**INTRODUCTION**

The present interest of the project The banking institutions have an important role within the financial sector, for the well functioning of economic units and of the overall economy. Furthermore, the creation of a functional banking system has become a permanent preoccupation, one capable of offering a wide range of products and services, in order to satisfy the requirements of all potential clients, from the very beginning of banking up to the present day. This project is specifically focuses on the banking system as a compulsory segment of the economic ensemble. Taking into account the financial and economic reality, both internationally and internally, each country and Romania as well is interested in creating a solid banking system, which would assure the proper organizational framework for the development of financial mechanisms.

I have chosen this project considering the impact of the banking system upon the whole economy and considering the fact that a market economy cannot function without profitable and consolidated banks. Once the economy developed and the business environment improved, the Romanian banking system evolved exponentially. The determinant factors for the New Zealand economy – transition, integration and globalization – offer important opportunities for increasing the economic-financial performances, obtaining supplementary profits, but also implying major risks.

Once the effects of the crisis manifested and extended, the existence of some unhealthy banking practices has revealed. The lack or insufficient coverage of certain highly volatile capital market segments has proved, as well as the insufficient control of hybrid financial products and inadequate risk management practices within banks. The present interest of this paper lies in the need to create a banking system analysis that would provide information, arguments and necessary solutions in order to avoid triggering certain events leading to the vulnerability of the banking sector.

**PART I - PROJECT PROPOSAL**

**PROJECT GOAL**:-

Enables the customers to perform the basic banking transactions by sitting at their office or at homes through PC or laptop

**PROJECT OBJECTIVES:-**

The objectives of the research The following objectives are established:-

1. To analyze the NZ banking system in the context of globalization and NZ Union integration;
2. To establish the bank management strategies in the context of NZ Union integration and its prospects
3. To identify and analyze the key financial performance indicators within the NZ banking system
4. To analyze the key financial performance indicators within some banking institutions
5. To determining the specifics, the identification principles and the categories of banking risks
6. To determine the content, the structure and the methods of identification, evaluation and control of banking risks, especially for the credit risk, the liquidity risk, the market risk and the operational risk.
7. To analyze the liquidity of a banking institution
8. To analyze the requirements of the banking supervisory units, and the Basel Committee agreements
9. To analyze some cases of failures and banking crises, focusing on identifying the causes that led to bank bankruptcies
10. To analyze the evolution of the financial crisis and its impact upon world banking.

The bank account management system is an application for maintaining a person's account in a bank. In this project, I tried to show the working of a banking account system and cover the basic functionality of a bank account management system.

This enables the customers to perform the basic banking transactions by sitting at their office or at homes through PC or laptop. The system provides the access to the customer to create an account, deposit/withdraw the cash from his account, also to view last ten transactions list of account. It can also add their beneficiary and transfer funds to their beneficiary. The customers can access the banks web app for viewing their Account details and perform the transactions on account as per their requirements.

Now a day’s, managing a bank is tedious job up to certain limit. So software that reduces the work is essential. Also today’s world is a genuine computer world and is getting faster. Thus, considering above necessities, the software for bank management has became necessary which would be useful in managing the bank more efficiently. All transactions are carried out online by transferring from accounts in the same Bank or different bank. The software is meant to overcome the drawbacks of the manual system.

1) **System Analysis:-**

‘Internet banking’ or ‘online banking’ is a system which allow individuals to perform regular banking activities via Internet. Customers can keep tracks of all the banking issues by accessing their account online at their own convenience and have no worries about stucked in a traffic jam or queing up in a long line of customers at the counter for services. By using internet banking, Account information can be accessed anytime, day or night, and can be done from anywhere. Internet banking provides 24 hours services and enable customers to perform all routine transactions, fund transfer, avail banking services like sms service, email banking statement, add beneficiaries and display the last 6 days processed transactions on the graph chart. Almost all the banks in New Zealand have their own secured website for internet banking transactions which enable customers to perform routine banking transaction as mentioned above. Although the concept and functions of each bank are similar, the systems are varies and designed according to every banks system requirements.

**2) System Design:-**

Many organizations consider information systems in order to produce useful information by capturing and managing data to employees, customers, suppliers and partners. It is necessary for their ability to compute or gain competitive advantage. Information systems can be classified by the functions such as

1) Transaction processing systems

2) Management information systems

3) Decision support systems

4) Executive information systems

5) Expert systems

6) Communication and collaborative systems

7) Office automation systems.

**System Design in Internet Banking:-**

Steps of System design in Internet banking are as follow:

1. Firstly, the customer needs to request the URL.
2. For Customer login the system, then the system checks User Name and Password.
3. If it is valid, then the customer can see the Main Dashboard page of the Internet banking website.
4. Then the customer can choose from many menus such as viewing Account Information, Funds Transfer, Manage Beneficiaries, Calculate Interest Rate, Apply Checkbook, Update Profile and so on.
5. For example, if the customer chooses the Funds transfer menu, then the customer need to select Funds Transfer type such as Funds Transfer to My own A/C, Funds Transfer to Other A/C and Funds Transfer to added beneficiaries.
6. Then customer needs to choose From Account, To Account and Amount.
7. After that, submit this information to the system.
8. And then ensure the detailed information and click Confirm button to accomplish the transaction.
9. Key in his/her Transaction password for Secure PIN to complete this transaction
10. After logout, customer needs to clear cache for security reason.

**3) Programming:-**

**a) Used Angular 7.1.4 as frontend language**

Main benefits that the technology suggests to the engineering community, including the component-based architecture, mobile-oriented philosophy, performance, and some other characteristics that make Angular stand out.

### 1) Component-based architecture that provides a higher quality of code

Components can be thought of as small pieces of an interface that are independent of each other. Imagine that you have a simple application with a list of items and a corresponding search box to retrieve the items by word matches. The box with listed names, the search box, and the main sheet where the other two boxes are placed are all considered separate components in Angular.

**2)** **Reusability**

Components of a similar nature are well encapsulated, in other words, self-sufficient. Developers can reuse them across different parts of an application. This is particularly useful in enterprise-scope applications where different systems converge but may have many similar elements like search boxes, date pickers, sorting lists, etc.

**3) Readability**

Encapsulation also ensures that new developers – who’ve been recently on boarded to a project – can read code better and eventually reach their plateau of productivity faster.

**4) Unit-test friendly**

The independent nature of components simplifies unit tests, quality assurance procedures aimed at verifying the performance of the smallest parts of the application, units

.

**5) Maintainability**

Components that are easily decoupled from each other can be easily replaced with better implementations. Basically, your engineering team will be more efficient in maintaining and updating the code within the iterative development workflow.

**PROJECT SCOPE:-**

**1) Context :- Create online banking system for the end to end user.**

From an end-user perspective, the Online Banking System Project consists of two functional elements: End User and Customer transaction module.

**End User Module**

For the end user point of view, anyone can view the number of facilities we provide if anyone open account in our site. It can send email to admin by filling the contact form. View number of team members and our success story from the past 10 years.

### Customer Transaction Module

An enhanced atomized system is developed to maintain customer transaction. Features includes

* Creation of new banking customer
* Customer Creation Form.
* Existing customer details
* Customer Access Form
* Banking Main menu option like.
* Transaction – Debit, Credit, Transfer
* Customer Detail – Modify Details
* Help – User Manual.
* Transaction Summary
* Account Closing

**2)** **Target language platform : Used Angular 7.0 as frontend and .NET Web API as backend.**

**3) Stakeholders of the product: Anyone can become our stakeholder for the banking application. It provides number of banking features just on by single click**

**4) What you will NOT be doing : We have some menus which currently have just simple entries. Not dynamically get data from database.**

**PROJECT DELIVERABLES:-**

**1) Analysis Documentation:-** The project that we have undertaken aims to develop a banking system that is clean, user-friendly and multi-functional. Development of this application includes a number of fields such that user feels comfortable and the system appears as dynamic to him. The project “Banking System” includes the following functionalities:

* Transactions can be done with minimum user events.
* All transactional details and accounts are stored in database files on stable storage.
* Customers can view their own account details and cause them as necessary
* Customer can inquire an account and can inquire about interest
* All customer’s data are stored in files on a stable storage
* Account holders have to pass through a login system to enter their accounts
* This system possess password-protected administrative access; thus preventing the whole management system from unauthorized access
* Customer can get help in any time he/she wants

**2) Design Documentation:-** The application will be extremely beneficial for the Customers intending to use and operate their bank account and will get various benefits in the field of management of accounts on a clean and user-friendly platform.

* Complex Banking operations and Transaction operations are efficiently handled by the application
* It is cost effective
* It has ease of use along with complete reference
* It is highly secured and less time consuming; hence time wastage can be avoided
* Up to date records of the customers are maintained by the authority.

**3) Executable Program:-** Below list of some code snippets for the logic part of the banking application.

a) User Model used for the save entries of registered customer.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace EBankingMain.Models

{

public class UsersDto

{

public int id { get; set; }

public string firstName { get; set; }

public string lastName { get; set; }

public string userName { get; set; }

public string password { get; set; }

public string newPassword { get; set; }

public string transactionPassword { get; set; }

public string newTransactionPassword { get; set; }

public string holderName { get; set; }

public string accountNumber { get; set; }

public DateTime createdOn { get; set; }

public string image { get; set; }

public string contactNumber { get; set; }

public string email { get; set; }

public DateTime lastLogin { get; set; }

public string location { get; set; }

public string gender { get; set; }

public DateTime dateofbirth { get; set; }

public string randomText { get; set; }

}

}

b) Controller action used to save the new registered user.

namespace EBankingMain.Controllers

{

[EnableCors(origins: "\*", headers: "\*", methods: "\*")]

public class RegistrationController : ApiController

{

eBankingEntities entity = null;

public RegistrationController()

{

entity = new eBankingEntities();

}

[HttpPost]

public IHttpActionResult Register(UsersDto usersDto)

{

var currentRecord = entity.users.Where(x => x.email == usersDto.email).ToList();

if(currentRecord.Count > 0)

{

return Ok("email");

}

var currentRecordUser = entity.users.Where(x => x.username == usersDto.userName).ToList();

if (currentRecordUser.Count > 0)

{

return Ok("username");

}

else

{

string token = Guid.NewGuid().ToString().Substring(0, 10);

user addUser = new user

{

firstname = usersDto.firstName,

lastname = usersDto.lastName,

gender = usersDto.gender,

location = usersDto.location,

username = usersDto.userName,

password = usersDto.password,

transactionpassword = usersDto.password,

holdername = usersDto.holderName,

accountnumber = usersDto.accountNumber,

createdon = DateTime.Now,

image = "profile.png",

contactnumber = usersDto.contactNumber,

email = usersDto.email,

accountbalance = "100000"

};

entity.users.Add(addUser);

entity.SaveChanges();

return Ok(addUser);

}

}

}

}

c) Beneficiary Model Used for the adding new beneficiaries.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace EBankingMain.Models

{

public class BeneficiaryDto

{

public int id { get; set; }

public string name { get; set; }

public string nickName { get; set; }

public string accountNumber { get; set; }

public string ifscCode { get; set; }

public string maxAmount { get; set; }

public string maxTransactions { get; set; }

public string address { get; set; }

public string usersAccountNumber { get; set; }

}

}

d) Transaction Model for transaction funds.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace EBankingMain.Models

{

public class TransactionsDto

{

public int id { get; set; }

public string name { get; set; }

public string nickName { get; set; }

public string senderaccountnumber { get; set; }

public string receiveraccountnumber { get; set; }

public string ifscCode { get; set; }

public DateTime transactiondate { get; set; }

public string creditamount { get; set; }

public string debitamount { get; set; }

}

}

e) Message Model used while sending the contact message to admin.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace EBankingMain.Models

{

public class MessageDto

{

public string name { get; set; }

public string email { get; set; }

public string subject { get; set; }

public string message { get; set; }

public string messageType { get; set; }

}

}

f) Account Controller list actions.

using EBankingMain.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web.Http;

using System.Web.Http.Cors;

using SendGrid;

using SendGrid.Helpers.Mail;

using System.Web;

using System.Diagnostics;

using System.Net.Http.Headers;

using System.IO;

using System.Web.Script.Serialization;

using bankingsystem;

using System.Data.Objects;

namespace EBankingMain.Controllers

{

[EnableCors(origins: "\*", headers: "\*", methods: "\*")]

public class AccountController : ApiController

{

eBankingEntities entity = null;

public AccountController()

{

entity = new eBankingEntities();

}

[HttpPost]

public IHttpActionResult AddBeneficiary(BeneficiaryDto beneficiaryDto)

{

beneficiary addBeneficiary = new beneficiary

{

name = beneficiaryDto.name,

nickname = beneficiaryDto.nickName,

accountnumber = beneficiaryDto.accountNumber,

ifsccode = beneficiaryDto.ifscCode,

maxamount = beneficiaryDto.maxAmount,

maxtransactions = beneficiaryDto.maxTransactions,

address = beneficiaryDto.address,

usersaccountnumber = beneficiaryDto.usersAccountNumber

};

entity.beneficiaries.Add(addBeneficiary);

entity.SaveChanges();

return Ok(addBeneficiary);

}

[HttpGet]

public IHttpActionResult ViewBeneficiary(string accountNumber)

