Segundo Examen Parcial. TC2026: Web Applications Development. 2021  
  
NOMBRE: Lucia Cantu-Miller

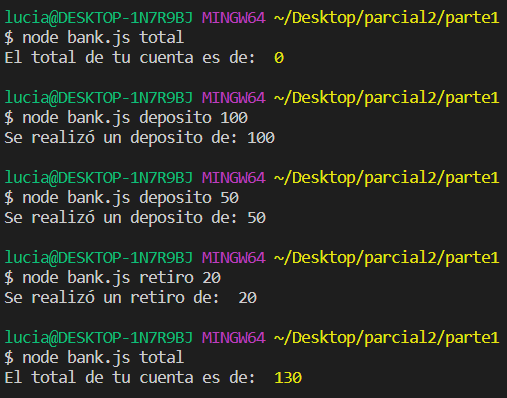
Instrucciones:

* Mantén tu cámara encendida en todo momento.
* Puedes utilizar el material visto en clase, así como laboratorios.
* Codifica todo desde cero.
* Desarrolla lo que se te pide en cada carpeta: parte1 y parte2.
* Cuando termines: Asegúrate de haber llenado tu nombre, de incluir las capturas de pantalla donde se te indique y subir este documento junto con el código a Canvas.

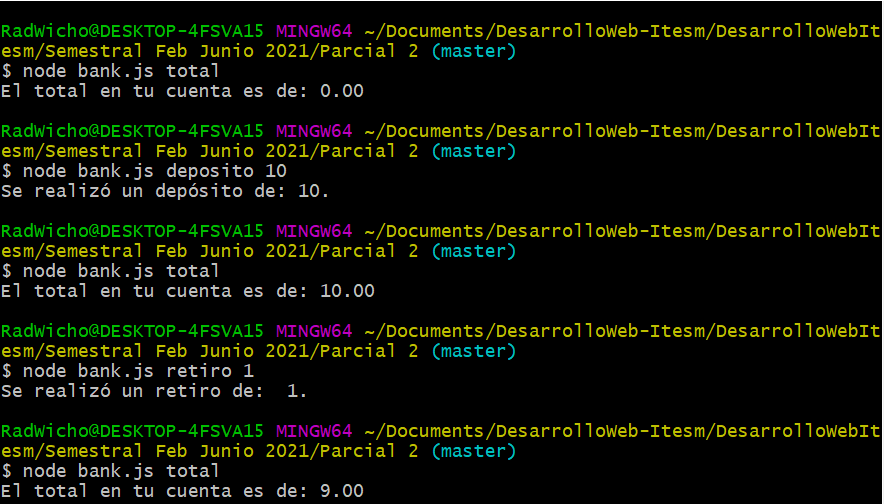
**Parte 1 - NodeJS**

* Desarrolla una aplicación en NodeJS llamada “bank.js” que tomará de la línea de comandos instrucciones para registrar o ejecutar operaciones.
* Las posibles operaciones son:
  + **total**: deberá mostrar el total en la cuenta del usuario.
  + **deposito** <*numero*>: deberá agregar una cantidad positiva al total de la cuenta.
  + **retiro** <*numero*>: deberá agregar una cantidad negativa al total de la cuenta.
* Para las operaciones de **depósito** o **retiro** se deberá escribir la transacción en un archivo llamado banco.txt
* Cuando termines incluye una captura de pantalla con algunas operaciones de prueba.

*[Agrega aquí una captura de pantalla con las pruebas ]*

**

EJEMPLO:



El contenido de “banco.txt” es:   
, 10, -1

**Tips:**

* Como estaremos interactuando con archivos necesitamos importar el paquete “FS”.
* process.argv te ayudará a obtener los argumentos de la línea de comandos.
* Necesitarás implementar las siguientes operaciones: fs.readFile , fs.appendFile (este método creará el archivo en caso de no existir, solo asegúrate de que al menos exista el archivo antes de leerlo / utilizar la operación de total).
* En las operaciones de depósito, escribe en el archivo “banco.txt” la cantidad depositada y en retiro , la cantidad retirada pero anteponiendo un guion “-“ de esta manera podrás identificar fácilmente operaciones de retiro o depósito. Separa cada operación por coma.
* Para obtener el total, lee el archivo “banco.txt” y separa cada elemento por coma ,ejemplo: data.split(“, “) y después suma todos los números para obtener el total.
* Recuerda que los datos dentro del archivo son Strings así que deberás convertirlos a número para poder realizar alguna operación numérica.
* Está fuera del alcance en este proyecto validaciones de si hay saldo suficiente o no. Solo enfócate en las operaciones principales, **total**, **deposito**, **retiro**. Aun y si el total dá negativo.

**Parte 2 – Express + EJS**

Desarrolla un servidor con NodeJS + express + ejs capaz de soportar diferentes rutas para mostrar algunos datos.

* Crea un archivo llamado “server.js” y agrega todo el código necesario para tener un servidor con express ejecutándose en el puerto 5000. (¡No olvides de inicializar tu proyecto!)
* Prepara tu servidor para soportar EJS. En layouts crea un archivo para el header y footer. (utilizando como base una página html simple). Y en las páginas individuales necesitarás dos: 1) “animal.ejs” 2) “index.ejs
* Utiliza el siguiente arreglo para simular la BD:

var animals = [

  {

    animalType: "dog",

    pet: true,

    fierceness: 4

  }, {

    animalType: "cat",

    pet: true,

    fierceness: 10

  }, {

    animalType: "giraffe",

    pet: false,

    fierceness: 4

  }, {

    animalType: "zebra",

    pet: false,

    fierceness: 8

  }, {

    animalType: "lion",

    pet: false,

    fierceness: 10

  }

];

* Necesitarás crear las siguientes tres rutas:   
  1. ***GET***escuchando en la url: “/api/:animal” (es una ruta dinámica) que buscará el contenido del arreglo y si encuentra el tipo de animal que se está buscando renderizará la página “animal.ejs” y mostrará de contenido:

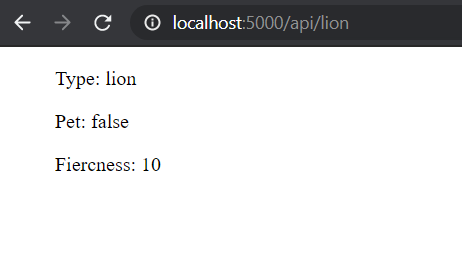
<p>Type: <animalType> </p>

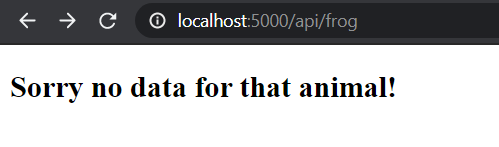
<p>Pet: <pet> </p>

<p>Fierceness: <fierceness></p>

Sustituye los valores por los correspondientes del animal. En caso de no encontrar datos, la página deberá mostrar solamente <h2> Sorry no data for that animal </h2>

[Agrega una captura de pantalla probando con un animal existente y con uno no existente. ]





* 1. **GET** escuchando en la url: “/all-pets . Esta ruta deberá renderizar la página “index.ejs” y deberá mostrar solamente el arreglo de datos donde la propiedad **pet** sea verdadera. Utiliza la siguiente estructura como base:

<ul>

  // por cada elemento que coincida repite lo siguiente con el dato correspondiente

    <li>

      <p>Type: <animalType></p>

      <p>Pet: <pet></p>

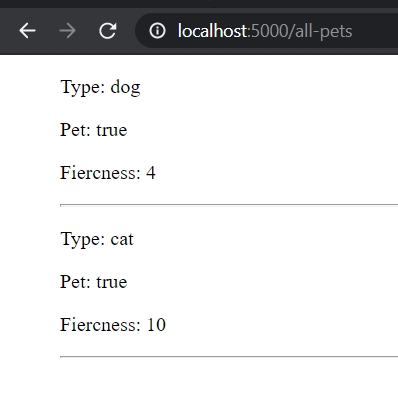
      <p>Fierceness: <fierceness></p>

    </li>

  // aqui termina la iteración

</ul>

*[Agrega una captura de pantalla probando la ruta ]*

**

* 1. **GET** escuchando en la url: “/all-non-pets . Esta ruta deberá renderizar la página “index.ejs” y deberá mostrar solamente el arreglo de datos donde la propiedad **pet** no sea verdadera. Es lo inverso a la ruta del paso b). Utiliza la misma página “index.ejs” para mostrar la información.

[Agrega una captura de pantalla probando la ruta ]

