



UNIVERSIDAD DE MÁLAGA



Grado Ingeniería Informática

Física Interactiva: simulación de circuitos RC y RL en
corriente continua y estado transitorio

Interactive Physics: RC and RL circuits simulation in direct
current and transient state

Realizado por
David Gómez Pérez

Tutorizado por
José Manuel Peula García
Inmaculada Alados Arboledas

Departamento
Física Aplicada
UNIVERSIDAD DE MÁLAGA

MÁLAGA, (mes y año)

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA
Grado Ingeniería Informática

**Física Interactiva: Simulación de circuitos RC y RL en
corriente continua y estado transitorio**

**Interactive Physics: RC and RL circuits simulation in
direct current and transient state**

Realizado por
David Gómez Pérez

Tutorizado por
José Manuel Peula García
Inmaculada Alados Arboledas

Departamento
Física Aplicada II

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2017

Fecha defensa: 7 de julio de 2017

Abstract

Most of the students who takes physics lectures at college related with engineering or architecture, often find some complexity in trying to understand the topics they're studying. Math difficulty isn't the main problem; however, the interpretation of the physical phenomena that are being studied could be a real challenge for them.

To solve this problem a solution could be the production of a simulation software that allows us to observe the evolution of each physical magnitude, specifically of RC and RL circuits in direct current and transient state, with some theory explanation at its side.

Keywords: simulation, RC circuit, RL circuit, direct current, transient state

Resumen

La mayor parte de los alumnos que cursan alguna asignatura de física en ramas técnicas como en ingeniería o arquitectura, suelen encontrar cierta complejidad al intentar comprender los temas estudiados. Esto no se debe a la dificultad matemática de los problemas que se plantean, sino más bien en interpretar la evolución de los fenómenos físicos que se estudian.

Así que para tratar de solucionar este problema y facilitar su estudio, se plantea el desarrollo de una aplicación web que sea capaz de mostrar la evolución de diferentes magnitudes físicas de los circuitos RC y RL en corriente continua y estado transitorio acompañada de una explicación teórica sobre las mismas.

Palabras clave: simulación, circuito RC, circuito RL, corriente continua, estado transitorio

Índice

1. Introducción	9
1.1. Motivación	9
1.2. Objetivos y metodología	10
1.3. Estructura del documento	13
1.4. Tecnologías usadas	14
2. Circuitos RC y RL	17
2.1. El condensador y el circuito RL	19
2.2. El condensador y el circuito RC	21
2.3. El inductor y el circuito RL	24
3. Estudio de las tecnologías	33
3.1. Tipos de aplicaciones según la compatibilidad	33
3.2. La web y su evolución	35
3.3. Estructura de una aplicación web	38
3.4. Tecnologías <i>front-end</i>	41
3.4.1. HTML	43
3.4.2. CSS y Bootstrap	45
3.4.3. JavaScript	47
3.4.4. ReactJS	48
4. Implementación y pruebas	55
4.1. Simulación de los circuitos RC y RL	55
4.1.1. Etapas del proceso	55
4.1.2. Generación de datos	56
4.1.3. Tiempo absoluto y tiempo relativo de la simulación	59
4.1.4. Condiciones de parada y escala de tiempo	60
4.2. Comparación de resultados	61
4.2.1. Caso de prueba 1: carga y descarga de un condensador	62

4.2.2.	Caso de prueba 2: carga y descarga de un condensador (con restricciones)	64
4.2.3.	Caso de prueba 3: carga y descarga de un inductor	66
4.2.4.	Caso de prueba 4: carga y descarga de un inductor (con restricciones)	68
5.	Conclusions and Futures Lines of Research	71
5.1.	Conclusions	71
5.2.	Future lines of Research	71
6.	Conclusiones y Líneas Futuras	73
6.1.	Conclusiones	73
6.2.	Líneas Futuras	73
Apéndice A.	Modelado de los circuitos RC y RL	77
Apéndice B.	Resolución matemática del circuito RC	79
B.1.	Estado de almacenamiento de energía	79
B.1.1.	Carga del condensador	79
B.1.2.	Intensidad de corriente	80
B.1.3.	DDP Resistencia	80
B.1.4.	DDP Condensador	81
B.2.	Estado de disipación de energía	81
B.2.1.	Carga del condensador	81
B.2.2.	Intensidad de corriente	82
B.2.3.	Diferencia de potencial en la resistencia	83
B.2.4.	Diferencia de potencial en el condensador	83
B.3.	Energía almacenada en un condensador	83
Apéndice C.	Resolución matemática del circuito RL	85
C.1.	Estado de almacenamiento de energía	85
C.1.1.	Intensidad de corriente	85
C.1.2.	Diferencia de potencial en la resistencia	86
C.1.3.	Diferencia de potencial en la bobina	86
C.2.	Estado de disipación de energía	87

C.2.1.	Intensidad de corriente	87
C.2.2.	Diferencia de potencial en la resistencia	87
C.2.3.	Diferencia de potencial en la bobina	88
C.3.	Energía almacenada	88
C.4.	Flujo magnético	89
Apéndice D. Requisitos y casos de uso		91
D.1.	Requisitos	91
D.2.	Casos de uso	93
Apéndice E. Manual de instalación y de usuario		95
E.1.	Manual de instalación del proyecto	95
E.2.	Manual de usuario de la aplicación	96
E.3.	Manual de usuario del script <i>python</i> utilizado para las pruebas	99

1

Introducción

1.1. Motivación

La física, es una asignatura impartida en los grados de ingeniería, arquitectura o en matemáticas, que suele tener una gran complejidad de comprensión por parte del alumnado que las cursa. La principal causa no se debe a la gran dificultad que presentan las matemáticas usadas en la resolución analítica de los problemas, sino más bien en el entendimiento de los conceptos teóricos que hay tras los fenómenos físicos estudiados, y por consiguiente, en su aplicación a los problemas concretos.

Por otro lado, tenemos la informática, una ciencia bastante joven en comparación con la física que, de alguna manera u otra, ha hecho uso de sus leyes fundamentales desde su aparición. Se puede llegar a pensar que son ciencias sin relación alguna pero, desde el hardware del computador más rudimentario como el Z1 [1] construido por Konrad Zuse en 1938 hasta el procesador cuántico más sofisticado desarrollado por IBM [2], son un claro ejemplo de la simbiosis existente entre ellas. Sin embargo, esta relación no es unilateral, pues tal y como se demostró en 1948, fue cuando se realizaron una de las primeras simulaciones por computador [3] en el conocido *Proyecto Manhattan* para recrear el impacto de un artefacto nuclear; o más reciente, la realizada por la NASA [4] en el año 2019 de un agujero negro meses después de tomar la primera fotografía de uno. Pero, ¿qué es una simulación?

Por definición (según la RAE), **simulación** es la "representación de algo, fingiendo o imitando lo que no es". Llevando esta explicación al terreno de la física, se trata de construir abstractamente el comportamiento de una situación (como las enumeradas anteriormente) para observar cómo evoluciona el entorno (o algunas propiedades específicas de este) dependiendo

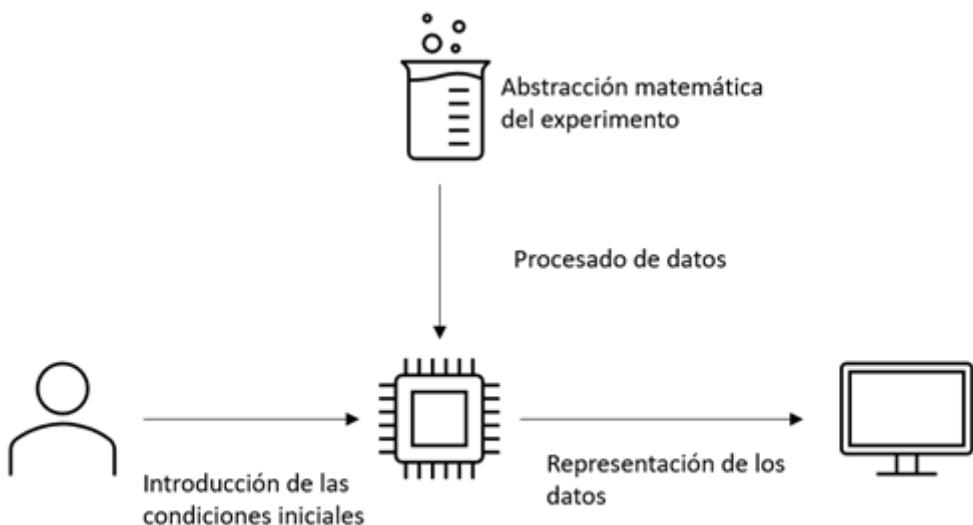


Figura 1: Representación abstracta de simulación.

de unas condiciones previas, todo ello haciendo uso de las ciencias de la computación. Así que para tratar el problema presentado al inicio de este apartado, podemos seguir el modelo tomado por la *Universidad de Colorado*, pues cómo se puede observar en su web oficial [5], desde 2002 se están construyendo simulaciones como material complementario para el estudio de física, matemáticas, química y otras áreas de la ciencia.

Dada esta motivación, la línea de este trabajo consiste en la elaboración de las simulaciones físicas correspondientes a los *circuitos RC y RL en corriente continua y estado transitorio*, estudiados en la asignatura *Fundamentos Físicos de la Informática* de los grados de *Ingeniería Informática, Software y Computadores* de la *Universidad de Málaga*.

1.2. Objetivos y metodología

Como objetivo principal, tal y como se plantea al final del apartado anterior, es ofrecer una solución informática que simplifique el estudio de los alumnos que estudien alguna asignatura de física, en la interpretación de los circuitos RC y RL en corriente continua y estado transitorio. Como toda simulación, tendrá un *engine* (motor) que se encargue de generar los resultados, los cuales vendrán dados dependiendo de las entradas proporcionadas por el usu-

rio del software, acompañados de animaciones cuya principal función será la de imitar a los circuitos reales. Este será nuestro **objetivo principal**, así que para alcanzarlo, deberemos de cumplir los siguientes **objetivos específicos**.

Como primera meta, tendremos que realizar un **análisis matemático** de ambos circuitos, para así obtener un modelo matemático que nos permita expresar analíticamente la evolución de las magnitudes físicas que caracterizan estos sistemas, como la intensidad de corriente de los circuitos, el voltaje de los dispositivos o la energía eléctrica o magnética intercambiada por éstos. Una vez obtenidas estas expresiones, dejaremos a un lado la parte física del proyecto y avanzaremos con la **evaluación de las soluciones y tecnologías** que nos permitirán desarrollar cada una de las simulaciones y, una vez barajadas todas las posibilidades, el siguiente objetivo será **aprender la tecnología**.

Finalmente, nos quedará la **implementación** haciendo uso de las ecuaciones resultantes del análisis y de la tecnología seleccionada. A este objetivo, le sumaremos una serie de **pruebas** manuales para comprobar que los resultados obtenidos son los adecuados.

Cada uno de estos objetivos planteados, se podría corresponder en cierta manera a cada una de las fases o etapas por la que irá pasando el proyecto hasta finalizarlo. Esta *metodología*, conocida como *incremental* [6], pertenece a los modelos abstractos tradicionales, exactamente a la categoría de los *evolutivos*, entre los que también encontramos otros modelos como *espiral*, *iterativo* o *diseño por planificación*. Todas estas presentan la característica de estar principalmente dividida por cuatro etapas:

- **Análisis.** En esta primera fase se llevará a cabo un análisis global del sistema. Es muy importante establecer un diálogo en el que el equipo técnico (desarrollo) y el cliente de la aplicación, pues aquí se tomarán todas las ideas básicas del software, así como la resolución de las dudas que los desarrolladores puedan tener sobre el objetivo esperado del cliente. Además, el planteamiento del proyecto debe de quedar cerrado en esta etapa, salvo en la metodología en espiral (en este caso, el ciclo de desarrollo como su nombre indica es una espiral, por lo que esta etapa sucederá varias veces).

- **Diseño.** Aquí, es donde se realiza un primer modelo del sistema haciendo uso de los requisitos tomados en la anterior etapa.
- **Implementación.** Como su propio nombre indica, esta etapa consiste en la transcripción a código de las ideas iniciales y obtener como resultado un producto visual de ellas.
- **Pruebas.** Ya completada la implementación, se necesita comprobar que todo funciona correctamente, y para ello se realiza una serie de pruebas generales al software para asegurar que las ideas inicialmente planteadas están perfectamente operativas y no suponen ningún riesgo.
- **Despliegue y mantenimiento.** Una vez el *software* se encuentra terminado y las pruebas realizadas dan como resultado lo que esperábamos, podemos concluir que el producto está listo para su distribución. Sin embargo, al repartirlo entre los usuarios es probable que aparezcan errores que no se tuvieron en cuenta en el momento del diseño y elaboración de las pruebas. Una vez el sistema se encuentre desplegado y sea accesible al público, es necesario llevar un mantenimiento tanto de la aplicación como del *hardware* del servidor dónde se encuentra alojada. Como veremos más adelante utilizaremos un servicio de terceros para realizar el despliegue, así que el mantenimiento *hardware* no depende de nosotros.

Normalmente, se suele añadir una quinta etapa llamada mantenimiento. Esta fase final, consiste en una vez desplegado el sistema en una infraestructura, revisar que todo funcione correctamente y, en caso de cualquier imprevisto, encontrar una solución lo antes posible.

Para la elaboración de este TFG, se va a emplear una metodología incremental; y aunque no va a tener exactamente la estructura anterior, sí que va a compartir sus fundamentos. Las etapas a llevar a cabo son los siguientes (ver Figura 2):

1. Reunión con los tutores.
2. Investigación de los fenómenos físicos propuestos.
3. Análisis matemático.
4. Evaluación de las soluciones y tecnologías.

5. Aprendizaje de las tecnologías

6. Implementación y pruebas.

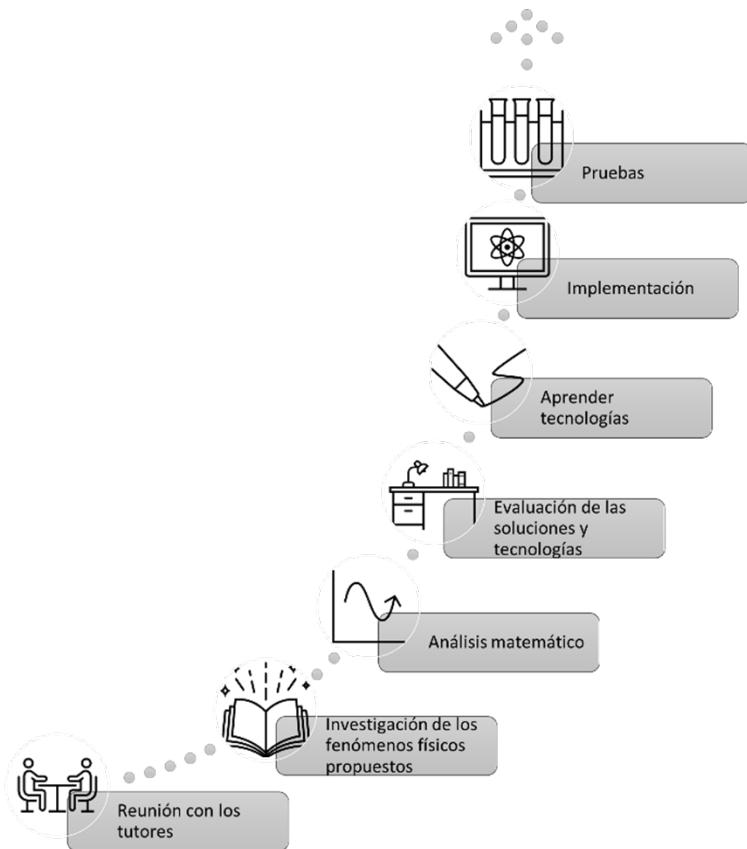


Figura 2: Diagrama de la metodología utilizada. Al estar en una metodología incremental, en cada una de las etapas se van cumpliendo los objetivos propuestos, así como la funcionalidad de la aplicación.

1.3. Estructura del documento

Por otro lado, siguiendo el orden del desarrollo de las simulaciones, la división de este documento se puede separar en los capítulos mostrados a continuación.

- **Capítulo 1. Introducción.** En esta primera parte, se expondrán las principales motivaciones que llevan a realizar este proyecto, cuáles son sus objetivos finales y específicos, la estructura de la memoria y una breve descripción de cada una de las tecnologías a utilizar durante el desarrollo.

- **Capítulo 2. Circuitos RC y RL.** En esta segunda sección de la memoria, se realizará un repaso a los fenómenos físicos estudiados, en el que se dará explicación a algunos de sus aspectos clave, los parámetros a estudiar y las ecuaciones que los modelan (todo el desarrollo matemático y físico correspondiente a cada uno de los circuitos se muestra en el apartado de los *Apéndices*).
- **Capítulo 3. Estudio de las tecnologías.** Una vez aclarados los conceptos clave, continuaremos con el estudio de las tecnologías que podrían utilizarse para ejecutar la implementación. Se consideraran varias opciones y será utilizada aquella más conveniente a nuestras necesidades.
- **Capítulo 4. Implementación y pruebas.** Ya seleccionadas y aprendidas las herramientas, seguiremos con la implementación. En este capítulo, se recogerá los principios de estas tecnologías y se aplicarán al caso.
- **Capítulo 5. Conclusión.** Finalmente, se expondrá una conclusión y cómo una breve idea de cómo este trabajo podría extenderse.

1.4. Tecnologías usadas

Para terminar con esta primera parte del documento, haremos un repaso de las herramientas que usaremos a lo largo del desarrollo del proyecto. La lista es la siguiente:

- **Visual Studio Code.** *Visual Studio Code* o *VSCCode* [7], es un editor de código desarrollado por *Microsoft*. Una clara diferencia respecto a otros editores, es que éste nos permite la instalación de *plugins* que nos ayudan durante la tarea de programación, como por ejemplo, alerta de errores de sintaxis, la vista del directorio del proyecto actual en forma de árbol (facilidad para ver su estructura) o una terminal incorporada entre otros.
- **Overleaf.** Se trata de una herramienta online, la cuál nos ofrece un entorno de trabajo para elaborar nuestros documentos en *L^AT_EX*, además de un repositorio en el que almacenarlos.
- **Adobe Photoshop.** Es un software para edición de imágenes, cuya principal misión de esta herramienta informática será la elaboración de las animaciones que acompañarán

a la simulación.

- **GitHub** es un directorio en la nube que nos permite alojar el código fuente de nuestro proyecto. Además de tener la utilidad de repositorio, nos servirá para realizar el despliegue de la aplicación utilizando el servicio **GitHub Pages**.

Como podemos ver, el número de programas informáticos que vamos a utilizar es muy reducido, puesto que no se necesitan de más. A continuación, se da una lista con los lenguajes de programación y *frameworks* usados.

- **HTML.** *HTML* de sus siglas *HyperText Markup Language* [8], como su propio nombre señala, es un lenguaje basado en etiquetas encargado de definir la estructura básica del contenido de la web. Siempre va acompañado de otras tecnologías como **CSS** o **JavaScript**.
- **CSS o Cascade Style Sheets.** [9] Se trata de un lenguaje encargado en la presentación de la web. Su finalidad es mejorar la apariencia visual de los bloques escritos en HTML.
- **JavaScript.** Es un sencillo lenguaje de programación [10] utilizado en la mayoría de los sitios Web. Usando solamente HTML y CSS, los documentos generados son estáticos, es decir, su vista es siempre la misma. En cambio, *JavaScript* nos permite dinamizar estas páginas, siendo esto causa de que sea atractiva al usuario. Como veremos más adelante, este lenguaje no se aplica solamente en el **front**, sino también en el **back**. En una arquitectura web, llamaremos *front* o *front-end* al contenido que se ejecuta en el lado del cliente y que por lo tanto, permite al usuario interactuar con los servicios que proporciona la aplicación. Por otro lado, el *back* o *back-end* es el proceso o conjunto de procesos que se ejecutan fuera del alcance del usuario de la aplicación, como por ejemplo en un servidor. Esta estructura se verá con más detalle a lo largo del capítulo 3.
- **Python.** Se trata de otro lenguaje de programación con una sintaxis bastante similar a *JavaScript*. En este caso, no profundizaremos en detalle en los usos de este lenguaje, pues solamente lo utilizaremos para elaborar un sencillo *script* que se encarga de generar una imagen con los resultados de una simulación sobre un circuito concreto, y así poder compararlos aquellos que obtengamos de la aplicación. Para ello, se hará uso de las librerías de *numpy* [11] y *matplotlib* [12].



Figura 3

Aunque con las herramientas presentadas hasta el momento se podría construir cualquier vista *front-end*, daremos un paso más y para facilitar la programación del software empleando los siguientes entornos y *frameworks*:

- **NodeJs.** Se trata de un entorno de ejecución [13] para *JavaScript* fuera del navegador. Será de utilidad para la instalación de librerías.
- **React.** Es un *framework* [14] de programación front-end cuyo principal objetivo es la construcción de la interfaz de usuario mediante la reutilización de componentes. Utiliza NodeJs como entorno de ejecución y JavaScript como lenguaje principal.
- **Bootstrap.** Durante el desarrollo de la aplicación, veremos que CSS puede complicarse bastante cuando queramos posicionar elementos en un lugar concreto de nuestra pantalla. Bootstrap [15] se encargará de facilitarnos esto, además de la creación de componentes básicos (botones, enlaces, *grid-layout*, ...) con estilos predefinidos.

2

Circuitos RC y RL

El corazón humano contrae sus músculos mediante pulsos eléctricos, pero en algunos casos estos no son capaces de ser generados regularmente. El continuo progreso de la ciencia ha permitido la creación de dispositivos llamados marcapasos cada vez más avanzados cuya función es generar estos pulsos y así mejorar la salud del corazón del paciente. Los impulsos eléctricos son creados mediante los llamados *circuitos de estímulos*, que no es más que un conjunto de resistencias y condensadores unidos a una fuente de alimentación; o lo que es lo mismo, un circuito RC. [16]

Las aplicaciones de este tipo de circuitos no se centran solamente en el área de la medicina, sino que también podemos encontrarlos en objetos de uso cotidiano. Si observamos algunos de los auriculares de última generación que llevan incorporados una serie de funciones que permiten aislar ruido del exterior, vemos que estos utilizan lo que se conoce como *filtros de paso*, que dependiendo de si se quieren escuchar frecuencias altas, bajas o en un rango, se utilizarán respectivamente los filtros de paso alto, bajo o de banda. Estos filtros no son más que circuitos RC donde sus componentes se encuentran dispuestos de diferente manera según el objetivo que se quiere conseguir. Y, aunque normalmente las frecuencias de filtrado ya vienen configuradas de fábrica, existen modelos en los que estas pueden cambiarse mediante el uso de resistencias variables.

Otro uso de estos circuitos serían en las *fuentes de luz estroboscópicas*. Se trata de una fuente de luz cuya emisión se produce de forma intermitente. Este tipo de dispositivos podemos encontrarlos en los *strokes* de un avión (los focos de color rojo y verde que se sitúan en los extremos de las alas y cuya función es posicionar a la aeronave para evitar colisiones) o en cámaras de fotografía para tomar instantáneas de objetos en movimiento, como se puede ver

en la figura 4.



Figura 4: Movimiento de un balón usando una *cámara estroboscópica*. https://upload.wikimedia.org/wikipedia/commons/3/3c/Bouncing_ball_strobe_edit.jpg

Por otro lado tenemos los circuitos RL. Un ejemplo de uso de este tipo de circuitos lo encontramos durante el proceso de combustión en un motor de gasolina. Para que esta se lleve a cabo, el combustible debe de mezclarse con una serie de gases inflamables a alta presión y posteriormente, se debe de producir una chispa que queme esta mezcla. Para ello se hace uso de las bujías, unos dispositivos compuestos por dos hilos separados entre sí entre los que se produce una alta diferencia de potencial creando así un arco voltaico que es capaz de iniciar la combustión en el interior del motor. Pero normalmente los vehículos que emplean estos motores suelen ser alimentados por baterías de 12V o 24V, siendo estos voltajes insuficientes para que las bujías puedan llegar a inducir esa chispa.

Además de en motores de combustión de gasolina, este tipo de circuitos podemos hallarlos en dispositivos de filtrado de frecuencia (similares a los construidos con circuitos RC) o en circuitos osciladores. Estos últimos pueden usarse para la construcción de generadores de ondas de radio o televisión y en receptores usados en la telefonía móvil. Estos se encuentran formados por una serie de condensadores, resistencias y bobinas conectadas, también llamados circuitos RLC. Aunque estos sean otro caso de uso de bobinas y condensadores, estos circuitos no son objeto de estudio en este trabajo así que a continuación, nos centraremos en el estudio de los circuitos RC y RL en corriente continua y estado transitorio.

2.1. El condensador y el circuito RL

En primer lugar, comenzamos definiendo qué es un condensador. Este se trata de un dispositivo compuesto generalmente por dos placas conductoras aisladas entre sí, a través de un medio vacío, aire o algún material aislante; comúnmente llamado diélectrico, en el que podemos almacenar energía eléctrica.

La cantidad de energía que puede almacenarse en un condensador depende de la relación que existe entre la carga de cada una de las armaduras y de la diferencia de potencial que se aplica entre ellas. A dicha relación se le conoce como *capacidad* y su unidad de medida en el Sistema Internacional(a partir de ahora S.I.) es el *faradio* (F); que es equivalente a un culombio (C) de carga por cada voltio (V) de potencial eléctrico aplicado.

$$C = \frac{Q}{V} \quad (1)$$

Los conductores más utilizados suelen tener forma cilíndrica, aunque también los hay planos o esféricos. La geometría de estos dispositivos afecta directamente a su capacidad, lo que puede complicar su cálculo si la figura del condensador es demasiado compleja. Así que para simplificar lo máximo posible tanto los cálculos como la posterior implementación de un condensador en la simulación, no tendremos en cuenta la forma del condensador sino que se trabajará directamente con la capacidad del mismo.

