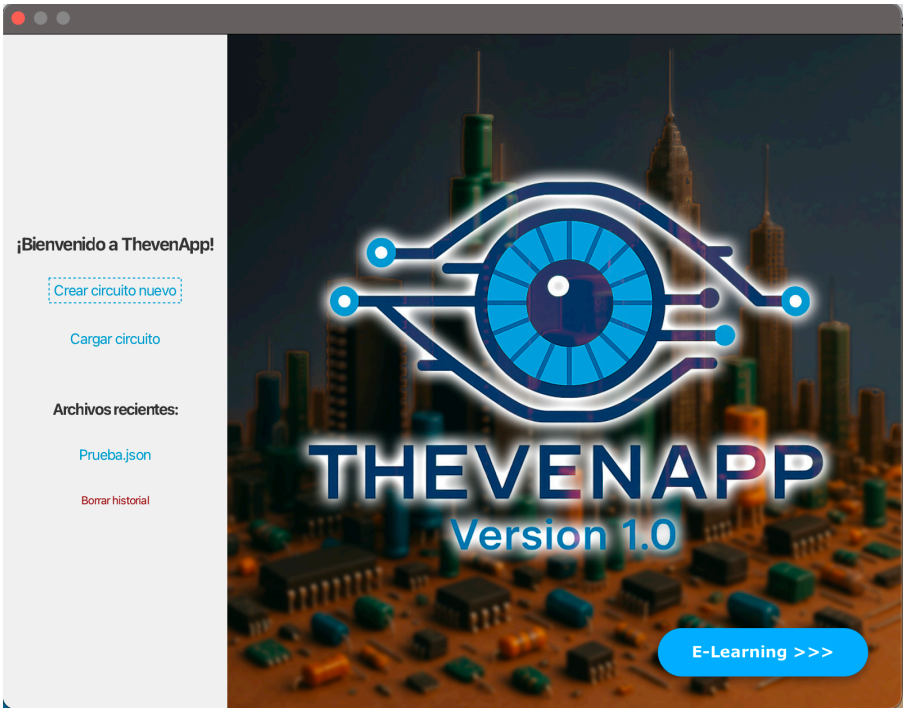


Figura 4.10: Imagen del panel de bienvenida de inicio a la aplicación.

4.5.2. Diseño de circuitos

El área de diseño (Figura 4.11) es la sección donde el usuario podrá realizar sus esquemas; podrá, tanto crear un circuito nuevo como modificar un circuito, previamente, guardado en un archivo. Esta sección se compone de tres áreas:

- **Panel de Diseño**: es el panel principal de esta sección. Se trata de un panel tipo lienzo (*canvas*) donde se podrán añadir los componentes, realizar las conexiones con cables y las acciones permitidas tales como mover, eliminar, escalar o rotar componentes.
- **Panel de Componentes**: este panel está ubicado, visualmente, a la izquierda del panel de diseño y contiene imágenes de los componentes disponibles (resistencia, fuente de tensión independiente, fuente de corriente independiente, fuente de tensión dependiente, fuente de corriente dependiente y tierra). Desde aquí, se puede hacer clic sobre la imagen de un componente y arrastrarlo hasta el panel de diseño para agregar uno de estos componentes al circuito.
- **Panel de Propiedades**: este panel, por defecto, permanece oculto hasta que se selecciona un elemento del área de diseño (componentes o cables). Al hacerlo, se abre y muestra la información y/o permite modificar todos los parámetros asociados al elemento seleccionado. Las propiedades editables dependen del elemento seleccionado:
  - ***Resistencia***:
    - Permite modificar su valor óhmico.
    - Permite marcar/desmarcar como componente de carga.
    - Informa de los nodos a los que está conectada pero no permite modificación ya que el sistema los asigna automáticamente.
  - ***Fuente de Tensión Independiente***:
    - Permite modificar su valor de tensión.
    - Informa de los nodos a los que está conectada pero no permite modificación ya que el sistema los asigna automáticamente.
    - Si se intenta marcar como componente de carga genera una alerta explicando que esta acción no tiene sentido físico.
  - ***Fuente de Corriente Independiente***:

- Permite modificar su valor de corriente.
- Informa de los nodos a los que está conectada pero no permite modificación ya que el sistema los asigna automáticamente.
- Si se intenta marcar como componente de carga genera una alerta explicando que esta acción no tiene sentido físico.

- ***Fuente de Tensión Dependiente***:
  - Permite modificar su ganancia, nodos de control y magnitud de control (tensión o corriente).
  - Informa de los nodos a los que está conectada pero no permite modificación ya que el sistema los asigna automáticamente.
  - Si se intenta marcar como componente de carga genera una alerta explicando que esta acción no tiene sentido físico.
- ***Fuente de Corriente Dependiente***:
  - Permite modificar su ganancia, nodos de control y magnitud de control (tensión o corriente).
  - Informa de los nodos a los que está conectada pero no permite modificación ya que el sistema los asigna automáticamente.
  - Si se intenta marcar como componente de carga genera una alerta explicando que esta acción no tiene sentido físico.
- ***Tierra***:
  - Informa del nodo de tierra (potencial 0) pero no permite modificación ya que es asignado automáticamente al nodo 0.
- ***Cable***:
  - Informa sobre los nodos qué conecta y los componentes asociados.

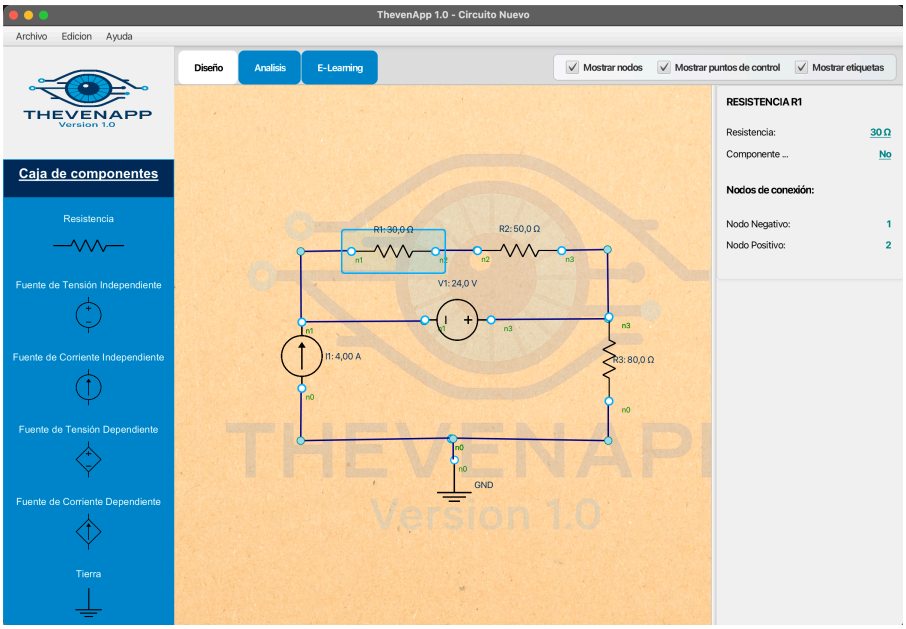


Figura 4.11: Sección de Diseño de la aplicación.

En la Figura 4.11 se puede observar el panel de componentes a la izquierda, el panel de diseño en el centro y el panel de propiedades a la derecha.

4.5.3. Análisis de circuitos

La sección de Análisis (Figura 4.12) se centra en las opciones necesarias para cumplir con el objetivo para el que fue diseñada. En este área se ejecuta el análisis del circuito diseñado según el tipo de análisis seleccionado

por el usuario y muestra, gráficamente, este circuito equivalente de una forma didáctica. Al igual que el panel de diseño se divide en tres áreas bien diferenciadas:

- ***Selección de opciones***: se encuentra a la izquierda de la sección de Análisis y dispone de dos botones que permiten seleccionar el tipo de análisis a realizar: THÉVENIN o NORTON. Si, a la hora de realizar el diseño, se ha marcado algún componente como carga, ésta será detectada por el sistema de análisis y no requerirá seleccionar, manualmente, los nodos de análisis. Si esto no fuera así, se desbloquearán dos selectores de nodos donde el usuario podrá elegir los nodos sobre los que se realiza el análisis. Antes de realizarlo, el propio sistema hace una validación del circuito contemplando posibles situaciones imposibles (como el caso de matriz singular por dos fuente de tensión diferentes en paralelo o dos fuentes de corriente diferentes en serie, etcétera). Además, esta sección incluye un interruptor (checkbox) que permite activar/desactivar una pequeña miniatura con la imagen del circuito original.
- ***Panel de análisis***: en esta panel dispuesto en la parte central de esta sección es donde se muestra el circuito equivalente, una vez realizados los cálculos. Se trata de un panel, meramente, informativo que no permite modificación. Con el objetivo de mejorar la usabilidad y que el alumno pueda comparar ambos circuitos se muestra arriba a la izquierda una miniatura con una imagen del circuito original que se podrá mover, ocultar o ampliar.
- ***Descripción del análisis***: es un panel similar al panel de propiedades del área de diseño que muestra una completa descripción del análisis realizado obteniendo los valores clave del análisis (resistencia equivalente y tensión en el caso de Thevenin o corriente en el caso de Norton); además ofrece un resumen del circuito obtenido.

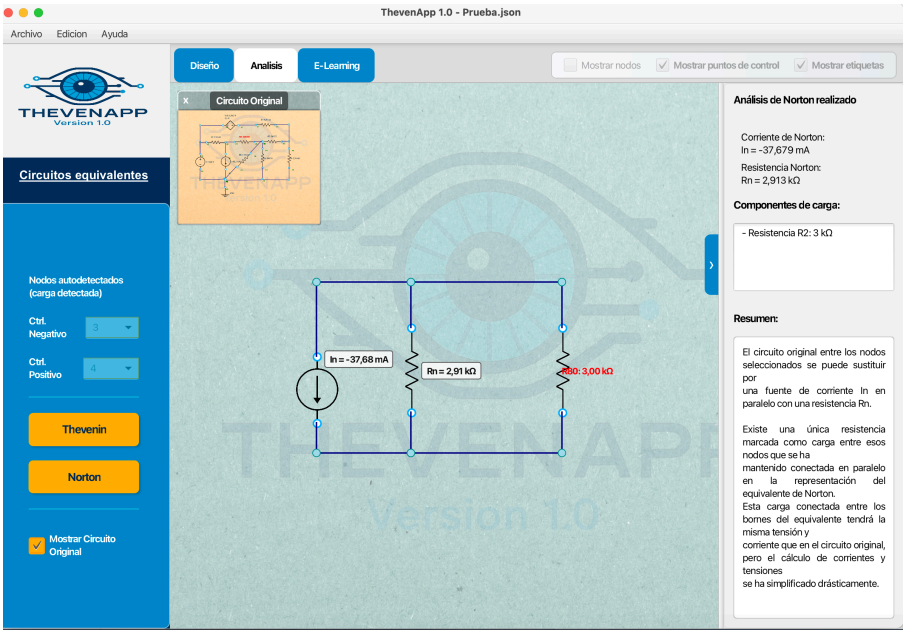


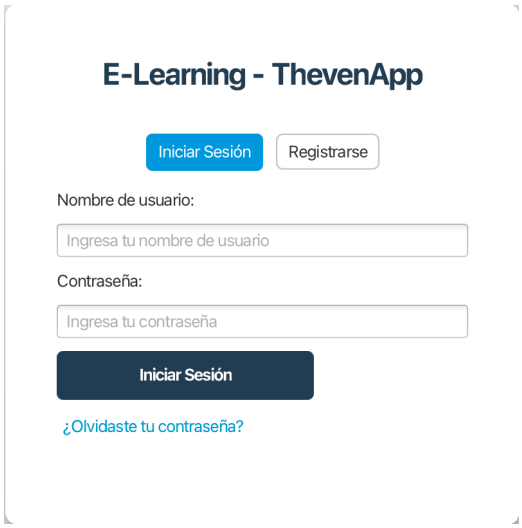
Figura 4.12: Sección de Análisis de de la aplicación.

En la Figura 4.12 se puede observar el panel de opciones de análisis a la izquierda, el panel de análisis en el centro con un circuito equivalente de Norton y la miniatura del circuito original en la parte superior izquierda; a la derecha vemos el panel con toda la información del análisis realizado.

4.5.4. *E-Learning*

La sección de *e-learning* (Figuras 4.14 y 4.15) es una plataforma didáctica completa donde los usuarios interactúan en base a un rol seleccionado durante el registro en la base de datos y que puede ser de ESTUDIANTE o PROFESOR.

Cuando un usuario accede a esta sección, accede a un panel de login (Figura 4.13a) donde pide el nombre de usuario y la contraseña para acceder al sistema. En el caso de que el usuario no esté registrado se mostrarán nuevos campos para iniciar un registro en la aplicación (Figura 4.13b).



(a) Panel de login

(b) Panel de registro

Figura 4.13: Paneles de login y registro en la aplicación.

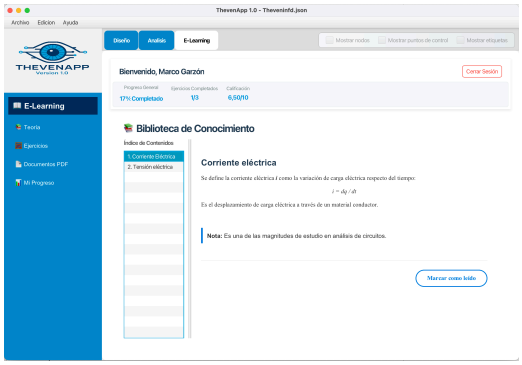
Una vez autenticado en la aplicación, la sección de *e-learning* mostrará diferentes opciones según el rol del usuario.

4.5.4.1. Rol de estudiante

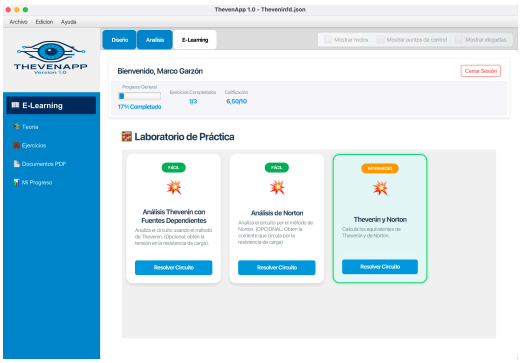
Cuando un usuario accede a la plataforma con el rol de estudiante se le habilitan opciones que le permitirán conocer y aprender contenidos teóricos actualizados y resolver ejercicios propuestos por su profesor.

- *Bienvenida:* nada más entrar al sistema, el alumno visualiza una pantalla de bienvenida con un menú de navegación a la izquierda desde donde puede acceder a la teoría, a los ejercicios, a los documentos PDF o a revisar su progreso. Además en la parte superior dispone de un resumen del progreso realizado.
- *Teoría:* en esta sección verá una lista de los temas disponibles y, una vez seleccionado uno, verá su contenido en un panel a la derecha (ver Figura 4.14a).
- *Ejercicios:* en esta sección tendrá los ejercicios disponibles en forma de tarjetas indicando el título del ejercicio y la dificultad. Cuando accede a un ejercicio se le muestra el enunciado y le llevará al panel de diseño donde estará el circuito diseñado por el profesor que deberá resolver manualmente y tendrá una ventana de diálogo en la que debe introducir los valores calculados correctamente para validar el ejercicio. Cuando un ejercicio ha sido resuelto su tarjeta se pondrá de color verde (ver Figura 4.14b).

- *Documentos PDF:* se muestra una lista de documentos PDF con contenido adicional subidos por el profesor y que el alumno podrá descargar y visualizar en un visor externo (ver 4.15a).
- *Mi progreso:* el alumno podrá visualizar en todo momento su progreso, ver su calificación y leer comentarios enviados por su profesor (ver figura 4.15b).

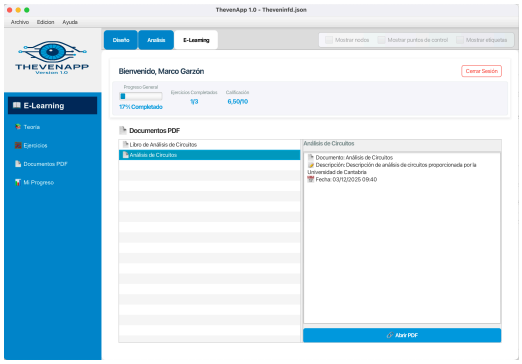


(a) Contenido teórico disponible.

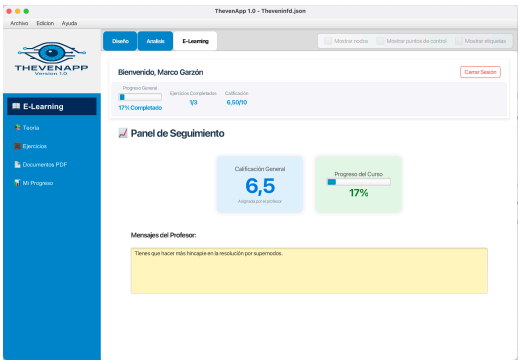


(b) Ejercicios disponibles

Figura 4.14: Secciones de teoría y ejercicios para un usuario con rol de estudiante.



(a) Documentos PDF disponibles.



(b) Progreso de un estudiante

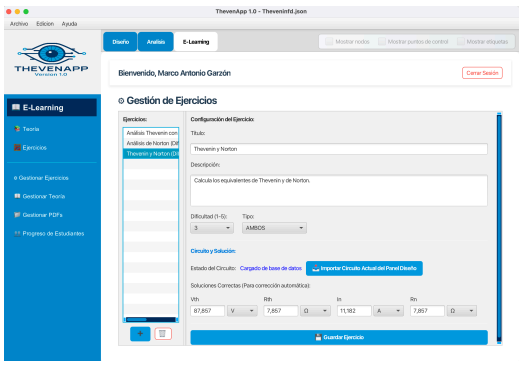
Figura 4.15: Secciones de documentos PDF y progreso para un usuario con rol de estudiante.

4.5.4.2. Rol de profesor

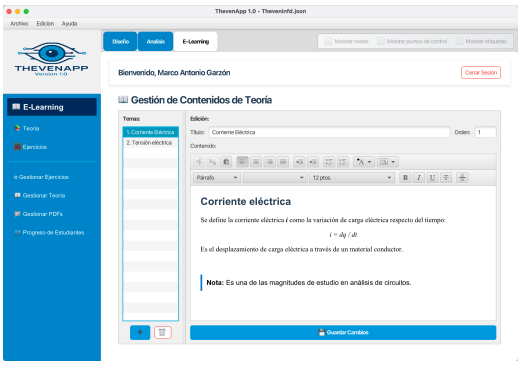
Un usuario con rol de *profesor*, además de visualizar la teoría y los ejercicios tal como lo hace un estudiante, puede gestionar todo el contenido de la plataforma. Para ello, cuenta con algunas opciones más en el menú de navegación tales como *Gestionar Teoría*, *Gestionar Ejercicios*, *Gestionar PDF* o *Progreso de estudiantes*.

- *Gestionar Teoría*: en esta sección, el profesor tendrá disponible un listado de los temas subidos donde podrá añadir nuevos temas o eliminar los existentes. Para crear uno nuevo tendrá disponible un editor HTML con una plantilla disponible de uso opcional donde podrá escribir el tema tal como considere oportuno, añadir imágenes, resaltar textos, etcétera (ver Figura 4.16b).
- *Gestionar Ejercicios*: el profesor tendrá disponible un listado de los ejercicios propuestos donde podrá añadir nuevos ejercicios o eliminar los existentes. Para crear un nuevo ejercicio deberá ir al panel de *Diseño* y crear su propio esquema para, posteriormente, desde esta sección importar el circuito donde el sistema calculará los valores que serán solución del ejercicio (tensión y resistencia en el caso de Thévenin y corriente y resistencia en el caso de Norton), escribirá el enunciado y la descripción y establecerá una dificultad para el ejercicio (ver Figura 4.16a).
- *Gestionar PDF*: en esta sección, el profesor podrá visualizar un listado de documentos PDF ya subidos y podrá eliminarlos o subir documentos nuevos añadiéndoles un título y una descripción (ver Figura 4.17a).

- *Progreso de estudiantes*: el profesor tendrá el listado de estudiantes registrados en la aplicación y podrá ver el progreso de cada uno de ellos. Además podrá ponerles una calificación o enviarles comentarios que podrá leer el alumno (ver Figura 4.17b).

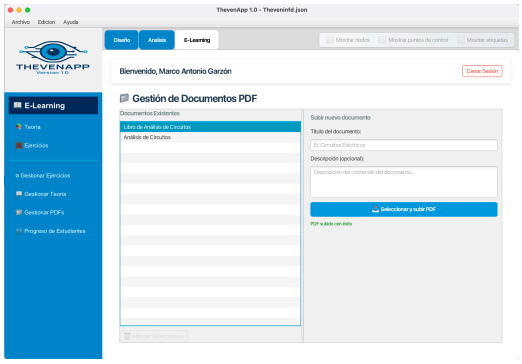


(a) Gestión de Ejercicios

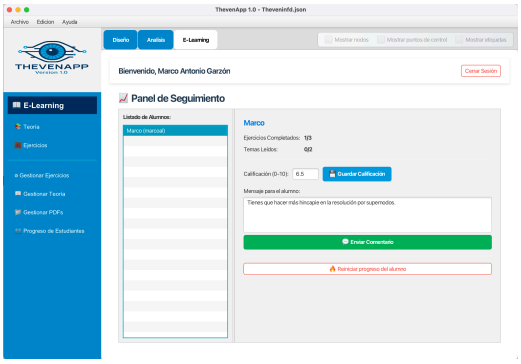


(b) Gestión de Teoría

Figura 4.16: Secciones de Gestión de Ejercicios y Gestión de Teoría disponibles para usuario con rol de profesor.



(a) Gestión de Documentos PDF



(b) Gestión de Progreso de Estudiantes

Figura 4.17: Secciones de Gestión de Documentos PDF y de Progreso de los estudiantes disponibles para usuario con rol de profesor.

4.5.5. Paneles superiores

La parte superior de la aplicación contiene la clásica barra de menús que, en este caso, tiene 3 opciones:

1. **Archivo**:
   - *Nuevo*: permite crear un circuito nuevo en el área de diseño. (Atajo Cmd+N o Ctrl+N).
   - *Abrir recientes*: despliega un menú lateral con los últimos archivos abiertos que permite abrir cualquiera de ellos y la última opción permite borrar la lista con *Borrar historial*.
   - *Cargar*: permite cargar un diseño previamente guardado en un archivo JSON en disco. (Atajo Cmd+O o Ctrl+O).
   - *Guardar*: permite guardar el diseño actual en un archivo JSON. (Atajo Cmd+S o Ctrl+S).
   - *Guardar como*: permite guardar una copia del diseño actual en un archivo JSON con diferente nombre. (Atajo Shift+Cmd+S o Shift+Ctrl+S).
   - *Salir*: permite salir de la aplicación. (Atajo Cmd+Q o Ctrl+Q).
2. **Edición**:
   - *Deshacer*: deshace las últimas acciones realizadas en el diseño ordenadas en una pila tipo LIFO (*Last-In-First-Out*). (Atajo Cmd+Z o Ctrl+Z).

- *Rehacer*: rehace las últimas acciones deshechas en el diseño ordenadas en una pila tipo LIFO. (Atajo Cmd+Y o Ctrl+Y).
- *Seleccionar todo*: permite seleccionar todos los elementos existentes en el área de diseño. (Atajo Cmd+A o Ctrl+A).
- *Deseleccionar todo*: permite deseleccionar cualquier elemento que estuviera seleccionado en el área de diseño. (Atajo Click izquierdo en cualquier área vacía).
- *Limpiar área de dibujo*: elimina todos los elementos del área de diseño. (dAtajo Shift+Cmd+C o Shift+Ctrl+C).

3. **Ayuda**:
   - *Ver logs*: muestra el visor de logs.
   - *Ver Manual de Usuario (PDF)*: muestra un documento PDF con el manual de usuario en un visor de PDF externo.
   - *Acerca de*: muestra la información sobre la aplicación, la versión actual y el/los creadores.

Además en la parte superior derecha hay un panel para activar o desactivar ciertas opciones visuales compuesto por 3 *checkbox*. Este panel sólo está activo si el usuario se encuentra en el panel de *Diseño*, ya que en el panel de *Análisis* o de *e-learning* no son necesarios. Estos tres interruptores permiten:

- Activar/Desactivar etiquetas de componentes.
- Activar/Desactivar etiquetas de nodos.
- Activar/Desactivar puntos de control.

