Angular ATM Project

Project Summary

This project will perform the following:

- Have a user withdraw money from a stock of money (stored as data)
    - For this project, the ATM will be stocked with a default of ten of each USD denomination ($100, $50, $20, $10, $5, $1)
    - This stock of money can have more than ten bills of each denomination.
- Have the ATM system dispense success and error messages upon attempting to withdraw cash.
- Have the ATM system track and display the money in stock.
- Have the ATM system track a list of each transaction that occurs.
- Have a user able to restock the ATM system as needed.

Project Functions

Below are a list of variables and functions required to operate the ATM system, organized by the component or service they are a part of.

MoneyService

This service handles all of the ATM transactions that the user will be using.

atmStock

- An array that holds all of the stored values of each bill in the ATM, split up by denominations.
- $100 bills are in position zero, going all the way down to $1 in position five.

denominations

- An array that holds all the values of each denomination.

withdrawCash(cashAmount)

- Takes the amount requested as an argument name (cashAmount).
- Check the stock of bills in storage and determine if the amount requested is available in stock using **getBills()** to get the number of bills from each denomination.
- If the returned array from **getBills()** contains a true value, run **updateStock().**

updateStock(add, billsRequested, cashAmount)

- Takes a boolean (add) to determine if a user is withdrawing or adding money (from an array called billsRequested)
- Depending on the value of the argument boolean, update the ATM stock by adding or subtracting the requested amount of money, using the values of the billsRequested array compared with the stock of cash in the ATM system.
- Call the **AlertService** to generate a success message.

getBills(cashAmount)

- Takes the amount of money requested as an argument.
- Returns an array containing a boolean and an array of bill denominations.
- Divide up the amount using a while loop and returns an array showing how many bills are requested from each amount in the ATM stock.
- After the while loop finished, if the remaining amount of cash is greater than zero, this means the ATM didn't have enough cash in stock. If this happens, call the **AlertService** to generate an error message. This also sets the boolean value in the return array to false.
- Return the array.

AlertService/AlertModule

This service handles all of the alert messages seen on the front end. This is part of an **AlertModule** that contains its own html, component, model, and lastly, the service that handles it all. I built this module as part of another project I did on my own time, so I decided to include it as part of the exercise.

subject

- Creates a new RxJS Subject, using the Alert object. Can be subscribed to.

defaultId

- A string, set to 'default-alert'. This is used if you want to display multiple alerts in different locations, and is not used as part of this application's functionality.

userAlertHistory

- An array of objects that is meant to be filled with the alert messages generated by the system. Contains the alertMessage and timestamp.

anAlert(id)

- Takes an id variable as an argument. In this case, it's been set to another variable called defaultId.
- Whenever a new alert component is created, its **ngOnInit()** function will call this function to subscribe to any new alert notifications.

- Returns an observable to subscribe to whenever an alert is triggered.

success(message, options)

- Takes the alert message and options object as arguments. The options argument can be set in the components that use them.
    - autoClose: can be set to true or false to trigger an automatic fadeout of the alert.
    - keepAfterRouteChange: can be set to true or false to keep the alerts from destroying themselves after a route change.
- Generates a new success alert message, calling **alert()** using the arguments passed to it.
- Take the timestamp and alert message and add it to userTransactionHistory.

error(message, options)

- Takes the alert message and options object as arguments.
- Generates a new error alert message, calling **alert()** using the arguments passed to it.

alert(alert)

- Takes an alert object as an argument.
- Sets the alert's ID value to whatever was set in the defaultId variable.

clear(id)

- Takes the default ID as an argument.
- Clears all of the alerts using an empty alert to trigger the observable.

getMessageHistory()

- Returns the userAlertHistory variable.

AlertComponent

This component is used by the AlertService to create new alert messages.

ngOnInit()

- Subscribes to new alerts, using the AlertService's **onAlert().**
- Subscribe to any route changes, which triggers the AlertService's **clear()** function.

ngOnDestroy()

- Unsubscribes from alert and route subscriptions whenever the alert is removed from the DOM.

removeAlert(alert)

- Takes the alert object as an argument.
- Sets up the fade out animations and removes the alert from the DOM.

cssClass(alert)

- Takes the alert object as an argument.
- Called by the HTML of the component to append the appropriate Bootstrap 4 classes to the alert HTML.

WithdrawComponent

This component handles all of the functionality of the Withdraw Page.

withdrawForm

- A FormGroup that holds the value of the amount of money requested on the Withdraw Page.

onSubmit()

- Grabs the cash value from the withdrawalForm and, if the value entered is a positive, non-zero value, calls the MoneyService's **withdrawCash()** function using the number entered as an argument.
- If the value is negative or 0, generate an error message with the **AlertService**.

RestockComponent

restockForm

- A FormGroup object that will eventually be set in the constructor. Holds all of the form values for each bill denomination.

onSumbit()

- Grabs the values from the restockForm and passes them along to **addCash().**

addCash(restockAmount)

- and checks if any values are negative. If any values are negative, trigger an error message with the **AlertService** and immedately end the function.
- If all the values are valid, call the MoneyService's **updateStock()**, using the array of values as an argument.

OverviewComponent

This component displays the bills currently stored, as well as a list of the apps transaction history.

transactions

- An empty variable that will hold the array of messages retrieved by **displayHistory().**

atmMoney

- An empty array that will hold the values of each bill denomination and bill amount after **displayDenominations()** finishes.

displayHistory()

- Calls the AlertService's **getMessageHistory()** to get the list of messages generated by the service.

displayDenominations()

- Grabs the amount of bills per denomination from the **MoneyService**, then pushes those values to the atmMoney object using a while loop.

Project Views

Below is a list of views used by the application, as well as expected behavior for each.

Withdraw Page

- Users should be able to enter a dollar amount and press a withdraw button when desired dollar amount has been entered.
- Withdraw button should remain disabled until the user enters an amount.
- Once the withdrawal button is pressed, display message whether or not transaction was successful.

Restock Page

- User can enter quantities for each of the standard USD denominations.
- Once the restock button is pressed, the total quantity of each denomination is updated.
- Once the restock button is pressed, display message whether or not transaction was successful.

ATM Overview Page

- Display the quantities of each denomination currently in the ATM.
- Display the transaction history of transaction messages.