Ya se comentó en la introducción de este capítulo que un circuito RC es una asociación de resistencias eléctricas y condensadores. Es por eso que, como el objetivo que queremos alcanzar es elaborar una simulación que recoja los fundamentos de la carga y descarga del condensador, nos ceñiremos a los circuitos RC de primer orden en corriente continua; es decir, aquellos que cuentan con una resistencia y un condensador conectados en serie alimentados por una pila.

Partimos entonces de un condensador de capacidad C completamente descargado conectado en serie a una resistencia de valor R . En el instante de tiempo $t = 0s$, aplicamos una diferencia de potencial de ε V, cuyo valor coincide con el voltaje característico de la fuente

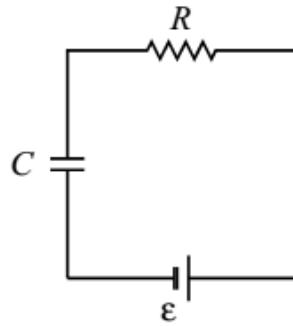


Figura 5: Representación de un circuito RC de primer orden.

utilizada (figura 5).

En ese mismo instante, la carga del condensador es nula, así que utilizando la definición de capacidad del condensador (1) llegamos a la conclusión de que el potencial en él también es cero. Por otro lado, tenemos que la energía potencial entre los bornes de la resistencia ha de ser máxima, y por consiguiente la intensidad de corriente que circula a través de ella. Esto se debe principalmente a que debe de cumplirse el principio de conservación de la energía; que en términos de potencial, la energía suministrada por la fuente debe de ser en todo momento igual a la consumida por los componentes pasivos del circuito: el condensador y la resistencia.

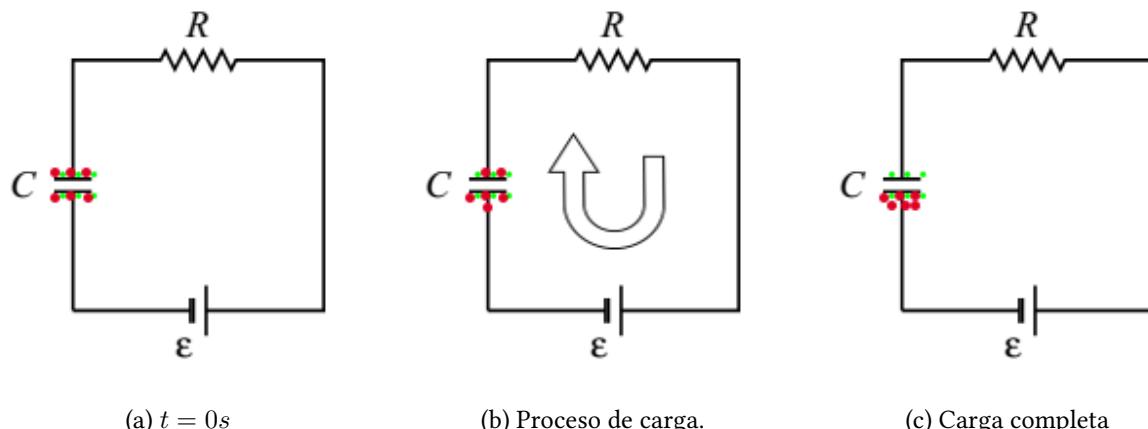


Figura 6: Fases del estado de carga de un condensador. Los puntos rojos hacen referencia a los electrones en movimiento.

Este principio, debe de mantenerse durante todo el proceso de carga hasta que la diferencia de potencial en el condensador sea máxima, y por tanto lo sea la carga almacenada en él. Como debe de volver a cumplirse el principio de conservación energético, ahora la energía potencial

entre los bornes de la resistencia es cero, y utilizando de nuevo la Ley de Ohm, tenemos que la intensidad de corriente es nula. Matemáticamente, podemos expresar este balance utilizando la siguiente ecuación diferencial:

$$\varepsilon = V_C(t) + V_R(t) = \frac{q(t)}{C} + R \frac{\partial q(t)}{\partial t} \quad (2)$$

, donde ε es la energía suministrada por la fuente por cada unidad de carga que la atraviesa, $V_C(t)$ es la *ddp* en el condensador y $V_R(t)$ la diferencia de potencial en la resistencia. Si la resolvemos, podemos obtener las expresiones recogidas en la tabla 1.

Concepto	Expresión	Resolución
Carga del condensador	$q(t) = C\varepsilon \left(1 - e^{\frac{-t}{RC}}\right)$	B.1.1
Intensidad de corriente	$I(t) = \frac{\varepsilon}{R} e^{\frac{-t}{RC}}$	B.1.2
Diferencia de potencial resistencia	$V_R(t) = \varepsilon \cdot e^{\frac{-t}{RC}}$	B.1.3
Diferencia de potencial condensador	$V_C(t) = \varepsilon \left(1 - e^{\frac{-t}{RC}}\right)$	B.1.4
Energía del condensador	$E(t) = \frac{1}{2}CV_C(t)^2$	B.3

Tabla 1: Modelado del estado de carga del condensador.

2.2. El condensador y el circuito RC

Con esto ya conocemos cuáles son las expresiones que se encargan de modelar la carga del condensador y que utilizaremos posteriormente cuando hagamos la implementación de este apartado de la simulación del circuito RC.

Continuamos entonces con el modelado de la descarga del condensador. Antes de dar comienzo a la descarga, supondremos que estamos en la situación descrita por la figura 6c. Partimos entonces de un condensador completamente cargado, cuyo valor de carga denotaremos por q_{max} ; y que además los valores de la diferencia de potencial en los terminales del dispositivo (V_L) y de la energía almacenada (E) en él son máximos. Puesto que no se está produciendo movimiento de cargas, la intensidad de corriente es nula, al igual que la diferencia de potencial

en la resistencia.

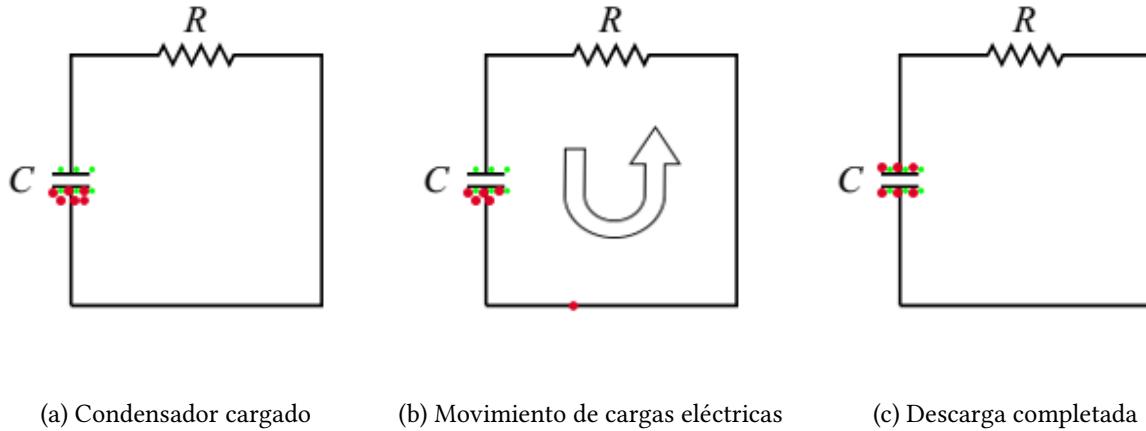


Figura 7: Fases del estado de descarga de un condensador

Así que para descargar el condensador, debemos de quitar del circuito la fuente de alimentación. Sin energía que suministrar, las cargas de la placa negativa del condensador se verán atraídas por las cargas de la placa positiva, produciendo así un movimiento de electrones que inducirán corriente eléctrica en sentido contrario a como lo hacían anteriormente. El número de cargas en movimiento disminuirá con el tiempo, produciendo así una caída de tensión en los bornes del condensador y por consiguiente, la pérdida de energía, tal y como se puede ver en la figura 7. Al final de este proceso el condensador volverá a estar en equilibrio electrostático.

Al igual que en el estado de carga, volvemos a realizar un balance energético. En este caso como no disponemos de una fuente de energía, entonces el valor de ε será cero. Así que planteamos la siguiente ecuación:

$$0 = V_C(t) + V_R(t) \quad (3)$$

, que si resolvemos de forma similar a la anterior, llegamos a obtener las siguientes expresiones que modelan la descarga del condensador (tabla 2).

Si estudiamos las expresiones matemáticas de cada una de las magnitudes físicas del circuito, observamos que todas ellas dependen del tiempo (t). Además, este tiempo se ve afectado

Concepto	Expresión	Resolución
Carga del condensador	$q(t) = q_{max}e^{-t/RC}$	B.2.1
Intensidad de corriente	$I(t) = \frac{-\varepsilon}{R}e^{\frac{-t}{RC}}$	B.2.2
Diferencia de potencial resistencia	$V_R(t) = -\varepsilon \cdot e^{\frac{-t}{RC}}$	B.2.3
Diferencia de potencial condensador	$V_C(t) = \varepsilon e^{\frac{-t}{RC}}$	B.2.4
Energía del condensador	$E(t) = \frac{1}{2}CV_C(t)^2$	B.3

Tabla 2: Expresiones que modelan la descarga del condensador

por un valor constante que viene dado por el producto de la capacidad del condensador y del valor óhmico de la resistencia. Definimos a este valor como la *constante de tiempo* RC . Este indica la velocidad de reacción del circuito y que, cuanto mayor sea el valor de esta constante, antes se consigue el estado de equilibrio del mismo. O lo que es lo mismo, el circuito RC de primer orden estudiado se encuentra en estado transitorio, pues los valores de las diferentes magnitudes varían desde un estado inicial a otro final.

$$\tau_{RC} = R \cdot C \quad (4)$$

Para comprobar que efectivamente el comportamiento de cada una de las magnitudes es el esperado, vamos a analizar los resultados de un circuito de ejemplo, tomando una fuente con un voltaje de $\varepsilon = 12V$, una resistencia con valor óhmico de $R = 1000\Omega$ y un condensador con una capacidad de $C = 10\mu F$. Calcularemos los valores de cada una de las magnitudes físicas hasta un tiempo de $t = 5 \cdot \tau_{RC}$ segundos, el cuál debería de ser suficiente para comprender qué está ocurriendo en cada caso.

Analizamos entonces los resultados obtenidos (figura 8). En primer lugar, cuando el circuito se encuentra dispuesto para la carga del condensador, podemos observar que a medida que transcurre el tiempo, la carga de este dispositivo aumenta y, por consiguiente, lo hacen el potencial y la energía almacenada en el mismo. El número de cargas negativas que pueden moverse a una placa del condensador a la otra será cada vez menor, así que la intensidad de corriente que circula por el circuito irá disminuyendo hasta ser cero. De la misma forma, el potencial de la resistencia también caerá hasta ser nulo.

Para la descarga del condensador, el movimiento de las cargas es en sentido contrario (se

mueven de la placa cargada negativamente, hasta la cargada positivamente). Esto impulsa un movimiento de cargas negativas por lo que aparece una intensidad de corriente. El condensador actúa como una fuente de alimentación, así que las cargas que se desplazan de una placa a otra hacen que la carga de este dispositivo disminuya junto al potencial entre sus terminales y la energía almacenada. Como el número de cargas es cada vez menor, ocurre igual que en el estado anterior: tanto la intensidad de corriente como el potencial en la resistencia disminuirán con el tiempo.

2.3. El inductor y el circuito RL

Como hemos visto en las diferentes aplicaciones durante la introducción de este capítulo, podemos hacer uso de los circuitos RL para incrementar la intensidad de corriente y así, conseguir por ejemplo, aumentar la tensión y crear un arco voltaico con la energía suficiente para quemar combustible en el interior de un motor de gasolina. Para ello utilizamos la **bobina**, un dispositivo formado principalmente por un hilo conductor enrollado alrededor de un núcleo, normalmente de aire o de algún material ferruginoso (como el hierro o la ferrita).

Supongamos entonces que tenemos una bobina y provocamos una intensidad de corriente eléctrica utilizando cualquier fuente de energía, como una pila. Esta variación de corriente (que inicialmente supondremos nula) produce una perturbación en el espacio que es conocida como *campo magnético*, y que por sus propiedades, afecta a todos los objetos con una naturaleza similar; que en este caso, son las cargas en movimiento de dicha corriente eléctrica.

Puesto que la intensidad de corriente ha cambiado, el campo magnético creado por este movimiento de cargas también se ve afectado y como consecuencia, volverá a alterar el valor de la intensidad. A esto se le conoce como el **fenómeno de autoinducción**, produciendo en la bobina una *F.E.M autoinducida* cuyo valor es directamente proporcional a la variación de *flujo magnético*¹. A esta magnitud que relaciona intensidad de corriente y flujo magnético recibe el nombre de *coeficiente de autoinducción*, cuyo valor depende exclusivamente del medio donde la bobina esté sumergida y de su geometría, siendo la unidad de medida en el Sistema Internacional el *henrio (H)*. Pero al igual que ocurría con el caso de los condensadores, para

¹El flujo magnético es la cantidad de campo magnético que atraviesa una superficie, es nuestro caso, la bobina

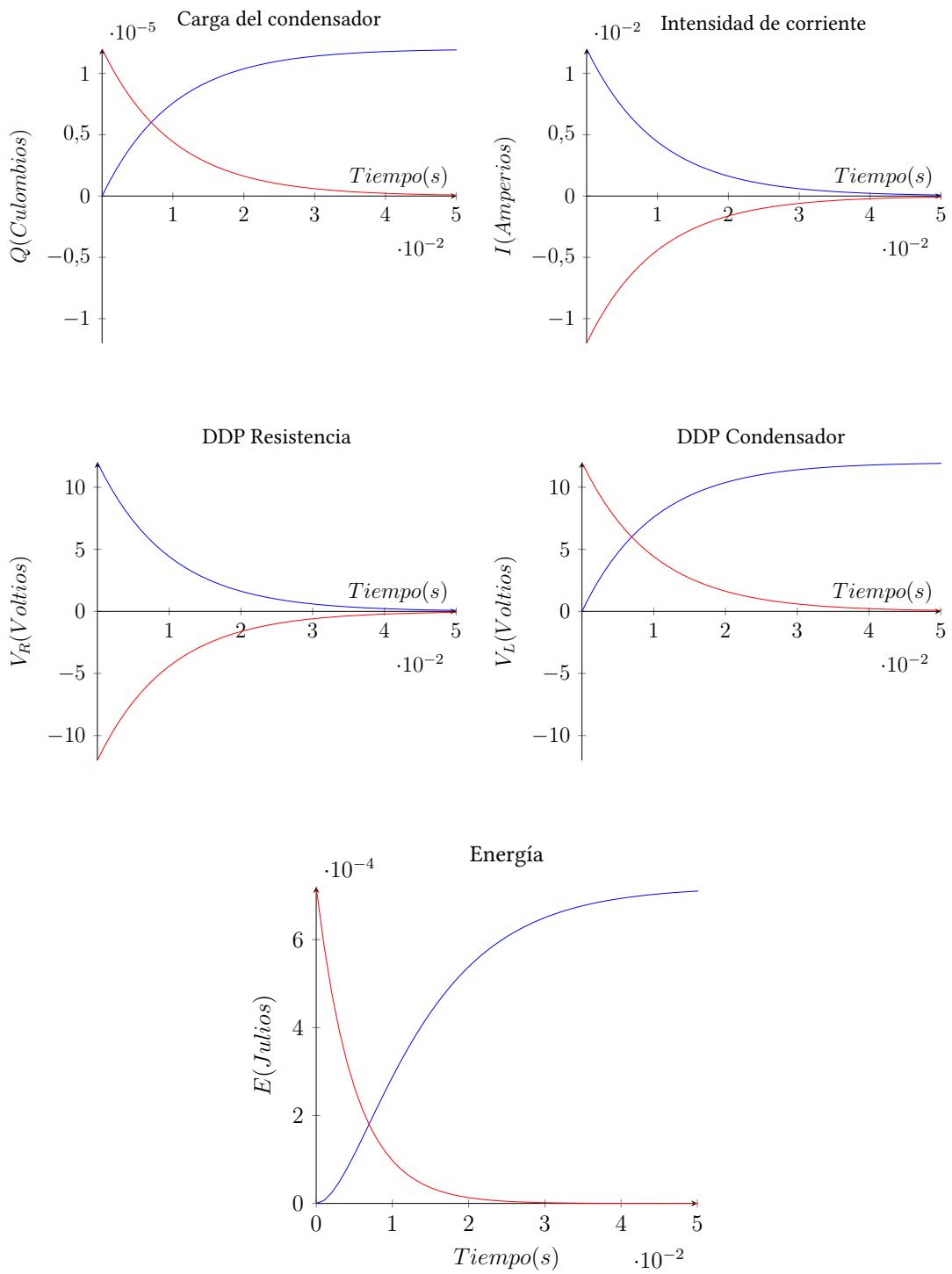
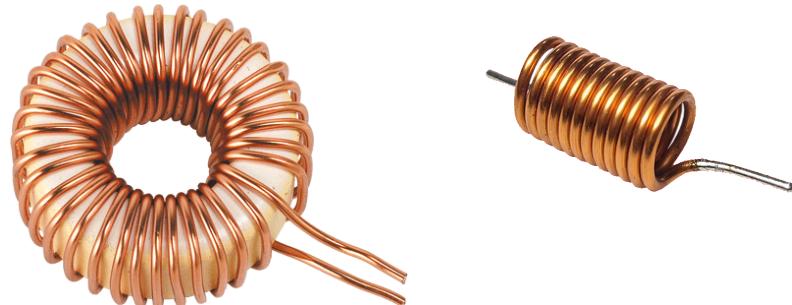


Figura 8: Evolución de las diferentes magnitudes del circuito RC de ejemplo. En *azul* se muestran los resultados en el estado de carga y en *rojo*, los de descarga.



(a) Bobina toroidal con núcleo férrico (b) Bobina espiral con núcleo de aire

Figura 9: Tipos de bobinas

simplificar los cálculos no vamos a tener en cuenta ni la geometría del inductor ni el medio donde se encuentre, sino que directamente usaremos el coeficiente de autoinducción asociado.

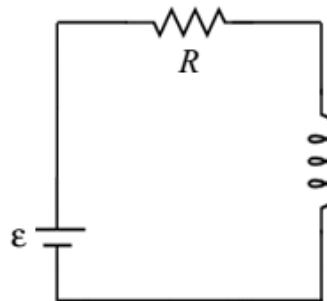
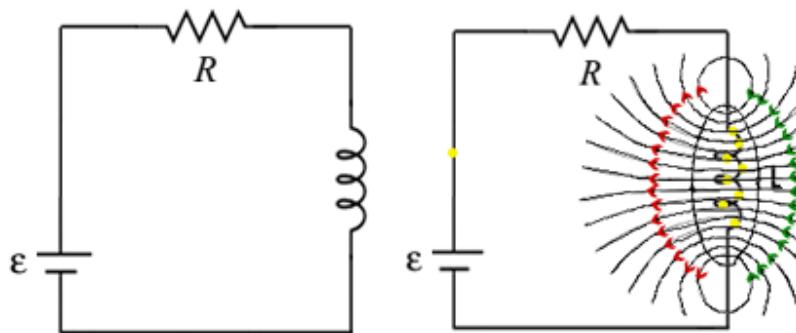


Figura 10: Representación de un circuito RL de primer orden.

Ya sabiendo qué es una bobina, un circuito RL es un conjunto de resistencias eléctricas y bobinas conectadas a una fuente de alimentación. El principal objetivo es comprender la evolución de las diferentes magnitudes físicas que afectan al circuito al inducir corriente eléctrica con estos dispositivos; así que en lugar de analizar un circuito complejo, el análisis y la simulación del mismo se realizará sobre un circuito RL de primer orden. Esto es, un circuito formado por una resistencia y una bobina conectadas en serie.

Partimos entonces de un circuito RL de primer orden, donde inicialmente supondremos que la intensidad de corriente que circula por él es nula. En el momento en el que aplicamos

una diferencia de potencial al circuito utilizando una fuente de alimentación con un *voltaje característico* de valor ε , se produce una emisión de cargas negativas desde el terminal con carga negativa de la pila. Estas cargas provocan la aparición de un campo magnético en la bobina cuando es atravesada por esta corriente eléctrica, el cuál se ve alterado por el *fenómeno de autoinducción* provocando así un incremento en la intensidad de corriente. Estas cargas en movimiento son impulsadas a través del hilo conductor hasta la placa con carga positiva de la pila; que para mantener el mismo potencial eléctrico entre sus terminales, se origina en su interior una sucesión de reacciones químicas para equiparar esta carga, impulsando cargas negativas desde el cátodo hacia el ánodo de la pila.



(a) Instante inicial. Intensidad nula. (b) Autoinducción de corriente (carga)

Figura 11: Almacenamiento de energía en una bobina

Por supuesto, el *principio de conservación de la energía* debe de cumplirse. Si bien el potencial en los bornes de la bobina disminuye a medida que se incrementa la intensidad de corriente, la diferencia de potencial en la resistencia aumenta. Cuando esta ddp en la resistencia se iguala al valor de la fuente ε , la intensidad del circuito será máxima y por lo tanto, la energía almacenada en el inductor en forma de campo magnético también lo es.

Obtenemos entonces las expresiones matemáticas del modelo realizando un balance energético del circuito, teniendo en cuenta la conservación de la energía en el mismo,

$$\varepsilon = V_L(t) + V_R(t) \quad (5)$$

, donde $V_L(t)$ es la diferencia de potencial en el inductor y $V_R(t)$ en la resistencia. De esta ecuación, podemos deducir las expresiones que se muestran en la tabla 3.

Concepto	Expresión	Resolución
Intensidad de corriente	$I(t) = \frac{\varepsilon}{R} \left(1 - e^{-\frac{Rt}{L}}\right)$	C.1.1
Diferencia de potencial resistencia	$V_R(t) = \varepsilon \left(1 - e^{-\frac{Rt}{L}}\right)$	C.1.2
Diferencia de potencial inductor	$V_L(t) = \varepsilon e^{-\frac{Rt}{L}}$	C.1.3
Energía del inductor	$E(t) = \frac{1}{2}LI(t)^2$	C.3
Flujo magnético	$\phi(t) = L \cdot I(t)$	C.4

Tabla 3: Expresiones que modelan la carga de la bobina

Continuamos entonces con el modelado de la descarga de la bobina. Inicialmente estaremos en el estado descrito por la figura 11b, cuando la intensidad de corriente que circula por el circuito es máxima. El proceso de disipación de la energía almacenada en el campo magnético de la bobina comienza cuando dejamos de suministrar energía con la fuente de alimentación, es decir, cuando $\varepsilon = 0$ (figura 12a).

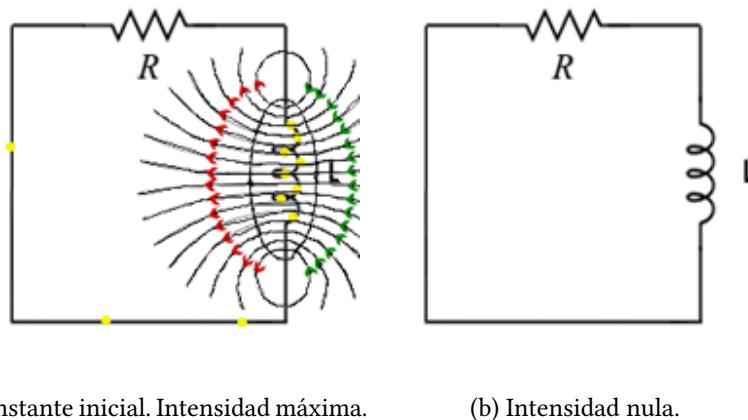


Figura 12: Disipación de energía en una bobina

Toda la energía cinética de las cargas en movimiento se disipan en forma de calor cuando estas chocan con las paredes del conductor (a este efecto se le conoce como *efecto Joule*), disminuyendo así la intensidad de corriente hasta ser cero. Puesto que la intensidad ha sufrido una variación respecto a su valor inicial, el campo magnético sufre variaciones (de nuevo

por el fenómeno de autoinducción). Además, conforme la intensidad disminuye también lo hacen las tensiones de la resistencia y el inductor, y con ellos la energía almacenada y el flujo magnético de la bobina. Realizando un nuevo balance energético del circuito, obtenemos la siguiente ecuación a resolver,

$$0 = V_L(t) + V_R(t) \quad (6)$$

, a partir de la cuál podemos deducir las expresiones que modelan el comportamiento de cada una de las magnitudes físicas que afectan al circuito durante la descarga de la bobina, tal y como se muestran en la tabla 4.

Concepto	Expresión	Resolución
Intensidad de corriente	$I(t) = \frac{\varepsilon}{R} e^{\frac{-Rt}{L}}$	C.2.1
Diferencia de potencial resistencia	$V_R(t) = \varepsilon e^{\frac{-Rt}{L}}$	C.2.2
Diferencia de potencial inductor	$V_L(t) = -\varepsilon e^{\frac{-Rt}{L}}$	C.2.3
Energía del inductor	$E(t) = \frac{1}{2}LI(t)^2$	C.3
Flujo magnético	$\phi(t) = L \cdot I(t)$	C.4

Tabla 4: Expresiones que modelan la descarga de la bobina

Como podemos observar en todas las expresiones que modelan el circuito RL, todas las magnitudes físicas estudiadas dependen del tiempo (t). Este se encuentra afectado por un valor constante que depende del coeficiente de autoinducción y del valor óhmico de la resistencia, al cuál llamaremos *constante de tiempo RL*. Este indica la velocidad de reacción del circuito, y cuanto mayor sea este valor antes se alcanza el estado de equilibrio del circuito. Denotamos a esta constante con el símbolo τ_{RL} . Decimos entonces que este circuito RL se encuentra en estado transitorio; es decir, cada una de las propiedades físicas que lo definen varían desde el estado inicial al final.

$$\tau_{RL} = \frac{L}{R} \quad (7)$$

Para comprobar entonces que el comportamiento de cada una de las magnitudes físicas es el esperado, analizaremos el siguiente circuito de ejemplo. Tomamos una fuente con un voltaje

$\varepsilon = 12V$, una resistencia de valor óhmico de $R = 10\Omega$ y una bobina con un coeficiente de autoinducción con valor $L = 10H$. Calculamos los valores para cada una de estas magnitudes en una simulación con una duración de $t = 5 \cdot \tau_{RL}$ segundos, que debería de ser suficiente para ver la evolución de cada una de ellas.

