```
pragma solidity ^0.5.9;

contract KYCContract {
    address admin;

    /*
    Struct for a customer
     */
    struct Customer {
        string userName; //unique
        string data_hash; //unique
        uint256 rating;
        uint8 upvotes;
        address bank;
    }

    /*
    Struct for a Bank
     */
    struct Bank {
        string bankName;
        address ethAddress; //unique
        uint256 rating;
        uint8 kyc_count;
        string regNumber; //unique
    }

    /*
    Struct for a KYC Request
     */
    struct KYCRequest {
        string userName;
        string data_hash; //unique
        address bank;
        bool isAllowed;
    }

    /*
    Mapping a customer's username to the Customer struct
    We also keep an array of all keys of the mapping to be able to loop through them when required.
     */
    mapping(string => Customer) customers;
    string[] customerNames;

    /*
    Final customer list, mapping of customer's username to the customer Struct
    */
    mapping(string => Customer) final_customers;
    string[] final_customerNames;

    /*
    Mapping a bank's address to the Bank Struct
    We also keep an array of all keys of the mapping to be able to loop through them when required.
     */
    mapping(address => Bank) banks;
    address[] bankAddresses;

    /*
    Mapping a customer's Data Hash to KYC request captured for that customer.
    This mapping is used to keep track of every kycRequest initiated for every customer by a bank.
     */
    mapping(string => KYCRequest) kycRequests;
    string[] customerDataList;

    /*
    Mapping a customer's user name with a bank's address
    This mapping is used to keep track of every upvote given by a bank to a customer
     */
    mapping(string => mapping(address => uint256)) upvotes;

```

```
71      /**
72       * Constructor of the contract.
73       * We save the contract's admin as the account which deployed this contract.
74       */
75      constructor() public {
76          admin = msg.sender;
77      }
78
79      /** project function 3 and 4
80       * Record a new KYC request on behalf of a customer
81       * The sender of message call is the bank itself
82       * @param  {string} _userName The name of the customer for whom KYC is to be done
83       * @param  {address} _bankEthAddress The ethAddress of the bank issuing this
         request
84       * @return {bool}        True if this function execution was successful
85       */
86      function addKycRequest(string memory _userName, string memory _customerData)
87          public
88          returns (uint8)
89      {
90          // Check that the user's KYC has not been done before, the Bank is a valid
            bank and it is allowed to perform KYC.
91
92          //checking if the bank is a vaid Bank
93
94          for (uint256 i = 0; i < bankAddresses.length; i++) {
95              if (msg.sender == bankAddresses[i]) {
96                  //checking if the customer KYC request alreay exist
97                  require(
98                      !(kycRequests[_customerData].bank == msg.sender),
99                      "This user already has a KYC request with same data in process."
100                 );
101                 kycRequests[_customerData].data_hash = _customerData;
102                 kycRequests[_customerData].userName = _userName;
103                 kycRequests[_customerData].bank = msg.sender;
104
105                 //incrementing the kyc_count for the bank
106                 banks[msg.sender].kyc_count++;
107
108                 //checking if the BANK is a trusted bank to add KYC requests
109                 if (banks[msg.sender].rating <= 50) {
110                     kycRequests[_customerData].isAllowed = false;
111                 } else {
112                     kycRequests[_customerData].isAllowed = true;
113                 }
114                 customerDataList.push(_customerData);
115                 return 1;
116             }
117         }
118         return 0; // 0 is returned in case of failure
119     }
120
121     /** project function 2
122      * Add a new customer
123      * @param {string} _userName Name of the customer to be added
124      * @param {string} _hash Hash of the customer's ID submitted for KYC
125      */
126     function addCustomer(string memory _userName, string memory _customerData)
127         public
128         returns (uint8)
129     {
130         //checking if the bank is a vaid Bank
131         for (uint256 i = 0; i < bankAddresses.length; i++) {
132             if (msg.sender == bankAddresses[i]) {
133                 //checking if the customerdata hash is valid
134                 for (uint256 k = 0; k < customerDataList.length; k++) {
135                     if (stringsEquals(customerDataList[k], _customerData)) {
136                         require(
137                             customers[_userName].bank == address(0),
138                             "This customer is already present, modifyCustomer to
                            edit the customer data"
139                         );
140                         require(
```

```
                            kycRequests[_customerData].isAllowed == true,
                            "isAllowed is false, bank is not trusted to perfrom the
                            transaction"
                        );
                        customers[_userName].userName = _userName;
                        customers[_userName].data_hash = _customerData;
                        customers[_userName].bank = msg.sender;
                        customers[_userName].upvotes = 0;
                        customerNames.push(_userName);
                        return 1;
                    }
                }
            }
        }
        return 0; // 0 is returned in case of failure
    }

    /** project function 6
     * Remove or Block KYC request
     * @param  {string} _userName Name of the customer
     * @return {uint8}         A 0 indicates failure, 1 indicates success
     */
    function removeKYCRequest(
        string memory _userName,
        string memory customerData
    ) public returns (uint8) {
        uint8 i = 0;
        //checking if the provided username and customer Data are mapped in
        kycRequests
        require(
            (stringsEquals(kycRequests[customerData].userName, _userName)),
            "Please enter valid UserName and Customer Data Hash"
        );

        //looping through customerDataList and then deleting the kycRequests and
        deleting the customer data hash from customerDataList array
        for (i = 0; i < customerDataList.length; i++) {
            if (stringsEquals(customerDataList[i], customerData)) {
                delete kycRequests[customerData];
                for (uint256 j = i + 1; j < customerDataList.length; j++) {
                    customerDataList[j - 1] = customerDataList[j];
                }
                customerDataList.length = (customerDataList.length - 1);
                return 1;
            }
        }
        return 0; // 0 is returned if no request with the input username is found.