{

var beneficiaries = entity.beneficiaries.Where(x => x.usersaccountnumber == accountNumber).ToList();

return Ok(beneficiaries);

}

[HttpGet]

public IHttpActionResult CheckAccountExists(string accountNumber)

{

var isAccountExists = entity.users.Where(x => x.accountnumber == accountNumber).ToList();

return Ok(isAccountExists.Count);

}

[HttpGet]

public IHttpActionResult AddTransactionsForBankServices(string accountNumber, string message, string amount)

{

transaction addTransaction = new transaction();

var checkSenderTransactions = entity.users.Where(x => x.accountnumber == accountNumber).OrderByDescending(x => x.id).First();

var currentSenderBalance = checkSenderTransactions.accountbalance;

try

{

addTransaction.name = message;

addTransaction.senderaccountnumber = accountNumber;

addTransaction.receiveraccountnumber = accountNumber;

addTransaction.transactiondate = DateTime.Now;

addTransaction.creditamount = "0";

addTransaction.debitamount = amount;

addTransaction.senderbalanceamount = (Convert.ToDouble(currentSenderBalance) - Convert.ToDouble(amount)).ToString();

addTransaction.receiverbalanceamount = "0";

entity.transactions.Add(addTransaction);

entity.SaveChanges();

var userAccount = entity.users.First(x => x.accountnumber == accountNumber);

userAccount.accountbalance = addTransaction.senderbalanceamount;

entity.SaveChanges();

return Ok(addTransaction);

}

catch (Exception ex)

{

return Ok("Error");

}

}

[HttpPost]

public IHttpActionResult AddFirstTransactions(TransactionsDto transactionsDto)

{

transaction addTransaction = new transaction();

try

{

addTransaction.name = transactionsDto.name;

addTransaction.nickname = transactionsDto.nickName;

addTransaction.senderaccountnumber = transactionsDto.senderaccountnumber;

addTransaction.receiveraccountnumber = transactionsDto.receiveraccountnumber;

addTransaction.ifsccode = transactionsDto.ifscCode;

addTransaction.transactiondate = DateTime.Now;

addTransaction.creditamount = transactionsDto.creditamount;

addTransaction.debitamount = transactionsDto.debitamount;

addTransaction.senderbalanceamount = transactionsDto.creditamount;

addTransaction.receiverbalanceamount = "0";

entity.transactions.Add(addTransaction);

entity.SaveChanges();

return Ok(addTransaction);

}

catch (Exception ex)

{

return Ok("Error");

}

}

[HttpPost]

public IHttpActionResult AddTransactionsForSameBank(TransactionsDto transactionsDto)

{

transaction addTransaction = new transaction();

string currentSenderBalance = "";

string currentReceiverBalance = "";

try

{

var checkSenderTransactions = entity.users.Where(x => x.accountnumber == transactionsDto.senderaccountnumber).OrderByDescending(x=>x.id).First();

currentSenderBalance = checkSenderTransactions.accountbalance;

var checkReceiverTransactions = entity.users.Where(x => x.accountnumber == transactionsDto.receiveraccountnumber).OrderByDescending(x => x.id).First();

currentReceiverBalance = checkReceiverTransactions.accountbalance;

}

catch (Exception ex)

{

currentSenderBalance = "0";

currentReceiverBalance = "0";

}

finally

{

addTransaction.name = transactionsDto.name;

addTransaction.nickname = transactionsDto.nickName;

addTransaction.senderaccountnumber = transactionsDto.senderaccountnumber;

addTransaction.receiveraccountnumber = transactionsDto.receiveraccountnumber;

addTransaction.ifsccode = transactionsDto.ifscCode;

addTransaction.transactiondate = DateTime.Now;

addTransaction.creditamount = transactionsDto.creditamount;

addTransaction.debitamount = transactionsDto.debitamount;

addTransaction.senderbalanceamount = ((Convert.ToDouble(currentSenderBalance) + Convert.ToDouble(transactionsDto.creditamount)) - Convert.ToDouble(transactionsDto.debitamount)).ToString();

addTransaction.receiverbalanceamount = ((Convert.ToDouble(currentReceiverBalance) + Convert.ToDouble(transactionsDto.debitamount))).ToString();

entity.transactions.Add(addTransaction);

entity.SaveChanges();

var userAccount = entity.users.First(x => x.accountnumber == transactionsDto.senderaccountnumber);

userAccount.accountbalance = addTransaction.senderbalanceamount;

entity.SaveChanges();

var userreceiverAccount = entity.users.First(x => x.accountnumber == transactionsDto.receiveraccountnumber);

userreceiverAccount.accountbalance = addTransaction.receiverbalanceamount;

entity.SaveChanges();

}

return Ok(addTransaction);

}

[HttpPost]

public IHttpActionResult AddTransactionsForDifferentBank(TransactionsDto transactionsDto)

{

transaction addTransaction = new transaction();

string currentSenderBalance = "";

try

{

var checkSenderTransactions = entity.users.Where(x => x.accountnumber == transactionsDto.senderaccountnumber).OrderByDescending(x => x.id).First();

currentSenderBalance = checkSenderTransactions.accountbalance;

}

catch (Exception ex)

{

currentSenderBalance = "0";

}

finally

{

addTransaction.name = transactionsDto.name;

addTransaction.nickname = transactionsDto.nickName;

addTransaction.senderaccountnumber = transactionsDto.senderaccountnumber;

addTransaction.receiveraccountnumber = transactionsDto.receiveraccountnumber;

addTransaction.ifsccode = transactionsDto.ifscCode;

addTransaction.transactiondate = DateTime.Now;

addTransaction.creditamount = transactionsDto.creditamount;

addTransaction.debitamount = transactionsDto.debitamount;

addTransaction.senderbalanceamount = ((Convert.ToDouble(currentSenderBalance) + Convert.ToDouble(transactionsDto.creditamount)) - Convert.ToDouble(transactionsDto.debitamount)).ToString();

entity.transactions.Add(addTransaction);

entity.SaveChanges();

var userAccount = entity.users.First(x => x.accountnumber == transactionsDto.senderaccountnumber);

userAccount.accountbalance = addTransaction.senderbalanceamount;

entity.SaveChanges();

}

return Ok(addTransaction);

}

[HttpPost]

public IHttpActionResult IssueCheckBook(CheckbookDto checkbookDto)

{

checkbook addCheckbook = new checkbook

{

usersaccountnumber = checkbookDto.usersAccountNumber,

issuedate = DateTime.Now,

numberofpages = checkbookDto.numberOfPages,

delivereddate = DateTime.Now.AddDays(7),

isdelivered = false

};

entity.checkbooks.Add(addCheckbook);

entity.SaveChanges();

return Ok(addCheckbook);

}

[HttpPost]

public IHttpActionResult UpdateCheckBookDeliveryStatus(CheckbookDto checkbookDto)

{

var checkBook = entity.checkbooks.First(x => x.usersaccountnumber == checkbookDto.usersAccountNumber);

checkBook.isdelivered = checkbookDto.status;

entity.SaveChanges();

return Ok(checkBook);

}

[HttpGet]

public IHttpActionResult ViewIssuedCheckBook(string accountNumber)

{

var checkBook = entity.checkbooks.Where(x => x.usersaccountnumber == accountNumber).ToList();

return Ok(checkBook);

}

[HttpPost]

public IHttpActionResult ApplyBankServices(ServicesDto servicesDto)

{

bankservice addBankService = null;

try

{

addBankService = entity.bankservices.First(x => x.usersaccountnumber == servicesDto.usersAccountNumber);

addBankService.services = servicesDto.services;

entity.SaveChanges();

return Ok(addBankService);

}

catch(Exception ex)

{

addBankService = new bankservice

{

usersaccountnumber = servicesDto.usersAccountNumber,

services = servicesDto.services

};

entity.bankservices.Add(addBankService);

entity.SaveChanges();

return Ok(addBankService);

}

}

[HttpGet]

public IHttpActionResult ViewAppliedBankService(string accountNumber)

{

var bankservices = entity.bankservices.Where(x => x.usersaccountnumber == accountNumber).ToList();

return Ok(bankservices);

}

[HttpGet]

public IHttpActionResult GetTransactions(string accountNumber, string startDate, string endDate)

{

DateTime startDateTime = Convert.ToDateTime(startDate);

DateTime endDateTime = Convert.ToDateTime(endDate);

var record = (from log in entity.transactions

where (EntityFunctions.TruncateTime(log.transactiondate) >= EntityFunctions.TruncateTime(startDateTime.Date)

&& EntityFunctions.TruncateTime(log.transactiondate) <= EntityFunctions.TruncateTime(endDateTime.Date))

&& ((log.senderaccountnumber == accountNumber && log.senderaccountnumber != "" && log.receiveraccountnumber != "") || (log.receiveraccountnumber == accountNumber && log.receiveraccountnumber != "" && log.senderaccountnumber != ""))

select log);

return Ok(record);

}

[HttpGet]

public IHttpActionResult GetLastTenTransactions(string accountNumber)

{

var record = (from log in entity.transactions

where (log.senderaccountnumber == accountNumber && log.senderaccountnumber != "" && log.receiveraccountnumber != "") || (log.receiveraccountnumber == accountNumber && log.receiveraccountnumber != "" && log.senderaccountnumber != "")

orderby EntityFunctions.TruncateTime(log.transactiondate)

select log);

return Ok(record);

}

[HttpGet]

public IHttpActionResult GetFinalLastTenTransactions(string accountNumber)

{

var record = (from log in entity.transactions

where (log.senderaccountnumber == accountNumber && log.senderaccountnumber != "" && log.receiveraccountnumber != "") || (log.receiveraccountnumber == accountNumber && log.receiveraccountnumber != "" && log.senderaccountnumber != "")

orderby log.id descending

select log).Take(10);

return Ok(record);

}

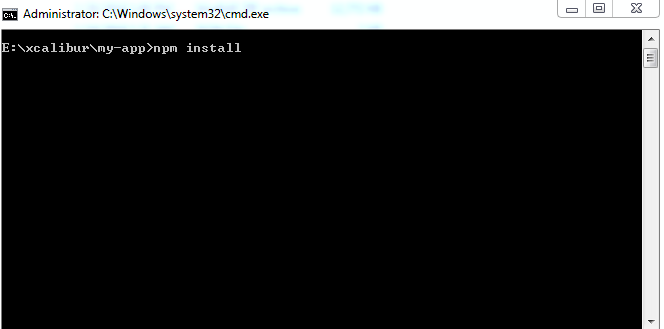
}

}

**4) User Documentation:-**  Below is the list of steps that will be used to perform the action of web application to make it run-able.

a) First of all run the "**bankingsystem.sln**" so that our .NET WEB API is become in running mode.

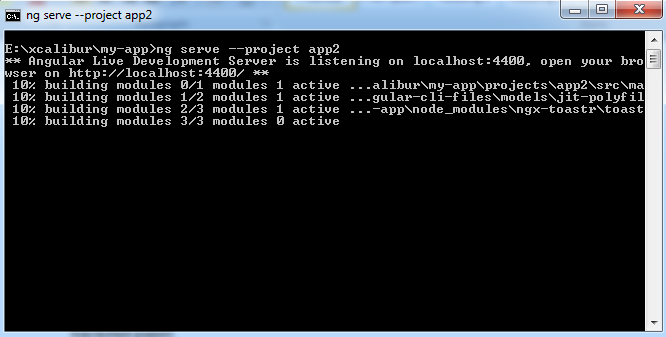
b) Go to the path where our angular app is located and execute the "**npm install**" command so that our used package is going to install on the new workstation.



c) Then execute to command "**ng serve --project app1**" for the execution of frontend panel



d) Then execute the command "**ng serve --project app2**" for the execution of backend panel



e) At last all run the "**webApplication.sln**" so that our web app GUI is going to display in browser.

**5) QUALITY PLAN:-** The testing Scope consists of a series of different tests that will fully exercise the system. The primary purpose of these tests is to uncover the limitations of the systems and measure its full capabilities. A list of the various planned test types and a brief explanation follows.

**A) Functional Testing**

Objective of this test is to ensure that all project functionality works without break. Includes validation Testing-- that is intensive testing of the Front-end fields and screens considering the business requirements. GUI standards; valid, invalid and limit data input; screen & field look and appearance, and overall consistency with the rest of the application. Specific functional testing – test the individual processes and data flows.

**B) Integration Testing**

This test ensures that all the modules are integrated and communicates with each other in the way they are expected to be. This test ensures consistent flow of data across the modules.

**C) Security Testing**

This test ensures that only authorized users can access the system. This check ensures that the user doesn’t have access to any modules, which they are not supposed to have.

**D) Performance Testing**

This test checks the overall performance of the system and response time for loading screens, creating a record, updating the record and deleting the record. This test ensures the response time of the system in multi-user environment.