Analizamos entonces los resultados de la simulación reflejados en la figura 13. Cuando la bobina se encuentra en estado de almacenamiento de energía, debido al *fenómeno de autoinducción* anteriormente explicado, tanto la intensidad de corriente como la cantidad de campo magnético (o *flujo magnético*) aumentan; y por consiguiente, lo hacen la energía almacenada en este dispositivo así como la diferencia de potencial en la resistencia. Esta última como se puede ver en los resultados, aumenta hasta tener un valor igual al de la fuente de alimentación. Como además debe de cumplirse el principio de conservación de energía, el potencial de la bobina cae hasta ser cero.

Por otro lado, cuando la bobina se encuentra en estado de disipación de energía, partimos de un circuito dónde la corriente que circula por él es máxima. Cuando desconectamos la fuente, podemos observar cómo se sufre una caída en esta intensidad de corriente debido a qué cada vez el número de cargas negativas en circulación es menor. Por lo que, tanto la energía como el flujo magnético y los potenciales en ambos dispositivos, también serán nulos cuando esta intensidad sea cero.

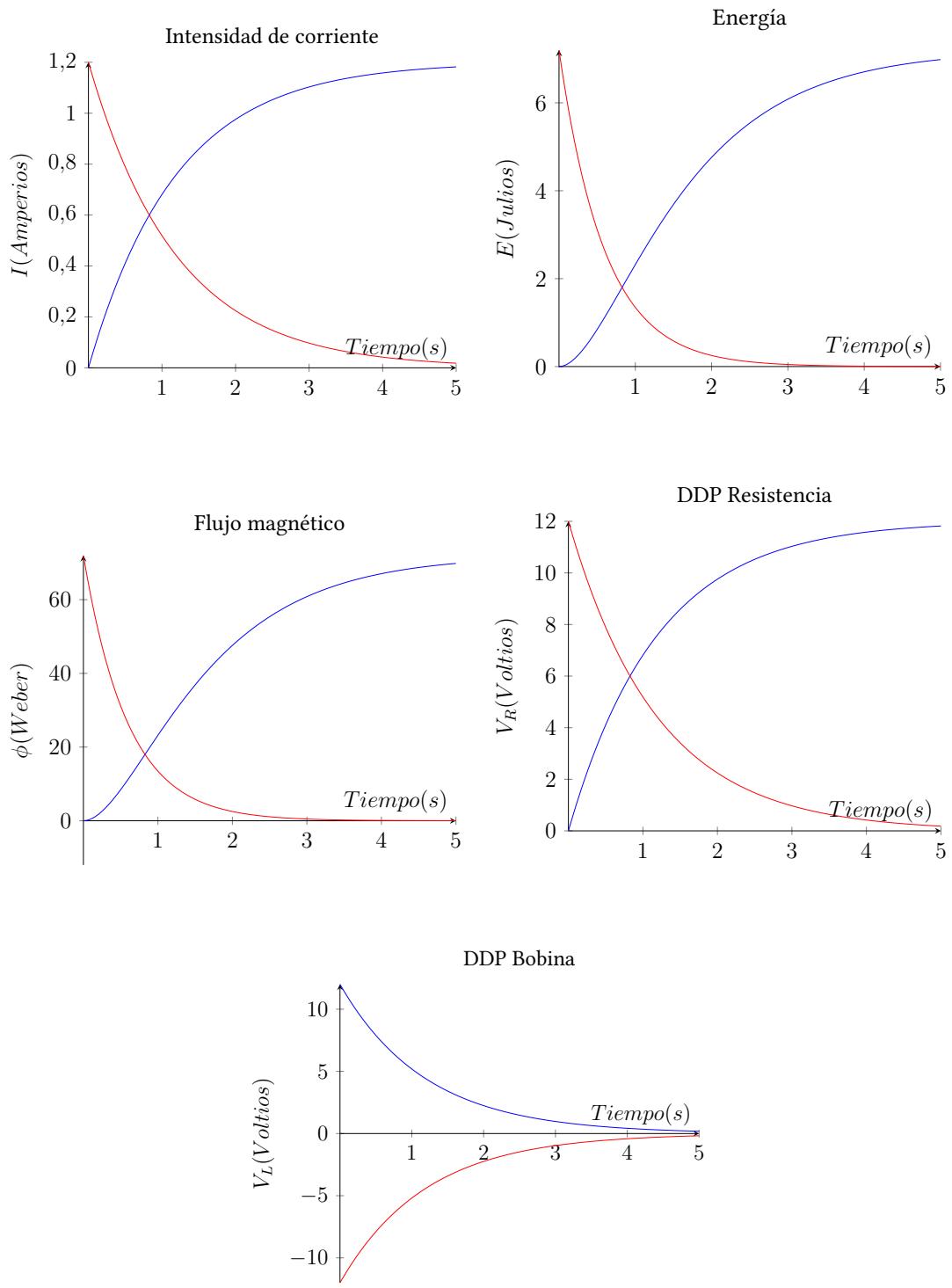


Figura 13: Evolución de las diferentes magnitudes del circuito RL de ejemplo. En *azul* se muestran los resultados en el estado de carga y en *rojo*, los de descarga.

3

Estudio de las tecnologías

Una vez estudiados los circuitos RC y RL de primer orden en corriente continua y estado transitorio, podemos utilizar las expresiones de cada uno de los modelos matemáticos que los describen para implementar la simulación en un programa de ordenador. Pero antes, debemos de hacer un estudio de las posibles tecnologías que nos permiten realizar esta implementación, pues dependiendo del tipo de aplicación, serán más convenientes unas tecnologías que otras.

3.1. Tipos de aplicaciones según la compatibilidad

Comenzamos el estudio de las tecnologías seleccionando el tipo de aplicación que más se ajuste a nuestras necesidades, dependiendo de la compatibilidad con los diferentes dispositivos del mercado. Distinguiremos entre tres grandes grupos de aplicaciones.

- **Navitas.** Este tipo de aplicaciones son desarrolladas para ser lanzadas en una plataforma en específico. Debido a esto, una clara ventaja que presentan este tipo de *software* es que, al estar diseñado para ser ejecutado en una arquitectura propia, su rendimiento respecto a los otros grupos es muy superior. Estas aplicaciones son útiles cuando se requiere un alto rendimiento y se quiere aprovechar el máximo de los recursos *hardware* del sistema. Y en cuanto al coste, al tener que desarrollar el *software* una vez por cada dispositivo con el que queramos que este sea compatible, se necesitará mayor tiempo de desarrollo y por lo tanto, un alto coste de producción.
- **Híbridas.** En este segundo grupo encontramos aquellas aplicaciones que pueden ser ejecutadas en varias plataformas. Gracias a esto, la mayor parte de los dispositivos pueden hacer uso de estos programas, eso sí, el acceso a las funcionalidades del *hardware* del

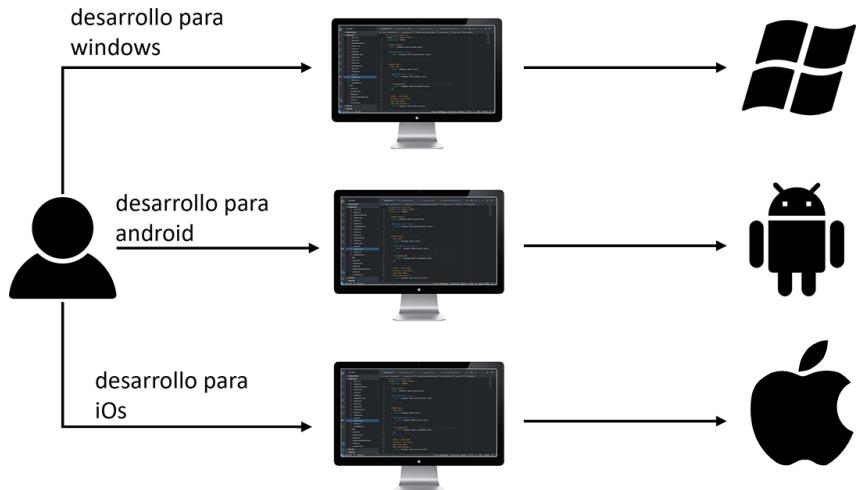


Figura 14: Aplicaciones Nativas

dispositivo se encuentran más limitadas. Esto ocasiona un decremento en el rendimiento de la propia aplicación, por lo que la velocidad del *software* aumentará o disminuirá dependiendo de las características del dispositivo.

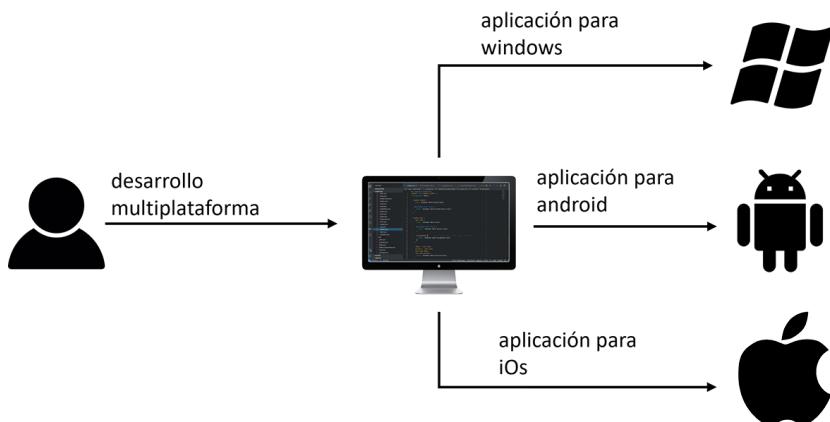


Figura 15: Aplicaciones Híbridas

- **Web.** Existe una estrecha correlación entre este grupo y el anterior, aunque las tecnologías utilizadas para su desarrollo no sean las mismas. Al igual que las *aplicaciones híbridas*, las *aplicaciones web* están orientadas a ser ejecutadas en varias plataformas. La única diferencia es que es necesario el uso de un navegador web para poder usarlas, y por lo tanto, requieren de conexión a internet. Esto puede resultar en una desventaja en caso de que se sufra una pérdida de conexión; sin embargo, tanto el coste de producción como el tiempo de desarrollo de este tipo de aplicaciones son muy inferiores respecto

a los otros grupos. Al igual que ocurría con las aplicaciones híbridas, el acceso a características *hardware* del dispositivo para la optimización del *software* se encuentran muy limitado. Y en cuanto a niveles de rendimiento, las aplicaciones web se encuentran muy por debajo en relación a los otros dos grupos.

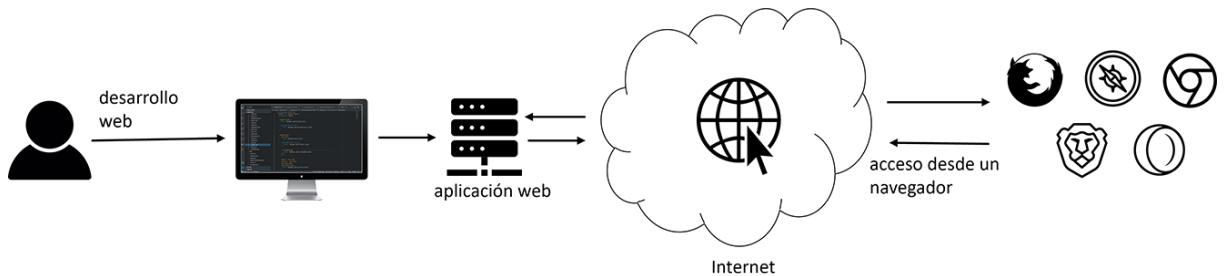


Figura 16: Aplicaciones Web

Entre estos grupos, debemos de seleccionar aquél que se ajuste a nuestras necesidades. Es preferible que el *software* que vamos a desarrollar sea accesible desde cualquier dispositivo, pues lo más probable es que los futuros usuarios no dispongan de una arquitectura específica para poder ejecutar la aplicación. Descartamos entonces las aplicaciones nativas. Otro punto importante a analizar es el tipo de acceso. Por un lado, las aplicaciones híbridas obligan al usuario a descargar el programa e instalarlo en su dispositivo, obligando así un uso de memoria del mismo. Sin embargo, las aplicaciones web no requieren de dicha instalación y, además, pueden ser accesibles desde cualquier dispositivo que tenga instalado un navegador y por supuesto, una conexión a internet. Al tratarse de una simulación sencilla que no va a requerir un alto rendimiento y la potencia *hardware* necesaria va a ser pequeña, el tipo de aplicación que más se ajusta a lo que estamos buscando son las *aplicaciones web*.

3.2. La web y su evolución

Si tuviésemos que dar una definición de qué es Internet, diríamos que se trata de una red de intercambio de información cuyo propósito es la transmisión de datos por todo el mundo. Fue en 1963 cuando apareció el primer concepto de esta red bajo el nombre de ARPA (*Advanced Research Projects Agency*); un proyecto financiado por el departamento de defensa del gobierno norteamericano con el objetivo de establecer una comunicación descentralizada entre diferentes bases militares del país. No fue hasta seis años mas tarde cuando en la Universidad de

California se consiguió dar un paso más allá, conectando entre sí dos ordenadores situados en diferentes universidades del país.

El número de usuarios que utilizaban esta red en ese momento era de apenas unas miles de personas, y no fué hasta principios de la década de los 80 cuando el uso de esta red (conocida como *ARPANET*) comenzó a extenderse al mundo gracias al lanzamiento del ISP (*Internet Service Provider*), un servicio el cuál permitía a cualquiera con una dirección IP (*Internet Protocol*) conectarse a esta red; alcanzando a principios de la década de los 90 a la cantidad de unas cien mil máquinas conectadas. Sin embargo esta fue reemplazada por una nueva red creada por la NSF (*National Science Foundation*) a la que llamaron NSFNET y que inspiró a un grupo de científicos del CERN (*Centro Europeo para la Investigación Nuclear* o *Conseil Européen pour la Recherche Nucléaire*) en la creación de lo que hoy día se conoce como la **Web**, con el objetivo de disponer de un sitio que alojase información y que fuese además accesible desde cualquier lugar del mundo.

En su versión más primitiva, llamada la **web 1.0** [17], las páginas eran fijas y no era común que estas se modificasen; además, estas páginas eran estáticas, careciendo completamente de interactividad. Para el renderizado de la web predominaba el uso de lo que comúnmente se llaman *marcos* o *framesets* utilizando una *etiqueta* HTML con el mismo nombre. Por definición, una etiqueta o *tag* en este lenguaje de hipertexto es el nombre que reciben cada uno de los componentes con el que se estructura un documento HTML.

Con ellas se establecía una estructura de diferentes marcos dentro de una página y, en cada uno de estos *framesets* se ejecutaba un documento HTML diferente, obteniendo una modularización del documento principal. Otra característica relevante es el envío de los formularios. Mientras que hoy día esto se realiza mediante una serie de peticiones del protocolo HTTP (*get*, *post*, *put*, ...) en ese momento la respuesta del formulario era enviada a través de un cliente de correo electrónico.

La segunda fase de la web se inició con la aparición de las primeras páginas centradas en la distribución de recursos, es por eso que a esta se le conoce como la *web social* o **web 2.0**. Ahora los sitios web no se encuentran limitados a mostrar información al usuario, sino que estos po-

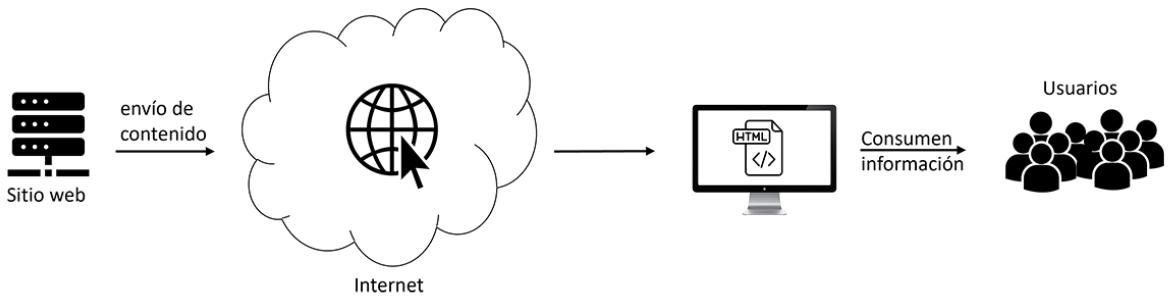


Figura 17: La web 1.0

drían añadir, editar, modificar, eliminar o compartir contenido con el resto de la comunidad, resultando ser las precursoras de las actuales redes sociales, blogs y wikis. A este nuevo grupo de páginas que permitían interactuar con ellas comienzan a llamarse *aplicaciones web*, dejando el nombre de *sitio* o *página web* a las plataformas que ofrecían información no dinámica. Se introducen nuevas técnicas de formato para la estructura del documento, como CSS (*Cascading Style Sheets*), AJAX (*Asynchronous JavaScript and XML*) para el funcionamiento asíncrono de la web, uso de URLs semánticas que nos permiten entender cuál es el contenido que se va a consultar y el formato JSON (*JavaScript Object Notation*) que es usado para dar estructura a los datos que se quieran transmitir entre servicios de una misma aplicación o como medio de comunicación con una API [18] (*Application Programming Interface*).

Actualmente esta versión de la web se encuentra en desarrollo, así que decimos que está en una *beta* constante. En *software*, nos referimos a beta como una de las primeras versiones del producto totalmente funcional, sobre las cuales se realizan pruebas en sus funcionalidades y rendimiento. Además, el continuo lanzamiento de nuevas tecnologías permiten un desarrollo más acelerado de las aplicaciones web, las cuales ofrecen un mayor rendimiento y una mejor experiencia para el usuario. Sin embargo, mientras que la web 2.0 está en construcción, ya se está hablando de una nueva fase en la web.

La **web 3.0** o *web semántica* tiene su propósito en facilitar la navegación e indexación de las diferentes aplicaciones y sitios web, haciendo uso de técnicas avanzadas de análisis de datos e inteligencia artificial, consiguiendo así que la web se convierta en una BBDD (base de datos) mundial descentralizada. Cada uno de los usuarios de internet, dispondrá de un perfil

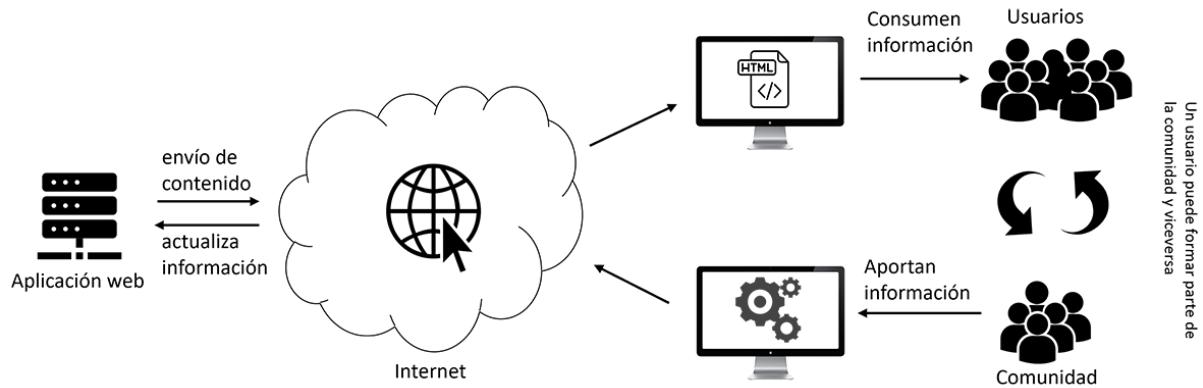


Figura 18: La web 2.0

único que usará para la autenticación de los diferentes servicios disponibles en la nube, creando así un espacio con mayor seguridad, privacidad, una experiencia personalizada durante la navegación en internet o mayor continuidad de los servicios (al tratarse de una red descentralizada, el número de interrupciones o cortes de conexión disminuirá). Pero esta nueva versión también podrá presentar una serie de desventajas: se requerirá de un hardware que ofrezca mayores prestaciones que sea capaz de soportar la transacción de mucha más cantidad de información, actualización de las diferentes aplicaciones y sitios actualmente desplegados o una nueva manera de proteger nuestros datos.

3.3. Estructura de una aplicación web

Otro aspecto importante a tener en cuenta en el desarrollo, es la estructura base del proyecto. Si bien el lenguaje de programación puede facilitar y ayudar en la implementación, la buena definición de una estructura puede favorecer al desarrollo de la misma. Aunque la base de una aplicación de este tipo no se está bien definida y existe cierta libertad en su configuración y organización, podemos establecer un modelo generalista que divide internamente una aplicación web en tres capas bien diferenciadas.

- La primera de las capas llamada **capa de datos** o **capa lógica** es la parte más abstracta e interna de la aplicación. En ella definiremos el esqueleto principal de cómo organizar nuestros datos y archivos multimedia.
- Por encima encontramos la llamada **capa de negocio**. En esta, se implementa toda la lógica necesaria que permite interactuar con la *capa lógica*, como por ejemplo añadir,

eliminar o modificar contenido de la misma. Se trata de un gestor de información encargado de la comunicación entre la *capa de datos* y el usuario de la aplicación, quién hace uso de la llamada *capa de presentación*. A este gestor también se le conoce como *middleware*.

- En el nivel más alto, tenemos la **capa de presentación**. Mientras que las dos anteriores se ejecutan en el lado oculto de la aplicación (lo que llamaremos servidor), esta lo hará en el lado visible de la misma (cliente). En ella se hace una interpretación de las diferentes peticiones de los usuarios haciendo uso de las operaciones implementadas en la *lógica de negocio*. Las acciones que un usuario puede hacer dependerá de entre otras cosas del diseño y restricciones que se presenten en esta capa.

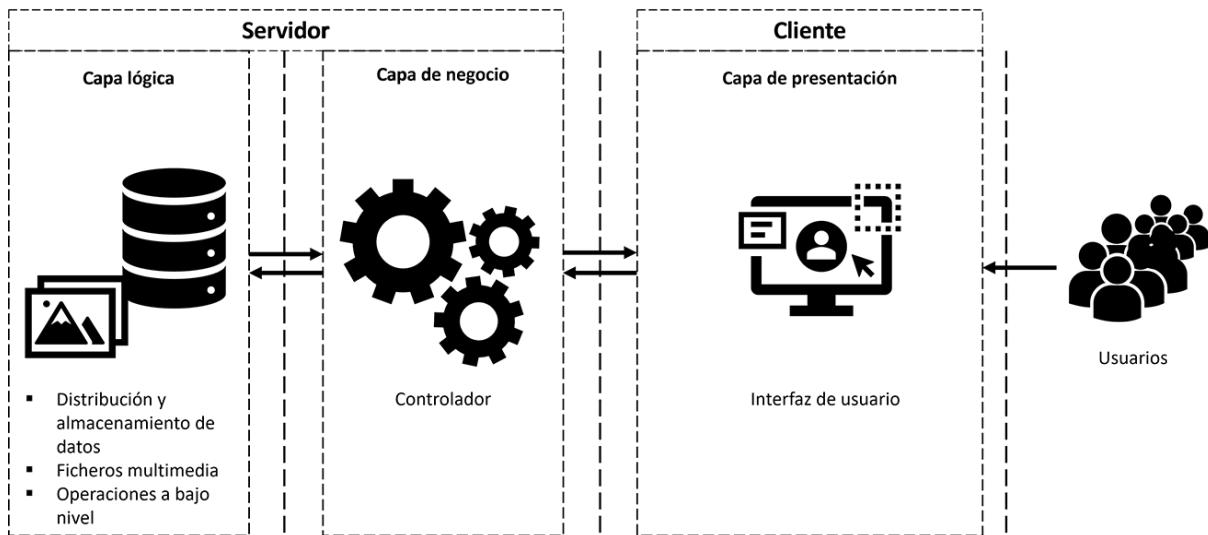


Figura 19: Aplicación web por capas

Para simplificar aún más esta división por capas, podemos añadir un nivel más de abstracción. Este se basa en diferenciar las dos partes de la aplicación que están interactuando entre sí. Como se puede observar en la figura 19, el usuario actúa directamente con la interfaz gráfica, ignorando completamente todo lo que ocurre por detrás: distribución y almacenamiento de los datos, operaciones a bajo nivel, procedimientos ejecutados en el controlador de la capa de negocio, ... Nombraremos entonces a todas aquellas capas que interactúen de forma directa con el usuario como **front-end** de la aplicación mientras que, todo lo que sea ajeno a él será el **back-end**.

Esta división nos permite agilizar el proceso de desarrollo, así como facilitar el despliegue de la aplicación. Por un lado, como se vió en el primer capítulo, el ciclo de vida del *software* consta de varias etapas y una de ellas es la fase del *diseño*. En ella se llevará a cabo la construcción a muy alto nivel de cada una de las capas de la aplicación, como por ejemplo la nomenclatura de las acciones en el controlador o la organización de los datos dentro de la BBDD. Si el diseño es lo bastante bueno y se encuentra bien documentado, deberíamos de ser capaces de desarrollar a la par el *back-end* y el *front-end*, pues al tratarse de módulos independientes y al usar un conector común, basta con que éste se encuentre bien definido.

En cuanto al despliegue, al tener separados en dos bloques diferentes la parte del cliente y del servidor, cada uno de los módulos pueden estar funcionando en *hosts* diferentes, teniendo así una aplicación web descentralizada. Esto no se aplica solamente a estas dos partes, sino también a cada una de las capas que forman el *back-end* y el *front-end*.

