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 Almeyda, Sergio Mendoza, Miguel Tostado, Daniel López y Emilio Rivas

Índice

1. Introducción

El objetivo principal para este proyecto correspondiente a la materia de algoritmos, consiste en realizar un programa que integre el algoritmo basado en programación dinámica para cumplir con las funciones de un cajero automático, de igual manera se requiere de la implementación de archivos JSON para almacenar la información.

2. Definición general del Proyecto

De manera general el proyecto consiste en simular la funcionalidad de un cajero automático usando el algoritmo (Dynamic Programming Algorithm), así como otras funciones e implementaciones aprendidas a lo largo del semestre. La finalidad es simular lo que sería un intercambio de dinero, depósitos, retiros, así como pagos, para ello serán utilizadas las divisas mexicanas y principalmente la parte que da el cambio es la que debe dar las denominaciones de mayor conveniencia dependiendo de la cantidad de dinero disponible a dar y la cantidad a cambiar.

3. Especificación de requerimientos

El programa contará con un apartado de inicio de sesión mediante un número de tarjeta de crédito. Se pretende cubrir dos perfiles, el de administrador y el de un usuario común. En el caso del usuario común se debe cubrir con la función de ver el saldo actual, historial de transacciones, en este se debe poder filtrar por tipo de transacción así como permitir ver todas. Se debe poder realizar retiros con una opción mínima de retiro de $50, en caso de no existir efectivo disponible para realizar el retiro se deben mostrar los mensajes correspondientes. Cumplir con la función de hacer depósitos, esta transacción debe ser agregada al historial de transacciones. Cumplir con la función de pagar servicios, facilitando al usuario elegir el servicio y la cantidad a pagar, en caso de ser pago en efectivo, se debe devolver el cambio correspondiente, la denominación mínima es de $1, de no ser posible la transacción mostrar los mensajes correspondientes.

En el caso del administrador, mostrar el historial de transacciones cumpliendo con los criterios mencionados en el usuario normal, realizar depósitos con los criterios mencionados

anteriormente, por ultimo cubrir con la administración de los servicios, donde se pueden agregar o quitar de las funciones del cajero.

4. Listado de herramientas utilizadas

Para el desarrollo de este proyecto se implementó el lenguaje de programación JavaScript, manejando el entorno de desarrollo integrado (IDE) Visual Studio Code, manejando el stack de tecnologías MERN (MongoDB, Express js, React js y Node js), se usaron archivos JSON para almacenar la información necesaria, se recurrió a MongoDB para el maneja de la base de datos, a Node js y Express para el backend y a React js para el frontend. El proyecto se realizó a modo de página web implementando etiquetas HTML.

5. API

5.1. Backend.

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:

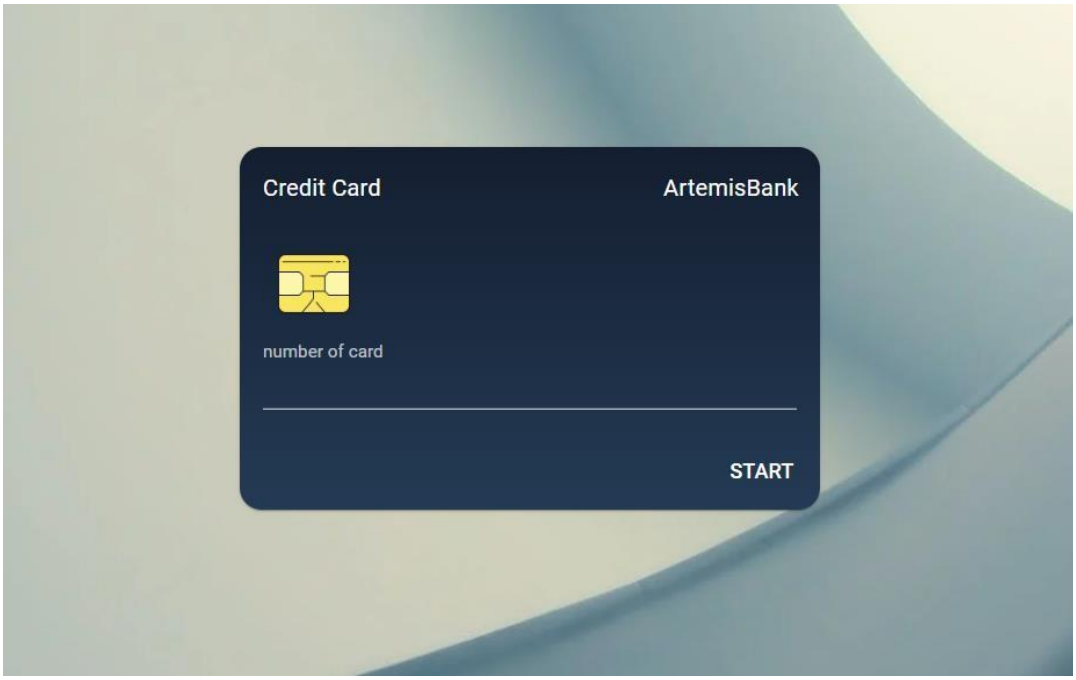
- transactionCtrl: Variable que almacena la información de las transacciones.
- transactionModel: Variable que almacena las actualizaciones de las transacciones en el historial.
- typeId: Variable que almacena tipo de Id (Cliente o Admin).
- accountId: Variable que almacena el Id del usuario.
- utilitiesId: 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:

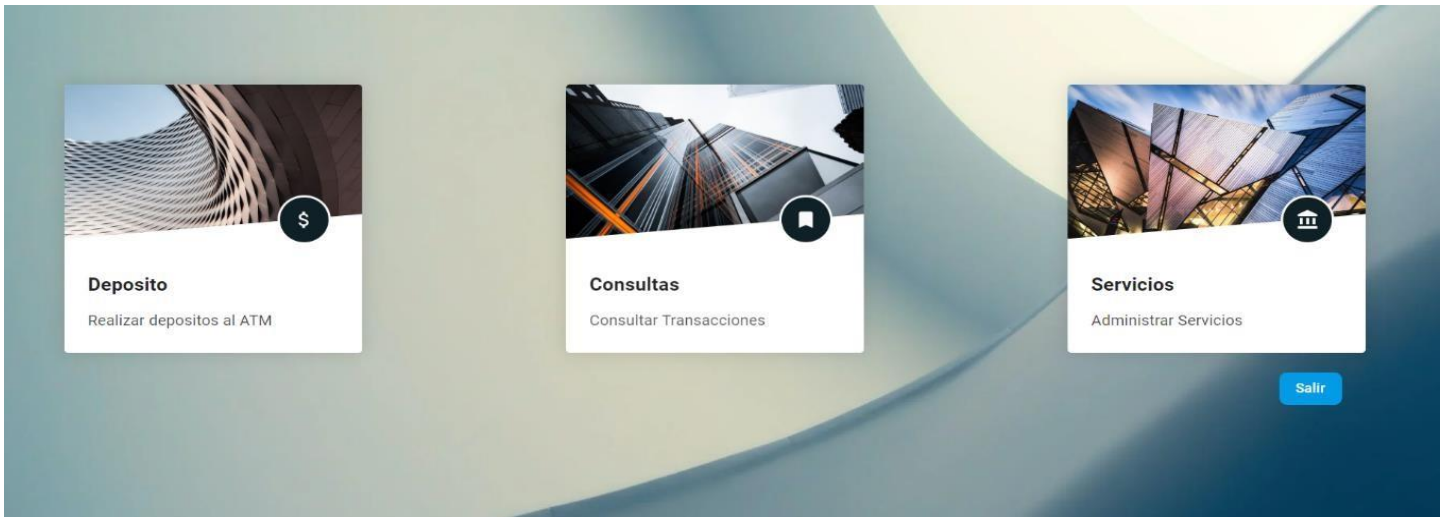
- getAllTransactions: Función que almacena la información de todas las transacciones.
- getUserTransactions: Función que almacena los usuarios que hicieron transacciones.
- getTypeTransactions: Función que representa los tipos de transacciones hechas.
- getServiceTransactions: Función que almacena las 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.

