MANUAL DE DESAFÍOS – Módulo I

Programación Backend

CODERHOUSE

¡Bienvenidas y bienvenidos!

Qué bueno encontrarlos/as en este espacio, el cual hemos creado para que puedan conseguir en un mismo lugar, de manera rápida y ágil, todos los desafíos entregables que plantea el módulo I.

A continuación presentamos el sistema de entregas de los cursos de Coder. Luego, en un tablero, podrán ver **las 18 clases establecidas en el módulo I, además de la clase 0**, marcando con el ícono correspondiente las clases que sí tienen entregables.

De esta forma podrás tener un pantallazo del cronograma de clases y los desafíos que deberás completar.

CODERHOUSE

Sistema de entregas

Desafíos entregables

Tienen una vigencia de 7 días, es decir, a partir de la fecha (de la clase) en la que se lanza el desafío, empezarán a correr 7 días continuos, para que puedas cargar tu desafío en la plataforma.

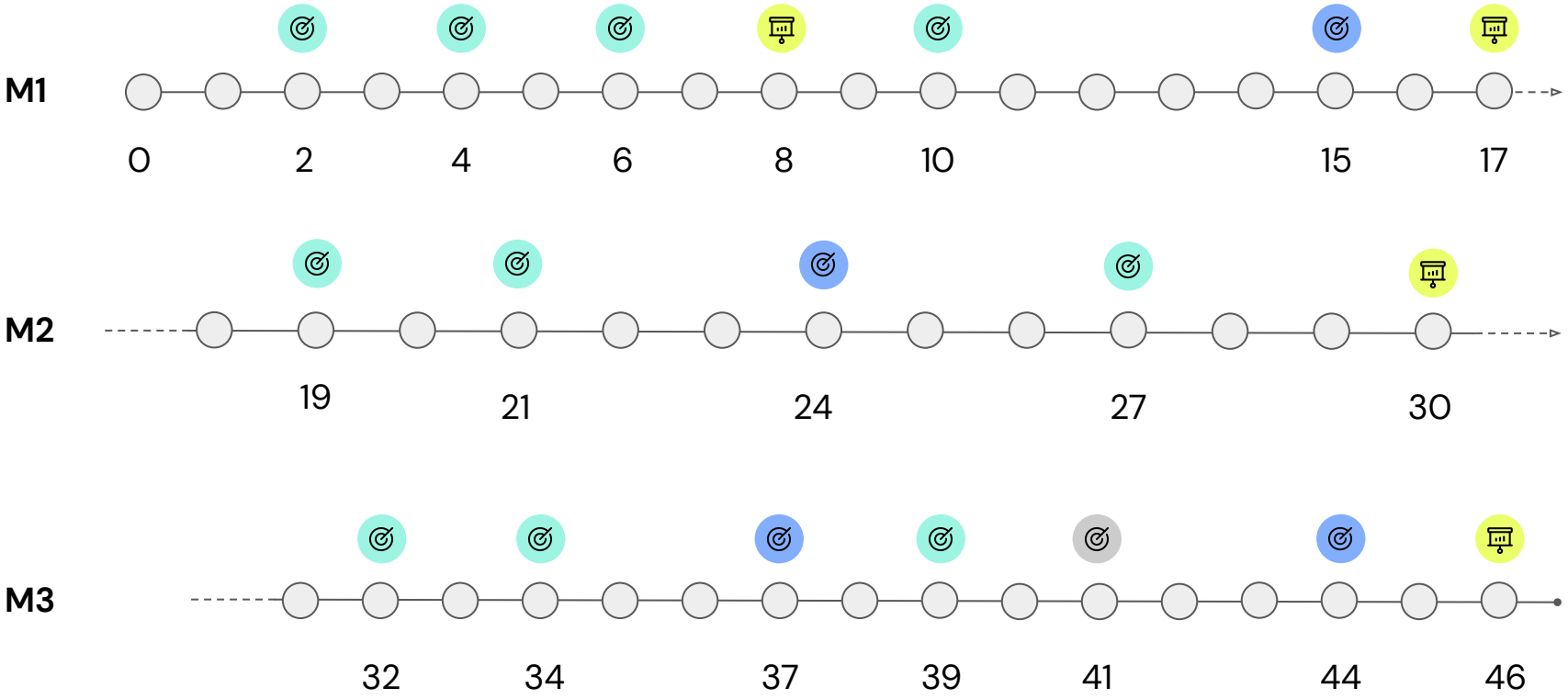
Pre-entrega del PF

También tiene una vigencia o duración de 7 días antes de que el botón de “entrega” se deshabilite. Por este motivo te recomendamos estar al día con todas las actividades planteadas.