En la figura 20 podemos observar un diagrama con el despliegue de una aplicación web descentralizada. Disponemos de un *host* para el despliegue de cada una de las capas. Para enlazar cada una de ellas como vemos en la figura 19, basta con conectarlas a internet y dentro de cada una configurarlas para que redireccionen las peticiones a la capa correspondiente. Por ejemplo, tenemos en la nube una aplicación web basada en una enciclopedia sobre historia. Ahora, un usuario quiere añadir una entrada a esta wiki así que cuando entra al navegador y utiliza la URL de la aplicación, este accede a la dirección del servidor que contiene la capa de presentación. Si añade una entrada, la capa de presentación se comunica con el controlador a través de internet y este, con la capa lógica de igual manera. El usuario no necesita conocer dónde se alojan las capas de negocio o lógica (*back-end*) pero sí que puede acceder a la interfaz de usuario y utilizarla para añadir una nueva sección en la enciclopedia (*front-end*).

Y esto se puede complicar todo lo que deseemos. Podemos añadir una nueva capa de datos con su respectivo controlador de tal forma que tengamos dos servicios diferentes en una misma aplicación, distribuir el almacenamiento por varios puntos geográficos (por ejemplo, uno para datos y otro para fotos y videos) y unificar estos directorios como uno solo o tener varios controladores para una misma capa lógica.

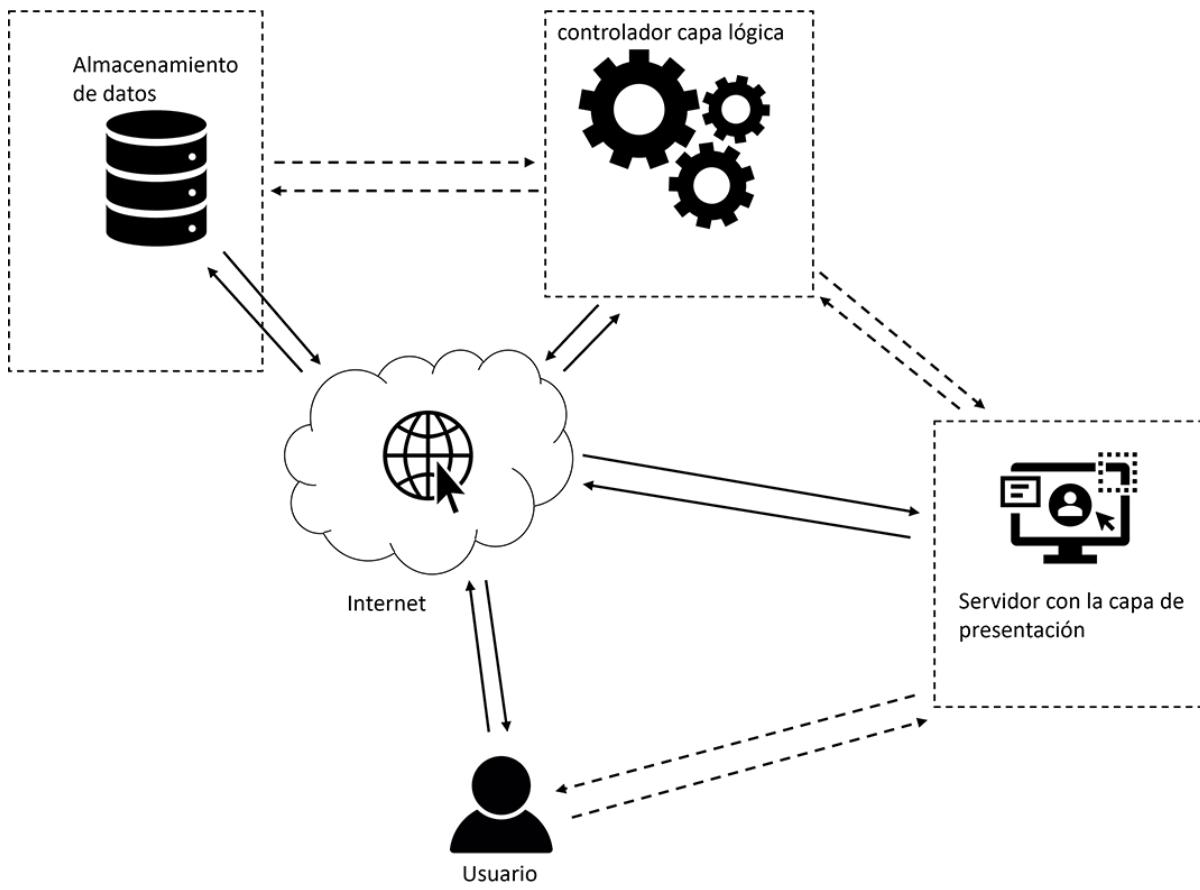


Figura 20: Diagrama de una aplicación web

Con esto hemos visto una idea general de cómo se puede organizar una aplicación web. Sin embargo, para la implementación de las simulaciones no va a ser necesario montar este tipo de infraestructura. En primer lugar, el almacenamiento de datos no va a ser necesario ya que los resultados que se van a generar serán meramente informativos para el usuario. Puesto que vamos a prescindir de una base de datos, tampoco necesitaremos de un controlador que se encargue de actualizar constantemente la información guardada. Solamente nos queda centrarnos en el *front-end* de la aplicación, reduciendo considerablemente el desarrollo del *software*. Pero antes, debemos de barajar todas las tecnologías posibles que nos ayuden en la implementación.

3.4. Tecnologías *front-end*

Antes de realizar un estudio de las posibles tecnologías en profundidad, la primera tecnología considerada para la implementación de las simulaciones fue la elaboración de un *applet*. Un

applet [19] es un *software* o programa de ordenador escrito en el lenguaje de programación *Java*. Estos se podían añadir a un documento HTML y, gracias a la *JVM* o *Java Virtual Machine* ejecutarlos en el navegador del cliente. El uso de este tipo de *software* se extendió rápidamente ya que al poder ejecutarse en cualquier dispositivo independientemente del sistema operativo y navegador, permitía a cualquier usuario ver el contenido que le proporcionaba el applet. Esta es la definición de lo que conocemos como una *aplicación multiplataforma*.

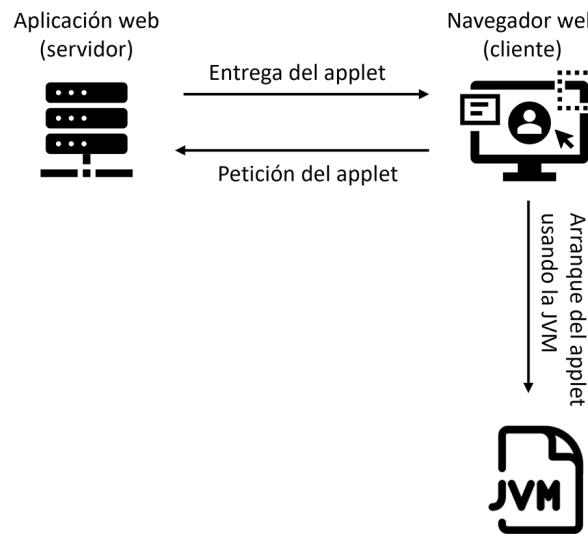


Figura 21: Arranque de un applet incrustado en una página web.

Pero con el tiempo comenzarían a aparecer vulnerabilidades en este tipo de aplicaciones que afectarían directamente a la JVM y posteriormente al sistema, exponiendo al usuario a ser víctima de ciberdelincuentes, en ataques de *ransomware* (técnica utilizada para encriptar los ficheros del sistema, denegando el acceso al propietario y pidiendo posteriormente un pago por su desbloqueo) o mediante la instalación de *malware* (se trata de un software malicioso cuyo principal objetivo es dañar cualquier dispositivo o red). Es por es que hoy día navegadores como *Chorme* o *Firefox* no estén habilitados para ejecutar *applets*, aunque existen otros que lo permiten si se hace el uso de ciertos *plugins*; es decir, programas cuyo objetivo es extender la funcionalidad de otro software sin necesidad de modificarlo desde código.

Algunas de las ventajas que presentaban este tipo de programas informáticos eran objetos animados e interactivos, reproducción de sonidos, implementación de funciones encargadas de proyectar los gráficos del *applet* correctamente o la visualización de gráficas y diagramas. Ya

que carece de sentido utilizar este tipo de software por estar prácticamente obsoleto, usaremos algunas de las nuevas herramientas introducidas con la *web 2.0*, como *CSS* o *JavaScript*.

3.4.1. HTML

Creado en 1991 por Tim Berners-Lee, HTML (del acrónimo anglosajón *HyperText Markup Language*) es un lenguaje basado en etiquetas utilizado para definir la estructura de las páginas web. En su primera versión se disponía de unas dieciocho etiquetas las cuales nos permitían añadir elementos de vídeo, texto o imágenes; y que hoy día con la versión de HTML5, el número de estas etiquetas asciende a más de 130. La definición de estos elementos se realiza encerrando entre corchetes el nombre del componente que queramos añadir y que en ocasiones se les puede agregar atributos. También es posible encontrar etiquetas que no necesitan de un elemento de cierre, como es el caso de la línea horizontal (*hr*) o el salto de línea (*br*).

```
1 <nombre-etiqueta atributo1="valor_atributo_1" atributo2="valor_atributo_2"> ... </nombre-etiqueta>
2
3
```

Figura 22: Definición etiqueta HTML.

La estructura elemental de un documento HTML queda definida por la declaración de los siguientes elementos:

- En primer lugar tenemos la sección **head** o cabecera de la página. En ella se define la información necesaria que describe el contenido del documento HTML, como por ejemplo, establecer el título que aparece en la pestaña del navegador, formato del texto a incluir el documento (codificación de los caracteres como UTF-8) o los diferentes enlaces que nos permitan acceder a *scripts* y *hojas de estilos* ya sea de manera local como remota. Su definición no es obligatoria.
- Como segundo grupo tenemos el cuerpo de la página (**body**). En él vamos a definir todo el contenido que va a ser visualizado en el navegador. Solamente se puede definir uno por documento.

```

1      <!DOCTYPE html>
2      <html>
3          <!-- Cabecera de la página -->
4          <head>
5              <!-- Definición de metadatos -->
6              <meta name="language" content="es" >
7              <meta name="description" content="descripción del sitio" >
8              <!-- Título de la página -->
9              <title> Título </title>
10             <!-- Enlace a ficheros de script -->
11             <script src="ruta-del-script" type="tipo-de-script"></script>
12             <!-- Enlace a ficheros de hoja de estilos -->
13             <link rel="stylesheet" href="ruta-de-hoja-estilos" >
14
15         </head>
16
17         <!-- Cuerpo de la página -->
18         <body>
19             <!-- Párrafo de cabecera. Va desde h1 hasta h6 (más a menos importancia) -->
20             <h1> ... </h1>
21             <!-- Párrafo de texto -->
22             <p> ... </p>
23             <!-- Sección de un documento -->
24             <div> ... </div>
25             ...
26         </body>
27     </html>
28

```

Figura 23: Ejemplo muy básico de la definición de un documento HTML. Además de las mostradas, existen más elementos que pueden utilizarse cada cuál tiene sus propios atributos. En [20] se definen con más detalle cada una de las etiquetas del ejemplo además de otras que no aparecen.

Sin embargo, todo lo que podemos hacer con HTML está limitado a aquello que ya se encuentra definido por las diferentes etiquetas y atributos que este lenguaje nos proporciona. Así que para dar forma a esta estructura y hacerla más atractiva para el usuario, utilizaremos lo que se conoce como una hoja de estilos.

3.4.2. CSS y Bootstrap

Definimos como una *hoja de estilos* a aquellos ficheros utilizados en la definición de los formatos de elementos HTML, cuya principal finalidad es mejorar tanto el aspecto visual como el diseño de los mismos. Para ello hacemos uso de **CSS** o *Cascading Style Sheets*. Se trata de un lenguaje de programación basado en la definición de atributos mediante el uso de *selectores*. Un selector es la unión que se encarga de vincular el elemento HTML definido en el documento web con el estilo definido. Esto nos permite aplicar formatos a diferentes objetos que se rijan por un mismo selector, como el uso de un identificador o mediante algún atributo específico de dicho elemento. En [21] se puede ver con más detalle los selectores existentes acompañados de ejemplos.

Otra característica importante es que la definición de estos estilos se puede realizar de dos formas diferentes. La primera de ellas es mediante el uso de una etiqueta *style* en la cabecera del documento. Entre el nombre de apertura y de cierre se definen todas las modificaciones que se quieran hacer a los elementos de nuestra página web utilizando los selectores previamente vistos, con la inconveniencia de que los estilos definidos solamente pueden ser aplicados a elementos de la misma página. Así que para evitar este problema, la opción más eficaz y que nos permite la modularización de código y por consiguiente una mejor organización del mismo, es definir nuestra hoja de estilos en un fichero independiente. Esto nos permite definir una serie de estilos que pueden ser reutilizados en varios documentos HTML, mediante el uso de la etiqueta *link* tal y como se puede observar en el código de la figura 23.

Aunque estos sean los dos métodos más utilizados, existe un tercero que puede llegar a carecer de utilidad si lo que se quiere conseguir es un código limpio y organizado. A cada objeto HTML se le puede agregar en su *tag* de apertura un atributo llamado *style*. La función de este es incluir un formato a dicho elemento. En cambio, se puede sacar provecho de este atributo si el diseño a aplicar es muy puntual y simple y además queremos evitar conflictos con elementos similares. Aunque sin duda alguna, la segunda opción es la más adecuada.

El uso de selectores no es complejo mientras que el diseño de la interfaz gráfica sea sen-

cillo. Si se quiere elaborar una interfaz un poco más compleja, el nivel de dificultad de CSS se incrementa bastante, convirtiendo la creación de los estilos de la página web en un trabajo complicado. Para facilitar su elaboración, podemos hacer uso de una serie de módulos ya creados que nos permite dar a nuestros elementos HTML formatos ya elaborados a los que posteriormente se les pueden cambiar el aspecto. Una de estas herramientas es **bootstrap**.

Creada por Mark Otto y Jacob Thornton, *bootstrap* surge como una biblioteca usada en el diseño e implementación de las UI (*Users Interfaces* o interfaces de usuario) de Twitter y no fué hasta 2011 cuando este proyecto se cataloga como software *open-source* o de código abierto. Un software es de código abierto cuando este es de dominio público y su código fuente puede ser descargado y utilizado para agregar nuevas funciones, corregir problemas de seguridad o simplemente realizar una optimización del mismo. Además, el software modificado puede volver a distribuirse siempre que esta sea bajo la misma licencia. La principal característica por la que se conoce a este *framework* es la consistencia que se añade a la página web. Existen un número bastante alto de selectores de clase ya definidos para la creación de botones, deslizadores, *inputs* de datos o barras de navegación que permiten a los desarrolladores utilizar una nomenclatura en común y facilitar así el desarrollo de la aplicación. Pero por lo que realmente se caracteriza *bootstrap* es en favorecer en la construcción de *webs responsive*, o lo que es lo mismo, permite redimensionar la web dependiendo del dispositivo sobre el que se esté utilizando, teniendo que producir una nueva vista adaptado al dispositivo en cuestión.

En el caso de la aplicación que vamos a desarrollar solamente vamos a contemplar aquellos dispositivos útiles que permitan al alumno una mejor visualización de los resultados de las simulaciones, como pueden ser un portátil o una tableta. Por ello, se tendrán en cuenta las pantallas cuya resolución sea mayor a diez pulgadas o con un ancho que supere los 1280 píxeles para evitar que los elementos de la ventana se colapsen entre ellos. En caso de querer adaptarlo para otros dispositivos (móvil u otras pantallas especiales), habrá que rediseñar e implementar la interfaz de usuario ajustadas a estos equipos.

3.4.3. JavaScript

JavaScript (JS) es un lenguaje de programación *script* e *interpretado* creado por Brendan Eich bajo el nombre de *Mocha*, en su trabajo por añadir nuevas funcionalidades al navegador web Netscape, además de tratar de mejorar las ya existentes [22]. Con una sintaxis heredada de *C++* y *Java*, JavaScript está habilitado para la programación orientada a objetos e incluso tiene variantes que permiten el tipado de sus funciones y variables (*TypeScript*).

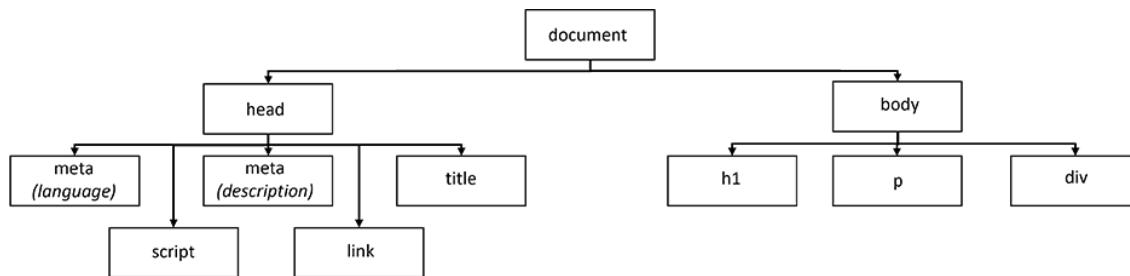


Figura 24: Ejemplo de árbol DOM

Considerado como el lenguaje por excelencia de la programación web, una de las principales características de JS es la creación de scripts sencillos que permiten dinamizar los documentos HTML mediante la modificación de sus elementos. Para ello, se hace uso del *DOM* o *Document Object Model*. Se trata de una estructura de nodos organizados jerárquicamente (árbol) en la que se incluye el contenido del sitio web; es decir, estructura del documento HTML con sus respectivos estilos en CSS. Todos estos árboles tienen un elemento raíz que hace referencia al documento sobre el que se va a ejecutar dicho script (*document*), del cuál se extienden el resto de elementos. Además, para poder interactuar con este árbol de una manera más fácil, JavaScript tiene integrado una API que nos permite acceder, modificar, crear o eliminar elementos [23]. Podemos ver en la figura 24 el árbol DOM correspondiente al código HTML de la figura 23.

Otra de las características de este lenguaje es que es interpretado. La ejecución de programas escritos en otros lenguajes de programación como *C++*, *Haskell* o *Fortran* sucede gracias a la intervención de un programa compilador que se encarga de traducir el código fuente escrito a código máquina, siendo este último el consumido por el procesador, ofreciendo cualidades

como una mejor velocidad, eficiencia y robustez. Luego, un lenguaje interpretado es aquel cuyos programas se ejecutan instrucción a instrucción directamente mediante el uso de un intérprete sin necesidad de ser traducido a código máquina en un paso intermedio. Aunque los lenguajes interpretados suelen ser más lentos en ejecución que los compilados, ofrecen ventajas como la de poder ser ejecutados en cualquier plataforma o aumentar el rendimiento de la aplicación; pues lenguajes como JavaScript al estar diseñados para ser utilizados en el servicio del cliente, disminuyen la carga en el servidor.

Aunque JavaScript sea un lenguaje de programación fácil de aprender y además se dispone de una API nativa que facilita la interacción con los documentos HTML, cuando la aplicación que queremos desarrollar es demasiado compleja la mejor opción a la que podemos recurrir es a utilizar un *framework* de desarrollo; al igual que ocurría con *bootstrap* para acelerar la creación de estilos CSS. Existen muchos *frameworks front-end* escritos con JavaScript, como pueden ser *Vue.js*, *Angular*, *Svelte*, *ReactJS* o *jQuery* entre otros. Cada uno de ellos presentan sus respectivas ventajas frente al resto aunque, si tuviésemos que elegir por relación entre dificultad de aprendizaje y velocidad de desarrollo, *ReactJS* sería el entorno más adecuado.

3.4.4. ReactJS

ReactJS es un conjunto de librerías *open source* escritas en JavaScript y que utiliza NodeJS como entorno de ejecución. Una de las características que definen a este *framework* de programación, es que permite el desarrollo de interfaces de usuario para la web basadas en *single-page*.

Mientras estamos navegando por una aplicación web en busca de información y durante la navegación tiene lugar una actualización de su contenido, normalmente seguiremos viendo la información sin esas modificaciones. Para renderizar el nuevo contenido debemos de realizar una recarga de la página, durante la cual se realiza una petición para obtener la nueva información ya actualizada. Sin embargo, las aplicaciones web que utilizan la tecnología de *single-pages* no necesitan realizar esta recarga de la página, ofreciendo así una mejor experiencia de usuario. Estas utilizan AJAX (*Asynchronous JavaScript and XML*), una técnica que permite recargar solamente aquellas partes de la web que han sufrido cambios; que en el ca-

so de este *framework* estos cambios se producen por la actualización independiente de los **componentes** de la página, facilitando así tanto la implementación como el renderizado de la misma. Algunos ejemplos de aplicaciones *single-page* son: Gmail, Google Maps o Paypal.

Un componente en React es la definición de cualquier objeto que queramos incluir en nuestra interfaz de usuario, como puede ser un párrafo de texto, una imagen, ... o elementos algo más complejos como un menú de navegación. La ventaja que estos presentan es que pueden ser reutilizados en cualquier parte del código, ahorrando así tiempo en la implementación.

Cualquier componente definido en React, consta de tres estados: inicialización (*mounting*), actualización (*updating*) y destrucción (*unmounting*).

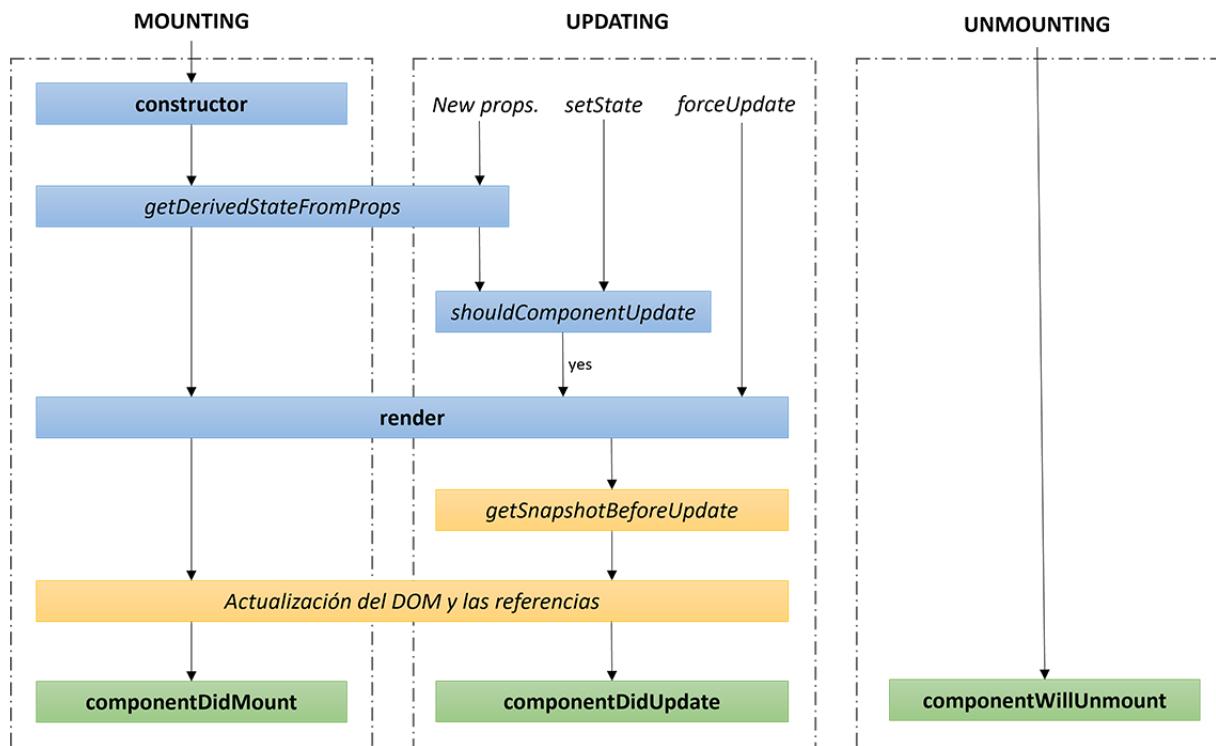


Figura 25: Ciclo de vida de un componente en ReactJS [24]

En primer lugar tenemos la etapa de *mounting*, que abarca desde la inicialización del componente con unas propiedades y estado predeterminados hasta que es renderizado y mostrado al usuario por primera vez. La linea que sigue esta etapa es la siguiente:

- En la mayoría de lenguajes que permiten la programación orientada a objetos (POO), existe un método llamado **constructor** que se encarga de inicializar el objeto cuando

se crea una instancia de él. En el caso de un componente de React, los propósitos del constructor son la creación del estado local del componente así como ligar las funciones asociadas a la captura de eventos a una variable (también conocido como *binding*). En caso de que no sea necesario hacer ninguna de estas operaciones, la definición del constructor se puede omitir.

- A continuación, se invoca un método llamado **getDerivedStateFromProps**, el cuál se invoca justo antes del renderizado del componente. Este es rara vez utilizado, y su principal objetivo es cambiar el estado local del componente cuando las propiedades (*props*) iniciales son modificadas. El uso de este procedimiento no es obligatorio cuando los valores de las *props* no cambian.
- Una vez creada la instancia del componente y establecido un estado inicial del mismo, lo siguiente que ocurre es la renderización del componente mediante el método **render**. Para ello se hace uso de JSX, una extensión de JavaScript que permite la definición de los componentes utilizando un lenguaje de hipertexto similar a HTML al que se le añade la potencia de JavaScript para personalizarlos. La sintaxis de JSX es igual a la de HTML, solo que además de poder utilizar las etiquetas predeterminadas de este lenguaje, también podemos añadir nuevas etiquetas con el nombre de un componente que hayamos creado.

Ya montado el componente, este se actualiza en el DOM y justo después se ejecuta el método **componentDidMount**. Normalmente este procedimiento es utilizado para extraer datos de una API o realizar algunas modificaciones del estado local. Su definición no es obligatoria.

Por otro lado tenemos la etapa de *updating* o actualización, la cuál podría decirse que es la etapa del ciclo en la que más tiempo pasa un componente; pues cada vez que hagamos una modificación en alguna de las propiedades, cambiemos el estado local o forcemos una actualización (*refresh*), se llevará a cabo una nueva renderización (*re-render*).