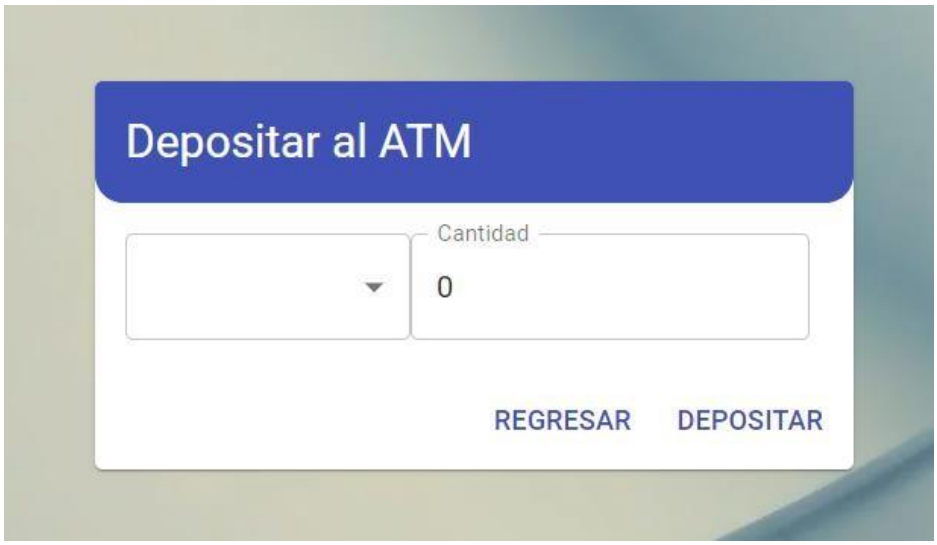
5.2. Frontend.



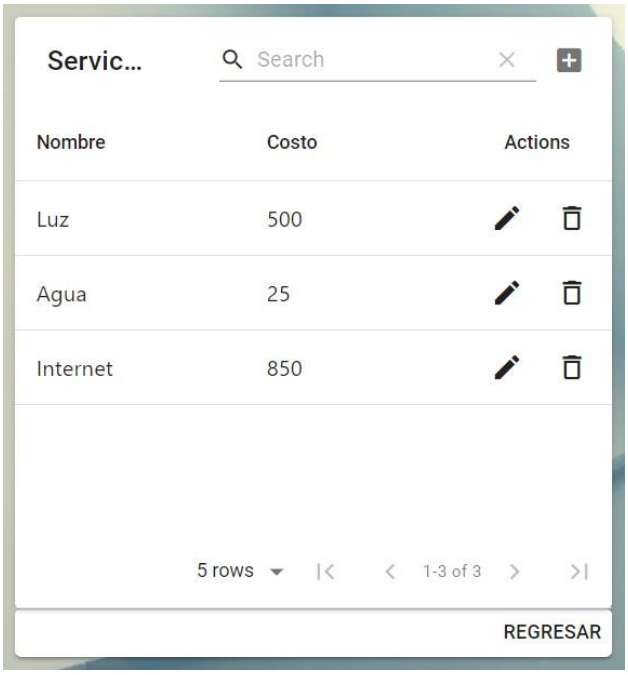
Ventana de inicio, donde el usuario debe ingresar el número de su tarjeta para poder iniciar sesión.