Proyecto Final

A diferencia de los anteriores puntos, cuenta con un lapso de 20 días continuos luego de finalizada la última clase (**n° 47**). Posterior a ese tiempo, el botón de la entrega quedará inhabilitado y no será posible entregarlo o recibirlo por otros medios.

GRILLA DE ENTREGAS



MÓDULO I - CLASE 02

Nuevas funcionalidades de los lenguajes ECMAScript

CODERHOUSE

Clases con ECMAScript y ECMAScript avanzado

CODERHOUSE

DESAFÍO ENTREGABLE

Clases con ECMAScript y ECMAScript avanzado

Consigna

- ✓ Realizar una clase “ProductManager” que gestione un conjunto de productos.

Te acercamos esta ayuda 👉

Hands on lab sobre creación de clases (clase 1)

Aspectos a incluir

- ✓ Debe crearse desde su constructor con el elemento products, el cual será un arreglo vacío.
- ✓ Cada producto que gestione debe contar con las propiedades:
  - title (nombre del producto)
  - description (descripción del producto)
  - price (precio)
  - thumbnail (ruta de imagen)
  - code (código identificador)
  - stock (número de piezas disponibles)

CODERHOUSE

DESAFÍO ENTREGABLE

Aspectos a incluir

- ✓ Debe contar con un método “addProduct” el cual agregará un producto al arreglo de productos inicial.
  - Validar que no se repita el campo “code” y que todos los campos sean obligatorios
  - Al agregarlo, debe crearse con un id autoincrementable
- ✓ Debe contar con un método “getProducts” el cual debe devolver el arreglo con todos los productos creados hasta ese momento
- ✓ Debe contar con un método “getProductById” el cual debe buscar en el arreglo el producto que coincida con el id
  - En caso de no coincidir ningún id, mostrar en consola un error “Not found”

Formato del entregable

- ✓ Archivo de Javascript listo para ejecutarse desde node.

Proceso de testing de este entregable ✅

CODERHOUSE

MÓDULO I - CLASE 04

Manejo de archivos en Javascript

CODERHOUSE

Manejo de archivos

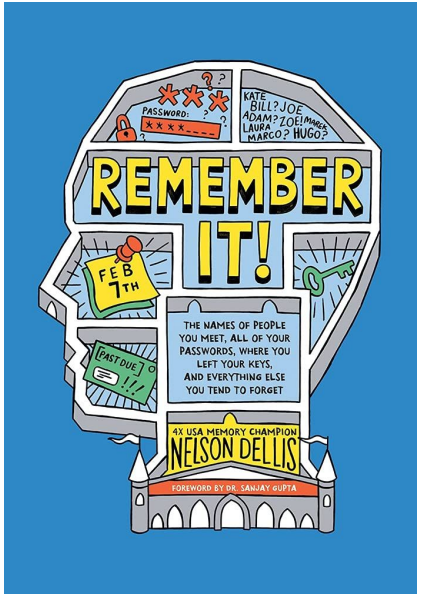
Agregamos fileSystem para cambiar el modelo de persistencia actual

CODERHOUSE

Recordemos...

Basado en el entregable de la clase 2

- Estructuramos nuestra primera clase
- Agregamos los métodos necesarios a nuestra clase para trabajar con un arreglo de productos



Ahora... agregamos fileSystem para cambiar el modelo de persistencia actual

CODERHOUSE

DESAFÍO ENTREGABLE

Manejo de archivos

Consigna

- ✓ Realizar una clase de nombre “ProductManager”, el cual permitirá trabajar con múltiples productos. Éste debe poder agregar, consultar, modificar y eliminar un producto y manejarlo en persistencia de archivos (basado en entregable 1).

Aspectos a incluir

- ✓ La clase debe contar con una variable this.path, el cual se inicializará desde el constructor y debe recibir la ruta a trabajar desde el momento de generar su instancia.
- ✓ Debe guardar objetos con el siguiente formato:
  - id (se debe incrementar automáticamente, no enviarse desde el cuerpo)
  - title (nombre del producto)
  - description (descripción del producto)
  - price (precio)
  - thumbnail (ruta de imagen)
  - code (código identificador)
  - stock (número de piezas disponibles)