4.6. Diseño de la base de datos

La aplicación *ThevenApp* incluye una base de datos SQLite para la gestión de la plataforma de *e-learning* [17]. Se ha tomado la decisión de no incluir diseños esquemáticos de circuitos en la base de datos porque requeriría una base de datos mucho más grande y podría no gestionar adecuadamente la reconstrucción de los circuitos en el área de diseño. Para esto se ha utilizado con mucho mejor resultado el almacenamiento en archivos JSON. Sin embargo, la base de datos es de gran utilidad para la gestión de usuarios en la sección de *e-learning*, gestión de teoría, gestión de ejercicio, de documentos pdf o de progreso de los estudiantes. Por ello se ha creado una base de datos local con la tecnología SQLite que permitirá con un único archivo poder trabajar de la misma manera que lo haríamos en una base de datos remota sin la necesidad de configurar un servidor para ello y con la ventaja de que podemos transportar el archivo entre diferentes computadoras para tener todo el contenido de ella disponible.

La base de datos utiliza una arquitectura clásica JDBC (Java DataBase Connectivity) que es muy popular cuando se utiliza el lenguaje Java, montada sobre el motor de SQL SQLite. Se ha implementado en un modelo de dos niveles, es decir, la aplicación se comunica directamente con la base de datos a través de la API (Interfaz de Programación de Aplicaciones) y el controlador JDBC.

Uno de los puntos fuertes es la facilidad con la que se crea una base de datos para una aplicación Java de esta forma, ya que basta con importar las librerías *sql* (JDBC), cargar la clase `org.sqlite.JDBC` y escribir el código SQL para crear las tablas y, posteriormente, realizar las consultas.

El funcionamiento es el siguiente:

- Clases que llaman a la base de datos: son las que realizan las consultas (usuarios, teoría, ejercicios, pdf o progreso).
- JDBC prepara la consulta en un formato adecuado.
- Driver SQLite traduce la consulta al formato de SQLite.
- SQLite lee el archivo almacenado con la base de datos, obtiene el dato y lo devuelve al Driver quien, a su vez, lo manda al JDBC que adapta, de nuevo, el formato para que la clase Java que realizó la consulta disponga del dato.

Para esta aplicación, se han creado un total de 6 tablas en la base de datos (ver Figura 4.18):

1. **Tabla de usuarios**: almacena los usuarios registrados con los campos *id, username, password_hash, role, nombre, apellido1, apellido2, pregunta_seguridad, respuesta_seguridad, calificacion_general, comentarios_profesor y fecha_creacion*. Un dato importante es que el campo *contraseña* no almacena la contraseña propiamente sino el *hash* para obtener la contraseña cifrada con SHA-256.
2. **Tabla de teoría**: almacena el contenido teórico de la aplicación con los campos *id, titulo, contenido, indice_orden, creado_por, fecha_creacion*.
3. **Tabla de ejercicios**: almacena los ejercicio creados por el profesor con los campos *id, titulo, descripcion, datos_circuito, solucion_vth, solucion_rth, solucion_in, solucion_rn, tipo_analisis, dificultad, creado_por, nodo_a, nodo_b, fecha_creacion*.
4. **Tabla de documentos PDF**: almacena los documentos PDF subidos por el profesor con los campos *id, titulo, descripcion, nombre_archivo, path_archivo, tamano_archivo, subido_por, fecha_subida*.
5. **Tabla de progreso de estudiantes**: almacena el progreso de los estudiantes registrados en la aplicación con los campos *id, estudiante_id, ejercicio_id, completado_fecha, puntuacion, intentos, tiempo*.
6. **Tabla de teoría leída**: almacena los id de los tema leídos por un usuario concreto. Para ello usa los campos *user_id, teoria_id, fecha_lectura*.

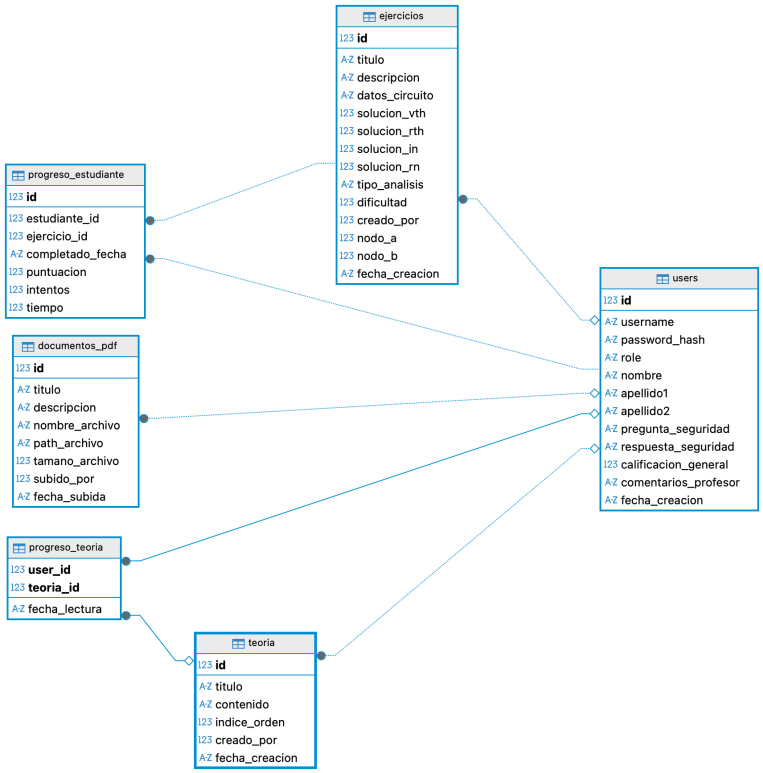


Figura 4.18: Esquema de la base de datos de la aplicación.

4.7. Gestión de logs

Una parte importante en el diseño de la aplicación se ha centrado en garantizar la mantenibilidad y depuración de errores; por ello, se ha diseñado un completo sistema de *logs* con diversos objetivos:

- Facilitar, durante el desarrollo, la depuración de errores.
- Facilitar al usuario una herramienta que puede usar para comunicar errores que le puedan haber surgido.
- O simplemente ofrecer una visión técnica de las operaciones que realiza la aplicación en base a las peticiones de los usuarios.

Se ha usado `java.util.logging`, en primer lugar, porque el hecho de pertenecer al JDK evita el tener que usar

librerías externas con las ventajas que conlleva tales como evitar errores en el despliegue de la aplicación, mejorar el tamaño del archivo final y, sobre todo, porque se integra muy fácilmente con Java ya que, los comandos de log son similares a escribir comandos de salida por consola (`System.out.println()`). Se ha utilizado también la clase *Level* del paquete *util* (`java.util.Level`) que permite escribir mensajes de log en diferentes niveles tales como:

- **INFO**: Información general de operaciones en la aplicación.
- **WARNING**: Advertencias en operaciones con riesgo.
- **SEVERE**: Fallos graves en la aplicación.
- **FINE-FINER**: Nivel de depuración más detallado.

Todas las acciones que producen algún cambio de estado en la aplicación generan un mensaje de log y las excepciones causadas en la aplicación son capturadas y mostradas en un mensaje de log orientando al usuario que tome una acción en consecuencia para evitar cerrar la aplicación inesperadamente. Esto mejora la usabilidad del sistema para el usuario que puede saber en todo momento el estado de la aplicación. Desde el menú *Ayuda* puede acceder a una ventana de logs (Figura 4.19) donde podrá filtrar por colores, tipo de log generado o realizar una búsqueda de texto; podrá guardar su contenido o imprimir en papel la lista de logs. Además, todos los logs se van guardando dentro de la carpeta 'logs' en el directorio de la aplicación.

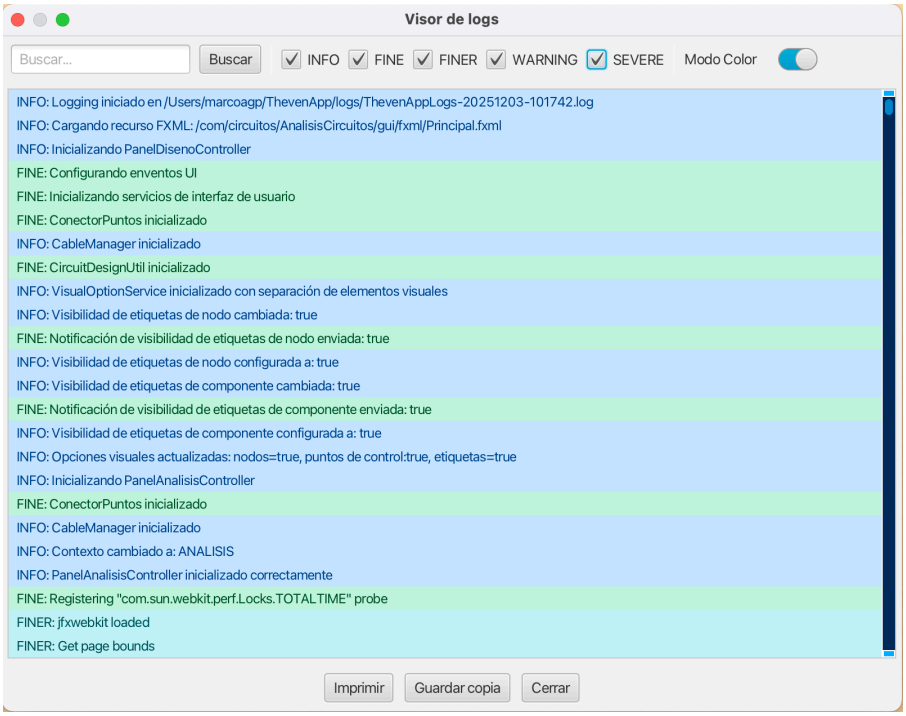


Figura 4.19: Visor de logs de la aplicación.

4.8. Librerías y otras herramientas utilizadas

Una de las grandes ventajas de utilizar Java como lenguaje principal de la aplicación es la posibilidad de usar librerías ya creadas que resuelven problemas que requerirían cientos de líneas de código. De acuerdo al principio de reutilización de código y de mantenibilidad se ha optado por integrar librerías estandarizadas que facilitan el desarrollo del software, la reducción de probabilidad de introducir errores y la comprensión por parte de otros desarrolladores.

4.8.1. Apache Commons Math 3

Figura 4.20: Logotipo de Apache Commons [18].

Se trata de una librería Java con recursos matemáticos y estadísticos para resolver la mayoría de problemas matemáticos complejos en lenguaje Java. En la Figura 4.20 se puede ver el logotipo de la librería. En el caso de *ThevenApp* es el gran motor de cálculo que realiza las operaciones más costosas de la resolución de circuitos. Se estudió la posibilidad de utilizar la librería JAMA pero se descartó por ser Apache Commons Math 3 más moderna, sencilla y con mejor aceptación actualmente. Tal y como se han planteado los algoritmos para el método nodal modificado (MNA) empleado en esta aplicación, se requiere realizar operaciones como suma o multiplicación de matrices complejas para resolver los sistemas de ecuaciones generados por el MNA. Por ello, se han empleado las siguientes clases pertenecientes a esta librería:

- **RealMatrix**: representa una matriz con números reales y permite las operaciones de suma, resta, multipliación de matrices, transposición u operaciones con vectores o escalares. Se usa la implementación *Array2DRowRealMatrix*.
- **RealVector**: representa un vector con números reales y permite las operaciones de suma, resta, multiplicación, división u operaciones con escalares. Se usa la implementación *ArrayRealVector*.
- **LUDecomposition**: es una clase que calcula la descomposición LU de una matriz cuadrada que es un algoritmo para resolver sistemas de ecuaciones adecuado a los problemas que se plantean en la aplicación.

4.8.2. Jackson

Jackson es una librería altamente utilizada en Java para manejar archivos JSON. Jackson facilita todas las operaciones de convertir objetos del dominio en archivos JSON y viceversa. Se ha optado por usarlo en la aplicación porque es sencillo, rápido y tiene una solidez notable en grandes proyectos Java.

En la aplicación se utiliza para convertir los circuitos, con todos sus componentes, cables, posiciones respecto al área de diseño, parámetros, etc. en un archivo JSON para poder almacenarlo en el disco local. Posteriormente, se puede cargar ese archivo en la aplicación. Por tanto, Jackson realiza las operaciones de conversión de formato de objetos Java a objetos JSON y viceversa; es decir, se encarga de serializar y deserializar circuitos en la aplicación.

4.8.3. Maven

Figura 4.21: Logotipo de la herramienta Maven [19].

Maven es una herramienta de automatización de proyectos de software, especialmente, orientada a Java. Una de sus grandes ventajas es no tener que buscar cada librería en los repositorios en línea e instalarlas manualmente.

Se puede ver el logotipo de la aplicación en la Figura 4.21.

Maven dispone de un archivo de configuración `pom.xml` que permite definir dependencias, metadatos o plugins de forma que ella misma se encarga de descargar, instalar y configurar el proyecto según la configuración elegida. Esto facilita mucho la creación de software y por eso se ha utilizado como herramienta de automatización en esta aplicación. Además automatiza la compilación y el empaquetado final del archivo JAR

4.8.4. JUnit 5

Figura 4.22: Logotipo de JUnit 5 [20].

JUnit es el estándar en Java para la realización de pruebas unitarias. Se trata de un framework *Open Source* que permite realizar pruebas automatizadas para verificar que ciertas partes del código funcionan como se espera. En la Figura 4.22 se puede ver el logotipo JUnit.

En esta aplicación se ha utilizado para realizar pruebas en el motor de cálculo y verificar, en circuitos de diferentes niveles de complejidad, que los resultados obtenidos para la tensión de Thévenin, resistencia equivalente o corriente de Norton son correctos y, así, validarlo.

4.8.5. Eclipse IDE

Figura 4.23: Logotipo del IDE Eclipse [21].

Sería imposible terminar esta sección sin mencionar el IDE utilizado para realizar esta aplicación. Eclipse es un entorno de desarrollo integrado (IDE) que, aunque se puede utilizar en muchos lenguajes, está orientado

principalmente a proyectos desarrollados en Java. Contiene todas las herramientas necesarias para el desarrollo de software tanto para escribir, como para compilar o depurar código. En la Figura 4.23 se muestra el logotipo de Eclipse. Se ha optado por este IDE por las siguientes razones:

- Es de código abierto y gratuito.
- Es multiplataforma (se puede usar en Mac, Windows o Linux).
- Es extensible: tiene un gran catálogo de plugins para personalizarlo o herramientas para mejorarlo.

La mejor herramienta siempre es la que mejor se sabe utilizar y Eclipse es un IDE ampliamente utilizado en carreras de ingeniería por diferentes universidades y, por ello, se ha seleccionado como la mejor opción para el desarrollo de esta aplicación.

4.9. Conclusiones

Las aplicaciones comerciales actuales se caracterizan por un diseño moderno, elegante, minimalista y muy cuidado. En este aspecto, se ha pretendido adoptar ese estilo de diseño en **ThevenApp** para obtener una interfaz moderna, sencilla y atractiva. Por ello, se han empleado colores llamativos pero con un uso lógico dentro de la aplicación, botones grandes y modernos y, sobre todo, se han respetado los principios de diseño estético y minimalista y de reconocimiento antes que recuerdo. El resultado, finalmente, ha sido una aplicación visualmente agradable, además, de funcional.

Capítulo 5

Implementación de la aplicación

*El desarrollo de la aplicación **ThevenApp** ha requerido una planificación previa que siga los principios de la Ingeniería de Software para garantizar una implementación lógica y estructurada que permita su mantenibilidad. Esta aplicación hace uso de diferentes patrones de diseño, algoritmos y estructuras de datos que han sido cuidadosamente planificados para cumplir los objetivos marcados [22]. En este capítulo se detalla cómo se han solucionado los diferentes problemas que planteaba este proyecto.*

5.1. Estructura del Proyecto

La organización en un proyecto de programación es uno de los fundamentos clave para obtener el éxito. A medida que un proyecto crece, algunas operaciones como la depuración, la mejora o la mantenibilidad pueden resultar tediosas si no está perfectamente organizado.

Este proyecto ha sido planificado desde su inicio con el objetivo de evitar situaciones insostenibles durante su crecimiento. Para ello, se han llevado a cabo una serie de pautas:

- Implementación del patrón Modelo-Vista-Controlador (MVC) para separar lógicas diferentes.
- Estructuración de los paquetes en base a este patrón y separación de las clases de utilidad, servicios, etc. en otros diferentes.
- Uso de la herramienta Maven para organizar el proyecto, gestionar las librerías externas y conseguir que el proyecto compile adecuadamente.
- Programación de clases cortas y dirigidas a una función en particular cumpliendo, así, el principio de responsabilidad única.
- Documentación de código para facilitar la lectura, el mantenimiento y la reusabilidad.

5.1.1. Organización de paquetes

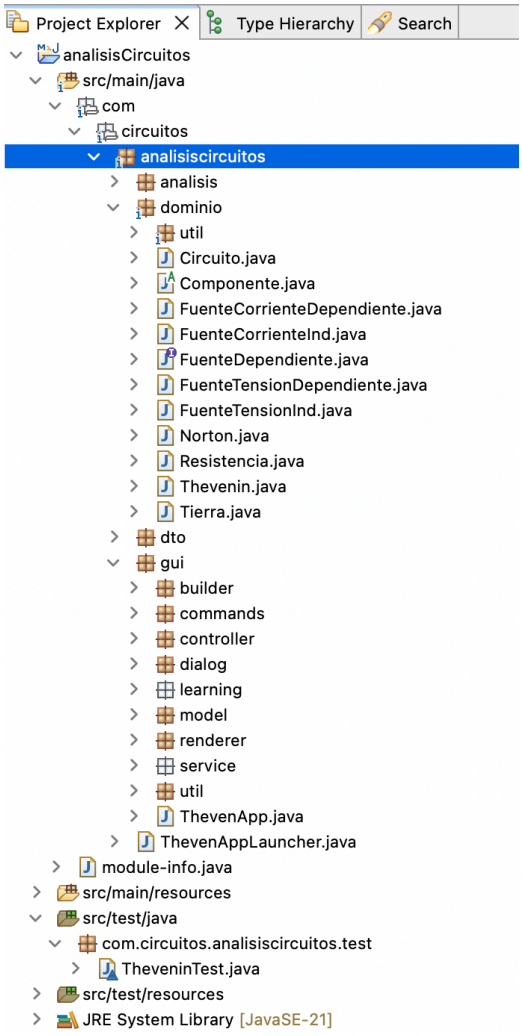
Todo el proyecto está contenido en un paquete que cumple con la convención de dominio inverso normalizada en proyectos Java con el objetivo de que cada nombre sea único y no entre en conflicto con nombres similares de librerías externas. Esta decisión responde a las "buenas prácticas" de la Ingeniería de Software.

El paquete principal se ha nombrado como `com.circuitos.analisiscircuitos`; a partir de ahí separa el código Java en la carpeta `src/main/java` como se puede ver en la Figura 5.1a del proyecto Maven (capas Modelo y Controlador) y la carpeta `src/main/resources` (capa Vista) como aparece en la Figura 5.1b siguiendo esta jerarquía:

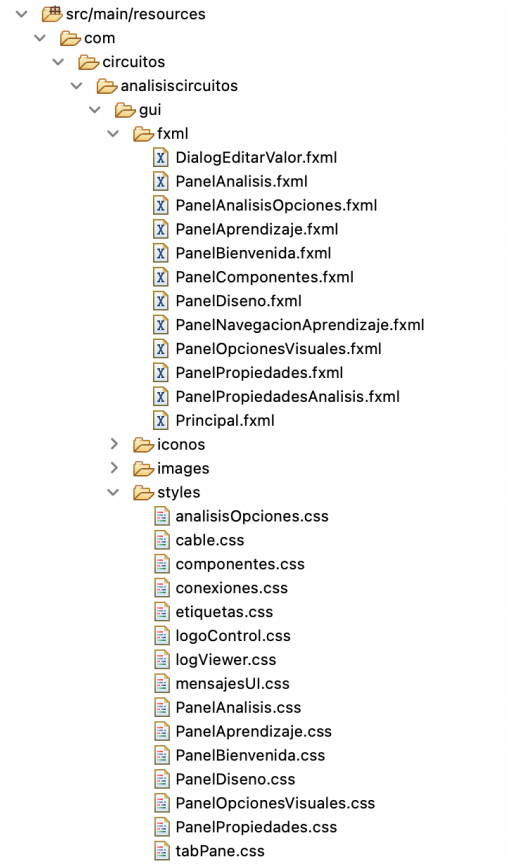
- `com.circuitos.analisiscircuitos` (capa Modelo y capa Controlador):
  - `analisis`: Analizadores de circuitos para Thévenin y Norton (capa Modelo).

- `dominio`: Clases de dominio para definición de la capa modelo (incluye utilidades).
- `dto`: Records para almacenamiento en archivos JSON (capa Modelo).
- `gui`: Clases relacionadas con la Interfaz Gráfica de Usuario (GUI) que incluye controladores, definiciones de objetos, servicios, gestión de base de datos y utilidades para el control de la GUI (capa Modelo-Controlador).

- `com.circuitos.analisiscircuitos.gui` (capa Vista):
  - `fxml`: Contiene los archivos FXML que definen la interfaz gráfica.
  - `iconos`: Contiene los iconos de componentes que usa la interfaz gráfica.
  - `images`: Contiene las imágenes que utiliza la interfaz gráfica.
  - `styles`: Contiene los archivos de hojas de estilo CSS.



(a) Organización de paquetes de las capas Modelo y Controlador.



(b) Organización de paquetes de la capa Vista.

Figura 5.1: Organización de paquetes de la aplicación según el patrón MVC.

5.1.2. Gestión de dependencias

El uso de Maven como herramienta de automatización permite usar librerías externas sin necesidad de perder tiempo en instalaciones o configuraciones. Se ha decidido usar Maven porque se encarga de este tipo de gestiones y permite centralizar el esfuerzo en la aplicación.

Maven pone a disposición del desarrollador un archivo de nombre `pom.xml` en el cual se configuran todas las

dependencias del proyecto como **Apache Commons**, **Jackson** o **SQLite**. Una vez se encuentran en el archivo, Maven se encarga automáticamente de instalar y configurar estas herramientas para el proyecto y asegurarse de que compile correctamente.

5.2. Implementación de la Interfaz Gráfica

Existen diferentes librerías en Java para desarrollar una interfaz gráfica. Hasta hace relativamente poco se utilizaban las librerías *Java AWT* o *Java Swing*; sin embargo, la librería JavaFX abre un mundo de posibilidades para diseñar una interfaz gráfica moderna, sencilla y reutilizable para una aplicación de escritorio usando las ventajas de las interfaces gráficas de aplicaciones Web como archivos XML (FXML en el caso de JavaFX) o el uso de hojas de estilo CSS para la personalización de la interfaz. Este ha sido uno de los principales motivos para usar JavaFX en esta aplicación.

JavaFX dispone de una aplicación llamada **SceneBuilder** (ver Figura 5.2) que permite construir una interfaz gráfica con la modalidad de WYSIWYG (*What You See Is What You Get*, lo que ves es lo que obtienes) sin necesidad de usar código. Aunque esta herramienta se ha utilizado para obtener el esqueleto de los archivos FXML ha sido necesaria la manipulación de código para que la interfaz tuviera el aspecto actual.

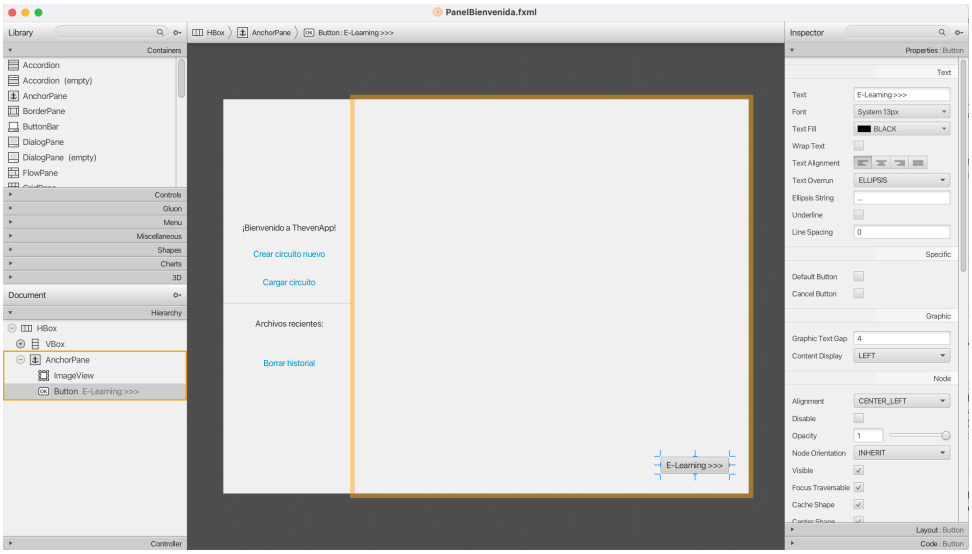


Figura 5.2: SceneBuilder con el esqueleto de la pantalla de bienvenida.

El funcionamiento de JavaFX [23] es el siguiente:

- **FXML**: contiene toda la parte visual en forma de paneles, cajas, botones, etc. unos encima de otros siguiendo una jerarquía (ver Código 5.1). Está asociado a un archivo java que actúa de controlador.
- **Controlador**: el controlador de un FXML es el mediador entre la vista y el modelo. Recoge las peticiones que haya en la vista y contacta con las clases del modelo para efectuar las acciones necesarias y devolverlo, de nuevo a la vista. Cada archivo FXML tiene un controlador asociado.
- **CSS**: a los componentes de un FXML se le puede dar un estilo propio basado en CSS para personalizar las ventanas, paneles o botones con un estilo propio.

Código 5.1: Fragmento del archivo FXML de la pantalla de Bienvenida.

```
<HBox fx:id="panelBienvenida"
    prefHeight="600.0"
    prefWidth="800.0"
    xmlns="http://javafx.com/javafx"
    xmlns:fx="http://javafx.com/fxml/"
    fx:controller="com.circuitos.analisiscircuitos.gui.controller.
   PanelBienvenidaController">
```

5.2.1. Carga de Vistas

Un archivo FXML debe ser *cargado* por una clase Java principal a través de la clase *FXMLLoader* del paquete de JavaFX. Esto permite instanciar la jerarquía de objetos del archivo FXML y vincularla con su clase controladora.

En el propio archivo FXML se crean contenedores de elementos, paneles, cajas o botones de forma jerárquica y cuando un elemento debe realizar una acción (por ejemplo un botón) se le asigna una función `onAction` que será implementada en Java dentro del controlador de forma que éste pueda contactar con el modelo para realizar una acción necesaria y devolverla a la interfaz gráfica a través de algún panel, cuadro de diálogo u otra herramienta declarada en el archivo FXML. Algunos FXML contienen elementos creados en otro archivo FXML y es el controlador el que se encarga de cargarlos con la clase *FXMLLoader* y situarlos en el lugar adecuado importando todas sus funcionalidades. (Ver Código 5.2).

Código 5.2: Inicialización del controlador principal cargando los componentes FXML.

```
  @FXML
  public void initialize() {
    configurarLogger();
    if(panelAnalisisOpciones!=null) {
      panelAnalisisOpciones.setVisible(false);
      panelAnalisisOpciones.setManaged(false);
    }
    if(panelNavegacionAprendizaje!=null) {
      panelNavegacionAprendizaje.setVisible(false);
      panelNavegacionAprendizaje.setManaged(false);
    }
    configurarTabs();
    cargarSubPaneles();
    circuitoRenderer=new CircuitoRender(panelDisenoController);
    InteraccionComponenteUtil.setContextoActual(
   InteraccionComponenteUtil.Contexto.DISENO);
    configurarUndoRedo();
    configurarMenuArchivo();
    configurarMenuEdicion();
    configurarOpcionesVisuales();
    Platform.runLater(this::conectarControladoresAprendizaje);
  }
```

En esta aplicación sirve de ejemplo el caso del panel principal (`Principal.fxml`) cuyo controlador es (`ThevenAppController.java`) que contiene:

- Panel de Diseño (`PanelDiseno.fxml`).
- Panel de Análisis (`PanelAnalisis.fxml`).
- Panel de *E-Learning* (`PanelAprendizaje.fxml`).
- Panel de Componentes (`PanelComponentes.fxml`).

- Panel de Propiedades (`PanelPropiedades.fxml`).
- Panel de Opciones Visuales (`PanelOpcionesVisuales.fxml`).

Estos paneles se *cargan* con el controlador principal de forma que se permita el cambio de panel a través de elementos accionables como pestañas (*TabPane*) o con la selección de un componente del área de diseño para mostrar u ocultar el panel de propiedades de ese componente.

5.2.2. Drag and Drop (Arrastrar y Soltar)

Uno de los puntos fuertes de la aplicación que aporta una gran usabilidad es la posibilidad de añadir componentes a un circuito arrastrando componentes de una paleta de componentes hasta el área de diseño y soltar su icono en este área. Esta funcionalidad conlleva una serie de pasos a nivel de implementación para obtener el resultado requerido.

Para implementar esta funcionalidad se ha optado por incluir el icono de cada componente dentro de un *StackPane*, es decir, una pila de elementos que permite gestionar el conjunto de elementos en su interior como si fuera uno. En el panel de componentes de la aplicación una resistencia, por ejemplo, lleva asociada un icono que es su imagen visual, un recuadro (*Rectangle*) que es el cuadrado que permite su selección, una etiqueta asociada y todo el conjunto debe poderse manipular como tal; por ello, se ha optado por usar un *StackPane* para implementar esto. En la Figura 5.3 se muestra un StackPane de forma esquematizada.

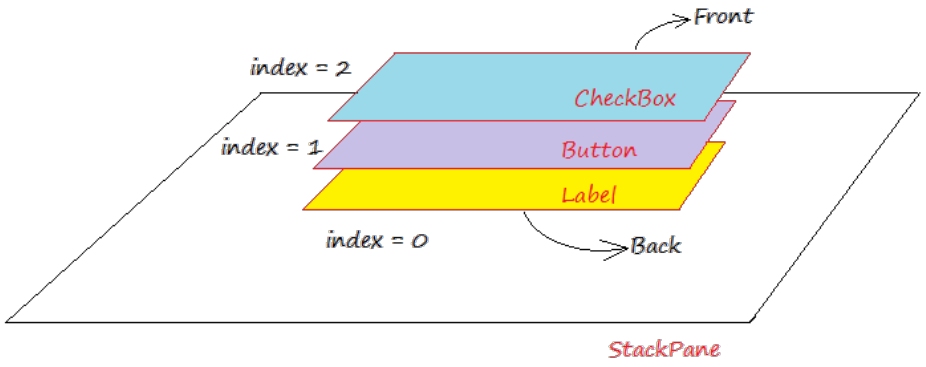


Figura 5.3: Esquema de un StackPane de JavaFX [24].

Para realizar el *drag and drop* se han utilizado diferentes eventos de ratón:

- **startDragAndDrop**: para iniciar el movimiento en modo copia (`TransferMode.COPY`), es decir, crea una copia del componente que se arrastra. Esta llamada se hace desde el panel de componentes.
- **setOnDragOver**: se indica al panel de diseño que acepte elementos arrastrados desde el panel de componentes (`acceptTransferModes(TransferMode.COPY_OR_MOVE)`). Esto prepara el panel de diseño para elementos que están siendo arrastrados desde otro panel.
- **getDragBoard**: el área de diseño obtiene el elemento arrastrado y crea una copia del propio elemento de dominio que añade al circuito asignando una identificación propia y unos valores preconfigurados.
- **setDropCompleted**: cuando se ha realizado todo el proceso, crea el elemento visual en la posición donde lo suelta el ratón y termina la operación.

5.2.3. Implementación de cables

La implementación de conexiones o cables supone un desafío técnico importante en la aplicación. A diferencia de los componentes que son entidades fijas, los cables son entidades dinámicas que deben poder actualizar sus

coordenadas geométricas en tiempo real cada vez que hay un movimiento de componente.

En primer lugar, se han creado puntos de conexión asociados a cada componente (pertenecientes a su `StackPane`) siguiendo la siguiente estrategia:

- Cada nodo de un componente lleva asociados dos puntos de conexión (negativo y positivo) representados visualmente por la clase `Circle` del paquete `javafx.scene.shape` que permite definir un círculo pequeño dado por su radio y las coordenadas de su centro y darle un estilo personalizado.
- Estos pequeños círculos son los puntos donde se conectan los cables imitando un efecto *snap*. El cable se ajusta automáticamente a este punto de conexión.
- Los puntos de conexión van asociados a una red de nodos que simula una red eléctrica real. Si dos puntos de conexión son consecutivos se les hace pertenecer a una misma red. Este tipo de conexión se ha implementado con una clase `Net` que agrupa puntos de conexión pertenecientes a un mismo nodo eléctrico. (Ver Códigos 5.3, 5.4 y 5.5).

Código 5.3: Definición de PuntoConexion y asociación a un objeto Net.

```
// Clase PuntoConexion (Extiende de Circle para la representacion visual)
public class PuntoConexion extends Circle {
    // Propiedad que vincula el punto con su red electrica (Net)
    private final ObjectProperty<Net> netProperty = new SimpleObjectProperty<>(null);
    private final IntegerProperty nodoProperty = new SimpleIntegerProperty(this, "nodo", -1);

    public PuntoConexion(Componente componente, boolean esPositivo, Posicion posicion) {
        super(RADIO); // Define el tamano del circulo visual
        this.componente = componente;
        // si cambia la Net, se actualiza el ID del nodo
        nodoProperty.bind(Bindings.createIntegerBinding(
            () -> netProperty.get() != null ? netProperty.get().getId() : nodoPredeterminado,
            netProperty
        ));
    }
}
```

Código 5.4: Clase Net para la agrupación lógica de conexiones.

```
public class Net {
    private final IntegerProperty idProperty = new SimpleIntegerProperty(-1);
    private final List<PuntoConexion> pins = new ArrayList<>();

    // Agrupa un nuevo punto de conexion a esta red electrica
    public void addPin(PuntoConexion pin) {
        if(!pins.contains(pin)) {
            pins.add(pin);
            pin.setNet(this); // Vinculacion bidireccional
        }
    }
}
```

Código 5.5: Lógica de conexión en ConectorPuntos.

```
public Cable conectarPuntos(PuntoConexion a, PuntoConexion b) {
    Net netA = a.getNet();
    Net netB = b.getNet();
    Net netFinal;

    // Logica para determinar la red resultante
    if (netA == null && netB == null) {
        netFinal = crearNet(); // Nueva red si ambos estan libres
        netFinal.addPin(a);
        netFinal.addPin(b);
    } else if (netA != netB) {
        // Fusion de redes si son diferentes
        netFinal = (netA.getId() < netB.getId()) ? netA : netB;
        Net toRemove = (netFinal == netA) ? netB : netA;
        fusionar(netFinal, toRemove);
    } else {
        netFinal = netA;
    }
    // ... Creacion visual del cable ...
}
```

Una vez obtenidos los puntos de conexión donde conectar los cables, la siguiente decisión ha sido cómo representar visualmente el cable. Para ello, se ha utilizado la clase `Polyline` del paquete `javafx.scene.shape` que, después de barajar otras opciones, se ha optado por ella por diversas razones:

- Permiten definir un cable como una lista de coordenadas unidas por líneas rectas, similar a la definición de un polígono pero sin trayectoria cerrada. Esto lo hace la opción ideal para el tipo de trayectorias que se representan en la aplicación.
- Se puede personalizar con estilos CSS.
- Permiten añadir o eliminar puntos de control para definir su trayectoria.

Los cables en la aplicación siguen **trayectorias ortogonales** con líneas horizontales y verticales con giros de 90º siguiendo un estilo tipo *"Manhattan"*. Se ha optado por este estilo con el objetivo de que los esquemas de circuitos sean sencillos, claros y, visualmente, atractivos.

Los cables llevan asociados *listeners* que permiten "escuchar" cualquier tipo de movimiento de los componentes a los que están conectados, de forma que actualicen sus coordenadas en tiempo real. Esto aporta un dinamismo y una versatilidad notable para este tipo de aplicación. (Ver Código 5.6).

Para crear cables se ha implementado el patrón **Builder** para facilitar el código en su creación tal como se explica en la Sección 4.4.3 sobre los patrones de diseño utilizados en la aplicación.

Código 5.6: Actualización dinámica de cables mediante Listeners.

```
public class Cable extends Group {
    private final Polyline cable = new Polyline();

    // Asocia listeners a los nodos conectados para reaccionar al
    movimiento
    private void bindListeners(PuntoConexion inicio, PuntoConexion fin
    ) {
        ChangeListener<Number> listener = (obs, oldVal, newVal) ->
    actualizarCable(inicio, fin);

        Node nodoInicio = inicio.getParent();
        Node nodoFin = fin.getParent();

        if (nodoInicio != null) {
            // Escuchar cambios en posicion, escala y rotacion
            nodoInicio.layoutXProperty().addListener(listener);
            nodoInicio.layoutYProperty().addListener(listener);
            nodoInicio.rotateProperty().addListener(listener);
        }
        // ... (Igual para el nodo fin)
    }
}
```

5.3. Implementación de la resolución de circuitos

Tal como se detalla en la Sección 2.6.2, para resolver los circuitos, la aplicación utiliza el método del *Análisis Nodal Modificado*. Esto requiere solucionar diferentes problemas que surgen para llevar a cabo esta técnica, tales como crear un algoritmo que resuelva Thévenin y Norton, crear las matrices necesarias o resolver los sistemas de ecuaciones en forma matricial para llegar a una solución concreta.

5.3.1. Algoritmo de Thévenin

El planteamiento que se hizo para resolver un circuito de tipo resistivo en corriente continua mediante el teorema de Thévenin se resume en obtener un valor de tensión conocido como **tensión de Thévenin** y un valor óhmico conocido como **resistencia equivalente de Thévenin**. En principio, se implementó un método de reducción de resistencias serie/paralelo pero fallaba con fuentes dependientes y por ello se optó por resolver el circuito completo haciendo uso del álgebra lineal.

En primer lugar, se calcula la **Tensión de Thévenin, Vth** de la siguiente manera:

1. Se obtienen los nodos de cálculo de la siguiente manera:
   - Si hay una única resistencia de carga, se utilizan sus nodos de conexión.
   - Si hay más de una resistencia de carga, se verifica que sean conexas consecutivas con un recorrido BFS (`GraphUtil.java`) para comprobar que se puedan desconectar del resto del circuito sin comprometer la integridad del resto.
   - Si no hay componentes de carga, se usan los nodos especificados por el usuario.
2. Se genera una copia temporal del circuito abierto entre esos nodos (sin conexión entre ellos).
3. Se forman las matrices necesarias (ver sección 5.3.3) y se realiza la llamada al motor de cálculo (`MatrixUtil.java`) para resolver el circuito.

4. Una vez realizados los cálculos, se obtiene un vector de voltajes de cada nodo con respecto a tierra (nodo 0 asignado automáticamente).
5. El valor de Vth se obtiene calculando la diferencia de potencial entre los nodos de cálculo:

$$V_{th} = V_A - V_B \tag{5.1}$$

Una vez obtenido este valor, se calcula el valor de la **Resistencia Equivalente de Thévenin, Rth** usando el **Método de Inyección de Corriente de Prueba** ya que, gracias a este método, se evitan divisiones por cero y se simplifica el cálculo con fuentes dependientes. El procedimiento es el siguiente:

1. Se genera una copia del circuito con las fuentes independientes "apagadas", es decir:
   - Las **fuentes de tensión** se sustituyen por cortocircuitos, es decir, 0V (un cable).
   - Las **fuentes de corriente** se sustituyen por circuitos abiertos, es decir 0A (se eliminan).
2. Se inyecta (conecta) una **Fuente de Corriente Independiente** de valor $I_{\text{test}} = 1{,}0\,\text{A}$ entre los nodos de cálculo.
3. Se crean las matrices necesarias y se realiza la llamada al motor de cálculo para que resuelva el sistema.
4. El valor obtenido es el de la tensión en los nodos de cálculo por ello basta con resolver la **ley de Ohm** para obtener el valor de la resistencia.

$$R_{th} = \frac{V_A - V_B}{1{,}0} = |V_A - V_B| \tag{5.2}$$

5.3.2. Algoritmo de Norton

La resolución de un circuito mediante el teorema de Norton se resume en obtener un valor de corriente conocido como **corriente de Norton** y un valor óhmico conocido como **resistencia equivalente de Norton** que es exactamente igual a la resistencia equivalente de Thévenin.

Para ello, se ha optado por reutilizar el algoritmo de Thévenin como se indica en la Sección 5.3.1 y aplicar, simplemente, la **Ley de Ohm**. Es decir:

1. Calcula V_{th} y R_{th} usando la clase Thevenin (`Thevenin.java`).
2. Aplica la **Ley de Ohm** para transformar la fuente de tensión en una fuente de corriente:

$$I_{Norton} = \frac{V_{th}}{R_{th}} \tag{5.3}$$

$$R_{Norton} = R_{th} \tag{5.4}$$

3. Si $R_{th} = 0$, el algoritmo lo maneja como una fuente ideal de corriente infinita a la que le asigna `Double.POSITIVE_INFINITY` y sugiere al usuario revisar el circuito.

5.3.3. Construcción de Matrices

El algoritmo de Thévenin requiere construir matrices con valores concretos del circuito original. El objetivo es formar un sistema de ecuaciones lineales de tipo:

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{Z} \tag{5.5}$$

y, una vez, completadas las matrices, resolverlo.

Para rellenar las matrices se ha usado la técnica de **estampado** que consiste en recorrer la lista de componentes una sola vez y cada componente aporta un dato a las matrices. Se ha optado por este método en lugar de recorrer gráficamente el circuito "leyendo" mallas por ser mucho más simple, menos propenso a errores y tener menor coste computacional que muchos algoritmos de recorrido de grafos.

La lógica interna de cálculo utiliza los números de nodo para saber qué componentes comparten una conexión eléctrica de forma que los cables son meros elementos visuales que facilitan la visualización de los esquemas a los usuarios pero que no intervienen, en absoluto, en el motor de cálculo.

Las siguientes matrices se van rellenando según se realiza un recorrido de la lista de componentes:

- **Matriz A (Matriz del Sistema)**: esta matriz se divide en cuatro submatrices.
  - **Matriz G (Conductancias)**: Es una matriz de tamaño $(N-1) \times (N-1)$ siendo N el número de nodos del circuito. Cada resistencia de la lista de componentes suma su conductancia [1] en las posiciones diagonales *(i, i)* y se restan en las posiciones cruzadas *(i, j)* (ver Código 5.7). En esta matriz también se estampan las fuentes de corriente controladas por voltaje (VCCS). Estas fuentes estampan una corriente en el nodo de salida positivo que absorben del nodo de salida negativo controlada por la diferencia de tensión entre dos nodos, es decir:

    $$I_{out} = g_m \cdot (V_{c+} - V_{c-}) \tag{5.6}$$

    Por tanto, al pasar esta corriente a la izquierda de la *Ley de Corrientes de Kirchhoff, KCL* ($\sum I_{in} = \sum I_{out}$), los signos se invierten de forma que, en la fila del nodo de salida positivo, se estampa $-g_m$ en la columna $c+$ y $+g_m$ en la columna $c-$ y, en la fila del nodo de salida negativo, se estampa $+g_m$ en la columna $c+$ y $-g_m$ en la columna $c-$ (ver Código 5.8).
  - **Matrices B y C (Ampliaciones)**: La matriz G se amplia con estas submatrices B y C añadiendo una fila extra para cada fuente de tensión del circuito ya sean independientes o fuentes de tensión controladas por voltaje (VCVS). Ésta última impone la restricción de que el voltaje que genera depende de otro voltaje existente, es decir,

    $$(V_{out+}) - (V_{out-}) - \mu \cdot (V_{c+}) + \mu \cdot (V_{c-}) = 0 \tag{5.7}$$

    por tanto, en la matriz C, que se corresponde con la fila creada para esta fuente, se estampa +1 y -1 en las columnas de los nodos de salida, $-\mu$ en la columna del nodo de control positivo ($c+$) y $+\mu$ en la columna del nodo de control negativo ($c-$) (ver Código 5.9).
  - **Matriz D (Dependencias Directas)**: se trata de una matriz de tamaño $F \times F$ donde F es el número de fuentes de tensión controladas por corriente (CCVS); en caso de que no existan en el circuito, será 0. La tensión que genera esta fuente es la ganancia por la corriente de control (corriente entre dos nodos), es decir,

    $$V_{out} = \alpha \cdot I_{control} \tag{5.8}$$

    y, por tanto, depende de una corriente desconocida. Se estampa el valor $-\alpha$ en la matriz D en el cruce entre la fila de la fuente dependiente con la columna de la fuente que controla.
- **Vector x (Incógnitas)**: en este vector se incluyen todas las incógnitas que hay en el circuito y que se quieren conocer. Está dividido en dos partes:
  - **V (Voltajes Nodales)**: en la parte superior del vector (primeras $N-1$ filas se encuentra el voltaje desconocido de cada nodo del circuito excepto el del nodo de tierra (0V).
  - **J (Corrientes de Rama)**: en la parte inferior del vector se encuentran las corrientes que atraviesan cada fuente de tensión (al ser tensión ideal, son desconocidas). En el código fuente se dispone del método `extraerVoltajes` que extrae la parte V del vector.
- **Vector Z (Estímulos)**: Este vector es el segundo miembro de la ecuación a resolver y contiene los valores conocidos de las fuentes independientes del circuito.
  - **I (Fuentes de Corriente)**: la parte superior del vector contiene la suma algebraica de las corrientes que entran o salen de cada nodo que vienen de fuentes de corriente independientes. Si la fuente sale del nodo, se resta su valor en la fila del nodo; si entra, se le suma.
  - **E (Fuentes de Tensión)**: la parte inferior del vector contiene los valores de tensión de las fuentes independientes.
- **Caso Especial - Fuente de Corriente Controlada por Corriente (CCCS)**: según el algoritmo diseñado, este tipo de fuentes dependientes requería una fila y una columna extra en la matriz del sistema, sin embargo, gracias a la **Ley de Ohm** se ha optado por un cálculo más inteligente convirtiéndola en una fuente de corriente controlada por voltaje (VCCS) evitando, así, ampliar la matriz del sistema más de lo necesario. La ecuación de este tipo de fuente es

    $$I_{out} = \beta \cdot I_{control} \tag{5.9}$$

[1] Se entiende por **conductancia** el valor inverso de la resistencia $Y = 1/R$.

siendo β la ganancia de la fuente e $I_{control}$ una nueva incógnita. Basta con obtener la resistencia entre los nodos de control (por donde circula la corriente de control) y aplicar la Ley de Ohm:

$$I_{control} = \frac{V_{ctrl+} - V_{ctrl-}}{R_{ctrl}} \tag{5.10}$$

sustituyendo esta ecuación en la ecuación de la fuente 5.9 se obtiene la siguiente ecuación:

$$I_{out} = \beta \cdot \frac{V_{ctrl+} - V_{ctrl-}}{R_{ctrl}} \tag{5.11}$$

Puesto que tanto β como R_{ctrl} son conocidas, se puede definir una ganancia equivalente para la nueva VCCS, $g_m(eq) = \frac{\beta}{R_{ctrl}}$ obteniendo, finalmente,

$$I_{out} = g_m(eq) \cdot (V_{ctrl+} - V_{ctrl-}) \tag{5.12}$$

que es la ecuación de una fuente de corriente controlada por tensión y por tanto, se estampa directamente en la matriz de conductancias (G) sin necesidad de ampliarla. Este caso se puede observar en el método `StampUtil.stampSelectFuenteCorriente` (ver Código 5.10).

Una vez creadas las matrices queda formado el sistema de ecuaciones matricial que deba resolver el motor de cálculo:

$$\begin{bmatrix} \boldsymbol{G} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{V} \\ \boldsymbol{J} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{E} \end{bmatrix} \tag{5.13}$$

Código 5.7: Estampado de conductancias en la matriz G.

```
//Metodo para anadir conductancia a la matriz G
public static void stampG(RealMatrix G, int fila, int col, double
    valor, int ref) {
     // Comprime los indices para ignorar el nodo de referencia
     int i = comprimir(fila, ref);
     int j = comprimir(col, ref);
     if(i >= 0 && j >= 0) {
         // Acumula el valor (1/R) en la posicion correspondiente
         G.addToEntry(i, j, valor);
     }
}
```

Código 5.8: Estampado de fuente de corriente controlada por voltaje (VCCS).

```
public static void stampVCCS(RealMatrix G, int pout, int nout, int
    cpos, int cneg,
         double gm, int ref) {
     //Estampa gm en las cuatro posiciones afectadas por la dependencia
     stampG(G, nout, cpos, +gm, ref);
     stampG(G, nout, cneg, -gm, ref);
     stampG(G, pout, cpos, -gm, ref);
     stampG(G, pout, cneg, +gm, ref);
}
```

Código 5.9: Estampado de coeficientes para fuentes de tensión (Matrices B y C).

```
public static void stampB(RealMatrix B, int i, int j, double valor,
    int ref) {
     int fila = comprimir(i, ref);
     if(fila >= 0) {
         B.addToEntry(fila, j, valor); // Incidencia en ecuacion nodal
     }
}

public static void stampC(RealMatrix C, int i, int j, double valor,
    int ref) {
     int col = comprimir(j, ref);
     if(col >= 0) {
         C.addToEntry(i, col, valor); // Incidencia en ecuacion de
    restriccion
     }
}
```

Código 5.10: Transformación de CCCS a VCCS mediante Ley de Ohm.

```
public static void stampSelectFuenteCorriente(RealMatrix G,
    FuenteCorrienteDependiente src,
         Map<Integer, Integer> nodos, int ref, Circuito c) {
     // ...
     if(src.getControlType() == FuenteCorrienteDependiente.ControlType.
    CORRIENTE) {
         //Busca la resistencia de la rama de control
         Optional<Resistencia> rCtrl = c.getResistencias().stream()
                 .filter(/* logica de busqueda de nodos */)
                 .findFirst();

         if(rCtrl.isPresent()) {
             double R = rCtrl.get().getValor();
             // Calcula valor gm equivalente: gm = Beta / R
             double gm = src.getValor() / R;
             //Estampa como si fuera una VCCS
             stampVCCS(G, pout, nout, cpos, cneg, gm, ref);
             return;
         }
     }
     // ...
}
```

5.3.4. Resolución del Sistema Matricial (Apache Commons Math)

El proceso de estampado de matrices finaliza con un sistema matricial con dos estructuras de datos diferentes de la librería **Apache Commons Math** siendo una opción mejor que programar innecesarias líneas de código para realizar operaciones con matrices con su correspondiente gasto computacional y su posible tendencia a errores de cálculo.