**E) Test Items**

a). Procedures/Functions

* + - getUser
    - updateProfile
    - getProfile
    - Change Password
    - getUserLoginStatus
    - getAccountDetails
    - createUserProfile
    - updateAccountBalance
    - transferFund
    - Authorization – Customer
    - Log off

b) Code Review

* Complete and proper implementation of design
* Use of standard naming conventions
* Variable and parameter declarations
* Initialization & resetting of variables Checking for code optimization
* Comments and indentation
* Readability of code
* Logic of the code

c) Integration & system testing

* The units within a module are tested to ensure that they work in tandem. Data flow among the units involved and the performance are tested.
* Check for the correct navigation order.
* Ensure the modules within the system function according to the user requirements

**6) EXECUTIVE SUMMARY OF THE PROJECT:-**

**Hardware Description**: The selection of hardware is very important in the existence and proper working of any software. When selecting hardware, the size and requirements are also important.

Minimum Requirements:

Processor : INTEL Core i3

RAM : 3MB

Hard Disk Drive : 250GB

The proposed System is developed on:

Processor : INTEL Core i5

RAM : 4GB

Hard Disk Drive : 500GB

**Software Description:**

Operating System : Windows 7 Service Pack 1

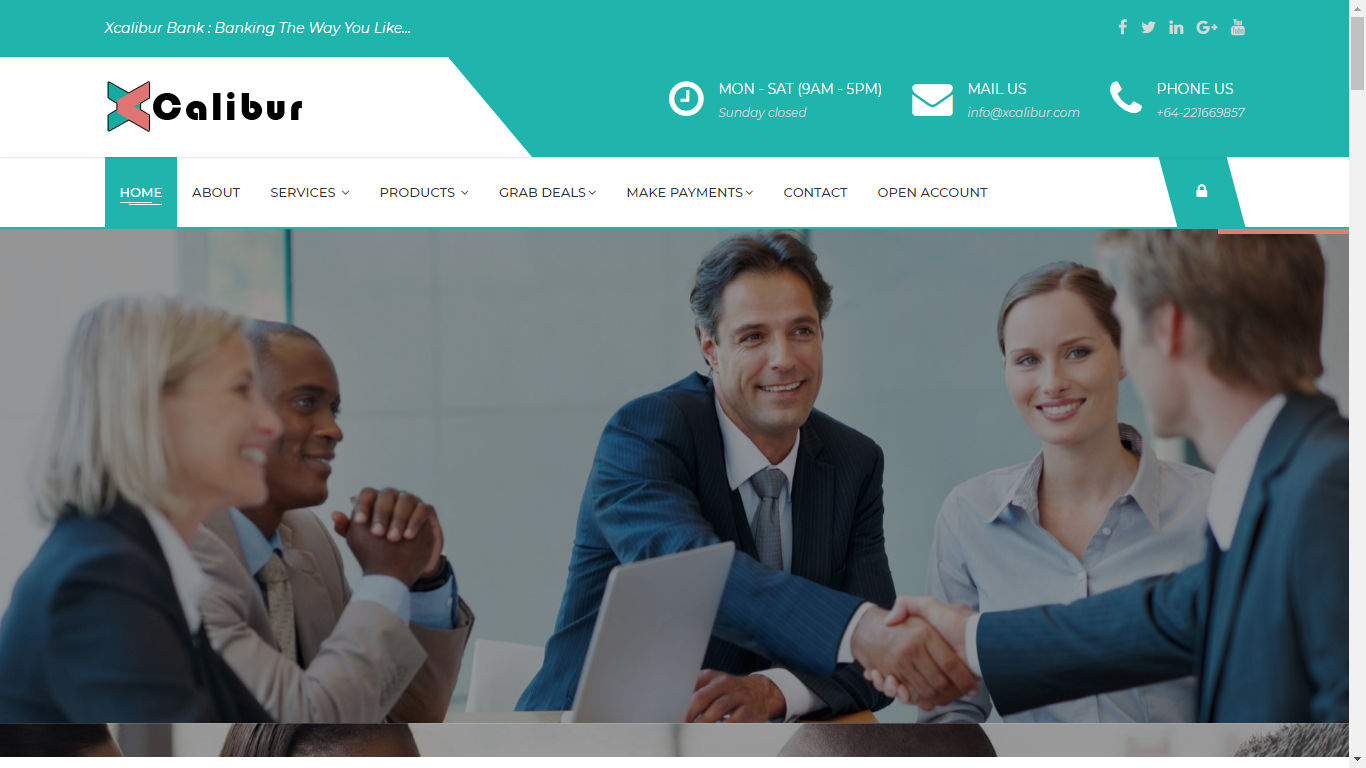
Front- End : MVC.NET AND ANGULAR 7 and NODEJS