    }

    /** Project function 5
     * Remove or blovk customer information
     * @param  {string} _userName Name of the customer
     * @return {uint8}         A 0 indicates failure, 1 indicates success
     */
    function removeCustomer(string memory _userName) public returns (uint8) {
        //checking if the customer is present in the customers list
        for (uint256 i = 0; i < customerNames.length; i++) {
            if (stringsEquals(customerNames[i], _userName)) {
                delete customers[_userName];
                //removing the customer from customerNames array
                for (uint256 j = i + 1; j < customerNames.length; j++) {
                    customerNames[j - 1] = customerNames[j];
                }
                customerNames.length--;
                return 1;
            }
        }
        return 0;
    }

    /** Project Function 9
     * View customer information
     * @param  {public} _userName Name of the customer
```

```
211         * @return {Customer}         The customer struct as an object
212         */
213        function viewCustomer(string memory _userName, string memory password)
214            public
215            view
216            returns (string memory)
217        {
218            //looping through customerNames to check if the _userName passes is valid
219            for (uint256 i = 0; i < customerNames.length; i++) {
220                if (stringsEquals(customerNames[i], _userName)) {
221                    //looping through passwordSet array, which is an string[] stores
                       USERNAME's of user whose password is set
222                    //if password is set no changes are made to password, if not set
                       then password is assigned a default value = '0'
223                    for (uint256 k = 0; k < passwordSet.length; k++) {
224                        if (stringsEquals(passwordSet[k], _userName)) {
225                            //no changes required
226                            continue;
227                        } else {
228                            password = "0";
229                        }
230                    }
231                }
232            }
233            //passwordStore is a mapping of username=>password, if given username and
               password match we return customer data hash
234            //else error is thrown informing user that password provided didn't match
235
236            if (stringsEquals(passwordStore[_userName], password)) {
237                return customers[_userName].data_hash;
238            } else {
239                return "password provided by the user didn't match";
240            }
241        }
242
243        /** Project Function 9
244         * Add upvote to provide ratings on customers
245         * Add a new upvote from a bank
246         * @param {public} _userName Name of the customer to be upvoted
247         */
248        function upvoteCustomer(string memory _userName) public returns (uint8) {
249            //checking if the customer exist in the customerNames
250            for (uint256 i = 0; i < customerNames.length; i++) {
251                if (stringsEquals(customerNames[i], _userName)) {
252                    require(
253                        upvotes[_userName][msg.sender] == 0,
254                        "This bank have already upvoted this customer"
255                    );
256                    upvotes[_userName][msg.sender] = 1;
257                    customers[_userName].upvotes++;
258
259                    //updating the rating of the customer
260                    customers[_userName].rating =
261                        (customers[_userName].upvotes * 100) /
262                        bankAddresses.length;
263                    //if the customer rating is higher then also adding the customer to
                       the final_customers list.