DESAFÍO ENTREGABLE

Aspectos a incluir

- ✓ Debe tener un método addProduct el cual debe recibir un objeto con el formato previamente especificado, asignarle un id autoincrementable y guardarlo en el arreglo (recuerda siempre guardarlo como un array en el archivo).
- ✓ Debe tener un método getProducts, el cual debe leer el archivo de productos y devolver todos los productos en formato de arreglo.
- ✓ Debe tener un método getProductById, el cual debe recibir un id, y tras leer el archivo, debe buscar el producto con el id especificado y devolverlo en formato **objeto**
- ✓ Debe tener un método updateProduct, el cual debe recibir el id del producto a actualizar, así también como el campo a actualizar (puede ser el objeto completo, como en una DB), y debe actualizar el producto que tenga ese id en el archivo. **NO DEBE BORRARSE SU ID**
- ✓ Debe tener un método deleteProduct, el cual debe recibir un id y debe eliminar el producto que tenga ese id en el archivo.

Formato del entregable

- ✓ Archivo de javascript con el nombre ProductManager.js

Proceso de testing de este entregable ✅

CODERHOUSE

MÓDULO I - CLASE 06

Servidores web

CODERHOUSE

Servidor con express

CODERHOUSE

Recordemos

Clase 4: Manejo de archivos

- Tomamos la clase ProductManager la cual tenía persistencia en memoria
- Desarrollamos un ProductManager basado en archivos con métodos asíncronos
- Conseguimos cambiar la persistencia de memoria a archivos

CODERHOUSE

DESAFÍO ENTREGABLE

Servidor con express

Consigna

- ✓ Desarrollar un servidor basado en express donde podamos hacer consultas a nuestro archivo de productos.

Aspectos a incluir

- ✓ Se deberá utilizar la clase ProductManager que actualmente utilizamos con persistencia de archivos.
- ✓ Desarrollar un servidor express que, en su archivo app.js importe al archivo de ProductManager que actualmente tenemos.

Aspectos a incluir

- ✓ El servidor debe contar con los siguientes endpoints:
  - ruta ‘/products’, la cual debe leer el archivo de productos y devolverlos dentro de un objeto agregar el soporte para recibir por query param el valor ?limit= el cual recibirá un límite de resultados.
- Si no se recibe query de límite, se devolverán todos los productos
- Si se recibe un límite, sólo devolver el número de productos solicitados

Servidor con express

- ruta ‘/products/:pid’, la cual debe recibir por req.params el pid (product Id), y devolver sólo el producto solicitado, en lugar de todos los productos.

Formato del entregable

- ✓ Link al repositorio de Github con el proyecto completo, el cual debe incluir:
  - carpeta src con app.js dentro y tu ProductManager dentro.
  - package.json con la info del proyecto.
  - NO INCLUIR LOS node_modules generados.

Testing de este entregable

MÓDULO I - CLASE 08

Router y Multer

CODERHOUSE

Primera pre-entrega de tu Proyecto final

Se desarrollará un servidor que contenga los endpoints y servicios necesarios para gestionar los productos y carritos de compra en el e-commerce

CODERHOUSE

Primera entrega

Se debe entregar

Desarrollar el servidor basado en Node.JS y express, que escuche en el puerto 8080 y disponga de dos grupos de rutas: /products y /carts. Dichos endpoints estarán implementados con el router de express, con las siguientes especificaciones:

Para el manejo de productos, el cual tendrá su router en /api/products/ , configurar las siguientes rutas:

- ✓ La ruta raíz GET / deberá listar todos los productos de la base.
- ✓ La ruta GET /:pid deberá traer sólo el producto con el id proporcionado

Primera entrega

- ✓ La ruta raíz POST / deberá agregar un nuevo producto con los campos:
  - id: Number/String (A tu elección, el id NO se manda desde body, se autogenera como lo hemos visto desde los primeros entregables, asegurando que NUNCA se repetirán los ids en el archivo.
  - title:String,
  - description:String
  - code:String
  - price:Number
  - status:Boolean
  - stock:Number
  - category:String
  - thumbnails:Array de Strings que contengan las rutas donde están almacenadas las imágenes referentes a dicho producto

Primera entrega

- ✓ La ruta PUT /:pid deberá tomar un producto y actualizarlo por los campos enviados desde body. NUNCA se debe actualizar o eliminar el id al momento de hacer dicha actualización.
- ✓ La ruta DELETE /:pid deberá eliminar el producto con el pid indicado.