Back- End : MS SQL SERVER 2012 EXPRESS & SQL SERVER MANAGEMENT STUDIO

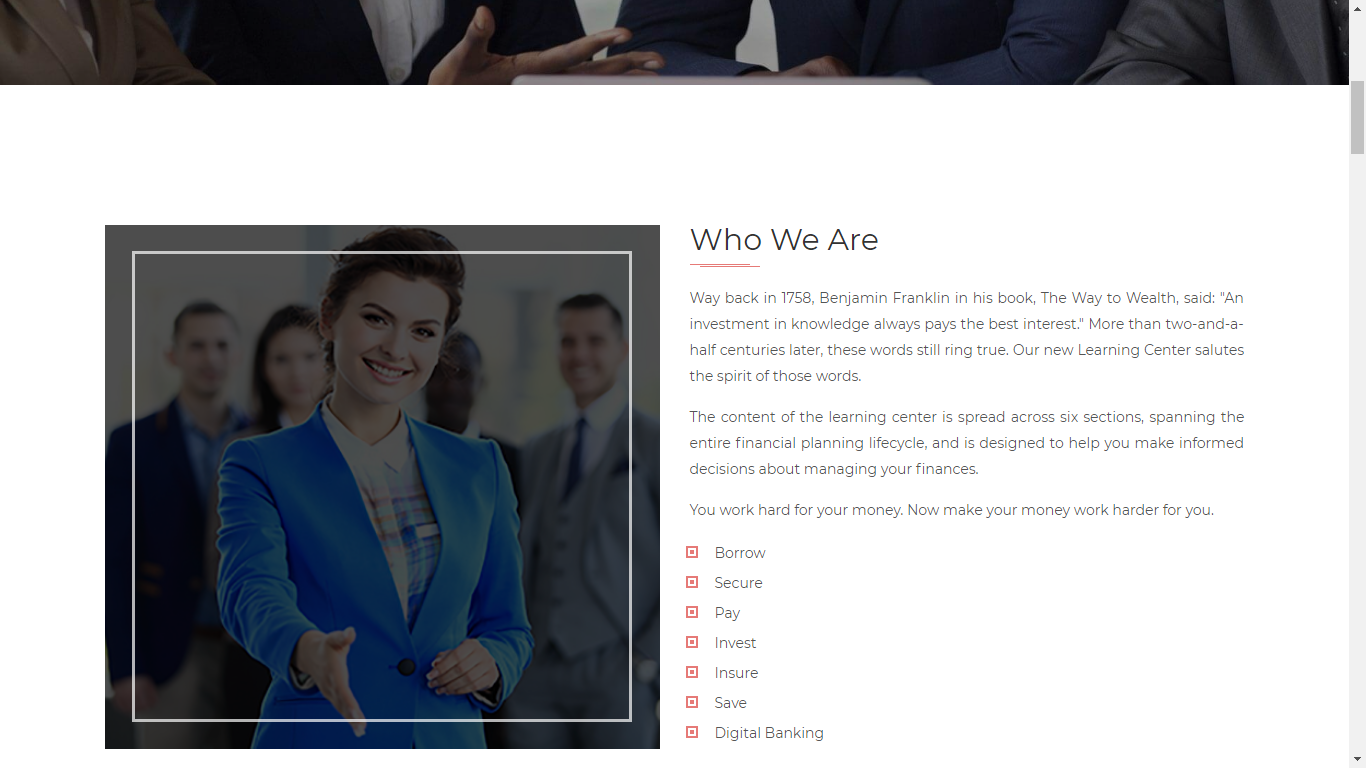
Editor : Visual Studio 2017

**7) PRESENTATION OF PROJECT:-**

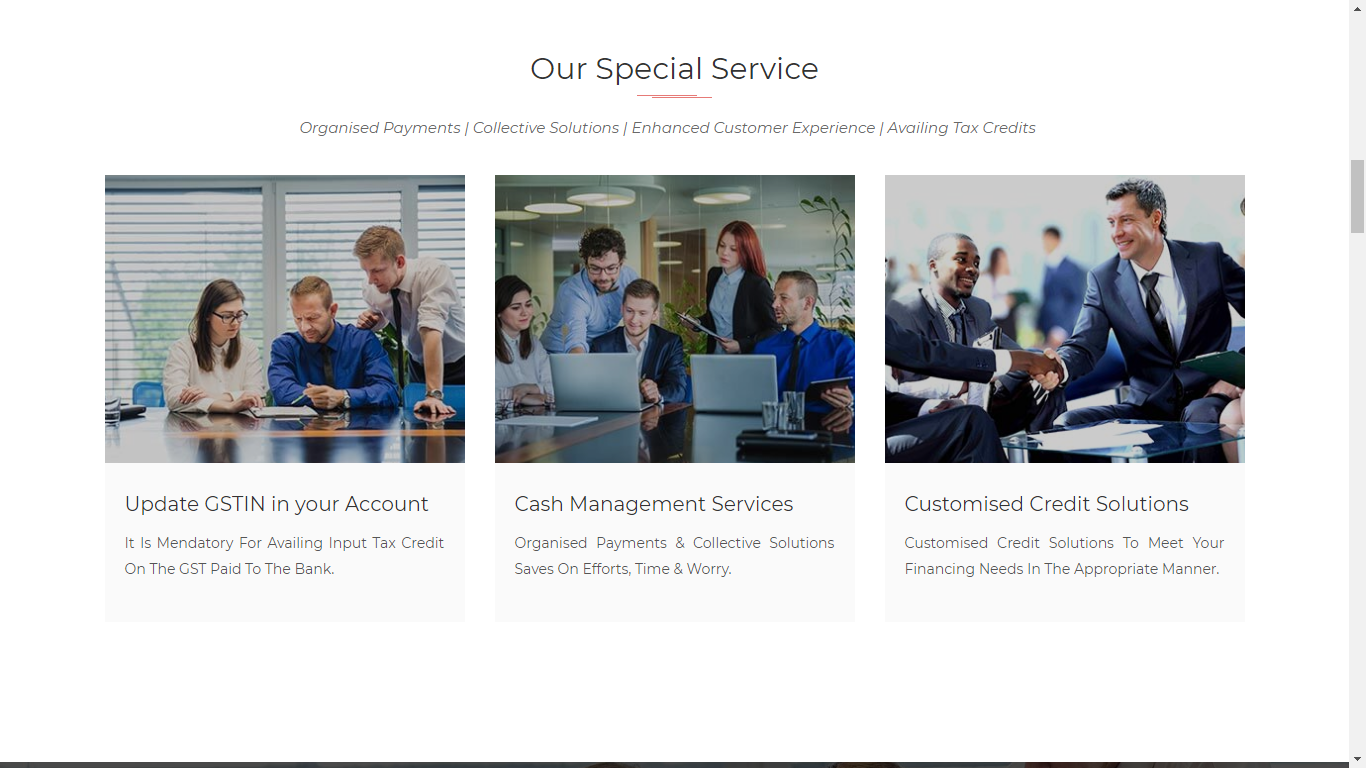
**HOME PAGE**

****

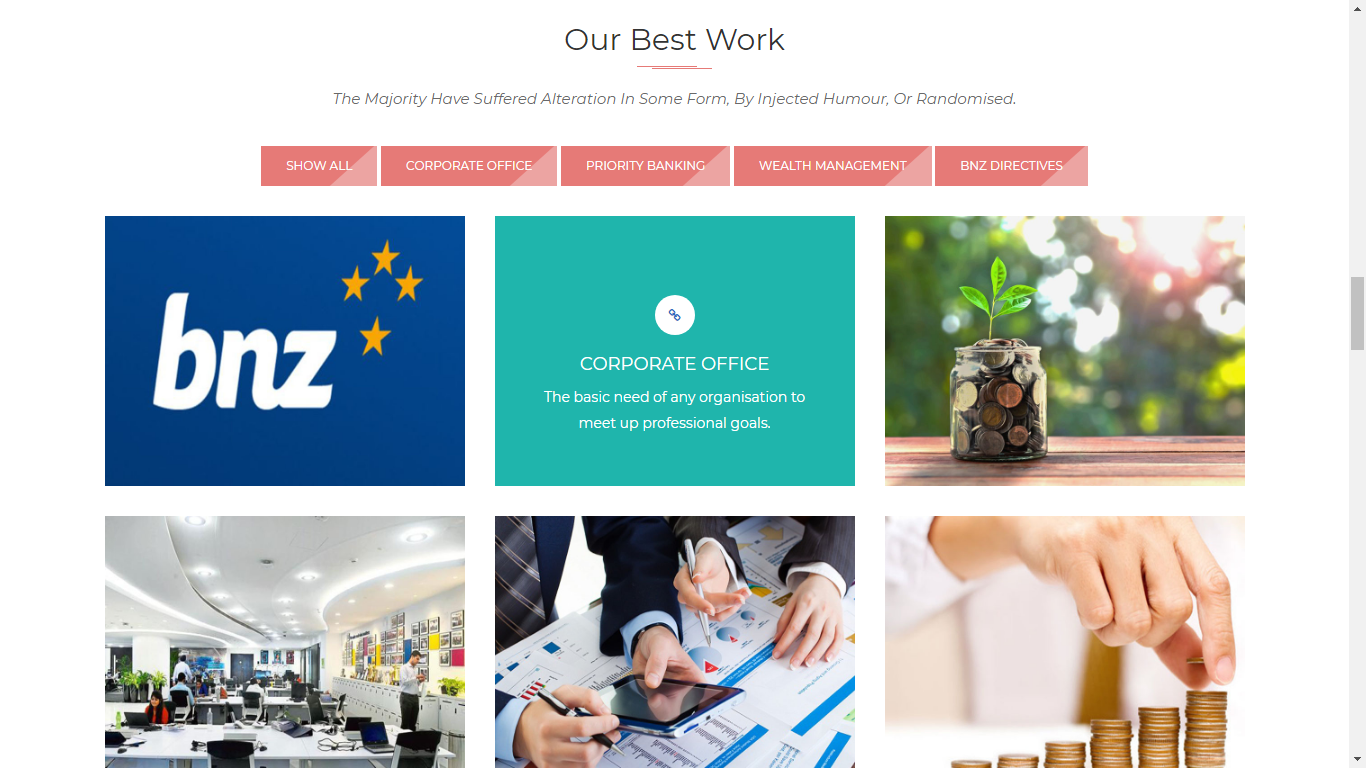
**HOMEPAGE#WHO WE ARE**

****

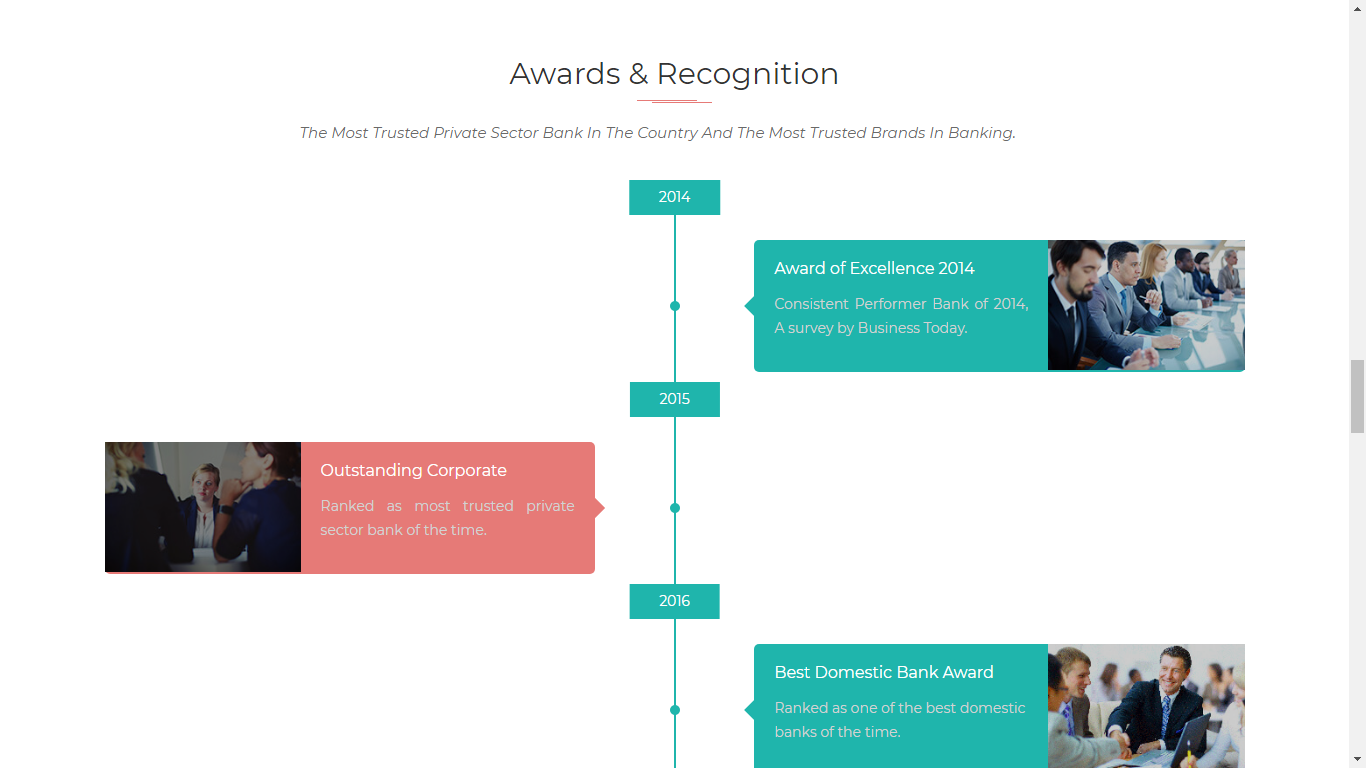
**HOMEPAGE#SERVICES**

****

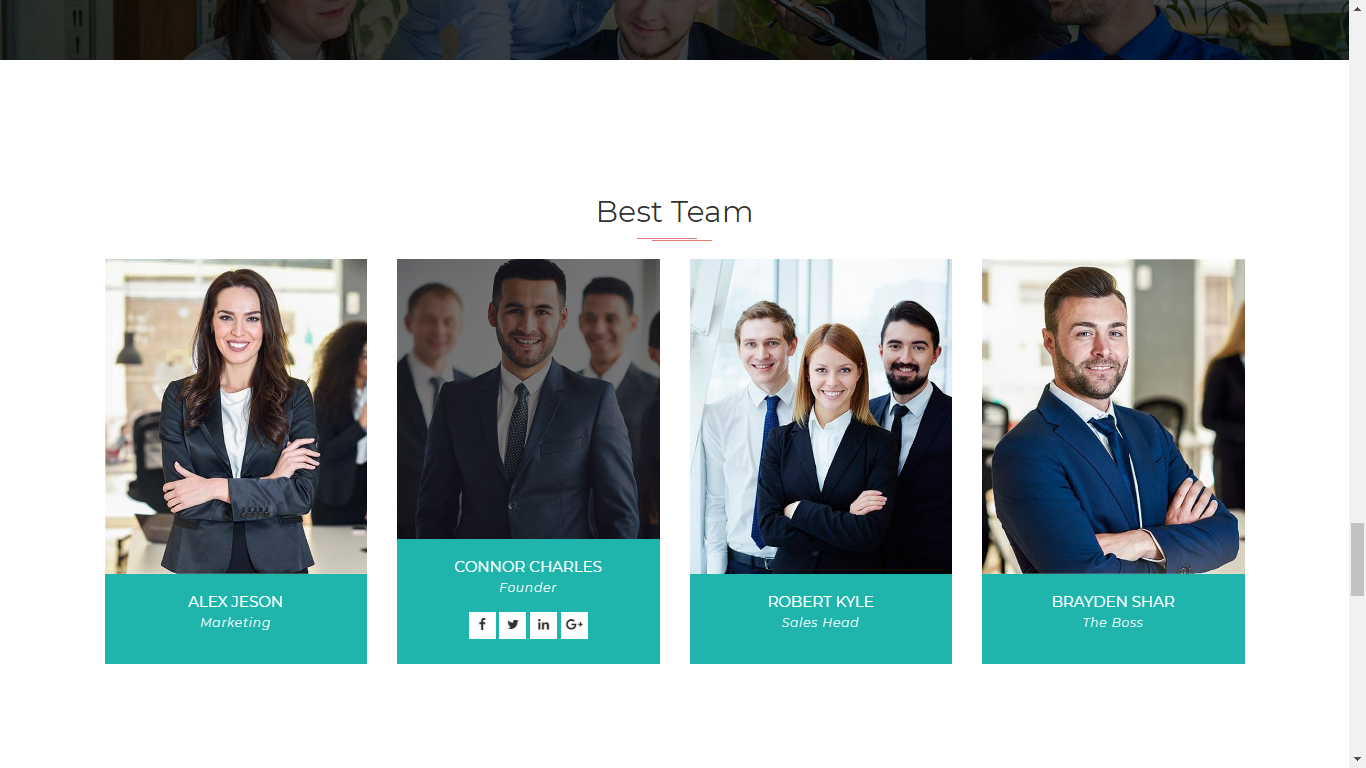
**HOMEPAGE#BESTWORK**

****

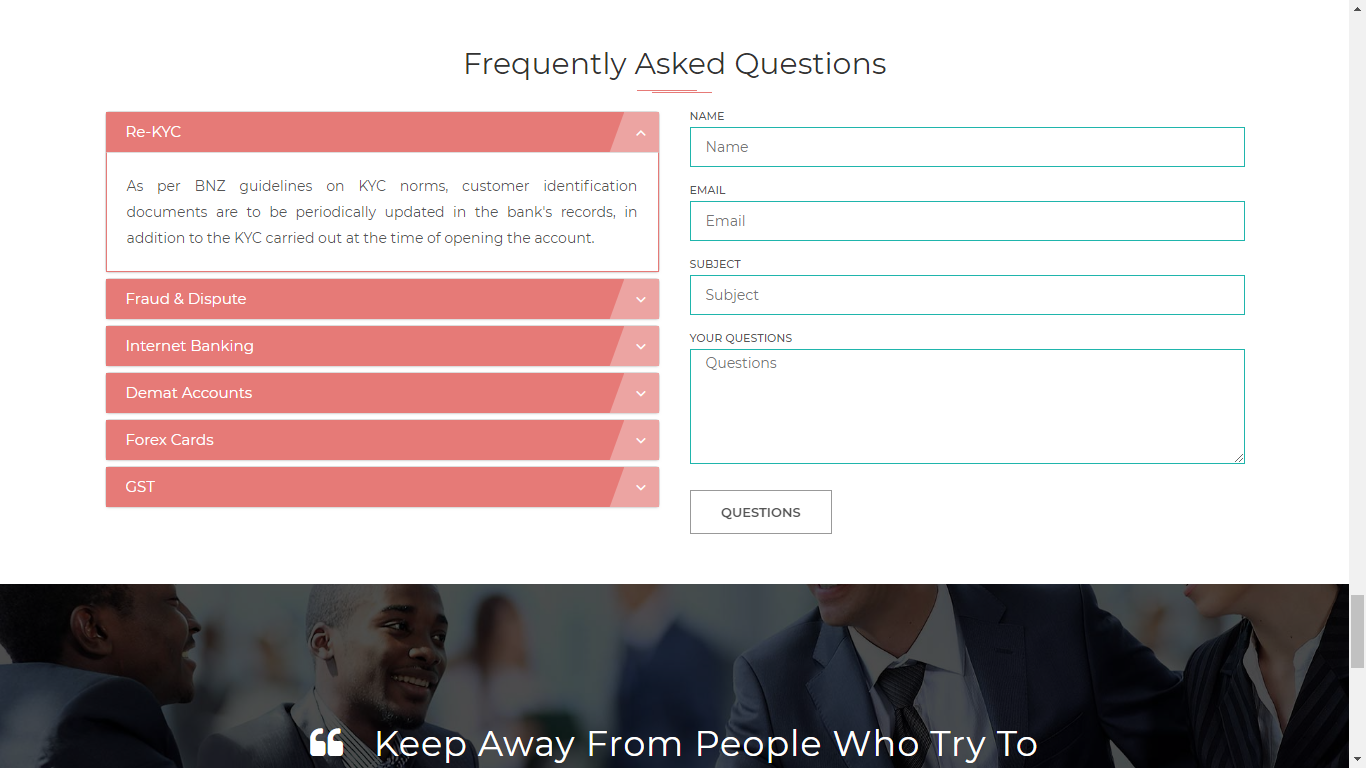
**HOMEPAGE#AWARDS**

****

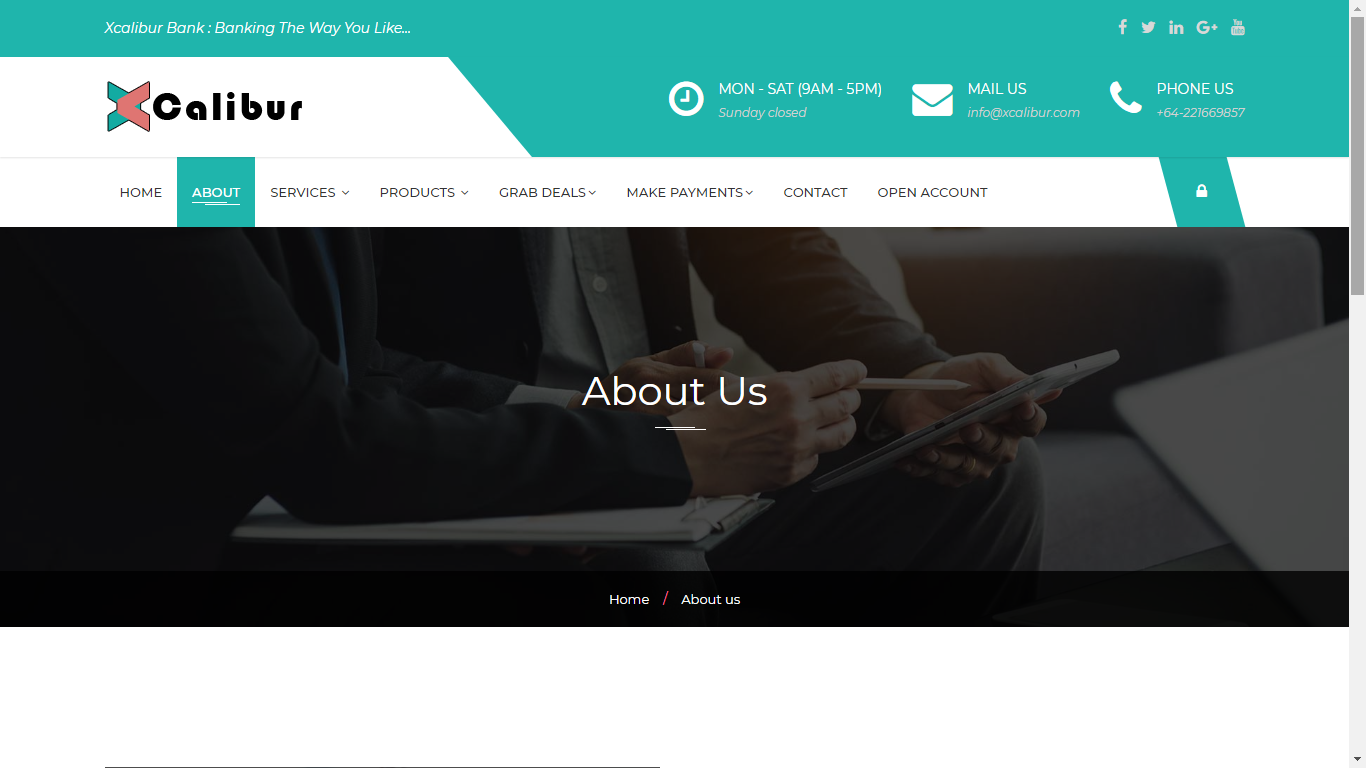
**HOMEPAGE#TEAM**

****

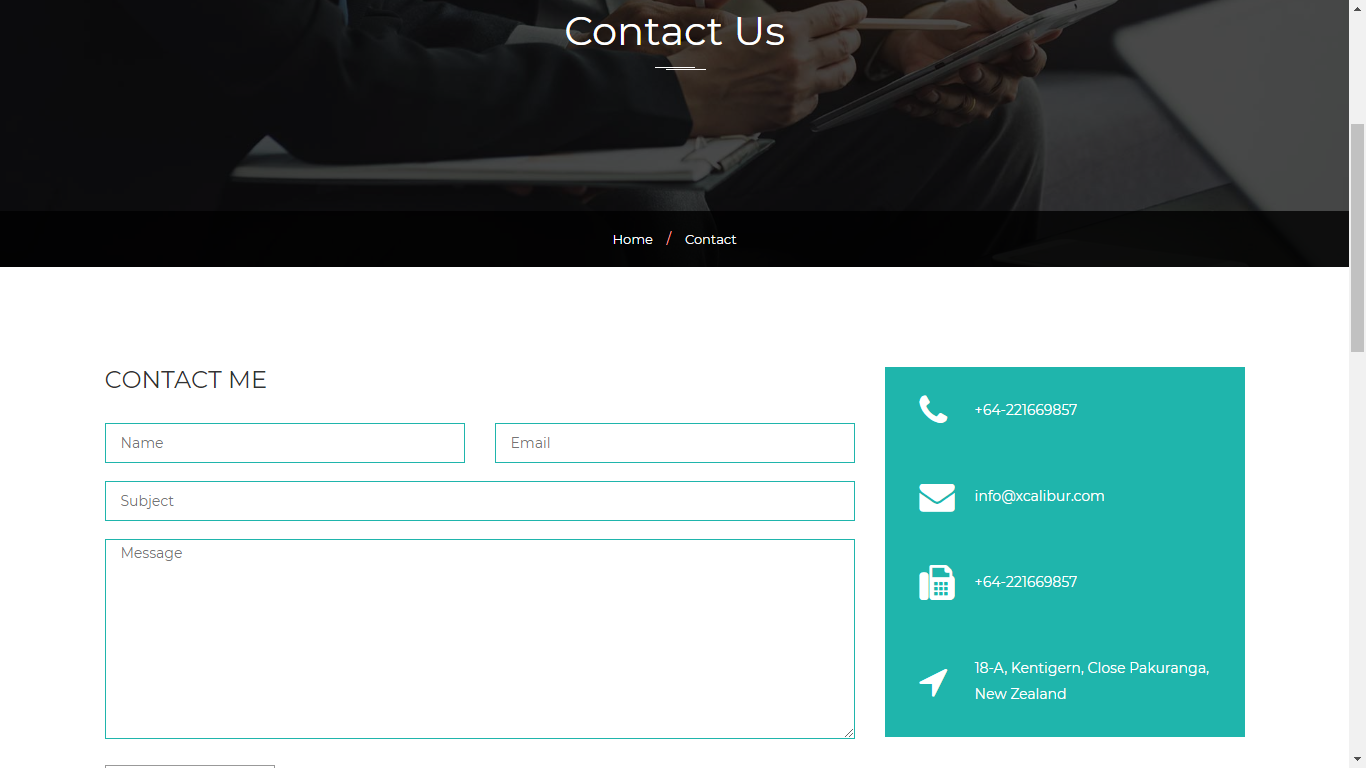
**HOMEPAGE#FAQ**

****

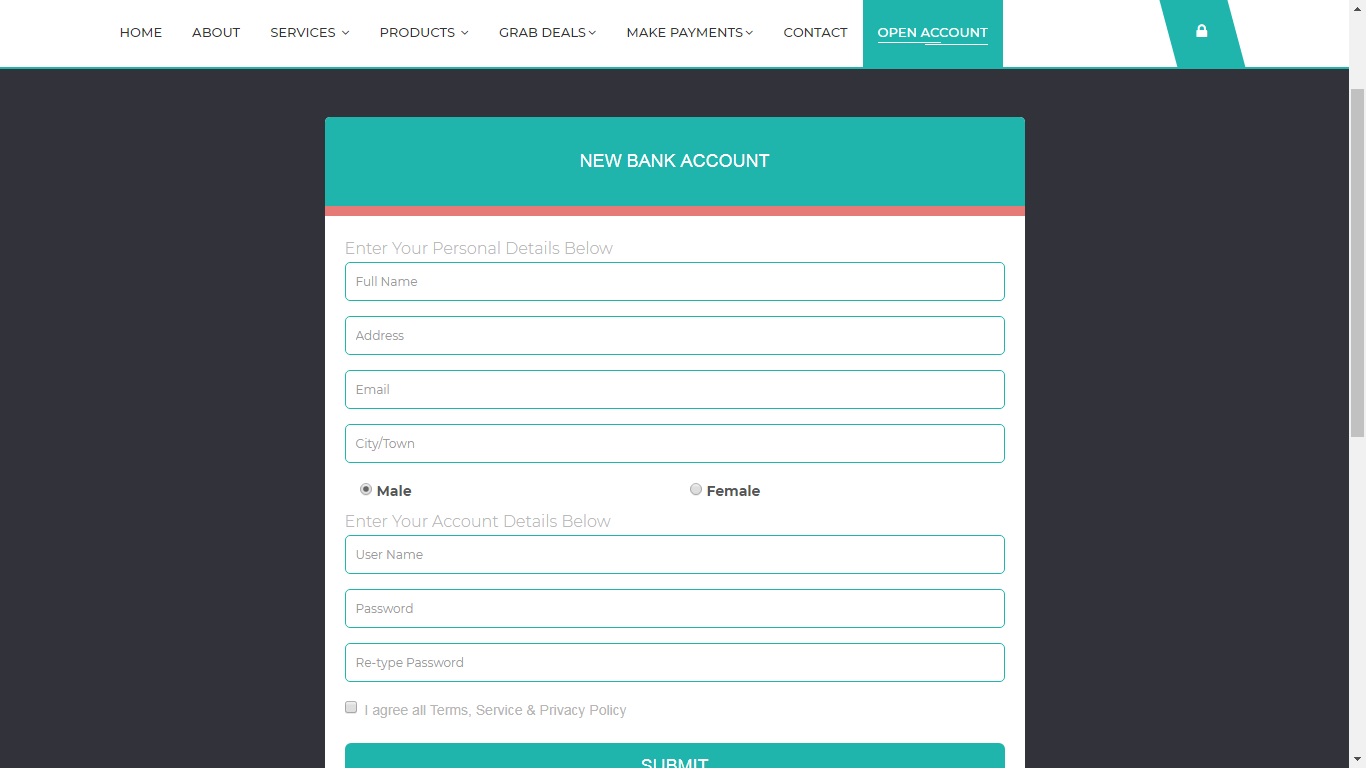
**ABOUT US**

****

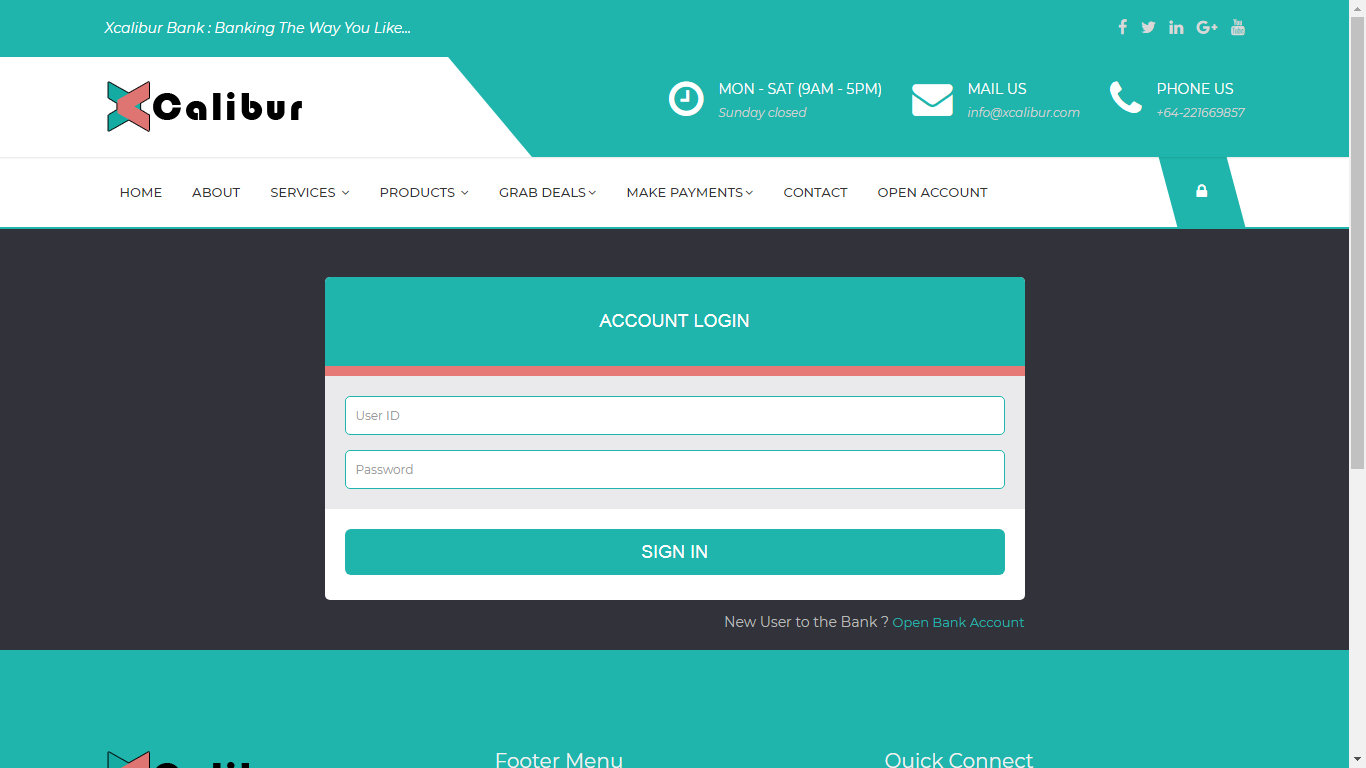
**CONTACT US**

****

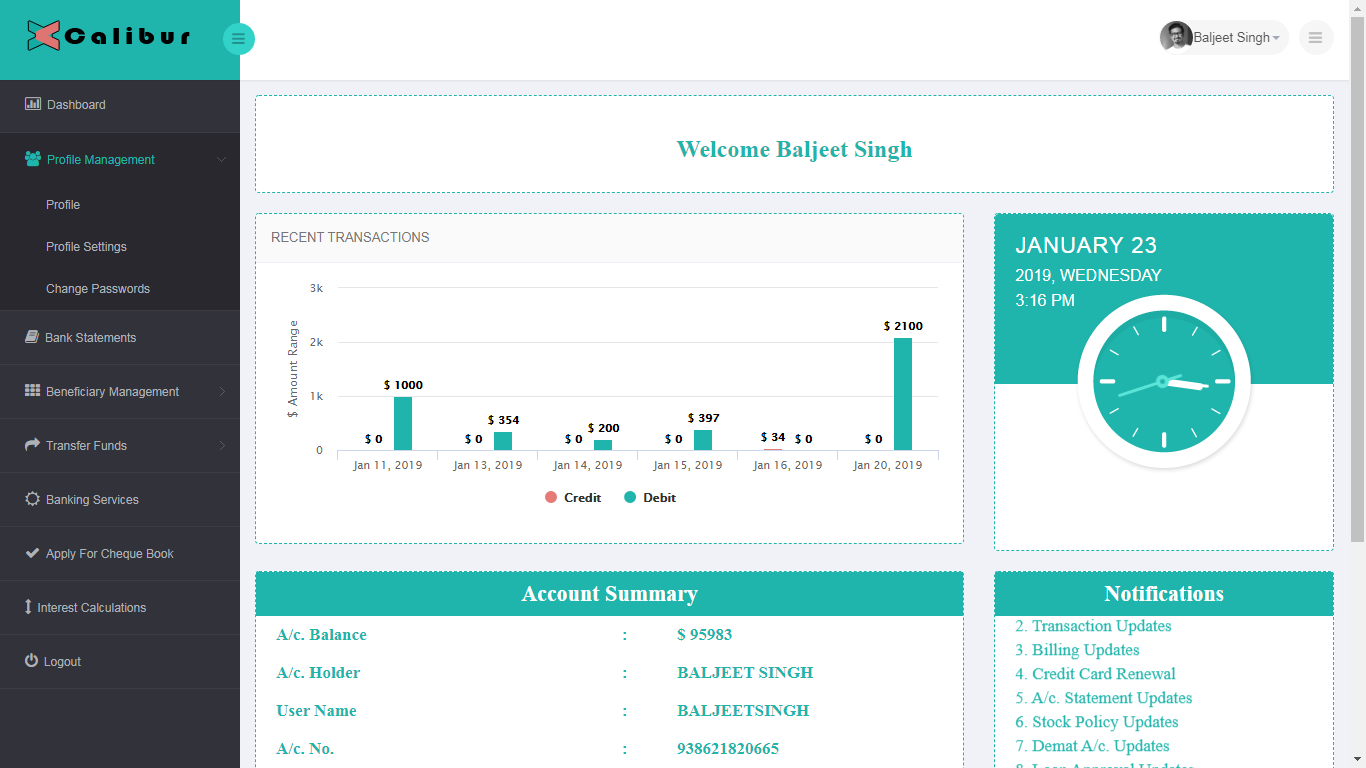
**ACCOUNT OPEN**

****

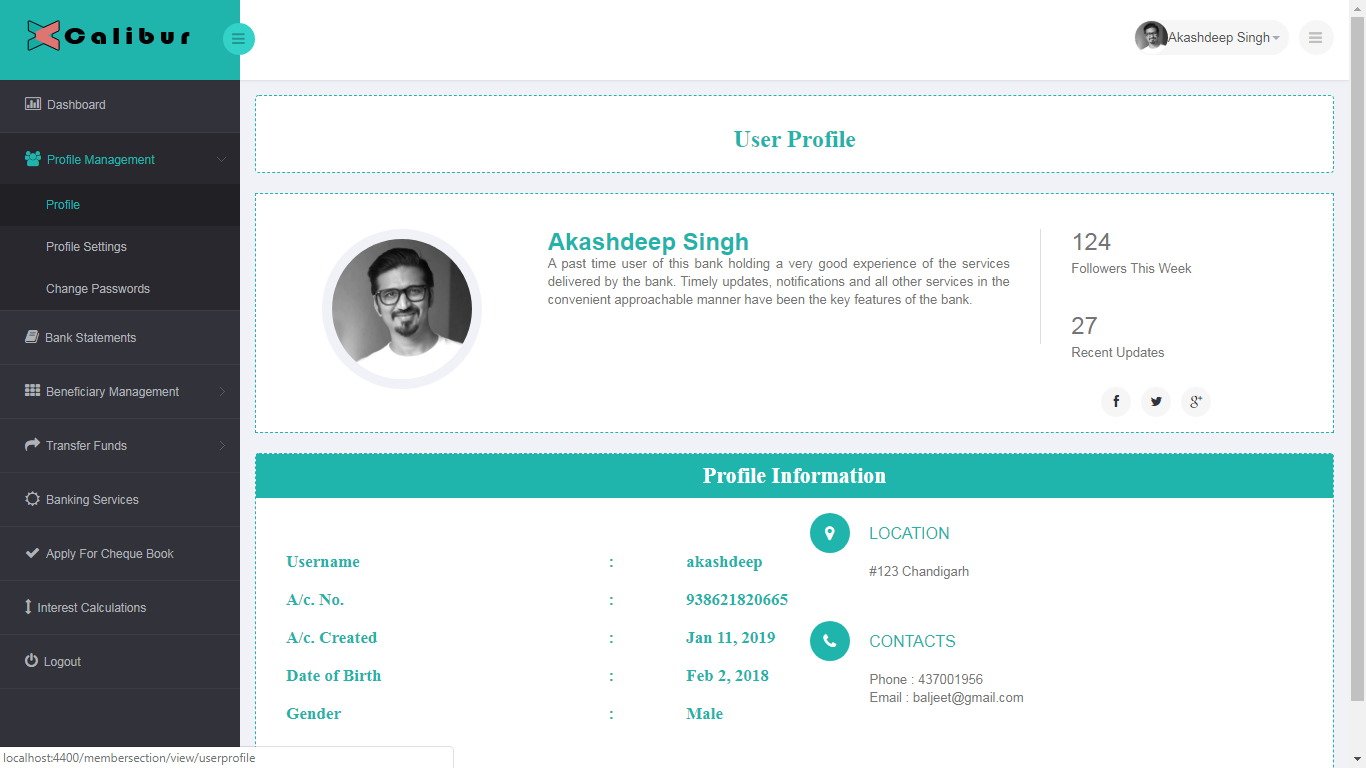
**LOGIN PANEL**

****

**DASHBOARD**

****

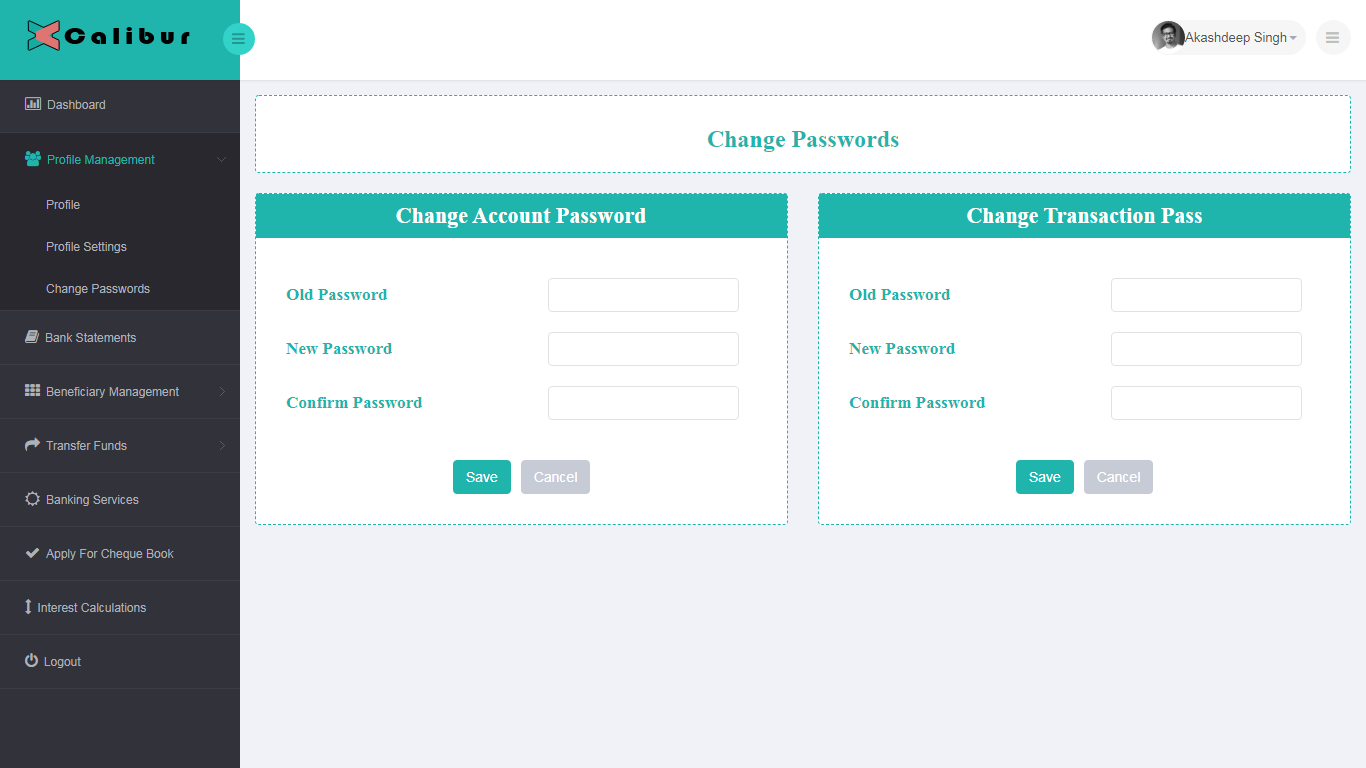
**USERPROFILE**

****

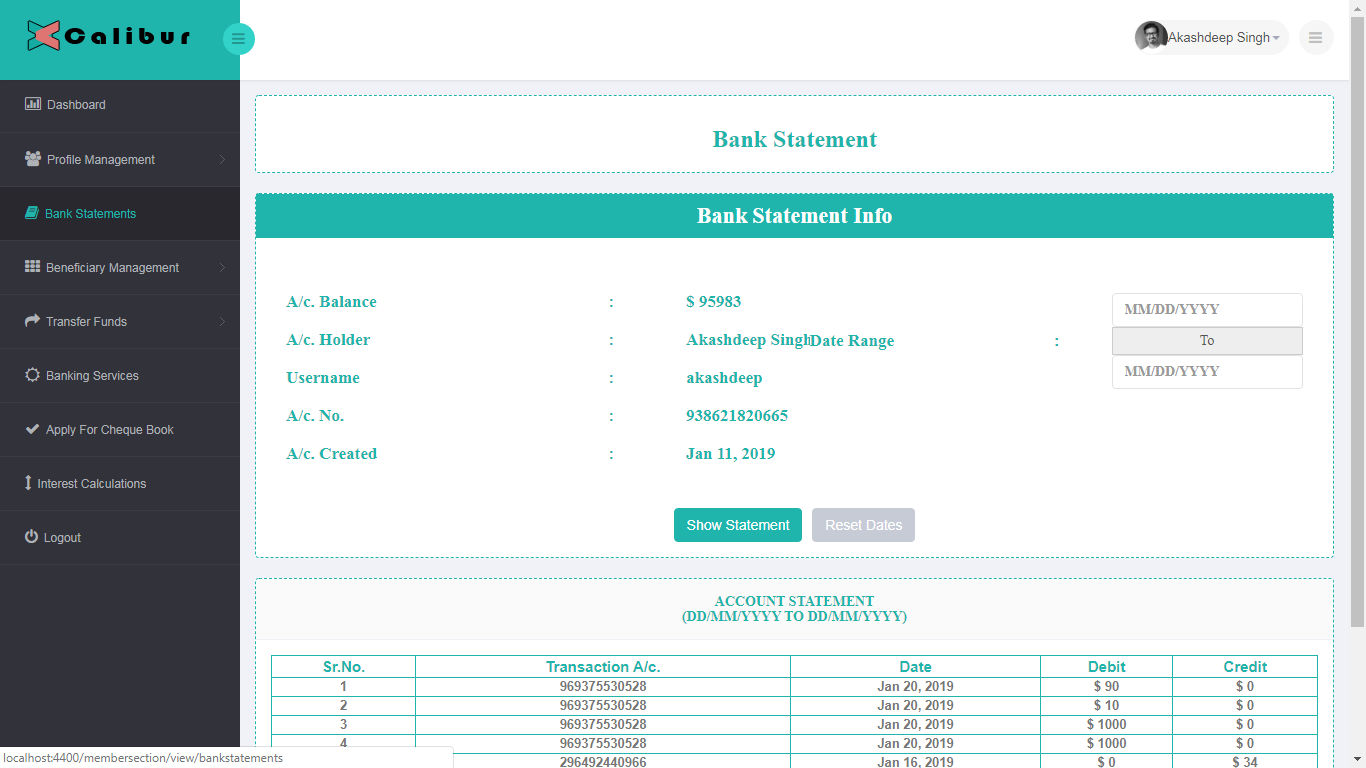
**EDITPROFILE**

****

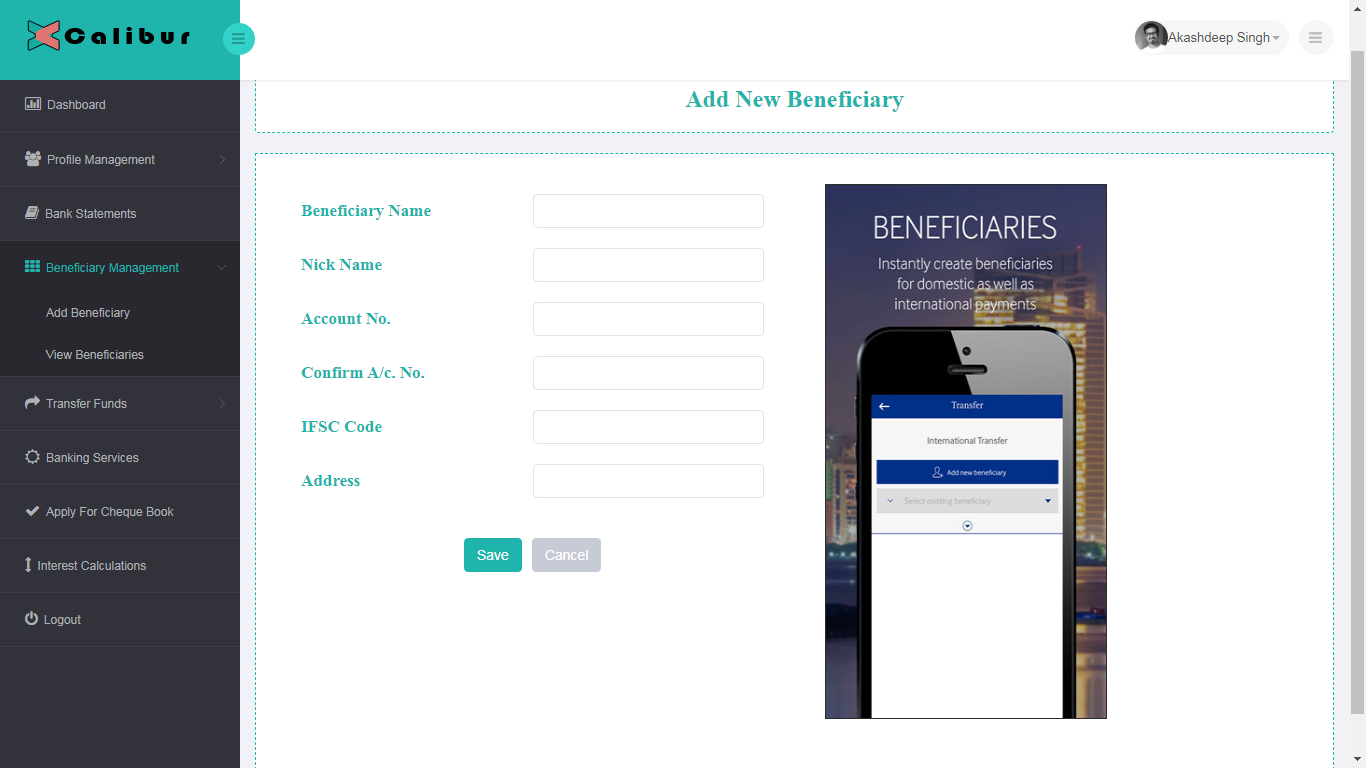
**CHANGE PASSWORD**

****

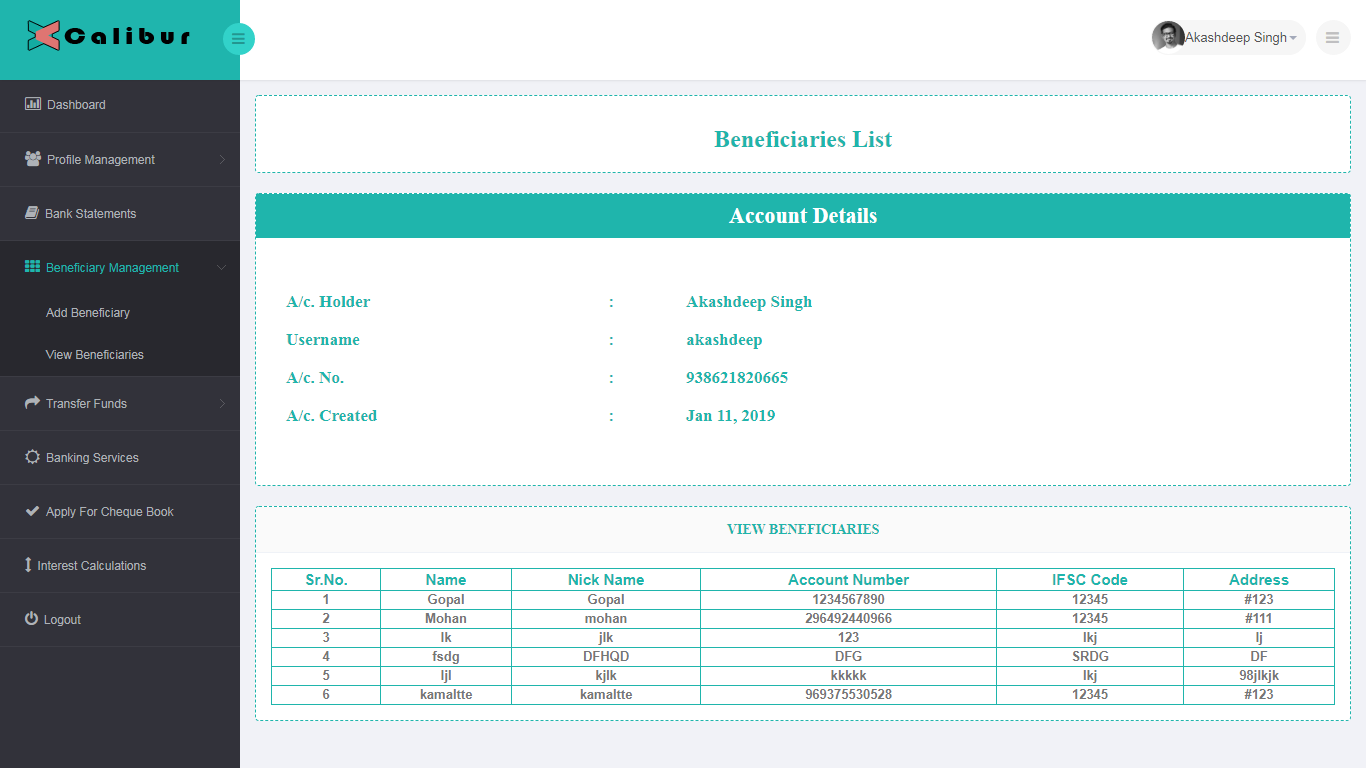
**BANK STATEMENTS**

****

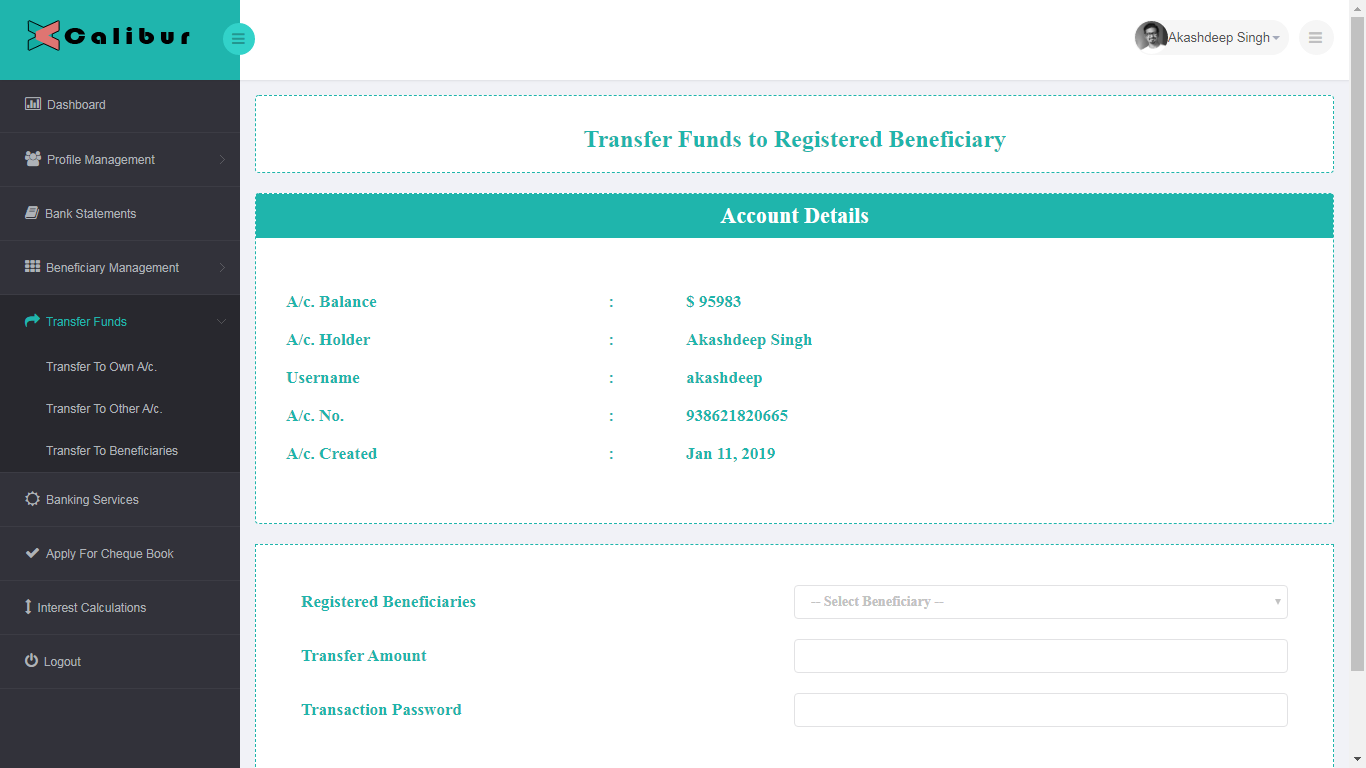
**ADD BENEFICIARY**

****

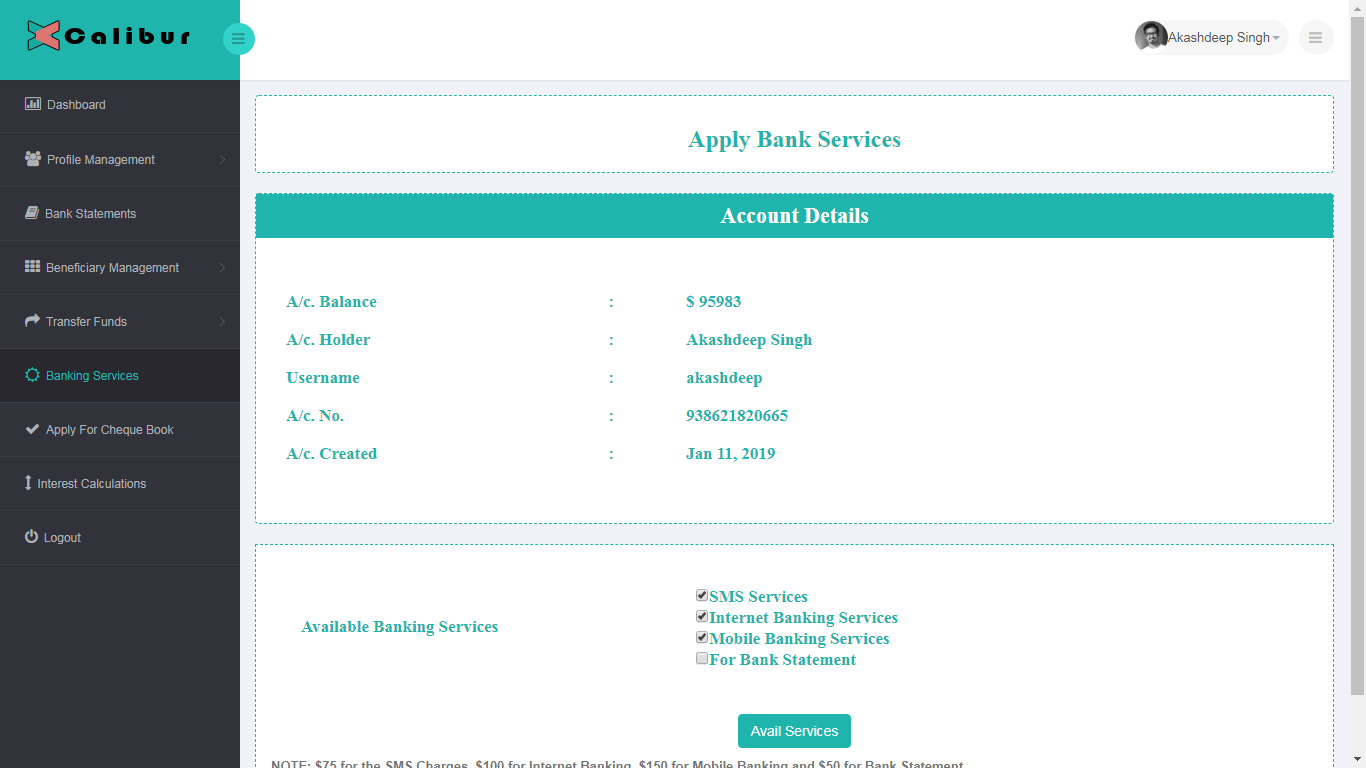
**VIEW BENEFICIARY**

****

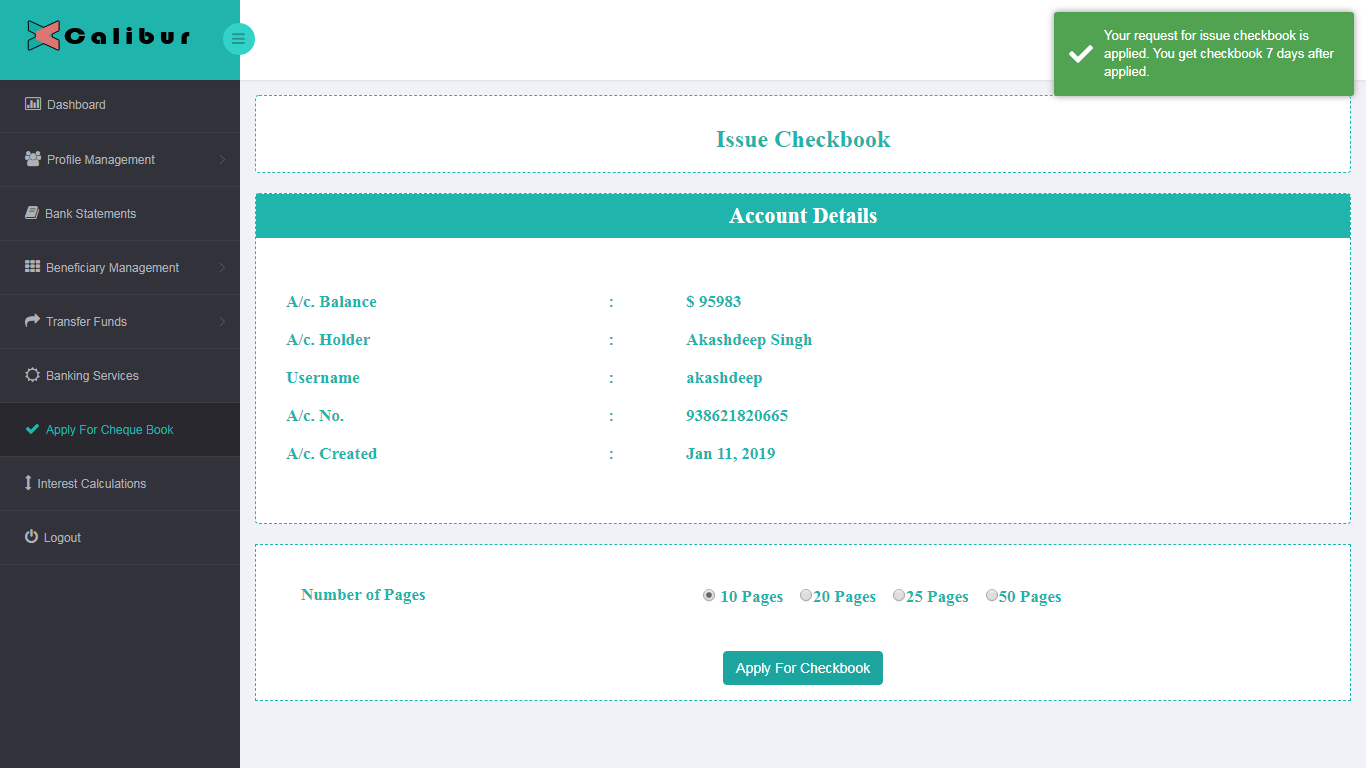
**FUNDS TRANSFER TO BENEIFICIARY**

****

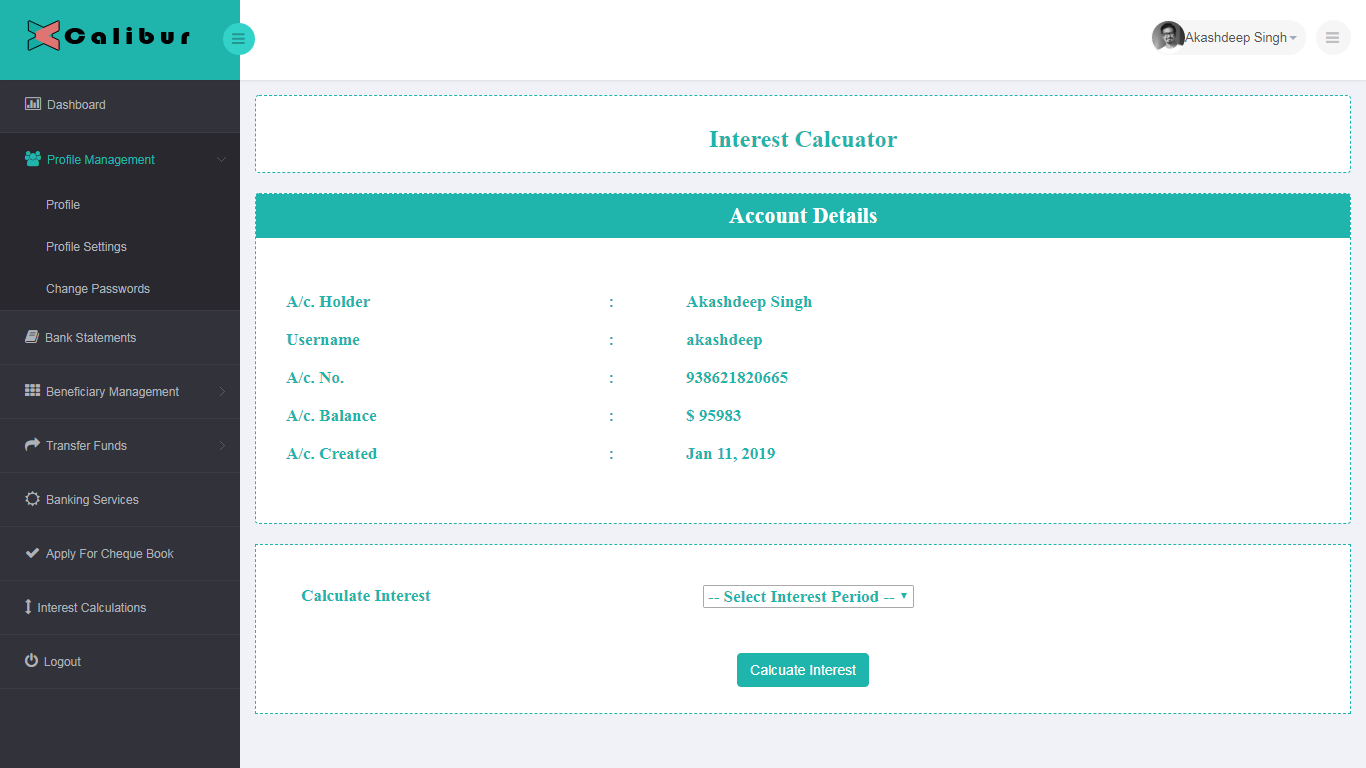
**BANKING SERVICES**

****

**APPLY CHECKBOOK**

****

**INTEREST CALCUATOR**

****

**RESOURCES:-**

**Hardware Description**: The selection of hardware is very important in the existence and proper working of any software. When selecting hardware, the size and requirements are also important.

Minimum Requirements:

Processor : INTEL Core i3

RAM : 3MB

Hard Disk Drive : 250GB

The proposed System is developed on:

Processor : INTEL Core i5

RAM : 4GB