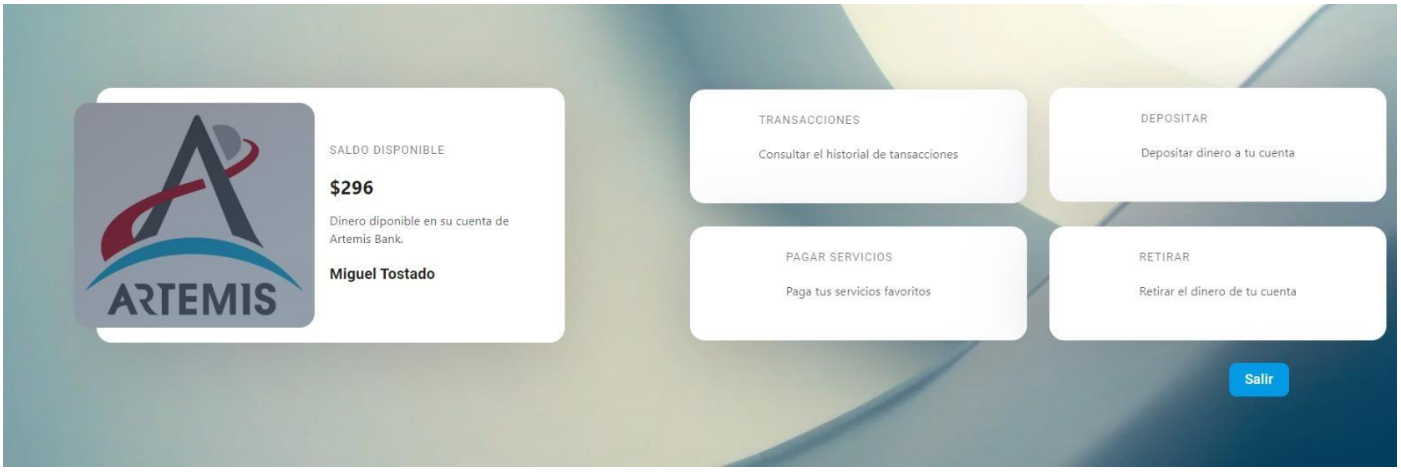
Menú con las funcionalidades de administrador del sistema, tales como realizar depósitos, consultas de transacciones o administrar servicios.



Ventana que cumple con la función de realizar depósito.

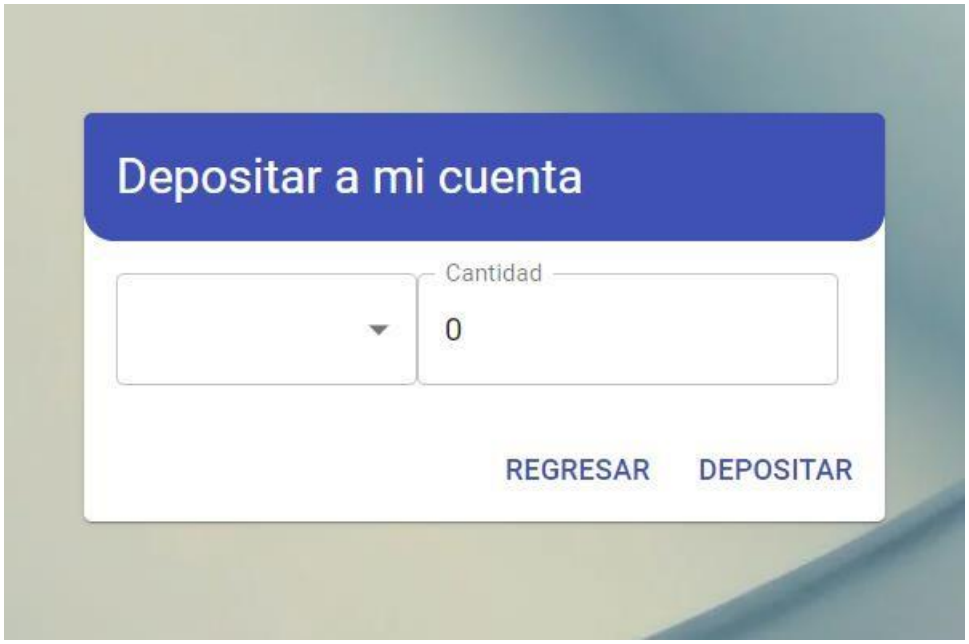


Ventana de administración de servicios que nos permite realizar acciones sobre ciertos movimientos realizados.



