NeuraClock

Felipe Salfate N.

Agosto 2021

1 Introducción

En este trabajo se implementó un Pomodoro como una aplicación web utilizando HTML, CSS y Javascript. Este fue diseñado como una página de escritorio y fue probado en Google Chrome. En esta aplicación se debía poder establecer los temporizadores Pomodoro, de pausa corta y pausa larga. Debían ser personalizables y se tenían que mostrar como barras de progreso circular.

IMPORTANTE: Para poder testear la aplicación se hizo que el Pomodoro avanzara cada 100ms. Para cambiar esto a 1 minuto, en neuralClock.html se deben quitar los slashs en la línea 29 y comentar la línea 30.

2 Aplicación

La aplicación consta de 2 vistas dentro de la misma página. Una vista donde están los inputs, por medio de los cuales se pueden configurar los temporizadores (figura 1), y una vista donde se visualizan los temporizadores del Pomodoro (figura 2). Dentro del código se asume que cuando uno es visible el otro no lo es y viceversa.

En un principio se pensó en solo tener la vista de Pomodoro y que el tiempo de cada temporizador se pudiese cambiar en tiempo real, sin embargo se hizo de esta forma ya que así es más complicado cambiar los tiempo, reduciendo la tentación de disminuir los tiempos de trabajo. Se tiene que tener en cuenta que cuando se pasa de Pomodoro a inputs el Pomodoro se reinicia, y que al bajar el tiempo total de los temporizadores, este reduce el tiempo actual si este es mayor que el total (en caso de cambiarse en tiempo real, lo cual se puede a nivel de código).

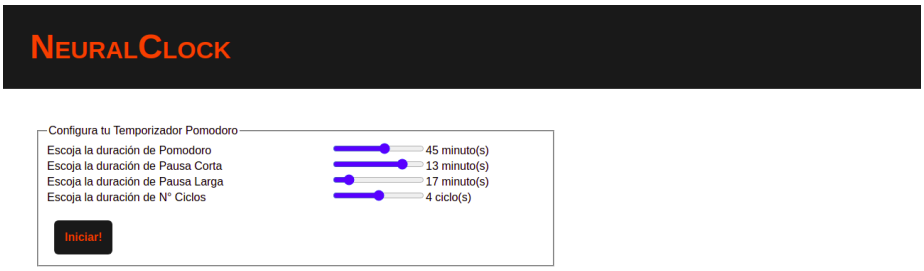

Figure 1: Vista Inputs

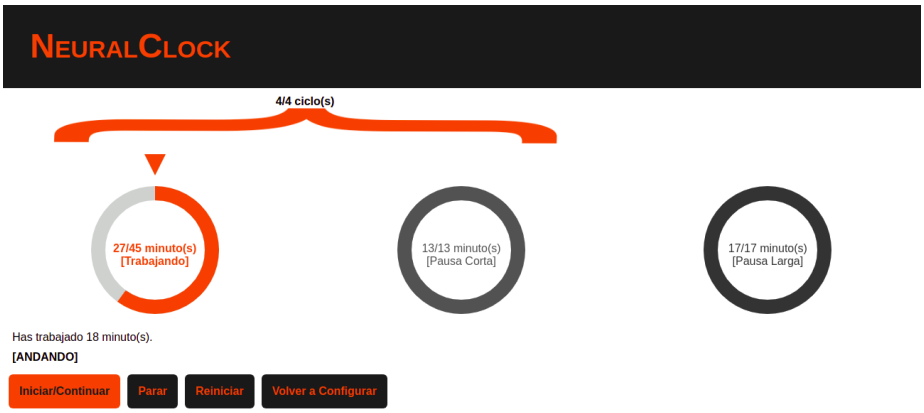

Figure 2: Vista Pomodoro.

2.1 Vista Inputs

Esta vista esta compuesta por 4 input tipo range, cada uno para cada temporizador y otro para el número de ciclos Pomodoro/pausa corta (antes de la pausa larga). El rango de estos inputs están definido en la clase Timer, que se verá más adelante. Cada temporizador, además del ciclo Pomodoro/pausa corta, están representados por una instancia de esta clase.

Para fijas estos límites se consideró que Pomodoro consta de sesiones de trabajo relativamente cortas, seguidas de una pequeña pausa, reemplazada por una pausa más larga cada cierto número de iteraciones. Estos límites pueden ser cambiadas dentro del código, pero los valores fijados son:

- Pomodoro: Entre 25 a 60 minutos.
- Pausa Corta: Entre 5 y 15 minutos.
- Pausa Larga: Entre 15 y 30 minutos.
- N° de Ciclos: Entre 3 y 5 ciclos.

Al presionar "Iniciar!" este pasa la vista Pomodoro, en donde comenzará a funcionar con los parámetros ingresados en esta vista.

2.1.1 Vista Pomodoro

En esta vista se observan los temporizadores del Pomodoro en acción. En la parte superior se muestran el ciclo Pomodoro/pausa corta en la que se está. En la parte media, de izquierda a derecha, se muestran los temporizadores de Pomodoro o trabajo, pausa corta y pausa larga. En la parte inferior se muestra cuanto tiempo a transcurrido, si el temporizador está activo, y botones de acción, con las opciones de iniciar/continuar, parar, reiniciar y volver a configurar la aplicación. Al presionar el botón de volver a configuración, se retornara a la vista de inputs.

Las barras de progreso circulares parten en el tiempo total y se van reduciendo hasta llegar a 0. Para indicar en que temporizador se está, el texto dentro de las barras se coloca en negrita, además, un triangulo del color del temporizador se posa sobre la barra. Para ver si se está o no en el ciclo Pomodoro/pausa corta el texto también se pone en negritas.

Luego de llegar a 0, las barras de Pomodoro y pausa corta se reinician al finalizar dicho ciclo, menos cuando se pasa a la pausa corta. La barra de pausa corta tampoco se reinicia después del penúltimo ciclo. Todas las barras se reinician cuando se finaliza la pausa larga.

Cada vez que la aplicación transiciona a un nuevo estado, este lanza un mensaje en la página para notificar al usuario que su tiempo de trabajo o de descanso a terminado.

3 Clases

Por temas de brevedad solo se mencionaran las clases, principales funcionalidades y el funcionamiento general de la aplicación. Si se desea saber más de cada clase estas se encuentran documentadas en el código. Las clases Javascript implementadas para esta aplicación fueron:

- **Timer:** Representa un temporizador que va de tiempo total a 0. El tiempo total puede ser cambiado, aunque como parte de los parámetros del constructor están los tiempos mínimos y máximos que el tiempo total puede tomar.
- **TimerManager:** Controla los Timer de Pomodoro, pausa corta, pausa larga y ciclos Pomodoro/pausa corta. Cada update hace avanzar el temporizador correspondiente y transiciona al siguiente State si este llega a 0.
- **States:** Grupo de Clases que representan el estado del Pomodoro. Encargados de la funcionalidad de TimerManager. Controlan el avance, activación, desactivación y reinicio de los Timers, además de la transición de estados de TimerManager.
  - **NullState:** Estado nulo. Actua como placeholder en TimerManager.
  - S**tartState:** Estado inicial. Reinicia y activa/desactiva los Timers de TimerManager al inicio del Pomodoro. Transiciona a PomodoroState.
  - **PomodoroState:** Estado "en Pomodoro". Avanza Timer de Pomodoro y ciclo si transiciona a LongPauseState. Dependiendo del ciclo en el que se esté transiciona a ShortPauseState o LongPauseState.
  - **ShortPauseState:** Estado "en pausa corta". Avanza Timer de pausa corta y ciclo. Transiciona a Pomodoro.
  - **LongPauseState:** Estado "en pausa larga". Transiciona a StartState.
- **TimeDisplayer:** Displayer básico de un temporizador. Simplemente muestra el temporizador como texto en forma de "tiempo actual/tiempo total". Recive como parametro el id de una division en la cual renderizará el displayer.

- **CircularTimeDisplayer:** Subclase de TimeDisplayer. Muestra el temporizador como una barra de progreso circular.
- **DisplayerRenderer:** Clase estática que crea las vistas de input y Pomodoro en la página.
- **NeuralClock:** Clase general que contiene y controla a todas las otras clases.

En términos generales la aplicación consta de 4 Timers que representan cada temporizador mencionado. Estos están asociados a un TimerDisplayer o CircularTimerDisplayer, dependiendo si es un temporizador normal o de ciclo Pomodoro/pausa corta, por medio del cual son representados gráficamente. TimerManager contiene estos 4 Timers y los controla por medio de los States, cuyas responsabilidades ya fueron mencionadas. Finalmente todas estas clases (menos los States) son creadas y contenidas en NeuralClock, el cual además llama las clases estaticas de DisplayerRenderer para generar las vistas.

Una vez la página carga, se llama la funcion load() de NeuralClock para generar las vistas en la división cuyo id es ingresado en su constructor. Por medio de setInterval(), fijado para funcionar cada minuto, se hace avanzar el Timer respectivo por una secuencia de acciones que es más o menos así: $NeuralClock.update() \rightarrow TimerManager.update() \rightarrow State.update() \rightarrow Timer.tick()$. Esto ocurre siempre y cuando NeuralClock allá sido cargado y su parámetro running sea verdadero.

Como se mencionó están las opciones de parar, continuar y reiniciar, en donde cada botón respectivo llaman a métodos de control de NeuralClock. Actualmente estos botones llaman directamente a la variable "neuralClock", independiente de si exista o no. El submit de la vista de inputs llama la función init() de NeuralClock, el cual hace que lea valores de los respectivos inputs y actualice los Timers.

4 Mejoras

La implementación realizada cumple con las funciones y requisitos especificados. De todas formas se mencionan las siguientes mejoras que se podrían hacer al código, que por temas de tiempo no pudieron ser realizadas. También se mencionan características que podría ser interesante agregar a la aplicación (si fuera un proyecto real):

- Convertir DisplayerRenderer en una clase de instancias, y hacer que una instancia de esta clase contenga a otra de NeuralClock, siguiendo un patrón MVC. De esta forma NeuralClock solo estaría encargado de la

lógica y DisplayerRenderer de la parte gráfica. Incluso puede que NeuralClock se vuelva redundante y solo se tenga TimerManager. Del mismo modo al crear los botones de control no se tendría que esperar a que una variable especifica existiese (var neuralClock = new NeuralClock(divId);).

- Remover la dependencia circular entre State y TimerManager, pero manteniendo la responsabilidad de State de transicionar TimerManager. Para esto quizas se pueda implementar un observer (si es posible en Javscript).
- Crear Interfaz o clase abstracta de State. De esta forma la adquisición de referencias y la función de alerta estarían definidas una sola vez.
- Si es posible hacer que las alertas cambien a la pestaña donde está la aplicación. Así se avisaría al usuario de la rotación del Pomodoro de forma más efectiva.
- Hacer que la barra de proceso llegue a 0 antes de lanzar el mensaje de alerta.
- Añadir la opción para avanzar al siguiente ciclo.
- Añadir la opción de tener un registro de trabajo local. Tener la posibilidad de poner texto (nota) y luego de presionar un botón, lo cual haría que el tiempo actual más el texto se guardara en una lista, mientras que el tiempo actual se reiniciaría. Ej: (en forma de lista) Tarea 1 - 20 minutos, Tarea 2 - 52 minutos, etc...