Hard Disk Drive : 500GB

**Software Description:**

Operating System : Windows 7 Service Pack 1

Front- End : MVC.NET AND ANGULAR 7 and NODEJS

Back- End : MS SQL SERVER 2012 EXPRESS & SQL SERVER MANAGEMENT STUDIO

Editor : Visual Studio 2017

**PROJECT MILESTONES**

|  |  |
| --- | --- |
| **DAY 1** | Collect the data options used for the prepare the design of the web app. Prepare the design option for the end user point of view and for the registered user. |
| **DAY 2** | Design the web template for the web app. Create the menu bars with the number of options that we collect to display the web page with the specific menu details. |
| **DAY 3** | Create the database structure with the number of tables that will be use for store the registered user details like his/her profile, store the details of his/her fund transfers, added beneficiaries etc. |
| **DAY 4** | Setup the Angular with .NET MVC Web API. Design the frontend web app structure in angular. Install the required software that will be needed for run the web app. |
| **DAY 5** | Start work on the web api for the registration, login, display registered user profile. |
| **DAY 6** | Integrate the web api with the frontend angular and bind with the form controls. Implement the validation on the registration and login form fields. |
| **DAY 7** | Create the web design for the registered user panel and set the number of menu options that will be use. |
| **DAY 8** | Start work on the add beneficiaries and view the added beneficiaries. |
| **DAY 9** | Work on the funds transfer. Design the fund transfer to same bank, different bank and to the added beneficiaries. |
| **DAY 10** | Check the validation on the add beneficiaries and fund transfer screen. |
| **DAY 11** | Show the last 10 bank statement and design the start and end date for the filter the bank statement dates. |
| **DAY 12** | Design the bar graph for the last 6 days with sum of number of credit and debit on the same day. |