Para el carrito, el cual tendrá su router en /api/carts/, configurar dos rutas:

- ✓ La ruta raíz POST / deberá crear un nuevo carrito con la siguiente estructura:
  - Id:Number/String (A tu elección, de igual manera como con los productos, debes asegurar que nunca se dupliquen los ids y que este se autogenere).
  - products: Array que contendrá objetos que representen cada producto

Primera entrega

- ✓ La ruta GET /:cid deberá listar los productos que pertenezcan al carrito con el parámetro cid proporcionados.
- ✓ La ruta POST /:cid/product/:pid deberá agregar el producto al arreglo “products” del carrito seleccionado, agregándose como un objeto bajo el siguiente formato:
  - product: SÓLO DEBE CONTENER EL ID DEL PRODUCTO (Es crucial que no agregues el producto completo)
  - quantity: debe contener el número de ejemplares de dicho producto. El producto, de momento, se agregará de uno en uno.

Además, si un producto ya existente intenta agregarse al producto, incrementar el campo quantity de dicho producto.

Primera entrega

La persistencia de la información se implementará utilizando el file system, donde los archivos: productos,json y carrito.json respaldarán la información.
No es necesario realizar ninguna implementación visual, todo el flujo se puede realizar por Postman o por el cliente de tu preferencia.

Formato

- ✓ Link al repositorio de Github con el proyecto completo, sin la carpeta de Node_modules.

Sugerencias

- ✓ Link al video donde se explica.

MÓDULO I - CLASE 10

Websockets

Websockets + Handlebars

Integrar vistas y sockets a nuestro servidor actual.

CODERHOUSE

DESAFÍO ENTREGABLE

Websockets

Consigna

- ✓ Configurar nuestro proyecto para que trabaje con Handlebars y websocket.

Aspectos a incluir

- ✓ Configurar el servidor para integrar el motor de plantillas Handlebars e instalar un servidor de socket.io al mismo.
- ✓ Crear una vista “index.handlebars” la cual contenga una lista de todos los productos agregados hasta el momento
- ✓ Además, crear una vista “realTimeProducts.handlebars”, la cual vivirá en el endpoint “/realtimeproducts” en nuestro views router, ésta contendrá la misma lista de productos, sin embargo, ésta trabajará con websockets.
  - ✓ Al trabajar con websockets, cada vez que creemos un producto nuevo, o bien cada vez que eliminemos un producto, se debe actualizar automáticamente en dicha vista la lista.

DESAFÍO ENTREGABLE

Sugerencias

- ✓ Ya que la conexión entre una consulta HTTP y websocket no está contemplada dentro de la clase. Se recomienda que, para la creación y eliminación de un producto, Se cree un formulario simple en la vista realTimeProducts.handlebars. Para que el contenido se envíe desde websockets y no HTTP
- ✓ Si se desea hacer con HTTP, deberás buscar la forma de utilizar el servidor de Sockets dentro de la petición POST. ¿Cómo utilizarás un socket.emit dentro del POST?

Formato de entrega

- ✓ Link al repositorio de Github, el cual debe contar con todo el proyecto.
- ✓ No incluir node_modules

Testing de este entregable

MÓDULO I - CLASE 15

Primera práctica integradora

CODERHOUSE

Práctica de integración sobre tu ecommerce

CODERHOUSE

DESAFIO COMPLEMENTARIO

Consigna

Continuar sobre el proyecto que has trabajado para tu ecommerce y configurar los siguientes elementos:

Aspectos a incluir

- ✓ Agregar el modelo de persistencia de Mongo y mongoose a tu proyecto.
- ✓ Crear una base de datos llamada “ecommerce” dentro de tu Atlas, crear sus colecciones “carts”, “messages”, “products” y sus respectivos schemas.
- ✓ Separar los Managers de fileSystem de los managers de MongoDb en una sola carpeta “dao”. Dentro de dao, agregar también una carpeta “models” donde vivirán los esquemas de MongoDB. La estructura deberá ser igual a la vista en esta clase
- ✓ Contener todos los Managers (FileSystem y DB) en una carpeta llamada “Dao”

DESAFIO COMPLEMENTARIO

Aspectos a incluir

- ✓ Reajustar los servicios con el fin de que puedan funcionar con Mongoose en lugar de FileSystem
- ✓ **NO ELIMINAR FileSystem de tu proyecto**.
- ✓ Corroborar la integridad del proyecto para que todo funcione como lo ha hecho hasta ahora.

Formato

- ✓ Link al repositorio de GitHub con el proyecto completo (No incluir node_modules).

Sugerencias

- ✓ Te recomendamos que, para este entregable, repitas las pruebas realizadas en la pre-entrega de la clase 8.

MÓDULO I - CLASE 17

Mongo Avanzado (Parte I)

CODERHOUSE

Segunda pre-entrega de tu Proyecto final

Deberás entregar el proyecto que has venido armando, cambiando persistencia en base de datos, además de agregar algunos endpoints nuevos a tu ecommerce

CODERHOUSE

Profesionalizando la BD

Objetivos generales

- ✓ Contarás con Mongo como sistema de persistencia principal
- ✓ Tendrás definidos todos los endpoints para poder trabajar con productos y carritos.

Objetivos específicos

- ✓ Profesionalizar las consultas de productos con filtros, paginación y ordenamientos
- ✓ Profesionalizar la gestión de carrito para implementar los últimos conceptos vistos.

Formato

- ✓ Link al repositorio de Github, sin la carpeta de node_modules

Sugerencias

- ✓ Permitir comentarios en el archivo
- ✓ La lógica del negocio que ya tienes hecha no debería cambiar, sólo su persistencia.
- ✓ Los nuevos endpoints deben seguir la misma estructura y lógica que hemos seguido.

ENTREGA DEL PROYECTO FINAL

Se debe entregar

- ✓ Con base en nuestra implementación actual de productos, modificar el método GET / para que cumpla con los siguientes puntos:
  - Deberá poder recibir por query params un limit (opcional), una page (opcional), un sort (opcional) y un query (opcional)
    - -limit permitirá devolver sólo el número de elementos solicitados al momento de la petición, en caso de no recibir limit, éste será de 10.
    - page permitirá devolver la página que queremos buscar, en caso de no recibir page, ésta será de 1
    - query, el tipo de elemento que quiero buscar (es decir, qué filtro aplicar), en caso de no recibir query, realizar la búsqueda general
    - sort: asc/desc, para realizar ordenamiento ascendente o descendente por precio, en caso de no recibir sort, no realizar ningún ordenamiento

ENTREGA DEL PROYECTO FINAL

Se debe entregar

- ✓ El método GET deberá devolver un objeto con el siguiente formato:

```
{
        status:success/error
        payload: Resultado de los productos solicitados
        totalPages: Total de páginas
        prevPage: Página anterior
        nextPage: Página siguiente
        page: Página actual
        hasPrevPage: Indicador para saber si la página previa existe
        hasNextPage: Indicador para saber si la página siguiente existe.
        prevLink: Link directo a la página previa (null si hasPrevPage=false)
        nextLink: Link directo a la página siguiente (null si hasNextPage=false)
}
```

- ✓ Se deberá poder buscar productos por categoría o por disponibilidad, y se deberá poder realizar un ordenamiento de estos productos de manera ascendente o descendente por precio.

ENTREGA DEL PROYECTO FINAL

Se debe entregar

- ✓ Además, agregar al router de carts los siguientes endpoints:
  - DELETE api/carts/:cid/products/:pid deberá eliminar del carrito el producto seleccionado.
  - PUT api/carts/:cid deberá actualizar el carrito con un arreglo de productos con el formato especificado arriba.
  - PUT api/carts/:cid/products/:pid deberá poder actualizar SÓLO la cantidad de ejemplares del producto por cualquier cantidad pasada desde req.body
  - DELETE api/carts/:cid deberá eliminar todos los productos del carrito
  - Esta vez, para el modelo de Carts, en su propiedad products, el id de cada producto generado dentro del array tiene que hacer referencia al modelo de Products. Modificar la ruta /:cid para que al traer todos los productos, los traiga completos mediante un “populate”. De esta manera almacenamos sólo el Id, pero al solicitarlo podemos desglosar los productos asociados.

ENTREGA DEL PROYECTO FINAL

Se debe entregar

- ✓ Crear una vista en el router de views ‘/products’ para visualizar todos los productos con su respectiva paginación. Cada producto mostrado puede resolverse de dos formas:
  - Llevar a una nueva vista con el producto seleccionado con su descripción completa, detalles de precio, categoría, etc. Además de un botón para agregar al carrito.
  - Contar con el botón de “agregar al carrito” directamente, sin necesidad de abrir una página adicional con los detalles del producto.

- ✓ Además, agregar una vista en ‘/carts/:cid (cartId) para visualizar un carrito específico, donde se deberán listar SOLO los productos que pertenezcan a dicho carrito.

Finalizamos el primer módulo

¡Felicitaciones! Has completado los desafíos y pre-entregas correspondientes al primer módulo.

Al iniciar el II, encontrarás el próximo manual de desafíos. Recuerda que estas instancias son fundamentales para la construcción del Proyecto Final.

¡Ánimo, falta cada vez menos! 🚀