- Como vemos en el diagrama de la figura 25, la actualización de un componente ocurre por los tres motivos comentados en la línea anterior. En caso de que se actualicen las *props* se llama obligatoriamente al procedimiento **getDerivedStateFromProps** ya descrito,

y luego, en caso de que el componente deba de actualizarse, este se vuelve a renderizar. En cambio, si lo que modificamos es el estado o forzamos la actualización se invoca directamente el procedimiento encargado en la renderización del objeto.

En caso de que el componente no deba de actualizarse, simplemente esta etapa del ciclo finaliza y volvemos al inicio de ella (*shouldComponentUpdate*).

- Justo después se invoca **getSnapshotBeforeUpdate**, un método cuyo objetivo es capturar información sobre el DOM antes de que este cambie, aunque no suele utilizarse. Un ejemplo de uso lo podemos ver en *WhatsApp*, si tuviésemos que programar la interfaz en React. Cuando recibimos un nuevo mensaje de uno de nuestros contactos, justo encima de él observamos que aparece una alerta de que tenemos una nueva notificación. Para mostrar esta alerta (la cuál no es necesaria de guardar en el historial de mensajes), se toma la posición de *scroll* del último mensaje recibido, se inserta esta alerta y después el nuevo mensaje. Cuando ya hemos visto el mensaje, simplemente eliminamos esta alerta actualizando de nuevo la posición de *scroll*.
- Por último se actualiza el DOM y, posteriormente, se llama al procedimiento **componentDidUpdate**, cuyo objetivo es el mismo que el método *componentDidMount*, solo que en este caso se ejecuta cada vez que se actualiza el componente.

Por último tenemos la fase de *unmounting* o destrucción del componente. Esta etapa comienza cuando el componente en cuestión es eliminado del DOM de la página, invocando así el método **componentWillUnmount**, el cuál puede ser utilizado para limpiar o cancelar peticiones a una base de datos, invalidación de temporizadores (como por ejemplo en funciones que se ejecutan cada cierto tiempo) o eliminar registros creados en la etapa de *mounting* con el método *componentDidMount*.

Ya visto el ciclo de vida de un componente en React y los diferentes métodos y procedimientos que se utilizan para la creación y renderización en cada uno de ellos, podemos observar que la estructura de un componente equivale a la clásica definición de *clase* utilizada en la programación orientada objetos, la cuál se implementa en otros lenguajes como en el caso de *Java*, donde cada objeto tiene una serie de atributos (que corresponde al estado local) que se inicializan en su respectivo método constructor utilizando unos valores por defecto (las *props*

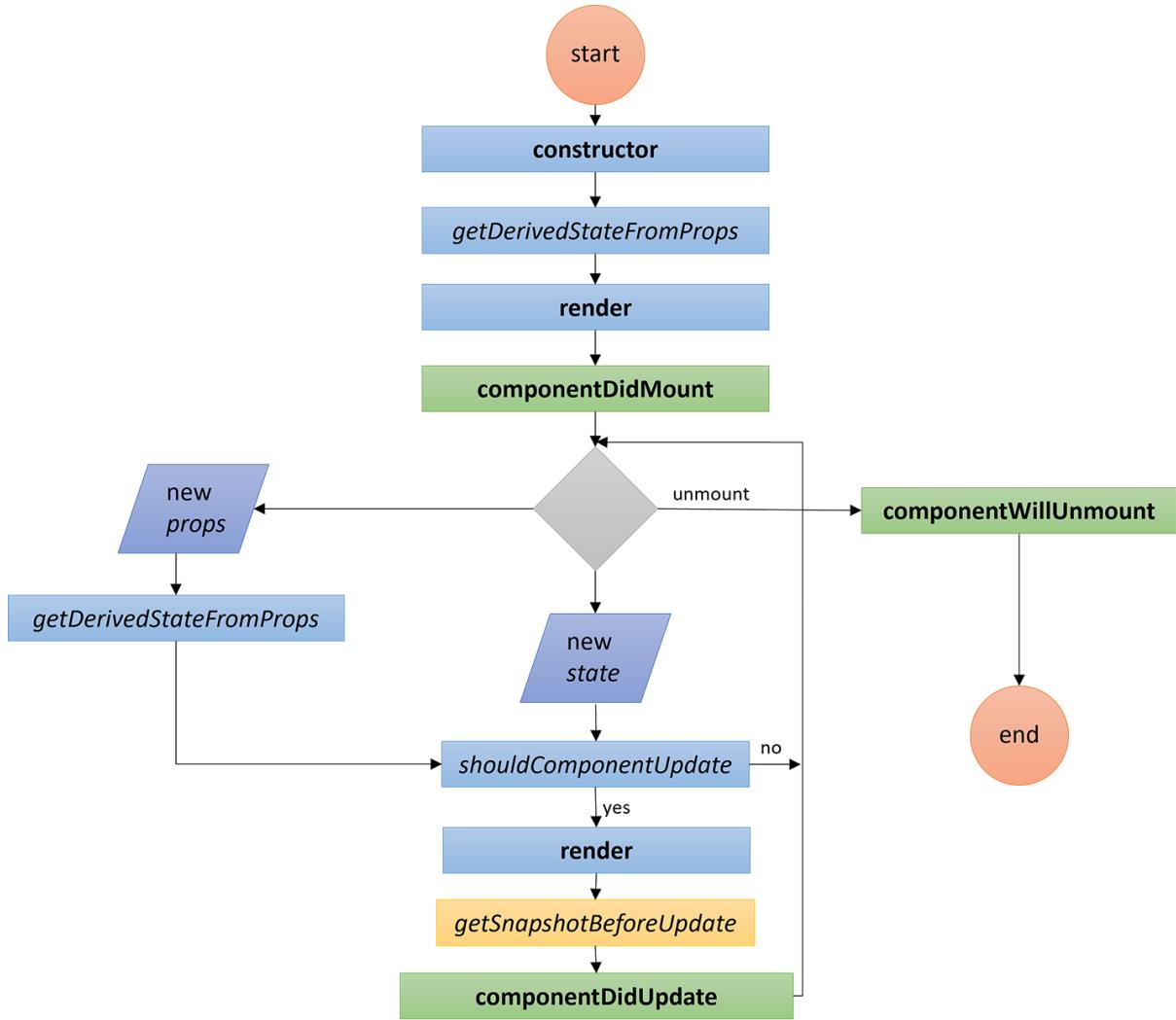


Figura 26: Diagrama de flujo de un componente en React.

dadas al crear la referencia del componente) y una serie de métodos que permiten actualizar e interactuar con sus atributos. Sin embargo, en las últimas versiones de ReactJS; concretamente a partir de la versión 16.8, se introduce un nuevo concepto en la programación de estos componentes llamado **hook**. Aunque la lógica sigue siendo la misma y en lenguaje base del *framework* también, esta nueva definición se encarga de sustituir la estructura clásica de un componente basada en el uso de clases por la división del mismo en funciones atómicas.

Cuando un componente crece en complejidad también lo hace el número de funciones declaradas, cada una con un propósito diferente. Supongamos que en el *state* del componente tenemos un atributo llamado *contador* que indica el número de usuarios que han visitado nuestra web y que, para actualizarlo, usamos un botón que haga una petición a una API. Utilizando

la definición de clase, el código implementado quedaría algo parecido a como se muestra en la figura 27.

```
1 import React, {Component} from "react";
2 export default class Contador extends Component {
3     constructor(props){
4         super(props);
5         this.state = {
6             contador : 0
7         }
8     }
9
10    setContador(nC) {
11        this.setState(prevState => {
12            return {
13                ...prevState,
14                contador: nC
15            }
16        });
17    }
18
19    async getNumeroUsuarios() {
20        //petición a la API y llamada al método "setContador"
21        //para cambiar el atributo del estado local "contador"
22    }
23
24    render(){
25        return(
26            <div>
27                El número de usuarios es : {this.state.contador}
28                <button onClick={(ev) => { this.getNumeroUsuarios(); }}>ACTUALIZAR</button>
29            </div>
30        );
31    }
32 }
```

Figura 27: Componente “Contador” con clases.

Si por el contrario decidimos de utilizar la definición de *hook* de un componente, recurriremos entonces a las funciones de JavaScript para su implementación. El contenido de estas funciones será el siguiente:

- Lo primero es la creación de las constantes que hagan referencia tanto a los atributos como a sus métodos correspondientes que actualicen el valor. Por defecto, estas se crean nombrándolas y referenciándolas a *useState*; aunque pueden modificarse para que hagan más operaciones. Esto hace referencia al constructor.
- A continuación deberán de definirse todas aquellas funciones y métodos que se encargan tanto del procesamiento de datos como de la llamada a APIs.

- Como toda función, esta debe de retornar algo, y este caso, es la vista del componente utilizando JSX. Desempeña el mismo papel que la función *render*.

```

1 import React, { useState, useEffect } from "react";
2 export default function Contador () {
3   const [contador, setContador] = useState(0);
4
5   async function getNumeroUsuarios () {
6     //petición a la API y llamada al método "setContador"
7     //para cambiar el valor de la variable "contador"
8   }
9
10  return (
11    <div>
12      El número de usuarios es : {contador}
13      <button onClick={(ev) => {this.getNumeroUsuarios()}}>ACTUALIZAR</button>
14    </div>
15  )
16}

```

Figura 28: Componente “Contador” con hooks.

Con esto damos por finalizado el estudio de las tecnologías a utilizar en la implementación de las simulaciones, por lo que en el siguiente capítulo continuaremos con su programación, utilizando para ello las expresiones de la tabla 5.

4

Implementación y pruebas

4.1. Simulación de los circuitos RC y RL

El primer paso a llevar a cabo en la implementación de las simulaciones de los circuitos RC y RL, será plantear un algoritmo que nos permita generar los datos correspondientes a cada una de las magnitudes físicas de ambos circuitos.

4.1.1. Etapas del proceso

Comenzamos entonces identificando cuáles son las etapas en las que podemos dividir el proceso de llevar a cabo una simulación:

- En primer lugar tenemos la etapa de *stop* en la que, como su propio nombre indica, el sistema se encuentra totalmente en reposo, pausando así los subprocessos de generación y representación de resultados. Por defecto, esta etapa será activada cada vez que se cree una nueva instancia de cualquiera de los dos circuitos.

Dado que el proceso de la simulación se encuentra pausado en esta fase, para pasar a cualquiera de las otras (ver siguientes) el usuario debe de forzar al sistema utilizando el interfaz de usuario de la aplicación.

- La segunda etapa a destacar y que podríamos considerar como la principal, consta de un subprocesso cíclico en el que se lleva a cabo la generación, el procesamiento y el almacenamiento de los datos; y paralelamente, su representación. Esta etapa estará activa

siempre y cuando la simulación no haya finalizado por alguna de las siguientes causas: el circuito a simular no haya alcanzado su estado de equilibrio y por lo tanto, no tiene sentido seguir generando resultados; se haya cumplido alguna de las restricciones impuestas por el usuario; o bien, la simulación haya sido reiniciada manualmente. Este último caso es algo especial, el cuál veremos en el siguiente punto.

A esta etapa del proceso de la simulación la denominaremos etapa activa o *running*.

- Como hemos comentado anteriormente, es posible reiniciar la simulación de forma manual, iniciando entonces la etapa de *reload* o reinicio. En ella se hace una limpieza de todos los resultados que se han ido generando previamente y justo después, se vuelve a iniciar la simulación del último circuito establecido, volviendo así a la etapa de *running*.
- Por último, tenemos el subprocesso de cambiar de valor un componente del circuito. Aunque esta no será considerada como una etapa más del proceso sino más bien como un paso intermedio entre la simulación de dos circuitos diferentes ya que en este no se actúa directamente sobre los resultados de la simulación actual, cabe destacar que al igual que al igual que la etapa de *reload*, se trata de un subprocesso donde la finalización de este no depende del usuario, sino del propio flujo del sistema.

Para comprender mejor el funcionamiento global de la simulación, se ha elaborado el diagrama de la figura 29, en el que se puede ver con mayor detalle la interacción entre los diferentes subprocessos.

A continuación presentamos los aspectos más importantes de las simulaciones, en los que trataremos conceptos como *ventana de datos* o *condiciones de parada*.

4.1.2. Generación de datos

Que el sistema se encuentre en las etapas de *stop* o *reload* depende más de las acciones que tome el usuario de la aplicación que del propio flujo del proceso, por lo que nos centraremos en la etapa activa para la generación de datos en tiempo real. Nos apoyaremos entonces en el método *componentDidMount* del componente de la simulación, que como ya sabemos,

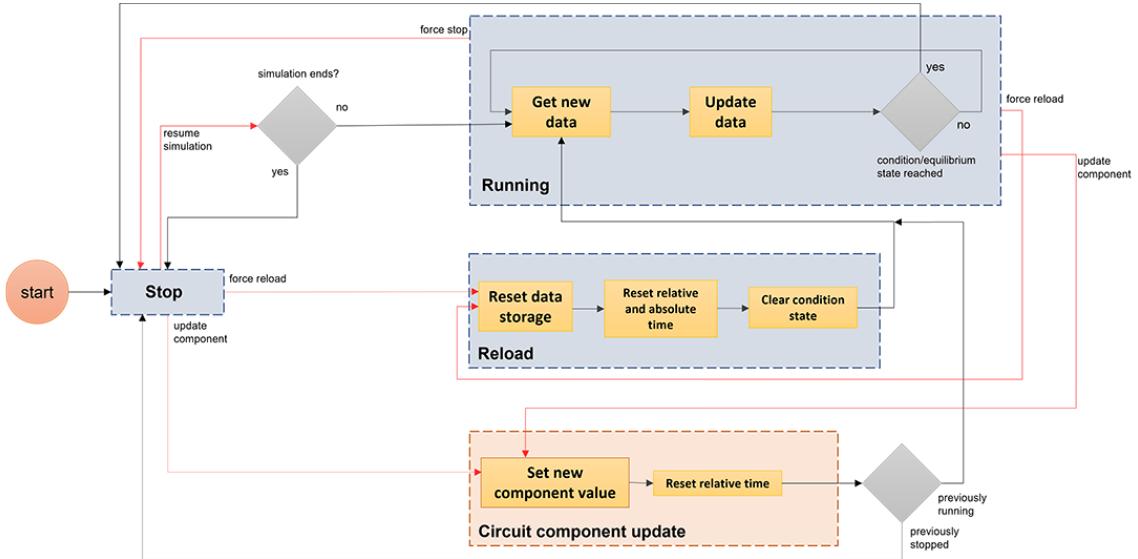


Figura 29: Representación abstracta la aplicación. Las flechas en *rojo* indican las acciones que el usuario puede tomar sobre la aplicación; en *negro* las acciones que toma el sistema por su cuenta.

solamente se ejecuta una vez durante la vida de un componente (concretamente durante su creación).

Aprovechando la capacidad de JavaScript para crear funciones privadas en el cuerpo de otros métodos y funciones, crearemos la instancia de una nueva función local y asíncrona dentro del procedimiento *componentDidMount*, la cuál se ejecutará repetitivamente cada cierto tiempo, como por ejemplo cada 150 milisegundos (a estas funciones también se le conocen como *setInterval*). La elección del valor de este intervalo se toma en base a la velocidad de generación de los datos que queramos. Si este es muy grande, el tiempo de espera entre la producción de dos valores crece demasiado, ralentizando así la representación de los mismos; pero si es muy pequeño, la cantidad de información generada podría llegar a ser demasiado grande, pudiendo llegar a colapsar la propia aplicación. Frente a las alternativas que presenta este dilema nos quedamos con la segunda opción, pues podemos hacer uso de una *ventana de datos* de longitud fija para almacenar los resultados.

Por cada magnitud física a representar, utilizaremos una ventana de datos diferente. Se trata de una lista de longitud no variable sobre la que debemos de controlar la inserción de nuevos resultados. Cuando el número de datos de la ventana es inferior a su longitud no existe

ningún problema, pues podemos ir añadiendo otros nuevos al final de la misma y mantener así el orden de producción (figura 30). Sin embargo, si excedemos el límite, lo que haremos será eliminar el de mayor antigüedad, desplazando el resto una posición hacia la izquierda (*shift*) y obteniendo así un hueco libre para añadir nueva información (figura 31). El objetivo de esto no es otro que mantener el orden de llegada para que la representación de los resultados sea lo más cómoda posible.

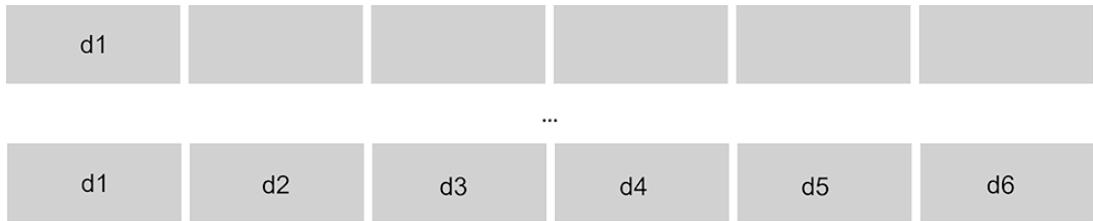


Figura 30: Cada que generamos un nuevo resultado y aún queda espacio libre, este se añade al final de la lista, respetando así el orden en el que fueron generados.

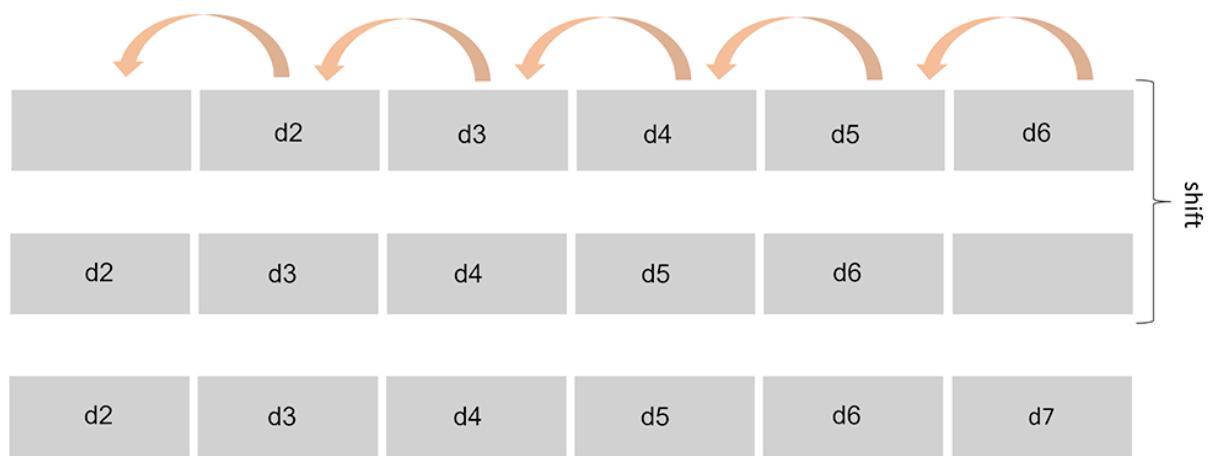


Figura 31: Cuando ya no queda hueco libre en la ventana de datos, realizamos un *shift* de esta lista, dejando así espacio libre para la inserción de nuevos resultados.

Además, las funciones del tipo *setInterval* retornan un valor identificativo que hacen referencia al temporizador de la función. Este será utilizado por el componente asociado al simulador en su etapa de destrucción para eliminar la llamada reiterativa de este proceso. Y aunque realmente esta limpieza no es necesaria (ya que solamente tendremos una función de este tipo ejecutándose al mismo tiempo), si que es de buena práctica llevarla a cabo, pues si el número de funciones *setInterval* utilizadas es mayor puede causar problemas de rendimiento

en la aplicación si no se toman este tipo de medidas.

4.1.3. Tiempo absoluto y tiempo relativo de la simulación

Por otro lado, para garantizar la correcta visualización de los datos correspondientes a cada una de las magnitudes físicas en las gráficas, tenemos que considerar dos valores de tiempo diferentes, los cuales vamos a denominar como *tiempo absoluto* y *tiempo relativo* de la simulación.

El primero de ellos, al que hemos llamado como *tiempo absoluto*, será de utilidad para la representación de cada una de las magnitudes en un mismo eje. Como es lógico, esta variable de tiempo se verá incrementada a medida que se van obteniendo nuevos resultados de la simulación, tomando solamente valor nulo cuando se cree por primera vez la instancia del circuito o bien, la simulación sea reiniciada manualmente por el usuario en la etapa de *reload*.

Sin embargo, para calcular los resultados de cada una de las magnitudes físicas que describen el circuito en un instante de tiempo determinado, utilizaremos el *tiempo relativo* de la simulación, el cuál tomará siempre valor cero cada vez que alguno de los componentes del circuito sea alterado y que se verá incrementado a la par que lo hace la variable de *tiempo absoluto*.

De esta manera, utilizando una variable para el tiempo relativo podremos obtener los resultados de la simulación utilizando las expresiones del modelado en un instante de tiempo determinado y, al mismo tiempo, dicho valor lo almacenaremos con el valor de la variable de tiempo absoluto de la simulación. Esto permite visualizar en un mismo eje los resultados obtenidos de varias simulaciones, facilitando así la comparación entre ellos. Por ejemplo, en la figura 32 podemos observar con más detalle el uso de cada una de estas variables durante la simulación de un circuito RC, en el que vemos la evolución de la carga de un condensador cuando pasamos de una fuente de 12V a una de 6V.

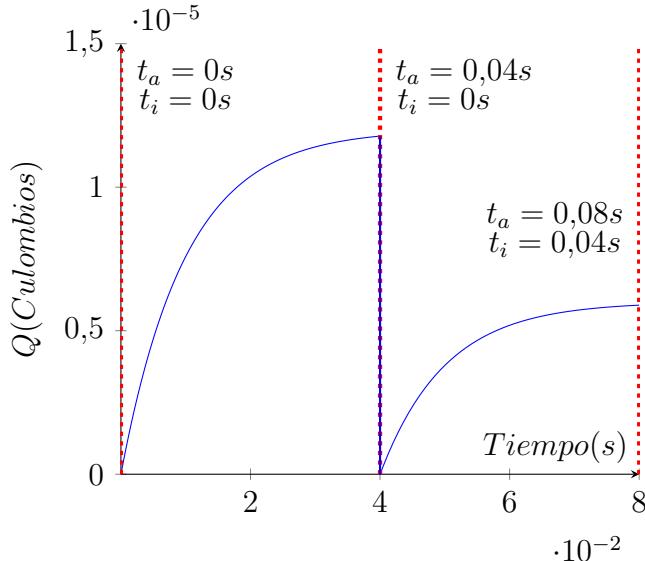


Figura 32: Ejemplo de cómo se vería la evolución de la carga de un condensador en una simulación del circuito RC, donde se aprecian los valores que toman tanto el *tiempo absoluto* (t_a) como el *tiempo relativo* (t_i) cuando pasamos de una fuente de $12V$ a una de $6V$.

4.1.4. Condiciones de parada y escala de tiempo

Otro aspecto importante a tener en cuenta es la finalización de las simulaciones de los circuitos RC y RL. Por defecto, una simulación de cualquier circuito se dará por terminada cuando este alcance su estado de equilibrio; es decir, cuando las magnitudes físicas estudiadas tomen su valor máximo o mínimo dependiendo de si estas son crecientes o decrecientes en el tiempo. Sin embargo, lo más probable es que en ocasiones no sea necesario recopilar información sobre la simulación completa, sino que basta con obtener datos hasta que se cumpla cierta condición. Estas condiciones son las siguientes:

- En el caso del circuito RC, el cuál lo estudiamos según la evolución de la carga del condensador, tomaremos como restricciones el valor o porcentaje máximo de carga del condensador.
- En el caso del circuito RL, la magnitud física que tendremos en cuenta para finalizar la simulación será la intensidad de corriente, ya sea dando su valor en *amperios* o el tanto por ciento de dicha magnitud.

- Además, en ambos casos se implementará la posibilidad de simular los circuitos durante un tiempo determinado.

Sin embargo, nos podemos encontrar con el problema presentado en la figura 33a, en la que se puede ver que la precisión de los resultados obtenidos no es muy buena. Nos encontramos entonces un margen de error entre los resultados esperados (líneas verticales y discontinuas) y los resultados obtenidos en la simulación (final de las líneas naranja y azul), cuya diferencia mostrada de color rojo es el error cometido. Para reducir este error lo que se propone es implementar un modificador de la escala de tiempo, el cuál permita al usuario establecer cuál será el intervalo de tiempo que transcurre en la generación entre cada par de datos, con el objetivo de minimizar lo máximo posible el margen de error cometido.

Esto cuenta con un inconveniente y es que, cuanto menor sea la escala de tiempo utilizada mayor será el tiempo empleado en la generación de datos y por consiguiente, mayor tiempo se necesitará en mostrar los resultados. Esto dependerá del usuario de la aplicación en utilizar una escala de tiempo que se ajuste a sus necesidades.

En la figura 33 se puede ver con más detalle cómo el error cometido se reduce al disminuir la escala de tiempo para un mismo circuito RC, utilizando como condición de parada que el porcentaje de carga no sobrepase el 63.08 % de la carga máxima.

4.2. Comparación de resultados

Por último, para comprobar que efectivamente los resultados que se están generando en nuestra aplicación coinciden o se aproximan lo suficiente a los reales, deberemos de realizar una serie de pruebas que nos permitan comparar ambos resultados. Para ello, se ha elaborado un script de *python* utilizando las librerías de *numpy* [11] y *matplotlib* [12], el cuál se encarga de generar una imagen con los resultados de una simulación cualquiera en la que se muestra información adicional de la misma, como la carga o intensidad de corriente final que se ha alcanzado en cada uno de los estados del circuito o el tiempo empleado en realizar dicha simulación.