264                    if (customers[_userName].rating > 50) {
265                        final_customers[_userName].userName = _userName;
266                        final_customers[_userName].data_hash = customers[_userName]
267                            .data_hash;
268                        final_customers[_userName].rating = customers[_userName]
269                            .rating;
270                        final_customers[_userName].upvotes = customers[_userName]
271                            .upvotes;
272                        final_customers[_userName].bank = customers[_userName].bank;
273                        //final_customerNames is array to itterate over customers
274                        final_customerNames.push(_userName);
275                    }
276
277                    return 1;
278                }
279            }
```

```
        return 0;
    }

    /** Project Function 9
     * Edit customer information
     * @param  {public} _userName Name of the customer
     * @param  {public} _hash New hash of the updated ID provided by the customer
     * @return {uint8}         A 0 indicates failure, 1 indicates success
     */
    function modifyCustomer(
        string memory _userName,
        string memory password,
        string memory _newcustomerData
    ) public returns (uint8) {
        //checking if the user exist
        for (uint256 i = 0; i < customerNames.length; i++) {
            if (stringsEquals(customerNames[i], _userName)) {
                for (uint256 k = 0; k < passwordSet.length; k++) {
                    if (stringsEquals(passwordSet[k], _userName)) {
                        continue;
                    } else {
                        password = "0";
                    }
                }

                if (stringsEquals(passwordStore[_userName], password)) {
                    customers[_userName].data_hash = _newcustomerData;
                    customers[_userName].bank = msg.sender;
                    //after modifying customer data removing them from the
                    final_customers list and final_customerNames array
                    for (uint8 j = 0; i < final_customerNames.length; j++) {
                        if (stringsEquals(final_customerNames[i], _userName)) {
                            delete final_customers[_userName];
                            customers[_userName].rating = 0;
                            customers[_userName].upvotes = 0;

                            for (
                                uint256 k = i + 1;
                                j < final_customerNames.length;
                                k++
                            ) {
                                final_customerNames[j -
                                    1] = final_customerNames[j];
                            }
                            final_customerNames.length--;
                        }
                    }
                    return 1;
                }
            }
        }
        return 0;
    }

    /* Project function 4
    get bank requests
    Parameters : Unique bankAddress and Index which will rertun 1 of the yet to be
    validated requests.
    Returns : Will rertun KYC_UnValidated[index]
    */
    //Array to count number of invalidated KYC requests and store its customer data
    hash.
    string[] KYC_UnValidatedCount;

    function getBankRequset(address bankAddress, uint256 index)
        public
        returns (
            string memory,
            string memory,
            address,
            bool
        )
    {
```

```
350         //looping through bankAddresses array to check if the passed bankAddress is
            valid
351
352         for (uint256 i = 0; i < bankAddresses.length; i++) {
353             if (bankAddresses[i] == bankAddress) {
354                 //looping through customerDataList to find all the KYC requests
                    initiated by the bank whose address is passed
355                 for (uint256 k = 0; k < customerDataList.length; k++) {
356                     //kycRequests whose isAllowed value is False and
                        bankAddress==bankAddress passed as Parameter
357                     //store it in KYC_UnValidatedCount array.
358
359                     if (
360                         (kycRequests[customerDataList[k]].bank ==
361                             bankAddress) &&
362                         (kycRequests[customerDataList[k]].isAllowed == false)
363                     ) {
364                         KYC_UnValidatedCount.push(customerDataList[k]);
365                     }
366                 }
367             }
368         }
369         return (
370             kycRequests[KYC_UnValidatedCount[index]].userName,
371             kycRequests[KYC_UnValidatedCount[index]].data_hash,
372             kycRequests[KYC_UnValidatedCount[index]].bank,
373             kycRequests[KYC_UnValidatedCount[index]].isAllowed
374         );
375     }
376
377     /*
378     Upvotes to provide rating on other banks
379     */
380     mapping(address => mapping(address => uint256)) upvotesBank;
381     mapping(address => uint256) upvoteCount;
382
383     function upvoteBank(address bankAddress) public returns (uint8) {
384         //checking if the bank exist
385         for (uint256 i = 0; i < bankAddresses.length; i++) {
386             if (msg.sender == bankAddresses[i]) {
387                 require(
388                     upvotesBank[bankAddress][msg.sender] == 0,
389                     "You have already upvoted this bank"
390                 );
391                 upvotesBank[bankAddress][msg.sender] = 1;
392                 upvoteCount[bankAddress]++;
393                 banks[bankAddress].rating =
394                     (upvoteCount[bankAddress] * 100) /
395                     bankAddresses.length;
396
397                 return 0;
398             }
399         }
400         return 1;
401     }
402
403     /*
404     Get customer rating
405     */
406
407     function getCustomerRating(string memory userName)