Ventana de usuario común, donde puede ver su saldo disponible, el propietario de la cuenta, consultar su historial de transacciones, pagar servicios, depositar o retirar efectivo.

Ventana de usuario que permite observar su historial de transacciones.



Ventana que permite realizar depósito de efectivo a la cuenta.

Ventana para realizar el pago de los servicios disponibles en el sistema.

Ventana que permite realizar retiro en efectivo del dinero de la cuenta.

5.3 Instrucciones de conexión mediante localhost

Al momento de abrir el archivo del proyecto en Visual Studio Code, se dirige a la carpeta del backend y le da click derecho, presionara la opción de “Open in terminal” y eso abrirá la terminal de ese apartado, de igual forma repetirá el mismo proceso con la carpeta del frontend, una vez abiertas las dos terminales, en ambas terminales se ejecuta el comando “npm i”, posteriormente en la terminal del frontend se ejecuta el comando “npm start” y en la terminal del backend se ejecuta el último comando que es “npx nodemon src/index.js”, aunque es indiferente el orden en que se pongan estos dos últimos comandos, finalmente el localhost del proyecto se abra abierto automáticamente con el programa funcional.

6. Lista de bugs

De momento no se encontró ningún bug o error.

7. Conclusiones

Almeyda Roger: En conclusión, el trabajo del desarrollador web (esa especie de alquimista obligado a dominar y combinar variables tan dispares, y a veces de apariencia incompatible, como la usabilidad y el atractivo de cara al usuario, y la efectividad técnica), este trabajo, decía, es un proceso largo que requiere un enorme grado de instrucción y aprendizaje técnico, pero también generosas dosis de intuición y anticipación. A fin de cuentas, la Red muta y crece cada poco tiempo, y no podemos quedarnos atrás.

López Daniel: Se llegó a la conclusión que aprendimos a utilizar diferentes herramientas para llegar a la meta, el análisis, el diseño, el proceso de las distintas partes del sistema que nos llevaron a comprender y aprender cómo realizar un proyecto de esta magnitud partiendo de cero.
También pudimos notar que un buen análisis y diseño nos da como resultado un sistema con buen funcionamiento y escalable.

Mendoza Sergio: En programación, como en cualquier otra actividad, lo mejor para mejorar es practicar. Realizar proyectos personales es la forma ideal para reforzar los conocimientos conseguidos e ir adquiriendo otros nuevos. Finalmente, para concluir el proyecto, cabe recalcar que sirvió demasiado para practicar y mejorar en el ámbito de las herramientas de desarrollo web, implementando un lenguaje como JS y utilizando herramientas para volver más interactiva la página.

Rivas Emilio: El ámbito web es un mundo enorme totalmente aparte, sin embargo, con el trabajo en equipo, las recomendaciones y los consejos, entendí los conceptos básicos para colaborar en el proyecto, una buena organización y ganas de hacer las cosas bien permiten conseguir mejores resultados, por lo tanto, me siento enteramente satisfecho con el resultado final.

Tostado Miguel: En conclusión, el diseño web es un elemento imprescindible en cualquier negocio, ya que es el que se encarga de crear la cara pública online de una marca o empresa, como en este caso, se implementó para la simulación de funcionamiento de un cajero automático. Es algo en lo que hay que dedicar cuidado para conseguir los mejores resultados y se pueda disfrutar de todos los beneficios que ofrece una web cuidada.

Vadillo Jimena: Para finalizar, en conclusión, una página web es un documento estructurado que contiene texto y etiquetas HTML. Gracias a JavaScript podemos acceder a la estructura de la página y modificarla, de esta manera se obtienen resultados bastante complejos y efectivos, de tal magnitud como la requerida para el proyecto que llevamos a cabo en conjunto. No era la meta principal para cumplir con el proyecto, pero, al final se pudo conseguir cumplir el objetivo.