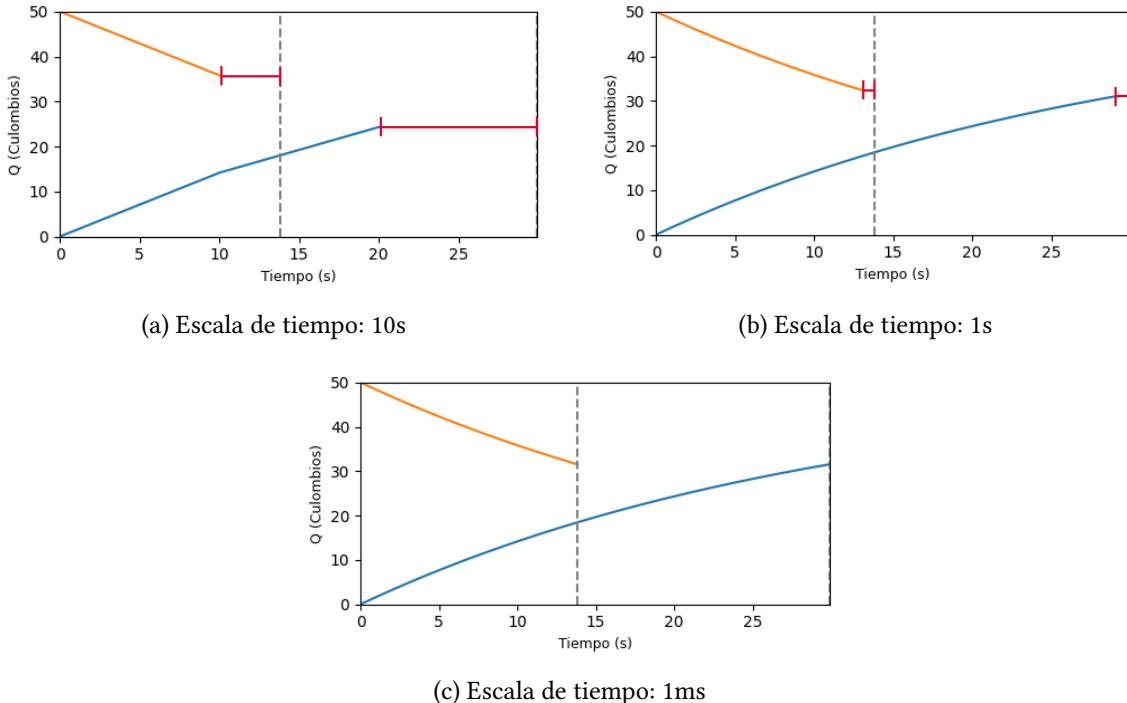


Figura 33: Simulación de la carga del condensador bajo la condición de parada de que esta sea menor que el 63.08 % de su capacidad. En *azul* se muestran los resultados obtenidos en estado de almacenamiento de energía; en *naranja* en estado de disipación de energía.

Para los casos de prueba, utilizaremos un circuito RC con una resistencia de 3Ω , una fuente de $5V$ y un condensador con una capacidad de $10F$. Por otro lado, para las pruebas del circuito RL usaremos una resistencia de 15Ω , una fuente con un valor de $20V$ y una bobina de $5H$ de inductancia. En el caso de la escala de tiempo a utilizar, se tomará aquella que genere resultados con el menor índice de error posible (escalas muy pequeñas). Además, el número de pruebas se han limitado a cuatro, en las cuales se probarán las simulaciones completas de los circuitos, así como estos mismos bajo ciertas condiciones.

4.2.1. Caso de prueba 1: carga y descarga de un condensador

El primer caso de prueba simularemos el circuito RC, sin condiciones de parada y utilizando una escala de tiempo de $1s$. Si realizamos la simulación en la aplicación, obtenemos los resultados que se muestran en las figuras 34 y 35.

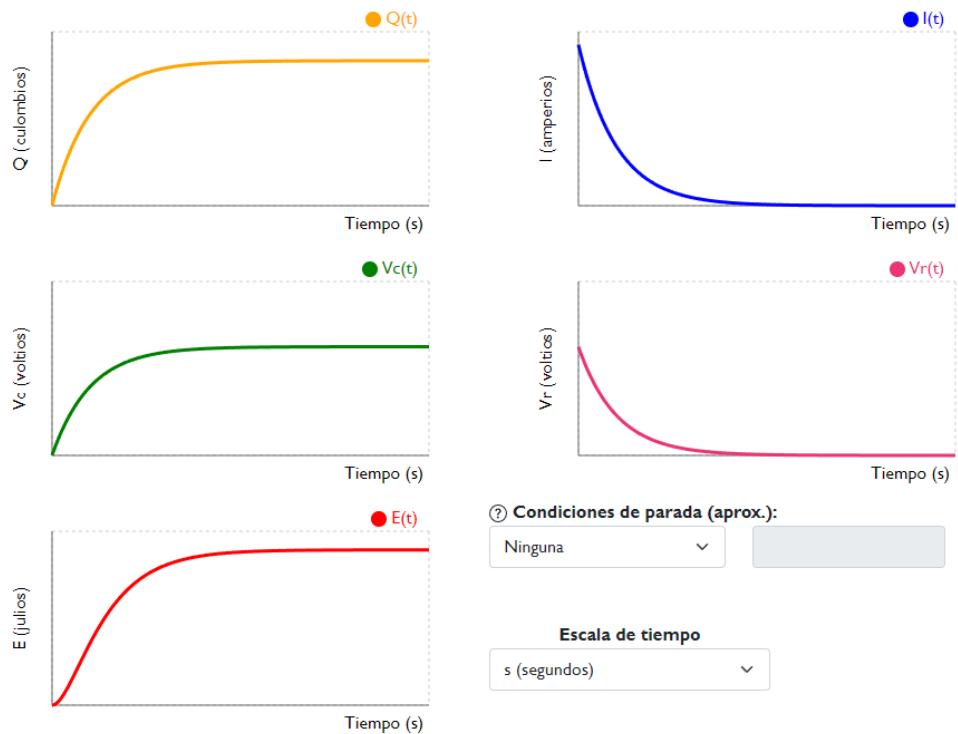


Figura 34: Caso de prueba 1: resultados de la aplicación. Estado de carga.

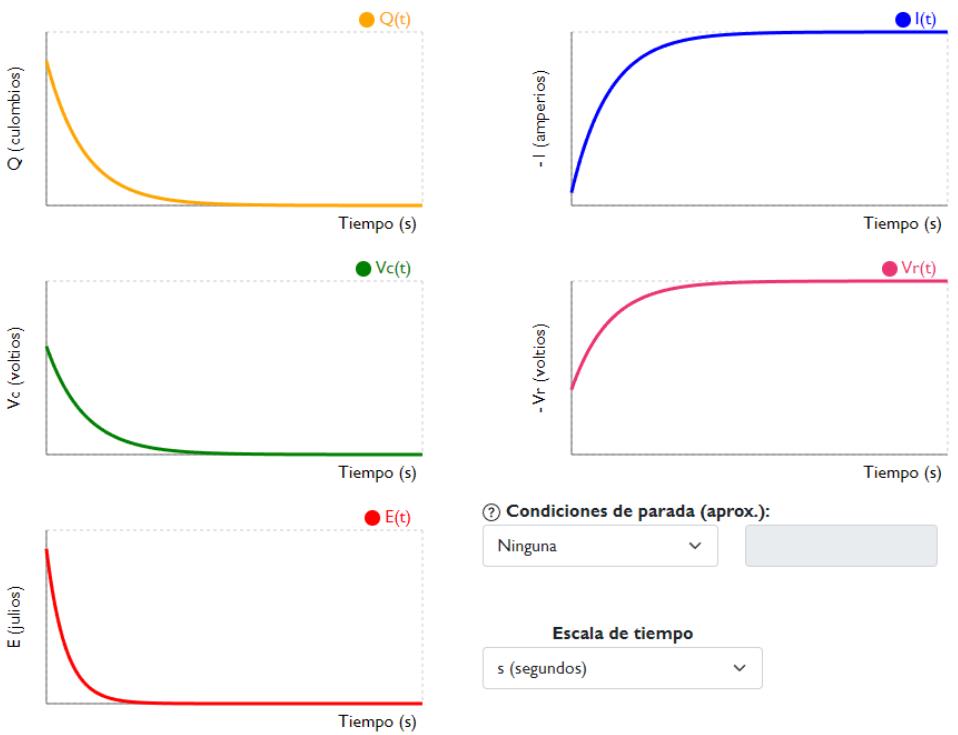


Figura 35: Caso de prueba 1: resultados de la aplicación. Estado de descarga.

El estado de equilibrio del circuito se alcanza a los 150 segundos. luego, si simulamos durante ese tiempo en el script que hemos desarrollado, obtenemos los siguientes resultados (figura 36), en el que comprobamos que efectivamente la carga máxima alcanzada es de 50 culombios.

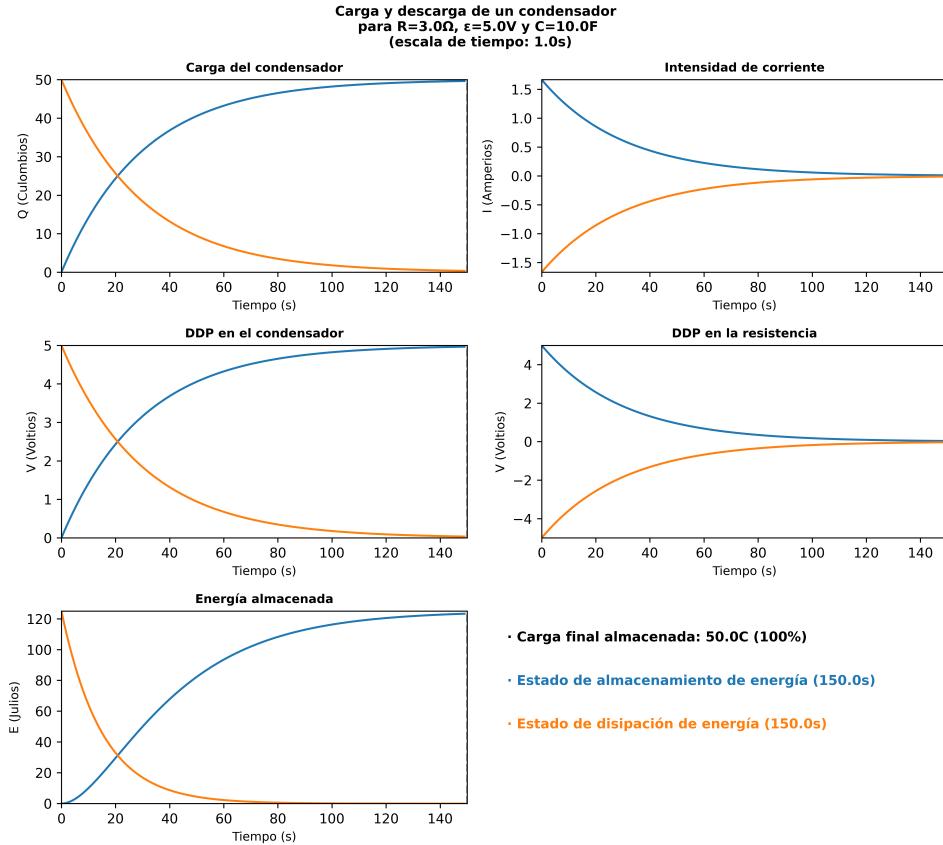


Figura 36: Caso de prueba 1: resultados del script.

4.2.2. Caso de prueba 2: carga y descarga de un condensador (con restricciones)

Para este segundo caso de prueba, utilizaremos como restricción que el porcentaje de carga del condensador no sea superior al 40 %. Los resultados obtenidos en la aplicación son los siguientes mostrados en las figuras 37 y 38; además, volvemos a utilizar los segundos como escala de tiempo.

Según los resultados de la aplicación, el valor de la carga cuando la simulación finaliza a

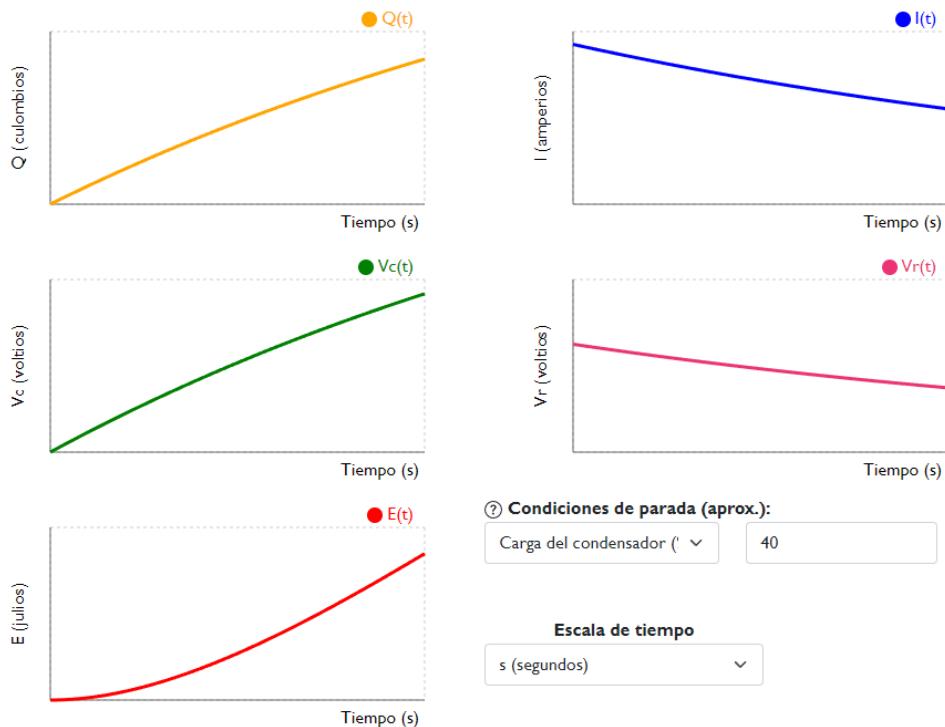


Figura 37: Caso de prueba 2: resultados de la aplicación. Estado de carga.

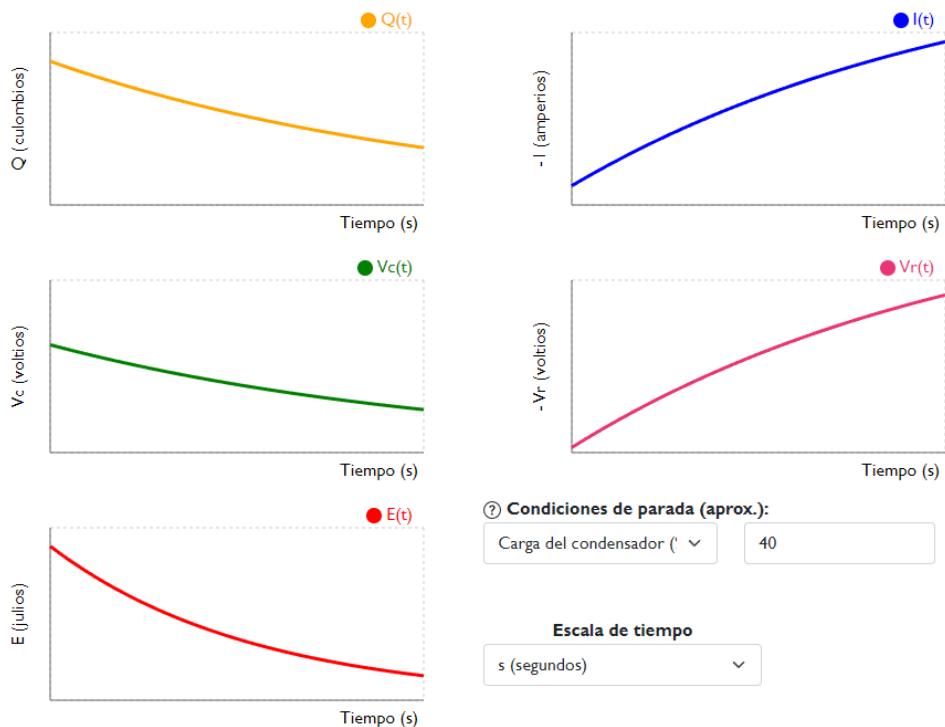


Figura 38: Caso de prueba 2: resultados de la aplicación. Estado de descarga.

los 15 segundos es ligeramente inferior a $20C$, el cuál coincide con los resultados obtenidos al utilizar el *script* de *python*, dónde obtenemos que la carga está sobre los $19,67C$. Sin embargo, en la información adicional que nos proporciona la imagen generada (figura 39) se detalla que la carga es de $20C$ y que esta se alcanza a los 15,32 segundos; y aunque los resultados se asemejan bastante, estos no llegan a coincidir, así que para obtener datos más precisos deberemos de reducir la escala de tiempo, tal y como ya se comentó anteriormente.

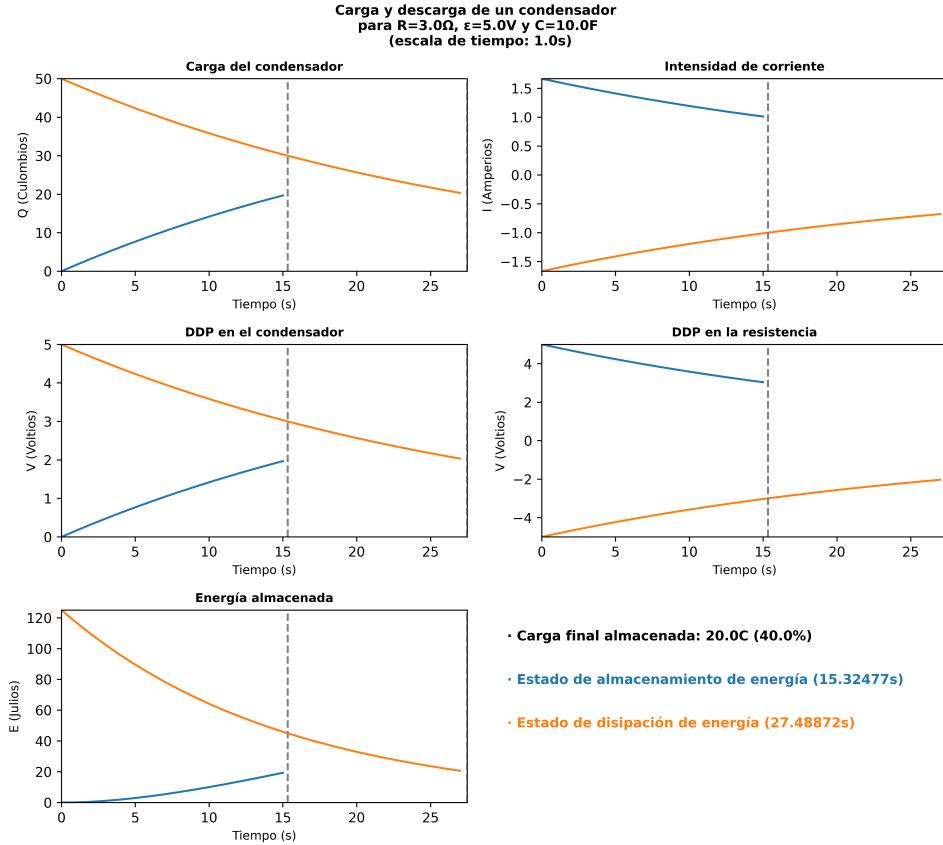


Figura 39: Caso de prueba 2: resultados del script.

4.2.3. Caso de prueba 3: carga y descarga de un inductor

Pasamos entonces al tercer caso de prueba, donde simularemos un circuito RL con los parámetros indicados anteriormente con una escala de tiempo en *milisegundos*. Mediante la aplicación, obtenemos los resultados que se muestran en las figuras 40 y 41.

El estado de equilibrio se alcanza a los 1,67 segundos de la simulación, en los que se llegan a tener una intensidad de corriente máxima de unos $1,33A$. Además, si realizamos dicha simu-

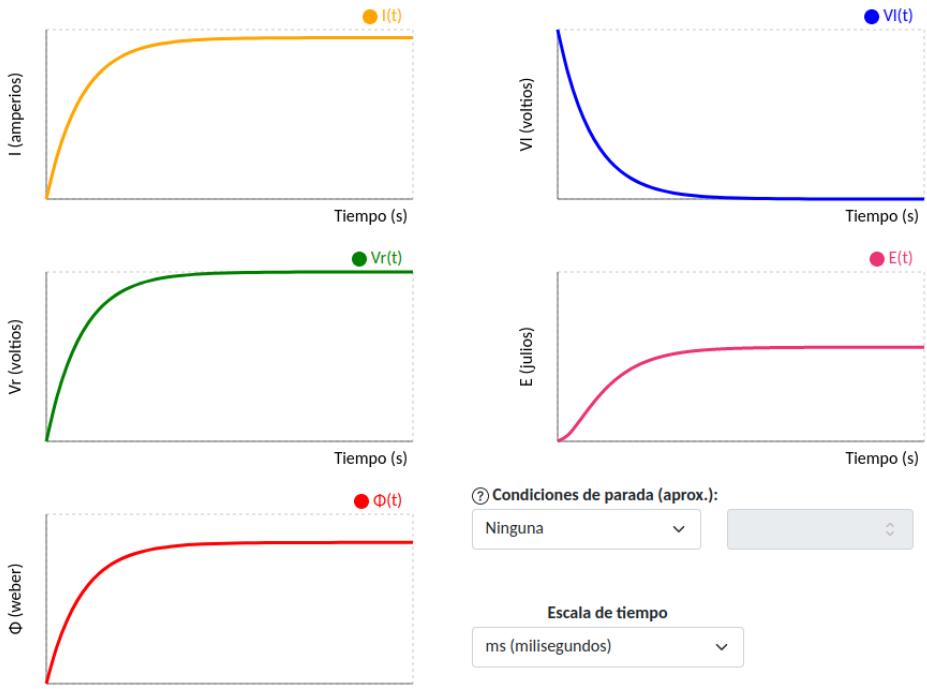


Figura 40: Caso de prueba 3: resultados de la aplicación. Estado de carga.

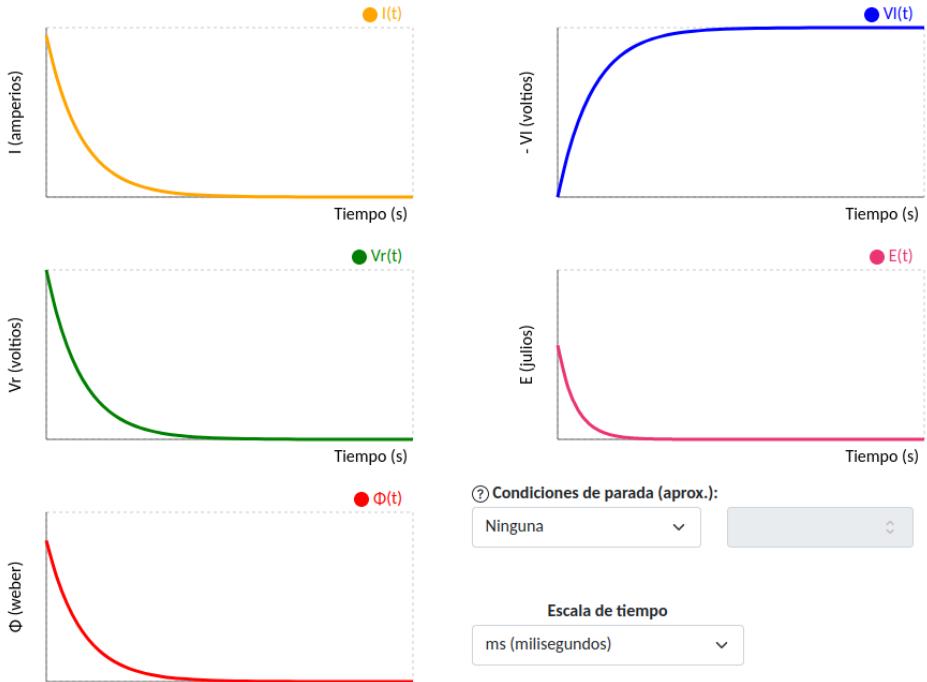


Figura 41: Caso de prueba 3: resultados de la aplicación. Estado de descarga.

lación utilizando el *script* de *python* elaborado los resultados no difieren en mucho (figura 42).

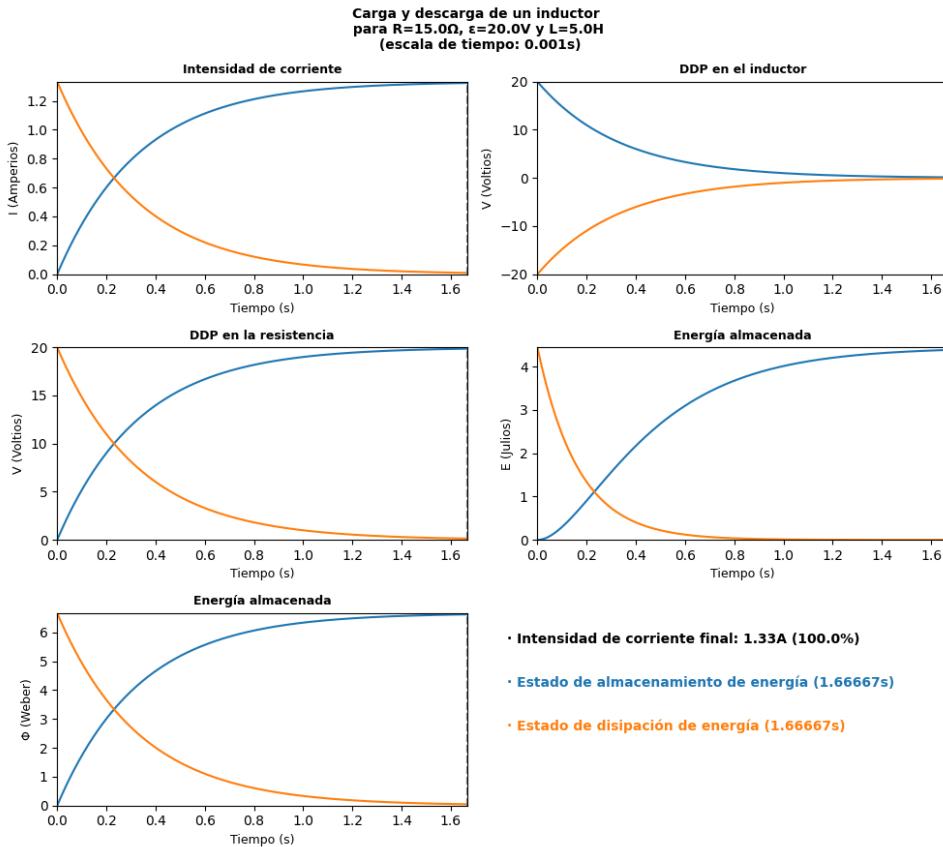


Figura 42: Caso de prueba 3: resultados del script.