408         public
409         view
410         returns (uint256)
411     {
412         for (uint256 i = 0; i < customerNames.length; i++) {
413             if (stringsEquals(customerNames[i], userName))
414                 return customers[userName].rating;
415         }
416     }
417
418     /*
419     Get bank Rating
```

```
420       */
421       //checking if the bank exist
422       function getBankRating(address bankAddress) public view returns (uint256) {
423           for (uint256 i = 0; i < bankAddresses.length; i++) {
424               if (bankAddresses[i] == bankAddress) {
425                   return banks[bankAddress].rating;
426               }
427           }
428       }
429
430       /*
431       Retrieve access history for a resource
432       */
433       function retrieveHistory(string memory userName)
434           public
435           view
436           returns (address)
437       {
438           for (uint256 i = 0; i < customerNames.length; i++) {
439               if (stringsEquals(customerNames[i], userName)) {
440                   return customers[userName].bank;
441               }
442           }
443       }
444
445       /*
446       Set password
447       */
448       //mapping of username to passwordStore
449       mapping(string => string) public passwordStore;
450       string[] public passwordSet;
451
452       function setPassword(string memory userName, string memory password)
453           public
454           returns (bool)
455       {
456           //checking if the user exist
457           for (uint256 i = 0; i < customerNames.length; i++) {
458               if (stringsEquals(customerNames[i], userName)) {
459                   passwordStore[userName] = password;
460                   //adding username to passwordSet array to itterate over user whose
                      passwords are set
461                   passwordSet.push(userName);
462                   return true;
463               }
464           }
465       }
466
467       /*
468       Get Bank Details
469       */
470       function getBankDetail(address bankAddress)
471           public
472           view
473           returns (
474               string memory,
475               address,
476               uint256,
477               uint8,
478               string memory
479           )
480       {
481           //checking if bank exist
482           for (uint256 i = 0; i < bankAddresses.length; i++) {
483               if (bankAddresses[i] == bankAddress) {
484                   return (
485                       banks[bankAddress].bankName,
486                       banks[bankAddress].ethAddress,
487                       banks[bankAddress].rating,
488                       banks[bankAddress].kyc_count,
489                       banks[bankAddress].regNumber
490                   );
491               }
```

```
        }
    }

    /* Project Function 7 and 8
    Add Bank to the smart contract
    add bank = bank can only be added by the admin
    admin = account which is deploying smart contract
    mapping to store bankRegistration => bank address
    */

    mapping(string => address) bankRegStore;

    function addBank(
        string memory bankName,
        address bankAddress,
        string memory bankRegistration
    ) public returns (string memory) {
        //checking if the account used to perform add operation is an Admin
        require(msg.sender == admin, "You are not an admin");
        require(
            banks[bankAddress].ethAddress == address(0),
            "This bank is already added to the samrt contract"
        );
        //making sure that the registration number is unique
        require(
            bankRegStore[bankRegistration] == address(0),
            "This Registration number is already assocaited with another bank"
        );
        //adding bank
        banks[bankAddress].bankName = bankName;
        banks[bankAddress].ethAddress = bankAddress;
        banks[bankAddress].rating = 0;
        banks[bankAddress].kyc_count = 0;
        banks[bankAddress].regNumber = bankRegistration;

        bankAddresses.push(bankAddress);
        bankRegStore[bankRegistration] = bankAddress;
        return "successful entry of bank to the contract";
    }

    /* Project function 5 and 6
    Remove Bank from the smart contract
    remove bank = bank can only be removed by the admin
    admin = account which is deploying smart contract
    */
    function removeBank(address bankAddress) public returns (string memory) {
        //checking if the account used to perform remove operation is an Admin
        require(msg.sender == admin, "You are not an admin");
        for (uint256 i = 0; i < bankAddresses.length; i++) {
            if (bankAddresses[i] == bankAddress) {
                delete banks[bankAddress];
                for (uint256 j = i + 1; j < bankAddresses.length; j++) {
                    bankAddresses[j - 1] = bankAddresses[j];
                }
                bankAddresses.length--;
                return "successful removal of the bank from the contract.";
            }
        }

        return "The bank is already removed from the contract";
    }

    // if you are using string, you can use the following function to compare two
    strings
    // function to compare two string value
    // This is an internal fucntion to compare string values
    // @Params - String a and String b are passed as Parameters
    // @return - This function returns true if strings are matched and false if the
    strings are not matching
    function stringsEquals(string storage _a, string memory _b)
        internal
        view
        returns (bool)
```

```
563         {
564             bytes storage a = bytes(_a);
565             bytes memory b = bytes(_b);
566             if (a.length != b.length) return false;
567             // @todo unroll this loop
568             for (uint256 i = 0; i < a.length; i++) {
569                 if (a[i] != b[i]) return false;
570             }
571             return true;
572         }
573     }
574
```