Functionality Documentation

# Routes

The program has 4 different routes: ‘ ’ which redirects to ‘Log-In’

‘Log-In’ which goes to LogInPageComponent

‘Register’ which goes to SignUpPageComponent

‘Main-Account-Page’ which goes to MainAccountPageComponent

# Interfaces

## Account

Account {

id: number;

cardNum:number;

pinNum:number;

savings:number;

chequeings:number;

transactions: Arrary<Transaction>

}

## Transaction

Transaction {

date: string;

accountType: string;

amount: number;

balance: number;

}

## Credentials

Credential {

cardNum:number;

pinNum: number;}

## LocalStorageError

LocalStorageError {

error: string;

}

# Components

## Log-In-Page component

This is where the program starts off. It displays authenticate credentials html. If authenticate credentials event emitter onAuthenticateCredentials emits with the user’s card number and pin number then authenticateCredentials($event) triggers where $event is the user’s inputted card number and pin number. It uses Account Service and subscribes to AccountService.getAccounts(), so, the file always has the up to date list of accounts. The authenticateCredentials(Credentials) uses a helper method getAccount(Credentials) which returns the account with the matching card number if any. Then authenticateCredentials checks to see if any account exists. If so then it checks if the pin numbers are correct. If they are correct, the set the account as logged in with the account service and navigate to ‘Main-Account-Page’ route, otherwise it sends an appropriate alert to the user telling them what went wrong.

## Authenticate credentials component

The html file is a form which asks for the user’s card number and pin number. It also has a submit button. It also has an a tag which allows the user to register if they do not already have an account by taking them to the signUpPageComponent. If the user presses submit and the card number or pin number is missing, it does not allow the user to proceed and pops up an alert telling the user to input the required information. If a card number and pin number are both inputted, then an event emitter onAuthenticateCredentials emits this information.

## Sign-Up-Page component

It has an a tag which redirects user back to log in page if they already have an account. It also shows createCredentials component html page. If onCreateCredentials emits than createAccount($event). It uses Account service’s createAccount method to add the account to the server. It also stops displaying createCredentials html and starts displaying a label which saying account successfully created and an a tag which redirects to log in page.

## Create Credentials component

Shows a randomly generated card number that does not already exist in the server with the help of Card Num Service. It has a form that asks for the user to input their pin number twice. It also has a submit button. When the user presses the submit button, it checks to make sure both pin numbers are the same and not empty. It has an event emitter onCreateCredentials which emits an Account object it just created with the card number and pin number.

## Main account page component

It always shows the go back button at the bottom right. It always takes you back to the previous view. There are variables which decide which view to show the user. If showMoneyAcc an showTransactionHistory are both false, than it shows the main view which is just chequeings, savings and transaction history buttons which the user can click. If the chequeings or savings button is clicked showMoneyAccount is true and money account page html is then displayed. If transaction history button is clicked showTransactionHistory is true and transactionHistoryPage html is then displayed. It uses Storage Service, so, if user refreshes page, showMoneyAcc and showTransactionHistory can be retrieved back from storage service. It also has a subscription for UiService. The subscription lets us always get the correct value for transactionInProgress variable. If there is a transaction in progress then if the user presses the close button showMoneyAcc should still be true because it is the previous page. If it is not true than showMoneyAcc should be false because the page with just chequeings, savings and transaction history buttons are the previous page.

## Money account page component

It displays the account type (chequeings or savings), the balance, a deposit button and a withdraw button if showTransactionPage is false. Otherwise a transaction is in progress and it displays the html in transactionPage component. It uses UiService to make sure the main account page component also knows if a transaction is in progress or not since the close button needs to know an it is located in the main account page html. It uses Storage Service to store or retrieve necessary variables like accountType (chequeings or savings) an transactionType (withdraw or deposit). It uses Account Service to get the logged in account, so, it can display the amount in chequeings or savings to the screen. If TransactionPage component onCompletedTransaction emits then it changes showTransactionPage to false since the transaction is done and it tells UiService too.

## Transaction Page component

It contains a form which asks for the amount they want to deposit/withdraw in chequeings/savings and has a submit input button. It uses account Service to get the logged in account an storage service to get accountType and transactionType. If the user presses submit it checks if an amount was inputted if not it alerts the user that there was no amount inputted. It checks if there is enough money in the bank for this transaction if it was a withdrawal. It records the transaction if it was valid and then uses account service to update the database. It has an event emitter onCompletedTransaction which emits and tells Money Account Page component that the transaction is done.

## Transaction History Page component

It uses account service to get the logged in accounts list of all transactions it has made. It has a metho which gets the last 20 transactions this account has made. It then passes each transaction into the transaction item component.

## Transaction Item component

It takes in a Transaction. It displays all the fields that a Transaction has to the screen.

## Button component

It makes a button and takes in the text to be displayed in the button and the color the button should be.

## Header component

It creates a green border on top and displays a blue bank symbol after the “Angular Bank”

## SideBar component

It creates side bars.

# Services

## Account Service

This can getAccounts() from the JSON server. It can get and set the current logged in account. CreateAccount() adds a new account to the server. updateAcc() switches out the old account on the server with an account with the updated information. It uses Storage Service and gets the logged in account from there if it exists.

## Storage Service

It has store and retrieve functions for al variables the program must remember. If the user refreshes the page, the variables will be lost, so, Storage Service stores these in localStorage. If the user has not reached the part of the program where these variables get values and they are not stored in local storage then an LocalStorageError object is passed. Whenever another service or component use this service, they always check if a LocalStorageError object was passed instead of what they were looking for and act appropriately.

## Ui Service

It is responsible for making sure main account page component and money account page component both are aware if there is a transaction in progress, so, they display the correct divs.

## Card Num Service

It generates a card number between 1000 0000 – 9999 9999 that does not already exist in the server.

# Extra app.module.ts imports

Import FormsModule since some components need to use a form to get data from user.

Import HttpClientModule since the account service needs to be able to access and alter the JSON database

# Extra documentation

You can use Compodoc to see the variables and methods each component and service have. tsconfig.doc.json has already been created.

npm install -g @compodoc/compodoc

compodoc -p tsconfig.doc.json -s