| **DAY 13** | Start work on the apply online check book and bank services like sms service, email bank statement service etc. |
| **DAY 14** | Implement the calculation of interest rate on the 3, 6 9 months and 1 year. |
| **DAY 15** | Fixes the design issues on the end user screens. |
| **DAY 16** | Testing and fixing the bug related to the logic part for the end user . |
| **DAY 17** | Apply the contact us page functionality. |
| **DAY 18** | Design the lock screen and set the time out if specified time the registered user is idle. |
| **DAY 19** | Set the design fixes. |
| **DAY 20** | Create a documentation file with all the screen shots. |

**PROJECT SCHEDULE**

List all of the activities you will need to do to complete the deliverables, along with expected time needed to perform them and dates they are due by.

Activities includes are:

1. Completion of initial investigation
2. Design Drafted
3. Design signed-off
4. Prototype program completed
5. Stages of program refinement signed-off
6. Program completed
7. Program tested
8. System documentation completed
9. User acceptance completed
10. User report on system acceptability completed

**PROJECT ASSUMPTIONS:-**

1) As we all know that while performing the banking application login, for the safety hands all banks generates OTP on the registered mobile number, because the mobile OTP is a paid service. so we assume the OTP is generated automatically while login the account.

2) For the initial bank balance amount is also the assumption. In our project we used the $100000 as the initial bank balance after creating the bank account.

3) All the funds transfer and adding the bank account is limited to the our database, and if we could enter the real time account number for the funds transfer, we could not credit amount on that bank account. because we need to purchase the banking services for real time funds transfers.

**PART III - REVIEW YOUR PROJECT**

**1) FINAL PRODUCT/ OUTCOME:-**

a) Designed the home page.

b) Designed the about us page.

c) Designed the contact us page. Able to send contact message to the admin email.

d) Registration panel for new customer.

e) Login panel for the existing one.

f) Able to view his/her profile, manage profile, change photo etc.

g) Able to change password and transaction password.

h) Could add the new beneficiaries.

i) View the existing one.

j) Funds transfer to the same bank, different bank and already added beneficiaries.

k) View bank statements of the funds transfers.

l) Avail the bank services like email bank statements, sms services etc.

m) Calculate the interest rate monthly and yearly on the current balance amount.

**2) PRODUCT QUALITY:-**

As per decided to number of functionality will be cover-up for the 20 days on the banking project, i achieved that all functionality within the same number of day.

**3) MEASURES AGAINST OBJECTIVE AND MEASURE OF SUCCESS:-**

Project performance is primarily measured on time, cost and scope. It means the ongoing project tasks are compared against the scheduled tasks. The cost of tasks undertaken is compared with the budgeted cost. The features and functions built in the solution are compared with the scope objectives to check if the application delivers as per defined scope.

It is not easy to orchestrate a project. There are conflicting objectives of various stakeholders that scream for attention. At the same time, project has to be successful in terms of cost, time and scope as well as when measured by other parameters.

At the time of project planning, key stakeholders need to decide what should be evaluated to measure the success of the project. This will help in identifying Key Performance Indicators (KPIs) that can be used to measure project performance. Here are some measurement indices that can be used to track the project.

### Project Performance Measures

**Earned Value Management (EVM) –** EVM is a technique used to track progress of a project in terms of schedule and cost. It aims to forecast the future performance of the project. It is an essential project tracking concept in a majority of software projects. The following measures are tracked and compared as part of EVM –

**Cost Variance (CV) -** CV measures project performance in terms of how much the project is over or under the budget.

If CV is a negative value, it means the project is over budget and if it is a positive number, the project is under budget.

**Cost Performance Index (CPI) –**It shows how efficiently resources are being utilized.

A CPI value above 1 tells that resource utilization is efficient. CPI value below 1 indicates resource utilization is not efficient.

**Schedule Variance (SV) -** SV measures how much ahead or behind schedule the project is in terms of cost.

A positive value indicates that the project is ahead of schedule and a negative indicates that it is behind schedule.

**Schedule Performance Index (SPI) –** It shows how efficiently is time utilized in the project.

**FUTURE SCOPE:-**

This project was developed to fulfill user requirement; however there are lots of scope to improve the performance of the Banking System in the area of user interface, database performance, and query processing time. Etc.

So there are many things for future enhancement of this project. The future enhancements that are possible in the project are as follows.

* Linking and integration of any legacy system for accounting.
* Integration with other bank and government agencies through Web Services
* Connection to third-party OLAP applications
* Electronic Data Interchange (EDI) system for ATM machine
* Web Interface for net banking.
* In the area of data security and system security.