4.2.4. Caso de prueba 4: carga y descarga de un inductor (con restricciones)

Por último, comprobaremos que la simulación de un circuito RL funciona adecuadamente cuando sometemos al circuito a una restricción por valor, es decir, para que la intensidad de corriente máxima no supere los $0,6A$, volviendo a utilizar la escala de tiempo de los *milisegundos*. Los resultados obtenidos por la aplicación son los que se muestran en las figuras 43 y 44.

Como se puede observar, los $0,6A$ se alcanzan (y se sobrepasa un poco) tras simular durante 0,2 segundos en estado de carga y 0,3 segundos en descarga; mientras que en los resultados generados por el *script* (figura 45), este valor se alcanza tras $0,19s$ y $0,26s$ en estado de almacenamiento y disipación de energía respectivamente. Igualmente, la causa de que los resultados difieran entre sí apenas unas centésimas es por la escala de tiempo utilizada.

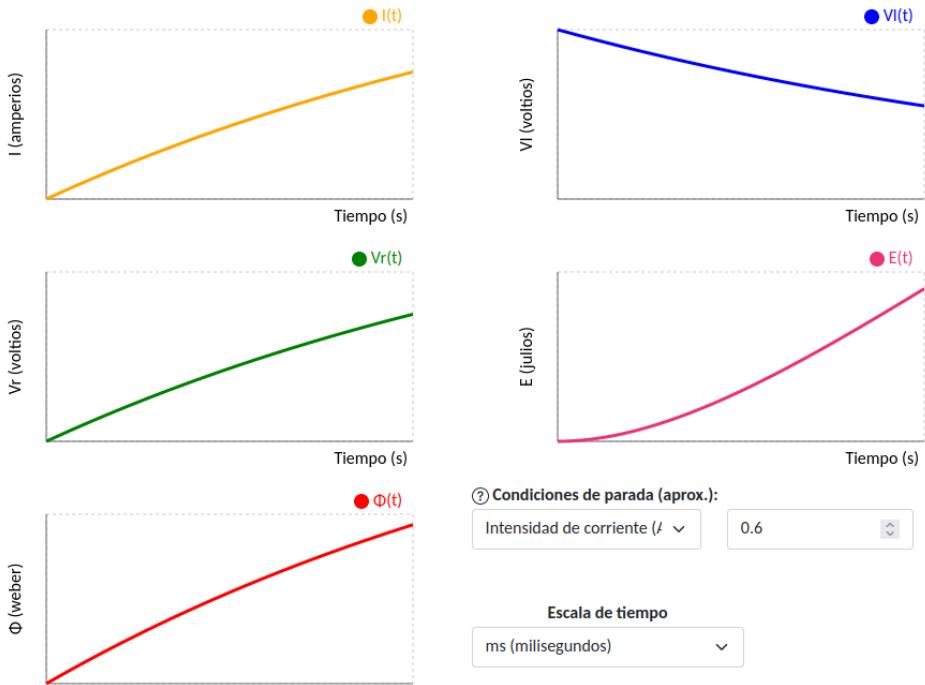


Figura 43: Caso de prueba 4 resultados de la aplicación. Estado de carga.

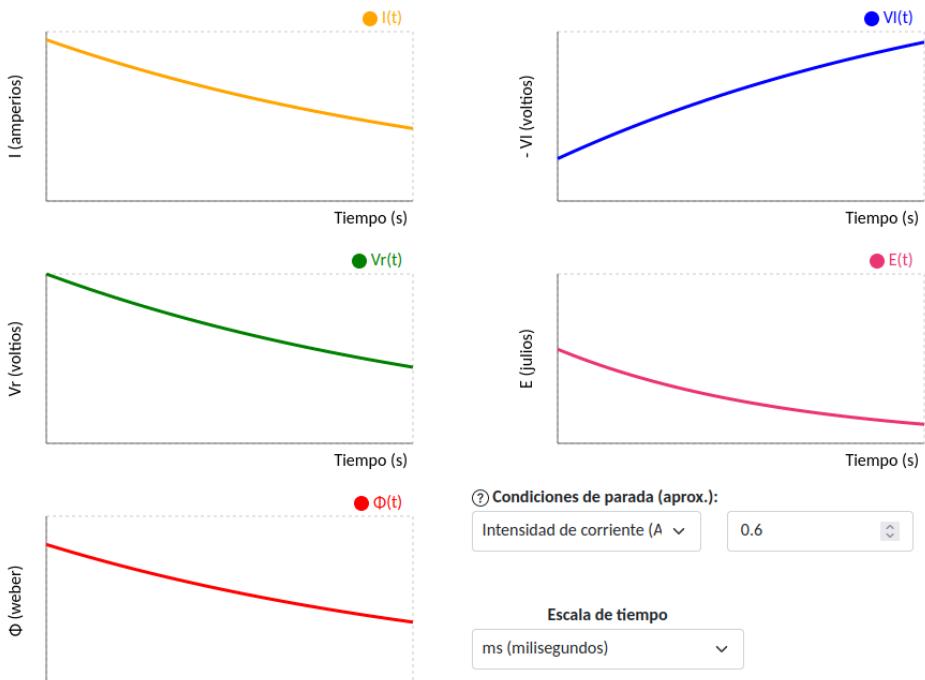


Figura 44: Caso de prueba 4: resultados de la aplicación. Estado de descarga.

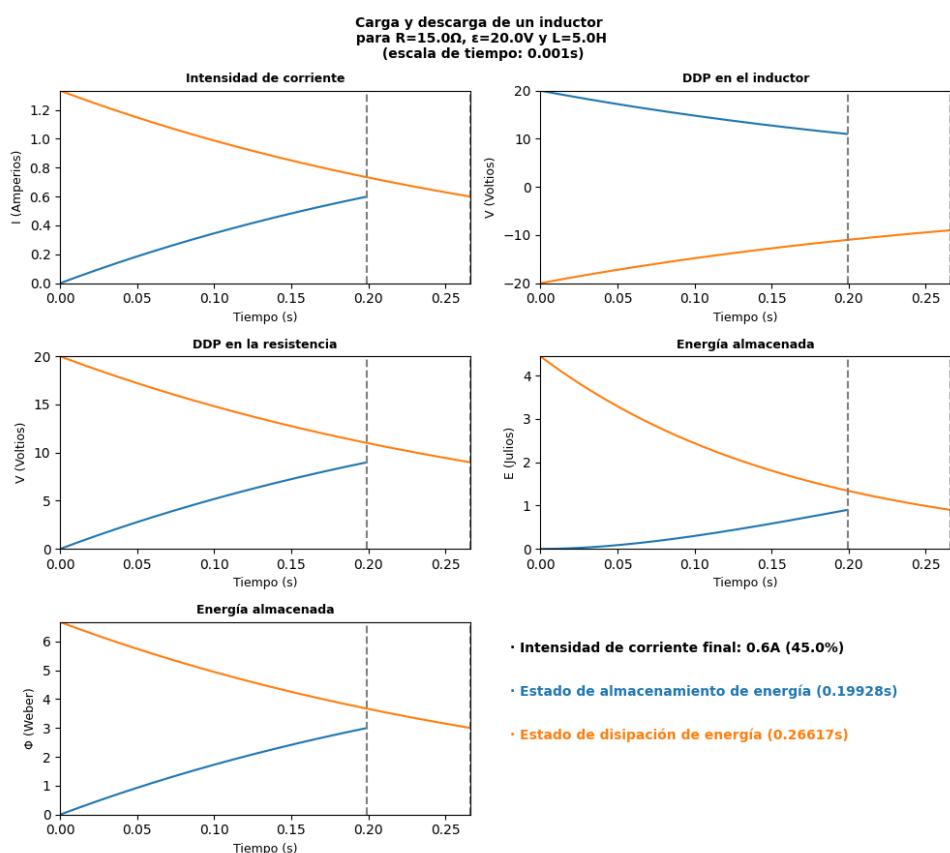


Figura 45: Caso de prueba 4: resultados del script.

5

Conclusions and Futures Lines of Research

5.1. Conclusions

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

5.2. Future lines of Research

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna

dictum turpis accumsan semper.

6

Conclusiones y Líneas Futuras

6.1. Conclusiones

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

6.2. Líneas Futuras

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Referencias

- [1] Diego Pareja Heredia. “BREVE HISTORIA DEL COMPUTADOR.” En: (2008), págs. 16-17.
- [2] Dial O. y Gambetta J. Chow J. *IBM Quantum breaks the 100 qubit processor barrier / IBM Research Blog. IBM Research.* URL: https://research.ibm.com/blog/127-qubit-quantum-processor-eagle?social_post=5922821977&linkId=140350920.
- [3] Wikipedia la enciclopedia libre. *Simulación por computadora.* URL: https://es.wikipedia.org/wiki/Simulaci%C3%B3n_por_computadora#:~:text=La%20simulaci%C3%A9n%20por%20computadora%20se,emple%C3%B3%20el%20M%C3%A9todo%20de%20Montecarlo..
- [4] NASA [@NASA]. *Visualize a mind-bending black hole. The gravity of a black hole is so intense, it distorts its surroundings like.* 2019. URL: <https://twitter.com/nasa/status/1177025023867006976>.
- [5] University of Colorado Boulder. *Interactive Simulations for Science and Math.* URL: <https://phet.colorado.edu/>.
- [6] J. Cervantes Ojeda y María del Carmen Gómez Fuentes. “Taxonomía de los modelos y metodologías de desarrollo de software más utilizados.” En: (2012).
- [7] Microsoft. *Visual Studio Code.* URL: <https://code.visualstudio.com/>.
- [8] Web Hypertext Application Technology Working Group (WHATWG). *HTML.* <https://html.spec.whatwg.org/>.
- [9] World Wide Web Consortium (W3C). *CSS: Cascading Style Sheets.* <https://www.w3.org/Style/CSS/Overview.en.html>.
- [10] MDN. *JavaScript.* <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- [11] *numpy.* <https://numpy.org/>.
- [12] *Matplotlib.* <https://matplotlib.org/>.
- [13] OpenJS Foundation. *Node.js.* <https://nodejs.org/en/>.
- [14] Jordan Walke. *React.* <https://reactjs.org/>.

- [15] Jacob Thornton Mark Otto. *Bootstrap*. <https://getbootstrap.com/docs/5.2/getting-started/introduction/>.
- [16] openstax. *Circuitos RC*. URL: <https://openstax.org/books/f%C3%ADsica-universitaria-volumen-2/pages/10-5-circuitos-rc>.
- [17] Wikipedia la enciclopedia libre. *Web 1.0*. URL: https://es.wikipedia.org/wiki/Web_1.0.
- [18] AWS. *¿Qué son las integraciones de las API?* URL: <https://aws.amazon.com/es/what-is/api/>.
- [19] NTT Data. *Aplicaciones y Applets*. URL: <https://ifgeekthen.nttdata.com/es/aplicaciones-y-applets>.
- [20] w3schools. *HTML Tags*. URL: <https://www.w3schools.com/tags/>.
- [21] MDN. *CSS selectors*. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors.
- [22] Marijn Haverbeke. *Eloquent JavaScript*. URL: <https://eloquentjavascript.net/>.
- [23] WHATWG. *DOM*. URL: <https://dom.spec.whatwg.org/>.
- [24] Jordan Walke. *React Component*. <https://reactjs.org/docs/react-component.html>.

Apéndice A

Modelado de los circuitos RC y RL

		Almacenamiento de energía		Dissipación de energía	
	Concepto	Expresión	Resolución	Expresión	Resolución
RC	Carga del condensador $q(t)$	$C\varepsilon(1 - e^{-t/RC})$	B.1.1	$C\varepsilon e^{-t/RC}$	B.2.1
	Intensidad de corriente $I(t)$	$\frac{\varepsilon}{R} \cdot e^{-t/RC}$	B.1.2	$-\frac{\varepsilon}{R} \cdot e^{-t/RC}$	B.2.2
	DDP resistencia $V_R(t)$	$\varepsilon \cdot e^{-t/RC}$	B.1.3	$-\varepsilon \cdot e^{-t/RC}$	B.2.3
	DDP condensador $V_C(t)$	$\varepsilon(1 - e^{-t/RC})$	B.1.4	$\varepsilon \cdot e^{-t/RC}$	B.2.4
	Energía almacenada $E(t)$	$\frac{1}{2}CV_C(t)^2$	B.3	*	*
RL	Intensidad de corriente $I(t)$	$\frac{\varepsilon}{R}(1 - e^{-Rt/L})$	C.1.1	$\frac{\varepsilon}{R}e^{-Rt/L}$	C.2.1
	DDP resistencia $V_R(t)$	$\varepsilon(1 - e^{-Rt/L})$	C.1.2	$\varepsilon \cdot e^{-Rt/L}$	C.2.2
	DDP inductor $V_L(t)$	$\varepsilon \cdot e^{-Rt/L}$	C.1.3	$-\varepsilon \cdot e^{-Rt/L}$	C.2.3
	Energía almacenada $E(t)$	$\frac{1}{2}LI(t)^2$	C.3	*	*
	Flujo magnético $\phi(t)$	$L \cdot I(t)$	C.4	*	*

Tabla 5: Modelado de los circuitos RC y RL

*En las columnas rellenas con este símbolo, la expresión/resolución es la misma que en la columna anterior.

Apéndice B

Resolución

matemática del

circuito RC

En este primer apéndice sobre la resolución de ecuaciones diferenciales, se presentan los pasos a seguir para obtener las expresiones que modelan el circuito RC.

B.1. Estado de almacenamiento de energía

B.1.1. Carga del condensador

Partimos de la ecuación planteada a la hora de hacer el balance energético (2). Utilizando las definiciones de capacidad del condensador y la *ley de Ohm*; y posteriormente, la definición de *intensidad de corriente*, derivamos a la siguiente expresión:

$$\varepsilon = \frac{q(t)}{C} + R \cdot I(t) = \frac{q(t)}{C} + R \frac{\partial q(t)}{\partial t}$$

Para resolver esta ecuación diferencial ordinaria, utilizaremos el método de separación de variables, así que la reescribimos de la siguiente forma:

$$\frac{\partial q(t)}{C\varepsilon - q(t)} = \frac{\partial t}{RC}$$

Partiendo de las condiciones iniciales, donde la carga del condensador es cero

$$q(0) = 0$$

, integramos entonces en ambos lados bajo estas condiciones

$$\int_0^{q(t)} \frac{\partial q(t)}{C\varepsilon - q(t)} = \int_0^t \frac{\partial t}{RC}$$

$$\left[-\ln(C\varepsilon - q(t)) \right]_0^{q(t)} = \frac{1}{RC} \left[t \right]_0^t$$

$$-\ln(C\varepsilon - q(t)) + \ln C\varepsilon = t/RC$$

Despejamos $q(t)$:

$$e^{-\ln(C\varepsilon - q(t)) + \ln C\varepsilon} = e^{\frac{t}{RC}}$$

$$\frac{1}{C\varepsilon - q(t)} C\varepsilon = e^{\frac{t}{RC}}$$

$$e^{\frac{t}{RC}} C\varepsilon = C\varepsilon - q(t)$$

Obteniendo finalmente

$$q(t) = C\varepsilon \left(1 - e^{\frac{-t}{RC}} \right) \quad (8)$$

Además, es posible hallar que la carga máxima del condensador vendrá determinada por

$$q_{max} = \lim_{t \rightarrow \infty} q(t) = C \cdot \varepsilon$$

B.1.2. Intensidad de corriente

Utilizando entonces la definición de *intensidad de corriente*

$$I(t) = \frac{\partial q(t)}{\partial t}$$

, si derivamos la expresión de carga del condensador obtenemos que

$$I(t) = \frac{\varepsilon}{R} e^{\frac{-t}{RC}} \quad (9)$$

B.1.3. DDP Resistencia

Para obtener la diferencia de potencial entre los bornes de la resistencia, hacemos uso de la *ley de Ohm*

$$V_R(t) = R \cdot I(t) \quad (10)$$

, que usando la expresión resultante en B.1.2 obtenemos

$$V_R(t) = R \cdot e^{\frac{-t}{RC}} \quad (11)$$

B.1.4. DDP Condensador

Por otro lado, la diferencia de potencial entre los terminales del condensador hacemos uso de la definición de capacidad del mismo

$$V_C(t) = \frac{q(t)}{C} \quad (12)$$

, así que usando la expresión obtenida en B.1.1 obtenemos

$$V_C(t) = \varepsilon \left(1 - e^{\frac{-t}{RC}} \right) \quad (13)$$

B.2. Estado de disipación de energía

B.2.1. Carga del condensador

Partimos de la ecuación planteada a la hora de hacer el balance energético (3). Utilizando las definiciones de capacidad de condensador y la *ley de Ohm*; y posteriormente, la definición de *intensidad de corriente*, obtenemos que:

$$0 = \frac{q(t)}{C} + R \frac{\partial q(t)}{\partial t}$$

Para resolver esta ecuación diferencial, utilizaremos el método de separación de variables, así que la reescribimos de la siguiente manera:

$$\frac{-\partial t}{RC} = \frac{\partial q(t)}{q(t)}$$

Si partimos de un condensador completamente cargado, donde la carga en el instante inicial se corresponde a

$$q(0) = q_{max} = C \cdot \varepsilon$$

,

integramos entonces en ambos lados,

$$\int_{C\varepsilon}^{q(t)} \frac{\partial q(t)}{q(t)} = \int_0^t \frac{-\partial t}{RC}$$

Resolvemos en ambos lados:

$$\left[-\ln q(t) \right]_{C\varepsilon}^{q(t)} = \frac{-1}{RC} \left[t \right]_0^t$$

$$\ln q(t) - \ln C\varepsilon = \frac{-t}{RC}$$

$$e^{\ln q(t) - \ln C\varepsilon} = e^{\frac{-t}{RC}}$$

$$q(t) \frac{1}{C\varepsilon} = e^{\frac{-t}{RC}}$$

Despejamos $q(t)$:

$$q(t) = C\varepsilon e^{\frac{-t}{RC}} \quad (14)$$

B.2.2. Intensidad de corriente

Utilizando la definición de *intensidad de corriente*

$$I(t) = \frac{\partial q(t)}{\partial t}$$

y la expresión correspondiente a la carga del condensador en estado de dispacción de energía (14), obtenemos:

$$I(t) = -\frac{\varepsilon}{R} e^{\frac{-t}{RC}} \quad (15)$$

B.2.3. Diferencia de potencial en la resistencia

Para obtener la diferencia de potencial entre los bornes de la resistencia, usamos la *ley de Ohm*

$$V_R(t) = R \cdot I(t)$$

, que usando la expresión resultante en B.2.2 obtenemos:

$$V_R(t) = -\varepsilon \cdot e^{\frac{-t}{RC}} \quad (16)$$

B.2.4. Diferencia de potencial en el condensador

Para calcular la diferencia de potencial en los terminales del condensador, usamos la definición de capacidad en un conductor

$$V_C(t) = \frac{q(t)}{C}$$

, así que usando la expresión obtenida en B.2.1 obtenemos:

$$V_C(t) = \varepsilon e^{\frac{-t}{RC}} \quad (17)$$

B.3. Energía almacenada en un condensador

Si partimos de un condensador completamente descargado, entonces la energía en ese momento también es cero.

$$E(0) = 0$$

Por definición, sabemos que la carga en el condensador es

$$q(t) = C \cdot V_C(t)$$

,

y que la intensidad de corriente que lo atraviesa viene dada entonces por la siguiente expresión

$$I(t) = C \frac{\partial V_C(t)}{\partial t}$$

Usando las definiciones de energía y potencia consumidas por el condensador

$$\partial E(t) = p(t)\partial t = V_C(t)I(t)\partial t$$

obtenemos que la energía almacenada en es la solución de la siguiente ecuación diferencial de primer orden:

$$\partial E(t) = C \cdot V_C(t)\partial V_C(t)$$

Resolvemos integrando usando las condiciones iniciales previas

$$\int_0^{E(t)} \partial E(t) = C \int_0^{V_C(t)} V_C(t)\partial V_C(t)$$

$$E(t) = C \left[\frac{V_C(t)^2}{2} \right]_0^{V_C(t)}$$

Luego, la energía almacenada en el condensador viene dada por:

$$E(t) = \frac{1}{2}CV_C(t)^2 \quad (18)$$

, donde $V_C(t)$ es la diferencia de potencial en el condensador.

Apéndice C

Resolución

matemática del

circuito RL

En este apéndice se tratará la resolución de las ecuaciones diferenciales que nos darán como resultado las ecuaciones que modelan el circuito RL y que posteriormente, serán utilizadas en la implementación de dicha simulación.

C.1. Estado de almacenamiento de energía

C.1.1. Intensidad de corriente

Partimos de la ecuación planteada por el balance energético del circuito RL (5). Utilizando la *ley de Ohm* y la definición de potencial entre los terminales de una bobina, obtenemos la siguiente expresión

$$\varepsilon = L \frac{\partial I(t)}{\partial t} + R \cdot I(t)$$

, la cual podemos reescribir de la siguiente manera:

$$\frac{\partial I(t)}{\frac{\varepsilon}{R} - I(t)} = \frac{R}{L} \partial t$$

Puesto que inicialmente supondremos que la intensidad en el circuito es cero

$$I(0) = 0$$

, integramos a ambos lados y resolvemos la ecuación diferencial mediante el método de separación de variables.

$$\int_0^{I(t)} \frac{\partial I(t)}{\frac{\varepsilon}{R} - I(t)} = \int_0^t \frac{R}{L} \partial t$$

$$\left[-\ln \frac{\varepsilon}{R} - I(t) \right]_0^{I(t)} = \frac{R}{L} [t]_0^t$$

$$-\ln \varepsilon - I(t) + \ln \frac{\varepsilon}{R} = \frac{Rt}{L}$$

$$e^{-\ln \varepsilon - I(t) + \ln \frac{\varepsilon}{R}} = e^{\frac{Rt}{L}}$$

Tras simplificar y despejar el término $I(t)$ de la expresión anterior obtenemos que

$$I(t) = \frac{\varepsilon}{R} (1 - e^{-Rt/L}) \quad (19)$$

Además, es posible hallar que la intensidad máxima inducida vendrá dada por:

$$I_{max} = \lim_{t \rightarrow \infty} I(t) = \frac{\varepsilon}{R}$$

C.1.2. Diferencia de potencial en la resistencia

Para calcular la diferencia de potencial en la resistencia, basta con hacer uso de la *Ley de Ohm*.

$$V_R(t) = \varepsilon (1 - e^{-Rt/L})$$

C.1.3. Diferencia de potencial en la bobina

Para hallar la expresión de la diferencia de potencial en los terminales de la bobina podemos hacer uso de la *Ley de Faraday-lenz*

$$\varepsilon = -L \frac{\partial I(t)}{\partial t}$$

, en la que usaremos la expresión de intensidad de corriente que hemos obtenido anteriormente en C.1.1.

$$V_L(t) = \varepsilon \cdot e^{-Rt/L}$$

C.2. Estado de disipación de energía

C.2.1. Intensidad de corriente

Partiendo del balance energético planteado para el circuito RL en estado de disipación de energía

$$0 = V_L(t) + V_R(t)$$

obtenemos la siguiente ecuación diferencial, al aplicar *Ley de Faraday-Lenz* y la *Ley de Ohm* en la expresión anterior

$$\begin{aligned} 0 &= L \frac{\partial I(t)}{\partial t} + R \cdot I(t) \\ \frac{\partial I(t)}{I(t)} &= -\frac{R}{L} \partial t \end{aligned}$$

Suponiendo, que partimos de una posición donde la intensidad de corriente es máxima (denotaremos por I_0 para simplificar), resolvemos la ecuación diferencial anterior utilizando el método de separación de variables. Integrando en ambos lados y resolviendo

$$\begin{aligned} \int_{I_0}^{I(t)} \frac{\partial I(t)}{I(t)} &= \int_0^t -\frac{R}{L} \partial t \\ \left[\ln I(t) \right]_{I_0}^{I(t)} &= \frac{-R}{L} [t]_0^t \\ \ln I(t) - \ln I_0 &= -\frac{R}{L} t \end{aligned}$$

Si despejamos el término $I(t)$ de la expresión anterior, obtendremos la siguiente ecuación que modela la intensidad de corriente en el circuito (sabiendo que $I_0 = \frac{\varepsilon}{R}$):

$$I(t) = \frac{\varepsilon}{R} \cdot e^{-Rt/L}$$

C.2.2. Diferencia de potencial en la resistencia

Usando la *Ley de Ohm*, y la expresión que modela la intensidad de corriente en estado de disipación de energía, obtenemos que

$$V_R(t) = \varepsilon \cdot e^{-Rt/L}$$

C.2.3. Diferencia de potencial en la bobina

Por otro lado, para hallar la diferencia de potencial en el inductor, hacemos uso de la *Ley de Faraday-Lenz*. Puesto que el signo nos indica el sentido de la corriente, podemos ignorarlo. La ecuación que modela este parámetro del circuito es:

$$V_L(t) = -\varepsilon \cdot e^{-Rt/L}$$

C.3. Energía almacenada

Para hallar la energía que almacena un condensador, utilizaremos la definición de potencia almacenada en el inductor, usando para ello, la *diferencia de potencial* en este dispositivo así como la intensidad de corriente eléctrica del circuito.

$$p(t) = V_L(t) \cdot I(t)$$

que esta *ddp* puede reescribirse como

$$V_L(t) = L \frac{\partial I(t)}{\partial t}$$

, luego

$$p(t) = L \frac{\partial I(t)}{\partial t} I(t)$$

Sabiendo que, la energía consumida se expresa como

$$\partial E(t) = p(t) \partial t$$

Tenemos que

$$\partial E(t) = L I(t) \partial I(t)$$

Integramos a ambos lados, tomando como instante inicial un circuito donde la corriente eléctrica que atraviesa sus componentes es nula y, por consiguiente, la energía almacenada en el inductor también lo es

$$\begin{aligned} \int_0^{E(t)} \partial E(t) &= \int_0^{I(t)} L I(t) \partial I(t) \\ E(t) &= L \left[\frac{I(t)^2}{2} \right]_0^{I(t)} \end{aligned}$$

Siendo la expresión que modela la energía almacenada en la bobina

$$E(t) = \frac{1}{2} L I(t)^2$$

C.4. Flujo magnético

Para obtener un modelo que exprese el *flujo magnético* el función del tiempo, hacemos uso de la *Ley de Faraday-Lenz*.

$$\partial\phi(t) = L\partial I(t)$$

Partimos de igual manera de un circuito por el que no circula corriente eléctrica, así que resolvemos la ecuación diferencial anterior integrando a ambos lados, resultando que, la ecuación que modela el comportamiento del flujo magnético del inductor es:

$$\phi(t) = L \cdot I(t)$$

Apéndice D

Requisitos y casos de uso

D.1. Requisitos

Los requisitos son una lista de propiedades que un sistema debe de cumplir. Existen dos tipos de requisitos: **funcionales** y **no funcionales**.