Las estructuras de datos de la libreria Apache Commons Math necesarias para resolver el sistema matricial son:

- **RealMatrix A**: que representa la matriz ampliada del sistema (ver Código 5.11).
- **RealVector Z**: que representa el vector de términos independientes.

Código 5.11: Construcción de la matriz ampliada A usando Apache Commons Math.

```
1  //Metodo en MatrixUtil.java para ensamblar la matriz del sistema
2  private static RealMatrix construirMatrizAmpliada(RealMatrix G,
       RealMatrix B, RealMatrix C, RealMatrix D) {
3      int M = G.getRowDimension();
4      int F = B.getColumnDimension();
5      //Crea una matriz de tamano (M+F) x (M+F)
6      RealMatrix A = new Array2DRowRealMatrix(M + F, M + F);
7
8      //Rellena los bloques
9      A.setSubMatrix(G.getData(), 0, 0); //Bloque de conductancias
10     A.setSubMatrix(B.getData(), 0, M); //Bloque de incidencia de
       fuentes
11     A.setSubMatrix(C.getData(), M, 0); //Bloque de restriccion de
       fuentes
12     A.setSubMatrix(D.getData(), M, M); //Bloque de dependencias
       directas
13     return A;
14 }
```

La última estructura será el vector de incógnitas **x** que será la solución del sistema

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{Z} \tag{5.14}$$

Para resolver el sistema, se utiliza la técnica de descomposición LU a través de la clase `LUDecomposition` de la librería Apache Commons Math. Esta técnica divide la matriz del sistema **A** en el producto de la matriz triangular inferior **L** y la matriz triangular superior **U**.

En los métodos `resolverConSupernodos` y `resolverSinFuentes` se analiza la matriz A y, en caso de que sea singular (determinante 0) lanza una excepción capturada para avisar al usuario del problema. Después realiza la sustitución hacia adelante y hacia atrás para obtener el vector **x** (ver Código 5.12).

Código 5.12: Resolución del sistema mediante descomposición LU.

```
// Metodo resolverConSupernodos en MatrixUtil
try {
    //Resuelve Ax = Z utilizando descomposicion LU
    X = new LUDecomposition(A).getSolver().solve(Z);
} catch (SingularMatrixException e) {
    throw new IllegalStateException("Matriz ampliada singular (
    circuito invalido)", e);
}
```

Una vez obtenido el vector de incógnitas (que contiene corrientes y voltajes), extrae los voltajes creando un array de voltajes (`double[] V`) de tamaño N (número de nodos) y asigna al nodo de tierra el valor de tensión $0{,}0V$.

La clase `Thevenin` accede a este array para obtener el voltaje de circuito abierto, es decir, el que hay entre los nodos de cálculo seleccionados:

$$V_{th} = V[nodoA] - V[nodoB] \tag{5.15}$$

Finalmente, con las fuentes desactivadas e inyectada la corriente de prueba $I_{test} = 1A$ se obtiene el voltaje en el nodoA y el voltaje en el nodoB; la diferencia entre estos será el voltaje en la rama donde se inyectó la corriente de prueba, por tanto, se aplica la ley de Ohm (dividir este voltaje entre 1 amperio) y se obtiene el valor de R_{th} (ver Código 5.13).

Código 5.13: Cálculo de parámetros de Thevenin en el dominio.

```
//Clase Thevenin.java
private double calcularVth() {
    // ... resolver circuito abierto ...
    double[] voltajes = MatrixUtil.resolverCircuitoNodal(
    circuitoAbierto);
    //Vth = Va - Vb
    return voltajes[indiceA] - voltajes[indiceB];
}

private double calcularRth() {
    // ... desactivar fuentes e inyectar corriente de prueba ...
    circuitoDesactivado.addComponente(new FuenteCorrienteInd(I_TEST,
    nodoA, nodoB));

    double[] voltajes = MatrixUtil.resolverCircuitoNodal(
    circuitoDesactivado);
    //Rth = |Va - Vb| / I_TEST (donde I_TEST = 1.0)
    return Math.abs(voltajes[idxA] - voltajes[idxB]);
}
```

5.4. Implementación de la Persistencia de Datos

ThevenApp utiliza dos formas diferentes de tratar la persistencia de datos; por un lado, utiliza archivos JSON para guardar circuitos en un archivo local y, por otra parte, utiliza una base de datos SQLite para gestionar la sección de *E-Learning*. Se ha evitado almacenar los circuitos en la base de datos porque requeriría crear tablas relacionales complejas que complicarían los accesos a la base de datos generando un gran coste computacional (a diferencia de JSON) y una mayor tendencia a errores.

5.4.1. Archivos JSON para almacenamiento de circuitos

La decisión de usar el formato JSON para almacenar un circuito ha requerido un planteamiento lógico de los elementos, valores o parámetros que se deben almacenar para conseguir un archivo consistente con toda la información requerida para su posterior recuperación. JSON permite que, de una manera sencilla, se almacenen los datos perfectamente ordenados y sean, fácilmente, accesibles. Esta es la razón por la que se ha seleccionado este formato. El contenido de un archivo JSON de la aplicación es el siguiente:

- **Metadatos**: se almacena un sector de metadatos con el nombre de la aplicación, la versión y la fecha de creación; importante para que el archivo sea único.
- **Datos de componentes**: se almacenan los componentes del circuito declarando el tipo, de componente, nodo 1 (negativo), nodo 2 (positivo), id, carga (true/false) y valor. En caso de fuentes dependientes añade los nodos de control y la magnitud de dependencia.
- **Posiciones**: se almacenan las posiciones de cada componente identificado por su id: posición x, posición y, ángulo de rotación.
- **Cables**: para almacenar los cables no hay más que guardar todos sus parámetros para poder recrearlos: id, id del componente origen, pin del componente origen (negativo o positivo) y posición del pin, número de nodo de origen, id del componente destino, pin del componente destino y posición del pin, número de nodo de destino, posiciones x e y de todos sus puntos de control.

Para llevar a cabo esta tarea se ha utilizado la librería **Jackson**; un estándar en el mundo de Java para gestión de archivos JSON. En primer lugar, se establecen marcas en los métodos getters de las clases del dominio en forma de `JsonProperty("valor")`, por ejemplo para asignar ese getter al campo "valor" dentro del archivo (ver Código 5.14).

Código 5.14: Anotaciones Jackson para serialización en clase Componente.

```
@JsonTypeInfo(
    use=JsonTypeInfo.Id.NAME,
    include=JsonTypeInfo.As.PROPERTY,
    property="@type"
)
@JsonSubTypes({
    @JsonSubTypes.Type(value=Resistencia.class, name="Resistencia"),
    @JsonSubTypes.Type(value=FuenteTensionInd.class, name="
   FuenteTensionIndependiente"),
    // ... resto de subtipos ...
})
public abstract class Componente {
    @JsonProperty("nodo1")
    public int getNodo1() {
        return nodo1;
    }
    // ...
}
```

Como estructura de almacenamiento y recuperación de datos se han creado records DTO para almacenar posiciones (x, y) y rotación de los componentes o todos los parámetros que necesitan los cables (ver Código 5.15).

Código 5.15: Estructura del DTO para almacenamiento de archivos.

```
public record CircuitoFileDto (
    Metadata metadata,
    Circuito circuito,
    List<PosicionComponenteDto> posiciones,
    List<CableDto> cables
) {
    // Constructor estatico
    public static CircuitoFileDto metaDatosGenerados(Circuito circuito
    ,
            List<PosicionComponenteDto> posiciones, List<CableDto>
    cables) {
        return new CircuitoFileDto(new Metadata(), circuito,
    posiciones, cables);
    }
}
```

El *serializador* crea un tipo de estructura DTO de todos los elementos del circuito (incluidos los metadatos) y los escribe en el archivo usando un objeto `jackson-databind`.

Posteriormente, para recuperar el circuito de un documento JSON, se ha creado una clase para recuperar componentes (`ComponenteRender`) y otra para recuperar cables (`CableRender`). Estas clases llaman al *deserializador* que extrae el texto del archivo en una estructura DTO y a partir de ahí se lee cada valor de cada elemento para reconstruir el componente o cable en el área de dibujo (ver Código 5.16).

Código 5.16: Serialización y escritura en archivo con ObjectMapper.

```
public void guardarCircuitoArchivo(File archivo, Circuito circuito,
    Pane zonaDibujo) throws IOException {
    List<PosicionComponenteDto> posiciones = extraerPosiciones(
    zonaDibujo);
    List<CableDto> cables = extraerCables(zonaDibujo);

    CircuitoFileDto fichero = CircuitoFileDto.metaDatosGenerados(
    circuito, posiciones, cables);

    //Escritura en archivo JSON
    mapper.writeValue(archivo, fichero);
}
```

5.4.2. Base de Datos SQLite

Se ha utilizado una base de datos SQLite para gestionar el contenido de la sección de *E-Learning*. SQLite permite que la base de datos funcione sin necesidad de un servidor, ofrece portabilidad al tratarse de un archivo único que se puede transferir entre diferentes computadoras y no requiere, prácticamente, configuración. Estas razones han motivado la selección de SQLite como motor de base de datos para la aplicación ThevenApp.

Para utilizar SQLite sólo ha sido necesario modificar el archivo `pom.xml` del proyecto Maven y crear la dependencia de forma que sea Maven quien instale y configure SQLite en el proyecto para poder comunicarse con Java.

Al abrir la aplicación, si no existe, se crea el archivo físico que contiene la base de datos. Se ha utilizado la ubicación `System.getProperty("user.home") + /ThevenApp/` para garantizar que la ubicación sea compatible con Windows, Linux o MacOS. Es importante mencionar, que se ha utilizado el patrón **Singleton** para garantizar una única instancia de la base de datos (ver Sección 4.4.3).

Para establecer la comunicación entre Java y SQLite se utiliza el controlador **JDBC** (Java Database Connectivity). La cadena de conexión a la base de datos es la que se muestra en el Código 5.17.

Código 5.17: Conexión JDBC a SQLite.

```
//Clase DatabaseService
private static final String APP_FOLDER=System.getProperty("user.home")
    + File.separator + "ThevenApp";
private static final String DB_URL="jdbc:sqlite:" + APP_FOLDER + File.
   separator + "thevenapp_database.db";

public Connection getConnection() throws SQLException {
    return DriverManager.getConnection(DB_URL);
}
```

Para controlar los accesos a la base de datos se utiliza la clase `LearningService` para evitar que lo hagan los controladores de la interfaz gráfica y, así, prevenir errores de acceso. Además se han creado clases específicas para los elementos almacenables (usuarios, ejercicios, teoría) que encapsulan las operaciones CRUD (*Create, Read, Update, Delete*). Cuando se accede a la base de datos para obtener información, se realiza una consulta SQL y con los datos obtenidos se crea un objeto Java con los parámetros obtenidos.

Se ha evitado en todo momento el uso de inyección SQL pura, en primer lugar, por seguridad y, en segundo lugar, para evitar caer en mala praxis, asegurar las buenas prácticas en el proyecto y evitar que usuarios malintencionados rompan la base de datos con código SQL puro. Por esta razón, para cualquier acceso, consulta, modificación o actualización de la base de datos se utiliza `PreparedStatement` que utiliza marcadores de posición (?) en lugar de valores directos para ejecutar sentencias SQL precompiladas. El Código 5.18 muestra cómo se realiza una consulta de autenticación de usuario en la base de datos.

Código 5.18: Uso de PreparedStatement para consultas seguras.

```
//Metodo autenticarUsuario en UserData.java
String sql="SELECT id, username, role FROM users WHERE username = ?";

try (Connection conn=db.getConnection();
     PreparedStatement pstmt=conn.prepareStatement(sql)) {
    pstmt.setString(1, username);

    ResultSet rs=pstmt.executeQuery();
    if (rs.next()) {
        // ... procesar resultado ...
    }
}
```

Se debe mencionar en esta sección, que las contraseñas no se almacenan como tales en la base de datos sino que se almacena el código hash para poder obtener la contraseña cifrada con el protocolo de seguridad SHA-256 lo que las hace, completamente, seguras e irrecuperables si el usuario la desconoce permitiendo, exclusivamente, crear una nueva si conoce la respuesta de seguridad con la que fue registrado.

5.5. Implementación de Patrones de Diseño

En la Sección 4.4.3 se muestra un resumen de los patrones de diseño aplicados en este proyecto. El uso de ellos aporta soluciones a problemas frecuentes en el código con la intención de simplificarlo, mejorarlo o reutilizarlo.

En esta sección se muestran pequeños fragmentos de código aplicando algunos de los patrones mencionados para mostrar cómo y en qué contexto se han utilizado.

5.5.1. Singleton

El patrón *Singleton* se utiliza para conseguir que algunas clases críticas sean instanciadas una sola vez y aseguremos que no haya ningún objeto más. Se ha utilizado en la clase `Tierra` para garantizar que en un circuito sólo haya un nodo de referencia.

También se ha utilizado en la creación de la base de datos para asegurar la consistencia de la misma permitiendo una única instancia que es la que utilicen las clases que tengan que llamarla. Esto mismo ocurre con el gestor de operaciones *Undo/Redo* para garantizar que es único de forma que sólo haya una llamada *deshacer/rehacer* y evitar que más gestores realicen operaciones de este tipo al mismo tiempo (ver Código 5.19).

Código 5.19: Implementación Singleton en UndoRedoManager.

```
//Clase UndoRedoManager
private static UndoRedoManager instancia;

private UndoRedoManager() {
    actualizarPropiedades();
}

public static UndoRedoManager getInstance() {
    if (instancia == null) {
        instancia = new UndoRedoManager();
    }
    return instancia;
}
```

5.5.2. Observer

El patrón *Observer* ha sido utilizado para actualizar un objeto al realizar un cambio en otra parte. Para ello, se han asignado variables tipo *Property* que pueden ser "escuchadas" (por ejemplo, el valor de una resistencia) por *listeners* que informan a otro elemento para que sea actualizado (por ejemplo, la etiqueta de un componente) (ver Código 5.20).

Código 5.20: Patrón Observer con Listeners en JavaFX.

```
//En EtiquetaBindingService.java
public static Label construirEtiqueta(Componente comp) {
    Label label=new Label();

    //Listener que observa cambios en la propiedad 'carga'
    comp.cargaProperty().addListener((obs, oldVal, newVal) -> {
        updateStyle(label, newVal);
        logger.fine(() -> "Etiqueta de " + comp.getId() + " cambio
   estilo a " + newVal);
    });

    //Binding (Observer unidireccional) para el texto
    bindValorConUnidad(label, comp);
    return label;
}
```

5.5.3. Factory

El patrón *Factory* ha sido utilizado para facilitar la creación de componentes para que, con una sola llamada al *Factory Method*, ésta sepa qué tipo de componente debe crear y con qué parámetros.

Este patrón facilita precisamente que no haya una llamada diferente para cada tipo de componente sino una única llamada para cualquier tipo de componente (ver Código 5.21).

Código 5.21: Patrón Factory para creación de componentes visuales.

```
//En ComponenteVisualFactory.java
public static StackPane crearVisual(
        String tipoIcono, double x, double y, ConectorPuntos conector,
        Pane areaDibujo,
        // ... otros parametros) {
    //Identificar el tipo basado en el icono
    TipoComponenteVisual tipo=TipoComponenteVisual.fromIcono(tipoIcono
   );
    ImageView icono=crearIcono(tipo);

  //Crear la instancia del dominio
    Componente componente=tipo.crearComponente();

    //Construir el panel contenedor
    StackPane panel=construirPanel(tipo, icono, x, y, componente);

    // ...codigo adicional...
    return panel;
}
```

5.5.4. Builder

El patrón *Builder* se ha utilizado para reducir la complejidad en la construcción de objetos `Cable`. Dado el número de parámetros de configuración que requiere la creación de cables, el uso de un constructor tradicional resulta poco legible. De esta manera, se ha implementado un constructor sencillo y fluido que permite crear cables de una manera legible y clara (ver Código 5.22).

Código 5.22: Patrón Builder para construcción de cables.

```
// Clase CableBuilder.java
public class CableBuilder {
    public CableBuilder desde(PuntoConexion inicio) {
        this.inicio = inicio;
        return this;
    }
    public CableBuilder hasta(PuntoConexion fin) {
        this.fin = fin;
        return this;
    }
    public CableBuilder en(Pane zonaDibujo) {
        this.zonaDibujo = zonaDibujo;
        return this;
    }
    public Cable construir() {
        if (inicio == null || fin == null)
            throw new IllegalStateException("Faltan parametros
   obligatorios.");
        Cable cable = new Cable(inicio, fin, zonaDibujo, conector);
        return cable;
    }
}
```

Para construir un cable con el patrón Builder basta con escribirlo como se detalla en el Código 5.23.

Código 5.23: Construcción de un cable usando el patrón Builder.

```
  Cable cable=new CableBuilder()
    .desde(a)
    .hasta(b)
    .en(zonaDibujo)
    .usando(this)
    .construir();
  zonaDibujo.getChildren().add(cable);
```

5.6. Retos y Soluciones

La creación de una aplicación de este estilo requiere una planificación previa y un continuo crecimiento para mejorar decisiones que, a priori, se habían tomado como buenas y, durante el desarrollo, comprobar que hay soluciones mejores que se deben implementar.

Como ejemplo de esto, en un principio se barajó la opción de asignación manual de nodos pero, dado que es una aplicación diseñada para la enseñanza, esta decisión llevaría a múltiples errores de conexiones y a circuitos mal configurados. Por esta razón, finalmente se optó por realizar una asignación automática de nodos asignando obligatoriamente el nodo 0 para la referencia de tierra y de manera ordenada reasignar los nodos siguientes como 1, 2, 3, etc.

Con la intención de que no hubiera nodos con números muy altos si se eliminaban componentes del diseño, se implementó un gestor de nodos que, además de realizar la reasignación de nodos al conectar tierra, almacenara un nodo eliminado como el primero en ser asignado de forma que se reutilicen los nodos menores no utilizados los primeros.

Otro problema que complicó la implementación de la aplicación fue diseñar un sistema de nodos eléctricos reales.

Físicamente, en un circuito, *nodos* con diferentes posiciones visuales son un mismo nodo porque comparten una conexión lógica; por ejemplo, las múltiples ramas que llegan al nodo de referencia terminan en nodo 0. El sistema debía entender de alguna manera que todos los puntos de conexión del final de cada rama son un mismo nodo eléctrico, es decir, comparten la misma conexión lógica aunque, visualmente, diferente (ver Figura 5.4).

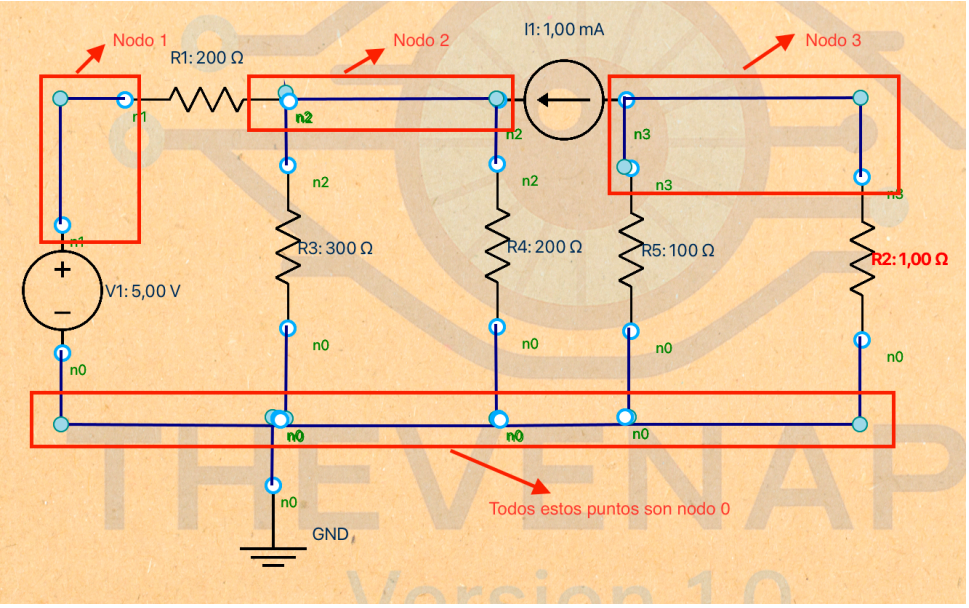


Figura 5.4: Representación de puntos que pertenecen a un mismo nodo eléctrico.

Para solucionar esto se creó un pequeño sistema de **Net** que representa una red eléctrica de forma que en un circuito se crean tantas *nets* como nodos reales haya y cada punto de conexión creado se asigna a su net correspondiente. De esta forma, sólo con conocer a qué *net* pertenece un punto de conexión sabemos dónde y con quién comparte conexión. Esta solución ha resultado ser una pieza angular del proyecto permitiendo separar la representación visual (dónde está el componente) de su representación topológica (cómo está conectado eléctricamente).

5.7. Conclusiones

La implementación de la aplicación ha supuesto un desafío técnico significativo, abordado mediante una estrategia de desarrollo iterativo solucionando cada problema de forma incremental. Se han diseñado algoritmos y soluciones de código perfectamente válidas para los objetivos marcados.

Algunas soluciones han priorizado la claridad y la mantenibilidad frente a la optimización del coste computacional, pero a un nivel educativo el rendimiento es óptimo. Por tanto, el resultado ha sido positivo y refleja la madurez técnica adquirida durante el desarrollo del proyecto.

Capítulo 6

Pruebas y Validación

El desarrollo de software a nivel profesional requiere superar una fase estricta de verificación y validación con el objetivo de garantizar la calidad del producto. Se pretende con ello encontrar deficiencias, errores de ejecución que no habían sido detectados durante la fase de compilación, resultados inesperados, anomalías o cualquier detalle que comprometa la calidad o, incluso pueda afectar a la funcionalidad del software desarrollado. Aunque en la industria, es común lanzar al mercado versiones beta para que usuarios finales puedan probarlo en su totalidad y comunicar errores a los desarrolladores que, de otra manera, hubieran sido difíciles de encontrar, en esta aplicación se ha seguido una estrategia de pruebas de diferentes niveles para validar el cumplimiento de los requisitos y asegurar el resultado final.

6.1. Pruebas Unitarias

Las primeras pruebas que requiere la aplicación sirven para validar que el motor de cálculo de la aplicación funciona correctamente; es una forma de comprobar que con cualquier tipo de circuito, el algoritmo construye las matrices correctamente y resuelve el sistema de ecuaciones adecuadamente obteniendo los valores de tensión y resistencia adecuados, en el caso de Thévenin o de corriente, en el caso de Norton.

Para estas pruebas se ha usado la librería externa **JUnit 5** [20] que permite automatizar este tipo de pruebas en Java y verificar de forma rápida y segura que todo funciona correctamente.

Para ello, se ha creado un archivo java con diferentes pruebas que han permitido validar el motor de cálculo. El archivo utilizado `TheveninTest` contiene las siguientes pruebas unitarias:

1. **Test 0 - Divisor de tensión**: calcula Thévenin en un circuito con una fuente de tensión independiente y dos resistencias.
2. **Test 1 - Circuito mixto simple**: calcula Thévenin en un circuito con una fuente de tensión independiente, una fuente de corriente independiente y cuatro resistencias.
3. **Test 2 - Fuentes mixtas**: calcula Thévenin en un circuito con una fuente de tensión independiente, una fuente de corriente independiente y cuatro resistencias.
4. **Test 3 - Fuentes mixtas**: calcula Thévenin en un circuito similar a test 1 y test 2 pero con una disposición diferente.
5. **Test 4 - Múltiples fuentes de corriente**: calcula Thévenin en un circuito que contiene tres fuentes de corriente independientes, una fuente de tensión independiente y cuatro resistencias.
6. **Test 5 - Norton**: calcula Norton en un circuito con una fuente de tensión independiente, una fuente de corriente independiente y cuatro resistencias.
7. **Test 6 - Circuito complejo Norton**: calcula Norton en un circuito con tres fuentes de corriente independientes y cinco resistencias.
8. **Test 7 - Valores tipo String**: calcula Norton, en un circuito simple pero con los valores no numéricos sino tipo String con multiplicadores.
9. **Test 8 - Fuente CCVS**: calcula Thévenin en un circuito con una fuente de corriente independiente, una fuente de tensión controlada por corriente y tres resistencias.

10. **Test 9 - Fuente VCCS**: calcula Thévenin en un circuito con una fuente de corriente independiente, una fuente de corriente controlada por tensión y cuatro resistencias con valores tipo String.
11. **Test 10 - Fuente VCVS**: calcula Thévenin en un circuito con una fuente de corriente independiente, una fuente de tensión controlada por tensión y cuatro resistencias.
12. **Test 11 - Circuito complejo**: calcula Thévenin en un circuito con dos fuentes de tensión independientes, una fuente de corriente independiente y ocho resistencias.
13. **Test 12 - Fuente CCCS**: calcula Thévenin en un circuito con una fuente de tensión independiente, una fuente de corriente controlada por corriente y tres resistencias.

El resultado de las pruebas fue exitoso y, por tanto, quedó validado el motor de cálculo de la aplicación. En la Figura 6.1 se muestra un test JUnit realizado para la aplicación superado sin errores.

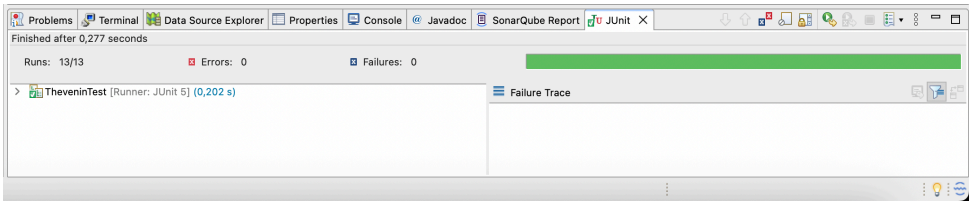


Figura 6.1: Test JUnit superado en la aplicación.

6.2. Pruebas Funcionales

En esta sección se detallan las **pruebas de caja negra** que pretenden verificar los requisitos funcionales que se establecieron para la aplicación y, por tanto, se pueda validar la aplicación (ver Tabla 3.1).

6.2.1. Pruebas del Área de Diseño

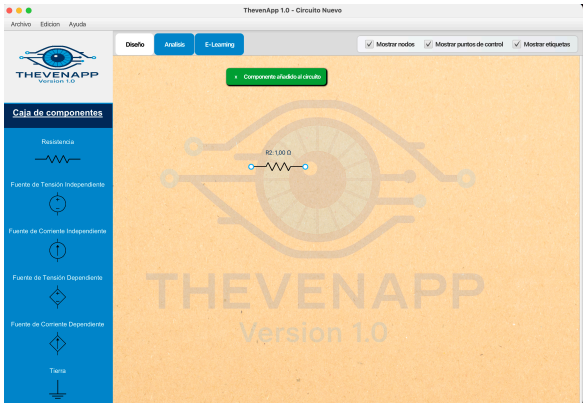
Las pruebas de diseño permiten comprobar que el usuario puede realizar un esquema de un circuito arrastrando componentes desde una paleta de componentes, puede realizar el cableado correspondiente y cambiar las propiedades de cualquier componente. Los casos de prueba para esta sección se encuentran recopilados en la Tabla 6.1.

Tabla 6.1: Casos de prueba para el área de Diseño.

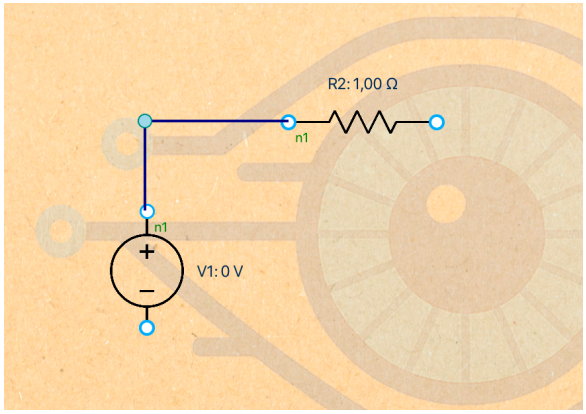
ID	Descripción	Pasos a realizar	Resultado Esperado	Estado
Test-01	*Drag and Drop* de componentes.	1. Seleccionar una resistencia del panel de componentes. 2. Arrastrar hacia el área de diseño. 3. Soltar el botón del ratón.	La resistencia aparece en la posición donde se soltó el ratón.	**OK**
Test-02	Conexión de cables.	1. Hacer clic en el terminal de un componente. 2. Hacer clic en el terminal de otro componente.	Se dibuja un cable uniendo ambos puntos con trayectoria ortogonal.	**OK**
Test-03	Edición de propiedades.	1. Seleccionar una resistencia. 2. Cambiar valor a 500Ω. 3. Aceptar cambio.	La etiqueta del componente se actualiza a 500Ω.	**OK**
Test-04	Gestión de Nodos.	1. Conectar varios componentes a un mismo punto común.	El sistema asigna el mismo identificador de nodo (Net) a los tres terminales.	**OK**
Test-05	Conexión Tierra.	1. Conectar componente Tierra al circuito.	El sistema asigna el nodo 0 a esta conexión y reasigna el resto de nodos.	**OK**
Test-06	Conexión mismo componente.	1. Conectar las dos terminales de un mismo componente.	El sistema no lo permite e informa visualmente al usuario.	**OK**

En la Figura 6.2 se muestran diferentes pruebas realizadas en la aplicación:

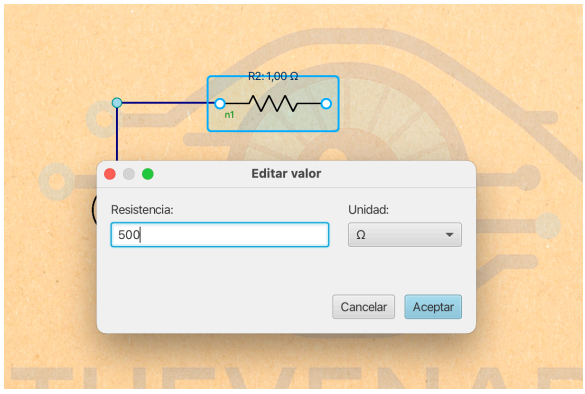
- Figura 6.2a muestra el resultado de realizar el *Test-01* de la Tabla 6.1.
- Figura 6.2b muestra el resultado de realizar el *Test-02* de la Tabla 6.1.
- Figura 6.2c muestra el resultado de realizar el *Test-03* de la Tabla 6.1.
- Figura 6.2d muestra el resultado de realizar el *Test-05* de la Tabla 6.1.



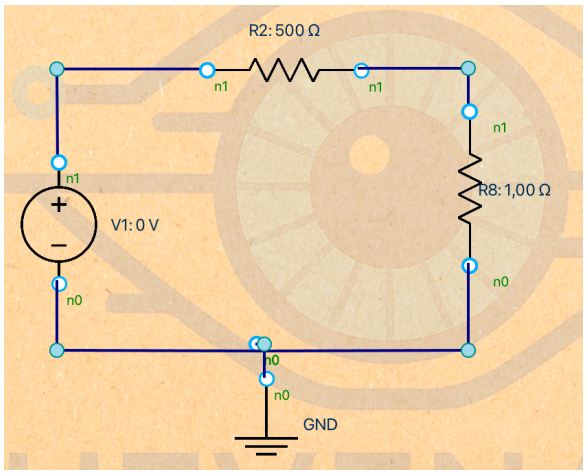
(a) Selección y arrastre de resistencia desde el panel de componentes.



(b) Conexión de dos componentes.



(c) Edición de valores en el panel de propiedades.



(d) Conexión de Tierra y reasignación de nodos.

Figura 6.2: Capturas de la realización de pruebas funcionales del área de Diseño.

6.2.2. Casos de prueba para el área de Análisis.

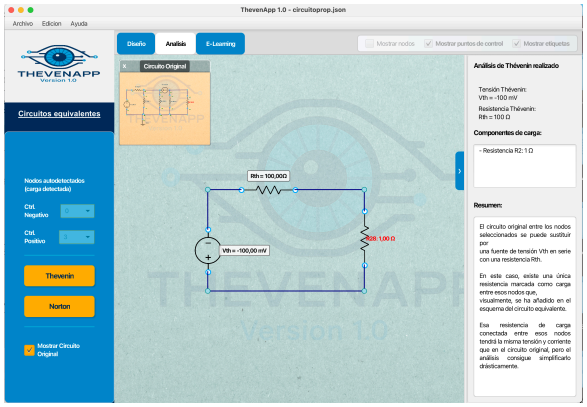
Estas pruebas validan la ejecución de los algoritmos de Thévenin y Norton y la visualización del circuito equivalente. En la Tabla 6.2 se muestran los test de casos de prueba realizados para la sección de Análisis.

Tabla 6.2: Casos de prueba para el área de Análisis.

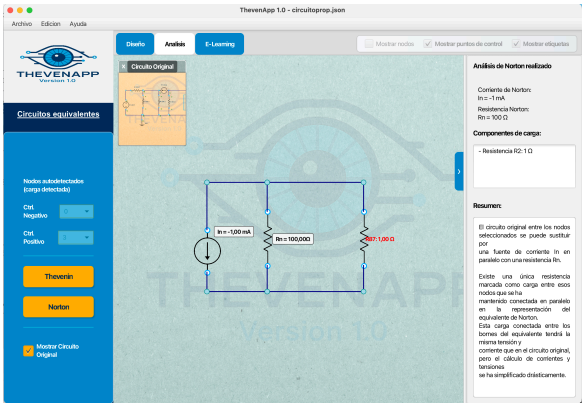
ID	Descripción	Pasos a realizar	Resultado Esperado	Estado
Test-07	Cálculo de Thévenin con carga.	1. Cargar un circuito válido con resistencia de carga. 2. Pulsar botón "Thévenin".	Se muestra el circuito equivalente de Thévenin con los valores correctos.	**OK**
Test-08	Cálculo de Thévenin sin carga.	1. Cargar un circuito sin componente de carga. 2. Selecciona nodos de análisis manualmente. 3. Pulsar botón "Thévenin".	Se muestra el circuito equivalente de Thévenin con los valores correctos.	**OK**
Test-09	Cálculo de Norton con carga.	1. Cargar un circuito válido con resistencia de carga. 2. Pulsar botón "Norton".	Se muestra el circuito equivalente de Norton con los valores correctos.	**OK**
Test-10	Cálculo de Norton sin carga.	1. Cargar un circuito sin componente de carga. 2. Selecciona nodos de análisis manualmente. 3. Pulsar botón "Norton".	Se muestra el circuito equivalente de Norton con los valores correctos.	**OK**
Test-11	Validación de Tierra.	1. Eliminar el componente de Tierra del diseño. 2. Intentar acceder a la pestaña de Análisis.	El sistema muestra un mensaje de error indicando “Falta referencia de tierra” y bloquea el análisis.	**OK**

En la Figura 6.3 se muestran diferentes pruebas realizadas en la aplicación:

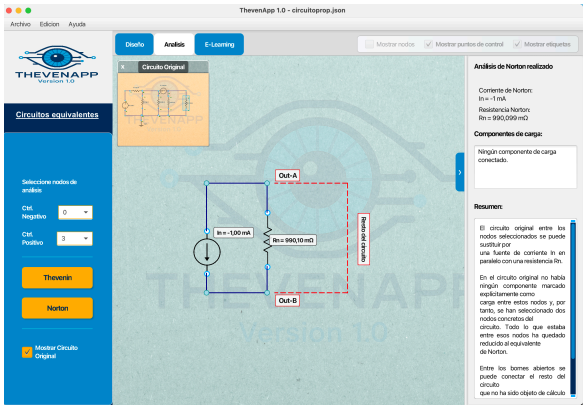
- Figura 6.3a muestra el resultado de realizar el *Test-07* de la Tabla 6.2.
- Figura 6.3b muestra el resultado de realizar el *Test-09* de la Tabla 6.2.
- Figura 6.3c muestra el resultado de realizar el *Test-10* de la Tabla 6.2.
- Figura 6.3d muestra el resultado de realizar el *Test-11* de la Tabla 6.2.



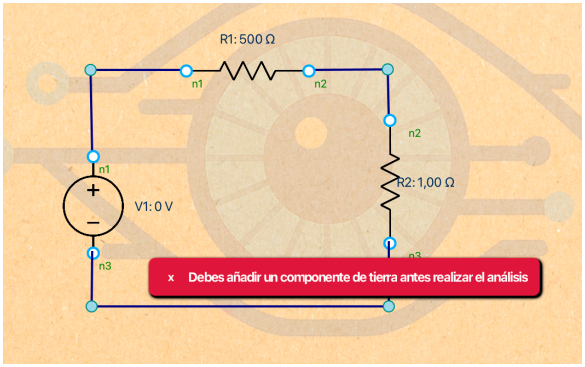
(a) Análisis de Thévenin con resistencia de carga.



(b) Análisis de Norton con resistencia de carga.



(c) Análisis de Norton sin carga (Selección manual de nodos).



(d) Validación de error en tiempo real (falta nodo de Tierra).

Figura 6.3: Capturas de la realización de pruebas funcionales del área de Análisis.

6.2.3. Casos de prueba para el área de *E-Learning*

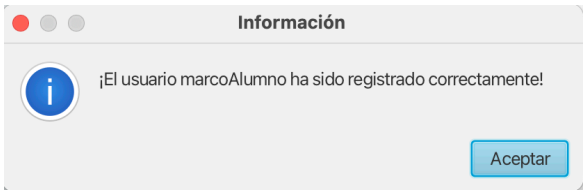
Estas pruebas verifican la gestión de usuarios y los roles de estudiante o profesor. Además verifican la interacción con la base de datos y el uso de la aplicación para la creación de ejercicios. En la Tabla 6.3 se muestran los test de casos de prueba realizados para la sección de *e-learning*.

Tabla 6.3: Casos de prueba para el área de E-Learning.

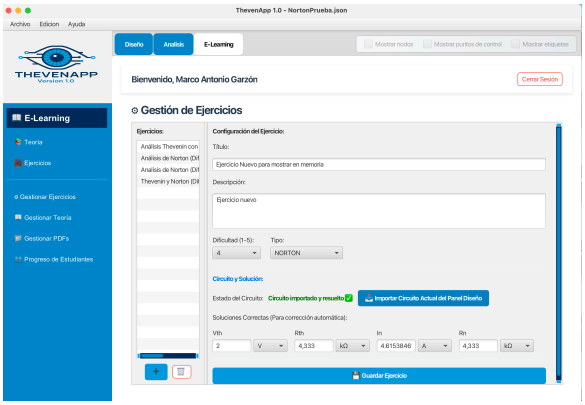
ID	Descripción	Pasos a realizar	Resultado Esperado	Estado
Test-12	Registro y autenticación de usuario.	1. Acceder a formulario de registro. 2. Introducir datos válidos y rol “Estudiante”. 3. Acceder al sistema mediante el login.	El sistema confirma el registro, autentica al usuario y permite el acceso.	**OK**
Test-13	Creación de Ejercicio (Profesor).	1. Login como Profesor. 2. Diseñar circuito. 3. Guardar como ejercicio propuesto.	El ejercicio aparece disponible en la lista de ejercicios de los alumnos.	**OK**
Test-14	Progreso del Estudiante.	1. Login como Estudiante. 2. Resolver correctamente un ejercicio. 3. Consultar "Mi Progreso".	Se actualiza el número de ejercicios completados y la barra de progreso.	**OK**
Test-15	Creación de tema (Profesor).	1. Login como Profesor. 2. Crear un tema de teoría. 3. Publicarlo.	El tema aparece en la lista de temas teóricos de los alumnos.	**OK**
Test-16	Leer tema de teoría (Estudiante).	1. Login como Estudiante. 2. Acceder a un tema de la lista. 3. Marcar como leído.	Se actualiza el número de temas leídos y la barra de progreso.	**OK**
Test-17	Subir un documento PDF (Profesor).	1. Login como Profesor. 2. Subir un documento PDF	El PDF aparece en la lista de PDFs de los alumnos.	**OK**
Test-18	Calificar a un alumno (Profesor).	1. Login como Profesor. 2. Seleccionar un alumno de la lista. 3. Poner una calificación y un comentario.	La calificación y el comentario aparecerán en la sección del alumno.	**OK**
Test-19	Abrir un documento PDF (Estudiante).	1. Login como Estudiante. 2. Abrir un documento PDF de la lista.	El documento se abre en un visor externo.	**OK**

En la Figura 6.4 se muestran diferentes pruebas realizadas en la aplicación:

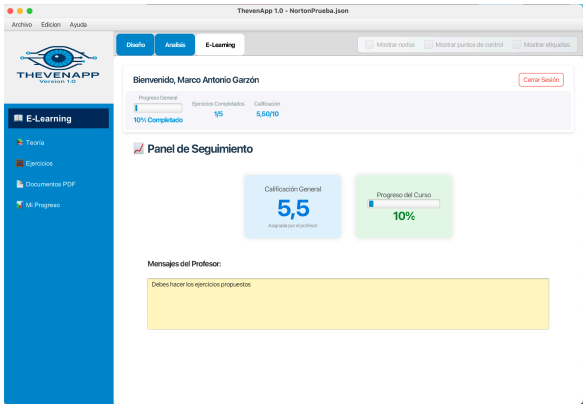
- Figura 6.4a muestra el resultado de realizar el *Test-12* de la Tabla 6.3.
- Figura 6.4b muestra el resultado de realizar el *Test-13* de la Tabla 6.3.
- Figura 6.4c muestra el resultado de realizar el *Test-14* de la Tabla 6.3.
- Figura 6.4d muestra el resultado de realizar el *Test-15* de la Tabla 6.3.
- Figura 6.4e muestra el resultado de realizar el *Test-18* de la Tabla 6.3.
- Figura 6.4f muestra el resultado de realizar el *Test-19* de la Tabla 6.3.



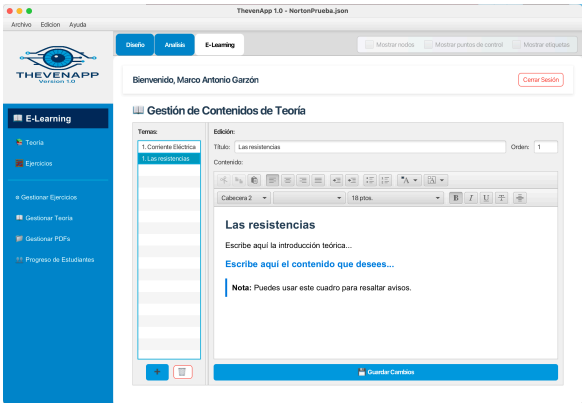
(a) Comprobación de registro de usuarios.



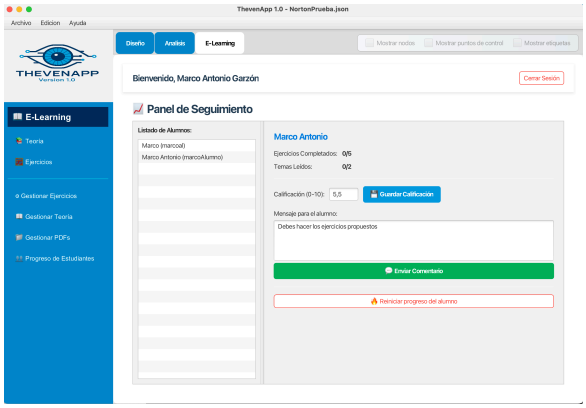
(b) Creación de Ejercicio (Profesor).



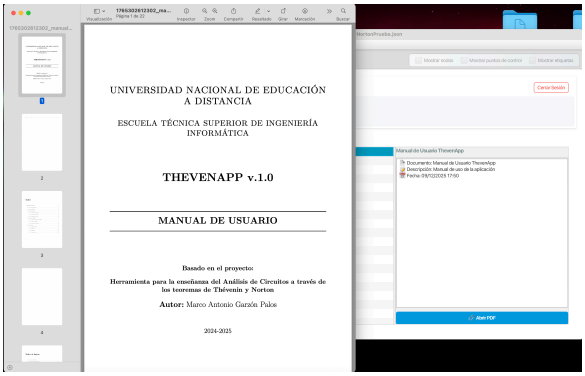
(c) Progreso de estudiante actualizado.



(d) Creación de nuevo tema teórico (Profesor).



(e) Calificar a un alumno (Profesor).



(f) Abrir un documento PDF de la lista.

Figura 6.4: Capturas de la realización de pruebas funcionales del área de *E-Learning*.

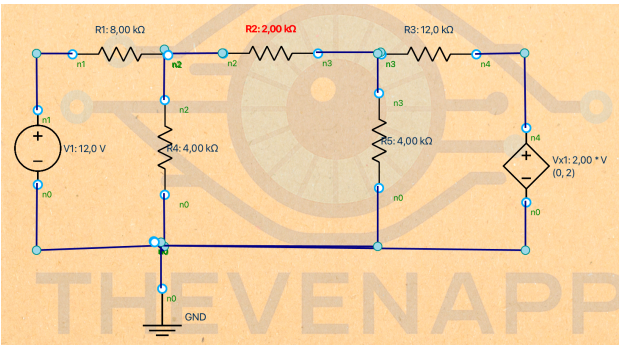
6.2.4. Pruebas de logs y persistencia

En esta sección se verifican los requisitos de almacenamiento de circuitos, la carga de diseños desde un archivo existente y el correcto funcionamiento del sistema de logs para depuración. En la Tabla 6.4 se muestran los test de casos de prueba del sistema de logs y del sistema de persistencia de la aplicación. En la Figura 6.5 se muestran diferentes pruebas realizadas en la aplicación:

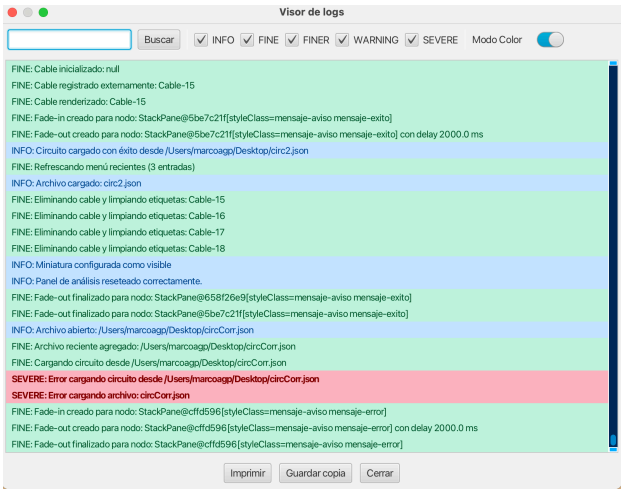
- Figura 6.5a muestra el resultado de realizar el *Test-20* de la Tabla 6.4.
- Figura 6.5b muestra el resultado de realizar el *Test-21* de la Tabla 6.4.

Tabla 6.4: Casos de prueba de logs y persistencia.

ID	Descripción	Pasos a realizar	Resultado Esperado	Estado
Test-20	Persistencia (Guardar/Cargar).	1. Diseñar un circuito. 2. Guardar en archivo JSON. 3. Reiniciar app y cargar archivo.	El circuito se restaura exactamente igual (componentes, posiciones y conexiones).	**OK**
Test-21	Control de Errores (Log).	1. Forzar un error (ej. cargar archivo corrupto). 2. Abrir visor de logs.	El error aparece registrado en el panel de logs con nivel SEVERE/WARNING.	**OK**



(a) Carga de un circuito guardado previamente.



(b) Log generado por intento de carga de archivo corrupto.

Figura 6.5: Capturas de la realización de pruebas de persistencia y de logs.

6.3. Matriz de Trazabilidad

Por último, para asegurar que se han cumplido todos los requisitos propuestos y garantizar la calidad de la aplicación, en la Tabla 6.5 se muestra la **matriz de trazabilidad (RTM)** que relaciona los requisitos funcionales establecidos en la Sección 3.1 con los casos de prueba realizados que se detallan en la Sección 6.2 y las pruebas unitarias realizadas que se detallan en la Sección 6.1.

Tabla 6.5: Matriz de Trazabilidad de Requisitos vs. Pruebas.

ID Req.	Descripción del Requisito	Prueba Asociada	Estado
Módulo de Diseño			
RF-DIS-01 RF-DIS-02	Panel de componentes y funcionalidad *drag and drop*.	*Test-01*	**OK**
RF-DIS-03	Manipulación (arrastrar, mover, escalar, eliminar).	*Test-01*	**OK**
RF-DIS-04	Edición de propiedades de componentes.	*Test-03*	**OK**
RF-DIS-05	Conexión de componentes mediante cables.	*Test-02*, *Test-04*	**OK**
RF-DIS-06	Avisos ante acciones críticas (incluye validación de Tierra obligatoria).	*Test-06*, *Test-11*	**OK**
Módulo de Análisis			
RF-ANA-01 RF-ANA-02	Selección de tipo de análisis y configuración de nodos.	*Test-08*, *Test-09*	**OK**
RF-ANA-03	Cálculo matemático del circuito equivalente.	Pruebas JUnit	**OK**
RF-ANA-04	Representación gráfica del resultado y resumen del análisis.	*Test-07*, *Test-09*	**OK**
Módulo E-Learning			
RF-EDU-01 RF-EDU-03	Gestión de usuarios y roles (Alumno/Profesor).	*Test-12*	**OK**
RF-EDU-02	Espacio interactivo y base de datos.	*Test-13*, *Test-15*	**OK**
RF-EDU-04	Acceso y seguimiento del alumno.	*Test-14*, *Test-16*	**OK**
RF-EDU-05	Publicación de material (Profesor).	*Test-15*, *Test-17*	**OK**
RF-EDU-06	Proponer ejercicios propios (Profesor).	*Test-13*	**OK**
Persistencia y Sistema			
RF-PER-01 RF-PER-02	Guardar y cargar circuitos en archivo JSON.	*Test-20*	**OK**
RF-PER-03	Registro de archivos recientes.	*Test-21*	**OK**

6.4. Conclusiones

Las pruebas realizadas han sido fundamentales para verificar el correcto funcionamiento de la aplicación y la herramienta básica para garantizar un software de calidad que cumple con los requisitos establecidos. Al tratarse de una aplicación orientada a fines educativos, la precisión obtenida en los cálculos es un factor clave. La validación de las pruebas confirma que **ThevenApp** es un recurso didáctico fiable para apoyar a los estudiantes a entender los conceptos para los que fue diseñada.

Capítulo 7

Conclusiones y Trabajos Futuros

*El desarrollo de este proyecto ha culminado en la creación de la aplicación **ThevenApp**, una herramienta de software diseñada para la enseñanza del Análisis de Circuitos a través de los teoremas de Thévenin y Norton, que cumple de forma satisfactoria la necesidad de apoyar la docencia en esta materia. El proyecto ha logrado integrar diferentes tecnologías, recursos y librerías para adaptarlas a una solución software de calidad con fines educativos.*

7.1. Conclusiones Generales

Actualmente, los grados de física e ingeniería enfocados en electrónica, informática, telecomunicaciones o tecnologías de la información requieren una serie de conocimientos sobre disciplinas complejas sobre matemáticas, física o, en el caso de este proyecto, el análisis de circuitos eléctricos. Los estudiantes perciben ciertas disciplinas como auténticos desafíos a pesar del material disponible.

Cada día se consiguen más herramientas educativas que ayudan al crecimiento de los alumnos pero algunas disciplinas como Análisis de Circuitos sólo disponen de herramientas profesionales difíciles de aprender o manejar. De ahí surge la necesidad de poner a disposición de profesores y estudiantes una herramienta que cumpla con las necesidades de un alumno en su aprendizaje.

En esta línea, se ha desarrollado una solución software que sirva de apoyo a la docencia en materia de Análisis de Circuitos y que sea una plataforma interactiva donde los profesores puedan poner a disposición de los alumnos todo tipo de material relacionado.

ThevenApp consigue paliar esa necesidad por diversas razones:

- Cuenta con un motor matemático robusto para realizar cálculos complejos con matrices.
- Tiene una arquitectura mantenible y ampliable, entre otras razones, por el uso del patrón Modelo-Vista-Controlador.
- Tiene una interfaz agradable y sencilla con una usabilidad notable debido a soluciones como *drag and drop*, abstracción de cálculo al usuario final o a la plataforma didáctica.

El resultado final ha logrado satisfacer la necesidad o el problema que inició su planteamiento.

7.2. Cumplimiento de Objetivos

En la Sección 1.2 se establecieron unos objetivos fundamentales que debía cumplir la aplicación diseñada para este proyecto. En esta sección se muestra que todos los objetivos específicos han sido cumplidos.

El **objetivo principal** de este proyecto era conseguir una herramienta completamente funcional orientada a fines educativos y, teniendo en cuenta, el resultado de su implementación y la superación de las pruebas funcionales a las que ha sido sometida, **ThevenApp** es la prueba de haberlo cumplido de forma satisfactoria.

Además, la aplicación debía cumplir ciertos objetivos para que, realmente, fuera una herramienta didáctica, usable y mantenible.

- **Diseño**: se ha logrado realizar un diseño sencillo de un sistema tipo *drag and drop* (arrastrar y soltar) para realizar esquemas de circuitos eléctricos.
- **Análisis**: se ha implementado un motor de cálculo basado en el método de *Análisis Nodal Modificado* para analizar los circuitos propuestos y ofrecer como resultado final un circuito equivalente más simple aplicando los teoremas de Thévenin y Norton.
- **Educación**: se ha integrado una plataforma de *E-Learning* con gestión de usuarios con diferentes roles (estudiante o profesor) y el uso de la simulación para recrear ejercicios y poner a disposición de los alumnos todo tipo de material educativo.
- **Persistencia**: se ha logrado almacenar diseños de circuitos eléctricos realizados en la aplicación en archivos locales y cargarlos, de nuevo, en la aplicación. Además se ha incluido una base de datos SQLite para gestionar el contenido de la plataforma de *E-Learning*.
- **Usabilidad y Accesibilidad**: se ha conseguido una herramienta sencilla que cumple las normas básicas de usabilidad establecidas por las heurísticas de Nielsen y la norma ISO 9241 sobre diseño centrado en el usuario. Además se ha logrado que la aplicación sea portable de forma que se pueda usar en entornos educativos desde cualquier equipo y cualquier sistema operativo.

7.3. Limitaciones Actuales

La ciencia del Análisis de Circuitos es una disciplina muy amplia que conlleva años de estudio para abordar todo su contenido. Este proyecto se ha centrado en una pequeña parte de ella, pero fundamental para poder avanzar en su estudio. Esta razón justifica el alcance actual que tiene la aplicación desarrollada.

La versión actual de **ThevenApp** se limita a circuitos de **Corriente Continua (DC)** en **régimen permanente**, es decir, puramente resistivos. En esta versión inicial no soporta ningún tipo de componente reactivo como condensadores o bobinas y tampoco componentes no lineales como diodos y transistores aunque la aplicación ha quedado preparada para incluirlo debido a su modularidad.

En circuitos extremadamente complejos (cientos de nodos) el rendimiento podría verse comprometido por el uso de algoritmos con cierta complejidad computacional, sin embargo éste no era el objetivo que se pretendía cumplir, sino la claridad y la sencillez a nivel educativo y, por tanto, para circuitos de un nivel didáctico el rendimiento es óptimo.

7.4. Líneas de mejora futura

La aplicación **ThevenApp** aunque, en la versión actual (1.0), tiene algunas limitaciones tal como se describen en la Sección 7.3, ofrece unas posibilidades enormes de ampliación y mejora para seguir trabajando en ella y que, en un futuro, sea una herramienta que abarque un espectro más amplio de estudio sobre análisis de circuitos. En el futuro sería interesante realizar ampliaciones o mejoras en los siguientes aspectos:

- **Ampliación del motor de cálculo**:
  - Dar soporte a circuitos en **Corriente Alterna (AC)** implementando el trabajo con números complejos de las matrices para tratar fasores e impedancias.
  - Añadir la variable tiempo (t) para realizar análisis en **régimen transitorio** de carga y descarga de componentes.

  - Realizar análisis con diferentes técnicas de resolución de circuitos como reducción serie/paralelo (incluyendo transformaciones delta a estrella y viceversa), transformación de fuentes, método de mallas puro o superposición.
  - Dar soporte a circuitos electrónicos didácticos con uso de placas tipo Arduino o Raspberry.
  - Añadir una ventana de simulación de la excitación de los circuitos para que el alumno pueda ver cómo trabaja su diseño.
- **Incluir nuevos componentes**: añadir la posibilidad de trabajar en circuitos con condensadores, bobinas, diodos, transistores u otros elementos.
- **Mejorar la plataforma de *E-Learning***:
  - Ampliar los tipos de ejercicios a corriente alterna o nuevos componentes si ya se ha realizado la ampliación del simulador.
  - Establecer un modo "examen" con tiempo limitado para probar las capacidades de los alumnos.
  - Convertir la base de datos SQLite local en una base de datos remota (en la nube) para que profesor y alumno se puedan sincronizar en tiempo real.
- **Ampliar la portabilidad**: Realizar una versión Web o versión móvil para iOS o Android (aunque esto implicaría usar una alternativa a JavaFX, pero podría ser una línea futura).

Bibliografía

[1] Real Academia Española, "Diccionario de la lengua española." https://www.rae.es/drae2001/, 2001. Edición 22.ª. [Acceso: 06-08-2025].

[2] Espacio Ciencia, "Corriente eléctrica." https://espaciociencia.com/la-corriente-electrica/, 2023. Tipos y efectos. [Acceso: 06-08-2025].

[3] Electricidad Básica, "Corriente eléctrica: Qué es, tipos, fórmulas y cómo se mide." https://electricidad-basica.com/conceptos-basicos/corriente-electrica/, 2025. [Acceso: 02-09-2025].

[4] P. A. Tipler, *Física para la ciencia y la tecnología Vol.2*. España: Editorial Reverté, 6ª edición ed., 2010.

[5] N. Krishnapura, "Modified nodal analysis." https://www.ee.iitm.ac.in/vlsi/_media/courses/ee2015_2017/mna.pdf, 2017. Análisis nodal modificado. [Acceso: 26-04-2025].

[6] D. E. Johnson, *Análisis básico de circuitos eléctricos*. México: Prentice Hall Hispanoamericana, 5ª edición ed., 1996.

[7] C. Burch, "Logisim 2.7.1." https://cburch.com/logisim/es/index.html, 2001. Simulador lógico. [Acceso: 12-09-2025].

[8] CADe SIMU, "Cade_simu: Simulador eléctrico." https://cade-simu.com/, 1995. [Acceso: 12-09-2025].

[9] Cadence Design Software, "Pspice technology." https://www.cadence.com, 1984. Información comercial. [Acceso: 12-09-2025].

[10] Standards Development Organization, "Unified modeling language (uml)." https://www.omg.org/uml/, 2025. [Acceso: 20-11-2025].

[11] Oracle, "Java." https://www.java.com/es/, 2025. [Acceso: 02-11-2025].

[12] OpenJFX, "Javafx." https://openjfx.io, 2025. Versión Open Source. [Acceso: 02-11-2025].

[13] Oracle, "Máquina virtual de java (jvm)." https://www.java.com/en/download/, 2025. [Acceso: 08-12-2025].

[14] Refactoring Guru, "Catálogo de patrones de diseño." https://refactoring.guru/es/design-patterns/java, 2014. Ejemplos en Java. [Acceso: 04-12-2025].

[15] K. Sharan and P. Späth, *Learn JavaFX 17*. EEUU: Apress, 2ª edición ed., 2022.

[16] SQLite, "Sqlite." https://sqlite.org, 2025. Base de datos. [Acceso: 04-11-2025].

[17] H. F. Korth, A. Silberschatz, and S. Sudarshan, *Fundamentos de Bases de Datos*. España: McGraw-Hill, 6ª edición ed., 2014.

[18] The Apache Software Foundation, "Apache commons math." https://commons.apache.org/proper/commons-math/, 2025. Librería matemática. [Acceso: 24-11-2025].

[19] The Apache Maven Project, "Maven." https://maven.apache.org, 2025. [Acceso: 02-11-2025].

[20] JUnit, "Junit framework." https://junit.org, 2021. Framework de pruebas. [Acceso: 10-09-2025].

[21] Eclipse Foundation, "Eclipse ide." https://eclipseide.org, 2025. Entorno de Desarrollo. [Acceso: 10-08-2025].

[22] M. A. Weiss, *Estructuras de datos en Java*. España: Pearson, 4ª edición ed., 2013.

[23] ProgrammingKnowledge, "Javafx tutorial for beginners." https://www.youtube.com/watch?v=9YrmON6nlEw&list=PLS1QulWo1RIaUGP446_pWLgTZPiFizEMq, 2015. Tutorial de JavaFX. [Acceso: 28-03-2025].

[24] o7planning, "Tutoriales de programación." https://o7planning.org/, 2014. Tutoriales Java. [Acceso: 06-12-2025].

Anexos I

Manual de Usuario

ThevenApp es una aplicación didáctica diseñada para la enseñanza del análisis de circuitos en base a los teoremas de Thévenin y Norton.

Se trata de una herramienta interactiva donde estudiantes y profesores pueden interactuar de forma que los estudiantes tengan a disposición el contenido teórico necesario y puedan practicar con ejercicios propuestos por el profesor haciendo uso de la herramienta.

I.1. Inicio de la aplicación

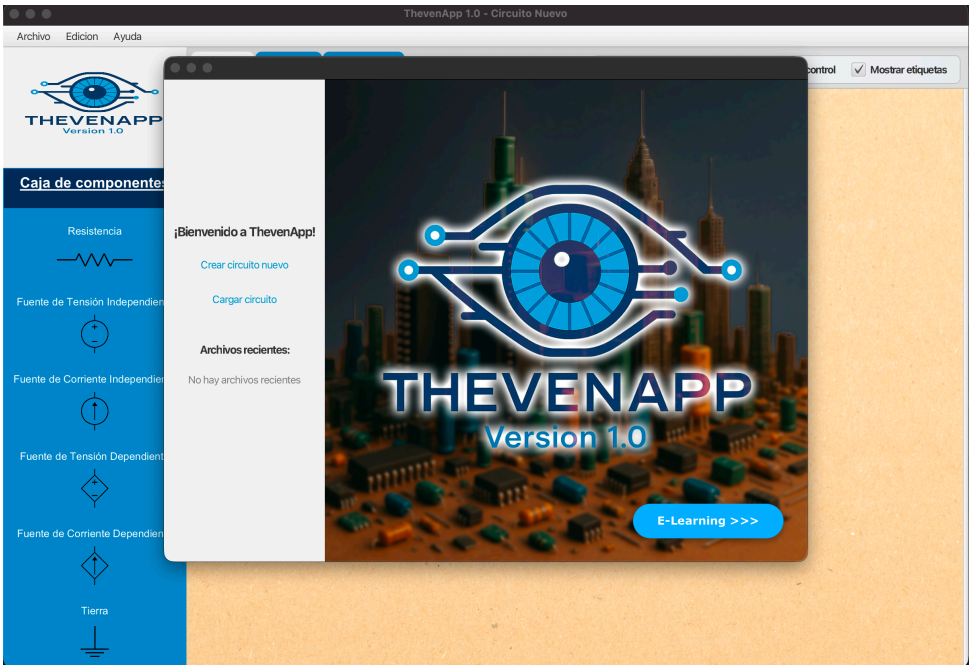


Figura I.1: Inicio de la Aplicación.

La aplicación se inicia con un panel de **Bienvenida** tal como se ve en la Figura I.1 desde el que se accede a la aplicación seleccionando una de las siguientes opciones:

- **Crear circuito nuevo**: Inicia la aplicación creando un circuito nuevo que se podrá diseñar en el **Panel de Diseño**. La aplicación salta directamente a este panel.

- **Cargar circuito**: Esta opción abre un cuadro de diálogo con el explorador de archivos donde se podrá seleccionar un archivo JSON con un circuito creado con la aplicación para cargarlo en el **Panel de Diseño** (ver Sección I.3).
- **Archivos Recientes**: Debajo de los enlaces anteriores aparece una lista de archivos recientes con los últimos archivos modificados en la aplicación. Pulsando cualquiera de ellos se accede al **Panel de Diseño** cargando el circuito contenido en el archivo. Debajo de esta sección hay un enlace **Borrar historial** que permite limpiar la lista de archivos recientes.
- **E-Learning**: En la esquina inferior derecha de este panel se encuentra un botón **E-Learning** que permite acceder a esta sección directamente sin tener que cargar o crear ningún circuito (Ver Sección I.5).

Si se cierra este panel con el botón cerrar, la aplicación finalizará.

I.2. Ventana Principal

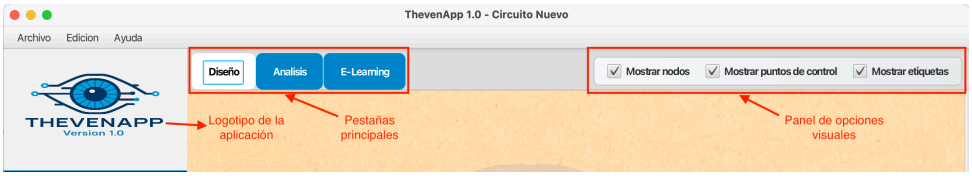


Figura I.2: Panel Principal de ThevenApp.

La ventana principal, tal como se muestra en la Figura I.2, contiene 3 pestañas que dan acceso a las secciones más importantes de la aplicación:

- **Diseño**: es la sección donde se realizan los esquemas de los circuitos (Ver Sección I.3).
- **Análisis**: es la sección donde se muestran los análisis realizados, una vez realizados los cálculos (Ver Sección I.4).
- **E-Learning**: es la plataforma de contenidos donde interactúan profesores y estudiantes (Ver Sección I.5).

En la esquina superior izquierda aparece el logotipo de la aplicación y en la esquina superior derecha se encuentra el panel de **Opciones Visuales**. Este panel sólo está activo si el usuario se encuentra en la sección de **Diseño** y contiene las siguientes opciones:

- **Mostrar Nodos**: muestra u oculta las etiquetas de nodos de color verde asignadas a los puntos de conexión.
- **Mostrar Puntos de Control**: muestra u oculta los puntos de control o puntos de conexión de los elementos que haya en el área de diseño.
- **Mostrar Etiquetas**: muestra u oculta las etiquetas asignadas a los componentes que contienen su ID y su valor asignado (y los nodos de control en caso de fuentes dependientes).

En la parte superior, se encuentra la barra de menús con tres menús desplegables (ver Figura I.3):

- **Menú Archivo** (Figura I.3a): Contiene las siguientes opciones:
  - ***Nuevo***: Crea un circuito nuevo para comenzar su diseño.
  - ***Abrir recientes***: Despliega un nuevo menú con los archivos modificados recientemente en la aplicación. Contiene una opción de *Borrar historial* que borra la lista completa.
  - ***Cargar***: Carga un circuito guardado en disco local.
  - ***Guardar***: Permite guardar el diseño actual en un archivo. Si no se había guardado, crea un archivo nuevo. Si se había guardado, actualiza el archivo guardado.
  - ***Guardar como***: Guarda una copia del diseño actual en un archivo diferente.
  - ***Salir***: Sale de la aplicación.
- **Menú Edición** (Figura I.3b): Contiene las siguientes opciones:

- ***Deshacer***: Deshace las últimas acciones realizadas en el diseño en orden regresivo.
- ***Rehacer***: Rehace las últimas acciones deshechas en el diseño en orden regresivo.
- ***Seleccionar todo***: Selecciona todos los elementos del área de diseño.
- ***Deseleccionar todo***: Deselecciona cualquier elemento que estuviera seleccionado en el área de diseño.
- ***Limpiar área de dibujo***: Limpia el área de diseño completamente borrando todos los elementos que contenga.

- **Menú Ayuda** (Figura I.3c): Contiene las siguientes opciones:
  - ***Ver logs***: Muestra una ventana con todos los mensajes de logs de la sesión actual.
  - ***Ver Manual de Usuario (PDF)***: Muestra el documento PDF con el manual de usuario en un visor externo.
  - ***Acerca de***: Muestra una ventana con información sobre la aplicación.

(a) Menú Archivo. (b) Menú Edición. (c) Menú Ayuda.

Figura I.3: Barra de menús de ThevenApp.

I.3. Sección de Diseño

Esta sección permite realizar los diseños de circuitos y mostrar las opciones visuales que se requieran; para ello, esta sección tiene activo el panel de opciones visuales superior. A la izquierda se encuentra la paleta de componentes disponibles desde donde se pueden arrastrar los componentes al área de dibujo central para agregarlo al circuito actual (ver Figura I.4).

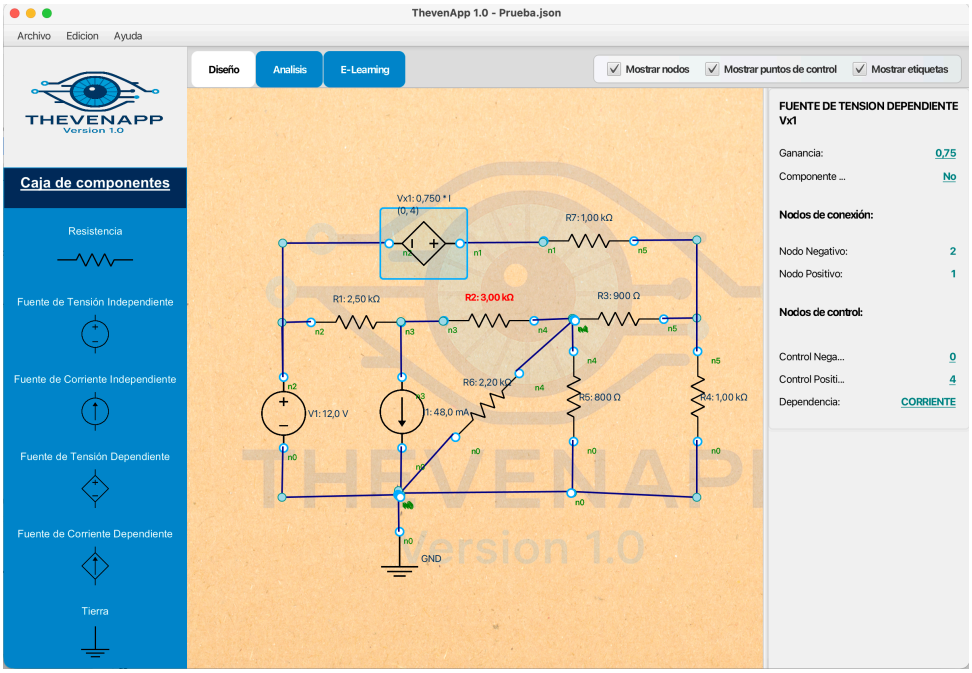


Figura I.4: Sección de Diseño de *ThevenApp*.

Una vez arrastrado un componente, éste se podrá seleccionar haciendo clic sobre él y se podrá mover a cualquier lugar dentro del área de dibujo. Una vez seleccionado se pueden modificar sus propiedades en el panel adyacente que se despliega a la derecha cuando hay un componente seleccionado.

Se pueden seleccionar varios componentes al mismo tiempo de la siguiente manera:

- **Ctrl+Clic (Cmd+Clic en MacOS)**: con esta combinación sobre cualquier componente se añadirá a la selección actual.
- **Caja de selección (marquee)**: desde cualquier lado puede crear una caja de selección haciendo clic y arrastrando el mouse y todos los elementos en su interior quedarán seleccionados.
- **Seleccionar todos**: puede seleccionar todos los componentes del área de dibujo a través del menú *Edición* y la opción *Seleccionar todo* o usando el atajo Ctrl+A (Cmd+A en MacOS).

Cuando hay varios componentes seleccionados el panel de propiedades no se despliega, en cambio puede mover el conjunto seleccionado o eliminarlo.

I.3.1. Panel de Propiedades

El panel de propiedades contiene las opciones modificables de cada elemento; se puede ver en la parte derecha de la figura I.4. Este panel tiene un diseño dinámico dependiendo del componente seleccionado:

- **Resistencia**:
  - Resistencia: se puede modificar su valor resistivo a través del diálogo que se abre introduciendo su valor en ohmios (Ω), kiloohmios (kΩ) o megaohmios (MΩ).
  - Componente de carga: se puede marcar/desmarcar como componente de carga.
  - Nodos de conexión (no modificables): muestra los nodos de conexión asignados automáticamente.
- **Fuente de Tensión Independiente**:
  - Voltaje: se puede modificar su valor de tensión a través del diálogo que se abre introduciendo su valor en voltios (V), kilovoltios (kV) o milivoltios (mV).
  - Componente de carga: salta un error si se intenta marcar una fuente como componente de carga.
  - Nodos de conexión (no modificables): muestra los nodos de conexión asignados automáticamente.

- **Fuente de Corriente Independiente**:
  - Corriente: se puede modificar su valor de corriente a través del diálogo que se abre introduciendo su valor en amperios (A) o miliamperios (mA).
  - Componente de carga: salta un error si se intenta marcar una fuente como componente de carga.
  - Nodos de conexión (no modificables): muestra los nodos de conexión asignados automáticamente.
- **Fuente de Tensión Dependiente**:
  - Ganancia: se puede modificar su factor de ganancia a través del diálogo que se abre al pulsar.
  - Componente de carga: salta un error si se intenta marcar una fuente como componente de carga.
  - Nodos de conexión (no modificables): muestra los nodos de conexión asignados automáticamente.
  - Nodos de control: permite asignar los nodos de control de los que depende la fuente.
  - Dependencia: permite modificar la magnitud de la que depende la fuente (Tensión o Corriente).
- **Fuente de Corriente Dependiente**:
  - Ganancia: se puede modificar su factor de ganancia a través del diálogo que se abre al pulsar.
  - Componente de carga: salta un error si se intenta marcar una fuente como componente de carga.
  - Nodos de conexión (no modificables): muestra los nodos de conexión asignados automáticamente.
  - Nodos de control: permite asignar los nodos de control de los que depende la fuente.
  - Dependencia: permite modificar la magnitud de la que depende la fuente (Tensión o Corriente).
- **Tierra**:
  - Nodo de tierra (no modificable): muestra el nodo de tierra asignado automáticamente (nodo 0).
- **Cable**: muestra información sobre el ID del cable seleccionado y los componentes que conecta.

I.3.2. Conexión de cables

Para conectar dos componentes basta con hacer click en los dos puntos de conexión que se quieren conectar y se creará un cable con la trayectoria que la aplicación considere adecuada. Para ello, cuando el puntero esté sobre un punto de conexión se marcará con una cruz y, al hacer click, ese punto de conexión quedará seleccionado. Cuando hagamos lo mismo en el segundo punto de conexión se creará el cable.

Un cable tiene puntos de control desde los cuales se podrá controlar su trayectoria. Desde estos puntos se podrán mover. Se pueden **crear puntos de control** en cualquier lugar de un cable o **eliminar un punto de control existente** haciendo doble click sobre ellos.

Se puede conectar un componente en un cable sin más que hacer click en el punto de conexión del componente y en cualquier lugar de un cable de forma que se creará un nuevo punto de conexión en el propio cable.

I.3.3. Mensajes de estado

En esta sección se muestran mensajes animados en la parte superior del área de dibujo que duran unos segundos y luego desaparecen.

- *Verde*: son mensajes informativos de una acción realizada con éxito; por ejemplo, crear un circuito nuevo, añadir o eliminar un componente, etc.
- *Amarillo*: son mensajes de precaución que se muestran cuando realizamos una acción que comprometa la integridad del circuito.
- *Rojo*: son mensajes de error cuando una acción genera una excepción o se realiza una acción errónea.

I.4. Sección de Análisis

Una vez diseñado se puede acceder al área de análisis donde se puede seleccionar el tipo de análisis a realizar y el resultado del mismo.

Al acceder a la sección de análisis, el programa verifica que exista un componente Tierra; de lo contrario, mostrará un mensaje de estado en rojo e impedirá acceder al panel. Esta sección tiene un diseño similar a la anterior con un color de fondo diferente (ver Figura I.5).

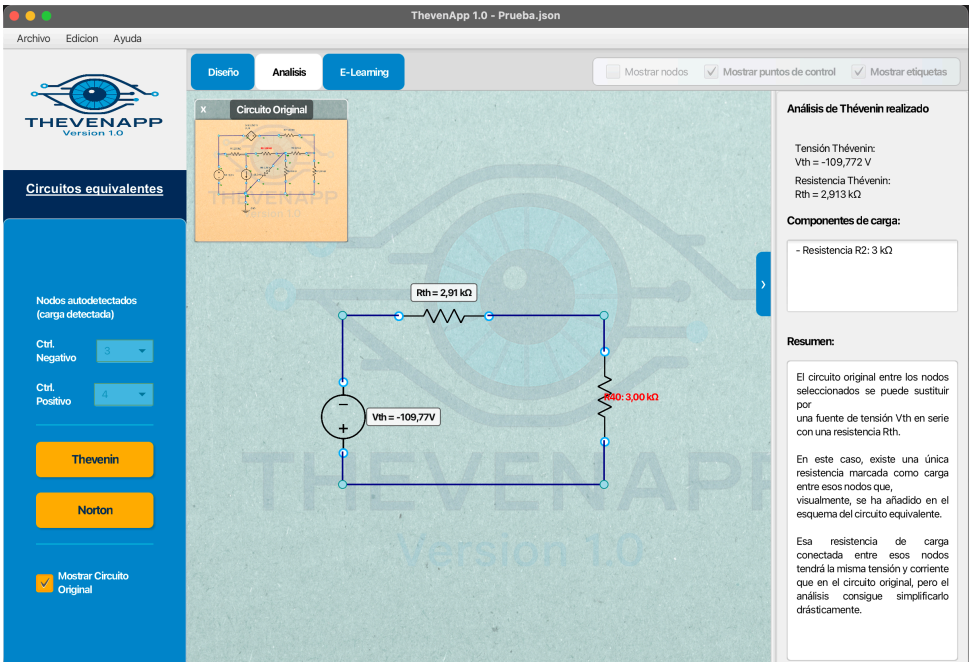


Figura I.5: Sección de Análisis de *ThevenApp*.

I.4.1. Panel de Opciones de Análisis

En la parte izquierda de la sección de análisis hay un panel que permite seleccionar las opciones de análisis. En el caso de que haya algún componente de carga, el sistema detectará automáticamente los nodos desde los cuales se realizará el análisis. Si no hay ningún componente marcado como carga, los selectores de nodos se activarán y permitirá seleccionar los nodos de análisis.

Una vez seleccionados, se podrá pulsar uno de los botones inferiores:

- **Thevenin**: realiza, si es posible, el análisis de Thevenin del circuito diseñado en el área de diseño y muestra su equivalente en el panel central.
- **Norton**: realiza, si es posible, el análisis de Norton del circuito diseñado en el área de diseño y muestra su equivalente en el panel central.

Una vez pulsado uno de estos botones, el sistema intenta realizar los cálculos necesarios para obtener el circuito equivalente y si todo es correcto mostrará el circuito equivalente resultante. Si hubiera algún error mostrará una alerta y sugerirá al usuario la modificación del diseño.

En la parte inferior de este panel hay una casilla que permite mostrar u ocultar la imagen del circuito original visible en una miniatura en la parte superior izquierda del panel central.

I.4.2. Panel de Análisis

En el panel central se podrá visualizar el circuito equivalente una vez realizados los cálculos. Mostrará una fuente de tensión en serie con una resistencia en el caso de Thévenin o una fuente de corriente en paralelo con

una resistencia en el caso de Norton.

Si en el diseño había una única resistencia seleccionada como carga, se mostrará conectada al circuito con etiqueta roja. Si había varios componentes de carga o se han elegido los nodos manualmente (no había componentes de carga) se mostrarán estos dos componentes (fuente y resistencia) con unas líneas discontinuas que muestran dónde iría conectado el resto del circuito.

En la parte superior izquierda hay una miniatura del circuito original para que se pueda comparar con el circuito equivalente. Esta miniatura se puede mover, ampliar arrastrando desde una esquina o visualizar en modo zoom haciendo click sobre ella.

I.4.3. Panel de Información

A la derecha de la sección de análisis hay un panel de información desplegable que muestra los datos del análisis realizado.

- **Valores**: muestra los valores obtenidos en el análisis. En caso de Thévenin, muestra el valor de la fuente de tensión y el valor de la resistencia equivalente. En caso de Norton, muestra el valor de la fuente de corriente y de la resistencia equivalente.
- **Componentes de carga**: muestra los componentes de carga del circuito original.
- **Resumen**: muestra un breve resumen del análisis realizado.

I.5. E-Learning

Esta sección es la plataforma didáctica de ThevenApp donde se puede acceder con el rol de profesor o de estudiante con un registro previo.

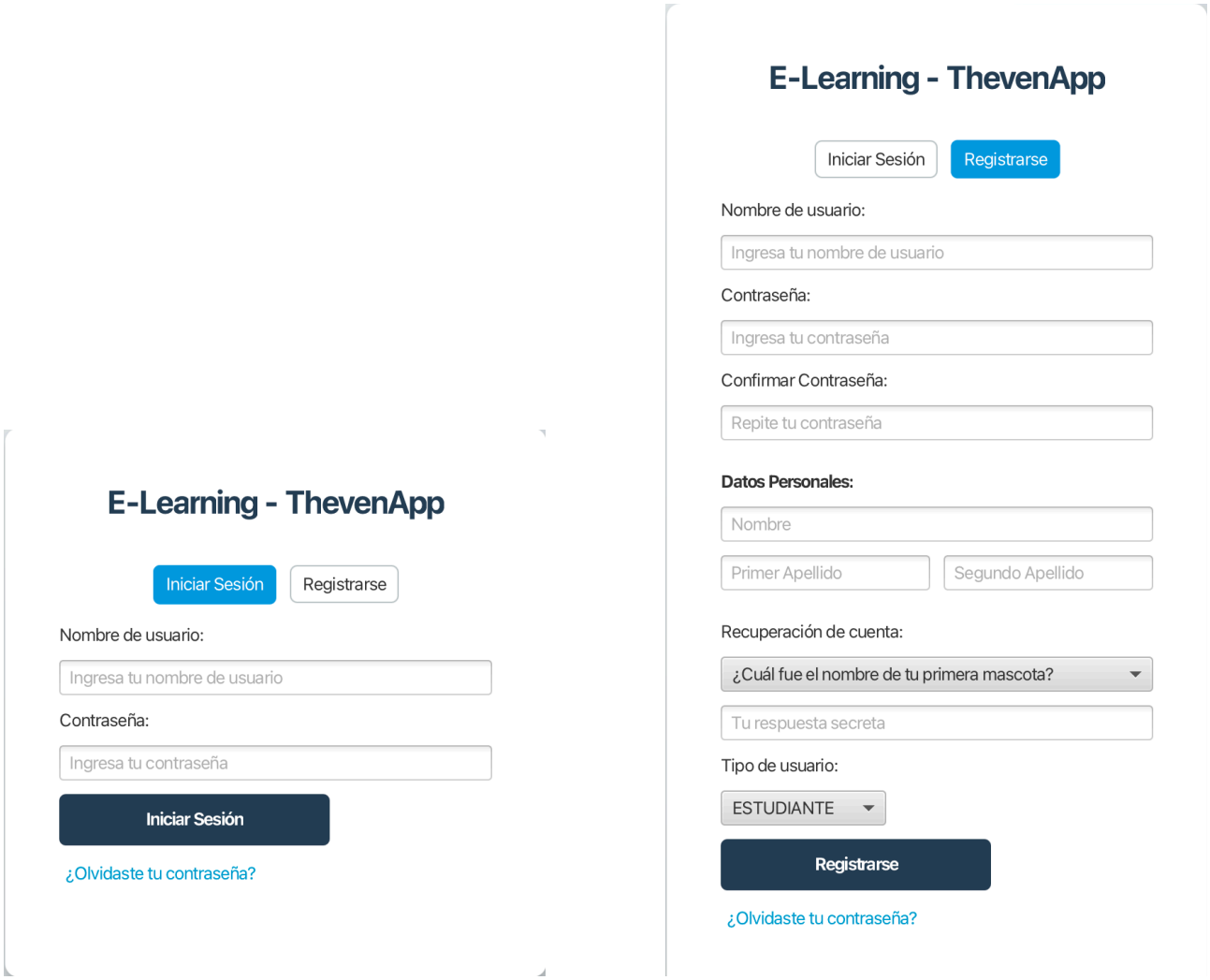
I.5.1. Login y Registro

Para entrar en esta sección se debe autenticar al usuario o registrarlo si no lo estuviera (ver Figura I.6).

- **Login** (Figura I.6a): para auntenticar en el sistema basta con escribir el nombre de usuario y la contraseña. En caso de olvidar la contraseña hay un enlace de recuperación que permitirá, previa comprobación, crear una contraseña nueva.
- **Registro** (Figura I.6b): para registrarse en el sistema tendrá que escribir un nombre de usuario, contraseña, nombre y apellidos, seleccionar una pregunta y respuesta de seguridad para recuperación de la cuenta y el tipo de usuario (estudiante o profesor).

I.5.2. Profesor

Como profesor puede acceder a algunas opciones de gestión que el estudiante no podrá realizar. Al acceder como profesor nos encontramos una ventana como se muestra en la Figura I.7. En esta pantalla se puede ver un menú de navegación por las diferentes opciones a la izquierda y un botón con letras rojas en la esquina superior derecha para *Cerrar Sesión*.



(a) Formulario de login. (b) Formulario de registro.

Figura I.6: Login y registro de la plataforma.

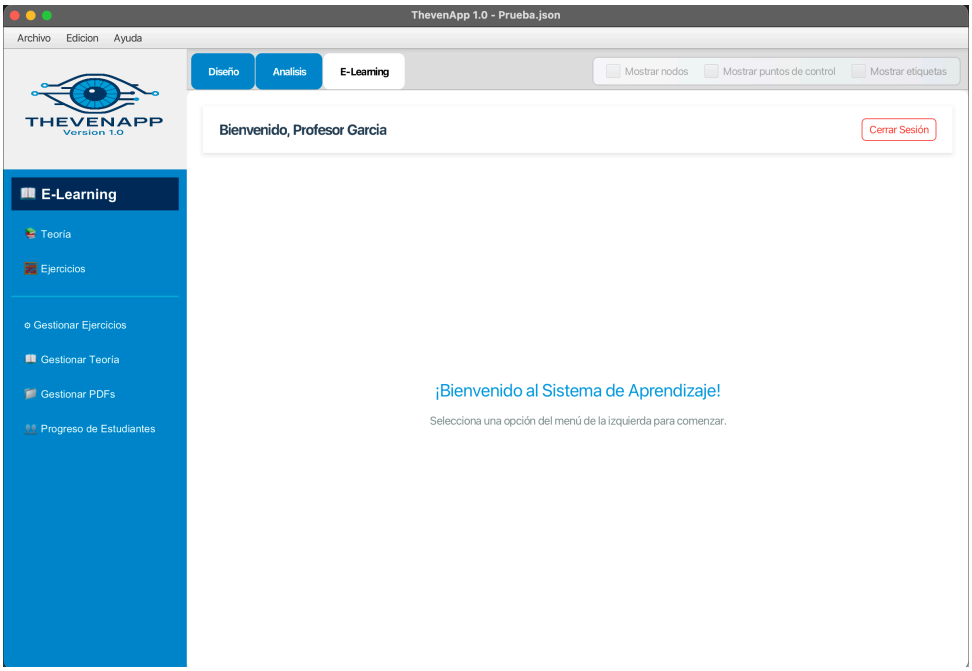


Figura I.7: Pantalla de bienvenida de profesores.

I.5.2.1. Teoría

Desde esta sección el profesor puede visualizar y comprobar los temas teóricos actualizados (Figura I.8).

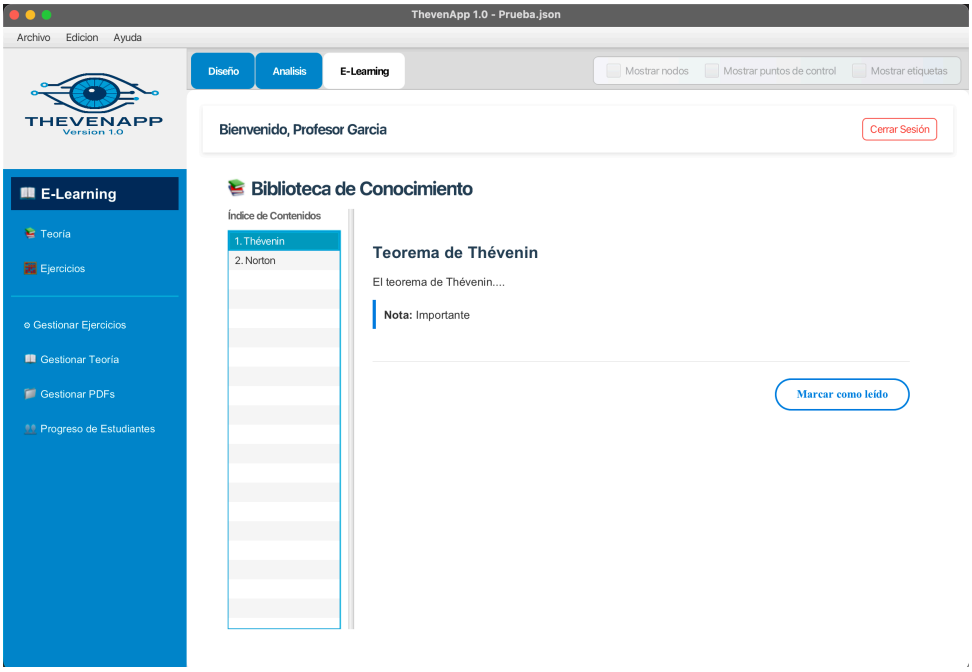


Figura I.8: Pantalla de visualización de teoría de profesores.

I.5.2.2. Ejercicios

Desde esta sección el profesor puede visualizar y comprobar los ejercicios propuestos (Figura I.9).

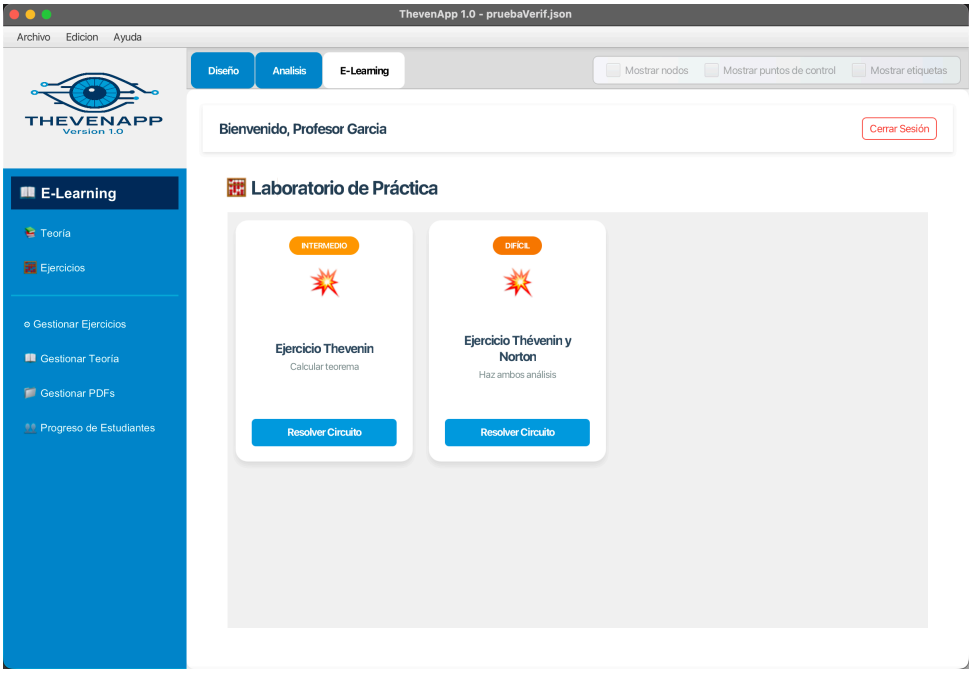


Figura I.9: Pantalla de visualización de ejercicios de profesores.

I.5.2.3. Gestionar Ejercicios

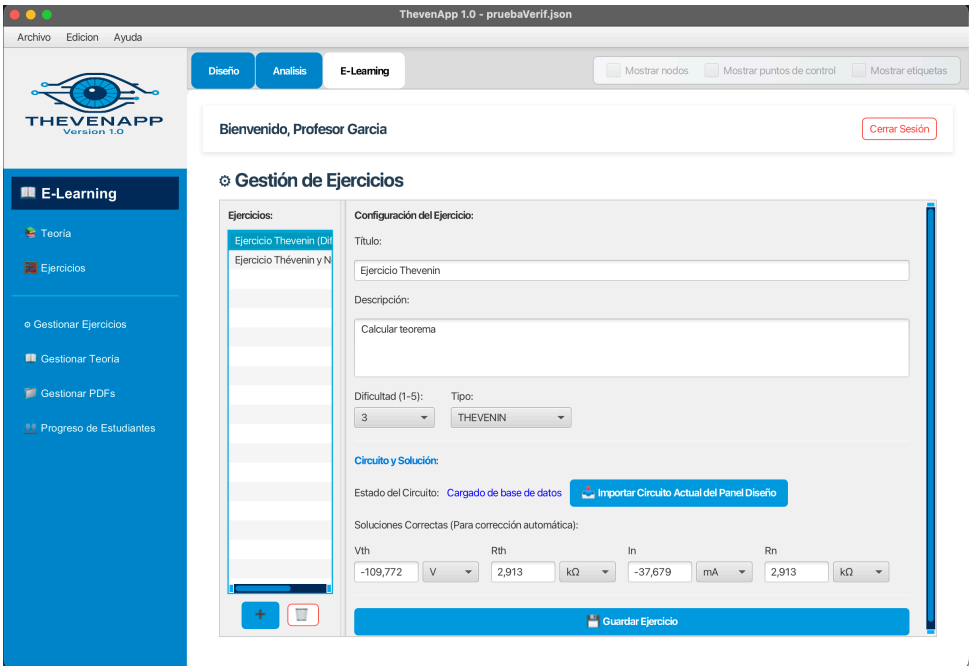


Figura I.10: Pantalla de gestión de ejercicios de profesores.

Desde esta sección el profesor puede crear nuevos ejercicios para proponer su resolución a los alumnos. Tal como se puede observar en la Figura I.10, a la izquierda aparece una lista con los ejercicios registrados y debajo dos botones (azul y rojo) desde donde se podrá crear un nuevo ejercicio o eliminar uno existente de la lista.

Para crear un ejercicio nuevo, el profesor puede ir al panel de diseño (ver Sección I.3) y dibujar un esquema de circuito eléctrico. Una vez creado, basta con rellenar el formulario:

- Escribir un título para el ejercicio.
- Escribir una descripción para que el alumno sepa, exactamente, qué debe hacer.
- Seleccionar dificultad (de 1 a 5).
- Seleccionar tipo de análisis que debe realizar el alumno: Thévenin, Norton o Ambos.
- Pulsar en el botón *Importar Circuito Actual del Panel de Diseño* y, automáticamente, se importará el circuito y se realizará el análisis de forma automática rellenando las casillas inferiores con los valores analizados que serán solución del ejercicio.
- Estos valores pueden ser modificados aunque no es recomendable hacerlo ya que el sistema lo hace automáticamente.
- Pulsar en *Guardar Ejercicio* y el ejercicio quedará registrado.

I.5.2.4. Gestionar Teoría

Desde esta sección el profesor puede crear nueva teoría y ponerla a disposición de los alumnos. Tal como se puede observar en la Figura I.11 a la izquierda aparece una lista con los temas teóricos registrados y debajo dos botones (azul y rojo) desde donde se podrá crear un nuevo tema o eliminar uno existente de la lista. Para crear un nuevo tema teórico se dispone de un editor en la parte derecha de la pantalla donde se podrá poner título, un número de orden y escribir siguiendo la plantilla preestablecida o creándolo desde cero. Este editor funciona igual que cualquier editor de texto conocido.

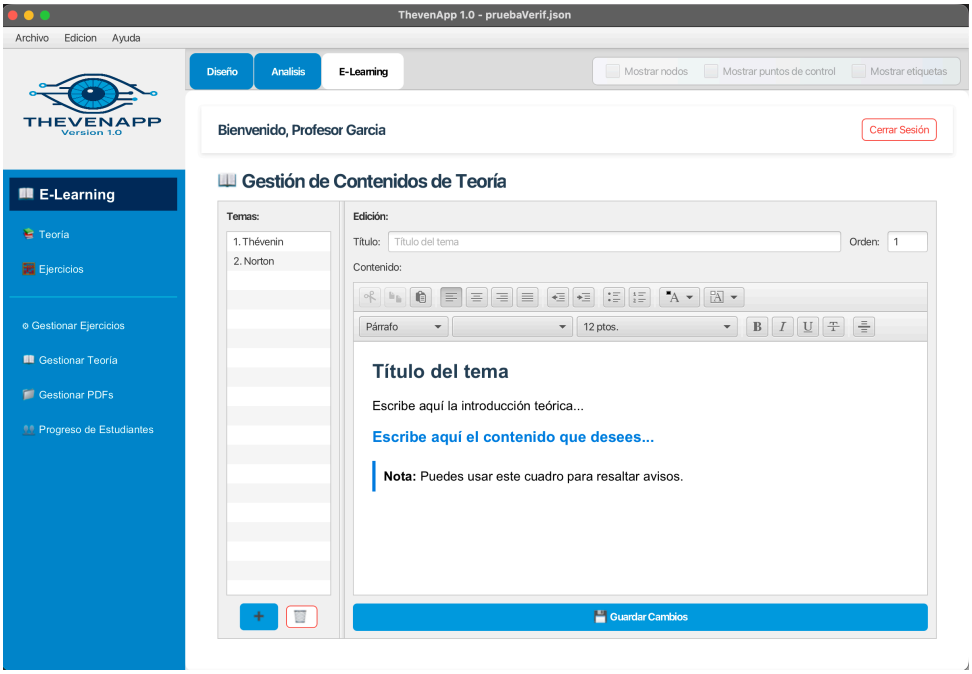


Figura I.11: Pantalla de gestión de teoría de profesores.

I.5.2.5. Gestionar Documentos PDF

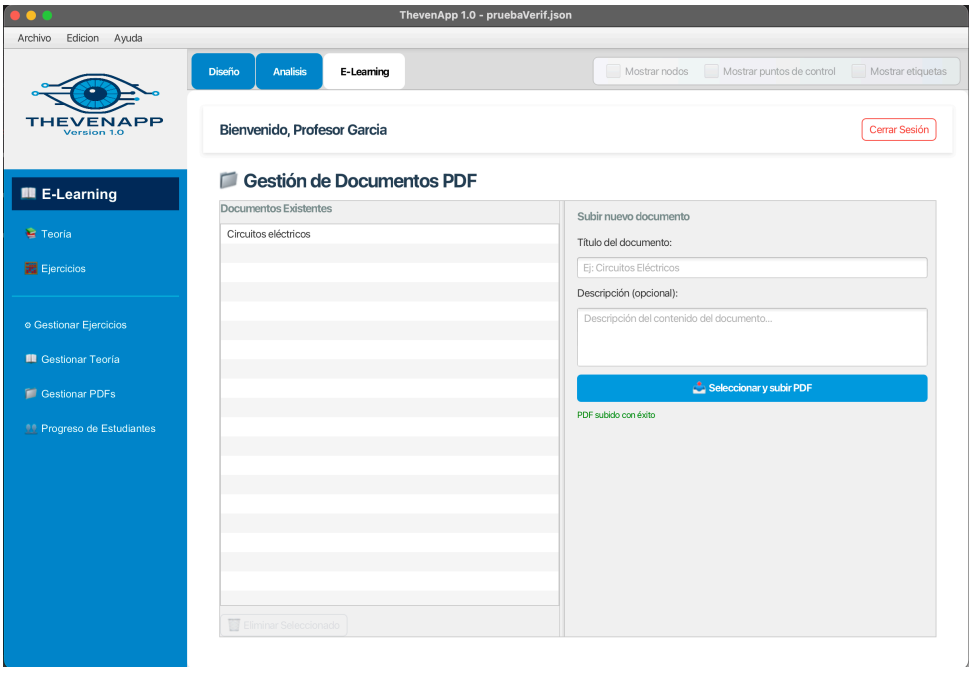


Figura I.12: Pantalla de gestión de documentos PDF para profesores.

Desde esta sección se puede comprobar la lista de documentos PDF existentes en la parte izquierda de la pantalla o cargar un nuevo documento desde la parte derecha poniendo un título y una descripción para que estén disponibles para los alumnos (ver Figura I.12).

I.5.2.6. Gestionar Progreso Estudiantes

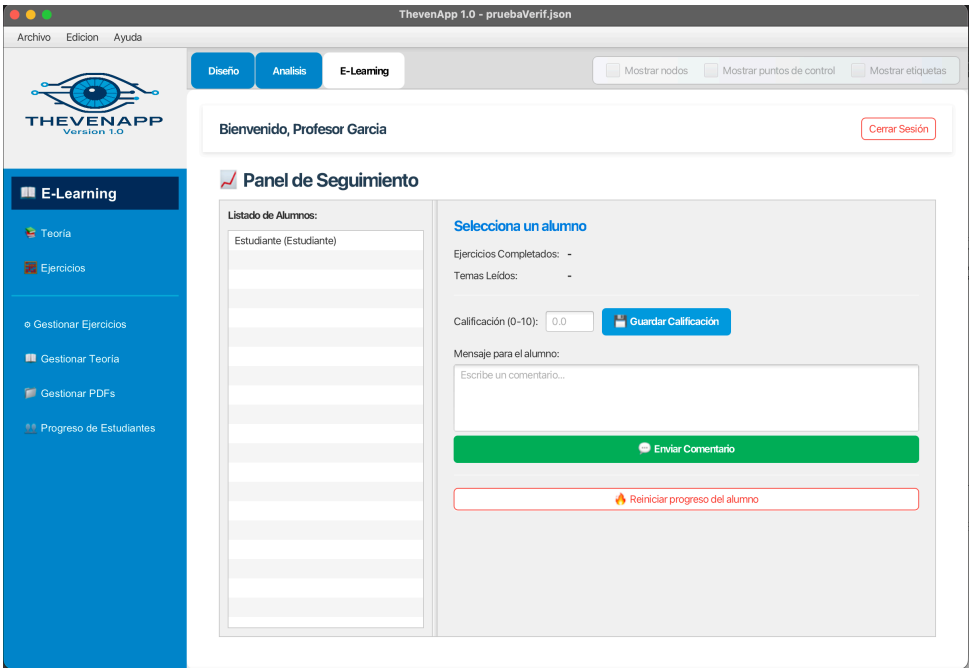


Figura I.13: Pantalla de progreso de estudiantes para profesores.

Desde esta sección se puede comprobar la lista de alumnos registrados en el sistema en la parte izquierda de la pantalla (ver Figura I.13). Al seleccionar uno de los alumnos de la lista un profesor puede:

- Visualizar el número de ejercicios completados por el alumno.
- Visualizar el número de temas leídos por el alumno.
- Calificar al alumno.
- Enviar comentarios que el alumno podrá visualizar.
- Reiniciar el progreso de un alumno.

I.5.3. Estudiante

Un estudiante dispone de una pantalla similar a la del profesor pero sin ninguna de las opciones de gestión y con un pequeño panel superior que contiene un resumen de su progreso: porcentaje completado con barra de progreso, número de ejercicios completados y calificación actualizada por el profesor (ver Figura I.14).

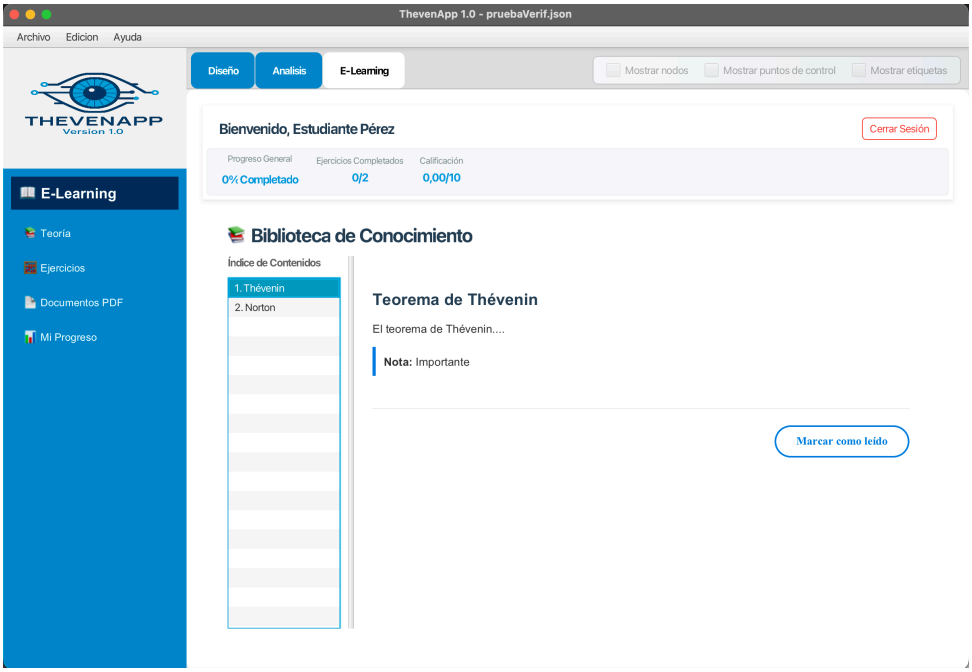


Figura I.14: Pantalla de *E-Learning* para estudiantes.

Un estudiante puede realizar las siguientes opciones:

- **Teoría**: Puede visualizar y estudiar todos los temas teóricos. Para que se contabilicen en el progreso deberá pulsar el botón de *"Marcar como leído"* situado al final del tema.
- **Ejercicios**: Puede visualizar los ejercicios propuestos en forma de tarjetas. Al pulsar en uno de ellos se puede ver los detalles del ejercicio y se podrá acceder a su resolución pulsando el botón correspondiente. Una vez pulsado, el estudiante será redirigido a la pestaña de Diseño (Figura I.4) donde podrá visualizar el circuito para que el alumno pueda resolverlo manualmente. El estudiante dispone de una ventana flotante como la que se muestra en la Figura I.15 donde podrá introducir los valores calculados y comprobar la resolución del ejercicio. Si se han introducido correctamente, el ejercicio se contabiliza y se podrá volver a la sección de *E-Learning* donde la tarjeta del ejercicio se habrá marcado en verde.
- **Documentos PDF**: Puede visualizar una lista de los documentos PDF subidos a la aplicación y visualizar cualquiera de ellos en un visor externo.
- **Mi progreso**: Puede visualizar los detalles de su progreso y la calificación y comentarios publicados por el profesor.

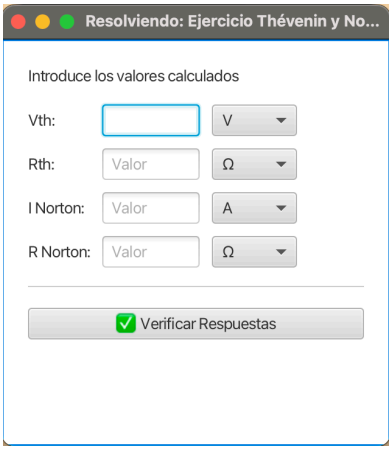


Figura I.15: Ventana de resolución de ejercicios.

I.5.4. Visor de logs

El visor de logs (Figura I.16) es una funcionalidad orientada a depuración o para desarrolladores. Se puede acceder a ella desde el menú superior *Ayuda* de la ventana principal (Figura I.2).

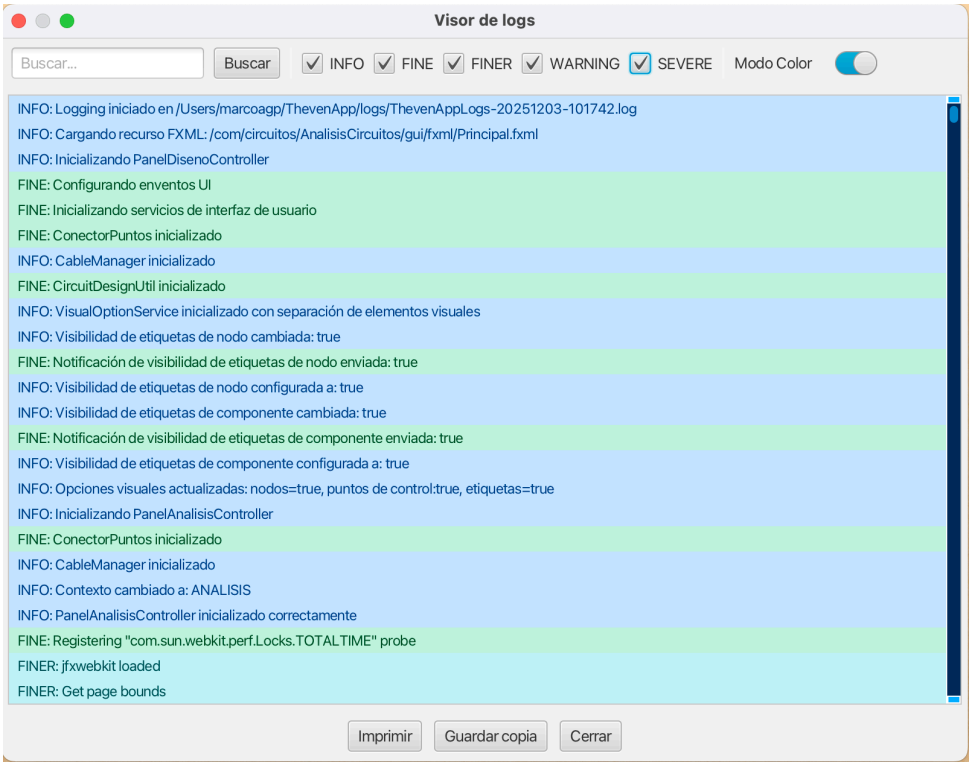


Figura I.16: Visor de logs.

Desde esta ventana podrá visualizar todos los mensajes de logs generados en la ejecución de *ThevenApp* dispuestos por colores:

- *Azul*: Nivel de log INFO.
- *Verde*: Nivel del log FINE.
- *Verde azulado*: Nivel de log FINER.
- *Amarillo*: Nivel de log WARNING.
- *Rojo*: Nivel de log SEVERE.

Desde la parte superior se puede filtrar por texto o por tipo de log; también se puede activar o desactivar los colores de los mensajes. Desde la parte inferior se puede imprimir la lista de logs a través de una impresora conectada, guardar una copia en un archivo o cerrar la ventana.

Anexos II

Manual Técnico de Instalación y Despliegue

El presente documento es un manual técnico de la aplicación **ThevenApp** que contiene las instrucciones para la ejecución de la aplicación, los requisitos técnicos y la configuración del entorno de desarrollo para compilar el código fuente y ejecutar la aplicación final.

II.1. Requisitos del Sistema

Para poder ejecutar **ThevenApp** o configurar el entorno de desarrollo deberá cumplir una serie de requisitos técnicos.

- **Hardware**:
  - **Arquitectura**: 64 bits (Necesario para JavaFX).
  - **Procesador**: Intel/AMD (x64) o Apple Silicon (ARM64).
  - **RAM**: 4 GB (mínimo), 8 GB (recomendado).
  - **Gráficos**: Tarjeta gráfica compatible con OpenGL (para renderizado de JavaFX).
- **Software**:
  - **Sistema Operativo**: Windows 10/11, MacOS 10.15 (Catalina) o superior, Linux (distribuciones actualizadas a GTK3).
  - **Java Development Kit, JDK**: versión 21 LTS o superior.

Requisitos de software para **entorno de desarrollo**:

- **IDE Eclipse** versión 4.38 o superior (recomendado).
- **Apache Maven** versión 3.8.x o superior.

II.2. Instalación del Entorno de Desarrollo

La configuración del entorno de desarrollo requiere realizar una serie de acciones secuenciales para asegurar la correcta compilación del proyecto:

1. **Instalación del JDK**: Instalar (si no lo tuviera) el JDK versión 21 LTS o superior.[1]

[1] Enlace de descarga oficial: https://www.oracle.com/es/java/technologies/downloads/.

2. **Configuración de variables de entorno**: Es necesario configurar la variable `JAVA_HOME` para que Maven y el sistema reconozcan la versión de Java activa. Siga los pasos correspondientes a su sistema operativo:

   Windows

   a) Buscar "Variables de entorno" en la configuración del sistema.
   b) Seleccionar "Variables de entorno" → "Nueva" (en la sección Variables del sistema).
   c) En **Nombre de la variable** escribir: `JAVA_HOME`.
   d) En **Valor de la variable** escribir la ruta de instalación. Ejemplo:

```
C:\Program Files\Java\jdk-21.0.1
```

   e) Buscar la variable `Path`, hacer clic en "Editar" y añadir al final:

```
%JAVA_HOME%\bin
```

   macOS

   a) Abrir una terminal.
   b) Ejecutar el siguiente comando para añadir la variable al perfil de la shell (zsh):

```
echo 'export JAVA_HOME=$(/usr/libexec/java_home)'
   >> ~/.zshrc
```

   c) Recargar la configuración para aplicar los cambios:

```
source ~/.zshrc
```

   d) Verificar que la ruta aparece correctamente escribiendo: `echo $JAVA_HOME`

   Linux

   a) Buscar la ruta de instalación del JDK mediante el comando:

```
sudo update-alternatives --config java
```

   b) Copiar la ruta mostrada (ej. `/usr/lib/jvm/java-21...`) excluyendo la parte final `/bin/java`.
   c) Editar el archivo de configuración (ej. `.bashrc`) con `nano ~/.bashrc` y añadir al final:

```
export JAVA_HOME="/usr/lib/jvm/java-21-openjdk-
   amd64"
export PATH=$JAVA_HOME/bin:$PATH
```

   d) Guardar, salir y recargar la terminal.

3. **Instalación del IDE**: Instalar **Eclipse IDE** (versión 2023-09 o superior recomendada).[2]
4. **Verificación de Maven**: El proyecto utiliza Apache Maven v.3.8.x. *Nota: Las versiones recientes de Eclipse ya incluyen la integración con Maven (plugin M2E) preinstalada, por lo que no suele ser necesaria ninguna acción adicional salvo importar el proyecto.*

[2]Enlace de descarga: https://eclipseide.org.

Una vez realizada la configuración del entorno, se debe descomprimir el archivo ZIP entregado en el que habrá dos carpetas una nombrada como "Proyecto" y otra como "Ejecutables". En la primera se encuentra todo el código fuente del proyecto. Ahora hay que realizar los siguientes pasos:

1. **Importación en Eclipse**: ir a "File"→"Import"→"Existing Maven Projects" y seleccionar la carpeta donde se encuentra el archivo `pom.xml` dentro de /Proyecto/analisisCircuitos/.
2. **Descarga de Dependencias**: Al importar el proyecto, Maven descarga todas las librerías definidas en el archivo `pom.xml` automáticamente.
3. **Compilación y obtención del JAR ejecutable**: Compilar y obtener un .jar ejecutable se hace a través de la herramienta Maven Shade Plugin (ya configurada en el archivo pom). Solo basta con hacer estos pasos:
   a) Hacer click derecho encima del proyecto en el explorador e ir a la opción "Run as"→"Maven Build".
   b) Escribir en "goals" el comando: `clean package`.
   c) Después de la exportación, en la carpeta `/target` dentro de la ruta del proyecto estará el archivo ejecutable .jar.

II.3. Ejecución

En el archivo zip entregado, dentro de la carpeta "Ejecutables" se encuentran dos archivos en formato .jar ejecutables en el ordenador que cumple los requisitos establecidos en la sección II.1.

Debido a las librerías nativas de JavaFX, se proporcionan dos versiones del archivo ejecutable con el objetivo de optimizar la compatibilidad. Los ordenadores Mac actuales utilizan dos arquitecturas de procesador diferentes: Intel (modelos anteriores a 2020) y Apple Silicon (modelos M1 y posteriores). Desgraciadamente, configurar el archivo `pom.xml` del proyecto para que descargue la versión correspondiente genera un conflicto de librerías para las diferentes arquitecturas Mac. Por esta razón se ha decidido proporcionar dos versiones de la siguiente manera:

ThevenApp-Standard.jar
Ejecutable para plataformas Windows (x64), Linux (x64) y MacOS (Intel).

```
java -jar ThevenApp-Standard.jar
```

ThevenApp-Silicon.jar
Ejecutable para plataformas Windows (x64), Linux (x64) y MacOS (AArch64 - M1, M2, M3, M4).

```
java -jar ThevenApp-Silicon.jar
```

En un ordenador con Windows o Linux se podrá ejecutar cualquiera de las dos versiones (no influye). En un ordenador Mac deberá elegir la opción correcta según su arquitectura. En el caso de ejecutar la versión incorrecta provocará el error de "librerías gráficas no encontradas".

II.4. Estructura de almacenamiento de datos

Cuando se ejecuta *ThevenApp*, la propia aplicación genera un directorio donde se almacenarán los datos generados. Este directorio se encuentra en la carpeta raíz de su sistema operativo (user.home) con el nombre de "/ThevenApp/".

En esta carpeta se almacena lo siguiente:

- **Archivo de base de datos**: se encuentra el archivo `thevenapp-database.db` que es el archivo de base de datos SQLite que se crea automáticamente si no existe.
- **Carpeta de logs**: hay una carpeta "/logs/" donde se encuentran los archivos de log de cada sesión.
- **Carpeta de PDFs**: hay una carpeta "/pdfs/" donde se almacenan los documentos PDF subidos a la plataforma de *e-learning*.

Si se desea realizar un **renicio de fábrica** basta con eliminar la carpeta "/ThevenApp" y se resetea toda la aplicación.

II.5. Solución de problemas

No se han encontrado problemas al realizar pruebas de la aplicación en diferentes arquitecturas o sistemas operativos excepto los habituales por no haber realizado la configuración del equipo correctamente. Algunos errores que se pueden encontrar son:

- **Error en versión de Java**: suele ser un error de tipo "Unsupported class file...". La solución está en instalar JDK 21 o superior.
- **Error de permisos**: puede dar un error de permisos al intentar crear la base de datos en el escritorio. Para solucionarlo se deberá revisar los permisos de escritura en la carpeta de usuario.
- **Error de JavaFX/Gráficos corruptos**: puede ocurrir por usar la versión incorrecta del ejecutable .jar. La solución está en revisar la arquitectura del ordenador, sobre todo, si se usa Mac (Intel o Silicon) y ejecutar el archivo correcto.

II.6. Información técnica (extraída del archivo POM)

- **Nombre de la aplicación**: ThevenApp.
- **Versión**: 1.0.
- **GroupId**: com.circuitos.
- **ArtifactId**: analisiscircuitos.
- **Versión de Java**: 21.
- **Dependencias instaladas**:
  - `javafx.controls`: v.23.0.1.
  - `javafx.fxml`: v.23.0.1.
  - `javafx.base`: v.23.0.1.
  - `javafx.web`: v.23.0.1.
  - `javafx.graphics`: v.23.0.1.
  - `jackson.databind`: v.2.18.2.
  - `sqlite-jdbc`: v.3.46.0.0.
  - `controlsfx`: v.11.2.2.
  - `commons-math3`: v.3.6.1.
  - `junit-jupiter`: v.5.11.0 (Testing).

Anexos III

Código Fuente de la Aplicación

III.1. Estructura del proyecto

En la figura III.1 se muestra una estructura resumida del proyecto para facilitar la navegación por la carpeta que contiene el código fuente.

```
analisisCircuitos
├── pom.xml
└── src/main
    ├── resources
    │   ├── fxml/                                   (Vistas de la interfaz)
    │   │   ├── Principal.fxml
    │   │   ├── PanelDiseno.fxml
    │   │   └── ... (12 archivos más)
    │   ├── images/                                 (Iconos y recursos gráficos)
    │   └── styles/            (Hojas de estilo CSS)
    │       ├── componentes.css
    │       └── ... (15 estilos más)
    └── java/com/circuitos/analisiscircuitos
        ├── ThevenApp.java                          (Punto de entrada main)
        ├── analisis/                               (Lógica de Thévenin/Norton)
        │   ├── Analizador.java
        │   └── ResultadoThevenin.java
        ├── dominio/                                (Modelo de Objetos)
        │   ├── Circuito.java
        │   ├── Componente.java
        │   ├── Resistencia.java
        │   ├── Fuentes/                            (Independientes y Dependientes)
        │   └── util/                               (Motor MNA)
        │       ├── MatrixUtil.java
        │       ├── StampUtil.java
        │       └── CircuitUtil.java
        ├── gui/                                    (Interfaz de Usuario - MVC)
        │   ├── controller/                         (Controladores JavaFX)
        │   │   ├── ThevenAppController.java
        │   │   └── ... (Controladores de paneles)
        │   ├── commands/                           (Patrón Comando - Deshacer/Rehacer)
        │   ├── state/                              (Gestión de estado visual)
        │   └── renderer/                           (Dibujado de componentes)
        ├── service/                                (Lógica de Negocio)
        │   ├── analisis/
        │   ├── cable/                              (Gestión de conexiones y nudos)
        │   ├── io/                                 (Guardado y carga de archivos)
        │   └── learning/                           (Sistema de aprendizaje y BBDD)
        └── learning/                               (Gestión de usuarios y progreso)
            ├── database/                           (Conexión SQLite)
            └── manager/                            (Lógica de ejercicios)
```

Figura III.1: Estructura de árbol del proyecto.

III.2. Estadísticas

El presente proyecto se ha llevado a cabo en el año 2025 y se ha presentado como Proyecto Final del Grado en Ingeniería en Tecnologías de la Información de la UNED con las métricas que se muestran en la Tabla III.1.

Métrica	**Valor**
Total de Ficheros (.java)	133
Total de Líneas de Código (aprox.)	24.400
Paquetes Funcionales	25
Ficheros de Vista (.fxml)	12
Hojas de Estilo (.css)	15

Tabla III.1: Resumen estadístico del código fuente.

III.3. Versiones de Dependencias (Maven)

En esta sección se incluyen las versiones de las librerías externas utilizadas en la Tabla III.2.

Librería	**Versión**
Apache Commons Math3	3.6.1
OpenJFX	23.0.1
JUnit	5.11.0
Jackson	2.18.2
SQLite-jdbc	3.46.0.0

Tabla III.2: Versiones de las dependencias matemáticas y gráficas.

III.4. Estructura de archivos JSON

La aplicación utiliza archivos en formato JSON para almacenar circuitos. A continuación se muestra un ejemplo de la estructura interna de un archivo guardado por la aplicación.

Código III.1: Ejemplo de estructura de un circuito guardado en formato JSON.

```
{
  "metadata" : {
    "app" : "ThevenApp",
    "version" : "1.0",
    "creacion" : "04/12/2025 12:32:40 CET (+01:00)"
  },
  "circuito" : {
    "componentes" : [
      {
        "@type" : "FuenteTensionIndependiente",
        "nodo1" : 0,
        "nodo2" : 1,
        "id" : "V1",
        "carga" : false,
        "valor" : 12.0
      }
      // ... otros componentes
    ],
    "cables" : [
      {
        "id" : "Cable-1",
        "origenId" : "V1",
        "origenPos" : "ARRIBA",
        "origenPositivo" : true,
        "origenNodo" : 0,
        "destinoId" : "R1",
        "destinoPos" : "IZQUIERDA",
        "destinoNodo" : 1,
        "puntos" : [
            { "x" : 70.0, "y" : 256.0 },
            { "x" : 69.5, "y" : 190.0 },
            { "x" : 111.0, "y" : 190.0 }
        ]
      }
      //...otros cables
    ]
  }
}
```

III.5. Licencia de Uso