## Universidad Modelo



## Escuela de Ingeniería.

## Carrera: Ingeniería en Desarrollo de Tecnología y Software Asignatura: Algoritmos

## Nombre del Profesor: Edson Geovanny Estrada López Nombre de la Actividad: Practi-Caja

## Fecha de Entrega: 30/06/2020

## Integrantes: Jimena Vadillo, Roger Almeida, Sergio Mendoza, Miguel Tostado, Daniel López y Emilio Rivas

API

**Nombre del archivo en el backend:** accountsController.js

**Descripción:** Este archivo tiene la funcionalidad de las interacciones en el apartado de las cuentas de los usuarios.

**Parámetros:**

* accountsCtrl: Variable que almacena cada cambió que se haga en las cuentas.
* accountsData: Variable que almacena los datos de las cuentas.
* respu: Variable que almacena los cambios hechos en las cuentas.
* newAccount: Variable que almacena los cambios hechos en una cuenta.
* tipo: Variable que almacena el tipo de cuenta del usuario.
* firstName: Variable que almacena el nombre del usuario.
* lastName: Variable que almacena el apellido del usuario.
* nip: Variable que almacena el nip del usuario.
* balance: Variable que almacena la cantidad de dinero de la cuenta del usuario.
* message: Es la impresión de un mensaje para el usuario dependiente de lo que haya hecho el usuario (Eliminar, actualizar o crear cuenta)

**Funciones:**

* getAccount: Obtiene los datos de una cuenta especificada.
* getAccounts: Obtiene los datos de todas las cuentas registradas.
* createAccount: Ejecuta la creación de una cuenta nueva.
* updateAccount: Ejecuta la actualización de datos de una cuenta.
* deleteAccount: Ejecuta la eliminación de una cuenta.

**Retorno:** La ejecución de este archivo puede retornar tres posibles cosas dependiendo de lo que haga el usuario, uno es la creación de una nueva cuenta, dos es la actualización de datos de una cuenta existente y tres es la eliminación de una cuenta.

**Nombre del archivo en el backend:** adminServicesController.js

**Descripción:** Este archivo tiene la funcionalidad de las interacciones con la pantalla de servicios desde la cuenta de administrador.

**Parámetros:**

* serviceCtrl: Variable que almacena cada interacción hecha en los servicios.
* servicesData: Variable que almacena la información de los servicios.
* transactionsData: Variable que almacena la información de las transacciones hechas en los servicios.
* description: Variable que almacena el tipo de servicio el cual se inserte.
* cost: Variable que almacena el costo de un servicio.
* newService: Variable que almacena los cambios hechos en un servicio o un nuevo servicio.
* message: Es la impresión del mensaje con los cambios hechos en el o los servicios.
* respu: Variable que almacena el cambio que se haga en un servicio.

**Funciones:**

* createNewService: Función que almacena los datos de un nuevo servicio insertado por el administrador.
* updateService: Función que almacena la actualización en la información de un servicio.
* allService: Función que almacena la llamada de todos los servicios existentes.
* getService: Función que almacena la llamada de un servicio específico.
* deleteService: Función que representa la eliminación de un servicio.

**Retorno:** La ejecución de este archivo puede retornar cuatro posibles cosas dependiendo de lo que indique el administrador, uno es la creación de un nuevo servicio, dos es la actualización de la información de un servicio, tres es la indicación o muestra de la información de uno o todos los servicios y cuatro es la eliminación de un servicio.

**Nombre del archivo en el backend:** depositoControllers.js

**Descripción:** Este archivo tiene la funcionalidad de las interacciones de las cuentas de admin y cliente para los depósitos.

**Parámetros:**

* depositoCtrl: Variable que almacena cada interacción hecha en los depositos.
* cashmodel: Variable que almacena las actualizaciones del dinero en la base de datos.
* usermodel: Variable que almacena la interacción del dinero en la cuenta de un usuario.
* denominacion: Variable que representa el tipo de denominación de algún billete o moneda.
* cantidad: Variable que almacena la cantidad del depósito hecho.
* coin: Variable que representa las denominaciones actuales que contiene el cajero.
* update: Variable que representa las variables que se van a actualizar en el depósito.
* quantity: Variable que representa el registro de dinero que contiene el cajero.
* UpdateBank: Variable que almacena la actualización del dinero del banco en cuanto a la cantidad depositada por el usuario que se sumara a la del cajero.
* updateUser: Variable que almacena la actualización del dinero actual del usuario.
* message: Mensaje mostrado para la confirmación de la acción hecha.

**Funciones:**

* hacerDepositoAdmin: Función que representa el depósito del administrador al dinero del cajero.
* hacerDepositoUser: Función que almacena la información del depósito hecho por un usuario.

**Retorno:** La ejecución de este archivo puede retornar dos posibles respuestas, uno es la actualización del registro de dinero del banco tras el depósito del administrador o dos, la actualización del dinero de la cuenta de un usuario (y del banco) por un depósito.

**Nombre del archivo en el backend:** pagoControllers.js

**Descripción:** Este archivo tiene la funcionalidad de las interacciones de la cuenta de un usuario con el pago de un servicio.

**Parámetros:**

* pagoCtrl: Variable que almacena cada interacción hecha en los pagos.
* cashmodel: Variable que almacena las actualizaciones del dinero en la base de datos.
* usermodel: Variable que almacena la interacción del dinero en la cuenta de un usuario.
* user: Variable que almacena la información del usuario que hace el pago.
* updateuser: Variable que representa la actualización de los datos (balance) del usuario.
* tipo: Variable que almacena el tipo de cuenta del usuario.
* firstName: Variable que almacena el nombre del usuario.
* lastName: Variable que almacena el apellido del usuario.
* nip: Variable que almacena el nip del usuario.
* balance: Variable que almacena la cantidad de dinero de la cuenta del usuario.
* costo: Variable que representa el costo del pago del servicio.
* pago: Variable que representa el pago del usuario por un servicio.
* denominacion: Variable que representa el tipo de denominación de algún billete o moneda.
* coin: Variable que representa las denominaciones actuales que contiene el cajero.
* update: Variable que representa las variables que se van a actualizar en el depósito.
* quantity: Variable que representa el registro de dinero que contiene el cajero.
* message: Mensaje mostrado para la confirmación de la acción hecha.

**Funciones:**

* pagoTarjeta: Función que representa el pago de un servicio por tarjeta
* pagoEfectivo: Función que representa el pago de un servicio por efectivo.

**Retorno:** La ejecución de este archivo retorna el pago de un servicio el cual puede ser hecho con efectivo (el cual suma el dinero al cajero) o directo de la cuenta del usuario (lo cual suma el dinero al cajero y lo resta de la cuenta del usuario).

**Nombre del archivo en el backend:** retiroControllers.js

**Descripción:** Este archivo tiene la funcionalidad de las interacciones de la cuenta de un usuario con los retiros de la cuenta.

**Parámetros:**

* retiroCtrl: Variable que almacena cada interacción hecha en los pagos.
* cashmodel: Variable que almacena las actualizaciones del dinero en la base de datos.
* usermodel: Variable que almacena la interacción del dinero en la cuenta de un usuario.
* user: Variable que almacena la información del usuario que hace el pago.
* updateUser: Variable que representa la actualización de los datos (balance) del usuario.
* tipo: Variable que almacena el tipo de cuenta del usuario.
* firstName: Variable que almacena el nombre del usuario.
* lastName: Variable que almacena el apellido del usuario.
* nip: Variable que almacena el nip del usuario.
* balance: Variable que almacena la cantidad de dinero de la cuenta del usuario.
* cantidad: Variable la cantidad que el usuario va a retirar
* id: Variable que representa el id de la cuenta del retiro
* inputArr: Es el arreglo de las denominaciones a ordenar.
* i: variable a iterar en el arreglo de denominaciones.
* matriz: Nuevo arreglo para iteraciones de las cantidades.
* fila1: Fila en la que se iteran las denominaciones.
* números: Arreglo que almacena los números a iterar.
* posiblsMonedas: Lista la cual se ordena para obtener el resultado.
* cantidadTotal: la cantidad final que se le dará al usuario
* denominacion: Variable que representa el tipo de denominación de algún billete o moneda.
* coin: Variable que representa las denominaciones actuales que contiene el cajero.
* update: Variable que representa las variables que se van a actualizar en el depósito.
* quantity: Variable que representa el registro de dinero que contiene el cajero.
* message: Mensaje mostrado para la confirmación de la acción hecha.

Funciones:

* actualizar: Función que representa la actualización del dinero del usuario.
* actualizarCuenta: Función que representa la actualización de la cuenta.
* ordenamiento: Representa la función en la que se hace el ordenamiento dinámico.
* dynamic: Es la función en la que se va a sacar la cantidad total

**Retorno:** La ejecución de este archivo retorna el efectivo del retiro hecho por el usuario.

**Nombre del archivo en el backend:** transactionsControllers.js

**Descripción:** Este archivo tiene la funcionalidad de almacenar la información de las transacciones para ser usada en el historial de transacciones.

**Parámetros:**

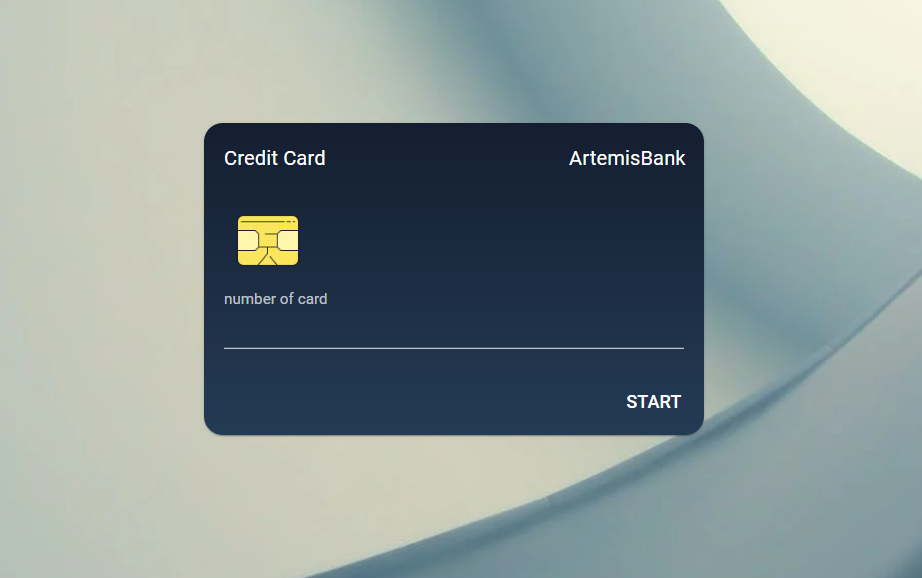
* transactionCtrl: Variable que almacena la información de las transacciones.
* transactionModel: Variable que almacena las actualizaciones de las transacciones en el historial.
* typeId: Variable que almacena tipo de Id (Cliente o Admin).
* accountId: Variable que almacena el Id del usuario.
* utilitiesId: Variable que almacena el Id de los servicios.
* ammount: Variable que representa el costo del pago del servicio.
* newTransaction: Variable que almacena una nueva transacción hecha.
* message: Variable que representa un mensaje al usuario.

Funciones:

* getAllTransactions: Función que almacena la información de todas las transacciones.
* getUserTransactions: Función que almacena los usuarios que hicieron transacciones.
* getTypeTransactions: Función que representa los tipos de transacciones hechas.
* getServiceTransactions: Función que almacena las transacciones hechas en servicios.
* postTransaction: Función que almacena los datos de transacciones ya hechas.

**Retorno:** La ejecución de este archivo retorna el historial de transacciones hechas por clientes y administrador.

**Frontend**

****