Por un lado, los *requisitos funcionales* son aquellos que describen la funcionalidad del software, proporcionando información sobre aquello que puede hacer y que es propio de él. Por otro lado, los *requisitos no funcionales* ofrecen información acerca de propiedades externas a su funcionalidad, por ejemplo, información de la escalabilidad, estabilidad o durabilidad del sistema.

A continuación, se muestra una tabla con los requisitos del software que se ha elaborado.

REF.	NOMBRE	DESCRIPCIÓN
R-1	Personalización de componentes	Al usuario se le permitirá ajustar los parámetros de cada uno de los componentes de los circuitos durante la simulación.
R-1.1	Cambio de la capacidad del condensador	En un circuito RC, el usuario podrá cambiar el valor de la capacidad del condensador.
R-1.2	Cambio del coeficiente de autoinducción de la bobina	En un circuito RL, el usuario podrá cambiar el valor del coeficiente de autoinducción del inductor.

R-1.3	Cambio del voltaje de la fuente	Los circuitos a implementar serán en corriente continua; así que se usará una pila, permitiendo así cambiar el valor del voltaje suministrado por la ella. Esta dispondrá de un interruptor interno, que permitirá alternar entre los dos estados del circuito (almacenamiento y disipación de energía).
R-1.4	Cambio valor óhmico de la resistencia	El usuario podrá cambiar el valor de la resistencia del circuito que esté simulando.
R-1.4.1	Bandas de la resistencia interactivas	Las resistencias usadas en las simulaciones serán de cuatro bandas. El color de cada una de ellas cambiará dependiendo del valor óhmico de este componente. La cuarta banda (asociada a la tolerancia) será fija, ya que se usará el valor teórico de la resistencia.
R-2	Tiempo interactivo	Durante la ejecución de la simulación, la obtención de resultados se podrá pausar o reanudar según sea conveniente; pudiendo además volver a reanudarla.
R-2.1	Escala de tiempo	Dado que en ocasiones el estado de equilibrio se obtiene demasiado rápido, las curvas características de estos circuitos de cada una de las magnitudes físicas no pueden verse correctamente. Se permite entonces ajustar esta escala de tiempo, pudiendo simular para valores en segundos, milisegundos, microsegundos, ...

R-2.2	Condiciones de parada	La simulación se podrá parar dada una condición de parada. El ajuste de los resultados obtenidos será mejor cuanto menor sea la escala de tiempo.
R-3	Gráficas de las magnitudes físicas	Se dispondrá de una serie de gráficas en la que se podrán ver la evolución de cada una de las magnitudes físicas asociadas a los circuitos.
R-3.1	Muestra de resultados	Los resultados se podrán observar directamente desde las gráficas, mostrando el instante de tiempo así como el valor de dicha magnitud.
R-4	Animación del circuito	Cada uno de los circuitos dispondrá de una animación en la que se mostrará cada uno de los componentes y el movimiento de los electrones, cuya velocidad dependerá del valor de la intensidad de corriente en el circuito.
R-5	Teoría	Dado que esto es una aplicación orientada a la educación, se implementa además una serie de secciones que permite ver teoría acerca del circuito, pulsando sobre los diferentes elementos de la simulación.

Tabla 6: Requisitos del sistema

D.2. Casos de uso

Definimos como *casos de uso* a la representación en forma de diagrama de cada uno de los *requisitos funcionales*. Estos, se hacen mediante la definición de *actores* (el usuario y el sistema), quienes interactúan con la aplicación y realizan ciertas acciones. Como se puede ver en la

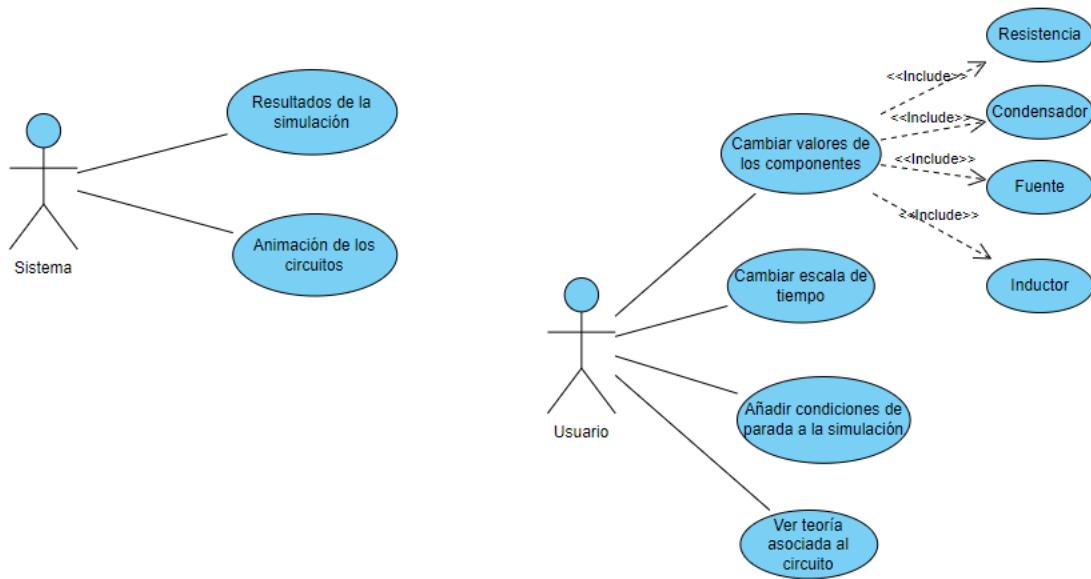


Figura 46: Diagrama de casos de uso

figura 46, el usuario puede cambiar los valores de cada uno de los componentes, cambiar la escala de tiempo de la simulación, añadir condiciones de parada o ver la teoría correspondiente a cada uno de los circuitos. Y en cuanto al sistema o aplicación, sus funciones son proporcionar los resultados de la simulación y la animación de los circuitos.

Apéndice E

Manual de instalación y de usuario

E.1. Manual de instalación del proyecto

En primer lugar, debemos de asegurarnos de tener instalado NodeJS² en la máquina en la que vaya a ser ejecutado el proyecto, ya sea un ordenador personal o en un servidor con exposición directa a internet.

Una vez la instalación del entorno de ejecución esté lista, procedemos a descargar el código fuente del proyecto. En este caso, este se encuentra guardado en un repositorio *git* privado, por lo que se necesitará permisos de colaborador para poder acceder a él.

```
$ git clone https://github.com/DavidGomez-coder/TFG.git
```

A continuación, debemos de abrir una consola de comandos y situarnos en el directorio raíz del proyecto, para instalar así todas las dependencias necesarias.

```
$ npm install
```

Cuando este proceso finalice iniciamos el proceso de la aplicación ejecutando el siguiente comando:

```
$ npm start
```

, tras el cuál se abrirá en una nueva pestaña del navegador web por defecto en la dirección <http://localhost:3000/TFG>; o si estamos utilizando un entorno de servidor, este se podrá acceder sustituyendo *localhost* por la dirección IP de la máquina en cuestión.

²<https://nodejs.org/en/download/>

E.2. Manual de usuario de la aplicación

Lo primero que vemos al iniciar la aplicación, es la siguiente interfaz presentada en la figura 47, en la que podemos distinguir claramente una división en dos de la misma, teniendo en la parte superior un panel de navegación que nos permite cambiar entre las dos simulaciones y una parte central con el contenido principal.

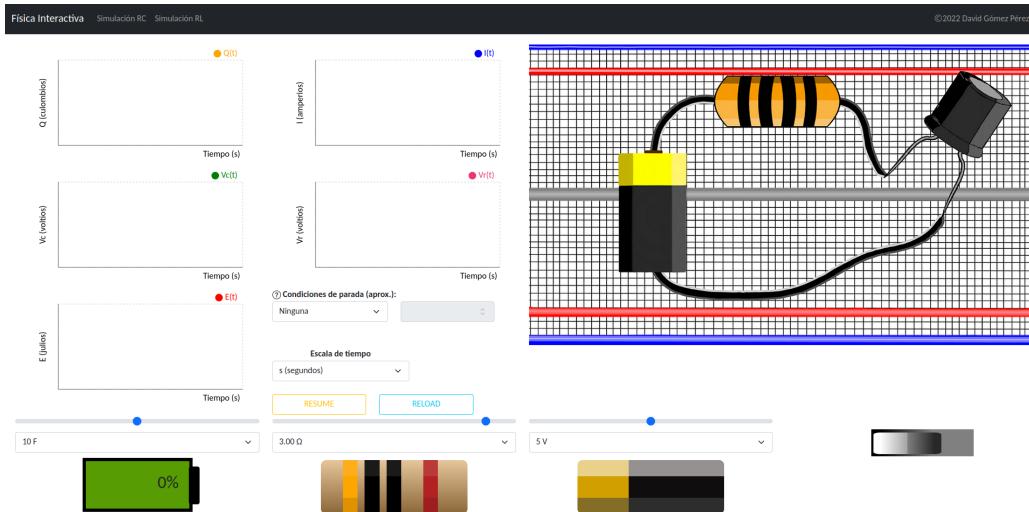


Figura 47: Interfaz de usuario de la aplicación (I).

Para poder diferenciar entre ambos circuitos y saber qué está ocurriendo, se añade una imagen de los mismos sobre el que aparecerán electrones en movimiento, ilustrando así el proceso que está ocurriendo durante la simulación (figura 48). Además, para los estados de dissipación de energía, a la fuente se le da cierta transparencia, para emular de alguna manera que esta ha sido quitada del circuito.

Junto a la representación del circuito, tenemos la sección dónde se van a mostrar los resultados de cada una de las magnitudes físicas, por las que si pasamos el cursor podremos ver con más en detalle los valores que estas toman (figura 49) .

En el centro de la pantalla, tenemos los controladores de la simulación. Por un lado en la parte superior se encuentra el controlador de las condiciones de parada, las cuáles se encuentran en un *dropdown* (lista desplegable) junto a un *input* que toma el valor de dicha condición.

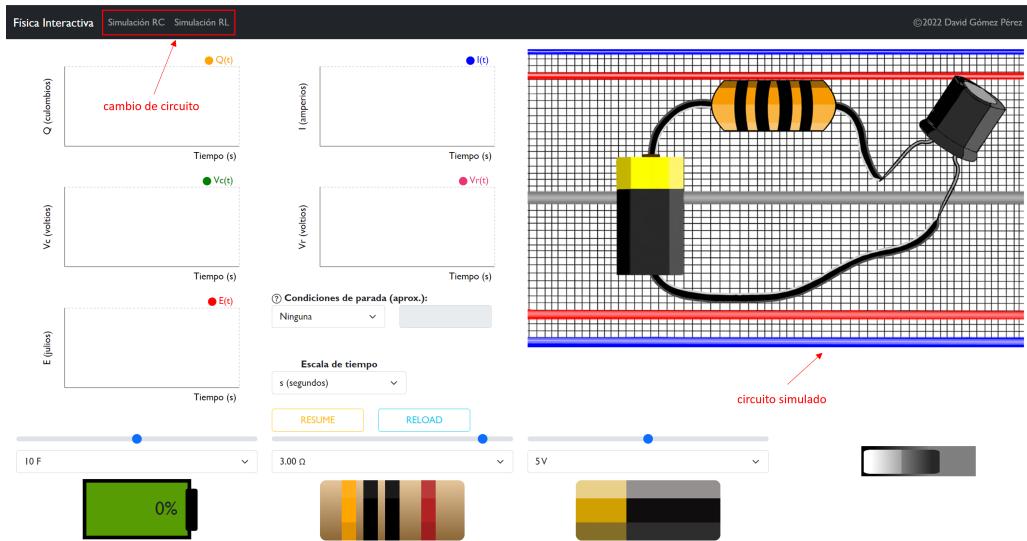


Figura 48: Interfaz de usuario de la aplicación (II).

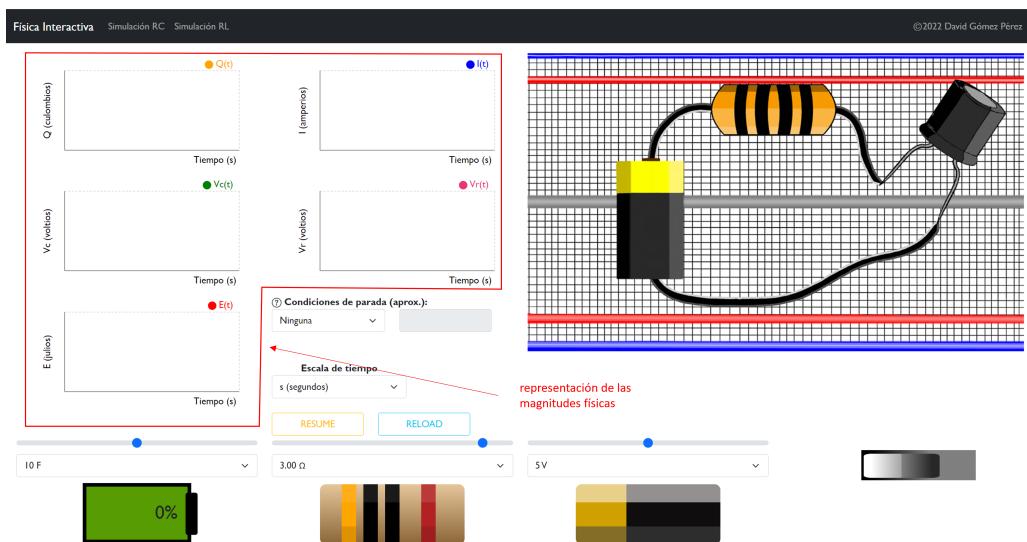
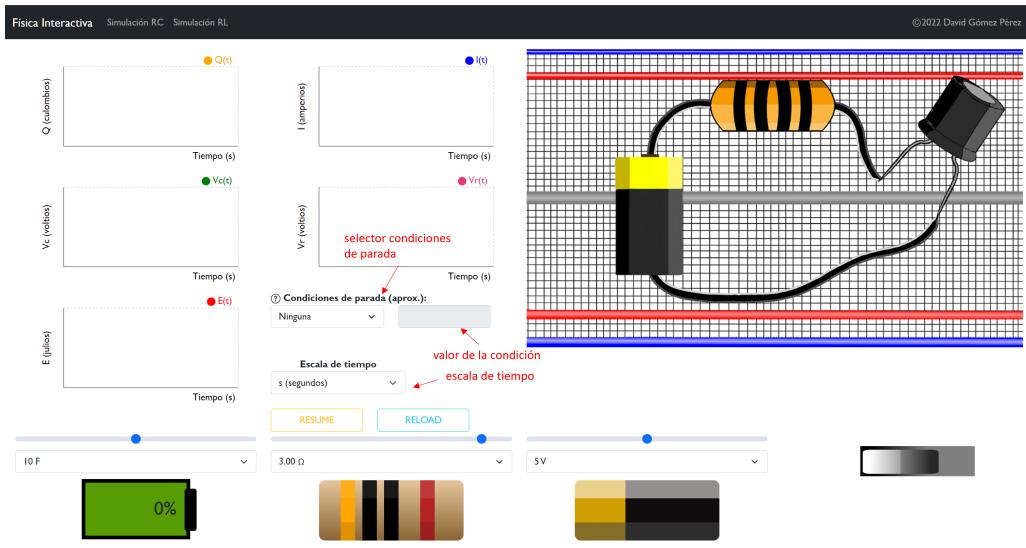


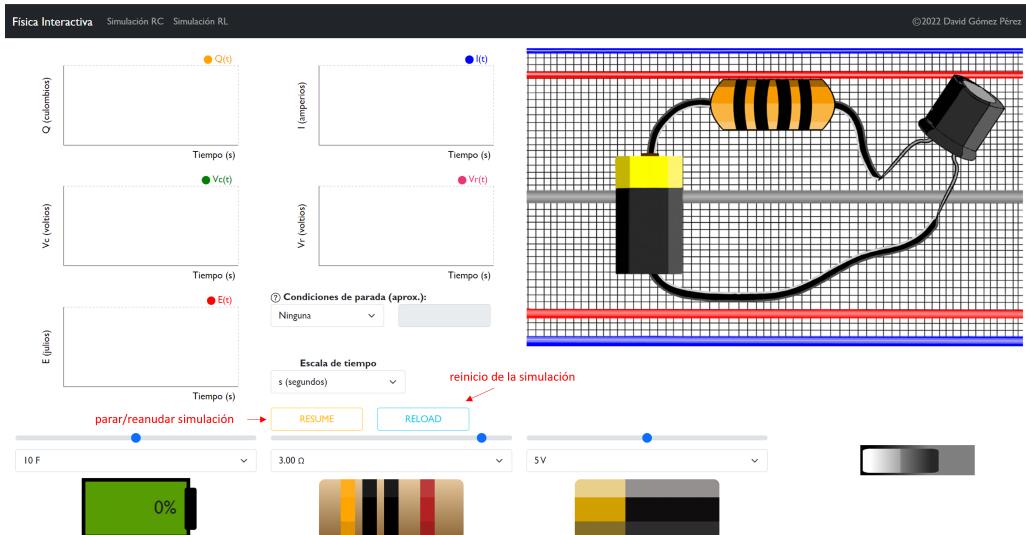
Figura 49: Interfaz de usuario de la aplicación (III).

Justo debajo, se dispone de dos botones cuyo objetivo es controlar el estado de la simulación. El botón de la derecha se encarga de reiniciar la simulación (*reload*) y el de la izquierda para o reanudar esta simulación, dependiendo del estado actual de la misma (figura 50a y 50b).

En la parte inferior tenemos los controladores que permiten cambiar los componentes del circuito. Estos se dividen en dos partes. La primera se trata de un *slider* para proporcionar el



(a)



(b)

Figura 50: Interfaz de usuario de la aplicación (IV).

valor numérico de los mismos mientras que la segunda, situada justo debajo, es un *dropdown* utilizado para cambiar el multiplicador (por ejemplo, para utilizar una resistencia en megaohmios).

Por último nos quedan los apartados teóricos con información acerca del circuito. Estos se encuentran ocultos, y para activarlos, debemos de pulsar sobre la magnitud física de la que se quiera conocer más. La información disponible es la siguiente:

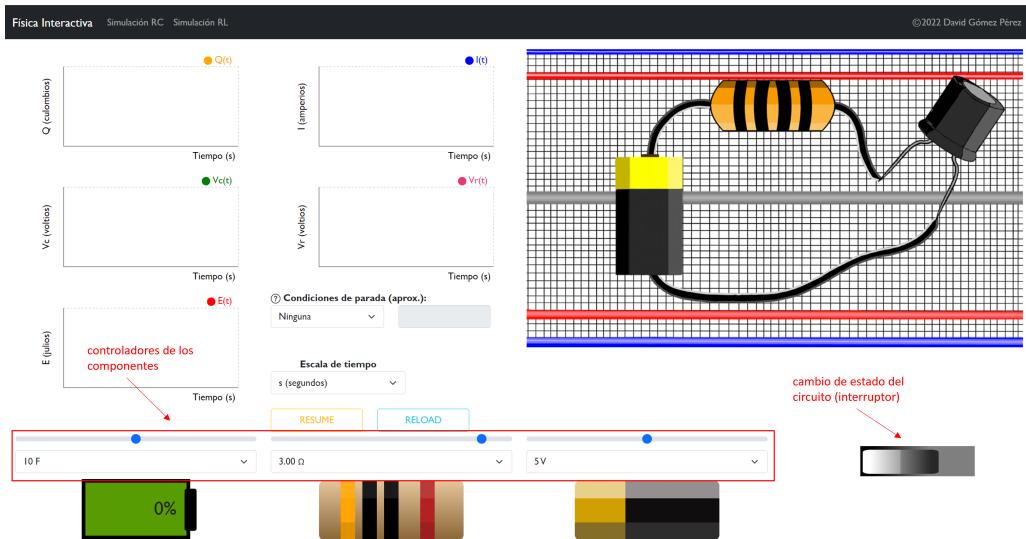


Figura 51: Interfaz de usuario de la aplicación (V).

- Para el caso del circuito RC, tenemos: *carga del condensador, intensidad de corriente, diferencia de potencial en la resistencia, diferencia de potencial en el condensador, energía almacenada, capacidad del condensador, ley de Ohm y fuerza electromotriz (F.E.M)*. Estas tres últimas se muestran haciendo click en los elementos situados bajo los controladores de los componentes.
- En el caso del circuito RL, tenemos: *intensidad de corriente, diferencia de potencial en el inductor, diferencia de potencial en la resistencia, energía almacenada, flujo magnético, fenómeno de autoinducción, ley de Ohm y fuerza electromotriz (F.E.M)*. Al igual que en el caso anterior, estas se muestran tras pulsar sobre cada uno de los elementos situados debajo de los controladores del circuito.

E.3. Manual de usuario del script *python* utilizado para las pruebas

Para utilizar el *script* debemos de abrir el directorio *simulation-tests* situado en la carpeta raíz del proyecto. El primer paso a realizar es instalar las dependencias guardadas en el fichero *requirements.txt*:

```
$ pip install -r requirements.txt
```

A continuación, podemos utilizar el script de la siguiente manera:

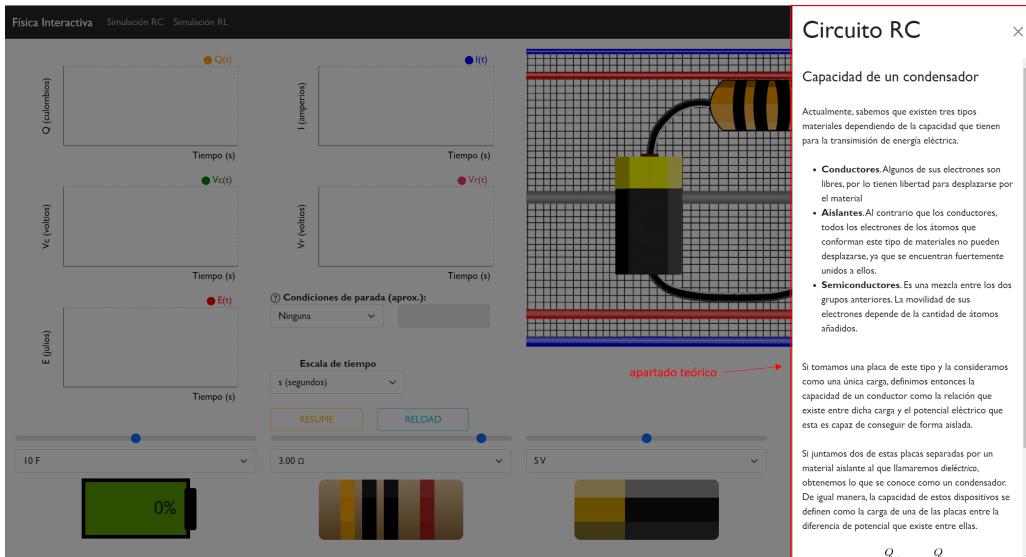


Figura 52: Interfaz de usuario de la aplicación (VI).

```
$ python3 main.py [arguments]
```

, donde los argumentos son los que siguen:

COMANDO	ABREV.	DESCRIPCIÓN
-time	-t	Establece la duración de la simulación.
-simulationType	-st	Tipo de simulación. Puede tomar los valores "RC" o "RL". Su definición es obligatoria.
-incrementValue	-inc	Escala de tiempo para el cálculo de los resultados. Por defecto su valor es de 0.001 segundos.
-capacitor	-c	Valor en Faradios de la capacidad del condensador. Solo es utilizado cuando el tipo del circuito a simular es RC. Por defecto su valor es de 0.005F.
-inductor	-i	Valor en Henrios de la inductancia de la bobina. Solo es utilizado cuando el tipo del circuito a simular es RL. Por defecto su valor es de 10H.
-resistor	-r	Valor en Ohmios de la resistencia. Por defecto su valor es de 3Ω.
-voltage	-v	Valor en Voltios de la fuente de alimentación. Por defecto su valor es de 5V.

-conditionVal	-condV	Establece la condición de parada de la simulación. Si el circuito a simular es del tipo RC, este valor hará referencia a la carga del condensador. En caso del circuito RL a la intensidad de corriente. *
-conditionPer	-condP	Establece la condición de parada de la simulación. Si el circuito a simular es del tipo RC, este valor hará referencia al porcentaje de carga del condensador (0-100). En caso del circuito RL, a la intensidad de corriente del mismo. *

Tabla 7: Argumentos válidos del *script* de python.

*En caso de que el valor de la carga o intensidad supere su valor máximo permitido (o en caso de indicar el porcentaje, este sea menor que cero o mayor que cien), se mostrará un mensaje de error indicando la carga o intensidad de corriente máximas permitidas para el circuito a simular.



UNIVERSIDAD
DE MÁLAGA | **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga